HyQoZ - Optimisation de requêtes hybrides basée sur des contrats SLA

Carlos-Manuel Lopez-Enriquez

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 7 août 2006

Présentée par

Carlos-Manuel LÓPEZ-ENRÍQUEZ

Thèse dirigée par **Christine COLLET**
et codirigée par **José-Luis ZECHINELLI-MARTINI**

préparée au sein **Laboratoire d'Informatique de Grenoble**
et de **École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

HyQoZ – Optimisation de requêtes hybrides basée sur des contrats SLA

Thèse soutenue publiquement le **23/Octobre/2014**,
devant le jury composé de :

Pr., Parisa GHODOUS
Université de Lyon 1, Président
Dr., Cedric DU MOUZA
Conservatoire National des Arts et Métiers, Rapporteur
Pr., Thierry DELOT
Université de Valenciennes, Rapporteur
Pr., Christine COLLET
Grenoble INP, Directeur de thèse
Dr., Genoveva VARGAS-SOLAR
CNRS, Co-Encadrant de thèse
Pr., José-Luis ZECHINELLI-MARTINI
UDLAP-LAFMIA, Co-Directeur de thèse

Carlos-Manuel LÓPEZ-ENRÍQUEZ
HYQOZ – SLA-AWARE HYBRID QUERY OPTIMIZATION
164 pages.

Impression : thesis.tex – 22 janvier 2015 – 09:15

Résumé

Dans ce travail, nous attaquons le problème de l'optimisation des requêtes hybrides sur les données produites par les services de données soit à la demande, soit en continu. Ces requêtes sont évaluées par des query workflows qui coordonnent les services de données et de calcul. L'exécution d'un query workflow doit respecter un contrat de niveau de service (Service Level Agreement) qui définit l'objectif d'optimisation en termes de la qualité de service attendue. L'objectif d'optimisation est donc representé par un vecteur d'attributs de coût pondérés tels que le prix, le temps, l'énergie, etc. Les poids définissent les préférences parmi les attributs de coût pour permettre la comparaison entre query workflows.

L'optimisation des requêtes hybrides dans ce contexte consiste à déterminer le meilleur query workflow satisfaisant le contrat SLA. Notre approche pour générer l'espace de recherche de query workflows, l'estimation de coût, et l'espace de solutions est donc orientée pour satisfaire des contrats SLA. Nos principaux résultats sont : (1) la génération de l'espace de recherche compte-tenu à la fois du flot de contrôle et du flux de données des query workflows, (2) une fonction de coût qui tient compte de l'absence de paramètres asociés aux données, et (3) l'adaptation d'un algorithme top-k pour sélectionner les query workflows appropriés pour un SLA donné.

Abstract

In this work we tackle the problem of optimizing hybrid queries over data produced by data services either on-demand or continuously. Such queries are implemented by query workflows that coordinate data and computing services. The execution of query workflows has to respect Service Level Agreement contracts that define the optimization objective in terms of the expected quality of service. The optimization objective is therefore described by a vector of weighted cost attributes such as the price, the time, the energy, etc. The weights define the preferences among the cost attributes for enabling the comparison among query workflows.

In this context, the hybrid query optimization is about to find the query workflow that best conforms the SLA contract. Our approach for generating the search space of query workflows, the cost estimation, and the solution space is therefore oriented to satisfy the SLA contracts. Our main results are: (1) the generation of the search space considering both control-flow and data-flow of query workflows, (2) a cost function that considers the absence of data-related parameters, and (3) the adaptation of a top-k algorithm for selecting the suitable query workflows for a given SLA.

Acknowledgments

Je tiens à remercier sincèrement ma directrice Christine Collet, mon encadrante Genoveva Vargas-Solar, mon co-directeur José-Luis Zechinelli-Martini pour leur critiques, support et patience dans la réalisation de ma thèse. Également, je remercie les rapporteurs et les examinateurs de leur disposition d'accepter de faire partie du jury de mon travail.

Grenoble, 23/Octobre/2014

Carlos Manuel López-Enríquez

Table of contents

List of tables

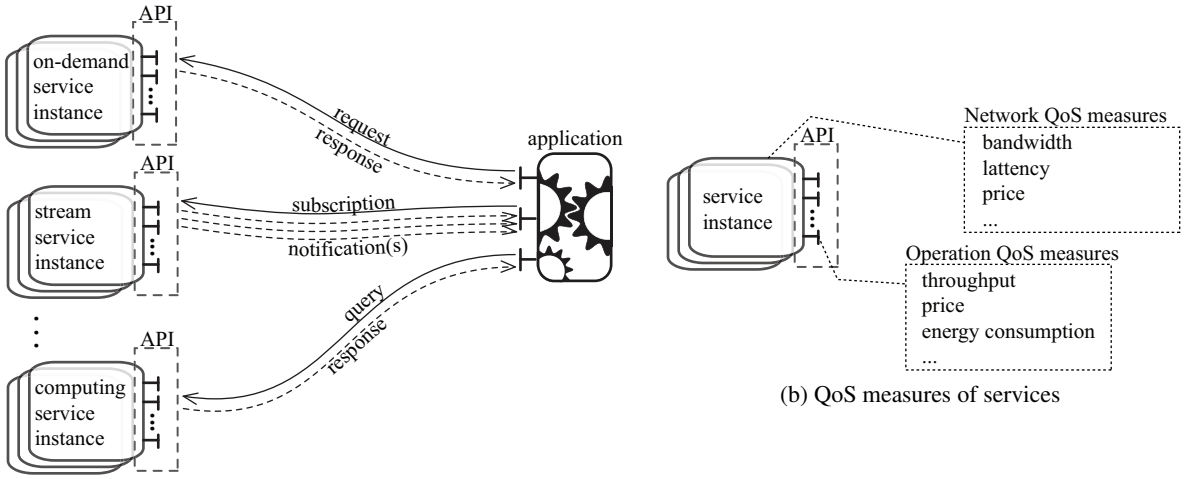
List of figures

CHAPTER 1

Introduction

1.1 Data management in service-based environments

The new challenges of data management reported by the community [AAB+09, Gou09, ACK+11] invite to revisit the way data are captured [BN08, Wei09], stored [KCC+11], protected [KP13, BEE+13, ZPL08], processed [Jin12], and queried [VSIAP10]. In particular, the democratization of service-based environments (*e.g.* Internet) by means of the service notion allows actors to be producers and consumers of data through applications running in different devices. As depicted in Figure 1.1a, applications coordinate either on-demand or stream data services, and computing services [1] accessible via Application Program Interfaces (API). Applications access the service operations described by API's for retrieving and transforming data according to the needs of users.



(a) Services coordinated by applications

(b) QoS measures of services

Figure 1.1 – Services in service-based environments

Service instances and network have a series of Quality of Service (QoS) measures denoting the levels of service they offer (see Figure 1.1b). The combination of the QoS measures determines the resulting QoS of the application using such resources. For instance, the bandwidth and latency of the network

1. An *on-demand data service* provides data related to unchanging information (or subject to a single point in time) delivered under a request-response pattern. A *stream data service* provides data related to information that changes during the time and delivered under a publish-subscriber pattern. A *computing service* provides computing operations in a query-response pattern.

along with the data size impact the communication time. The economic aspect is also important given the business model adopted by service providers of: data [BHS11, KUB+13, AKHLS13], computing and storage [KHAK09, DSL+08]. Further parameters are those related to the quality of data like provenance [KIT10] and accuracy [NLF99]; and with others like security, energy consumption, service availability, and throughput [Men02, BND+04].

Applications coordinating services may have different interests about the resulting quality of the coordination. One can be interested in the reduction of the execution time, another in the reduction of the resulting price, another in the maximization of the privacy of data, or may an application be interested in a combination of preferences among the potential QoS aspects.

In order to have service coordinations satisfying such QoS requirements, developers code programs for retrieving and transforming data produced by services implementing what an off-the-shelf database system does in traditional databases. Developers chose manually services and define the invocation order according to the QoS aspects they are interested in, or establish separate Service Level Agreements (SLAs) with service providers.

As with service providers, the QoS requirements of a service coordination can be defined by an SLA contract. Instead of coding a program implementing a service coordination satisfying an SLA contract, it is interesting to have a declarative query approach to specify a data-centric service coordination and generate the program that satisfies the SLA. In this context, the automatic generation of data-centric service coordinations implementing declarative queries imposes interesting optimization challenges due to the multiple QoS measures associated to the environment's resources, the autonomy of services, and the different combinations of interests.

1.1.1 Hybrid queries and SLA contracts

We adopt the notion of hybrid queries [CvVSCB09] for characterizing queries over on-demand and stream data services whose evaluation is done by a data-centric service coordination. We combine hybrid queries and SLA contracts towards the generation of data-centric service coordinations satisfying SLA contracts.

Consider the scenario where Alice wishes to know *Where is Bob?* to hang out with him in a bar. Bob shares his profile through a social network and his current location via the GPS of his smart-phone that continuously shares his coordinates via another social network. Alice, instead to access these social networks to see if Bob is at least 21 years old and he is nearby, uses the application FriendFinder that performs this hybrid query automatically. FriendFinder uses the service operations described by the following interfaces.

```
                friends:profile(nickname) →    profile⟨nickname, email, age, gender⟩
              friends:friendsof(nickname) → friendship⟨nickname, friend⟩
               wruservice:location(email) →   location⟨timestamp, email, lat, lon⟩
geocomp:distance(lat1, lon1, lat2, lon2) →   distance⟨distance⟩
```

The on-demand data services `friends:profile` and `friends:friendsof` provide the profile and the friendships of a given nickname respectively. The stream data service `wruservice:location` provides

a stream with the location of a given user email. The computing service `geocomp:distance` determines the geographical distance between two coordinates.

FriendFinder performs the evaluation of the *Where is Bob?* query by retrieving the Bob's profile from the `friends:profile` service and his location from `wruservice:location`. FriendFinder delegates query tasks to computing services like `geocomp:distance` for estimating the distance between Alice and Bob. In case of limited hardware capabilities [CM09], FriendFinder can use other computing services for transforming data.

Regarding on the QoS requirements defined by an SLA contract, suppose Alice has a budget of 1€ to access data and computing services and to use the 4G network. She also requires a response in less than 1 minute, and she does not want to spend too much energy so she limits the energy consumption to 50kJ. These cost attributes are examples of the possible cost attributes in service-based environments. Users might have different preferences. For instance, Bob privileges data privacy and he does not care about the resulting price as long as he gets the response as soon as possible. The combination of services have to fit the SLA contract coming either from Alice or Bob.

1.1.2 Query evaluation by data-centric service coordinations

For representing data-centric service coordinations implementing the evaluation of hybrid queries, we adopt the notion of query workflow proposed in [VVsC10]. A query workflow defines a control-flow among a series of activities invoking service operations for implementing data transformations, *e.g.* retrieval, filtering, correlation, projection.

Suppose Alice and Bob want to know *Which of their common friends are at least 21 years ?* to go with them into the bar. The execution of a query workflow implementing this hybrid query is depicted in Figure 1.2.

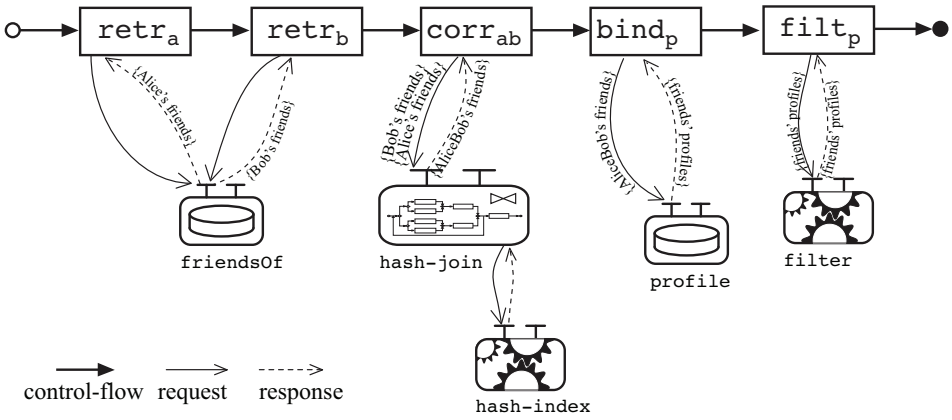


Figure 1.2 – Query workflow execution example

The activities $\texttt{retr}_\texttt{a}$ and $\texttt{retr}_\texttt{b}$ retrieve the friendships of Alice and Bob respectively form the on-demand data service `friends.friendOf`. The resulting tuples are correlated by the activity $\texttt{corr}_{\texttt{ab}}$ that looks for the common friends of Alice and Bob by means of a hash-join service. The profiles of the

common friends are retrieved from `friends.profile` by $\texttt{bind}_\texttt{p}$. Finally, only the friends that are of age are selected by $\texttt{filt}_\texttt{p}$ that invokes a filtering service.

This evaluation approach involves a trade-off between flexibility and cost. While using the resources of the service-based environment provides flexibility, this usage derives into multi-attribute costs. The resulting multi-attribute cost of a query workflow implementing a hybrid query must conform an SLA contract.

1.2 Hybrid query optimization challenges and objectives

Given a pair $\langle q, oo \rangle$ where q represents a hybrid query and oo represents an optimization objective of m cost attributes defined by an SLA contract, there is a search space $\mathcal{S}$ of equivalent query workflows that implement q through different control-flows, and a cost function $cost(qw)$ that maps each $qw \in \mathcal{S}$ to its corresponding cost $\langle x^1, x^2, ..., x^m \rangle$ where each $i \in [1 \,..\, m]$ denotes a cost attribute. The problem is to find the solution space of query workflows $\mathcal{S}_s \subset \mathcal{S}$ that best fit the optimization objective oo of the SLA.

Searching the query workflows whose composition of activities satisfies the SLA contract is a multi-attribute combinatorial problem that meets challenges already tackled in classic query optimization with new hypothesis to consider.

1.2.1 Challenges

There are three questions to consider in the design of a query optimizer: (1) How to define a small *Search Space* that includes the potential optimal query plans ? (2) How to estimate the cost in a way that the order among query plans is correct and the resulting costs are close to the real costs ? (3) How to avoid an exhaustive traversing of the search space for obtaining the global optimal plan ?

Hybrid query optimization meets these challenges with particular considerations:

— **Search space** contains query workflows sharing the equivalence property. This is that, given a hybrid query expression, each query workflow of the search space reaches a result that holds the semantics of the query through a different control-flow among activities. The enumeration of the complete search space requires computational effort that may be unfeasible as more data transformations participate.

The classic database techniques for optimizing queries such as join enumeration [SAC+79, SD89, OL90, PGLK97, RLL+01, MN08, BGLJ10], addition of sort-based operators [HFLP89, GM93], interesting orders [SAC+79, GD87], and the optimization in presence of access patterns [RSU95, FLMS99] are useful to enumerate the search space in a reasonable time. All these techniques are based on algebras (*e.g.* relational algebra), and are oriented to optimize the data-flow. This is not the case in service-based environments as we assume a lack of an algebra due to the possibility to add new data operators implemented by service operations belonging to different administrative domains. Besides, the multi-attribute costs and the preferences among them force to open the

search space in order to have more alternatives. Therefore, it is required a strategy in the absence of an algebra and oriented to optimize the control-flow of query workflows.

— **Cost model** proposes a cost function that maps a query plan to its corresponding cost by combining parameters such as CPU usage and I/O operations in centralized systems; and communication parameters and distributed resources usage in distributed systems [ML88].

The distributed systems introduce additional costs such as network usage [ML88] and resources price [SA80]. Besides, most of the cost models in the literature rely on data statistics as the core of the cost accuracy. For instance, the selectivity of attributes att_1,att_2, and the cardinality of the datasets ds_1, ds_2 determine the cardinality of the join between ds_1 and ds_2, *i.e.* $|ds1 \bowtie_{att_1=att_2} ds_2| = \frac{|ds_1|\cdot|ds_2|}{max(sel(att_1),sel(att_2))}$ [SS94]. The accuracy of such data statistics becomes difficult to achieve due to the autonomy of the services. For instance, in Garlic project [HKWY97, RS97, ROH99] it is assumed that cost parameters (*e.g.* access costs, cardinality, selectivity) are provided by distributed resources through wrappers. This assumption is difficult to hold in service-based environments as services belong to different administrative domains. Instead, cost attributes of query workflows result of the combination of QoS measures that describe the state of the resources (*i.e.* services, network). Therefore, it is required a cost model in the absence of data statistics that remains descriptive enough for determining the dominance among the query workflows in the search space.

— **Search strategy** is devoted to find the solution space of best plans into the search space. Traditional search strategies integrate both the search space characteristics (*i.e.* join shapes, operator heuristics, access patterns) and the cost estimation for applying mathematical programming algorithms and provide efficiency as the first class citizen of the optimization process.

The search strategies algorithms are (1)bottom-up such as dynamic programming [SAC$^+$79, ML88, MN08], greedy heuristic [Feg98], iterative dynamic programming [KS00, SY97], (2)top-down such as directed dynamic programming [GM93], iterative improvement [NSS86, Swa88], simulated annealing [IW87], (3)or a combination of both[MBHT96, DT07].

In any case the traditional algorithms are oriented to optimize the data-flow of well-known logical or physical operators regarding on the system performance, *i.e.* response time, throughput. In hybrid query optimization we are interested in the optimization of the control-flow that represents a wider space than the data-flow one. Besides, the search strategy has to consider the adoption of the SLA contract and the variety of potential cost attributes associated to the optimization objective, *e.g.* time, price, data quality, privacy, energy consumption.

1.2.2 Objectives

This work contributes to the optimization of data-centric service coordinations integrating both data requirements expressed as hybrid queries and SLA contracts. We aim to propose a hybrid query optimizer that adapts its optimization strategy according to an SLA contract.

The objectives of this work are:

1. Provide a query workflow generation with a control-flow perspective in order to open the search space with more alternatives in presence of multi-attribute costs.
2. Determine which query workflows dominate the search space through a cost function that assumes the absence of data-related parameters.
3. Propose an optimization strategy oriented to satisfy the optimization objective of an SLA contract that combines preferences among the cost attributes that are interesting for the user.

1.3 Main contributions

The main contribution of this thesis is the generation of query workflows implementing hybrid queries by coordinating services and whose cost is the closest to satisfy a Service Level Agreement expressed by the user. In detail, there are three contributions described below.

Control-flow based generation of query workflows. We propose an algorithm for generating the search space of query workflows implementing a given hybrid query. We provide the definitions and properties for ensuring the well construction and equivalence of query workflows. We define a series of data dependencies and rules for generating query workflows through sequential or parallel control-flows.

Build-time cost function. We propose a cost estimation based on the optimization objective of an SLA contract. We model a cost function considering the different ways the activities and services produce data through the control-flow. For optimization purposes, we propose a build-time cost function that is a relaxation of the first one. The build-time cost function explicitly evicts information only available at run-time and considers the activities' interactions across the control-flow.

Top-k^{qw} algorithm for selecting the best query workflows. We adapt a top-k algorithm for computing the solution space of query workflows representing the most suitable solutions for an SLA contract. In order to deal with no matter which combination of cost attributes, we use an abstraction of the cost attributes defined by the SLA contract. We adopt a weighted distance metric that considers the cost expectations and preferences among cost attributes.

1.4 Document organization

Chapter 2 Introduces the query optimization in distributed query processing, *i.e.* search space enumeration, cost model, search strategy. Afterwards, it reviews the state-of-the-art of works that consider QoS aspects for optimizing both queries and service coordinations.

Chapter 3 Presents the HyQoZ approach for optimizing hybrid queries with SLA contracts. It first introduces hybrid queries and the intermediate representation we adopt in form of data transformation functions. Then, it presents the optimization objective associated to an SLA contract and the definition of the query workflow cost w.r.t. the attributes of the optimization objective. Finally, the hybrid query optimization process we propose is presented.

Chapter 4 Describes the generation of alternative query workflows implementing a hybrid query. First, the query workflow model is presented along with the properties to ensure the well construction and equivalence of query workflows. Then, the composition of activities for generating equivalent query workflows by means of generation rules, and the generation algorithm are presented.

Chapter 5 Presents our cost estimation and search strategy guided by an SLA contract. It introduces the query workflow cost estimation and a relaxation for approximating the cost at build-time in absence of data-related parameters. Next, it presents top-k^{qw}, an adaptation of a top-k algorithm by means of a weighted distance metric that considers the optimization objective and the preferences among cost attributes.

Chapter 6 Presents HyQoZ, the service-based hybrid query optimizer; and its testbed HyQoZTestbed for enabling the access to HyQoZ under different configurations. First, it presents the hybrid query processing with the optimization performed by HyQoZ. Then, it presents the REST interfaces of the HyQoZ components, and the coordinations of the components by means of self-descriptive messages. Finally, it presents the implementation of HyQoZTestbed.

Chapter 7 Concludes this thesis and gives the perspectives for future research.

CHAPTER 2

Query optimization and QoS

This chapter reviews the query optimization in distributed query processing and works dealing with the optimization of queries and service coordinations considering QoS aspects.

The remainder of the chapter is organized as follows. Section 2.1 analyses the distributed query processing with particular attention on the problems that remain in nowadays systems such as data fragmentation, distributed query plans, and the trade-off between performance and resource usage. Section 2.2 stress the search space enumeration, cost estimation, and search strategy for optimizing queries in distributed environments. Section 2.3 analyses systems tackling the optimization considering QoS aspects such as mediation systems, web-based query systems, and service coordination systems. Finally, Section 2.4 concludes this chapter.

2.1 Distributed query processing

A query represents a data requirement over a collection of data sources. The query can be expressed in a structured language (*e.g.* SQL) and processed by a query processor. The components of the query processor transform the query expression into an optimal plan that is executed for obtaining the result.

Figure 2.1 presents the functional architecture of a distributed query processor[Kos00] and the phases where its components participate. There are components that can be distributed such as the metadata catalog [DGS+90, BAC+90], the database [GNnM+96, GDQ92], the query compilation/optimization [ML88], or the query evaluator [AH00, HFC+00]. Query processing is done in two phases: build-time where the query expression is transformed into an executable plan, and run-time where a query evaluator executes such a plan.

2.1.1 Build-time and run-time

At build-time, the query processor assumes a metadata catalog containing an up-to-date database of the schema information, access methods, and data statistics for optimization purposes. The query expression is parsed and mapped to a logical plan composing data operators. The logical plan is represented by a tree structure whose leafs represent data sources that can be either (fragmented) tables, streams, or data services. The intermediate nodes and the root, represent data operators.

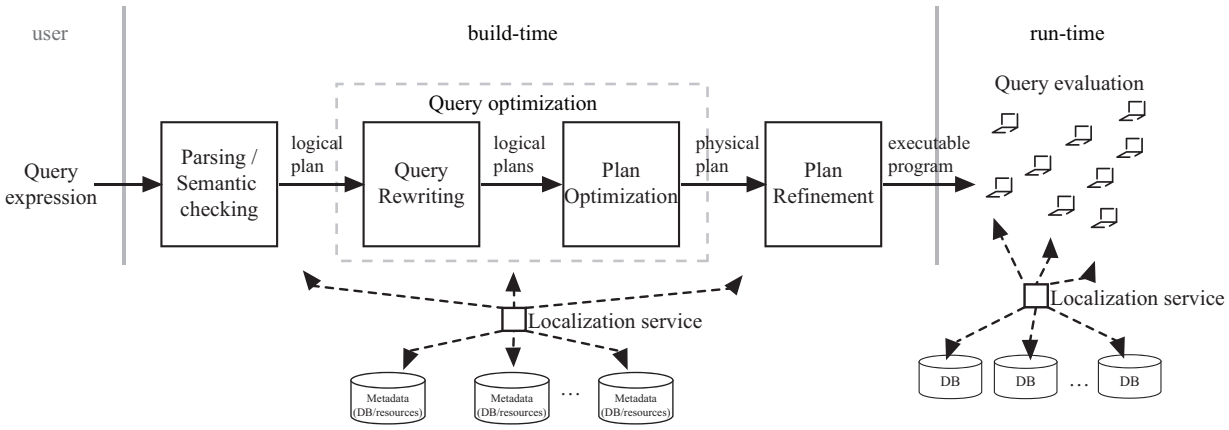


Figure 2.1 – Query processing [HFLP89]

The logical plan is used by the query optimizer that performs the query rewriting and plan optimization. Query rewriting generates alternative logical plans adopting a bottom-up approach [HFLP89, OL90] or a top-down approach [GD87]. The alternative logical plans are processed in the plan optimization phase for choosing the best (enough good) physical plan.

A physical plan is also represented by a tree. The nodes represent physical data operators, a.k.a. operator methods [Gra93]. Leafs represent access methods such as sequential scan or index scan. In the case of service-based queries, the only access method is the invocation of data services via standardized interfaces, *e.g.* *EVScan* in [GW00], wrappers [PGMW95, RS97, KTV97], binding patterns [BCD08]. Intermediate and root nodes represent physical operators like join methods[SD89, ME92] (*e.g.* nested-loop join, sort-merge join, hash-join) filtering methods, and others like sorting or grouping methods. The nodes have adornments as part of the cost estimation strategy. Such adornments denote, for instance, the intermediate size of data, the execution time required[ML86], and the computing resources to be used in distributed environments [GHK92, KTG06], *e.g.* memory, disk, CPU's, service instances, communication. The adornments are computed by cost functions that consider the physical data operators, the order among them, and the state of data and computing resources.

The resulting query plan is passed through a plan refinement that translates it into a procedural plan by assembling pieces of code and then producing a final program. This plan refinement can be seen as a translation of the query plan in a program readable by the query evaluator.

At run-time, the query evaluator executes the plan leveraging of distributed resources for accessing data sources, performing query tasks, and delivering the result. Data access, either metadata or data, is done through a localization service which knows data's location and fragmentation. In shared-nothing [1] architectures[AH00, TD03], like service-based environments, the evaluation relies on autonomous distributed resources (*e.g.* services) such as data sources, catalog, and the computing resources to perform query tasks. In contrast with query plans in centralized environments, distributed query plans contain communication primitives among computing resources such as send and receive that induce communication costs.

1. Multiple computing units with private memory and storage

2.1.2 Data fragmentation

The role of data fragmentation is to spread data among distributed storing resources in order to allow the parallelization of data access and query evaluation. The design of the fragmentation of a relation R in n resources, is about to chose a horizontal fragmentation, *i.e.* $R = \bigcup_{i=1}^{n} R_i$, or vertical fragmentation, *i.e.* $R = \bowtie_{i=1}^{n} R_i$. Vertical fragmentation is usually more complex for query splitting than horizontal fragmentation but it has a better performance with a careful design following a fragmentation strategy[DG92] like round-robin, range, and hash.

The quality of the fragmentation strategy relies on the symmetry of the data distribution or skewness [2] that balance the access to the resources. For instance, suppose that the fragmentation of tweets is done following a range strategy w.r.t. their hash-tags; if a hash-tag becomes a trend-topic (*e.g.* #Snowden, #Brazil2014) the throughput of the storing resources will be harmed because all the queries over such trend-topics require to access them. The round-robin strategy assigns the i tuple to the resource $i \ mod \ n$ among n resources. For instance, one may found many of the #Snowden tweets in every resource, and thus the query "*give me all the #Snowden tweets*" requires to access every resource allowing independent parallel access with the corresponding communication cost and resource usage. Finally, the hashing strategy applies a hash function to a tuple to assign it to a bucket (*i.e.* storing resource). The hash function may be location aware, for instance, the tweets related to the trend-topic #CumbreTajin2014 should be stored near to Mexico.

In nowadays services providing data, the fragmentation is not explicit. For instance, the friendships of someone are horizontally fragmented in serveral social network services (*e.g.* facebook, google+, twitter). Such a fragmentation can be seen also as a vertical fragmentation as the schemas are heterogeneous and maybe related by a single attribute (*e.g.* email). For instance, a data service may offer the friends' profile with interests information (*e.g.* reddit), while other data service may offer the friends' profile with professional information (*e.g.* linkedin). In such a case, the combination of data fragments is pulled-up to the query expressions.

2.1.3 Distributed query evaluation

Distributed query evaluation exploits the computing resources available in distributed environments either to access data or to perform query tasks. There are three approaches to evaluate queries in distributed environments. *Query-shipping* considers the capabilities of the computing resources and deploys query tasks to them. *Data-shipping* assumes that data consumers have enough storage and computing capabilities to retrieve, store and process data locally. Both have a response time and resource usage trade-off depending on the distribution of query tasks, data fragmentation, the hardware capabilities of computing resources, and the rate of incoming queries. There is a third approach *hybrid-shipping* [FJK96], which leverages of the advantages of the others under memory conditions of resources and data fragments sizes.

2. Skewness is a measure that denotes the asymmetry of data distribution. As closest to zero is the skewness, the best is the data distribution.

The service coordination approach for evaluating queries[BKK+01, CvVSCB09] adopts the use of both query-shipping and data-shipping. A dataset is retrieved from its data service (*i.e.* data shipping) and it is passed to a computing service (*i.e.* data shipping) along with the query task to process it (*i.e.* query-shipping).

2.1.3.1 Distributed query plans

A distributed query plan defines how query tasks are organized and distributed among the computing resources. The distribution of query tasks is limited by three conditions [GHK92, Mah10]: (1) resource contention given by the competition among query tasks accessing resources, for instance, in shared-nothing architectures the network and the computing resources are in contention; (2) data dependency given by logical relations among data operators, for instance a data operator filtering a dataset depends on the retrieval of such a dataset; and (3) data fragmentation (skewness), for instance, a dataset fragmented horizontally can be filtered by two query tasks in independent parallel.

Query tasks are organized by distributed query plans that can privilege either the performance or the resource usage depending on the granularity of the plan: inter-query, intra-query, and intra-operator. Next, these granularities are explained along with figures where a query Q_i is evaluated by a query plan from the input data represented by parallelograms and through data operators represented by rectangles. The dashed rectangles represent the computing resources performing the query tasks.

— **Inter-query** distributes multiple queries implemented each by a single query task (see Figure 2.2). For n concurrent queries, there are n computing resources executing each in parallel.

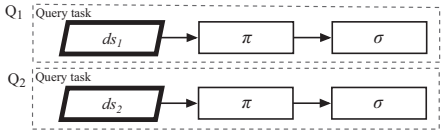


Figure 2.2 – Inter-query distribution

— **Intra-query** assigns a portion of the required data operators to a single query task. The query tasks execution is organized depending on the logical dependencies among data operators (see Figure 2.3). Dependent operators are executed by sequential query tasks (possibly) through a pipeline, *i.e.* as partial results are issued, the following query task processes them (*e.g.* tuples in the outer operand of a nested-loop join can be pipelined). Independent operators belonging to different data-flow branches are performed by query tasks in independent parallelism, *e.g.* the retrieval from two data sources (or fragments) ds_1 and ds_2 is performed in parallel. For a single query with n data operators, it can be performed by at most n computing resources.

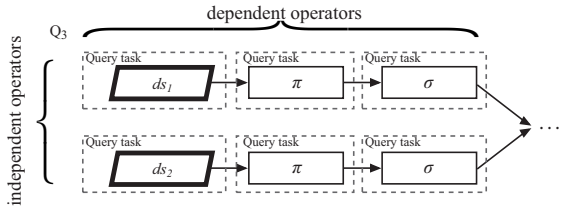


Figure 2.3 – Intra-query distribution

— **Intra-operator** splits a single data operator into several query tasks performed in independent parallelism (see Figure 2.4). Each query task transforms a dataset fragment. For instance, a dataset distributed horizontally in k disjunctive fragments can be projected and filtered by $2k$ computing resources. In the case of overlapping horizontal fragments, it is required an additional query task to perform the union. For $n <= m$, a single query with n data operators can be performed by m computing resources.

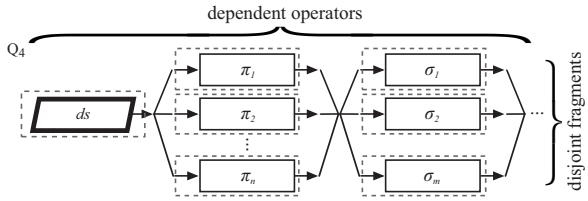


Figure 2.4 – Intra-operator distribution

The distributed query plans can provide best performance (*i.e.* response time, throughput) if more resources are used, or inversely, less resource usage reducing the performance.

2.1.3.2 Response time, throughput, and resource usage

The distribution of query tasks induces the trade-off between resource usage and performance. Such a trade-off is described by the notions of speed-up and scale-up. The speed-up notion describes the response time improvement as more resources participate. The scale-up notion describes the throughput improvement as more resources participate. The improvement is stabilized given the parallelizable portion of query tasks. In practice, the speed-up and scale-up are harmed by the network contention due to message passing among computing resources. This trade-off induce the need to define query optimization strategies to find the Pareto optimal by discarding plans either time optimal or resource optimal.

For instance, in [GHK92] the problem to reduce the response time is formulated with constraints on the throughput of the system, in particular the CPU usage. Through these constraints, the trade-off between response time and resource usage is parametric. This is done by defining a factor of throughput degradation and a limit on the cost benefit ratio. If R_p and T_p respectively represent the resource usage and the response time of the optimal plan p, the solution of the optimization is a plan p' if (1) $W_{p'} \leq k * W_p$, *i.e.* the plan p' requires almost k times the work required by the optimal W_p; and if (2) $\frac{T_p - T_{p'}}{R_p - R_{p'}} \leq r$,

i.e. the ratio of the response time and resource usage is almost r. Therefore, the best plan is not the one that has the best response time (*i.e.* p) but the one that respects the parameters k and r (*i.e.* p').

Looking at the parallelization of tasks, in [GI96, GI97], they look for the optimal independent and pipelined parallelization of the query evaluation in a shared-nothing architecture. Each query (sub-)plan is modeled as a requirement of multiple hardware capabilities. The computing resources in turn, offer a collection of hardware capabilities typified as time-shared (*i.e.* CPU, disk, network) and space-shared (*i.e.* memory). This model allows to conduct the optimization w.r.t. the available hardware capabilities.

2.2 Query optimization

The goal of query optimization is to obtain the solution space of query plans from a search space of possibilities. The solution space contains the plans whose costs represent the best (enough good) alternatives for implementing the query.

From this goal, three problems arise. (1) How to define a small *Search Space* that includes the potential optimal query plans ? (2) How to estimate the cost in a way that the order among query plans is correct and the resulting costs are close to the real cost ? (3) How to avoid an exhaustive traversing of the search space for obtaining the global optimal plan ?

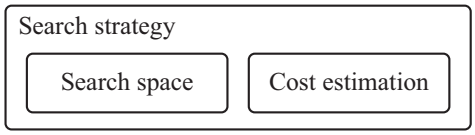


Figure 2.5 – Query optimization problems

In general such problems are tackled in an integrated search strategy (see Figure 2.5) in order to provide efficiency while they are tightly coupled to the data model, data operators, physical layer management, and cost model. Therefore, the extensibility of such solutions become difficult as new data models, applications, and costs arise.

These problems are wide and thus we scope on the challenges related to the hybrid query optimization problem. In particular we next discus (1) the search space computation and the considerations for generating the potential optimal plans, (2) the cost estimation regarding on the trade-off between response time and resource usage, and (3) the search strategies for traversing the search space.

2.2.1 Computing the search space

The search space is a collection of equivalent plans derived from a query expression, *e.g.* SQL. A query expression is rewritten in a set of equivalent logical plans and each of these also in a set of equivalent physical plans (see Figure 2.6). The logical plans vary on the logical operator order along the dataflow and the physical plans on the physical operators implementing each logical operator. The size of the search space grows exponentially given this double combinatory. The enumeration of the complete search space requires computational effort that can be reduced following join shape policies and adopting operator motion heuristics.

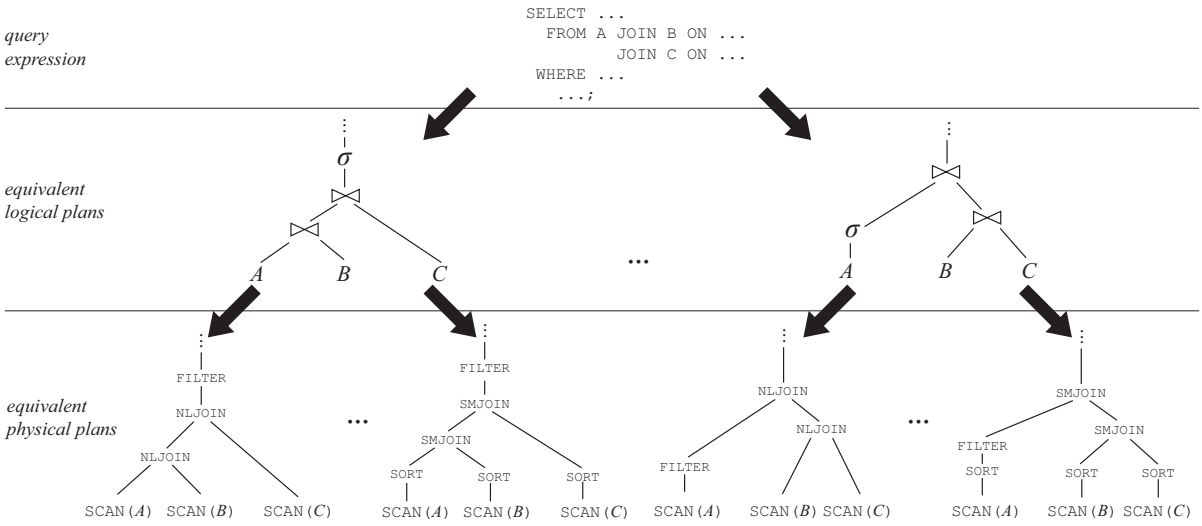
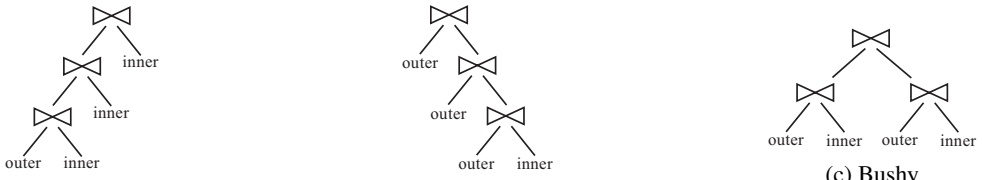


Figure 2.6 – Search space depiction

2.2.1.1 Join orderings

The order in which joins are organized determines the cost of a query plan. Finding the best join orderings is a problem that has been studied from the last four decades [SAC+79, SD89, OL90, PGLK97, RLL+01, MN08, BGLJ10] due to its complexity that is known to be NP-Hard [CM77, IK84, Swa88, SMK97] and because of the diverse join semantics (*e.g.* outer-join [RGL90, MN08]). There are polynomial time algorithms to find sub-optimal plans [IK84] or to find optimal plans under specific characteristics (*e.g.* acyclic queries [KBZ86]).



(a) Left deep (b) Right deep (c) Bushy

Figure 2.7 – Join shapes

One of the principles to make an efficient join orderings enumeration is the adoption of a join shape. A join shape (see Figure 2.7) indicates how the join operators are connected among them and with data sources. Left-deep shape composes query trees where the inner side is always a data source and the outer side is either a join operator or a data source (see Figure 2.7a). Right-deep shape composes query trees where the outer side is always a data source and the inner side is either a join operator or a data source (see Figure 2.7b). Bushy shape (a.k.a. composite inners [HCL+90]) allows the two operands to be either join or data sources (see Figure 2.7c).

Both linear and bushy shapes represent good opportunities to find optimal plans [IK91, SMK97] as they enable –under different circumstances– the pipeline parallelism (*i.e.* left and right-deep) and inde-

pendent parallelism (*i.e.* bushy). In particular for distributed environments, bushy shapes provide better opportunities [CYW96, FJK96] because there are more parallel joins running in independent computing resources.

Although the enumeration of bushy shapes is as expensive as a search space without constraints [OL90], it is feasible for small instances of the problem by applying either deterministic methods [VM96] or randomized methods [IK91]. Besides, bushy shapes avoid Cartesian products as shown in [FLMS99] in the context of queries over data sources with access patterns.

2.2.1.2 Operator heuristics

The basic heuristic to move operators along data-flows is to push-down inexpensive operator methods in order to reduce intermediate data size. This reduction can be done either horizontally (*e.g.* filterings over indexed attributes, joins/products between small datasets), or vertically (*e.g.* projections). As consequence expensive operators (*e.g.* filterings over un-indexed attributes with low selectivity, nested-loop joins with low selectivity, products) are pulled-up getting smaller datasets. These heuristics are typically combined with the cost of predicates and user-defined functions [Hel94].

The order of data is also important for adopting sort-based operator methods, *e.g.* sort-merge join, binary search filtering, duplicate elimination. Otherwise, the datasets must be scanned in sequence (*e.g.* nested-loop join, linear filtering). The inclusion of sort-based methods might include additional methods (*i.e. glue STARs* [HFLP89], *enforcers* [GM93]) when there is no an explicit order in the query or when the access method does not provide such an order (*i.e.* sequential or hash-based access).

Operators like order-by, group-by and join (sort-merge join) introduce *interesting orders* [SAC$^+$79], a.k.a. operator properties [GD87]. The principle is to look for query plans with interesting orders that represent better alternatives than the ones without sort-based operator methods.

The rules to move or add group-by operators without affecting the query semantics have been studied in [CS95] and more recently in [BGLJ10]. In particular, [BGLJ10] introduces polynomial heuristics for the motion of group-by, joins and semi-joins; and envisages the inclusion of anti-joins and outer-joins by considering their particular associative restrictions [GLR97, RLL$^+$01].

The addition of either order-by or group-by operators induces more alternatives that might harm the optimization process as it is a NP-hard problem even in a simplified form [GSDB11]. In [GSDB11] the enumeration of the search space with interesting orders is pruned through a heuristic based on favorable orders. An order is favorable if its cost maximizes the benefit for the plan by considering the information about indexing structures of the involved attributes.

2.2.1.3 Access patterns

The access patterns [RSU95, DL97] arise in the context of (1) data wrappers [RS97, KTV97] (a.k.a. translators [PGMW95]) in the mediation systems [Wie92, ZRV02], (2) User-Defined Functions (UDF) [CS96], and (3) foreign functions [CS93]. Access patterns introduce special considerations for enumerating the search space of query plans as the access to the output of such functions (and wrappers) is only

possible if all the bound attributes are provided. The notion of *binding pattern* is used for describing how to access functions and data provided by wrappers.

Binding patterns

A binding pattern (a.k.a. adornment [Ull88], predicate pattern [ACP96], view [Ull97, KP09]) describes which bound attributes are required as input to obtain a series of free attributes as output. Binding patterns, queries and query plans are represented by the notation of Datalog programs:

$$head(B_1?, ..., B_m?, F_{m+1}!, ..., F_{m+n}!) :- p_1 \wedge ... \wedge p_k.$$

The head denotes the binding pattern or a query with its m bound attributes adorned with '?' and the n free attributes adorned with '!'. The body contains a series of k conjunctive predicates whose true value produces the result. Each predicate p_i denotes either a fact into the Extensional Database (EDB) or a binding pattern into the Intentional Database (IDB).

Suppose the parenthood example used in [RSU95, DL97]. The EDB contains facts of the form $parent(C, P)$ that denotes the parenthood between a parent P and his child C. Such data are accessible via two binding patterns in the IDB. (1) $bp_1(C!)$ has no bound attributes and provides all the individuals how have parents in the EDB. (2) $bp_2(C?, P!)$ requires C to provide values of P, *i.e.* it gives the parents P of a given child C. Now consider the query *Which are the grandchildren for the individual g ?* denoted by $q_1(C!, g)$. Figure 2.8 presents a query plan of $q_1(C!, g)$ in the left and the bottom-up evaluation in the right.

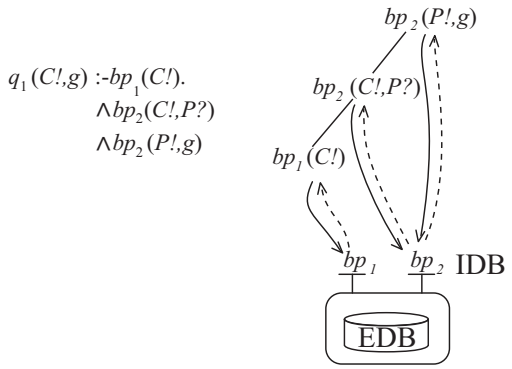


Figure 2.8 – Query evaluation with access patterns

First, the bp_1 is used to retrieve all the potential grandchild C, then bp_2 is used to find the parents P of the potential grandchild C, and once again bp_2 to retrieve the parents P that are child of g.

Safe and unsafe queries

A query Q is $safe$ if it has an equivalent query plan P where all the bound attributes related to the binding patterns mentioned in Q are bounded either (1) to a free attribute of a positive predicate of the plan P, or (2) to a constant.

In the context of mediation systems [CD96, DL97], there is a class of queries that allows to express $unsafe$ queries (*i.e.* queries with bound attributes not bounded to neither a free attribute nor a constant). This is, there is not enough information into the query expression for constructing a plan.

$Unsafe$ queries can be transformed into $safe$ queries[Li03, NL04, Nas04]. The query evaluator assumes that the (non-mentioned) binding patterns in the IDB and the facts in the EDB may participate during the evaluation in a natural-join way[LRO96, LC00]. This is, if a binding pattern included in the query has a free attribute whose abstract domain is the same than an attribute of a non-mentioned binding pattern, then the natural-join of both may provide useful bindings for answering the query.

While $unsafe$ queries provide expressiveness with incomplete information, the assertions to realize natural-joins of bound/free attributes become difficult to manage in a context such as web [LRO96, YKvBO03, SMWM06, BCD08]. We consider *safe* queries more appropriate for service-based environments even if they demand to developers the knowledge about the binding patterns describing data sources they are interested in.

Query plan enumeration

The alternative plans for queries with binding patterns is smaller than plans for queries with commutative joins presented before (*cf.* Section 2.2.1.1) because data dependencies tend to produce linear plans. For instance, answers for conjunctive queries with binding patterns are found in polynomial time [RSU95].

This problem becomes more complex when several binding patterns are allowed for a single data source. For instance, suppose the binding patterns $bp_2(C!, P?)$ and $bp_3(C?, P!)$ are associated to a single data source identified by $parenthood$, and the query dialect allows to pose queries over the identifier $parenthood$. In such a case the evaluation engine has to look for the feasibility of queries by considering the combinations of binding patterns of the data sources involved in the query [FLMS99].

The avoidance of unnecessary accesses (*i.e.* minimal closure) is a property of optimal plans of queries with access patterns. In [CM08] the authors tackle the problem by traversing a dependency graph $d-graph$ that represents all the possible ways to evaluate the query. The process is to (1)construct the $d-graph$ of data dependencies (arcs) among all the data sources attributes (nodes), (2)clean the $d-graph$ by deleting arcs that represent unnecessary dependencies and attributes; and (3)generate the optimal query plan by finding the path with minimal number of accesses. This approach is inspired from previous works [LC00, LC01b, LC01a] where the database is represented by a hypergraph that is reduced until an alternative plan is obtained. This is also the approach of the GYO reduction algorithm [GST83] and the adaptation for binding patterns done by [CVVSC12] in the context of hybrid queries.

2.2.2 Cost estimation

The cost is given by a function that maps a query plan to its corresponding cost. This cost function is a combination of parameters such as CPU usage and I/O operations in centralized systems [ML86] and communication parameters and distributed resources usage in distributed systems [ML88].

The distributed systems provide efficiency and scalability while the use of distributed resources introduces additional cost such as network [ML88] and resources price [SA80]. Besides, the accuracy of data statistics becomes more difficult to achieve as the autonomy of the computing resources increases. For instance, in Garlic project [HKWY97, RS97, ROH99] it is assumed that cost parameters (*e.g.* access costs, cardinality, selectivity) are provided by the distributed resources by means of wrappers.

2.2.2.1 Data statistics

Data statistics are used as parameters of the cost function. For instance, the selectivity of attributes att_1,att_2, and the cardinality of the datasets ds_1, ds_2 determine the cardinality of the join between ds_1 and ds_2, *i.e.* $|ds1 \bowtie_{att_1=att_2} ds_2| = \frac{|ds_1|\cdot|ds_2|}{max(sel(att_1),sel(att_2))}$ [SS94]. Data statistics are stored as part of the system database and their computation implies overhead as queries themselves.

In nowadays data-centric distributed environments, the assumption that distributed resources provide information to compute the cost is not realistic. It implies either (1) the harvesting and materialization of data statistics[ACP96], (2) the implementation of an interface (*i.e.* wrapper) in the data source side for accessing such an information[ROH99], (3) or the active participation of the data sources during the optimization process[TRV95]. In HERMES project [ACP96], and further in [ZL98], this assumption was deprecated accepting the autonomy of data sources is a deterrent for traditional cost-based query optimization.

Another way to gather data statistics is query sampling over data sources. For instance, to gather the cardinality of a data source *ds*, it is executed the query *select count(*) from ds*; to gather the response time for a given input value *val1*, it is executed the query *select * from ds where att1 = val1*. There are more costly statistics like the estimation of the number of distinct values of an attribute (*i.e. select distinct att1 from ds1*) that requires the elimination of duplicates, and the gathering of the two highest or lowest values (*i.e. select distinct att1 from ds order by att1 asc stop after 2*, *select distinct att1 from ds order by att1 desc stop after 2*). Such routines have been implemented in centralized and distributed databases (*e.g.* [ML88, ML86, Col12]) for internal usage of the query processor or for allowing the user to force the optimizer to select query plans with the desired properties.

2.2.2.2 Extensible cost model

The System-R framework integrates the search space and the cost estimation into the search strategy as a monolithic module. Such a framework is not adaptable when new data models and operators arise (*e.g.* big data processing [Ewe12]), and as new costs have to be considered (*e.g.* resource pricing [LSW+12], data pricing [KUB+12], energy [KB12]). This evolution demands extension of the query optimizer that is difficult to achieve in the System-R framework as it is data-model dependent[Bat86].

In Genesis [Bat86, Bat87], the extensibility is possible thanks to a functional data model that enables the addition of new data types and operators. For instance, in Genesis [Bat86, Bat87] data-object transformations are represented as function compositions in a high level expressions named GDL-expressions (Genesis Data Language). These expressions are mapped to low level expressions named r-expressions

whose domain are file-records.

The algorithms (*i.e.* operator methods) of abstract operators are written as a composition of basic operators (*i.e.* physical operators). Analogously, the cost function of an abstract operator is the composition of the cost functions of its basic operators. This is applied recursively for more sophisticated cost functions of algorithms composed by abstract operators.

2.2.3 Search strategy

In System-R [SAC+79] the enumeration of possible query plans is tackled with a dynamic programming algorithm for finding the possible left-deep joins, and interesting orders (a.k.a. operator properties[GD87]) for determining which physical operators are convenient for the data-flow according to the benefit they provide to the plan.

This algorithm has been widely used as it is good enough for queries with few data sources. Nevertheless, as queries become more complex such as in OLAP applications[IK84] and deductive databases [RU93], the problem must be tackled with algorithms that are efficient while sometimes only sub-optimal plans are achieved.

There are two approaches to enumerate alternative plans[GMUW02]: bottom-up and top-down. Next, we present these approaches along with algorithms implementing them.

2.2.3.1 Bottom-up approach

A bottom-up algorithm (*e.g.* [SAC+79, MN08]) first treats with smaller sub-plans (*i.e.* data sources first) and compute the possible compositions and costs with the reminding sub-plans (see Figure 2.9).

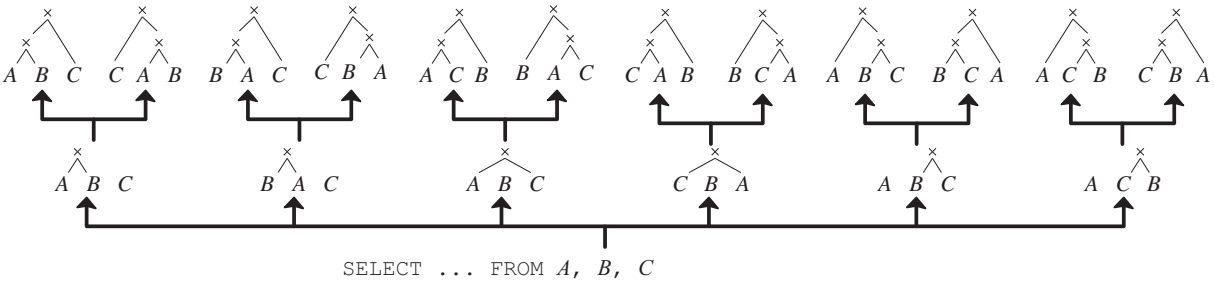


Figure 2.9 – Bottom-up enumeration

For the three relations A, B, C, the first iteration composes each pair of relations and apply the commutative property producing six sub-plans. Then, the second iteration composes the six resulting sub-plans with the reminding relation and also applies the commutative property producing twelve alternative plans. Next we enumerate some of the algorithms adopting the bottom-up enumeration approach.

— **Dynamic programming** has the principle of divide and conquer under the principle of optimality. For a (sub-)query Q involving n relations $\{R_1, \ldots, R_n\}$ to be joinded, the optimal plan of n relations is composed by an optimal plan of $n-1$ relations. At each iteration $i \in [2\,..\,n]$ there are

$i-1$ relations composed $\{R_1, ..., R_{i-1}\}$ and $n-i-1$ to be composed $\{R_i, ..., R_n\}$. At the end of the iteration, the generated plans are memoized [3] to use them in further iterations as a cache.

Dynamic programming has been proposed in the System-R [SAC+79] and its version for distributed databases R* [ML88]. The search space is constrained to consider left-deep shapes where the most of the optimal plans can be found [GMUW02]. Besides, the size of a search space with linear shapes is $(n-1)^2$ while the one with bushy shapes is $(n^3-n)/6$ [OL90]. In the worst case of star queries –where $n-1$ sources are all joined with a single one– the size of the search space is $(n-1)2^{n-2}$.

The algorithm has been improved during thirty years until the inclusion of bushy shapes with Cartesian products that result convenient for a small number of tuples[MN06, MN08]. The efficiency of the algorithm is proven to be better w.r.t. the naive generation. Nevertheless, for queries with $n > 10$ relations the algorithm is unfeasible [SMK97].

— **Greedy algorithm** has the same structure as the dynamic programming algorithm. The difference is that at each iteration $i \in [2\,..\,n]$ for n relations, it only takes the cheapest plan P_i^j such that $cost(P_i^j) = min\{cost(P_i^j) \mid j \in [1\,..\,m]\}$ for the m alternative sub-plans at i.

This algorithm may produce sub-optimal plans [SMK97] due to the assumption that an optimal sub-plan P_i^j will contribute to find the global optimal plan P_n. In [Feg98] the GOO (Greedy Operator Ordering) algorithm performs the enumeration in polynomial time, *i.e.* $O(n^3)$. It is compared with with randomized algorithms (*i.e.* iterative improvement) in order to show that the delivered plans are better because the algorithm considers bushy-trees that is proven to provide better optimal opportunities in a wide search space[FJK96, FLMS99]. Nevertheless, GOO always outperforms such randomized algorithms while the delivered plans still being potentially sub-optimal.

— **Iterative dynamic programming** [KS00] integrates the benefits of both dynamic programming and greedy algorithms into a series of eight algorithms some of them based on the patent [SY97]. Such variants consider the limitations of memory and a time threshold. If there is no enough memory/time the algorithms adopt a greedy heuristic IDP_1 in order to reduce the search space and assuming the intrinsic sub-optimality. Otherwise, the IDP_2 is chosen and thus better quality plans. The IDP_2 variant also uses a greedy heuristic but instead to compose only two sub-plans, it takes the reminding ones and makes a pair-wise composition of them. In this way, IDP_2 chooses the best sub-plan to explore allowing to produce bushy shapes. The eight variants of IDP arise by two more considerations besides the memory/time constraints: (1) the size of the intermediate results, and (2) the number of relations contained in sub-plans.

2.2.3.2 Top-down approach

A top-down strategy is based on transformation rules [RH86, PGLK97] and it considers first bigger expressions. The input is a logical (sub-)plan and it is transformed by applying logical-to-logical trans-

3. Memoization is about materialize input-output pairs of a function in order to avoid the computation in succeeding iterations. This technique is not exclusive neither of dynamic programming nor bottom-up approaches (*e.g.* [DT07])

formation rules for producing an equivalent logical plan. A logical (sub-)plan can also be transformed into a physical plan by applying logical-to-physical transformation rules. For instance, consider Figure 2.10 that shows these two transformations.

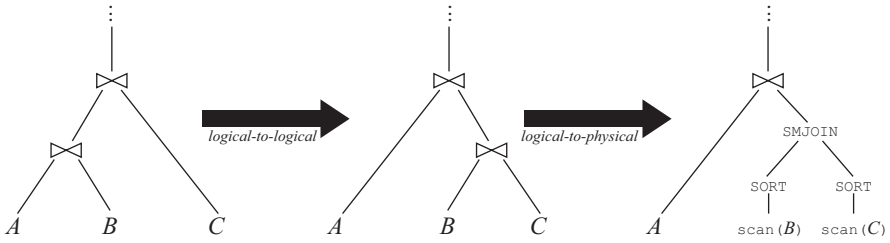


Figure 2.10 – Application of transformation rules

The logical-to-logical transformation in Figure 2.10 is based on the associative property of joins, *i.e.* $(A \bowtie B) \bowtie C = A \bowtie (B \bowtie C)$. The logical-to-physical transformation maps the sub-expression $B \bowtie C$ into a plan that implies the sort-merge join method [BE77] `SMJOIN` and the required *enforcers* methods [GM93] (a.k.a. *glue STARs* [HFLP89]) such as `SORT` that enables the execution of a sort-merge join. These transformations determine the search space and thus they constraint both the required computation effort to achieve an optimal plan and its quality.

— **Directed dynamic programming** is the top-down version of the System-R dynamic programming. In particular, [GM93] proposes a top-down variation of dynamic programming algorithm that is conjugated with goals. This turn the algorithm needs-driven [GM93] instead of possibilities-driven such as System-R [SAC$^+$79] does. This algorithm, produces equivalent logical and physical plans only for largest (sub-)plans that satisfy logical an physical properties instead of enumerating all the equivalent plans that may not satisfy such properties.

— **Branch-and-bound** explores the spanning tree where the root represents the original logical plan P with a cost C. During the exploration, there is an upper bound named incumbent that represents the best cost already known. The first incumbent I is defined by the cost C. Next, a series of n transformation rules are applied to the plan P for obtaining alternative plans $P'_1 \dots P'_n$ and the costs $C'_1 \dots C'_n$. The costly plans $\{C_i > I | 1 \in [i \,..\, n]\}$ are eliminated and thus the spanning tree is cut. The reminding plans are maintained as potential alternatives. The incumbent is updated with the minimum cost of the plans $I = min(C'_1 \dots C'_n)$. This bound is kept in subsequent explorations until there are no more sub-plans to explore, or discarded by backtracking if such explorations do not represent feasible alternatives.

— **Two phase optimization** [Swa89, IK90, IK91] uses two randomized algorithms that adopt the best profit heuristic of greedy algorithms. (1) The *iterative improvement* [NSS86, Swa88] produces a local optimal plan and (2) the *simulated annealing* [IW87] explores the neighborhood in order to look for a better solution.

The iterative improvement covers rapidly a large subset of the search space by step-wise traversing of the neighbors. For a given plan, the algorithm moves to the better neighbor (if there is)

and repeats this iteratively until a valley (*i.e.* local optimal) is reached, or until a time threshold is exceeded. Once iterative improvement finishes, the simulated annealing uses probabilities to determine which of the neighbors may provide a better solution in another valley. These two algorithms are complementary. While iterative improvement gives quickly a local optimal, the simulated annealing provides (when possible) global optimality because it moves to a neighbor even if it is not better than the current solution but it represents a good opportunity to reach a better one.

2.2.3.3 Hybrid approach

Neither bottom-up nor top-down approaches represent an absolute solution for efficiently enumerate the spectrum of interesting query plans. Thus there are works adopting these two approaches in both the query rewriting and the plan optimization phase.

The EROC system [MBHT96] combines both bottom-up for the enumeration of join orders based on the Starburst [PHH92] and the top-down optimization algorithm of Volcano [GM93]. In [DT07] the authors propose a branch-and-bound enumeration of left-deep and bushy shapes with or without Cartesian products. It is based on logical-to-logical and logical-to-physical transformations and the memoization principle taken from bottom-up dynamic programming. The memoization serves as a cache strategy to store and compare plans already optimized, and avoids duplicates. The top-down enumeration gets the benefits of memoization and also one of the drawbacks of dynamic programming because of the memory usage. Nevertheless, the branch-and-bound prunes the spanning tree reducing then the memory usage.

2.3 Optimizing queries in service-based environments

This section presents systems tackling the optimization of queries in service-based environments. The systems we consider include mediation systems, web-based query systems, and service coordination systems. Through the description of systems we highlight the dimensions characterizing the hybrid query optimization problem listed in Table 2.1.

Dimension	Description
First class objects	Service coordinations can solve either data requirements or functional requirements, *e.g.* e-business applications. Then, the first class objects can be either *data* for works considering services as data providers, or *service operations* for works considering services as computing providers.
Cost attributes	The purpose of the optimization and the cost model are defined by the cost attributes. For instance, time, price, reliability, data quality, etc.
Optimization approach	In presence of multiple cost attributes, the approach to deal with can be either *needs-driven* when the user needs are considered, or *possibilities-driven* when the optimizer looks for the best solutions according to the system state.
Adaptation	As long as a plan is being executed, the dynamic nature of the environment makes fluctuate the cost at run-time. The adaptation may be considered or not as part of the optimization depending if there are long-running queries.
Service selection	A service interface (*i.e.* API) can be implemented by a single service instance or by multiple service instances. This assumption leads to the service selection problem.

Table 2.1 – Dimensions of hybrid query optimization problem

We consider mediation systems and web-based queries where the first class objects are data. We also consider the service coordination systems where in general the first class objects are service operations. These works principally consider the time and price as optimization dimensions. There are other works considering QoS (possibly conflicting) aspects that we model as cost attributes.

For tackling the multiple cost attributes there are two main approaches (*i.e.* possibilities-driven and needs-driven) adopted by these systems. There is also the profile-driven approach that considers user nominations and is a specialization of the possibilities-driven approach.

Because of the dynamic nature and democratization of service-based environments, the adaptation and service selection are important dimensions of the hybrid query optimization problem. Nevertheless, we left aside these dimensions for future investigations.

2.3.1 Optimization approaches

Besides the performance related costs (*i.e.* response time, throughput) of classic query optimization, the use of services in service-based environments brings QoS aspects to consider as part of the cost. This cost is composed by multiple cost attributes. For instance, the communication cost due to message passing among resources leads to the usage of the network and bandwidth consumption, and has an impact into the response time. Depending on the speed-up and scale-up bounds, as much resources are used the most are enhanced the response time and throughput. This enhancement has a compromise with QoS aspects besides the ones related to the network [ACH98]. Such QoS aspects (a.k.a. non-functional aspects [OEtH02]) have to be considered into the hybrid query optimization.

Immediately related to the usage of resources, there is the financial cost either from the usage of computing resources [SA80, DSL+08, QRLCV11, LSW+12], data resources [BHS11, KUB+12, KUB+13, AKHLS13], storage [LBN07, DSL+08, KHAK09]. There is also the index storage [GG03] and energy consumption [LP09, HSMR09, Kon11, LBZ+11, CS12, KB12, FP13]. The notion of services as data and computing providers have other QoS aspects to consider[Men02, BND+04] such as security, availability, throughput, reliability, and reputation. The quality of data is also important so the data provenance [KIT10], completness [MSV+02], timeliness and accuracy [NLF99], and data coverage [NK01].
In our context, these QoS aspects are modeled as cost attributes. To deal with such a cost there are two approaches.

— **Possibilities-driven approach** is aimed to answer queries according to what is possible to do given the current state of the system. In [PY01], the system has an interface where the user can chose among the alternatives of the Pareto curve, *i.e.* optimal and near optimal alternatives.The construction of such a Pareto curve is done by a dynamic programming algorithm in polynomial time [PY00] under the assumption that the cost functions of the attributes are linear.

  In the context of queries performed over distributed data sources, [VRM04] studies the way to maximize the data coverage and minimize the execution price of the plan. They look for a percentage of the top results (*i.e.* *Top XX%*) to construct a quasi Pareto optimal whose accuracy is a function of *XX%* and it is proportional to the resulting cost.

— **Needs-driven approach** defines the optimization purpose according to the expectations of users about the query evaluation cost (*i.e.* response time threshold, price budget). Several works have adopted this approach [SAL96, Mad96, YNGM00, YKvBO03, YB08, FK09]. In particular, the Mariposa project [SAL96] introduced the query and storage optimization by considering the user requirements and the available resources on different administrative domains with heterogeneous service levels such as hardware capabilities, stored data, and pricing. Instead to look for a global optimal deployment of query tasks or data fragments, Mariposa adopts an economical model that guides an equitable assignation of resources. This induces a price/quality ratio for the selection of the resources for evaluating a given query. The resources decide which data fragments to buy or sell, and which query fragments to bid. Like in trading market, resources might not accept the task because of profit reasons or might not have enough capacity.

  In [YNGM00], it is addressed the problem to find the trade-off among time, price and data coverage of query plans over overlapping data sources. The search is conducted by a budget and thus they look for the maximization of the coverage with the budget constraint. More recently in [YB08], the query optimization problem adopts several cost attributes called QoWS parameters (*i.e.* latency, reliability, availability, price, reputation) given the nature of service-based environments.

  By adopting this approach some queries might not be feasible leading the user to modify his expectations or to wait for an adjustment of the market of resources.

Both approaches are profitable depending if the user wants the best possible query plan or the one that best fits his constraints. In [KSTI11] both approaches are integrated in search algorithms for finding the best deployment of processing operators in cloud-style computing resources that provide elasticity and have an associated price. They consider the constraints over time and price when one is preferred over the other, and when there is not an explicit constraint. Both scenarios are pertinent to deal with the multi-attribute costs and user preferences when leveraging of the service-based environment resources.

2.3.2 Response time and resource usage in mediation systems

The mediation systems in 90's [CGMH+94, CHS+95, LRO96, TAB+97, AK98] are the precedent of queries over data services such as web services[YB08]. Such systems assume a closed application domain feed by a collection of data sources (*e.g.* text files, HTML, XML, relational, search engines) providing data through a wrapper [PGMW95, RS97, KTV97]. Wrappers enable to deal with different data models and feed a mediation schema either in a global-as-view [4] or local-as-view [5] fashion. In such a way, the mediator provides an homogeneous view to facilitate querying to data consumers. It is in charge to locate data locations and split the query in sub-queries that are processed by the distributed resources, or the mediator. This leads to the optimization problem of deciding if it is convenient to privilege the resource usage, or the response time by adopting either the data-shipping, the query shipping or the hybrid shipping (*cf.* Section 2.1.3).

In mediation systems, there is certain homogeneity and control that makes traditional cost-based query optimization feasible. For instance, typically during the optimization process wrappers contribute to the cost estimation. It can be either by providing data statistics of the contained data (*e.g.* Disco [TRV95]), or participating during the optimization process (*e.g.* Garlic [ROH99]).

These assumptions are not possible in service-based environments because of the autonomy of service instances that may appear and disappear, and because of the absence of a single application domain shape. In HERMES [SAB+95], the assumption about the availability of data statistics is more realistic. They assume that the response time for a given wrapper invocation and the data cardinality is not provided by the wrapper[ACP96]. Therefore they use a histogram of wrapper invocations and perform aggregations over it as required by the cost function.

More dimensions are considered in mediation systems such as in MadiaGrid [CBB+04] where price, and data source quality is estimated through an Annotator[BCV08]; and there are considered user constraints such as threshold, budget, and data source preferences.

2.3.3 Web-based query optimization

In [YKB99, YKB03, YKvBO03], it is proposed a distributed query processing by means of web-service invocations. The optimization is based on QoS measures and user profiles. In particular, they consider the

4. global-as-view: The local schema is first and the description of the mediation schema is done in terms of local data[CGMH+94, TRV95, HKWY97].

5. local-as-view: The mediation schema is first and the description of local data is done in terms of the mediation schema[LRO96, Ull97, AK98].

cost attributes response time and price. Both attributes are computed considering the information about the network state, database servers load, and data fragmentation.

The users are classified in two levels of service according to their profiles. Each class of users guides the optimization towards either (1) the minimization of the resources usage or (2) the minimization of the response time. As pointed by [YKvBO03], it is important to handle the adaptation during the query execution given the dynamic nature of the environment but their experiments do not show such a claim as it seems to be considered only one-shot queries.

[SMWM06] presents a way to find an optimized query execution plan modeled as a pipelined web-service coordination. The data mappings are assumed following the First Schema Approach. The optimization is partially achieved considering the order of relational operators and the chunk size of transferred data between services. The computation of optimal plan time is performed by a greedy algorithm in a polynomial time.

In [BCD08] data services are modeled as search services and exact services (*i.e.* on-demand). It is about to combine both search and exact results given a datalog-style query while the rank provided by search services is kept. The query expression includes data retrievals, joins [BCCR08], and a constant k for bounding the tuples as they represent the k most interesting results. Services deliver data bounded by chunks (*i.e.* ordered pages of a fixed size) whose size is a parameter of the invocation. A query plan is represented by a service coordination that enables the pipeline processing of the results through the data-flow defined by the bindings between free and bound attributes. It is assumed that both search services and exact services provide the cardinality of data results and the selectivity of their attributes to compute, for example, the resulting size of a join. The optimization is conducted by one of the cost attributes (*e.g.* minimize the response time, minimize the total price).

We highlight the integration of search results and exact data. An interesting approach to deal with the cost estimation is the definition of chunks, also adopted by [SMWM06], and the k constant for estimating the intermediate data size and the evaluation cost. In particular for search services, it can be assumed that the most interesting results are on the top (*i.e.* earlier chunks). They also apply the chunk notion for exact services. Nonetheless, the occurrence of the attribute values to perform filtering, joining, or binding operators is not deterministic and it may lead to incomplete responses.

In the context of the Athena Distributed Processing Project (ADP) [TKK+09], input queries in the ADPLanguage are expressions of sequential compositions of arbitrary data processing operators (possibly) defined by users. The operators process data coming from data services. An operator is implemented by several service instances. The optimization is about to look for the best composition of service instances implementing the operators.

The query expression includes the specification of operator profiles describing the operators in the data-flow. An operator profile is provided by the user and it enables the generation of alternative logical plans through commutative and associative properties. The operator profile is decorated as long as the optimization process goes on. For example, the service instance implementing the required operator, and the resource where it is deployed. These decorations enable the estimation of the response time and the price [KSTI11].

ADP adopts either the possibilities-driven approach when there are no user constraints from the user, or the needs-driven approach to look for the dominant plan in presence of a user preference (*e.g.* time$\prec$price, price $\prec$time). Operators are possibly performed in an intra-operator parallelism [KSTI11] if they can process adjacent data fragments and if it is suitable for satisfying the user constraints.

In [SR11] web services providing data are accessible through a WSMEdiator service [SR07] that exports an integrated view of the available services. The WSMEdiator service allows to query web services in a SQL-like fashion. As web services have access patterns, they have to be arranged through a data-flow to reach the final result. The multiple sequential invocations of web services is time costly and thus an adaptive parallelization of web service invocation calls is proposed to reduce the response time.

The adaptive parallelization is done either manually by a *FF_APPLYP* operator or automatically by *AFF_APPLYP* [SR09] operator. Such operators split the web service invocations and gather the data from web services. Then the result is passed to another operator in charge to parallelize the invocations of the following web service(s) in the data-flow. In this way, parallelization is performed in multiple levels while pipeline execution is enabled. In particular, *AFF_APPLYP* adjusts automatically the parallelization by instantiating threads until an optimal performance is reached.

When the optimization of the response time is priority, it is interesting to adapt the parallelization of data service invocation. Clearly, it will increase the resource usage including, for instance, the energy consumption and price. In such a case, the same adaptation may adjust the parallelization as long as the query runs. It is also important to note that multiple parallel invocations to a single web service may reduce its throughput or even produce a denial of service in a multi-query environment. In such a case the adaptation should consider failures.

2.3.4 QoS-driven service coordination

A service coordination organizes autonomous software entities (*i.e.* services) providing data and computing capabilities to solve either data requirements or functional requirements. Services provide different levels of service characterized as QoS measures. Such QoS measures can be treated either as parameters of a cost function or as a cost itself resulting in any case in a multi-attribute cost.

[YKB07] addresses the problem of scheduling the tasks of a workflow into a set of services capable to execute them. Each service has different time and price costs and the users express a deadline and budget to be respected by the resulting scheduling(s). A series of Multi-Objective Evolutionary Algorithms (*i.e.* NSGAII, SPEA2 and PAES) are used to deal with the conflicting time and price attributes. They found the Pareto optimal in order to propose the (near)-optimal scheduling(s) subject to the time and price constraints. It is shown that these algorithms provide different accuracy and performance depending on the structure of the workflow.

[YB08] addresses the problem of finding web service coordinations considering functional requirements and quality constraints. It is assumed that a service exposes a service schema containing (1) which operations are implemented and (2) the graph of dependencies they have with other internal operations and with operations of other service schemes. Besides, a web service instance has a set of QoS maesures such as availability, price, and reliability. A requirement is expressed with an algebra that allows to de-

fine functional requirements and quality constraints over the service operations. The evaluator generates sequential Service Access Plans by merging the graphs of dependencies of the required operations and by aggregating their QoS measures to look for those compositions that hold the quality constraints. The enumeration of the SAPs is done by a dynamic programming based algorithm with a score function that estimates the resulting quality of the alternative plans. The algorithm computes in a bottom-up fashion the sub-plans with the best performance. Then, following a greedy heuristic, these sub-plans are filtered to get those whose score dominates the others and the resulting ones are passed to the next iteration to compose them. Finally a (sub-)optimal plan is computed.

2.3.5 Discussion

The works presented in previous sections tackle optimization of queries and service coordinations in service-based environments considering some of the nowadays QoS requirements. Table 2.2 shows the partial list of works tackling some of the dimensions of hybrid query optimization (*cf.* Table 2.1).

	First class objects	Cost attributes	Optimization approach	Adaptation	Service selection
[YKB99, YKB03, YKvBO03]	data	time, price	profile driven	no	join site
[YKB07]	service operations	time, price	needs driven	no	yes
[SMWM06]	data	time	possibilities driven	no	no
[YB08]	service operations	time, price, availability, reliability, reputation	needs driven	no	yes
[BCD08]	data (exact,search)	time, price	possibilities driven	no	access pattern
[SR11]	data	time	possibilities driven	number of local parallel threads	no
[TKK+09, KSTI11],	data	time, price	possibilities driven, needs driven	no	yes
[TGT13],	data	time, price	needs driven	no	yes

Table 2.2 – Related works

These dimensions are tackled with strong assumptions about the available information for performing the optimization, *i.e.* data dependencies, data statistics. Some of them are oriented to optimize service coordinations solving functional requirements, and others to solve data requirements. In service-based environments both data and service operations are opaque and we can abstract them as data providers. Hybrid query optimization allows to obtain service coordinations whose services may provide data or computing capabilities. Such service coordinations are optimal w.r.t. the optimization objective of an SLA contract that specifies a combination of QoS aspects represented by cost attributes. Hence, the resulting coordinations are oriented to satisfy the user needs, *i.e.* needs-driven approach. Nevertheless, as the SLA contract defines the ideal point to meet, the best coordination for a possibilities-driven approach can be defined as the closest to the origin.

Both, the adaptation and service selection are problems themselves that we left aside in this work and we recognize them as problems to tackle in future investigations.

Next, we precise our discussion w.r.t. the optimization of multi-attribute costs representing QoS aspects, and the extensibility of the optimizers.

Optimization in presence of multi-attribute costs. The works optimizing the performance and price do not consider the wide combinations of user expectations such as non-functional aspects and data quality. In service-based environments, the expectations of users and applications (*i.e.* SLA) should dictate the cost attributes to be considered during the optimization instead to fix them into the search strategy.

With this perspective, it is required an abstraction layer between the cost estimation and the solution space selection. In any case, the cost estimation remains a problem as it is required to model each of the multiple cost functions with the heterogeneous (or absent) information.

In such cost functions, data-related parameters cannot be assumed like in cost-based query optimization. Data statistics are hard to harvest in service-based environments because of the intensive monitoring and storage required to maintain up-to-date information. For instance, the nested structure of data (*e.g.* XML, JSON) makes prohibitive to materialize accurate selectivities of leaf data attributes.

The must of the works rely on the assumption that data statistics are either provided by the service instance or by a monitoring service that keeps an up-to-date histogram feed by query sampling or by storing previous invocation results. These statistics improve the accuracy of cost estimation if they are effectively available.

In the absence of data statistics, the adoption of operator motion heuristics can help to reduce data size and improve performance. The algorithms adopting such techniques are all driven to optimize the data-flow. Nevertheless, the wide possibilities of optimization interests require a wider search space that should consider not only the data-flow but the control-flow for organizing service invocations.

As the user might require either needs-driven or possibilities-driven optimization, the optimizer should be capable to adopt both in a single search strategy. We define the ideal point (*i.e.* SLA) and the optimizer looks for the closest solution(s). In the particular case of possibilities-driven approach, the optimizer can define the origin as the ideal point.

Extensible framework for optimizing service coordinations. In general, works optimizing queries or service coordinations in presence of multiple cost attributes, put the search space characteristics and cost functions inside the search strategy. Such a framework is not extensible when new data models and operators arise (*e.g.* big data processing [Ewe12]), and as new costs have to be considered (*e.g.* resource pricing [LSW+12], data pricing [KUB+12], energy [KB12]). The QoS requirements demand extension of the optimizer that is difficult to achieve in this framework as it is data-model dependent (*e.g.* System-R [Bat86]).

For dealing with QoS requirements, there is the extensible framework [RH86] that enables the separation of aspects and thus the easy extension of its parts, *i.e.* alternative plan enumeration, cost estimation, search strategy. This framework has been adopted in academic database systems such as EXODUS/ VOLCANO/ CASCADES[GD87, GM93, Gra95], STARBURST[HFLP89, PHH92], GENESIS [Bat86, Bat87]; and commercial database systems [WH09] such as SQL Server, and DB2. Recently it has been also adopted by new data processing paradigms such as in SCOPE[Jin12].

In the extensible framework, each optimization aspect is tackled individually. For instance, in [RR82] (and further in the STARBURST project [PHH92]), the optimizer is divided in two modules. The first is the join enumerator that looks for the join orders, and the second is the plan generator that completes the query plans and computes their associated costs. This separation also enables a configurable optimizer according to the characteristics of queries, data statistics, and the application, *e.g.* OLAP, OLTP. For instance, the join enumerator can be configured to generate only left-linear shapes and the plan generator to select sort-based operator methods.

The adoption of extensible framework is not exempt to provide efficiency to achieve what a good optimizer is expected to be. Therefore, extensible optimizers first provide generality to the optimization process and then efficient techniques are adopted in a dynamic function fashion. In this work, we adopt the extensible query optimizer hypothesis '*if the query optimizer is organized as a rule-based system, as new types, operators, and methods are added, the system can be extended by defining their properties via new rules*'[GD87]. We tackle separately, the query rewriting, search space generation, cost estimation, and solution space selection. In this context we privilege the separation of concerns and therefore the extensibility required in service-based environments.

2.4 Conclusions

This chapter reviewed the issues concerning the query optimization such as the search space enumeration, cost estimation, search strategy and the trade-off between response time and resource usage. These issues remain important for the new paradigm of evaluating queries with the service coordination approach.

We also made a partial revision of the state-of-the-art of systems tackling the optimization in presence of QoS aspects. This analysis leads us to identify that the most of recent systems still tackling the optimization problem with tight search strategies that make difficult the extension of the optimizer. Because everything in a service-based environment is subject to change, we can make few assumptions for a suitable optimization. It can be either the availability of service instances, the levels of service, the data quality, offer and demand, or the user requirements. We believe the extensibility is a key requirement for supporting the nowadays context where new data requirements, data models, cost attributes, and QoS requirements still appearing. A query optimizer should be ready to adjust its optimization strategy in presence of these various conditions.

CHAPTER 3

The HyQoZ approach

This chapter introduces the HyQoZ approach for optimizing hybrid queries with SLA contracts.

The remainder of the chapter is organized as follows. Section 3.1 presents hybrid queries over on-demand and stream data services. Section 3.2 describes the intermediate representation of hybrid queries by means of data transformation functions (*dt-functions*) and the derivation process for obtaining them from a hybrid query expression. Section 3.3 introduces the SLA contract notion and the query workflow cost to be considered in hybrid query optimization. Section 3.4 describes the optimization process we propose. Finally, Section 3.5 concludes this chapter.

3.1 Hybrid query

We adopt the notion of hybrid queries [CvVSCB09] for characterizing queries on service-based environments. A hybrid query expresses a data requirement over on-demand and stream services accessible through standardized interfaces, *e.g.* APIs. Data and services are represented as complex value types that enable the expression of hybrid queries by composing complex value operators. The formal definitions of complex value types, and complex value operators can be found in [CvVSCB09].

3.1.1 Data and service types

This section introduces the types representation of data and services for expressing hybrid queries over data services. Originally, this representation was defined in [CV11] and we adopt it for this work.

3.1.1.1 Complex value types and services

We assume a finite set of domains D of strings $\mathbb{S}$, booleans $\mathbb{B}$, integers $\mathbb{Z}$, reals $\mathbb{R}$, and timestamps $\mathbb{T} = \mathbb{Z}_0$. These domains denoted by their names $string$, $boolean$, $integer$, $real$, and $timestamp$ members of the set of names $\mathbb{A} \subset \mathbb{S}$.

Types are represented by lower-case letters with hats (*e.g.* $\hat{t}$) and are defined by a pair $A : def$, where $A \in \mathbb{S}$ is the name of the type and def its definition. The functions $name$ and def enable the access to the name and definition of a type $\hat{t}$. For instance, for the type $\hat{t} = nickname : string$, $name(\hat{t}) = nickname$ and $def(\hat{t}) = string$.

Complex value types enable to represent (1) tuples of the form $\hat{t} = A : \langle \hat{t}_1, ..., \hat{t}_n \rangle$ where $\hat{t}_i$ represents a type, (2) set types of the form $\hat{t} = A : \{\hat{t}_1\}$ where $\hat{t}_1$ represents the unique type, (3) and function types

of the form $\hat{f} = A : \hat{t}_1 \times ... \times \hat{t}_n \rightarrow \hat{t}_o$ where $\hat{t}_i$ represents an input parameter type and $\hat{t}_o$ represents a single output type.

For instance, the friendships of a person can be modeled by the tuple type $\hat{t} = friendships$: $\langle nickname : string, with : \{friend : string\}\rangle$ where the type $nickname : string$ identifies the person and the set type $with : \{friend : string\}$ represents the set of friends of such a person. The function type allows to represent service operations further defined.

A type $\hat{t}$ denotes a set of complex value instances $[\![\hat{t}]\!]$. For instance, the denotation of the type $\hat{t} = friendship : \langle nickname : string, with : \{friend : string\}\rangle$ is given by

$$
\begin{aligned}
[\![\hat{t}]\!] = \{ & friendships : \langle nickname : \text{'Alice'}, with : \{friend : \text{'Bob'}, friend : \text{'Claire'}, ..., friend : \text{'Zara'}\}\rangle, \\
& friendships : \langle nickname : \text{'Bob'}, with : \{friend : \text{'Alice'}, friend : \text{'David'}, ..., friend : \text{'Zoe'}\}\rangle, \\
& ... \\
& friendships : \langle nickname : \text{'Zuzana'}, with : \{friend : \text{'Bob'}, friend : \text{'David'}, ..., friend : \text{'Will'}\}\rangle\}.
\end{aligned}
$$

Analogously to complex value types, the functions $name$ and val obtain the components of complex value instances. For example, if $t \in [\![\hat{t}]\!]$ is a complex value instance of the form $primes : \{num : 2, num : 3, num : 5...\}$, then $name(t) = primes$ and $val(t) = \{num : 2, num : 3, num : 5...\}$.

In particular, for tuple values t of the form $A : \langle A_1 : v_1, ..., A_n : v_n\rangle$, the dot notation $t.A_i$ allows to access the values v_i of the attributes $A_i : v_i$ of t. For example, if t denotes $friendships : \langle nickname :$ 'Alice', $with : \{friend :$ 'Bob', $friend :$ 'Claire', ..., $friend :$ 'Zara'$\}\rangle$, then $t.nickname =$ 'Alice' and $t.with =$ $\{friend :$ 'Bob', $friend :$ 'Claire', ..., $friend :$ 'Zara'$\}$.

By extension, the function $def(A : v)$ allows the access to the definition of the complex value type. This is, for the type $\hat{t} = friendships : \langle nickname : string, with : \{friend : string\}\rangle$ and the type instance $t \in [\![\hat{t}]\!]$, $def(t) = \langle nickname : string, with : \{friend : string\}\rangle$.

3.1.1.2 Service

A service is an autonomous software entity accessible through a standardized interface whose constituent operations return data as result of an invocation. Service interfaces are represented by a tuple type $serv\_name : \langle \hat{f}_1, ..., \hat{f}_n\rangle$, where each $\hat{f}_i$ is a function type of the form $op\_name_i : f_i^{def}$ representing an operation.

The functions $registry$ and $choice$ enable the access to a service directory. For a given set of types S specifying service interfaces, and a service interface $s \in S$ the function $registry$ provides the set of service instances implementing s, *i.e.* $registry(s) \subseteq [\![s]\!]$. The function $choice$ selects a specific service instance from the set of instances obtained from the registry, *i.e.* $choice(registry(s)) \in registry(s)$.

The service notion is specialized for representing on-demand and stream data services.

3.1.1.3 On-demand data service

An on-demand data service provides data in a request-response fashion through synchronous data operations. An on-demand data operation is represented by a function type $\hat{f}_i$ of the form op_name_i :

$\hat{i}_1 \times ... \times \hat{i}_m \rightarrow \{\hat{t}_o\}$, where $\hat{i}_j$ is an input type and the output type $\hat{t}_o = A : \langle \hat{i}_1, ..., \hat{i}_m, \hat{o}_{m+1}, ..., \hat{o}_{m+p} \rangle$ contains attributes of the same type as the input parameters $\hat{i}_j$, and additional attributes $\hat{o}_k$ representing the result of the on-demand data operation. The complex value provided by the invocation of op_name_i contains the values that correspond with every input attribute name A_i such that $T = op\_name_i(A_1 : v_1, ..., A_m : v_m) = \{A : \langle A_1 : v_1, ..., A_m : v_m, A_{m+1} : v_{m+1}, ..., A_{m+p} : v_{m+p} \rangle\}$.

For instance, the interface of the on-demand service $friends$ is defined by $friends : \langle \hat{f}_1, \hat{f}_2 \rangle$ where $\hat{f}_1 = profile \ : \ nickname : string \rightarrow \{profile : \langle nickname : string, gender : string, email : string, age : integer \rangle\}$ and $\hat{f}_2 = friendsof \ : \ nickname : string \rightarrow \{friendships : \langle nickname : string, with : \{friend : string\} \rangle\}$.

For obtaining the friends of Alice, we perform the invocation $friends.friendsof(nickname$: 'Alice') that produces the output set $\{friendships : \langle nickname :$ 'Alice', $with : \{friend :$ 'Bob', $friend :$ 'Claire', ..., $friend :$ 'Zara'$\}\rangle\}$.

3.1.1.4 Stream data service

A stream data service provides data continuously to a subscribed consumer. In our model, we abstract the subscription mechanism and define the interface of a stream data service as a tuple of stream data operations. A stream data operation is represented by a function type $\hat{f}_i$ of the form $op_i \ : \ \hat{t} \rightarrow \{A : \langle \hat{o}_1, \ldots, \hat{o}_n, \hat{t} \rangle\}$ where each $\hat{o}_i$ represents an attribute and $\hat{t}$ represents a timestamp attribute of the form $\tau : timestamp$. The function thus produces a set of timestamped tuples at a given time value.

For instance, the interface of the stream service $whereRU$ is defined by $whereRU : \langle \hat{f}_1 \rangle$ where $\hat{f}_1 = location \ : \ nickname : string \rightarrow \{location : \langle nickname : string, lat : real, lon : real, timestamp : iteger \rangle\}$. The $whereRU$ operation allows to subscribe to the location of a person at a given time.

The subscription may contain windowing directives such as *give me the last five tuples*, or *give me the tuples from the last hour*. For instance, the last five locations of Bob are given by $whereRU.location(nickname :$ 'Bob', 'TOP 5') that produces

$\{ location : \langle nickname :$ 'Bob', $lat : 48.85214927$, $lon : 2.36789545$, $timestamp : 1406564492 \rangle$,
$location : \langle nickname :$ 'Bob', $lat : 48.8520214$, $lon : 2.36771888$, $timestamp : 1406564403 \rangle$,
$location : \langle nickname :$ 'Bob', $lat : 48.85197632$, $lon : 2.36759633$, $timestamp : 1406564311 \rangle$,
$location : \langle nickname :$ 'Bob', $lat : 48.85189498$, $lon : 2.36750442$, $timestamp : 1406564294 \rangle$,
$location : \langle nickname :$ 'Bob', $lat : 48.85174922$, $lon : 2.3673861$, $timestamp : 1406564240 \rangle\}$.

3.1.2 Hybrid query expressions

A hybrid query is represented by a stream-complex-value expression $\mathcal{SC}$-$\mathcal{O}$ that specifies the composition of complex value operators. Complex value operators are inspired from relational operators that also apply to the complex values produced by data services. The following grammar specifies how $\mathcal{SC}$-$\mathcal{O}$ expressions E are specified over on-demand and stream data service operations represented as o and s respectively.

D	$::=$	$o(v_1, ..., v_n)$ $\mid$ s		on-demand and stream operations
E	$::=$	D		single operation query expression
		$\mid$ $\sigma_{exp}(E)$		selection under expression σ_{exp}
		$\mid$ $\pi_{exp}(E)$		projection under expression π_{exp}
		$\mid$ $\rho_{exp}(E)$		renaming under expression ρ_{exp}
		$\mid$ $E \overset{\rightarrow}{\bowtie} E$		bind-join on on-demand operation o
		$\mid$ $E \bowtie_{\sigma} E$		theta-join under expression σ

The complex value operators can be applied to one or two arbitrary sub-expressions depending on whether the operator is unary or binary. Next we describe the complex value operators we consider.

3.1.2.1 Complex value operators

We consider the complex value operators selection, projection, and renaming, and the combination operators theta-join and bind-join.

— **Selection** $\sigma_{exp}(S)$ filters the complex value instances in a set S based on a selection expression exp, producing as output the subset of instances that satisfy the expression.

— **Projection** $\pi_{exp}(S)$ enables to retrieve certain attributes in a complex value instance. Such attributes may be nested and multivalued. The data elements to retrieve are specified in a projection expression exp, which is applied to each element s of the set complex value instance S.

— **Renaming** $\rho_{exp}(S)$ enables to rename either data services or attributes for manipulating multiple instances of them in a single expression. The element to rename and its new name is specified in a renaming expression exp, which is applied to each element s of the set complex value instance S.

— **Theta-join** $R \bowtie_{\sigma} S$ enables to combine the tuple values of two input sets R and S based on a selection expression denoted by σ [1]. The result is a subset of a Cartesian product between R and S such that each tuple holds the expression σ.

— **Bind-join** $R \overset{\rightarrow}{\bowtie} s$ enables to bound the tuple values of an output set in R with the input attributes of a data service operation that produces s.

In this work, we consider conjunctive Select-Project-Join queries and hence some complex value operators are not considered with respect to the original hybrid query specification in [CV11], *i.e.* windowing, Cartesian product, group, ungroup, set operators. In particular, we are not considering explicitly windowing operators as we assume they are performed by the data service instance in accordance with a subscription, *e.g. give me the last tuple value*, *give me the tuples within the last five minutes*. The Cartesian product is not considered because we privilege the reduction of intermediate results over the expressiveness of hybrid queries which in practice may dispense with it in the context of service-based environments.

1. The 'theta' historically stands for a selection condition θ [GMUW02] that we represent by σ in order to avoid the ambiguity with the comparison operators θ

3.1.2.2 Examples of hybrid queries

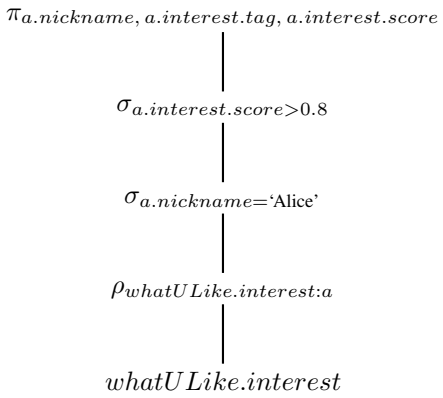
Consider the data service operations in Table 3.1. Suppose Alice and Bob share data though their smartphones via applications that access data services through their interfaces. For example, they share once their profile data such as email, gender, age and nickname via the $friends.profile$ data service operation and continuously their current location via the $whereRU.location$ operation. There is also a $friends.friendsOf$ service operation that stores the friendships of people, for instance the friends of Alice and Bob, and eventually their common friends. As they share things and manifest their interests, the data service $whatUlike.interests$ stores such an information and make it available.

Name	Operation signature	Description
Profile	$friends.profile(nickname:string) \rightarrow$ $\{profile:\langle nickname:string,$ $gender:string,$ $email:string,$ $age:integer\rangle\}$	The Profile service provides the profile of a person identified by a $nickname$. The $profile$ type contains personal information like $gender$, $email$, and age.
Friendship	$friends.friendsof(nickname:string) \rightarrow$ $\{friendship:\langle nickname:string,$ $with:\{friend{:}string\}\rangle\}$	The Friendship service provides the relationships of a user with other users both identified by the attributes $nickname$ and $friend$ respectively.
Interests	$whatUlike.interests(nickname:string) \rightarrow$ $\{interest:\langle nickname:string,$ $tag!:string,$ $score!:real\rangle\}$	The Interests service provides what and how much a user likes via the tag and $score$ attributes.
Location	$whereRU.location(nickname:string) \rightarrow$ $\{location:\langle lat{:}real,$ $lon:real,$ $timestamp:integer\rangle\}$	The Location service provides for a given user $nickname$ the location coordinates (*i.e.* lat, lon) in a given time (*i.e.* $timestamp$).

Table 3.1 – Data service operations

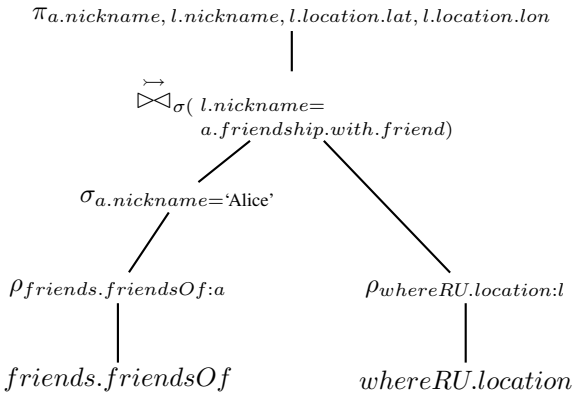
The following examples present hybrid queries over these service operations. We use a tree representation to show the complex value operator composition. Leafs represent data services. The intermediate and root nodes represent data operators. In particular, the root node denotes the result of the hybrid query. The operator parameters involve nested attribute types we access via the dot operator. The edges represent the operator composition. Additionally, in the case of bind-join operator, we expose the selection expression that bounds the input attributes of a service with the values of output attributes of another.

Example 3.1.1 Consider the hybrid query *Which are the interests of Alice with a score above* 0.8*?* that requires data from the on-demand service $whatULike.interest$. The $\mathcal{SC}$-$\mathcal{O}$ operator expression is as follows.



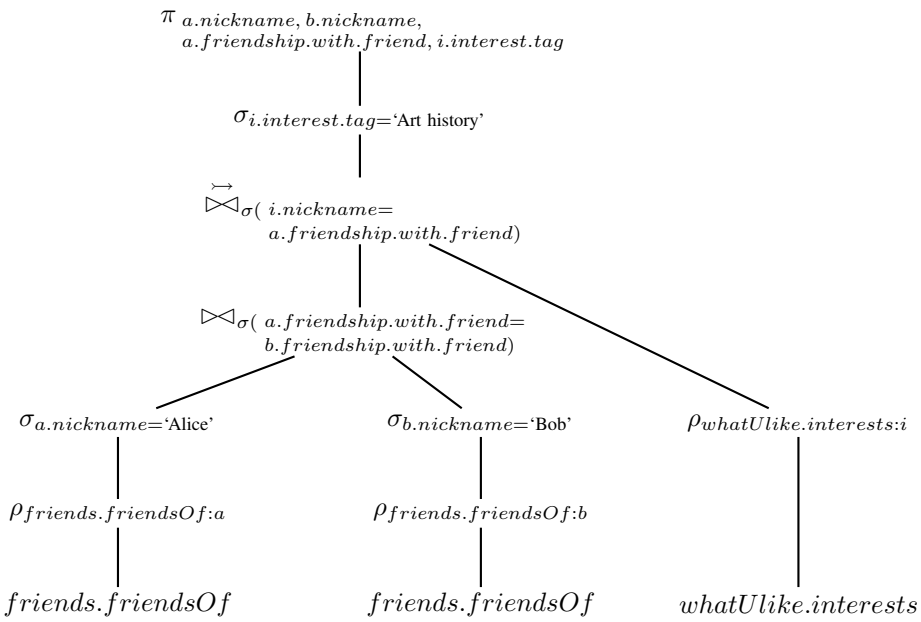
This $\mathcal{SC}$-$\mathcal{O}$ expresses the requirement to retrieve the interests of Alice from $whatULike.interests$ by bounding the $nickname$ attribute with the value 'Alice', and whose retrieved tuple set is renamed as a. The tuples are filtered by the σ operator that constraints the tuple values only to those whose interest score is above 0.8. The π operator projects the nickname of Alice, her interests and the score for each.

Example 3.1.2 Consider the hybrid query *Where are the friends of Alice ?* that combines data from the on-demand service $friends.friendsOf$ and a stream service $whereRU.location$. This combination is done using the output tuple values of the on-demand service $friends.friendsOf$ as input of the stream service $whereRU.location$ via the bind-join operator.



This $\mathcal{SC}$-$\mathcal{O}$ expresses the requirement to retrieve the friends of Alice from the $friends.friendOf$ on-demand service by providing the constant value 'Alice' to the input attribute $nickname$. The retrieved set of tuples is renamed to a and the tuples from $whereRU.location$ are renamed to l. The bind-join operator $\overset{\rightarrowtail}{\bowtie}$ expresses that the value of $a.nickname$ is passed to the input attribute $l.nickname$ for invoking $whereRU.location$. The π operator projects the nickname of Alice and of their friends along with their current location coordinates.

Example 3.1.3 Consider the hybrid query *Which of the common friends of Alice and Bob are interested in Art history ?* that combines data from the on-demand service $friends.friendsOf$ and from the on-demand service $whatULike.interests$. The $\mathcal{SC}$-$\mathcal{O}$ operator expression is as follows.



This $\mathcal{SC}$-$\mathcal{O}$ expresses the requirement to retrieve and combine the friends of Alice and Bob from the $friends.friendsOf$ service by providing their corresponding nickname values. Both sets of tuples are renamed to a and b respectively. There is a third on-demand service $whatULike.interests$ whose tuple values are retrieved via the $\overset{\rightarrowtail}{\bowtie}$ operator by passing the nickname of the Alice's friends to the input attribute $i.nickname$. The tuples are filtered to get those friends interested in Art history. The π operator projects the nicknames of Alice, Bob and their common friends along with their interests.

3.2 Intermediate representation / data transformation functions

The notion of *data transformation function*, or *dt-function*, is introduced to have an intermediate representation of the complex value operators of a $\mathcal{SC}$-$\mathcal{O}$ expression representing a hybrid query. Like complex value operators, *dt-functions* can be composed for producing the final result of the hybrid query. Besides, the *dt-functions*' signatures enable the analysis of the data dependencies among them towards the generation of alternative service coordinations implementing the hybrid query, *i.e.* query workflows.

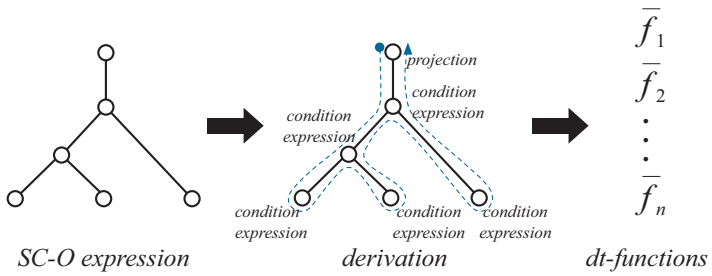


Figure 3.1 – Derivation of *dt-functions*

The complex value types, condition expressions, and projections of the $\mathcal{SC}$-$\mathcal{O}$ expression are analyzed by a derivation process. The derivation produces a set of *dt-functions* as depicted in Figure 3.1. Next we give the definitions and properties of *dt-functions*. Then, we present the rules for deriving the *dt-functions* from a given $\mathcal{SC}$-$\mathcal{O}$ expression.

3.2.1 *dt-function* definitions

A data transformation function (*dt-function*) is denoted by a lower-case symbol with a bar, *e.g.* $\bar{f}$. It represents a complex value operator and its signature describes the data transformation the operator performs over a series of complex value types. In order to represent the attributes of a type $\hat{t}$, we adopt the function $atts$ that gives the set of access paths (in dot notation) of the constituent attributes $\hat{t}$. Besides, the functions $iatts$ and $oatts$ provide separately the set of access paths of input and output attributes such that, $atts(\hat{t}) = iatts(\hat{t}) \cup oatts(\hat{t})$.

Definition 3.2.1 [*dt-function*] A *dt-function* $\bar{f} : \{\hat{t}_1\} \ldots \{\hat{t}_m\} \rightarrow \{\hat{t}'_1\} \ldots \{\hat{t}'_m\}$ (1) reads the sets of tuple values of m complex types, (2) restricts the tuples whose values hold a set of condition expressions, and (3) projects the attributes stated by a projection expression producing (possibly new) m complex types.

Signature: $\bar{f}(\mathcal{A}, \mathcal{E}, \mathcal{P})$ where

- $\mathcal{A}$: is a set of type names where each name $a_i \in \mathcal{A}$ has a definition $def(a_i) = def(\hat{t}_i)$ before the data transformation and $def(a_i) = def(\hat{t}'_i)$ after the data transformation;
- $\mathcal{E}$: is a set of conjunctive condition expressions over the attributes in $atts(a_i)$ for $a_i \in \mathcal{A}$
- $\mathcal{P}$: is a set of attributes to be projected where $\forall p \in \mathcal{P}$, $\exists!\, i \in [1 \,..\, n]$ such that $p \in atts(\hat{t}'_i) \subseteq atts(\hat{t}_i)$.

Property 3.2.1 [*dt-function* equivalence] The *dt-functions* $\bar{f}_1(\mathcal{A}_1, \mathcal{E}_1, \mathcal{P}_1)$ and $\bar{f}_2(\mathcal{A}_2, \mathcal{E}_2, \mathcal{P}_2)$ are said to be equivalent if each of their signature parameters are equal, *i.e.* $\mathcal{A}_1 = \mathcal{A}_2$, $\mathcal{E}_1 = \mathcal{E}_2$, and $\mathcal{P}_1 = \mathcal{P}_2$.

Property 3.2.2 [*dt-functions* composition] the *dt-function* $\bar{f}_r : \{\hat{t}_1\}...\{\hat{t}_p\} \rightarrow \{\hat{t}'_1\}...\{\hat{t}'_p\}$ represents the composition of two *dt-functions* $\bar{f}_1 : \{\hat{t}_1\}...\{\hat{t}_m\} \rightarrow \{\hat{t}'_1\}...\{\hat{t}'_m\}$ and $\bar{f}_2 : \{\hat{t}_1\}...\{\hat{t}_n\} \rightarrow \{\hat{t}'_1\}...\{\hat{t}'_n\}$ if the next two conditions hold:

1. **Minimal types.** The types read and produced by both $\bar{f}_1$ and $\bar{f}_2$, are read and produced by $\bar{f}_r$ such that the produced types have no more and no less constituent attributes as defined by the most restrictive projection expression of both $\bar{f}_1$ and $\bar{f}_2$. This is,
   - $m \leq p$, $n \leq p$;
   - $\forall\, i \in [1 \,..\, m], \exists!\, j \in [1 \,..\, p] \mid atts(\hat{t}_i) \subseteq atts(\hat{t}_j),\ atts(\hat{t}'_i) \supseteq atts(\hat{t}'_j)$;
   - $\forall\, i \in [1 \,..\, n], \exists!\, j \in [1 \,..\, p] \mid atts(\hat{t}_i) \subseteq atts(\hat{t}_j),\ atts(\hat{t}'_i) \supseteq atts(\hat{t}'_j)$.

2. **Tuple values subsumption.** The output tuple values of $\bar{f}_r$ hold the condition expressions of both $\bar{f}_1$ and $\bar{f}_2$ such that, $\forall j \in [1 \,..\, p]$,

   — if $term_1 \theta term_2 \in \mathcal{E}_1$ where $term_1 \in atts(\hat{t}'_j)$ or $term_2 \in atts(\hat{t}'_j)$, then $\nexists\, t \in [\![\hat{t}'_j]\!] \mid t \not\models term_1 \theta term_2$;

   — if $term_1 \theta term_2 \in \mathcal{E}_2$ where $term_1 \in atts(\hat{t}'_j)$ or $term_2 \in atts(\hat{t}'_j)$, then $\nexists\, t \in [\![\hat{t}'_j]\!] \mid t \not\models term_1 \theta term_2$.

According with this property, the composition of two *dt-functions* is given by the $merge$ function defined as follows.

Definition 3.2.2 [Merge function] For two given *dt-functions* $\bar{f}_1(\mathcal{A}_1, \mathcal{E}_1, \mathcal{P}_1)$ and $\bar{f}_2(\mathcal{A}_2, \mathcal{E}_2, \mathcal{P}_2)$, the function $merge$ gives the composition of $\bar{f}_1$ and $\bar{f}_2$ such that $merge(\bar{f}_1, \bar{f}_2) = \bar{f}_r(\mathcal{A}_r, \mathcal{E}_r, \mathcal{P}_r)$ where

$$\begin{aligned}
\mathcal{A}_r &= \mathcal{A}_1 \cup \mathcal{A}_2 \\
\mathcal{E}_r &= \mathcal{E}_1 \cup \mathcal{E}_2 \\
\mathcal{P}_r &= \{name.p \in \mathcal{P}_1 \cap \mathcal{P}_2 \mid name \in \mathcal{A}_1 \cap \mathcal{A}_2\} \;\cup\; \{name.p \in \mathcal{P}_1 \cup \mathcal{P}_2 \mid name \in \mathcal{A}_1 \triangle \mathcal{A}_2\}
\end{aligned}$$

Proof 3.2.1 Considering the two conditions of Property 3.2.2, the $merge$ function holds:

— The **minimal types** condition since the set of type names is given by $\mathcal{A}_r = \mathcal{A}_1 \cup \mathcal{A}_2$ and then the types read by both $\bar{f}_1$ and $\bar{f}_2$ are all read by $\bar{f}_r$. Besides, the projection $\mathcal{P}_r$ is the union of both (1) the attributes of shared types projected by both $\bar{f}_1$ and $\bar{f}_2$, and (2) the attributes of non-shared types projected by $\bar{f}_1$ or $\bar{f}_2$. Then, the resulting types contain the minimal types.

— The **tuple values subsumption** since the condition expressions of both $\mathcal{E}_1$ and $\mathcal{E}_2$ are conjunctive (*cf.* Def. 3.2.1), therefore $\mathcal{E}_r = \mathcal{E}_1 \cup \mathcal{E}_2$ ensures that all the produced tuples satisfy the conditions of both $\bar{f}_1$ and $\bar{f}_2$.

■

The *dt-function* definition is generic and allows to represent complex value operators and compositions of complex value operators. Next we present the derivation of *dt-functions* from a hybrid query expression.

3.2.2 Derivation of *dt-functions*

Given a hybrid query expression E, the *dt-functions* are derived by analyzing E via the function $der : E \rightarrow \bar{F}$ where $\bar{F}$ denotes the set of derived *dt-functions*. It is assumed that the E expression is safe, *i.e.* every input attribute of the involved data services is bounded either to a constant or to an output attribute of another data service.

For defining der we first identify four conditions expressions. Consider two service operations described by the types $\hat{t}_1$ and $\hat{t}_2$.

— *R-exp* is an expression of the form $T\,\theta\,constant$ where $T \in iatts(\hat{t}_1)$. This expression denotes the binding operation of the $constant$ to the input attribute represented by T for retrieving the data produced by $\hat{t}_1$.

— *F-exp* is an expression of the form $T\,\theta\,constant$ where $T \in oatts(\hat{t}_1)$. This expression denotes the filtering of tuple values in $[\![\hat{t}_1]\!]$ hold the expression $T\,\theta\,constant$.

— *C-exp* is an expression of the form $T_1\,\theta\,T_2$ where $T_1 \in oatts(\hat{t}_1)$ and $T_2 \in oatts(\hat{t}_2)$. This expression denotes the correlation of the tuple values $[\![\hat{t}_1]\!]$ and $[\![\hat{t}_2]\!]$ w.r.t. their corresponding output attributes T_1 and T_2.

— *B-exp* is an expression of the form $T_1\,\theta\,T_2$ where $T_1 \in iatts(\hat{t}_1)$ and $T_2 \in oatts(\hat{t}_2)$. This expression denotes the binding of the output attribute T_2 to the input attribute T_1 for retrieving the tuples values $[\![\hat{t}_1]\!]$.

Definition 3.2.3 [derivation of *dt-functions*] For a given $\mathcal{SC}$-$\mathcal{O}$ expression E the derivation of the set of *dt-functions* $der(E) = \bar{F}$ is given by the following rules:

1. If $E = \rho(serv.op : name)(serv.op)$, then $der(E) = \{\}$ and $def(serv.op) = def(name)$ is true.
2. If $E = \pi(exp)(E')$ where exp is a set of attributes to be projected [2], then

$$der(E) = \begin{cases} \{\bar{f}(\mathcal{A}, \mathcal{E}, \mathcal{P})\} \cup \bar{F} \text{ where} \\ \quad \mathcal{A} = \{name\} \text{ such that } \exists\, name.T \in exp \\ \quad \mathcal{E} = \{\} \\ \quad \mathcal{P} = \{name.T \mid name.T \in exp\} \\ \quad \bar{F} = \begin{cases} der(\pi(exp \setminus \mathcal{P})(E')); & \text{if } exp \setminus \mathcal{P} \neq \{\} \\ der(E'); & \text{otherwise} \end{cases} \end{cases}$$

3. If $E = \sigma(exp)(E')$ and exp is a ***R-exp*** or a ***F-exp*** condition expression

$$der(E) = \begin{cases} \{\bar{f}(\mathcal{A}, \mathcal{E}, \mathcal{P})\} \cup \bar{F} \text{ where} \\ \quad \mathcal{A} = \{name\} \text{ such that } exp{=}name.T\,\theta\,constant \\ \quad \mathcal{E} = \{exp\} \\ \quad \mathcal{P} = atts(name) \\ \quad \bar{F} = der(E') \end{cases}$$

4. If $E = E'_1 C_{exp} E'_2$ where C is a combination operator and exp is a ***B-exp*** or a ***C-exp*** condition expression

2. For the projection expression exp, we assume there are neither duplicate attributes nor a mandatory order among them. Then, we treat exp as a set.

$$der(E) = \begin{cases} \{\bar{f}(\mathcal{A}, \mathcal{E}, \mathcal{P})\} \cup \bar{F}_1 \cup \bar{F}_2 \text{ where} \\ \quad \mathcal{A} = \{name_1, name_2\} \text{ such that } exp{=}name_1.T_1\, \theta\, name_2.T_2 \\ \quad \mathcal{E} = \{exp\} \\ \quad \mathcal{P} = atts(name_1) \cup atts(name_2) \\ \quad \bar{F}_1 = der(E'_1) \\ \quad \bar{F}_2 = der(E'_2) \end{cases}$$

Example 3.2.1 The following example traces the derivation of *dt-functions* $der(E) = \bar{F}$. Consider the expression E (see Figure 3.2) of the hybrid query *Where are the friends of Alice ?* in Example 3.1.2. The trace is done in a depth-first traversal of E and we obviate the backtracking for obtaining the final set $\bar{F}$. In order to facilitate the reading of *dt-functions*, we identify each one with a unique symbol name that suggests what the *dt-function* does.

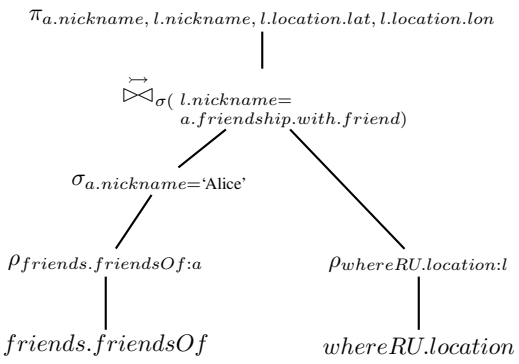
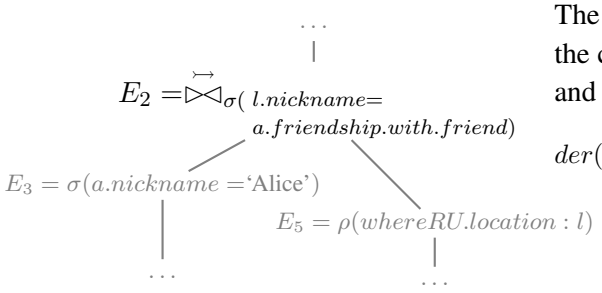


Figure 3.2 – *Where are the friends of Alice ?*

$$E_1 = \pi(\ a.nickname,\ l.nickname,\ l.location.lat,\ l.location.lon)$$
$$E_2 = \overset{\rightarrowtail}{\bowtie}_{\sigma(\ l.nickname= a.friendship.with.friend)}$$
$$\cdots \qquad \cdots$$

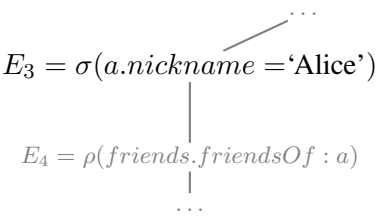
Given the expression E_1 we apply the Rule 2 for each unique type name into the projection expression. Then,

$$der(E_1) = \Big\{\ \bar{proj}_a(\{a\},\ \{\},\ \{a.nickname\}),\ \bar{proj}_l(\{l\},\ \{\},\ \{\ l.nickname,\ l.location.lat,\ l.location.lon\})\ \Big\} \cup der(E_2)$$



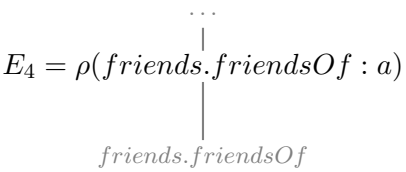
The expression E_2 holds the Rule 4 and it implies the combination of tuples of two subexpressions E_3 and E_5. Then,

$$der(E_2) = \Big\{ \bar{bind}_{a,l}(\{a,l\}, \{l.nickname = a.friendship.with.friend\}, atts(a) \cup atts(l)) \Big\} \cup der(E_3) \cup der(E_5)$$



The left branch goes first and the expression E_3 holds the Rule 3. Then,

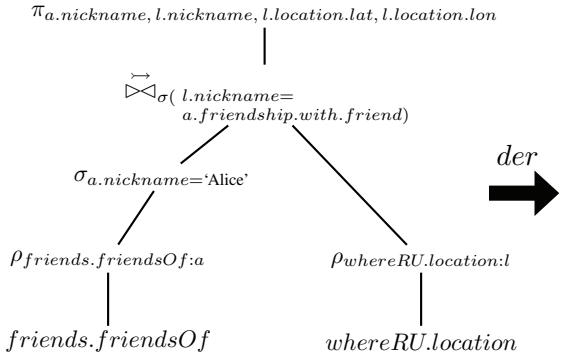
$$der(E_3) = \Big\{ \bar{retr}_a(\{a\}, \{a.nickname =\text{'Alice'}\}, atts(a)) \Big\} \cup der(E_4)$$



The expression E_4 holds the Rule 1 that considers the service operation $friends.friendsOf$ at the leaf node. Then, $der(E_4) = \{\}$ and $def(a) = def(friends.friendsof)$ is true.

Now the right branch is traversed and the expression E_5 holds the Rule 1 that considers the service operation $whereRU.location$ at the leaf node. Then, $der(E_5) = \{\}$ and $def(l) = def(whereRU.location)$ is true.

Finally, the backtracking leads to the following result:



$$\begin{aligned}
&\bar{proj}_a \quad (\{a\}, \quad \{\}, \{a.nickname\}),\\
&\bar{proj}_l \quad (\{l\}, \quad \{\}, \{l.nickname, l.location.lat, l.location.lon\}),\\
&\bar{bind}_{a,l} \quad (\{a,l\}, \{l.nickname = a.friendship.with.friend\}, atts(a) \cup atts(l)),\\
&\bar{retr}_a \quad (\{a\}, \quad \{a.nickname =\text{'Alice'}\}, atts(a)) \Big\}.
\end{aligned}$$

Each complex value operator in the $\mathcal{SC}$-$\mathcal{O}$ expression is represented by a *dt-function*. In particular for the renaming operator, it is assumed that the renaming of types is accessible via a dictionary. The *dt-functions* now can be analyzed for computing their data dependencies and the alternative orders to perform hybrid queries.

3.3 The SLA contract and query workflow cost

Associated to a hybrid query, there is a Service Level Agreement contract (SLA) that defines the cost expectations of the hybrid query evaluation. For representing SLA contracts, several languages have been proposed [SLE04] for expressing low-level aspects either from the network point of view[Ver04, BS05] as from the service provision point of view. Other languages reflect on high-level contracts in the context of concurrent privileges [DLP03, WCSO08] expressed as user nominations, *e.g.* platinum, gold, silver.

In this thesis we use an internal representation of an SLA contract in form of an optimization objective (see Figure 3.3). The optimization objective defines a combination of cost attributes, preferences among them, and the expected values to be reached by a query workflow (*i.e.* service coordination) implementing the hybrid query.

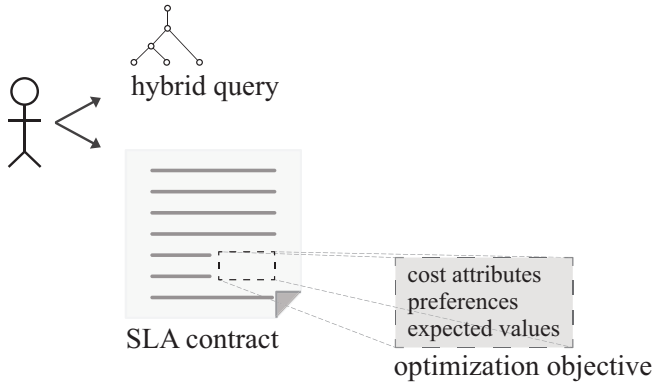


Figure 3.3 – Service Level Agreement / optimization objective

Applications or users requiring the evaluation of the same hybrid query may have different optimization objectives depending on their SLA contracts. For instance, suppose Alice wants to spend 1€ to access data and computing services and to use the 4G network. She also requires a response in less than 1 minute, and she does not want to spend too much energy so she limits the energy consumption to 50 energy units. These cost attributes are examples of the possible cost attributes in service-based environment. Users might have different preferences. For instance, Bob privileges data privacy and he does not care about the resulting price as long as he gets the response as soon as possible. The combination of QoS offered by services and networks have to fit the optimization objective depending if it comes from the SLA contract of Alice or Bob.

We define the optimization objective representation as follows.

Definition 3.3.1 [optimization objective] An optimization objective is represented by a vector of $m \in \mathbb{Z}_+$ weighted attributes $oo\langle(o^1, w^1), \ldots, (o^m, w^m)\rangle$ where for each $j \in [1\,..\,m]$:

o^j	represents an attribute-value pair of the form $att\_name^j = value^j$;
$att\_name^j \in \mathbb{S}$	represents the attribute name;
$value^j \in \mathbb{R}_+$	represents the ideal value for att_name^j attribute;
$w^j \in [1, \ldots, m]$	represents the weight of the attribute j.

The attribute name and attribute value are accessible via the functions $name$ and val respectively, *i.e.* $name(v^j) = att\_name^j$, $val(v^j) = value^j$.

The weights $w^1, .., w^m$ define a total order and denote the preferences among the m cost attributes. The semantics of the weights is defined as follows.

Definition 3.3.2 [Weight semantics] For every pair of attributes $p, q \in [1 \,..\, m],\ p \neq q$:

— the attribute p is preferred over the attribute q if $w^p > w^q$;

— the attribute p is preferred over the attribute $r \in [1 \,..\, m]$ if $w^p > w^q > w^r$ (*i.e.* transitivity [3]);

— there are no repeated weights $w^p \neq w^q$.

Example 3.3.1 Suppose the interests of Alice and Bob in previous example. The optimization objective of Alice is represented by $oo\langle(price = 1€, 3), (time = 1 \text{ min}, 2), (energy = 50 \text{ kJ}, 1)\rangle$; which means that the price has to be close to 1 €, and it is preferred over the time that has to be close to 1 min, and this in turn is preferred over the energy that has to be close to 50 kJ. On the other hand, the optimization objective of Bob is represented by $oo\langle(privacy = 1, 2), (time = 0 \text{ min}, 1)\rangle$; which means that the privacy is expected to be the maximum possible, and it is preferred over the time that is expected to be minimized.

As the optimization objective defines which combination of cost attributes must be considered by the hybrid query optimization, thus, it determines the cost attributes of query workflows.

Definition 3.3.3 [query workflow cost] Given an optimization objective $oo\langle(o^1, w^1), \ldots, (o^m, w^m)\rangle$ associated to a hybrid query, the cost of query workflows implementing such a hybrid query is represented by a vector of m attributes $qw\langle x^1, \ldots, x^m\rangle$ where for each $j \in [1 \,..\, m]$:

$name(x^j) = name(o^j)$ represents the j^{th} attribute name;

$val(x^j) \in \mathbb{R}_+$ is the cost attribute value of the j^{th} attribute.

The definitions above imply that for the same hybrid query and two optimization objectives with different cost attributes, the cost of query workflows implementing such a hybrid query will be different for each optimization objective. For instance, the costs of the query workflows implementing the hybrid query of Alice will be given by the cost attributes $price$, $time$, and $energy$; while the query workflows of the hybrid query of Bob will be given by the cost attributes $privacy$ and $time$.

3.4 Hybrid query optimization process

For optimizing hybrid queries we propose an optimization process where each stage is oriented to optimize the control-flow of query workflows implementing hybrid queries. The control-flow optimization responds to the need to open the search space in presence of multiple cost attributes whose optimal query workflow may not be met if considering only the data-flow.

3. There is a method in [Mor64] to define the preferences by pairwise judgments in order to avoid the transitivity and the implied cognitive effort in presence of many cost attributes (*e.g.* $m \gg 3$).

Consider the optimization process depicted in Figure 3.4. The input of the optimization is a hybrid query represented by a set of *dt-functions*. The *dt-functions* are used for generating alternative query workflows. Then, the query workflows are passed through the cost estimation that maps each query workflow to its corresponding multi-attribute cost according to the optimization objective. The result is the search space of query workflows with their costs. Finally, the solution space is computed by selecting the top-k query workflows.

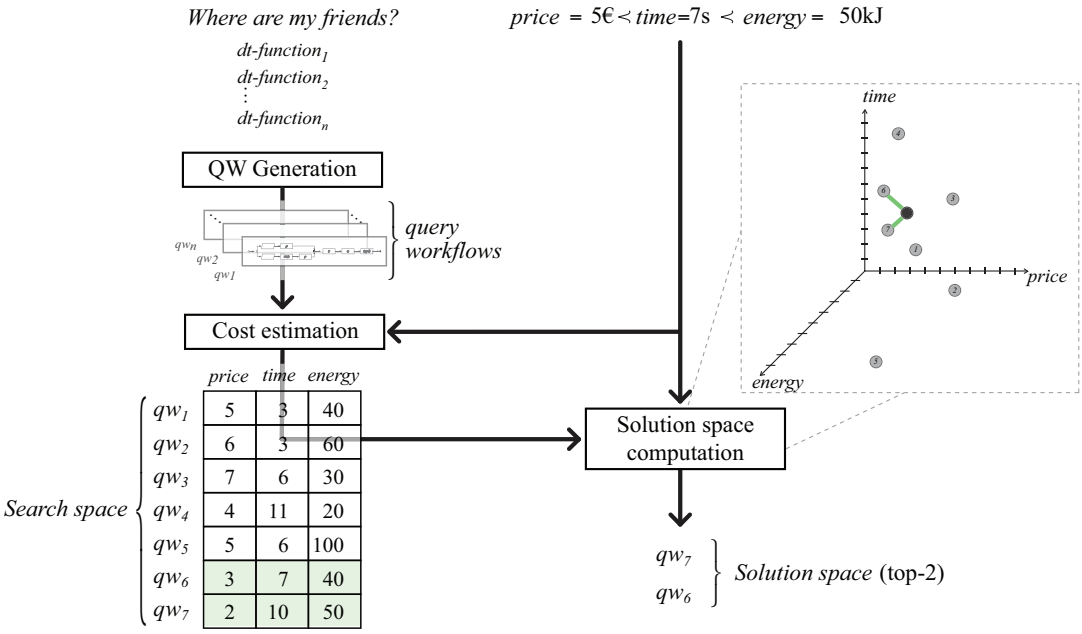


Figure 3.4 – Hybrid query optimization process

3.4.1 Query workflow generation

The generation of query workflows is oriented to open the search space of alternatives implementing a given hybrid query. This is because the multi-attribute optimization objective may not be reached with a search space oriented to optimize the data-flow. For instance, the optimization of the data-flow is interesting if the main interest is the reduction of the execution time. Nevertheless, if other cost attributes have to be considered (*e.g.* energy, price, privacy) more alternative query workflows are required.

The *dt-functions* representing a hybrid query are used to perform the generation of alternative query workflows with different control-flows. Figure 3.5 shows our model for generating query workflows. A query workflow organizes a set of activities through a control-flow that defines their execution order. An activity `f` implements a data transformation described by a *dt-function* $\bar{f}$ through the invocation of either a single service operation or several service operations. The control-flow among activities has to ensure the production of intermediate data such that the final result holds the hybrid query semantics. This implies that each activity receives the adequate data for performing its data transformation and produces the adequate data for the reminding activities. For this purpose we define a series of generation rules that produce query workflows with different grade of parallelism while the data dependencies among

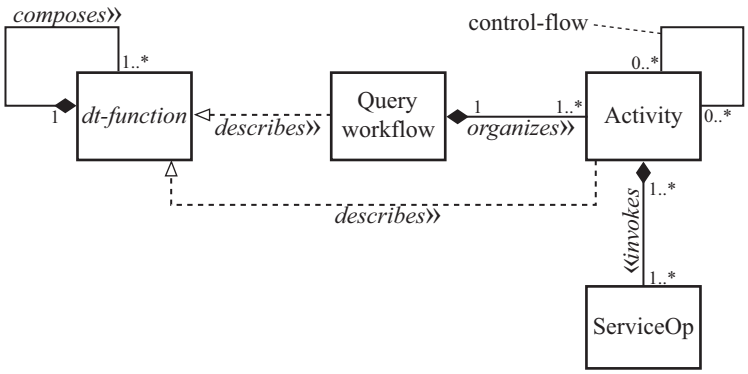


Figure 3.5 – Description of activities and query workflows by *dt-functions*

activities hold.

3.4.2 Cost estimation

Due to the autonomy of services, we assume the absence of data-related information (*e.g.* selectivity, cardinality, attribute size) during the optimization. Our approach to estimate the cost is the adoption of a build-time cost function (See Figure 3.6) that explicitly assumes the absence of such an information. In turn, we consider the cost attributes defined by the optimization objective (*cf.* Section 3.3) and how activities interact among them and with services through the control-flow.

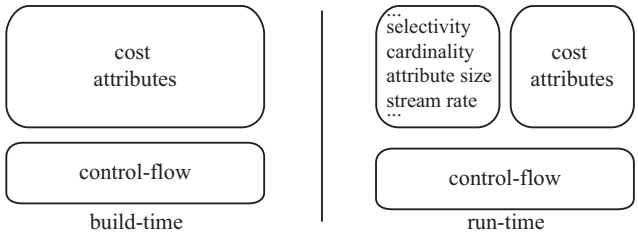


Figure 3.6 – Cost estimation for query workflows

The set of query workflows with their estimated costs form the search space. For instance, the search space is represented by the matrix $\{qw_1, ..., qw_7\} \times \{price, time, energy\}$ in Figure 3.4.

3.4.3 Solution space selection

The solution space selection obtains the closest query workflows from the optimization objective. This is done in two steps:

1. Computation of a distance between the optimization objective and the query workflows in the search space. The distance is weighted w.r.t. the attribute preferences defined by the optimization objective. This allows to make comparable query workflows with different costs but with the same distance from the optimization objective.

2. Selection of the query workflows that represent the best alternatives for implementing the hybrid query. We adopt a top-k algorithm [Fag99] that is devoted to treat with fuzzy data, *i.e.* query workflow cost.

In order to illustrate the solution space selection, consider the chart in Figure 3.7.

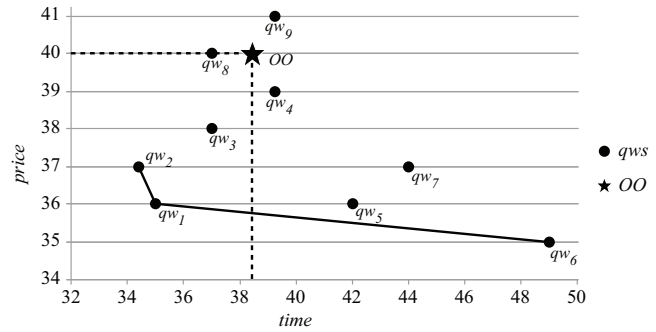


Figure 3.7 – Semantics of the optimization objective

The chart presents the search space formed by nine query workflows implementing the same hybrid query, and the optimization objective $oo\langle(time = 38.4, 1), (price = 40, 2)\rangle$. The query workflows qw_1, qw_2, and qw_6 form the Pareto optimal as they represent the smaller cost for both time and price attributes. Nevertheless, we are interested in those query workflows that are the closest to the optimization objective represented by the point $(38.4, 40.0)$. In such a case, qw_4, qw_8 and qw_9 represent the best alternatives for implementing the hybrid query. Using the weights of cost attributes, qw_4 represents the best alternative because the weights privilege the price over the time.

3.5 Conclusions

This chapter presented the approach for optimizing hybrid queries over on-demand and stream data services. We adopted the *dt-function* notion as an intermediate representation of complex value operators out of the hybrid query specification. Behind the notion of *dt-functions*, there is the interest to describe activities implementing complex value operators whose implementation details are assumed to be unknown in service-based environments.

We also introduced the Service Level Agreement contracts associated to hybrid queries and the internal representation in form of optimization objective. The optimization objective defines the cost attributes, preferences, and expected values of the query workflow execution. The idea to have an optimization objective associated to an SLA corresponds with the fact that the optimization interests in service-based environments may vary from applications or application instances, *e.g.* user interests. We therefore propose a hybrid query optimization process that generates the alternative query workflows based on the *dt-functions* representing the hybrid query. We consider both the data-flow and control-flow of query workflows in order to obtain a search space with more alternative query workflows in presence of the potential combinations of cost attributes. For selecting the most suitable query workflows we adapt a top-k that obtains the best query workflows for no matter which combination of attributes. The top-k algorithm enables the extension in case of new cost attributes to consider.

CHAPTER 4

Generation of the search space of query workflows

This chapter presents the generation of the search space of query workflows implementing a hybrid query represented by a set of *dt-functions*. The generation aims to obtain more alternative query workflows by considering both the data-flow and control-flow of query workflows.

The remainder of the chapter is organized as follows. Section 4.1 introduces the query workflow definition along with the properties to ensure the well construction, consistency, and equivalence of query workflows. Section 4.2 defines the composition of activities for generating consistent query workflows by means of generation rules. Section 4.3 presents the algorithm for generating equivalent query workflows by applying the generation rules. Finally, Section 4.4 concludes this chapter.

4.1 Query workflow definition and properties

A query workflow organizes a set of activities through a control-flow that defines their execution order. Formally, a query workflow is represented by a pair of (1) a Directed Acyclic Graph (DAG) whose vertices and arcs represent the sequential and parallel control-flows among activities, and (2) a *dt-function* that describes the data transformation done by the query workflow.

Definition 4.1.1 [query workflow] A query workflow is represented by a tuple $\langle qw, \bar{f} \rangle$ where

— qw represents a DAG $qw(A, P, V, E, \texttt{in}, \texttt{out})$ [1] where

- A: is a set of activities;
- P: is a set of parallel constructors such that $\{\texttt{par}^l, \texttt{end\_par}^l\} \subseteq P$;
- `in` and `out`: are the first and last vertices respectively;
- V: is a set of all the graph vertices such that $V = A \cup P \cup \{\texttt{in}, \texttt{out}\}$;
- E: is a set of arcs between vertices such that $E \subset V \times V$.

— $\bar{f}$ represents a *dt-function* describing the composition of every activity in A such that $\bar{f} = merge(\bar{f}_1, merge(\bar{f}_2, merge(..., \bar{f}_n)))$ where $\bar{f}_i$ describes the activity $\texttt{f}_i \in A$.

1. Sometimes we use $qw(A, P, E, \texttt{in}, \texttt{out})$ for short.

The vertices of the query workflow graph are identified by unique names. In particular for parallel constructors, we use (when ambiguity) the label l to identify a single parallel composition, *i.e.* $\texttt{par}^l$ and $\texttt{end\_par}^l$ split and join the parallel composition l. When convenient, we use $qw_{\bar{f}}$ for denoting the query workflow graph described by $\bar{f}$, and $\bar{f}_{qw}$ for denoting the *dt-function* describing qw.

Example 4.1.1 Consider the graphical representation of a query workflow in Figure 4.1, and its corresponding graph $qw(A, P, V, E, \texttt{in}, \texttt{out})$ where

— $A = \{\texttt{f}_a, \texttt{f}_b, \texttt{f}_c, \texttt{f}_d\}$;

— $P = \{\texttt{par}^l, \texttt{end\_par}^l\}$;

— $V = A \cup P \cup \{\texttt{in}, \texttt{out}\}$;

— $E = \{(\texttt{in}, \texttt{par}^l), (\texttt{par}^l, \texttt{f}_a), (\texttt{f}_a, \texttt{end\_par}^l), (\texttt{par}^l, \texttt{f}_b), (\texttt{f}_b, \texttt{f}_c), (\texttt{f}_c, \texttt{end\_par}^l), (\texttt{end\_par}^l, \texttt{f}_d), (\texttt{f}_d, \texttt{out})\}$.

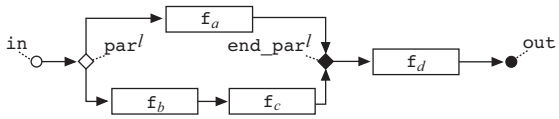


Figure 4.1 – Graphical representation of a query workflow

This query workflow defines the control-flow among four activities $\texttt{f}_a$, $\texttt{f}_b$, $\texttt{f}_c$ and $\texttt{f}_d$ described by $\bar{f}_a, \bar{f}_b, \bar{f}_c, \bar{f}_d$ respectively. The two first activities to be executed in parallel are $\texttt{f}_a$, and $\texttt{f}_b$. Once $\texttt{f}_b$ finishes, $\texttt{f}_c$ is executed. After the parallel composition and once $\texttt{f}_a$ and $\texttt{f}_c$ have finished, the last activity $\texttt{f}_d$ is executed. The `in` and `out` vertices denote respectively the beginning and termination of the query workflow execution. The result of the query workflow is described by the *dt-function* $\bar{f} = merge(\bar{f}_a, merge(\bar{f}_b, merge(\bar{f}_c, \bar{f}_d)))$.

Multiple query workflows can implement the same hybrid query through different control-flows which means that they are equivalent.

Property 4.1.1 [query workflow equivalence] Given a set of *dt-functions* $\bar{F}$ derived from a hybrid query expression, and two different query workflows qw_1 and qw_2, it is said that qw_1 and qw_2 are equivalent if both qw_1 and qw_2 are described by the same *dt-function* $\bar{f} = merge(\bar{f}_1, merge(\bar{f}_2, merge(..., \bar{f}_n)))$ where each $\bar{f}_i \in \bar{F}$.

In order to ensure the termination and the correct execution of query workflows, it is required to define the well construction and consistency of query workflows. The well construction of a query workflow is about the structural constraints to be held by its control-flow for ensuring that the last vertex `out` is reached and every activity is executed. The consistency of a query workflow is about its data-flow that ensures that every activity receives the correct data to perform its data transformation.

4.1.1 Well constructed query workflow

A well constructed query workflow has a control-flow that ensures the eventual execution and termination of every activity. For defining the well constructed query workflow property we assume the execution paths function and precedence relations.

Definition 4.1.2 [execution paths] The function $epaths(\mathtt{f_a}, \mathtt{f_z}, E)$ computes a set P of execution paths from the vertex $\mathtt{f_a}$ to the vertex $\mathtt{f_z}$ within E. For a given query workflow graph $qw(A, P, V, E, \mathtt{in}, \mathtt{out})$ and two different vertices $\mathtt{f_a}, \mathtt{f_z} \in V$, the set of execution paths between $\mathtt{f_a}$ and $\mathtt{f_z}$ are given by:

$$epaths(\mathtt{f_a}, \mathtt{f_z}, E) = \begin{cases} \{(\mathtt{f}_a, \mathtt{f}_z)\} & \text{if } (\mathtt{f}_a, \mathtt{f}_z) \in E \\ \{(\mathtt{f}_a, \mathtt{f}_b)\} \cup epaths(\mathtt{f_b}, \mathtt{f_z}, E) & \text{if } \exists (\mathtt{f}_a, \mathtt{f}_b) \in E \end{cases}$$

For instance, the execution paths $epaths(\mathtt{in}, \mathtt{out}, E)$ of the query workflow in Figure 4.1 are two, one for each branch of the parallel composition (see Figure 4.2).

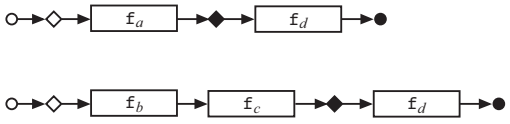


Figure 4.2 – Execution paths

Property 4.1.2 [precedence] The precedence of a vertex with respect to another denotes that both belong to the same execution path and are connected through a chain of sequential control-flows. For a given query workflow graph $qw(A, P, V, E, \mathtt{in}, \mathtt{out})$, the precedence of two different vertices $\mathtt{v_1}, \mathtt{v_2} \in V$, denoted by $\mathtt{v_1} \prec_{qw} \mathtt{v_2}$, is true if $|epaths(\mathtt{v_1}, \mathtt{v_2}, E)| \geq 1$.

For instance, in Figure 4.2, both $\mathtt{f_a} \prec_{qw} \mathtt{f_d}$ and $\mathtt{f_b} \prec_{qw} \mathtt{f_d}$ are true as $\mathtt{f}_a$ is in the same execution path with $\mathtt{f}_d$ and $\mathtt{f}_b$ does with $\mathtt{f}_d$.

Property 4.1.3 [absolute precedence] The absolute precedence of a vertex respect to another denotes that the termination of the first one triggers the execution of the second one in a sequential control-flow. For a given query workflow graph $qw(A, P, V, E, \mathtt{in}, \mathtt{out})$, the absolute precedence of two different vertices $\mathtt{v}_1, \mathtt{v}_2 \in V$, denoted by $\mathtt{v_1} \prec^{!}_{qw} \mathtt{v_2}$, is true if $\mathtt{v_1}$ and $\mathtt{v_2}$ are adjacent vertex such that $\mathtt{v_1} \prec_{qw} \mathtt{v_2}, \nexists \mathtt{v}_3 \in V \mid \mathtt{v_1} \prec_{qw} \mathtt{v_3} \prec_{qw} \mathtt{v_2},\ \mathtt{v_1} \neq \mathtt{v_2} \neq \mathtt{v}_3$.

For instance, in Figure 4.2, $\mathtt{f_b} \prec^{!}_{qw} \mathtt{f_c}$ is true as $\mathtt{f}_b$ and $\mathtt{f}_c$ are in the same execution path and there is not other vertex between them.

Given the precedence properties, we define a well constructed query workflow as follows.

Property 4.1.4 [well constructed query workflow] Given a query workflow graph $qw(A, P, V, E, \texttt{in}, \texttt{out})$, it is said that qw is a well constructed if every activity in A starts and terminates its execution in the middle of `in` and `out` vertices and once the `out` vertex is reached the query workflow execution terminates.

A well constructed query workflow holds the following conditions:

1. **No-ambiguity.** qw has no-ambiguity if there is only one initial vertex `in` and only one final vertex `out`. This is $\forall_{\texttt{v} \in V \setminus \{\texttt{in},\texttt{out}\}} \exists \texttt{v}_{\texttt{prev}}, \texttt{v}_{\texttt{succ}} \in V \mid \texttt{v}_{\texttt{prev}} \prec^{!}_{qw} \texttt{v} \prec^{!}_{qw} \texttt{v}_{\texttt{succ}}$.
2. **No-isolated activities.** qw has no isolated activities if there is not a vertex out of a path from `in` to `out`. The inclusion of all the vertices into the paths from `in` to `out` means that all the vertices precede the `out` vertex and thus there are no isolated activities. This is $\forall_{\texttt{v} \in V \setminus \{\texttt{out}\}} \texttt{v} \prec_{qw} \texttt{out}$.
3. **No-deadlock.** qw has no deadlock if its parallel compositions are all balanced. This is, for a vertex $\texttt{par}^l$ there is one and only one $\texttt{end\_par}^l$ such that $\forall_{\texttt{par}^l \in P} \exists! \texttt{end\_par}^l \in P \mid \texttt{par}^1 \prec_{qw} \texttt{end\_par}^1$, and symmetrically $\forall_{\texttt{end\_par}^l \in P} \exists! \texttt{par}^l \in P \mid \texttt{par}^1 \prec_{qw} \texttt{end\_par}^1$.
4. **No-incomplete activities** qw has no incomplete activities if once the `out` activity is reached, there are neither unexecuted nor unterminated activities. From the structural point of view, the *no-isolated activities* property ensures that all the activities are in a path from `in` to `out` and thus, all the activities executions are launched. From the termination point of view, the activity termination is ensured under the assumption that an activity within an execution path is reliable if it gets the required input.

Besides the structural constraints defined by the well constructed query workflow property, the query workflow must have a consistent data-flow among activities in order to ensure that each one gets the correct input from preceding activities and produces the correct output for succeeding activities.

4.1.2 Consistent query workflow

The consistency of a query workflow is a property that ensures that activities within its control-flow get the correct input, and eventually terminate –under the assumption that activities are reliable– producing then the correct output for the reminding activities. For defining such a property we reflect on the data dependencies among activities and the consistent data-flow among them. We refer the reader to the Appendix A.1 for definitions and examples of data dependencies.

1. **Retrieval dependency**, denoted by $\texttt{f}_a \xrightarrow{\text{r}} \texttt{f}_b$, states that $\texttt{f}_a$ retrieves the tuples produced by a data service and $\texttt{f}_b$ uses such tuples to perform data transformations.
2. **Anti-dependency**, denoted by $\texttt{f}_a \xrightarrow{\text{a}} \texttt{f}_b$, happens when an attribute required by the data transformation of $\texttt{f}_b$ is removed by $\texttt{f}_a$, *e.g.* $\texttt{f}_a$ does not project an attribute whose values are used by $\texttt{f}_b$ to invoke a service operation.

3. **Circular dependency**, denoted by $\mathtt{f}_a \leftrightarrow \mathtt{f}_b$, is a mutual dependency between $\mathtt{f}_a$ and $\mathtt{f}_b$ by either retrieval dependency or anti-dependency, *e.g.* $\mathtt{f}_a$ requires the output attribute values produced by $\mathtt{f}_b$, and this in turn requires the output attribute values produced by $\mathtt{f}_a$.

Property 4.1.5 [Consistent data-flow] Let be $\mathtt{f}_a$ and $\mathtt{f}_b$ two activities described by the *dt-functions* $\bar{f}_a(\mathcal{A}_a, \mathcal{E}_a, \mathcal{P}_a)$ and $\bar{f}_b(\mathcal{A}_b, \mathcal{E}_b, \mathcal{P}_b)$ respectively. It is said that $\mathtt{f}_a$ and $\mathtt{f}_b$ have a consistent data-flow if there is not a circular dependency (*i.e.* $\neg(\mathtt{f}_a \leftrightarrow \mathtt{f}_b)$) and one of the next rules holds:

1. if $\mathcal{A}_a \cap \mathcal{A}_b = \{\}$ is true and $\mathtt{f}_a$,$\mathtt{f}_b$ are activities of a query workflow $qw(A, P, E, \mathtt{in}, \mathtt{out})$ such that $\mathtt{f}_a, \mathtt{f}_b \in A$. Intuitively, as two activities are independent, the consistent data-flow between them is ensured if they belong to a well constructed query workflow.
2. if $\mathtt{f}_a \xrightarrow{\mathrm{r}} \mathtt{f}_b$ or $\mathtt{f}_a \xrightarrow{\mathrm{a}} \mathtt{f}_b$ is true and $\mathtt{f}_a$ is performed before $\mathtt{f}_b$ in a single execution path of a query workflow qw such that $\mathtt{f}_a \prec_{qw} \mathtt{f}_b$. Intuitively, as two activities have a data dependency they must belong to the same execution path in an order which ensures their correct execution.
3. $\mathcal{A}_a \cap \mathcal{A}_b \neq \{\}$ is true and there is neither a retrieval dependency nor an anti-dependency. The activities $\mathtt{f}_a$ and $\mathtt{f}_a$ are in a single execution path of a query workflow qw such that $\mathtt{f_a} \prec_{qw} \mathtt{f_b}$ or $\mathtt{f_b} \prec_{qw} \mathtt{f_a}$. Intuitively, two activities modify the same tuple values and, as they do not have data dependencies, they just have to belong to the same execution path regarding-less on the order.

Given the consistent data-flow property, we define a consistent query workflow as follows.

Property 4.1.6 [Consistent query workflow] A query workflow $qw(A, P, V, E, \mathtt{in}, \mathtt{out})$ is consistent if every pair of activities $\mathtt{f}_a, \mathtt{f}_b \in A$ have a consistent data-flow.

Now we turn our attention into the construction of consistent query workflows through a series of generation rules that define sequential or parallel compositions for certain control-flow relations among activities.

4.2 Generation rules

The generation of query workflows is performed by composing pairs of activities whose *dt-functions* hold a control-flow relation (cf-relation). For each cf-relation, we propose a series of generation rules that define either sequential or parallel control-flows and produces consistent query workflows.

4.2.1 Independent cf-relation

Two activities with an independent cf-relation transform different sets of tuples and then can be composed in a sequential or parallel control-flow.

Property 4.2.1 [Independent cf-relation] Given two activities $\mathtt{f}_a$ and $\mathtt{f}_b$ described by the *dt-functions* $\bar{f}_a(\mathcal{A}_a, \mathcal{E}_a, \mathcal{P}_a)$ and $\bar{f}_b(\mathcal{A}_b, \mathcal{E}_b, \mathcal{P}_b)$ respectively. It is said that they have an independent cf-relation, denoted by $\mathtt{f}_a \shortparallel \mathtt{f}_b$, if they do not share any type name such that $\mathcal{A}_a \cap \mathcal{A}_b = \{\}$. Observe that the independent cf-relation is commutative and thus $\mathtt{f}_a \shortparallel \mathtt{f}_b = \mathtt{f}_b \shortparallel \mathtt{f}_a$.

Both activities transform the disjunctive tuple values and the execution of one does not depend on the termination of the other. Therefore, the activities must be composed in any sequential or parallel order.

Proposition 4.2.1 [Compositions for independent cf-relation] Two activities with independent cf-relation $\mathtt{f}_a \shortparallel \mathtt{f}_b$, hold the consistent data-flow property by the Rule 1 if they are composed by the query workflows in Table 4.1.

Proof 4.2.1 [Coherent data-flow for independent cf-relation] Each resulting query workflow in Table 4.1 is a well constructed query workflow and both activities $\mathtt{f}_a$ and $\mathtt{f}_b$ belong the set of activities A. Therefore, if $\mathtt{f}_a \shortparallel \mathtt{f}_b$ holds, the query workflows in Table 4.1 are consistent query workflows. ■

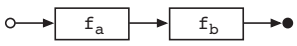
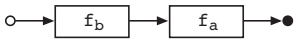
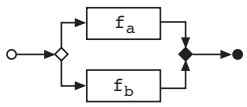
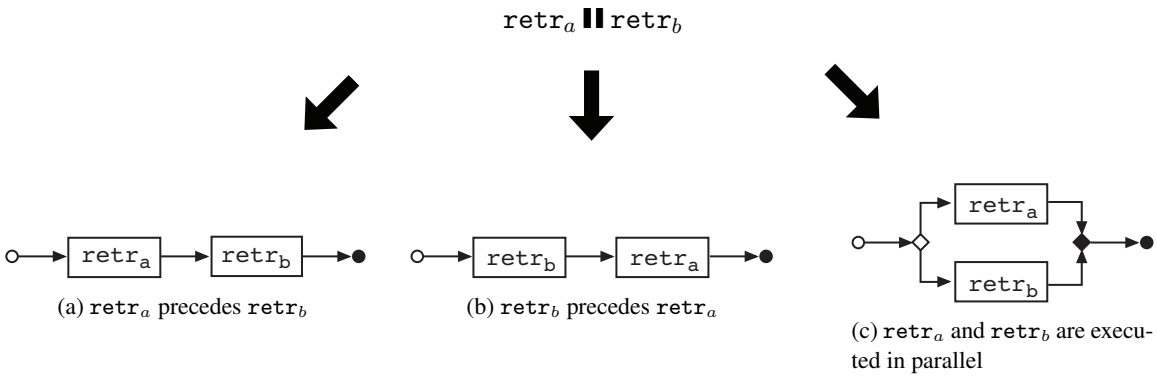
Query workflow graph			Graphical representation
$qw(A, P, E, \mathtt{in}, \mathtt{out})$	where	$A = \{\mathtt{f}_a, \mathtt{f}_b\}$; $P = \{\}$; $E = \{(\mathtt{in}, \mathtt{f}_a), (\mathtt{f}_a, \mathtt{f}_b), (\mathtt{f}_b, \mathtt{out})\}$.	
$qw(A, P, E, \mathtt{in}, \mathtt{out})$	where	$A = \{\mathtt{f}_a, \mathtt{f}_b\}$; $P = \{\}$; $E = \{(\mathtt{in}, \mathtt{f}_b), (\mathtt{f}_b, \mathtt{f}_a), (\mathtt{f}_a, \mathtt{out})\}$.	
$qw(A, P, E, \mathtt{in}, \mathtt{out})$	where	$A = \{\mathtt{f}_a, \mathtt{f}_b\}$; $P = \{\mathtt{par}^l, \mathtt{end\_par}^l\}$; $E = \{(\mathtt{in}, \mathtt{par}^l), (\mathtt{par}^l, \mathtt{f}_a),$ $(\mathtt{f}_a, \mathtt{end\_par}^l), (\mathtt{par}^l, \mathtt{f}_b),$ $(\mathtt{f}_b, \mathtt{end\_par}^l), (\mathtt{end\_par}^l, \mathtt{out})\}$.	

Table 4.1 – Query workflows for independent cf-relation $\mathtt{f}_a \shortparallel \mathtt{f}_b$

Example 4.2.1 Suppose the activities $\mathtt{retr}_a$, $\mathtt{retr}_b$ and their corresponding *dt-functions* $\bar{retr}_a(\{a\}, \{a.nickname{=}\text{'Alice'}\}, atts(a))$ and $\bar{retr}_b(\{b\}, \{b.nickname{=}\text{'Bob'}\}, atts(b))$. Both activities retrieve tuples of different type names, *i.e.* , a,b. As $\mathtt{retr}_a$ and $\mathtt{retr}_b$ do not share type names, therefore the cf-relation $\mathtt{retr}_a \shortparallel \mathtt{retr}_b$ is true and the activities are organized by the compositions defined by the query workflows in Figure 4.3.

The first and second query workflows define the sequential control-flow between $\mathtt{retr}_a$ and $\mathtt{retr}_b$. The independence is more evident in the parallel composition where both activities start their executions simultaneously (from the structural point of view) and, once both have finished, the parallel composition converges.



(a) $\texttt{retr}_a$ precedes $\texttt{retr}_b$ (b) $\texttt{retr}_b$ precedes $\texttt{retr}_a$ (c) $\texttt{retr}_a$ and $\texttt{retr}_b$ are executed in parallel

Figure 4.3 – Query workflows generated for the independent cf-relation

4.2.2 Dependent cf-relation

Two activities with a dependent cf-relation transform the same set of tuples and one of the activities requires the termination of the other one.

Property 4.2.2 [Dependent cf-relation] Given two activities $\texttt{f}_a$ and $\texttt{f}_b$ described by the *dt-functions* $\bar{f}_a(\mathcal{A}_a, \mathcal{E}_a, \mathcal{P}_a)$ and $\bar{f}_b(\mathcal{A}_b, \mathcal{E}_b, \mathcal{P}_b)$ respectively. It is said that they have a dependent cf-relation, denoted by $\texttt{f}_a \blacktriangleright\!\blacktriangleright \texttt{f}_b$, if one of the following conditions holds:

1. $\texttt{f}_a \xrightarrow{\text{r}} \texttt{f}_b$ There is a retrieval dependency
2. $\texttt{f}_a \xrightarrow{\text{a}} \texttt{f}_b$ There is an anti-dependency

Observe that the dependent cf-relation is not commutative and thus $\texttt{f}_a \blacktriangleright\!\blacktriangleright \texttt{f}_b \neq \texttt{f}_b \blacktriangleright\!\blacktriangleright \texttt{f}_a$.

Both activities transform the same tuple values and the execution of one depends on the termination of the other. Therefore, the activities must be composed in a sequential order.

Proposition 4.2.2 [Compositions for dependent cf-relation] Two activities with dependent cf-relation $\texttt{f}_a \blacktriangleright\!\blacktriangleright \texttt{f}_b$, hold the consistent data-flow property by the Rule 2 if they are composed by the query workflows in Table 4.2.

Proof 4.2.2 [Coherent data-flow for dependent cf-relation] The query workflow in Table 4.2 is a well constructed query workflow and the activities $\texttt{f}_a$ and $\texttt{f}_b$ are composed though the same execution path in strict order such that $\texttt{f}_\texttt{a} \prec_{qw} \texttt{f}_\texttt{b}$ is true. Therefore, if $\texttt{f}_a \blacktriangleright\!\blacktriangleright \texttt{f}_b$ holds, the query workflow in Table 4.2 is a consistent query workflow. ■

Example 4.2.2 [retrieval dependency] Suppose the activities $\texttt{retr}_a$, $\texttt{proj}_a$ and their corresponding *dt-functions* $\bar{retr}_a(\{a\}, \{a.nickname = \text{'Alice'}\}, atts(a))$ and $\bar{proj}_a(\{a\}, \{\}, \{a.nickname\})$. The activity $\texttt{retr}_a$ retrieves the tuples denoted by the type name a by providing the required input attribute values. Once the tuples of a have been retrieved, the activity $\texttt{proj}_a$ projects the attribute $a.nickname$. As $\texttt{retr}_a$

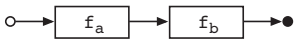
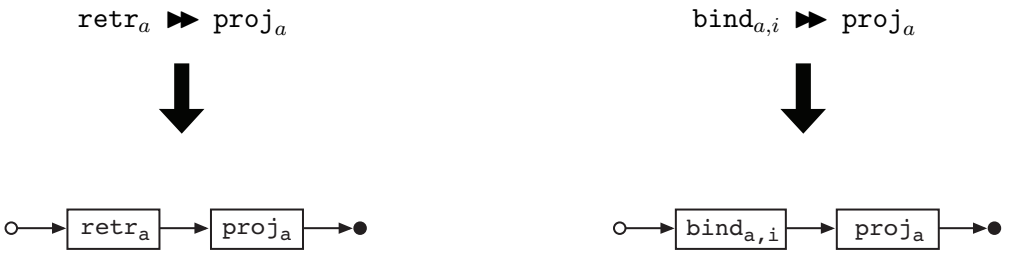
Query workflow graph		Graphical representation
$qw(A, P, E, \texttt{in}, \texttt{out})$	where $A = \{\texttt{f}_a, \texttt{f}_b\}$; $P = \{\}$; $E = \{(\texttt{in}, \texttt{f}_a), (\texttt{f}_a, \texttt{f}_b), (\texttt{f}_b, \texttt{out})\}$.	

Table 4.2 – Query workflow for dependent cf-relation $\texttt{f}_a$ ▶▶ $\texttt{f}_b$



(a) $\texttt{retr}_a$ precedes $\texttt{proj}_a$ (b) $\texttt{bind}_{a,i}$ precedes $\texttt{proj}_a$

Figure 4.4 – Query workflows generated for the dependent cf-relation

and $\texttt{proj}_a$ have a retrieval dependency $\texttt{retr}_a \xrightarrow{\text{r}} \texttt{proj}_a$, therefore the cf-relation $\texttt{retr}_a$ ▶▶ $\texttt{proj}_a$ is true and the activities are composed as shown in Figure 4.4a.

Example 4.2.3 [anti-dependency] Suppose the activities $\texttt{bind}_{a,i}$, $\texttt{proj}_a$ and their corresponding *dt-functions* $\bar{bind}_{a,i}(\{a, i\}, \{i.nickname = a.friendship.with.friend\}, atts(a) \cup atts(i))$ and $\bar{proj}_a(\{a\}, \{\}, \{a.nickname\})$. The activity $\texttt{bind}_{a,i}$ retrieves the tuples denoted by the type name i by binding the input attribute $i.nickname$ with the value of the attribute $a.friendship.with.friend$. On the other hand, $\texttt{proj}_a$ eliminates the attribute $a.friendship.with.friend$ and thus it is no longer available. As $\texttt{retr}_a$ and $\texttt{proj}_a$ have an anti-dependency $\texttt{bind}_{a,i} \xrightarrow{\text{a}} \texttt{proj}_a$, therefore the cf-relation $\texttt{bind}_{a,i}$ ▶▶ $\texttt{proj}_a$ is true and the activities are composed as shown in Figure 4.4b.

4.2.3 Concurrent cf-relation

Two activities with a concurrent cf-relation transform the same set of tuples but no one requires the termination of the other.

Property 4.2.3 [Concurrent cf-relation] Given two activities $\texttt{f}_a$ and $\texttt{f}_b$ described by the *dt-functions* $\bar{f}_a(\mathcal{A}_a, \mathcal{E}_a, \mathcal{P}_a)$ and $\bar{f}_b(\mathcal{A}_b, \mathcal{E}_b, \mathcal{P}_b)$ respectively. It is said that they have a concurrent cf-relation, denoted by $\texttt{f}_a$ ⋈ $\texttt{f}_b$, if they share type names such that $\mathcal{A}_a \cap \mathcal{A}_b \neq \{\}$ and the following conditions hold:

1. $\neg(\texttt{f}_a \xrightarrow{\text{r}} \texttt{f}_b) \wedge \neg(\texttt{f}_a \xrightarrow{\text{a}} \texttt{f}_b)$ — There is neither a retrieval dependency nor an anti-dependency
2. $\neg\ (\texttt{f}_a \leftrightarrow \texttt{f}_b)$ — There is not a circular dependency

Observe that the concurrent cf-relation is commutative and thus $\mathtt{f}_a \bowtie \mathtt{f}_b = \mathtt{f}_b \bowtie \mathtt{f}_a$.

Both activities transform the same tuple values and the order in which they are composed does not modify the result (*i.e.* commutativity) because they do not have data dependencies. Therefore, the activities may be composed in any sequential order.

Proposition 4.2.3 [Compositions for concurrent cf-relation] Two activities with concurrent cf-relation $\mathtt{f}_a \bowtie \mathtt{f}_b$, hold the consistent data-flow property by the Rule 3 if they are composed by the query workflows in Table 4.3.

Proof 4.2.3 [Coherent data-flow for concurrent cf-relation] The query workflows in Table 4.3 are well constructed query workflows and the activities $\mathtt{f}_a$ and $\mathtt{f}_b$ are composed though the same execution path such that $\mathtt{f_a} \prec_{qw} \mathtt{f_b}$ is true or $\mathtt{f_b} \prec_{qw} \mathtt{f_a}$ is true. Therefore, if $\mathtt{f}_a \bowtie \mathtt{f}_b$ holds, the query workflows in Table 4.3 are consistent query workflows. ■

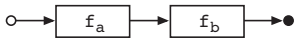
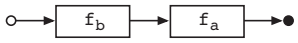
Query workflow graph			Graphical representation
$qw(A, P, E, \mathtt{in}, \mathtt{out})$	where	$A = \{\mathtt{f}_a, \mathtt{f}_b\}$; $P = \{\}$; $E = \{(\mathtt{in}, \mathtt{f}_a), (\mathtt{f}_a, \mathtt{f}_b), (\mathtt{f}_b, \mathtt{out})\}$.	
$qw(A, P, E, \mathtt{in}, \mathtt{out})$	where	$A = \{\mathtt{f}_a, \mathtt{f}_b\}$; $P = \{\}$; $E = \{(\mathtt{in}, \mathtt{f}_b), (\mathtt{f}_b, \mathtt{f}_a), (\mathtt{f}_a, \mathtt{out})\}$.	

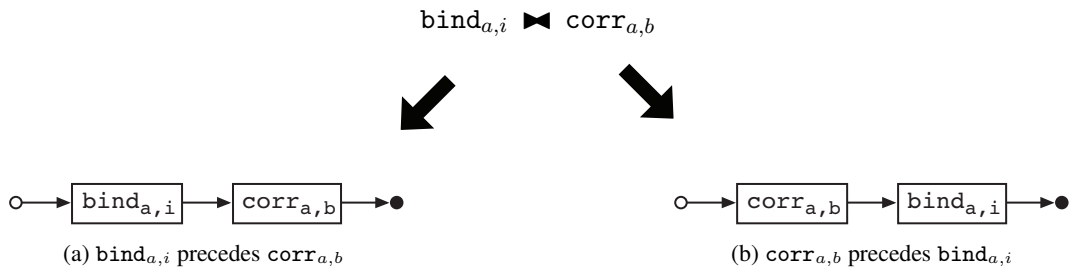
Table 4.3 – Query workflows for concurrent cf-relation $\mathtt{f}_a \bowtie \mathtt{f}_b$

Example 4.2.4 Suppose the activities $\mathtt{bind}_{a,i}$, $\mathtt{corr}_{a,b}$ and their corresponding *dt-functions* $\bar{bind}_{a,i}(\{a, i\}, \{i.nickname = a.friendship.with.friend\}, atts(a) \cup atts(i))$ and $\bar{corr}_{a,b}(\{a, b\}, \{a.friendship.with.friend = b.friendship.with.friend\}, atts(a) \cup atts(b))$. The activities $\mathtt{bind}_{a,i}$ and $\mathtt{corr}_{a,b}$ share the type name a. Whereas $\mathtt{bind}_{a,i}$ reads the tuples of a for retrieving the tuples of i, $\mathtt{corr}_{a,b}$ also reads the tuples of a to combine them with b. Therefore, there is not a data dependency and thus the cf-relation $\mathtt{bind}_{a,i} \bowtie \mathtt{corr}_{a,b}$ is true. The activities are organized by the compositions shown in Figure 4.5.

These query workflows define the sequential control-flow between $\mathtt{bind}_{a,i}$ and $\mathtt{bind}_{a,i}$. In both cases, the first activity terminates and produces the tuple values with the constituent attributes required by the second one and (implicitly) in any case the tuple values required by the second one are available.

4.3 Generation algorithm

The query workflow generation algorithm performs a pairwise reduction of a graph that represents the cf-relations among activities like the GOO algorithm in [Feg98]. The reduction can be done in different



(a) $\texttt{bind}_{a,i}$ precedes $\texttt{corr}_{a,b}$ (b) $\texttt{corr}_{a,b}$ precedes $\texttt{bind}_{a,i}$

Figure 4.5 – Query workflows generated for the concurrent cf-relation

orders producing equivalent query workflows through the application of the generation rules presented in Section 4.2.

4.3.1 Search space graph

The search space graph $ss\text{-}graph$ represents the ways to generate the search space of equivalent query workflows implementing a hybrid query. The vertices represent query workflows, with possibly only one activity, and the edges represent the cf-relations among them. The $ss\text{-}graph$ is defined as follows.

Definition 4.3.1 [ss-$graph$] For a given set of *dt-functions* $\bar{F}$ where each $\bar{f} \in \bar{F}$ describes an activity $\texttt{f}$, the search space of query workflows is described by a graph $ss\text{-}graph(QW, \vec{R}_h)$ where

— QW: is a set of query workflows $\langle qw, \bar{f} \rangle$ where qw is a one-activity query workflow (see Figure 4.6) such that $QW = \{\langle qw(\{\texttt{f}\}, \{\}, \{\texttt{f}\}, \{(\texttt{in}, \texttt{f}), (\texttt{f}, \texttt{out})\}, \texttt{in}, \texttt{out}), \bar{f} \rangle \mid \bar{f} \in \bar{F}\}$.

— $\vec{R}_h$: is the set of healthy cf-relations [2] among activities. Thus, $\vec{R}_h = heal(\{\vec{r} = rel(\texttt{f}_1, \texttt{f}_2) \mid \bar{f}_1, \bar{f}_2 \in \bar{F}\}$ where rel is a function that computes the cf-relations and heal is a function that keeps only healthy cf-relations.

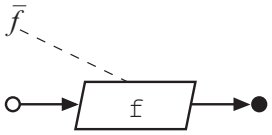


Figure 4.6 – One-activity query workflow

Example 4.3.1 Consider the *dt-functions* $\bar{F} = \{\bar{proj}_a, \bar{proj}_l, \bar{bind}_{a,l}, \bar{retr}_a\}$ derived from the hybrid query *Where are the friends of Alice ?* in the Example 3.2.1. The $ss\text{-}graph$ that represents ways to generate the search space of query workflows implementing the hybrid query is given by the set of one-activity query workflows

2. The set of healthy cf-relations contains only the cf-relations that can be composed in a consistent data-flow. This notion, the function $heal$, and examples are presented in Appendix A.2.

$QW = \{ \quad (qw(\{\texttt{proj}_a\}, \{\}, \{\texttt{proj}_a\}, \{(\texttt{in}, \texttt{proj}_\texttt{a}), (\texttt{proj}_\texttt{a}, \texttt{out})\}, \texttt{in}, \texttt{out}),\ \bar{proj}_a),$
$(qw(\{\texttt{proj}_l\}, \{\}, \{\texttt{proj}_l\}, \{(\texttt{in}, \texttt{proj}_l), (\texttt{proj}_l, \texttt{out})\}, \texttt{in}, \texttt{out}),\ \bar{proj}_l),$
$(qw(\{\texttt{bind}_{a,l}\}, \{\}, \{\texttt{bind}_{a,l}\}, \{(\texttt{in}, \texttt{bind}_{a,l}), (\texttt{bind}_{a,l}, \texttt{out})\}, \texttt{in}, \texttt{out}),\ \bar{bind}_{a,l}),$
$(qw(\{\texttt{retr}_a\}, \{\}, \{\texttt{retr}_a\}, \{(\texttt{in}, \texttt{retr}_a), (\texttt{retr}_a, \texttt{out})\}, \texttt{in}, \texttt{out}),\ \bar{retr}_a)\}$

and the set of cf-relations among them

$\vec{R}_h = \{ \quad \texttt{retr}_a \blacktriangleright\!\blacktriangleright \texttt{bind}_{a,l},$
$\texttt{proj}_l \,\|\, \texttt{proj}_a,$
$\texttt{bind}_{a,l} \blacktriangleright\!\blacktriangleright \texttt{proj}_a,$
$\texttt{bind}_{a,l} \blacktriangleright\!\blacktriangleright \texttt{proj}_l\}.$

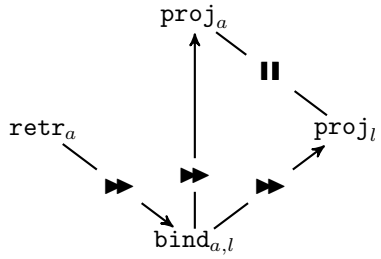


Figure 4.7 – $ss\text{-}graph$ example

Figure 4.7 shows the graphical representation of this $ss\text{-}graph$. For reading purposes, we use arrows for non-commutative cf-relations in order to show the direction of the cf-relation and lines for commutative cf-relations. Nevertheless, the graph structure does not distinguish between commutative/non-commutative cf-relations (*i.e.* edges, arcs) in order to simplify the generation algorithm. This difference is given by the semantics of each cf-relation.

4.3.2 Reduction-based generation algorithm

The generation algorithm is based on a pairwise reduction of the $ss\text{-}graph(QW, \vec{R}_h)$ into a $ss\text{-}graph$ with a single vertex representing a query workflow that implements the hybrid query. At each step of the pairwise reduction, a cf-relation is chosen along with a generation rule that corresponds to the cf-relation (see Figure 4.8).

The query workflows of the cf-relation are composed for generating a new query workflow. Such a composition is done via the $comp\_qw$ function defined below.

Definition 4.3.2 [$comp\_qw$] Given two query workflows $qw_1(A_1, P_1, V_1, E_1, \texttt{in}_1, \texttt{out}_1)$, $qw_2(A_2, P_2, V_2, E_2, \texttt{in}_1, \texttt{out}_2)$, and the query workflow generation rule $qw_{gr}(\{u_1, u_2\}, P_{gr}, V_{gr}, E_{gr}, \texttt{in}_{gr}, \texttt{out}_{gr})$; the composition of qw_1 and qw_2 is given by $comp\_qw(qw_1, qw_2, qw_{gr}) = qw_r(A_r, P_r, V_r, E_r, \texttt{in}_r, \texttt{out}_r)$ defined as follows:

Let be the first and last vertices $First_1, First_2$ and $Last_1, Last_2$ of each query workflow qw_1 and qw_2 such that

$First_1 = \texttt{v}|\texttt{in}_1 \prec^!_{qw_1} \texttt{v}$ and $Last_1 = \texttt{v}|\texttt{v} \prec^!_{qw_1} \texttt{out}_1$;
$First_2 = \texttt{v}|\texttt{in}_2 \prec^!_{qw_2} \texttt{v}$ and $Last_2 = \texttt{v}|\texttt{v} \prec^!_{qw_2} \texttt{out}_2$;

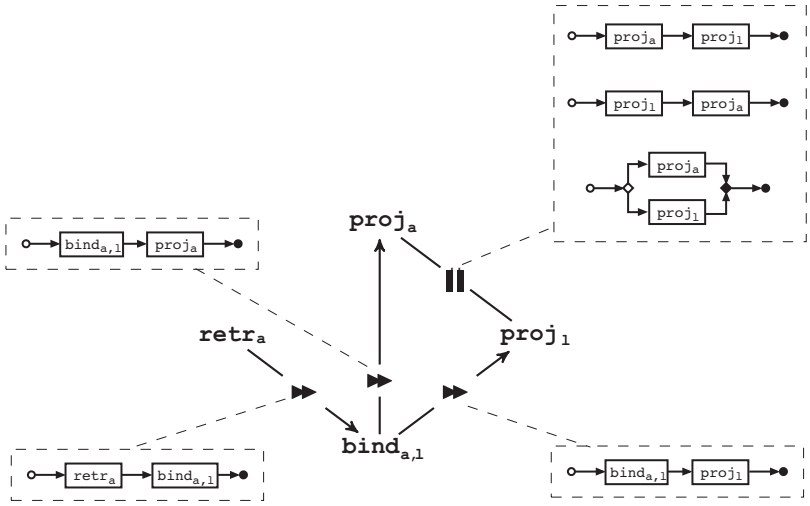


Figure 4.8 – Generation algorithm depiction

where

$$
\begin{aligned}
A_r = \;& A_1 \cup A_2;\\
P_r = \;& P_1 \cup P_2 \cup P_{gr};\\
V_r = \;& A_r \cup P_r \cup \{\mathtt{in}_r, \mathtt{out}_r\};\\
E_r = \;& E_1 \setminus \{(\mathtt{in}_1, First_1), (Last_1, \mathtt{out}_1)\} \cup E_2 \setminus \{(\mathtt{in}_2, First_2), (Last_2, \mathtt{out}_2)\}\\
& \cup \{(u_1^-, First_1) \mid \mathtt{u}_1^- \prec^!_{qw_{gr}} \mathtt{u}_1\} \cup \{(Last_1, u_1^+) \mid \mathtt{u}_1 \prec^!_{qw_{gr}} \mathtt{u}_1^+\}\\
& \cup \{(u_2^-, First_2) \mid \mathtt{u}_2^- \prec^!_{qw_{gr}} \mathtt{u}_2\} \cup \{(Last_2, u_2^+) \mid \mathtt{u}_2 \prec^!_{qw_{gr}} \mathtt{u}_2^+\}.
\end{aligned}
$$

The algorithm in Listings 3.1 gets as input a $ss\text{-}graph$ that describes the cf-relations among the set of one-activity query workflows with its *dt-function*.

Listing 3.1. [Query workflow generation]

Input: $ss\text{-}graph(QW, \vec{R}_h)$
Output: $ss\text{-}graph(\{\langle qw, \bar{f}\rangle\}, \{\})$
1 **For each** $\mathtt{f}_1 \mathfrak{R} \mathtt{f}_2 \in \vec{R}_h$, $qw_1, qw_2 \mid \langle qw_1, \bar{f}_1\rangle, \langle qw_2, \bar{f}_2\rangle \in QW$
2 $qw_{gr} = genrule(\mathtt{f}_1 \mathfrak{R} \mathtt{f}_2)$
3 $qw = comp\_qw(qw_1, qw_2, qw_{gr})$
4 $\bar{f} = merge(\bar{f}_1, \bar{f}_2)$
5 $QW = QW \setminus \{\langle qw_1, \bar{f}_1\rangle, \langle qw_2, \bar{f}_2\rangle\} \cup \{\langle qw, \bar{f}\rangle\}$
6 $\vec{R}_h = rels(\mathtt{f}_1, \mathtt{f}_2, \mathtt{f}, \vec{R}_h)$
7 **End for each**
8 **Return** $ss\text{-}graph(\{\langle qw, \bar{f}\rangle\}, \{\})$

In Line 1, a cf-relation is chosen along with its query workflows. At each iteration the algorithm performs three actions:

1. In Line 2, a generation rule is chosen for a given cf-relation $\mathfrak{R}$.

2. In Line 3, a new query workflow is generated by composing the chosen query workflows qw_1 and qw_2 into a single query workflow qw w.r.t. the generation rule qw_{gr}. This is done by the function $comp\_qw$ that receives qw_1, qw_2 and qw_{gr}.
3. In Line 4, both *dt-functions* $\bar{f}_1$ and $\bar{f}_2$ are merged into $\bar{f}$ that describes the data transformations done by the generated query workflow qw.
4. Finally, in Lines 5 and 6 the $ss\text{-}graph$ is reduced by updating the set of query workflows and by computing the cf-relations –via the function $rels$– with the new `f` that substitutes $\mathtt{f}_1$ and $\mathtt{f}_2$.

In this algorithm the complexity to generate a single query workflow is $O(n)$ in the number of edges $|\vec{R}_h|$. Nevertheless, in order to generate exhaustively the alternative query workflows, the complexity tends to $O(n!)$ for the combinations in which cf-relations are chosen at Line 1. There is also a nested combinatory into the possible query workflow generation rules $genrule(\mathfrak{R})$ for each cf-relation $\mathfrak{R} \in \vec{R}_h$ (*cf.* Line 3). For instance, each independent cf-relation has three possible generation rules.

4.3.3 Generating equivalent query workflows

This section presents examples to show how the algorithm generates equivalent query workflows. We use as input the hybrid query *Which are the common interests of Alice and Bob ?* of Example 3.1.3. First we present a query workflow with parallel compositions and then another query workflow with a completely sequential control-flow. In these examples, the difference relies on which query workflow generation rules are used. More query workflows can be generated by choosing the cf-relations in different order.

Consider the next *dt-functions* $\bar{F}$ derivated from the hybrid query expression of Example 3.1.3:

$$\begin{aligned}\bar{F} = \{&\bar{retr}_a(\{a\}, \{a.nickname = \text{'Alice'}\}, \{atts(a)\}),\\ &\bar{retr}_b(\{b\}, \{b.nickname = \text{'Bob'}\}, \{atts(b)\}),\\ &\bar{proj}_a(\{a\}, \{\}, \{a.nickname\}),\\ &\bar{proj}_b(\{b\}, \{\}, \{b.nickname, b.interests.*\}),\\ &\bar{corr}_{ab}(\{a, b\}, \{a.interests.tag = b.interests.tag\}, \{atts(a), atts(b)\})\}.\end{aligned}$$

and the search space graph $ss\text{-}graph(QW, \vec{R}_h)$ in Figure 4.9

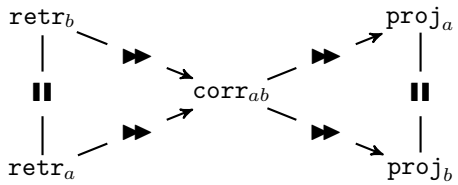


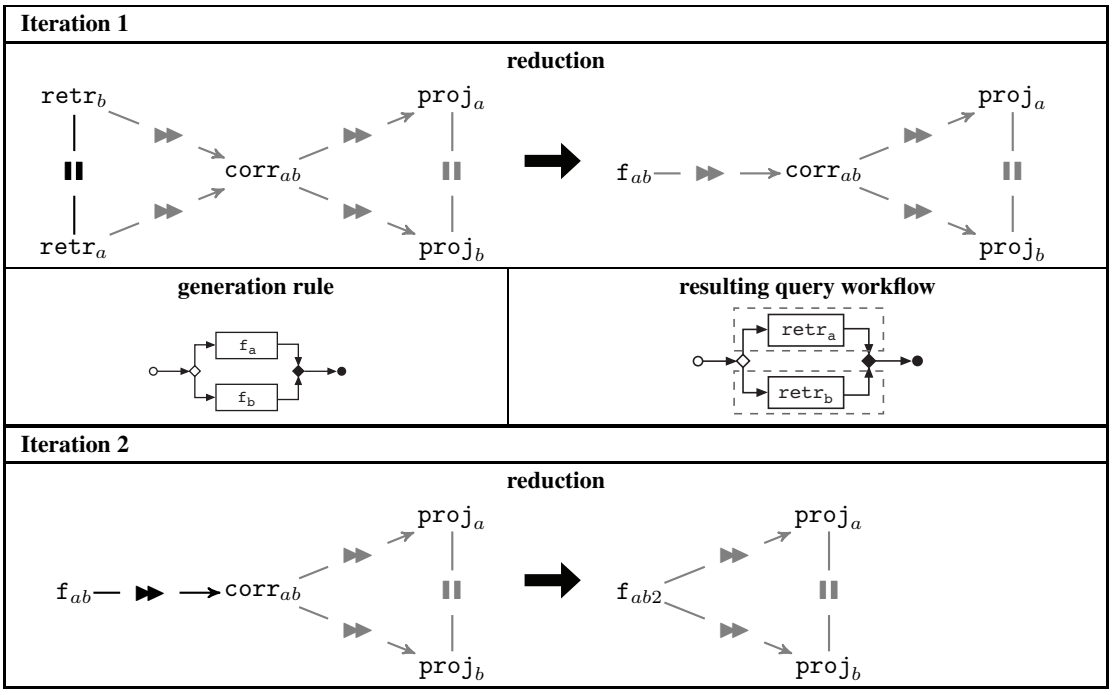
Figure 4.9 – Initial search space graph.

The following examples are illustrated with Tables 4.5 and 4.7 that show at each algorithm iteration the $ss\text{-}graph$ with the chosen cf-relation, the query workflow generation rule to perform the composition, the reduction of the $ss\text{-}graph$, and the resulting query workflow.

Example 4.3.2 Applying the query workflow generation algorithm to the $ss\text{-}graph$ $(QW, \vec{R}_h)$, at each iteration (see Table 4.5), the algorithm choses a cf-relation $\mathfrak{R} \in \vec{R}_h$ and a query workflow generation rule $genrule(\mathfrak{R})$. Such generation rule is used to put together the associated query workflows to each vertex.

1. In the first iteration, the chosen cf-relation is $\texttt{retr}_a \,\|\, \texttt{retr}_b$ and the reduction produces a new dependent cf-relation $\texttt{f}_{ab} \blacktriangleright\!\blacktriangleright \texttt{corr}_{ab}$. The related activities $\texttt{retr}_a$ and $\texttt{retr}_b$ are then composed with a parallel query workflow generation rule that corresponds with the independent cf-relation.
2. In the second iteration, the cf-relation $\texttt{f}_{ab} \blacktriangleright\!\blacktriangleright \texttt{corr}_{ab}$ is chosen and the reduction produces the two new cf-relations $\texttt{f}_{ab2} \blacktriangleright\!\blacktriangleright \texttt{proj}_a$ and $\texttt{f}_{ab2} \blacktriangleright\!\blacktriangleright \texttt{proj}_b$. The sequential query workflow is used to compose in sequence the previous parallel composition of $\texttt{retr}_a$ and $\texttt{retr}_b$ with the activity $\texttt{corr}_{ab}$.
3. In the third iteration, the cf-relation $\texttt{f}_{ab2} \blacktriangleright\!\blacktriangleright \texttt{proj}_a$ is chosen and the reduction produces the final dependent cf-relation $\texttt{f}_{ab3} \blacktriangleright\!\blacktriangleright \texttt{proj}_b$. The dependent query workflow generation rule is used to compose in sequence the new query workflow with $\texttt{proj}_a$.
4. Finally, the reminding cf-relation $\texttt{f}_{ab3} \blacktriangleright\!\blacktriangleright \texttt{proj}_b$ is reduced to $\texttt{f}_{ab4}$ and the sequential query workflow compose the final query workflow with the activity $\texttt{proj}_b$

The resulting query workflow composition depends on the order in which the of cf-relations from $\vec{R}_h$ and query workflow generation rule $genrule(\mathfrak{R})$ are chosen (*cf.* Lines 1,3). Therefore, each permutation of $\{(\mathfrak{R}, genrule(\mathfrak{R})) \mid \mathfrak{R} \in \vec{R}_h\}$ produces a query workflow.



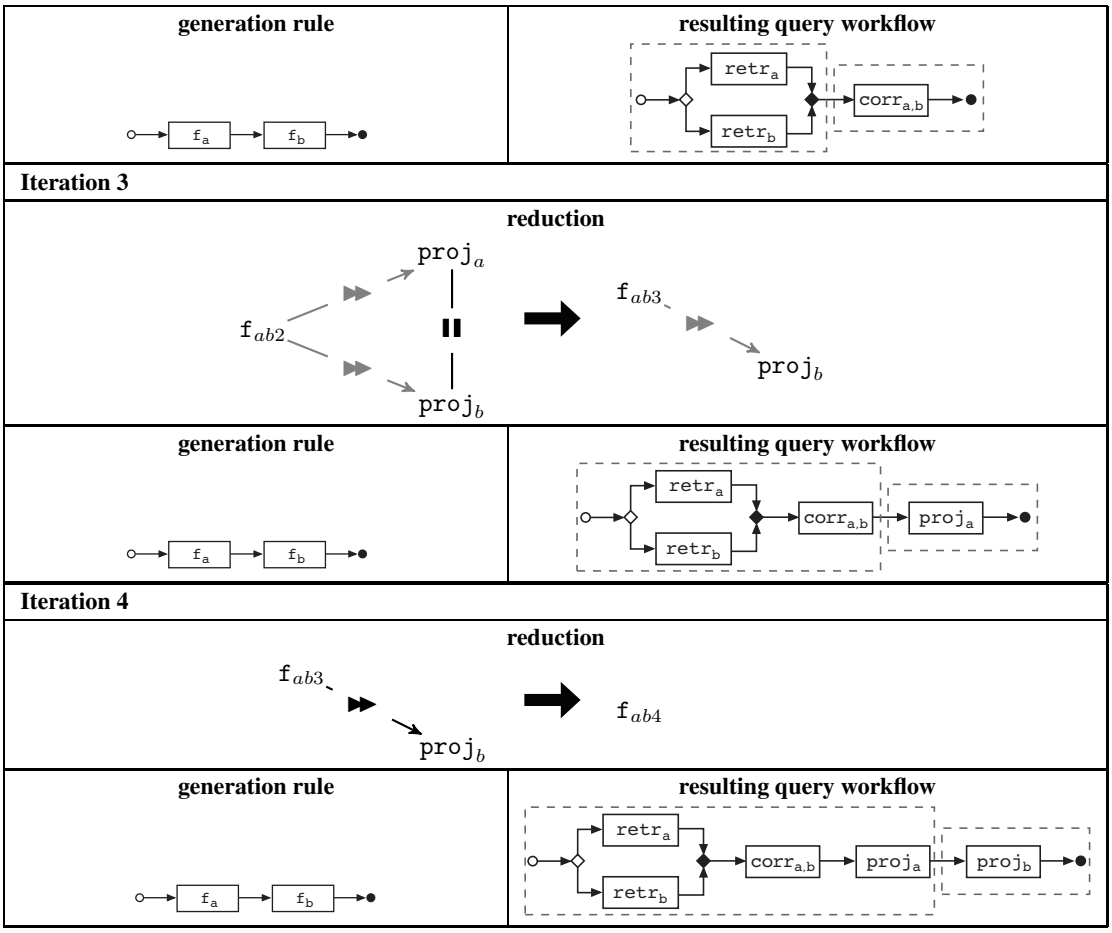


Table 4.5 – Iterations for generating a query workflow with a parallel composition

Example 4.3.3 For this instance the iterations are listed in Table 4.7. The order in which the cf-relations in $\vec{R}_h$ are chosen is the same than the previous example. The difference relies on the query workflow generation rule chosen for the independent cf-relation $\texttt{retr}_a \,\|\, \texttt{retr}_b$. Such choice produces a completely sequential query workflow.

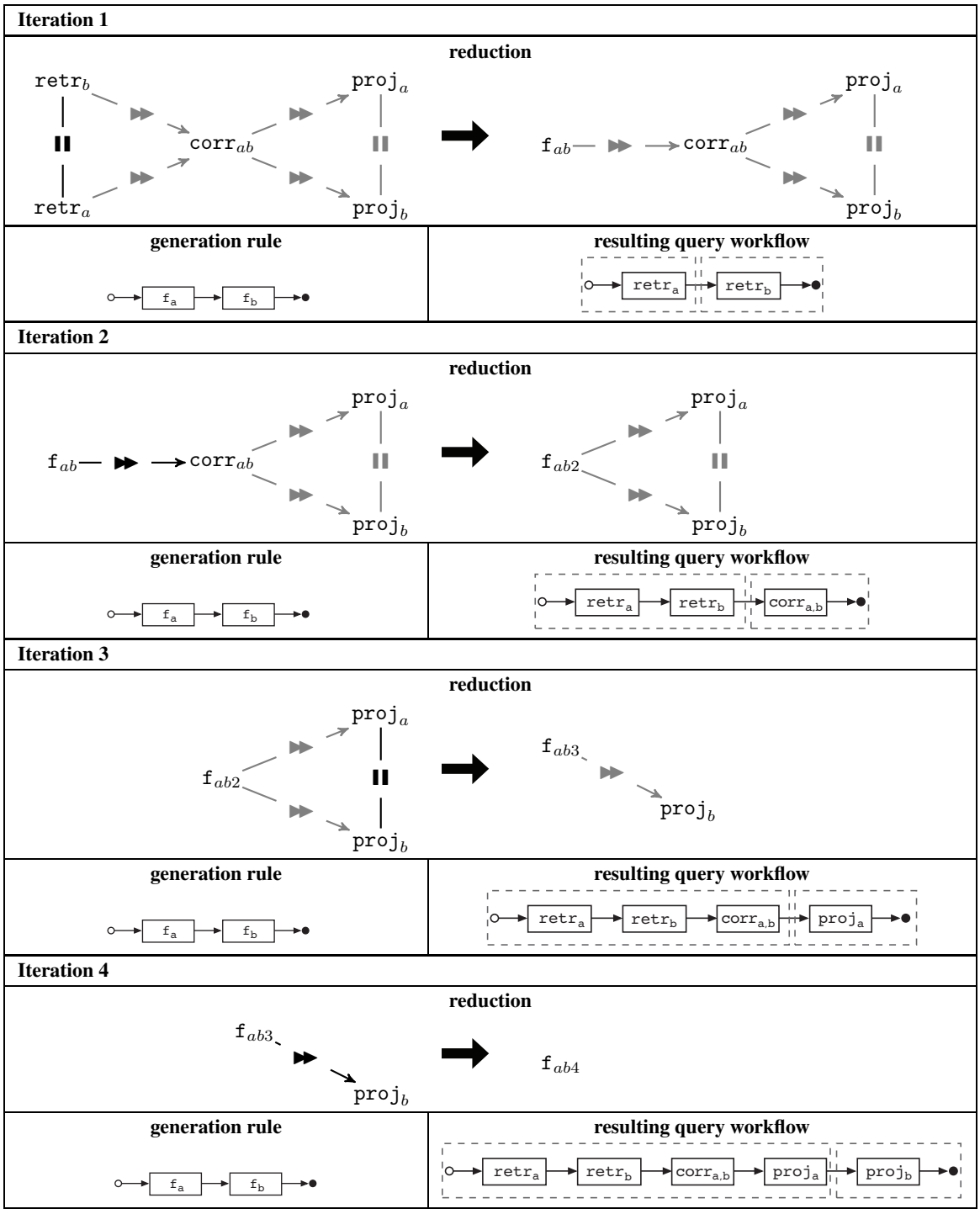


Table 4.7 – Iterations for generating a sequential query workflow

4.4 Conclusions

This chapter presented the generation of the search space of query workflows for a given hybrid query. The generation aims to open the search space in presence of an SLA contract that may not be satisfied if only the data-flow is considered. We therefore reflected on both, the data-flow and control-flow of query workflows by defining a series of generation rules that produces well constructed, consistent, and equivalent query workflows. Given that the application of any generation rule results in a well constructed and consistent query workflow, any chain of generation rules produces a well constructed and consistent query workflow.

The complexity to generate a single query workflow is $O(n)$ in the number of cf-relations. Nevertheless, if the algorithm is exhaustively executed to generate the complete search space, the complexity becomes $O(n!)$. Moreover, there is a nested combinatory in the possible query workflow generation rules for a single cf-relation. Therefore, the processing of the complete search space is unfeasible in the context of just-in-time request/response applications. Although the efficiency of the generation is out of the scope of this work, this algorithm allows to set up configurations in the form of rules. For instance, the spanning tree can be pruned by defining the selection of cf-relations at each algorithm iteration avoiding unnecessary branches, or pulling-up correlation activities.

CHAPTER 5

SLA-based solution space computation

This chapter presents the solution space computation based on an optimization objective derived from an SLA contract. Given an optimization objective specifying the expected values of a combination of cost attributes (*e.g.* time, price, energy) and the preferences among them, the solution space of a given hybrid query consists of the query workflows that are the closest to the optimization objective and that best fulfill the preferences.

As shown in Figure 5.1, given an optimization objective of an SLA contract and the search space of query workflows, first, we estimate the query workflow cost given a series of cost attributes pertinent to the SLA contract. The resulting costs are weighted according to the preferences among cost attributes expressed in the SLA contract. Then, we compute the distance of the query workflows from the optimization objective. Finally, we apply a top-k algorithm for selecting the k closest query workflows from the optimization objective which compose the solution space.

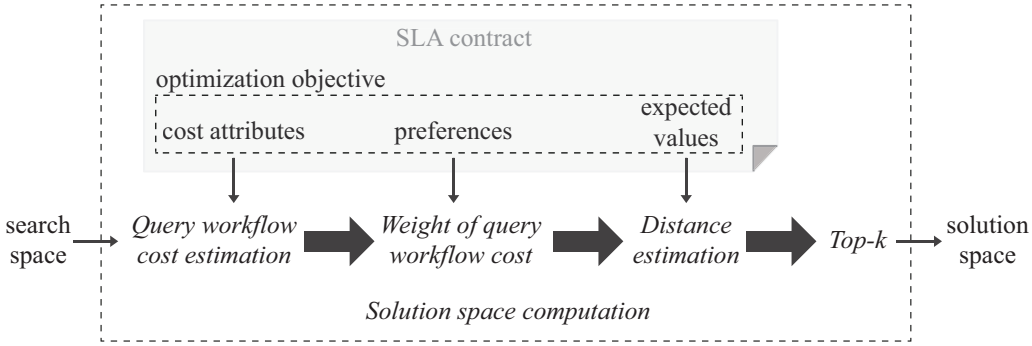


Figure 5.1 – Cost estimation and solution space computation overview

The remainder of the chapter is organized as follows. Section 5.1 defines the estimation of the query workflow cost including its constituent activities costs. Section 5.2 presents top-k^{qw}, an adaptation of a top-k algorithm by means of a weighted distance from the optimization objective. Finally, Section 5.3 concludes this chapter.

5.1 Query workflow cost estimation

The estimation of the query workflow cost involves the cost of its constituent activities (see Figure 5.2). The cost of both activities and the query workflow is represented by an m dimensional vector of the form $\langle x^1, ..., x^m \rangle$ where each x^j represents an attribute-cost pair, *cf.* Def. 3.3.1.

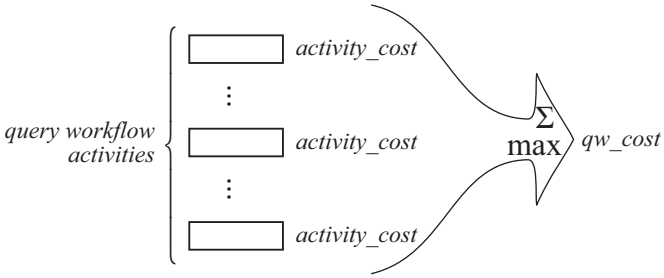


Figure 5.2 – Query workflow cost estimation overview

The query workflow cost is computed by aggregating the activities' costs according to the control-flow among them. Next, we define the activity cost and the query workflow cost.

5.1.1 Activity cost

The activity cost depends on the way the activity interacts with (1) the invoked service, (2) the preceding activities from which the activity receives data, and (3) the succeeding activities to which the activity delivers data (see Figure 5.3).

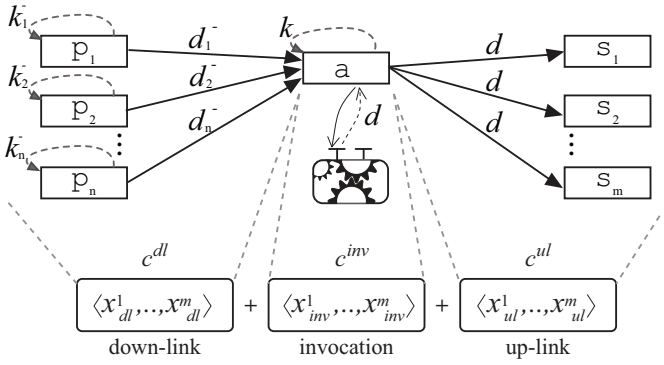


Figure 5.3 – Activity cost addends

Each interaction brings a cost represented by a vector $\langle x^1, ..., x^m \rangle$.

— The down-link vector c^{dl} represents the costs associated to the communication with preceding activities.

— The invocation vector c^{inv} represents the costs associated to the invocation of the service.

— The up-link vector c^{ul} represents the costs associated to the communication with succeeding activities.

Each cost attribute x^j of a cost vector is given by a specific cost function that combines QoS measures of service instances and network. In this work we consider the cost attributes for time, price, and energy, *i.e.* $\langle x^t, x^p, x^e \rangle$.

Definition 5.1.1 [activity cost] For a given activity $\mathtt{a}$ belonging to a query workflow $qw(A, P, E, \mathtt{in}, \mathtt{out})$, and its preceding activities $prec = \{\mathtt{p} \in A \mid \mathtt{p} \prec^{!}_{qw} \mathtt{a}\}$, and its succeeding activities $succ = \{\mathtt{s} \in A \mid \mathtt{a} \prec^{!}_{qw} \mathtt{s}\}$. The activity cost function is defined as follows.

$$activity\_cost(\mathtt{a}) = \sum_{i=1}^{|prec|} \sum_{j=1}^{k_i^-} d_{i,j}^- \cdot c^{dl} + k \cdot c^{inv} + |succ| \cdot \sum_{l=1}^{k} d_l \cdot c^{ul} \tag{5.1}$$

The activity $\mathtt{a}$ (1) receives $d_{i,j}^-$ amount of data from each preceding activity $i \in [1 \,..\, |prec|]$ at each data delivery $j \in \left[1 \,..\, k_i^-\right]$, (2) receives d_l amount of data from the invoked service at each data delivery $l \in [1 \,..\, k]$, and (3) delivers d_l amount of data to each succeeding activity in $succ$.

(a) batch delivery (b) stream delivery

Figure 5.4 – Data received from invoked service

Example 5.1.1 In order to exemplify the activity cost function, we consider the way an activity receives data from the invoked service, from the preceding activities; and the way it delivers data to the succeeding activities. The data delivery can be either in batch or stream (see Figure 5.4) bringing the following four cases.

— An activity $\mathtt{a}$ receives data in **batch** from both the preceding activities and the invoked service. Therefore, it delivers the resulting data in **batch**.

$$activity\_cost(\mathtt{a}) = \sum_{i=1}^{|prec|} d_i^- \cdot c^{dl} + c^{inv} + |succ| \cdot d \cdot c^{ul}$$

The down-link addend c^{dl} is multiplied by the summation of the data size d_i^- of each preceding activity $i \in [1 \,..\, |prec|]$. The invocation addend c^{inv} is not affected because $k = 1$. The activity delivers its result as many succeeding activities it has, thus the up-link addend c^{ul} is multiplied by the data size d and the number of succeeding activities $|succ|$.

— An activity $\mathtt{a}$ receives data in **batch** from all its preceding activities and in **stream** from the invoked service. Therefore, it delivers the resulting data in **stream**.

$$activity\_cost(\mathtt{a}) = \sum_{i=1}^{|prec|} d_i^- \cdot c^{dl} + k \cdot c^{inv} + |succ| \cdot \sum_{l=1}^{k} d_l \cdot c^{ul}$$

The down-link addend c^{dl} is multiplied by the summation of the data size d_i^- of each preceding activity $i \in [1 \,..\, |prec|]$. The invocation addend c^{inv} is multiplied by k times the service operation delivers data. The activity delivers its result as many succeeding activities c^{ul} it has, thus the up-link addend is multiplied by the data size produced at each execution $l \in [1 \,..\, k]$.

— An activity a receives data in **stream** from at least one of its preceding activities and in **batch** from the invoked service. Therefore, it delivers the resulting data in **stream**.

$$activity\_cost(\mathtt{a}) = \sum_{i=1}^{|prec|} \sum_{l=1}^{k_i^-} d_{i,l}^- \cdot c^{dl} + k \cdot c^{inv} + |succ| \cdot \sum_{j=1}^{k} d_j \cdot c^{ul}$$

The down-link addend c^{dl} is multiplied by the nested summation of the data sizes $d_{i,j}^-$ for all $i \in [1 \,..\, |prec|]$ and for all $j \in \left[1 \,..\, k_i^-\right]$. The up-link addend c^{ul} is multiplied by the data size produced at each execution $l \in [1 \,..\, k]$. In this case $k > 1$ affects the invocation addend c^{inv}. Given that the service produces data in batch, it is assumed that the invocation is performed as many times as the maximum k_i^-. Thus $k = \max(k_1^-, ..., k_{|prec|}^-)$.

— An activity a receives data in **stream** from at least one of its preceding activities and in **stream** from the invoced service. Therefore, the activity delivers the resulting data in **stream**.

$$activity\_cost(\mathtt{a}) = \sum_{i=1}^{|prec|} \sum_{l=1}^{k_i^-} d_{i,l}^- \cdot c^{dl} + k \cdot c^{inv} + |succ| \cdot \sum_{j=1}^{k} d_j \cdot c^{ul}$$

The down-link addend c^{dl} is multiplied by the nested summation of the data sizes $d_{i,j}^-$ for all $i \in [1 \,..\, |prec|]$ and for all $j \in \left[1 \,..\, k_i^-\right]$. The invocation addend c^{inv} is multiplied by as many k times the service delivers data. The up-link addend c^{ul} is multiplied by the data size produced at each execution in $l \in [1 \,..\, k]$.

Now we turn our attention to the computation of the query workflow cost that results from the aggregation of the activities costs.

5.1.2 Query workflow cost

The query workflow cost is computed following the aggregation schema proposed in [JRGM04, Car04]. The principle is to reduce the query workflow graph $qw(A, P, E, \mathtt{in}, \mathtt{out})$ by aggregating the activities costs according to the control-flow among them, *i.e.* sequential, parallel.

In this work we consider the cost attributes (1) execution time, (2) execution price, and (3) energy consumption; *i.e.* $activity\_cost(\mathtt{a}) = \langle x_{\mathtt{a}}^t, x_{\mathtt{a}}^p, x_{\mathtt{a}}^e \rangle$. For aggregating these cost attributes, either the

summation or the maximum is applied depending on the cost attribute and the control-flow [1]. Table 5.1 shows the aggregation rule for each pair of cost attribute and control-flow in $\{t, p, e\} \times \{seq, par\}$.

	seq	par
(t) Execution time	$x^t_{qw} = \sum_{\mathtt{a} \in A} x^t_{\mathtt{a}}$	$x^t_{qw} = \max_{\mathtt{a} \in A}(x^t_{\mathtt{a}})$
(p) Execution price	$x^p_{qw} = \sum_{\mathtt{a} \in A} x^p_{\mathtt{a}}$	$x^p_{qw} = \sum_{\mathtt{a} \in A} x^p_{\mathtt{a}}$
(e) Energy consumption	$x^e_{qw} = \sum_{\mathtt{a} \in A} x^e_{\mathtt{a}}$	$x^e_{qw} = \sum_{\mathtt{a} \in A} x^e_{\mathtt{a}}$

Table 5.1 – Aggregation rules for cost attributes $\times$ control-flow

These aggregation rules are applied via the $aggseq$ and $aggpar$ functions defined below.

Definition 5.1.2 [$aggseq$] Given a set of activities' cost vectors $C_{seq} = \{\langle x^t_{\mathtt{a}}, x^p_{\mathtt{a}}, x^e_{\mathtt{a}}\rangle = activity\_cost(\mathtt{a}) \mid \mathtt{a} \in A\}$ whose activities A belong to a query workflow $qw(A, P, E, \mathtt{in}, \mathtt{out})$ with a single sequential control-flow, *i.e.* $|epaths(\mathtt{in}, \mathtt{out}, E)| = 1$. The function $aggseq(C_{seq}) = \langle x^t_{qw}, x^p_{qw}, x^e_{qw}\rangle$ aggregates the cost vectors in C_{seq} such that each cost attribute is aggregated according to its corresponding rule in $\{t, p, e\} \times \{seq\}$ of Table 5.1.

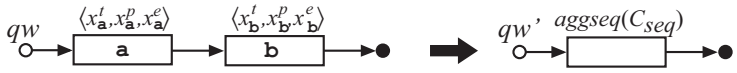


Figure 5.5 – Sequential aggregation

Example 5.1.2 Consider the query workflow in Figure 5.5 where the activities a, b are composed in sequence. The set of costs associated to the activities of qw is given by $C_{seq} = \{\langle x^t_{\mathtt{a}}, x^p_{\mathtt{a}}, x^e_{\mathtt{a}}\rangle, \langle x^t_{\mathtt{b}}, x^p_{\mathtt{b}}, x^e_{\mathtt{b}}\rangle\}$. Therefore, the sequential aggregation of the activities costs is given by

$$aggsec(C_{seq}) = \langle \sum(x^t_{\mathtt{a}}, x^t_{\mathtt{b}}), \sum(x^p_{\mathtt{a}}, x^p_{\mathtt{b}}), \sum(x^e_{\mathtt{a}}, x^e_{\mathtt{b}})\rangle.$$

Definition 5.1.3 [$aggpar$] Given a set of activities cost vectors $C_{par} = \{\langle x^t_{\mathtt{a}}, x^p_{\mathtt{a}}, x^e_{\mathtt{a}}\rangle = activity\_cost(\mathtt{a}) \mid \mathtt{a} \in A\}$ whose activities in A belong to a query workflow $qw(A, P, E, \mathtt{in}, \mathtt{out})$ with a single parallel control-flow, *i.e.* $\{(\mathtt{in}, \mathtt{par}^l), (\mathtt{end\_par}^l, \mathtt{out})\} \subset E \wedge |A| = 2$. The function $aggpar(C_{par}) = \langle x^t_{qw}, x^p_{qw}, x^e_{qw}\rangle$ aggregates the cost vectors in C_{par} such that each cost attribute is aggregated according to its corresponding rule in $\{t, p, e\} \times \{par\}$ of Table 5.1.

1. Other aggregations should be considered for other cost attributes and control-flows, *e.g.* [Car04].

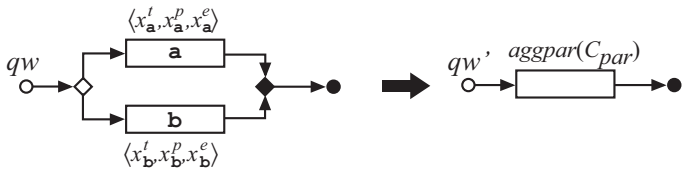


Figure 5.6 – Parallel aggregation

Example 5.1.3 Consider the query workflow in Figure 5.6 where the activities a, b are composed in parallel. The set of costs associated to the activities of qw is given by $C_{par} = \{\langle x^t_{\mathtt{a}}, x^p_{\mathtt{a}}, x^e_{\mathtt{a}}\rangle, \langle x^t_{\mathtt{b}}, x^p_{\mathtt{b}}, x^e_{\mathtt{b}}\rangle\}$. Therefore, the parallel aggregation of the activities costs is given by

$$aggpar(C_{par}) = \langle max(x^t_{\mathtt{a}}, x^t_{\mathtt{b}}), \sum(x^p_{\mathtt{a}}, x^p_{\mathtt{b}}), \sum(x^e_{\mathtt{a}}, x^e_{\mathtt{b}})\rangle.$$

Given both functions $aggsec$ and $aggpar$, the function qw_cost is defined as follows.

Definition 5.1.4 [qw_cost] Given a query workflow graph $qw(A, P, V, E, \mathtt{in}, \mathtt{out})$, the function $qw\_cost(qw) = \langle x^t_{qw}, x^p_{qw}, x^e_{qw}\rangle$ aggregates the costs of the activities in A by traversing the control-flow defined by E from in to out following the rules below. For simply, $qw\_cost(qw)$ is rewritten as $qw\_cost(\mathtt{v}, E)$ where $\mathtt{v} \in V$.

1. If $\mathtt{v} = \mathtt{out}$, then $qw\_cost(\mathtt{out}, E) = \langle 0, 0, 0\rangle$

2. If $\mathtt{v} = \mathtt{in}$ and s is the successor vertex in such that $(\mathtt{in}, \mathtt{s}) \in E$,
   then $qw\_cost(\mathtt{in}, E) = qw\_cost(\mathtt{s}, E \setminus \{(\mathtt{in}, \mathtt{s})\})$

3. If $\mathtt{v} = \mathtt{par}^l$ and C_{par} is the set of costs of each path from $\mathtt{par}^l$ to $\mathtt{end\_par}^l$ such that
   $$C_{par} = \Big\{ qw\_cost(\mathtt{f},\ \{(\mathtt{in}, \mathtt{f}), (\mathtt{l}, \mathtt{out})\} \cup epaths(\mathtt{f}, \mathtt{l}, E)) \mid\ (\mathtt{par}^l, \mathtt{f}) \in E,\ (\mathtt{l}, \mathtt{end\_par}^l) \in E \Big\}$$
   and s is the successor vertex of $\mathtt{end\_par}^l$ such that $(\mathtt{end\_par}^l, \mathtt{s}) \in E$,
   then $qw\_cost(\mathtt{in}, E) = aggseq\big(\ \{aggpar(C_{par}), qw\_cost(\mathtt{s}, E \setminus \{(\mathtt{end\_par}^l, \mathtt{s})\})\}\ \big)$.

4. Otherwise, if s is the successor vertex of v such that $(\mathtt{v}, \mathtt{s}) \in E$,
   then $qw\_cost(\mathtt{v}, E) = aggseq\big(\ \{activity\_cost(\mathtt{v}), qw\_cost(\mathtt{s}, E \setminus \{(\mathtt{v}, \mathtt{s})\})\}\ \big)$.

Rule 1 is the base case when the vertex out is reached. Rule 2 is the initial case where the traversing of a query workflow starts from in. Rule 3 computes the cost of a query workflow with a parallel composition from $\mathtt{par}^l$ to $\mathtt{end\_par}^l$ and aggregates it with the cost of the reminding control-flow after $\mathtt{end\_par}^l$. Finally, Rule 4 aggregates the cost of an activity with the reminding control-flow from the next activity.

Example 5.1.4 For tracing the aggregation of the query workflow cost, consider the query workflow in Figure 5.7. Suppose each activity $\mathtt{a} \in A$ has the synthetic cost $activity\_cost(\mathtt{a}) = \langle 1, 1, 1\rangle$.

The aggregation starts by applying the Rule 2 to the `in` vertex of qw in a). The next vertex to `in` is an activity and thus, the Rule 4 is applied. Then it follows a parallel composition whose branches are branched into the query workflows qw'_1 and qw'_2 by the Rule 3 in b). The Rule 2 is applied to both qw'_1 and qw'_2 and the Rule 4 aggregates their sequential compositions. This leads to the cost of $\langle 2, 2, 2\rangle$ for qw'_1 and $\langle 1, 1, 1\rangle$ for qw'_2 in c). By back-tracing, we get to the parallel composition –previously reached in b) by Rule 3– which is now aggregated to get the cost $\langle 2, 3, 3\rangle$. In d), the costs of activities in sequence are aggregated by the Rule 4. Finally, the `out` is reached to finally get the cost of qw $\langle 4, 5, 5\rangle$ in e).

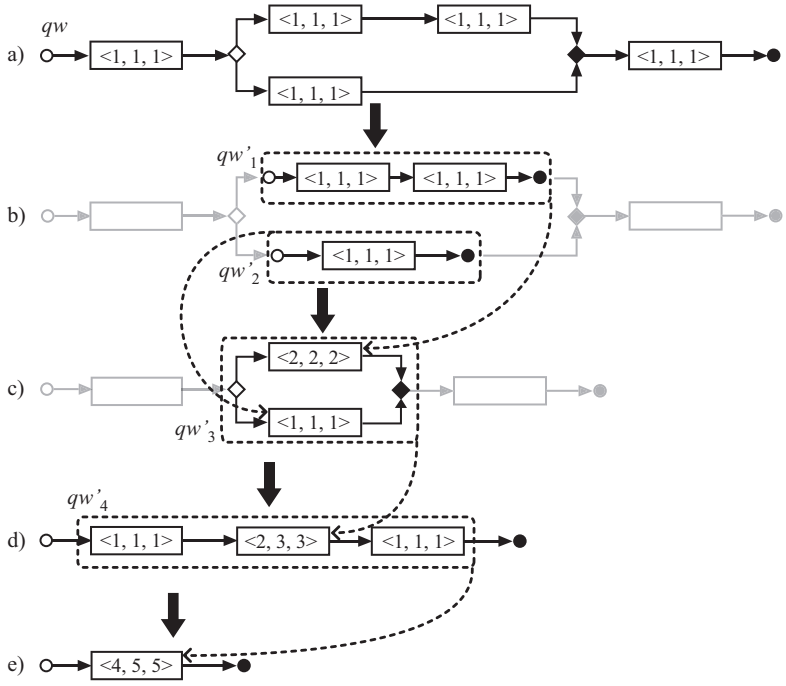


Figure 5.7 – Application of the qw_cost rules

The set of equivalent query workflows implementing a hybrid query and their corresponding costs form the *search space of tagged query workflows* defined as follows.

Definition 5.1.5 [search space of tagged query workflows] Given a set of equivalent query workflows QW, the search space of tagged query workflows $\mathcal{S}$ is given by the set QW where each query workflow is tagged with its corresponding cost via qw_cost. This is,

$$\mathcal{S} = \{qw_i\langle x_i^1, \ldots, x_i^m\rangle \mid qw_i \in QW, \langle x_i^1, \ldots, x_i^m\rangle = qw\_cost(qw_i)\}.$$

The absence of data-related parameters in service-based environments at build-time brings constraints to have an accurate query workflow cost. However, it is possible to have an approximation to the cost by means of a relaxed formulation of the $activity\_cost$ function.

5.1.3 Cost estimation at build-time

During the optimization process, both the size of data d transformed by an activity and the number of times k the invoked service delivers data are not available due to the absence of data-related parameters in service-based environments. We propose a build-time formulation of the activity cost function $activity\_cost$ that is a relaxation of the formulation in Equation 5.1 (See Figure 5.8).

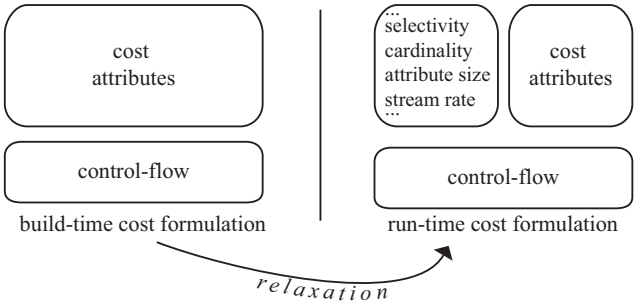


Figure 5.8 – Cost estimation for query workflows

The build-time formulation captures the interactions of the activity with its adjacent activities and the invoked service; and dispenses with data-related parameters. Hereafter, we refer to the activity cost formulation in Equation 5.1 as F_{rt} and to the build-time formulation as F_{bt}.

Property 5.1.1 [relaxed formulation] If a and b are two different activities belonging to two equivalent query workflows qw_1 and qw_2; and they hold a total order [2] by F_{rt} such that $F_{rt}(\mathtt{a}) > F_{rt}(\mathtt{b})$; F_{bt} is a relaxed formulation of F_{rt} iff a and b hold a partial order by F_{bt} such that $F_{bt}(\mathtt{a}) \geqslant F_{bt}(\mathtt{b})$.

Observation 5.1.1 Observe from the F_{rt} formulation in Equation 5.1 that every variable has its domain either in non-negative integers $\mathbb{Z}_+$, or non-negative reals $\mathbb{R}_+$.

$$d_i^- \in \mathbb{R}_+ \ \forall i \in [1 \,..\, |prec|] \qquad d \in \mathbb{R}_+$$
$$k_i^- \in \mathbb{Z}_+ \ \forall i \in [1 \,..\, |prec|] \qquad k \in \mathbb{Z}_+$$

Proposition 5.1.1 [build-time cost] The formulation F_{bt} is an relaxed formulation of F_{rt} such that

$$F_{bt} = |prec| \cdot c^{dl} + c^{inv} + |succ| \cdot c^{ul}. \tag{5.2}$$

Proof 5.1.1 Consider again two activities a and b belonging to two equivalent query workflows qw_1 and qw_2. By the Observation 1, every variable and summation in F_{rt} (*cf.* Eq. 5.1) are always non-negative scalars, and therefore F_{bt} holds the partial order between both activities, *i.e.* $F_{bt}(\mathtt{a}) \geqslant F_{bt}(\mathtt{b})$, *cf.* Def. 5.1.1. ■

As we are interested in the optimization of the control-flow of activities, both the run-time and build-time cost formulations entail the data transformations through the control-flow instead to the classical data-flow. In the particular case of the build-time formulation, the data related-parameters are evicted and

2. As costs are represented by vectors, some of them are not comparable. Therefore for this reflection we suppose the cost are scalars.

thus the formulation only considers the control-flow where the activity participates. Therefore, the cost estimation does not consider the semantics of data operators like in data-flow oriented optimizers, and the build-time formulation may not report interesting query workflows (false negatives), or may report non-interesting query workflows (false positives).

5.2 Computing the solution space

The solution space is computed by means of a top-k algorithm adapted to the hybrid query optimization context in order to have a single search strategy for any combination of cost attributes defined by an optimization objective. Top-k algorithms process items characterized by multiple features (*i.e.* fuzzy data), and select the k 'best' items.

Our algorithm top-k^{qw} is an adaptation of the Fagin's algorithm (FA) [FLN03]. It looks for the solution space $\mathcal{S}^k \subset \mathcal{S}$ whose k resulting query workflows represent the best ones into the search space $\mathcal{S}$. The notion of 'best' is given by the weighted distance of query workflows from the optimization objective.

5.2.1 Weighted distance from the optimization objective

We adopt a weighted distance for enabling the comparison of the costs of query workflows with a given optimization objective. This metric allows to (1) privilege those query workflows that best conform the cost attribute preferences and the expected values given by the optimization objective, and (2) evict query workflows that, despite having the same Euclidean distance, they do not represent interesting results.

The weighted distance has two functions:

— ***weight*** function moves the query workflow cost by applying penalties to its cost attributes. The direction in which a penalty is applied over an attribute x depends on its type $atype(x) = \{loss, profit\}$. A $loss$ attribute represents a best benefit when its value is smaller, *e.g.* execution price, execution time [SMWM06, LNZ04]. Therefore, the penalty is applied by increasing the cost attribute value. A $profit$ attribute represents a best benefit when its value is higher, *e.g.* reliability, throughput, reputation [SJ98, Car04, LNZ04, WCSO08]. Therefore, the penalty is applied by reducing the cost attribute value.

Definition 5.2.1 [$weight$] Given a query workflow cost $qw_i\langle x_i^1, \ldots, x_i^m\rangle$ and an optimization objective $oo\langle(o_1, w_1), .., (o_m, w_m)\rangle$, the $weight$ function moves the j^{th} cost attribute as follows:

$$weight(x_i^j, oo) = \begin{cases} \text{if } atype(x_i^j) = profit; & val(x_i^j) - |val(v^j) - val(x_i^j)| \cdot \dfrac{w^j}{\sum_{k=1}^{m} w^k} \\ \text{if } atype(x_i^j) = loss; & val(x_i^j) + |val(v^j) - val(x_i^j)| \cdot \dfrac{w^j}{\sum_{k=1}^{m} w^k} \end{cases}$$

— ***dist*** function gets the Euclidean distance of a weighted query workflow cost from the optimization objective. The formulation of the $dist$ function is defined as follows.

Definition 5.2.2 [$dist$] Given query workflow $qw_i\langle x_i^1, \ldots, x_i^m\rangle$ and an optimization objective $oo\langle(o_1, w_1), .., (o_1, w_m)\rangle$, the distance is given by the Euclidean distance between the weighted cost of qw_i and oo

$$dist(qw_i, oo) = \Big(\sum_{j=1}^{m}\Big[weight(x_i^j, oo) - val(o^j)\Big]^2\Big)^{1/2}.$$

Example 5.2.1 Consider the $oo\langle(time = 40, 2), (price = 38, 1)\rangle$ in Figure 5.9 as the center of a search space described by a circular convex hull of query workflow costs $QW = \{qw_1, \ldots, qw_{12}\}$.

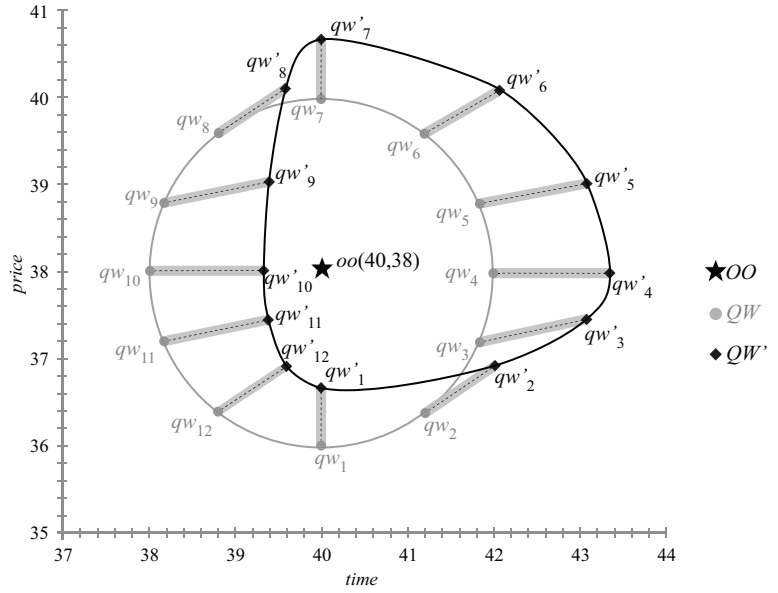


Figure 5.9 – Search space before and after applying $weight$ function

Each query workflow in QW has the same Euclidean distance from oo, *i.e.* 2. By applying the $weight$ function, each query workflow is moved either close to or away from oo obtaining $QW' = \{qw_i' = \langle weight(x_i^1), weight(x_i^2))\rangle \mid qw_i = \langle x_i^1, x_i^2\rangle \in QW$ (*cf.* Table 5.2).

Making a pairwise comparison, the query workflows $qw_1, qw_{12}, qw_{11}, qw_{10}$ form the Pareto optimal curve. Nevertheless, looking at the distances from the optimization objective in the chart of Figure 5.10, all the query workflows in $qw_1, ..., qw_{12}$ are equally good. In contrast, the weighted distance enables the comparison among scattered and equidistant query workflows. Those query workflows that do not represent interesting solutions are evicted (*e.g.* qw_4, qw_5, qw_6), and the ones representing good solutions are privileged (*e.g.* $qw_1, qw_{12}, qw_{11}, qw_{10}$). For instance, without a pairwise comparison, qw_{10} represents the best alternative.

Once the weighted distance of query workflows is obtained, the search space can be ordered for obtaining the query workflows that best conform the SLA contract are found on the top (see Table 5.3).

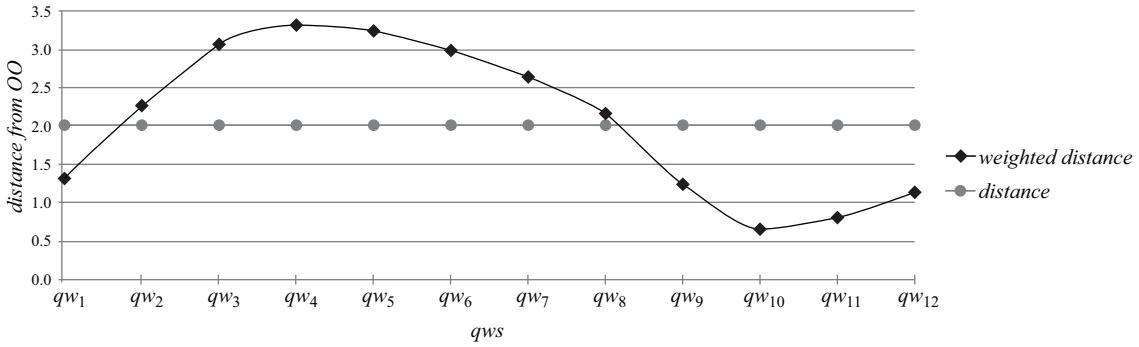


Figure 5.10 – Euclidean and Weighted distances

query workflows	price	time	distance	price'	time'	distance'
qw_1	40.0	36.0	2.0	40.0	36.667	1.3333
qw_2	41.2	36.4	2.0	42.0	36.933	2.2667
qw_3	41.833	37.2	2.0	43.055	37.467	3.1013
qw_4	42.0	38.0	2.0	43.333	38.0	3.3333
qw_5	41.833	38.8	2.0	43.055	39.067	3.2359
qw_6	41.2	39.6	2.0	42.0	40.133	2.9242
qw_7	40.0	40.0	2.0	40.0	40.667	2.6667
qw_8	38.8	39.6	2.0	39.6	40.133	2.1705
qw_9	38.167	38.8	2.0	39.389	39.067	1.2293
qw_{10}	38.0	38.0	2.0	39.333	38.0	0.6667
qw_{11}	38.167	37.2	2.0	39.389	37.467	0.811
qw_{12}	38.8	36.4	2.0	39.6	36.933	1.1392

Table 5.2 – Original and weighted cost attributes

query workflows	price'	time'	distance'↓
qw_{10}	39.333	38.0	0.6667
qw_{11}	39.389	37.467	0.811
qw_{12}	39.6	36.933	1.1392
...			

Table 5.3 – Naive top-k example

By traversing the complete search space, it is ensured that the k results are global optimal, *i.e.* there are no nearest query workflows from the optimization objective in $\mathcal{S} \setminus \mathcal{S}^k$. Nevertheless, when the search space is large it is desirable to avoid the processing of the complete search space. Thus, we adapt the Fagin's Algorithm (FA) [FLN03] that is reported to be optimal [Fag99] when the scoring function (*e.g.* $dist$) is monotone and strict.

5.2.2 The top-k^{qw} algorithm

The top-k^{qw} algorithm in Listings 2.1 assumes m inverted lists $L_1, \ldots, L_m$ each associated to the $[1..m]$ cost attributes of the optimization objective $oo\langle(o_1, w_1), .., (o_1, w_m))\rangle$. Each list L_j is of the form $[\ldots, weight(x_i^j, oo), \ldots]$ with size $|\mathcal{S}|$ for $i \in [1 \,..\, |\mathcal{S}|]$, and $j \in [1 \,..\, m]$.

Each of the m lists is ordered depending if it is associated to a loss or profit attribute. If an arbitrary attribute x^j is a loss attribute $atype(x^j) = loss$, then $L_j = sortAsc(L_j)$. If x^j is a profit attribute $atype(x^j) = profit$, then $L_j = sortDesc(L_j)$. The principle is that the top-k items are close to the top of the m lists.

Listing 2.1. [top-k^{qw} algorithm]

Input: $L_1, ..., L_m$, $oo\langle(o_1, w_1), .., (o_1, w_m))\rangle$, k
Output: $\mathcal{S}^k$
1 $SEEN_1, ..., SEEN_m = []$
2 $j = 1$
3 **While** $|\bigcap_{l=1}^{m} D_l| < k$
4 **Parallel**
5 $SEEN_1 = append(SEEN_1, L_1[j])$
6 ...
7 $SEEN_m = append(SEEN_m, L_m[j])$
8 **End parallel**
9 $j = j + 1$
10 **End while**
11 $DISTS = []$
12 **For each** $qw_j \in \bigcup_{l=1}^{m} SEEN_l$
13 $qw_j = \langle SEEN_1[j], ..., SEEN_m[j]\rangle$
14 $DISTS = append(DISTS, dist(qw_j, oo))$
15 **End for each**
16 $DISTS = sortAsc(DISTS)$
17 $\mathcal{S}^k = DISTS[1..k]$
18 **Return** $\mathcal{S}^k$

Top-k^{qw} traverses simultaneously the m lists by performing sequential access, *cf.* Lines 3-10. When the same k items have been seen in the m lists, it performs random access over the already seen items and computes their distances (*cf.* Lines 12-15). The distances are sorted in ascending order (*cf.* Line 16) and thus the closest k items from the optimization objective are on the $1..k$ positions of the distances list (*cf.* Line 17).

qw	$L_{time'} \downarrow$
qw_{10}	39.333
qw_9	39.389
qw_{11}	39.389
qw_8	39.600
qw_{12}	39.600
qw_1	40.000
qw_7	40.000
qw_2	42.000
qw_6	42.000
qw_3	43.055
qw_5	43.055
qw_4	43.333

a)

qw	$L_{price'} \downarrow$
qw_1	36.667
qw_2	36.933
qw_{12}	36.933
qw_3	37.467
qw_{11}	37.467
qw_4	38.000
qw_{10}	38.000
qw_5	39.067
qw_9	39.067
qw_6	40.133
qw_8	40.133
qw_7	40.667

b)

Table 5.4 – Input lists used by top-k^{qw}

Example 5.2.2 Consider the search space in Table 5.2, the optimization objective $oo\langle(time = 40, 2), (price = 48, 1)\rangle$, and $k = 1$. The cost of each query workflow is split into $m = 2$ lists $L_{time'}$ and $L_{price'}$ (see Tables 5.4a and 5.4b respectively). Each list is composed by pairs of qw and cost attribute, and it is ordered w.r.t. the cost attribute.

Iteration	qw	$L_{time'} \downarrow$	qw	$L_{price'} \downarrow$
$\rightarrow 1$	qw_{10}	39.333	qw_1	36.667
2	qw_9	39.389	qw_2	36.933

PA-1) Parallel access at iteration 1

Iteration	qw
$SEEN_{time'}$	qw_{10}
$SEEN_{price'}$	qw_1

D-1) Items seen at iteration 1

Iteration	qw	$L_{time'} \downarrow$	qw	$L_{price'} \downarrow$
1	qw_{10}	39.333	qw_1	36.667
$\rightarrow 2$	qw_9	39.389	qw_2	36.933
3	qw_{11}	39.389	qw_{12}	36.933

PA-2) Parallel access at iteration 2

Iteration	qw
$SEEN_{time'}$	qw_{10}, qw_9
$SEEN_{price'}$	qw_1, qw_2

D-2) Items seen at iteration 2

Iteration	qw	$L_{time'} \downarrow$	qw	$L_{price'} \downarrow$
2	qw_9	39.389	qw_2	36.933
$\rightarrow 3$	qw_{11}	39.389	qw_{12}	36.933
4	qw_8	39.600	qw_3	37.467

PA-3) Parallel access at iteration 3

Iteration	qw
$SEEN_{time'}$	qw_{10}, qw_9, qw_{11}
$SEEN_{price'}$	qw_1, qw_2, qw_{12}

D-3) Items seen at iteration 3

Iteration	qw	$L_{time'} \downarrow$	qw	$L_{price'} \downarrow$
3	qw_{11}	39.389	qw_{12}	36.933
$\rightarrow 4$	qw_8	39.600	qw_3	37.467
5	qw_{12}	39.600	qw_{11}	37.467

PA-4) Parallel access at iteration 4

Iteration	qw
$SEEN_{time'}$	$qw_{10}, qw_9, qw_{11}, qw_8$
$SEEN_{price'}$	$qw_1, qw_2, qw_{12}, qw_3$

D-4) Items seen at iteration 4

Iteration	qw	$L_{time'} \downarrow$	qw	$L_{price'} \downarrow$
4	qw_8	39.600	qw_3	37.467
$\rightarrow 5$	qw_{12}	39.600	qw_{11}	37.467
6	qw_1	40.000	qw_4	38.000

PA-5) Parallel access at iteration 5

Iteration	qw
$SEEN_{time'}$	$qw_{10}, qw_9, qw_{11}, qw_8, qw_{12}$
$SEEN_{price'}$	$qw_1, qw_2, qw_{12}, qw_3, qw_{11}$

D-5) Items seen at iteration 5

$DISTS$	
qw	$distance \downarrow$
qw_{10}	0.6667
qw_{11}	0.8110
qw_{12}	1.1392
qw_9	1.2293
qw_1	1.3333
qw_8	2.1705
qw_2	2.2667
qw_3	3.1013

K) distances of seen items

Table 5.5 – Top-k^{qw} example

At each iteration j shown in Table 5.5, the lists $L_{time'}$, $L_{price'}$ are accessed in simultaneously to read the j^{th} query workflows (see Tables 5.5PA-#). The j^{th} query workflows are appended to the lists

$SEEN_{time'}$, $SEEN_{price'}$ (see Tables 5.5D-#). At each iteration it is verified if there are k query workflows already seen in every list $SEEN_{time'}$ and $SEEN_{price'}$ (see Tables 5.5D-#). As we have defined $k = 1$, the iterations stop at the 5^{th} iteration where there are at least k query workflows already seen in $SEEN_{time'}$ and $SEEN_{price'}$, *i.e.* qw_{11}, qw_{12}. Then the query workflows in $SEEN_{time'}$ and $SEEN_{price'}$ are read by random access, and the distances by $dist$ are appended to $DISTS$. Finally, the $DISTS$ list is ordered increasingly and the best query workflow qw_{10} is found at the $k = 1$ position (see Tables 5.5D-5).

5.2.3 Optimality of top-k^{qw}

The optimality of top-k^{qw} is given by the *monotone* and *strict* properties of the scoring function (*i.e.* $dist$ function in our context). Fagin defined these properties in [Fag99, FLN03] and are sufficient for the optimiality of the algorithm in the worst case.

For a given scoring function $score$, the *monotone* property ensures that two items X and Y, each characterized by m attributes, keep an order such that $score(x^1, .., x^m) \leq score(y^1, .., y^m)$ iff every attribute $j \in [1 .. m]$ holds that $x^j \leq y^j$. For a given best score B, the $score$ function is *strict* if an item X gets the score B when every attribute of X is equal to B. This is, $score(x^1, .., x^m) = B$ iff $x^j = B$ for every attribute $j \in [1 .. m]$.

Such properties rely on the 'best item' semantics that is given by the proximity of the items from the lower bound B; *i.e.* the closer x^j is from B, the item is better. This intuition is analogous to our optimization objective given by $oo\langle(o^1, w^1), .., (o^m, w^m)\rangle$ that denotes the ideal point to reach. Therefore, the monotone and strict properties can be written in terms of the optimization objective oo and the scoring function $dist$ as follows.

Property 5.2.1 [monotone] Given an optimization objective $oo\langle(o^1, w^1), .., (o^m, w^m)\rangle$, and two equivalent query workflows $qw_1\langle x_1^1, .., x_1^m\rangle$, $qw_2\langle x_2^1, .., x_2^m\rangle$; the $dist$ function defines an order that allows to find the optimal query workflow(s). This is, $dist(qw_1, oo) \leq dist(qw_2, oo)$ iff $|val(x_1^j) - val(o^j)| \leq |val(x_2^j)) - val(o^j)|,\ \forall j \in [1 .. m]$.

Property 5.2.2 [strict] Given an optimization objective $oo\langle(o^1, w^1), .., (o^m, w^m)\rangle$, and a query workflow $qw\langle x^1, .., x^m\rangle$ the $dist$ function is strict if it takes the distance 0 when each cost attribute of qw is equal to each attribute of oo. This is, $dist(qw, oo) = 0$ iff $val(x^j) = val(o^j),\ \forall j \in [1 .. m]$.

As the weighted distance $dist$ has its domain on the positive reals, both the monotone and strict properties hold.

5.3 Conclusions

This chapter presented the solution space computation guided by an optimization objective of an SLA contract. The query workflow cost is computed by aggregating the activities' costs. In particular, for

obtaining the activity cost, it is considered the activity's interactions with other activities and with the invoked service. The data-related parameters (*e.g.* data statistics, data stream rate) are not available during the optimization. Therefore, we proposed a build-time cost formulation that evicts such parameters. The resulting cost is therefore an approximation.

The query workflows' costs are compared with the optimization objective by means of a weighted distance that considers the preferences among cost attributes. The weighted distance is used by the top-k^{qw} algorithm, an adaptation of the Fagin's Algorithm[FLN03] that we use to find the solution space of query workflows. The weighted distance holds the sufficient properties defined by Fagin for ensuring the top-k^{qw} optimality in the worst case. The optimization objective represents a lower bound and any query workflow represents an upper bound. Such bounds can help to avoid the exhaustive traversing of the search space.

Additional costs related to service-based environments may arise from quantitative measures of services [NLF99], network [ACH98], energy consumption [CS12, FP13], and storage [LBN07, LBZ$^+$11]; and the quality of data [OEtH02, MSV$^+$02]. Our search strategy is extensible to new costs as we treat the query workflows cost as fuzzy data.

Most of the forerunner query optimizers merge the search space generation and the search strategy in order to apply mathematical optimization techniques, *e.g.* dynamic programming, branch-and-bound, implicit enumeration. We consider that the separation of aspects should remain like in extensible optimizers [HFLP89, PHH92] in order to provide flexibility and portability of the optimization process in the context of hybrid queries and service-based environments.

CHAPTER 6

Implementation and validation

This chapter is organized as follows. Section 6.1, introduces the hybrid query processing with HyQoZ, the HYbrid Query OptimiZer. Section 6.2, describes the HyQoZ components through REST interfaces and the messages for coordinating them. Section 6.3, presents a series of coordinations of the HyQoZ componentes. Section 6.4, presents the implementation of the testbed system HyQoZTestbed that orchestrates the HyQoZ components and enables the access to each component under different configurations. Finally, Section 6.5 concludes this chapter.

6.1 Hybrid query processing

Figure 6.1 shows the component diagram for processing hybrid queries with the HYbrid Query OptimiZer (HyQoZ). An application representing a data consumer expresses hybrid queries based on the information about service instances provided by the API directory, and define the SLA to accomplish. Applications may require either the evaluation of the hybrid query or the optimum query workflow implementing the hybrid query for its further execution. In such cases applications request either the evaluator Hypatia or the optimizer HyQoZ.

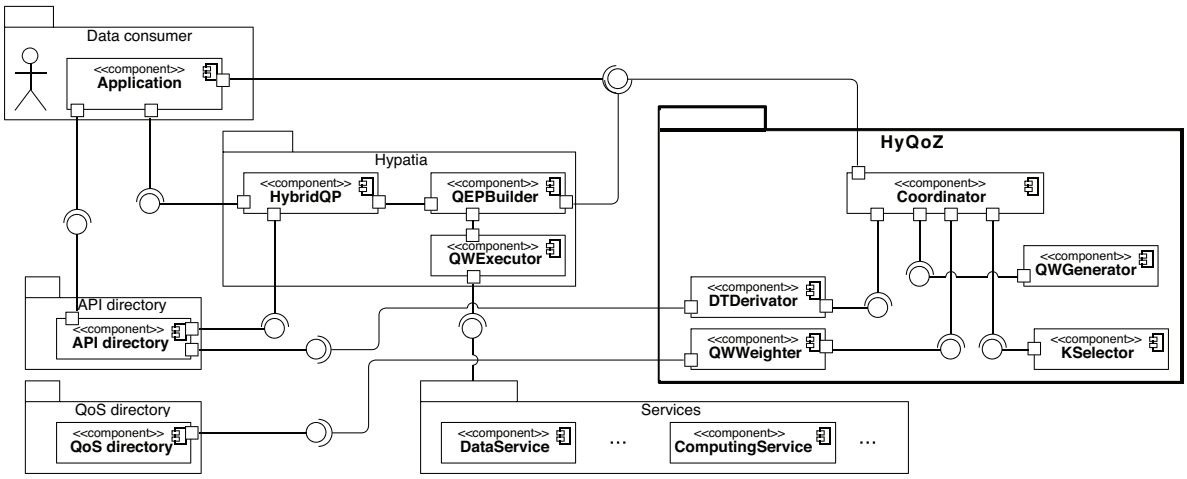


Figure 6.1 – Hybrid query processing components

— **Hypatia** [CVVSC12] accepts the hybrid query evaluation requests. HybridQP validates the expression according to the information provided by the API directory. QEPBuilder derives the optimization objective from the SLA contract and requests the hybrid query optimization to HyQoZ. The resulting query workflow is executed by the QWExecutor.

— **HyQoZ** accepts hybrid query optimization requests and looks for the satisfaction of the optimization objectives derived from the SLA contracts. HyQoZ is composed by a series of components that implement the optimization stages described in previous chapters.

Internally, HyQoZ is composed by a series of orthogonal components that together perform the optimization. Components exchange self-descriptive messages carrying the required information for articulating the optimization. We adopt Prolog style functors for representing optimization information, *e.g.* hybrid query expressions `sco/1`,data transformation function `dtf/4`, optimization objective `oo/3`, query workflows `qw/6`. We refer the reader to the Appendix B.1 for the syntax conventions.

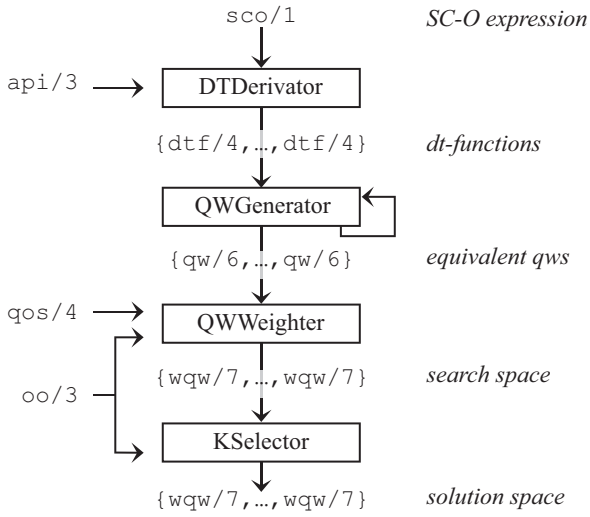


Figure 6.2 – HyQoZ information flow

Figure 6.2 shows how optimization information flows through the HyQoZ components.

— **DTDerivator** analyzes the hybrid query expression `sco/1` and asks to the API directory for the required data and computing services' APIs `api/3` for type validations. The result is a set of data transformations functions `dtf/4` representing the hybrid query.

— **QWGenerator** takes a set of data transformation functions `dtf/4` and generates the equivalent query workflows `qw/6` that implement the hybrid query.

— **QWWeighter** takes a series of query workflows `qw/6` and produces query workflows `tqw/7` tagged with their costs `cost/3`. The cost attributes are computed in accordance with the attributes defined by the optimization objective `oo/3` and using QoS measures `qos/4` provided by the QoS directory.

— **KSelector** looks for the solution space of tagged query workflows `tqw/7` that best conform the optimization objective `oo/3` of the SLA contract.

Next we present the REST interfaces of HyQoZ components that specify the rules for access them and the messages they exchange for enabling the optimization.

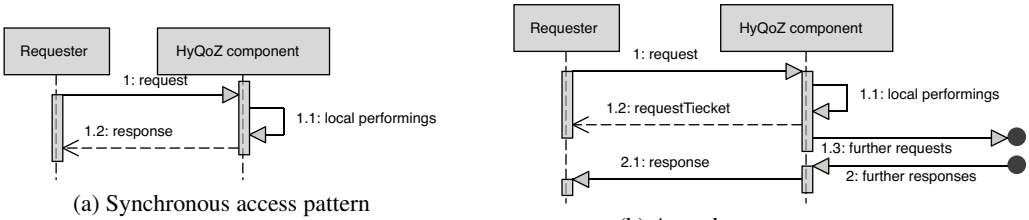
6.2 HyQoZ components

Each component is accessible through an interface described using the architectural style *Representational State Transfer* (REST) [Fie00]. Messages exchanged by HyQoZ components contain (1) information for enabling the optimization, and (2) a series of fields for articulating orchestrations or choreographies of the HyQoZ components. Components use a series of core libraries we implemented in GNU Prolog 1.4.4 (see Table 6.1).

Component core	GitHub repository
DTDerivator	`https://goo.gl/Bq3Cet`
QWGenerator	`https://goo.gl/GBQIgu`
QWWeighter	`https://goo.gl/P4zbGC`
KSelector	`https://goo.gl/ATnXfQ`
commons	`https://goo.gl/rqGvZx`

Table 6.1 – HyQoZ core libraries.

Every HyQoZ component implements the synchronous and asynchronous communication patterns via the request and response context messages described in Appendix B.2. The synchronous pattern (see Figure 6.3a) provides access to a single stage of the optimization process (*e.g.* derivation, generation) and its computation is done as an elementary service instance [FDBP01]. The asynchronous pattern (see Figure 6.3b) provides access to the complete optimization process (or the reminding stages).

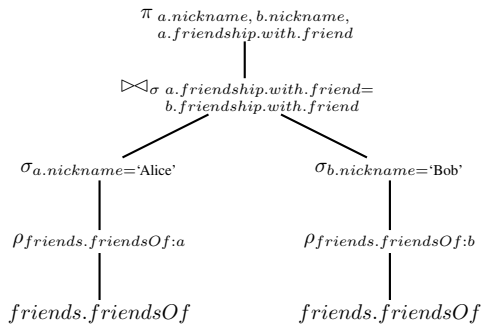


(a) Synchronous access pattern

(b) Asynchronous access pattern

Figure 6.3 – Communication patterns implemented by HyQoZ components

Next we describe the REST interfaces of the HyQoZ components and give examples of the input and output messages for enabling the optimization. Every REST method is exemplified with input and output messages containing the optimization information for the hybrid query *Which are the common friends of Alice and Bob ?* with the optimization objective $oo\langle(time = 38.4, 3), (price = 40.0, 1), (energy = 43.0, 2)\rangle$.



6.2.1 Data transformation function derivator (DTDerivator)

DTDerivator/derive Receives as input the hybrid query expression `sco/1` (see Figure 6.4). The output is the set of data transformation functions `{dtf/4,...,dtf/4}` along with an identifier of the hybrid query.

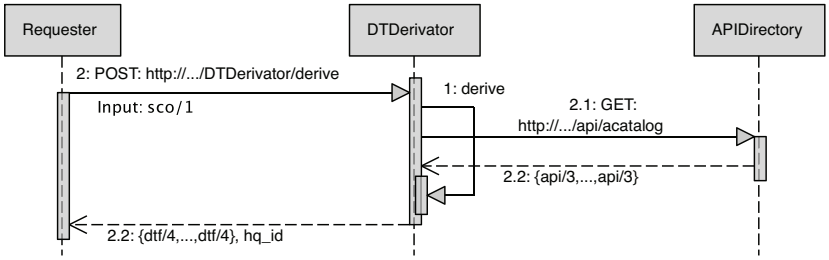


Figure 6.4 – Derivation request

URL : **`http://<baseuri>/DTDerivator/derive`**

Method : `POST`

Input message :

`requestContext`	Context of the request
`sco`	`sco/1`

Output message :

`requestContext`	Context of the request
`responseContext`	Context of the response
`hq_id`	Identifier of hybrid query
`dtfs`	{`dtf/4`,..,`dtf/4`}

Input message example :

```
“derivationRequest” :{
    “requestContext” :{...},
    “sco” :“sco(pi([a::friendship::with::friend],
                sigma(a::nickname=‘Alice’,
                  sigma(b::nickname=‘Bob’,
                    corr([a::friendship::with::friend
                          =c::friendship::with::friend],
                      rho(friends::friendsof as a,friends::friendsof),
                      rho(friends::friendsof as b,friends::friendsof))))))”}
```

Output message example :

```
“derivationResponse” :{
    “requestContext” :{...},
    “responseContext” :{...},
    “hq_id” :“ccl4li5n1q9b”
    “dtfs” :[“dtf” :“dtf([a],[],[a::friendship::with::friend],p_a1)”,
          “dtf” :“dtf([a],[a::nickname=‘Alice’],[a::nickname,a::friendship],r_a1)”,
          “dtf” :“dtf([b],[b::nickname=‘Bob’],[b::nickname,b::friendship],r_b1)”,
          “dtf” :“dtf([a,b],[a::friendship::with::friend=b::friendship::with::friend],
                [a::nickname,a::friendship,b::nickname,b::friendship],c_ab1)”]}
```

DTDerivator/optimize Receives as input the hybrid query expression `sco/1` and an optimization objective `oo/3` (see Figure 6.4) The output is the identifier of the hybrid query and a reception timestamp. The DTDerivator sends forward the request for the generation of query workflows to a QWGenerator instance. When the optimization is done, the requester gets the resulting query workflows with their costs `{tqw/7,...,tqw/7}` via a callback port specified into the requestContext field of the input message.

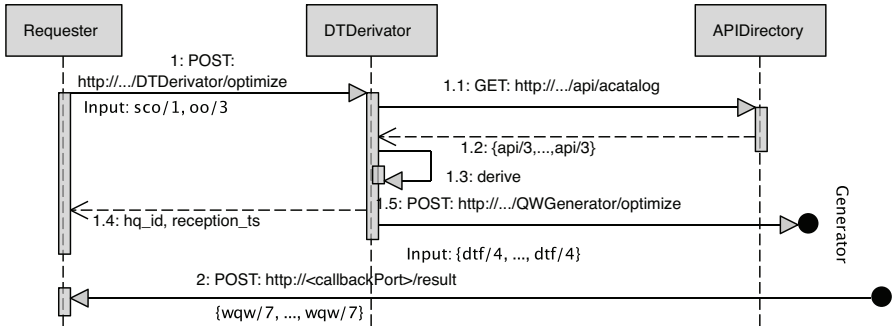


Figure 6.5 – Optimization request via DTDerivator

URL : **`http://<baseuri>/DTDerivator/optimize`**
Method : `POST`

Input message

requestContext	Context of the request
sco	Hybrid query expression `sco/1`
oo	Optimization objective `oo/3`

Output message

requestContext	Context of the request
responseContext	Context of the response
hq_id	Resource identifier of hybrid query
reception_ts	Reception timestamp

Input message example :

```
"derivationRequest" :{
    "requestContext" :{...},
    "sco" :"sco(pi([a::friendship::with::friend],
                sigma(a::nickname='Alice',
                  sigma(b::nickname='Bob',
                    corr([a::friendship::with::friend
                          =c::friendship::with::friend],
                      rho(friends::friendsof as a, friends::friendsof),
                      rho(friends::friendsof as b, friends::friendsof) ) ) ) ))",
    "oo" :oo(time(38.4, 3), price(40.0, 1), energy(43.0, 2)) }
```

Output message example :

```
"optimizationTicket" :{
    "requestContext" :{...},
    "responseContext" :{...},
    "hq_id" :"ccl4li5n1q9b"
    "reception_ts :" :"1379602397" }
```

6.2.2 Query workflow generator (QWGenerator)

QWGenerator/generate Receives as input a set `{dtf/4,...,dtf/4}` of data transformation functions (see Figure 6.6). The query parameter `flow` defines if the generation is data-flow oriented (*i.e.* `df`), or control-flow oriented (*i.e.* `cf`). The output is a set of equivalent query workflows `{qw/6,...,qw/6}` implementing the hybrid query along with an identifier of the data transformation functions.

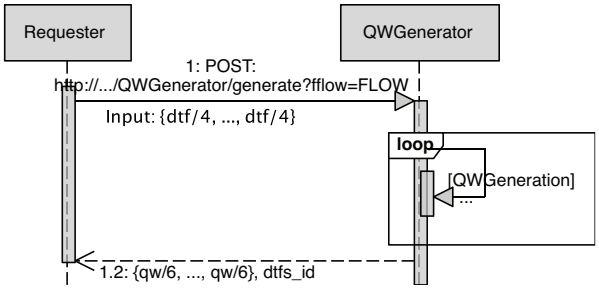


Figure 6.6 – Generation request

URL : **`http://<baseuri>/QWGenerator/generate`**

Method : `POST`

Input message :

requestContext	Context of the request
dtfs	`{dtf/4,..,dtf/4}`

Output message :

requestContext	Context of the request
responseContext	Context of the response
dtfs_id	Receipt of the hybrid query optimization
qws	Generated query workflows `{qw/6,..,qw/6}`

Query parameters :

flow Define if query workflows are generated with the data-flow (`df`) or control-flow (`cf`) perspective

Input message example :

```
"generationRequest" :{ "requestContext" :{...},
   "dtfs" :["dtf" :"dtf([a],[],[a::friendship::with::friend],p_a1)",
          "dtf" :"dtf([a],[a::nickname='Alice'],[a::nickname,a::friendship],r_a1)",
          "dtf" :"dtf([b],[b::nickname='Bob'],[b::nickname,b::friendship],r_b1)",
          "dtf" :"dtf([a,b],[a::friendship::with::friend=b::friendship::with::friend],
                 [a::nickname,a::friendship,b::nickname,b::friendship],c_ab1)"]}
```

Output message example :

```
"generationResponse" :{ "requestContext" :{...}, "responseContext" :{...}, "dtfs_id" :"dcm3oj7r2p3p",
   "qws" :["qw" :"qw([r_a1,r_b1,p_a1,c_ab1],[],[r_a1,r_b1,p_a1,c_ab1],
                  [(c_ab1,p_a1),(in,r_a1),(p_a1,out),
                   (r_a1,r_b1),(r_b1,c_ab1)],in,out)",
          ... ,
          "qw" :"qw([r_a1,r_b1,p_a1,c_ab1],[par, end_par],
                  [r_a1,r_b1,p_a1,c_ab1,par, end_par],
                  [(c_ab1,p_a1),(end_par,c_ab1),(in,par),
                   (p_a1,out),(par,r_a1),(par,r_b1),
                   (r_a1,end_par),(r_b1,end_par)],in,out)"]}
```

QWGenerator/optimize Receives as input a set of data transformation functions `{dtf/4,...,dtf/4}` and the optimization objective `oo/3` (see Figure 6.7). The query parameter `flow` defines if the generation is data-flow oriented (*i.e.* `df`), or control-flow oriented (*i.e.* `cf`). The output is the identifier of the set of data transformation functions for identifying the optimization result. The QWGenerator sends forward the optimization request for the weighting of query workflows to a QWWeighter instance. Once the optimization is done, the requester gets the optimization result of query workflows with their costs `{tqw/7,...,tqw/7}` via a callback port specified into the input message.

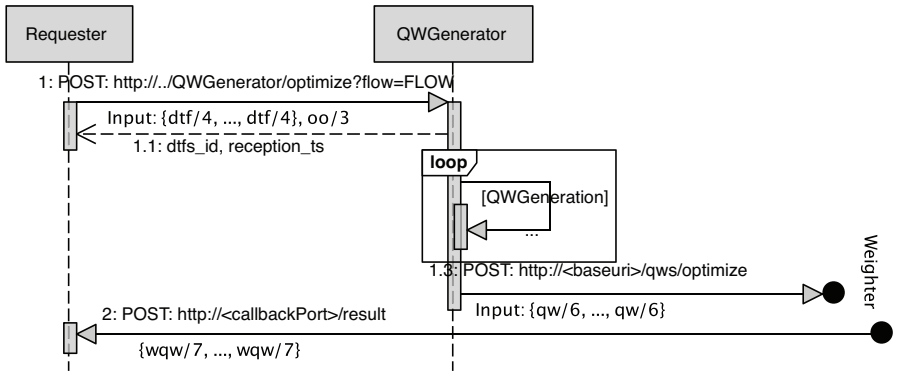


Figure 6.7 – Optimization request via QWGenerator

URL : **`http://<baseuri>/QWGenerator/optimize`**

Method : `POST`

Input message		**Output message**	
requestContext	Context of the request	**requestContext**	Context of the request
sco	Hybrid query `sco/1`	**responseContext**	Context of the response
oo	Optimization objective `oo/3`	**dtfs_id**	Resource identifier of *dt-functions*
		reception_ts	Reception timestamp

Query parameters :

flow Define if query workflows are generated with the data-flow (`df`) or control-flow (`cf`) perspective

Input message example :

```
"generationRequest" :{ "requestContext" :{...},
   "dtfs" :["dtf" :"dtf([a],[],[a::friendship::with::friend],p_a1)",
          "dtf" :"dtf([a],[a::nickname='Alice'],[a::nickname,a::friendship],r_a1)",
          "dtf" :"dtf([b],[b::nickname='Bob'],[b::nickname,b::friendship],r_b1)",
          "dtf" :"dtf([a,b],[a::friendship::with::friend=
                            b::friendship::with::friend],
                  [a::nickname,a::friendship,b::nickname,b::friendship],c_ab1)"
        ], "oo" :oo(time(38.4, 3), price(40.0, 1), energy(43.0, 2))}
```

Output message example :

```
"optimizationTicket" :{
     "requestContext" :{...},
     "responseContext" :{...},
     "dtfs_id" :"dcm3oj7r2p3p",
     "reception_ts :" :"1379602397"}
```

6.2.3 Query workflow cost weighter (QWWeighter)

QWWeighter/compute Receives as input a set `{qw/6,...,qw/6}` of query workflows and an optimization objective `oo/3` (see Figure 6.8). The output is a set of weighted query workflows `{tqw/7,...,tqw/7}` along with an identifier of the query workflows.

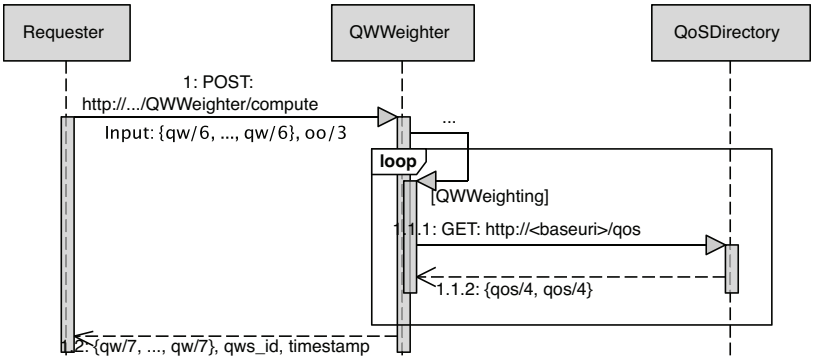


Figure 6.8 – Weighting request

URL :	**`http://<baseuri>/QWWeighter/compute`**		
Method :	`POST`		
Input message :		**Output message :**	
requestContext	Context of the request	**requestContext**	Context of the request
qws	`{qw/6,..qw/6}`	**responseContext**	Context of the response
		qws_id	Identifier of *dt-functions* and query workflows
		timestamp	Instant at which the costs are valid
		tqws	Search space `{tqw/7,..,tqw/7 }`

Input message example :

```
“weightingRequest” :{ “requestContext” :{...},
    “qws” :[“qw” :“qw([r_a1,r_b1,p_a1,c_ab1],[],[r_a1,r_b1,p_a1,c_ab1],
                  [(c_ab1,p_a1),(in,r_a1),(p_a1,out), (r_a1,r_b1),(r_b1,c_ab1)],in,out)”,
    ...      “qw” :“qw([r_a1,r_b1,p_a1,c_ab1],[par, end_par],[r_a1,r_b1,p_a1,c_ab1],
                  [(c_ab1,p_a1),(end_par,c_ab1),(in,par),(p_a1,out),(par,r_a1),
                   ,(par,r_b1),(r_a1,end_par),(r_b1,end_par)],in,out)”]}
```

Output message example :

```
“weightingResponse” :{ “requestContext” :{...}, “responseContext” :{...},
    “qws_id” : “y4r9I3Cv2p0t”, “timestamp :” :“1379618154”
    “tqws” :[“tqw” :{“qw” :“qw([r_a1,r_b1,p_a1,c_ab1],[],[r_a1,r_b1,p_a1,c_ab1],
                          [(c_ab1,p_a1),(in,r_a1),(p_a1,out),
                           (r_a1,r_b1),(r_b1,c_ab1)],in,out)”,
                “cost” :“cost(21, 23 20)’’},
    ... ,     “tqw” :{“qw” :“qw([r_a1,r_b1,p_a1,c_ab1],[par, end_par],
                          [r_a1,r_b1,p_a1,c_ab1, par, end_par],
                          [(c_ab1,p_a1),(end_par,c_ab1),(in,par),
                           (p_a1,out),(par,r_a1),(par,r_b1),
                           (r_a1,end_par),(r_b1,end_par)],in,out)”,
                “cost” :“cost(22, 23 21)”}] }
```

QWWeighter/optimize Receives as input a set `{qw/6,...,qw/6}` of query workflows and an optimization objective `oo/3` (see Figure 6.9). The output is the identifier of the query workflows. The QWWeighter instance sends forward the request for the selection of the solution space of query workflows to a KSelector instance. Once the optimization is done, the requester gets the resulting query workflows with their costs `{tqw/7,...,tqw/7}` via a reception port.

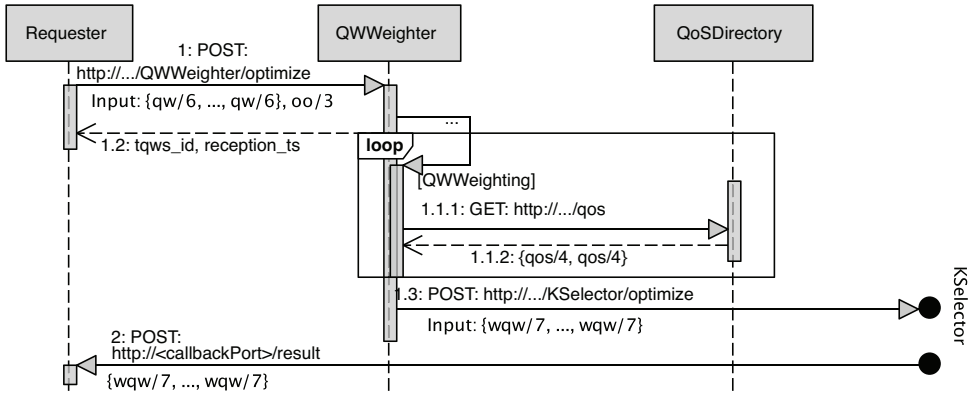


Figure 6.9 – Optimization request via QWWeighter

URL : **`http://<baseuri>/QWWeighter/optimize`**
Method : `POST`

Input message :		**Output message :**	
requestContext	Context of the request	**requestContext**	Context of the request
qws	`{qw/6,..qw/6}`	**responseContext**	Context of the response
		tqws_id	Identifier of *dt-functions* and query workflows
		reception_ts	Instant at the query workflows reception

Input message example :

```
"weightingRequest" :{ "requestContext" :{...},
   "oo" :oo(time(38.4, 3), price(40.0, 1), energy(43.0, 2))
   "qws" :["qw" :"qw([r_a1,r_b1,p_a1,c_ab1],[],[r_a1,r_b1,p_a1,c_ab1],
                  [(c_ab1,p_a1),(in,r_a1),(p_a1,out),
                   (r_a1,r_b1),(r_b1,c_ab1)],in,out)",
   ...      "qw" :"qw([r_a1,r_b1,p_a1,c_ab1],[par, end_par],
                  [r_a1,r_b1,p_a1,c_ab1, par, end_par],
                  [(c_ab1,p_a1),(end_par,c_ab1),(in,par),
                   (p_a1,out),(par,r_a1),(par,r_b1),
                   (r_a1,end_par),(r_b1,end_par)],in,out)"]}
```

Output message example :

```
"optimizationTicket" :{ "requestContext" :{...}, "responseContext" :{...},
     "tqws_id" :"r3u8OlTs6q2q"
     "reception_ts :" :"1379623365"}
```

6.2.4 Solution space selector (KSelector)

KSelector/optimize Receives as input a set of {tqw/7,...,tqw/7} of weighted query workflows, an optimization objective oo/3 (see Figure 6.10). The query parameter topk parametrize the number of expected k query workflows with default topk = 1. The output is the solution space of {tqw/7,...,tqw/7} and the id of the search space.

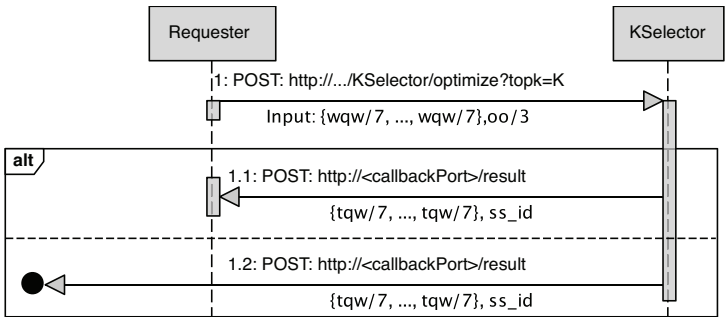


Figure 6.10 – Optimization request via KSelector

URL : **http://<baseuri>/KSelector/optimize**
Method : POST

Input message :

requestContext	Context of the request
tqws	{tqw/7,..tqw/7}

Output message :

requestContext	Context of the request
responseContext	Context of the response
ss_id	Resource identifier of the solution space
ss	Solution space {tqw/7,..tqw/7}

Query parameters :

topk	Positive integer that specifies the number of expected results

Input message example :

```
"selectionRequest" :{"requestContext" :{...},
   "oo" :oo(time(38.4, 3), price(40.0, 1), energy(43.0, 2))}
   "tqws" :["tqw" :{"qw" :"qw([r_a1,r_b1,p_a1,c_ab1],[],[r_a1,r_b1,p_a1,c_ab1],
                          [(c_ab1,p_a1),(in,r_a1),(p_a1,out),
                           (r_a1,r_b1),(r_b1,c_ab1)],in,out)",
                  "cost" :"cost(21, 23 20)''} },
   ...      "tqw" :{"qw" :"qw([r_a1,r_b1,p_a1,c_ab1],[par, end_par],
                          [r_a1,r_b1,p_a1,c_ab,par, end_par1],
                          [(c_ab1,p_a1),(end_par,c_ab1),(in,par),(p_a1,out),
                           (par,r_a1),(par,r_b1),(r_a1,end_par),(r_b1,end_par)],
                           in,out)", "cost" :"cost(22, 23 21)"} ] }
```

Output message example :

```
"weightingResponse" :{ "requestContext" :{...}, "responseContext" :{...}, "ss_id" :"y8c4mNE5k5a",
    "ss" :["tqws" :{ 'tqw" :"tqw([r_a1,r_b1,p_a1,c_ab1],[],[r_a1,r_b1,p_a1,c_ab1],
                          [(c_ab1,p_a1),(in,r_a1),(p_a1,out),
                           (r_a1,r_b1),(r_b1,c_ab1)],in,out)",
                  "cost" :"cost(21, 23, 20)''}}] }
```

6.3 Coordinating the HyQoZ components

Behind the use of REST style interfaces, there is the interest to separate the concerns of HyQoZ for enabling the portability along platforms, the scalability for enhancing the HyQoZ throughput, and the extensibility for adapting HyQoZ to the variety of potential cost attributes in service-based environments. The HyQoZ components can have an orthogonal evolution (*e.g.* generation strategies, cost models, solution space selection) as long as the proposed data structures and properties remain.

The stateless client-server interaction via the REST interfaces enables the (1) dynamic binding of HyQoZ components, (2) the parallelization of tasks (*e.g.* several QWWeighters performing the cost estimation of a fragmented search space), and (3) the performance enhancement of HyQoZ itself. In this way, the HyQoZ components can be coordinated depending on the interests of developers about the resource usage, communication cost, and desired query workflows optimality.

6.3.1 Orchestration

An orchestration is performed by an Orchestration Engine (OE) that interprets a program specified in a language; *e.g.* ASM-based [CV11, BT08], YAWL[vdAtH05], WSFL[Ley01], WS-BPEL[JE07]. The component instance binding may be dynamic or static. In the dynamic binding case, there are assumed multi-criteria decision capabilities [JMG05, CCDS04] of the orchestration engine for selecting the most suitable component instances for an orchestration execution. In the static binding case, the component instances are chosen a priori. In both cases, an instance of a HyQoZ component implements a synchronous communication pattern, *i.e.* it receives the orchestration requests from the OE, and the results are delivered to it.

The sequence diagram in Figure 6.11 shows the orchestration of HyQoZ components. The requester sends to the OE the hybrid query `sco/1` and the optimization objective `oo/3`. The `sco/1` is passed to a DTDerivator instance through the resource DTDerivator/derive and it computes the derivation getting the APIs `api/3` from the API directory.

The resulting `{dtf/4,..,dtf/4}` are returned back to the OE and passed to a QWGenerator instance through the QWGenerator/generate resource. Each QWGenerator instance implements its own generation rules regarding the compromise between efficiency and quality of the alternative query workflows. The resulting query workflows `{qw/6,...,qw/6}` are returned back to the OE.

The OE requests the QWWeighter through the resource QWWeighter/compute for computing the search space of query workflows according to the cost attributes specified by `oo/3`. The QWWeighter traverse the query workflows and retrieves the required QoS measures `qos/4` from the QoS directory. The QWWeighter delivers the search space `{tqw/7,...,tqw/7}`

Finally, the OE requests the selection of the solution space to a KSelector instances through the KSelector/optimize. The KSelector delivers the resulting `{tqw/7,...,tqw/7}` to the OE and this does to the requester.

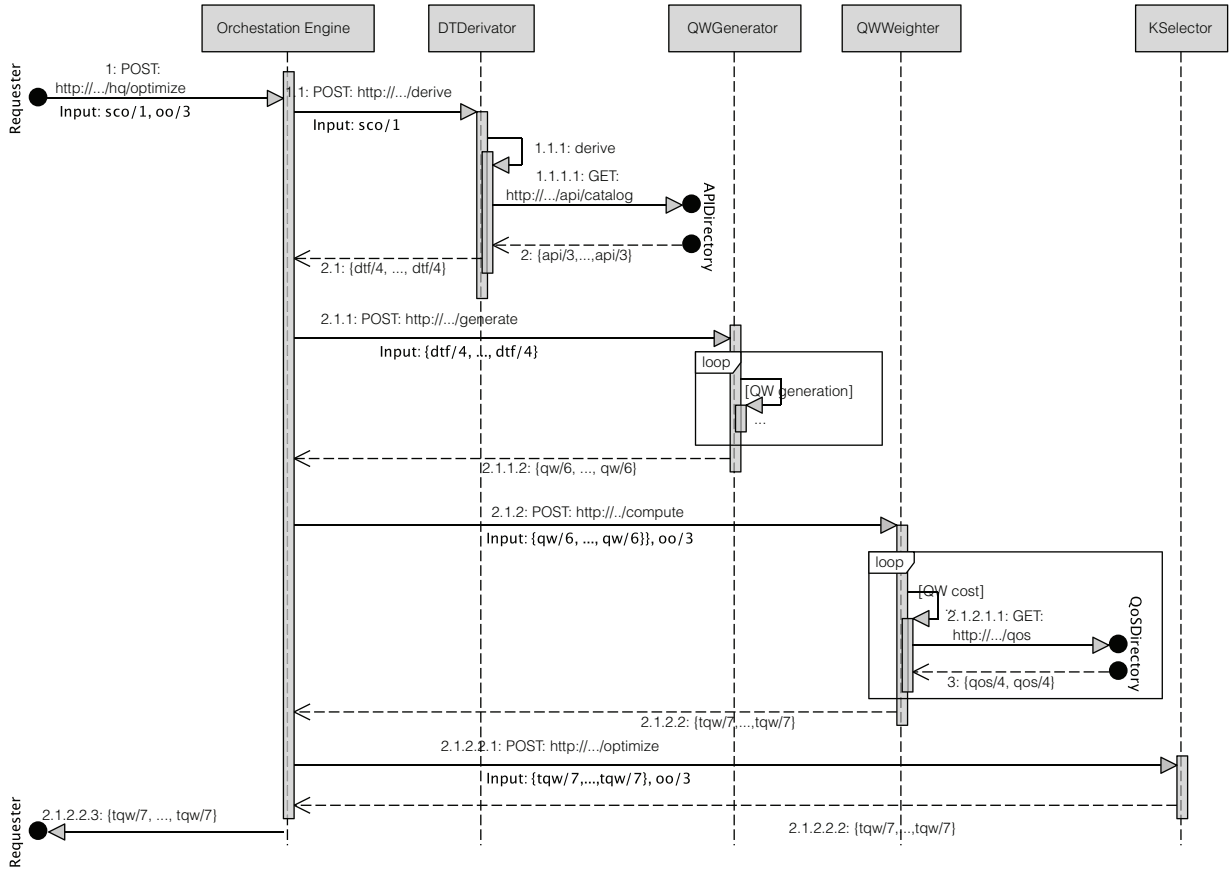


Figure 6.11 – Orchestration of HyQoZ components

6.3.2 Choreography

The choreography is a sequence of asynchronous requests via the resource /optimize implemented by every HyQoZ component. At each step the choreography is blocked until the component finishes its task. The messages among components carry both (1) the result of the preceding component and (2) the context information (*cf.* Appendix B.2). At each step, it is assumed the binding to the succeeding component instance. The sequence diagram in Figure 6.12 shows the choreography of HyQoZ components.

The choreography begins when the requester invokes a DTDerivator instance. The requester sends to the resource DTDerivator/optimize the hybrid query `sco/1`, the optimization objective `oo/3`, and a callback port specified in the context message. The DTDerivator derives the `{dtf/4,...,dtf/4}` and requests the generation of the query workflows to a QWGenerator instance through the resource QWGenerator/optimize. The QWGenerator generates the equivalent `{qw/6,...,qw/6}` and requests their cost estimation to a QWWeighter instance through the resource QWWeighter/optimize. The QWWeighter computes the search space of query workflows with their costs according to the cost attributes defined by the optimization objective `oo/3`. Finally, the QWWeighter requests the selection of the solution space to a KSelector instance through the resource KSelector/optimize. The KSelector delivers the solution space `{tqw/7,...,tqw/7}` to the requester via the callback port.

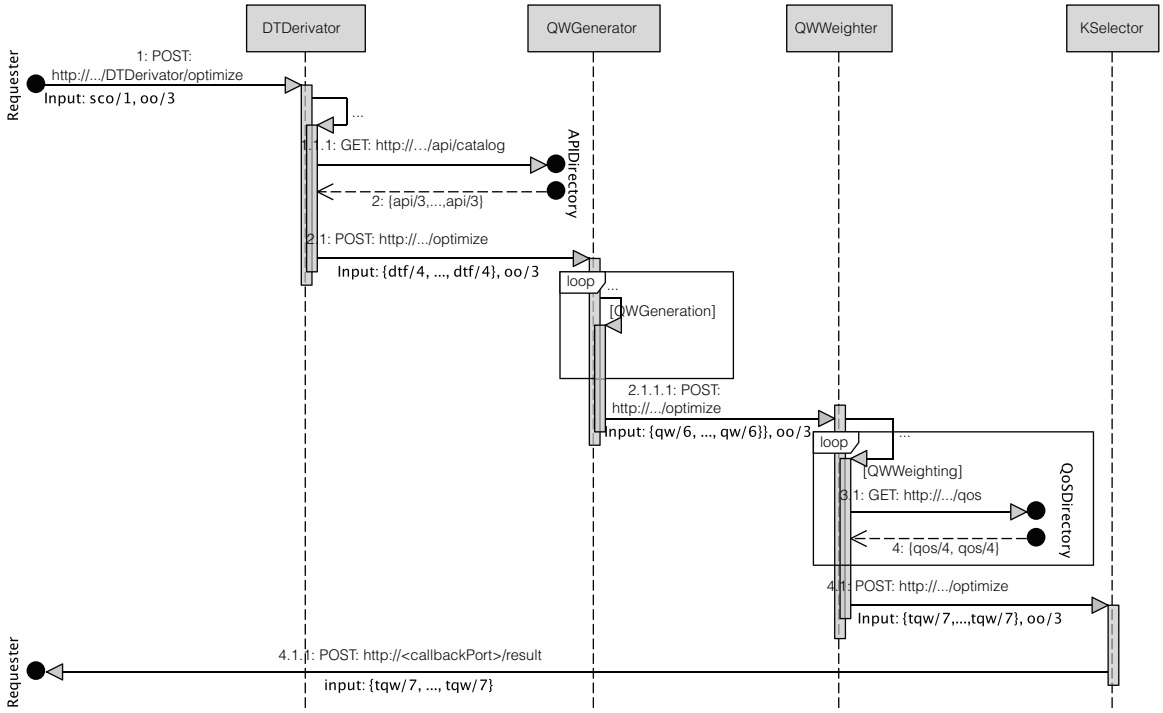


Figure 6.12 – Independent parallel choreography

6.3.3 Pipelined choreography

The pipelined choreography is a sequence of asynchronous requests via the resource /optimize implemented by every HyQoZ component. The pipeline starts when the QWGenerator produces the first query workflow `qw/6`. The pipeline finishes once a stop criteria is reached. Then, the HyQoZ components are stopped by calling back them with the result in stack fashion. The sequence diagram in Figure 6.12 shows the pipelined choreography.

The choreography begins when the requester invokes a DTDerivator instance. The requester sends to the resource DTDerivator/optimize the hybrid query `sco/1`, the `oo/3`; and (within the context message) the callback port and a stop condition, *e.g.* time threshold, first k good results, top-k results. Every HyQoZ component defines its own callback port. The DTDerivator requests the query workflow generation to a QWGenerator instance through the resource QWGenerator/optimize. As soon as the QWGenerator starts to produce query workflows `qw/6`, it request their cost estimation to the QWWeighter according to the optimization objective `oo/3`. The QWWeighter en-queues the tagged query workflows `tqw/7` to the KSelector through resource KSelector/optimize. When the KSelector reaches the stop condition, the results are sent back in sequence through the callback ports until the requester gets the solution space `tqw/7`.

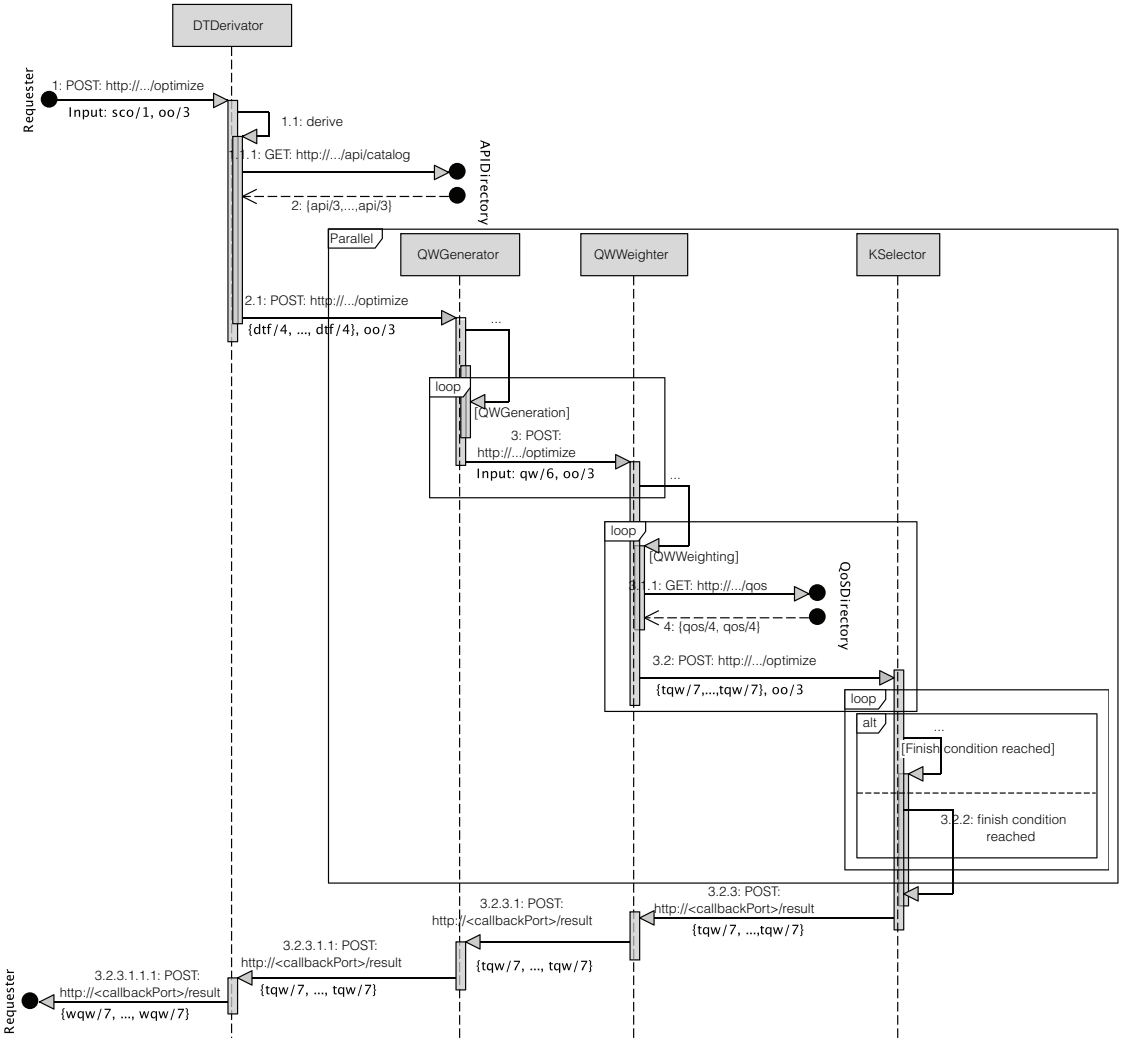


Figure 6.13 – Pipelined choreography

6.4 HyQoZTestbed

HyQoZTestbed is a testbed system that allows to characterize the hybrid query optimization implemented by HyQoZ components. Via HyQoZTestbed it is possible to (1) generate synthetic hybrid queries (2) perform requests to the HyQoZ components described in Section 6.2, and (3) compare the behavior of the hybrid query optimization under different conditions.

6.4.1 Architecture and implementation

Figure 6.14 shows the architecture of HyQoZTestbed.

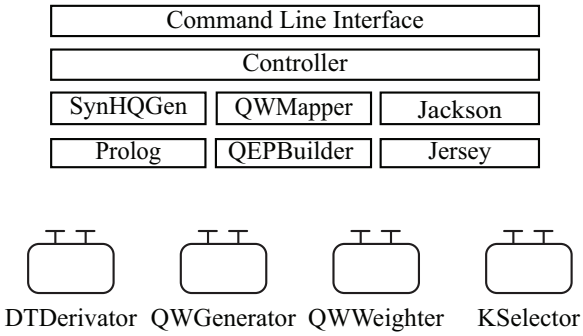


Figure 6.14 – HyQoZTestbed architecture

1. **Command Line Interface.** HyQoZTestbed is accessible through a Command Line Interface described in Appendix B.4. The parameters passed to CLI enable different configurations.
2. **Synthetic hybrid query generator (SynHQGen).** It generates synthetic hybrid queries. It takes as input a hybrid query signature that describes the number of involved data services, the way to correlate them, the filters, and projections. The signature denotes the complexity of the hybrid query.
3. **Hybrid query compiler (QEPBuilder) and query workflow mapper (QWMapper).** QEPBuilder is the Hypatia hybrid query compiler that generates a single (data-flow oriented) query workflow. The query workflows produced by QEPBuilder can be used for benchmarking the quality of the query workflows proposed by the HyQoZ w.r.t. different optimization objectives. For enabling the structural comparison, there is QWMapper that translates the Hypatia's query workflows into HyQoZ query workflows and assigns hashcodes for identifying them.
4. **HyQoZ testbed controller (HyQoZTestbed).** It orchestrates the hybrid query optimization with different configurations and enables the access either to each HyQoZ component independently or to a composition of them. It is possible to generate synthetic hybrid queries, generate data-flow or control-flow oriented query workflows, estimate either the build-time or run-time cost, and define the selection of the solution space w.r.t. different optimization objectives.

HyQoZTestbed provides the access to a synthetic hybrid query generator SynHQGen implemented for GNU Prolog 1.4.4. Synthetic queries are characterized by a signature that denotes the complexity of the hybrid query in terms of the effort for optimizing it (see Section 6.4.2).

In order to enable the benchmark of the hybrid queries produced by HyQoZ, we integrate the QEPBuilder class of hybrid query evaluator Hypatia. QEPBuilder is a hybrid query compiler written in Java that produces a single query workflow that can be compared with the solution space provided by HyQoZ via the QWMapper. QWMapper maps the Hypatia query workflow representation to our query workflow representation `qw/6`.

Table 6.2 lists the modules required by provided by HyQoZTestbed which however might be used independently. Table 6.3 lists additional libraries used by HyQoZTestbed.

HyQoZTestbed modules		
Module	**Repository**	**Platform**
SynHQGen	GitHub: `http://goo.gl/dvPqRM`	GNU Prolog 1.4.4
QEPBuilder (Hypatia)	CodePlex: `http://goo.gl/gGAU02`	Java JSE 1.6.0_65
HyQoZTestbed Controller	GitHub: `http://goo.gl/8LYkWO`	Java JSE 1.6.0_65

Table 6.2 – Modules of HyQoZTestbed

Library	**Version**	**Platform**
Apache CLI	1.2	Java JSE 1.6.0_65
Apache Configuration	1.9	Java JSE 1.6.0_65
Apache Collections	3.2.1	Java JSE 1.6.0_65
Apache Logging	1.1.1	Java JSE 1.6.0_65
Apache Lang	2.6	Java JSE 1.6.0_65
Guava library	14-rc1	Java JSE 1.6.0_65

Table 6.3 – Libraries used by HyQoZTestbed

HyQoZTestbed orchestrates the HyQoZ components performing REST invocations via Jersey 1.18 and data bindings via Jackson 2.1.0. The information related to the optimization is wrapped into Java objects that are mapped to its JSON representation by Jackson (see input and output message examples in Section 6.2).

Via the Command Line Interface, HyQoZTestbed enables the access to the HyQoZ components through their REST interfaces presented in Section 6.2. For instance, for a given hybrid query expression `sco/1` it is possible to request the derivation of the data transformation functions `dtf/4`, by means of the command

```
java -jar hyqoztestbed.jar -derivate -sco [sco/1] -output [OUTPUTFILE.txt]
```

or the generation of the equivalent query workflows for a given hybrid query expression by the command

```
java -jar hyqoztestbed.jar -generate -sco [sco/1] -output[OUTPUTFILE.txt] -controlflow
```

6.4.2 Synthetic hybrid queries generation

SynHQGen generates a `sco/1` expression for a given hybrid query signature of the form

$$\texttt{hq\_signature}(\#\texttt{DSs}, \#\texttt{BindJoins}, \#\texttt{Joins}, \#\texttt{Filters}, \#\texttt{BlockingProjections}, \#\texttt{Projections})$$

where $\#\texttt{DSs}$ denotes the number of data sources in the hybrid query, $\#\texttt{BindJoins}$ denotes the number of bind joins among the data sources, $\#\texttt{Joins}$ denotes the number of joins between the data sources [1], $\#\texttt{Filters}$ denotes the number of filtering operators, $\#\texttt{BlockingProjections}$ denotes the number of projections that generate anti-dependency, $\#\texttt{Projections}$ denotes the number of (non-blocking) projections.

The following command generates the possible hybrid query signatures for each $i \in [n \,..\, N]$ number of data services.

`java -jar hyqoztb.jar -genshqs -n` $=$ `[n] -N` $=$ `[N] -outputdir [OUTPUTDIR]`

For every number of data sources $i \in [n \,..\, N]$, HyQoZTestbed constructs the possible hybrid query signatures `hq_signature/6` and request the generation of the hybrid query expression `sco/1`. The resulting expressions are stored into the file `OUTPUTDIR/#DSs_DSs.txt` where `#DS` is the number of data services involved in the hybrid queries.

6.4.3 Measuring the search space sizes

HyQoZTestbed allows to generate the control-flow or data-flow search spaces for a given hybrid query expression or signature. We selected seven hybrid queries listed in Table 6.4 for measuring the hybrid query complexity. Each hybrid query complies with the examples used in previous chapters, *i.e.* friend finder, friends' interests, and fiendships.

— `HQ1` The purpose of the `HQ1` is to show the simplest case of a hybrid query with only one alternative query workflow.

— `HQ2` The purpose of the `HQ2` is to show a hybrid query with a single data source which however has some alternative query workflows.

— `HQ3` The purpose of the `HQ3` is to have a search space with bind-joins.

— `HQ4`, `HQ5` The purpose of `HQ4` and `HQ5` is to have a search space with joins, which is bigger than the one with bind-joins.

— `HQ6`, `HQ7` The purpose of `HQ6` and `HQ7` is to have a search space with both bind-joins and joins.

1. The signature has to hold $\#\texttt{DSs} = (\#\texttt{BindJoins} + \#\texttt{Joins}) + 1$. If a data source is not accessible via a bind-join operator, there is a retrieval expression bounding constants to the bound attributes.

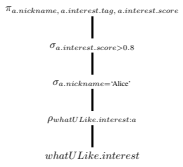
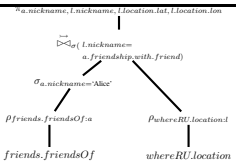
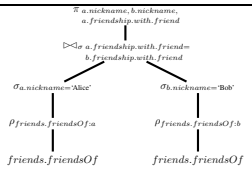
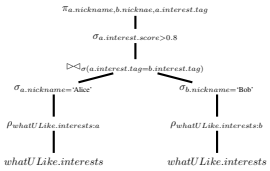
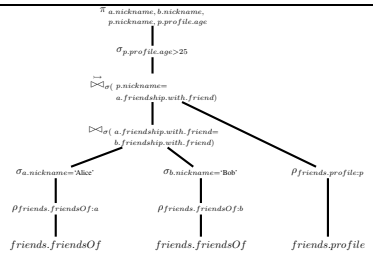
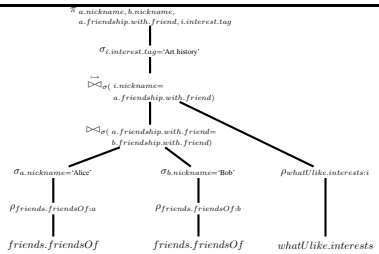
Hybrid query		#DSs	#BindJoins	#Joins	#Filters	#BlockingProjections	#Projections	complexity
HQ1: *Where is Bob ?*		1	0	0	0	0	1	-
HQ2: *Which are the interests of Alice with a score above 0.8 ?*		1	0	0	1	0	1	
HQ3: *Where are the friends of Alice ?*		2	1	0	0	1	1	
HQ4: *Which are the common friends of Alice and Bob ?*		2	0	1	0	1	1	
HQ5: *Which are the common interests of Alice and Bob with a score above 0.8 ?*		2	0	1	1	1	1	
HQ6: *Which of the common friends of Alice and Bob are more than 25 years ?*		3	1	1	1	2	1	
HQ7: *Which of the common friends of Alice and Bob are interested in art history ?*		3	1	1	1	1	2	+

Table 6.4 – Hybrid queries for experiments

The following commands generate the control-flow and data-flow search spaces.

1. `java -jar hyqoztb.jar -derive -hqsignature HQ_S -outputfile DER_OFILE`
2. `java -jar hyqoztb.jar -generate -inputfile DER_OFILE -outputfile CF_GEN_OFILE -controlflow`
3. `java -jar hyqoztb.jar -generate -inputfile DER_OFILE -outputfile DF_GEN_OFILE -dataflow`

The first command derives the data transformation functions and stores them in `DER_OFILE`. The second and third commands take the data transformation functions in `DER_OFILE` and invokes QWGenerator that produce the control-flow and data-flow search spaces respectively. The resulting search spaces are stored in `CF_GEN_OFILE` and `DF_GEN_OFILE` respectively. The chart in Figure 6.15 shows the search space sizes for every hybrid query.

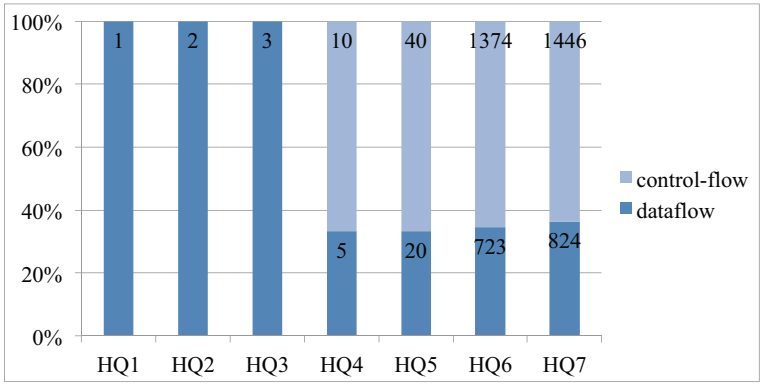


Figure 6.15 – Control-flow and data-flow search spaces

The first three hybrid queries `HQ1`, `HQ2`, `HQ3`; have the same data-flow and control-flow search spaces because the data dependencies among data transformation functions limit the generation rules to apply. The hybrid queries `HQ4`, `HQ5`, `HQ6`, and `HQ7`; have more flexible data dependencies and the control-flow search space is $\sim 3X$ than the data-flow search space.

6.4.4 Comparing cost estimation formulations

HyQoZTestbed enables to estimate the query workflows' costs using the build-time or run-time cost estimations implemented by QWWeighter. The following HyQoZTestbed commands are used for requesting the cost estimation of the query workflows in `CF_GEN_OFILE` generated previously in Section 6.4.3.

1. `java -jar hyqoztb.jar -weight -inputfile CF_GEN_OFILE -outputfile WBT_OFILE – buildtime`
2. `java -jar hyqoztb.jar -weight -inputfile CF_GEN_OFILE -outputfile WRT_OFILE – runtime`

HyQoZTestbed retrieves the query workflows from the input file `CF_GEN_OFILE`. For every query workflow, HyQoZTestbed requests the build-time and run-time cost estimations to the QWWeighter. In the case of the run-time cost estimation, QWWeighter uses a series of synthetic data statistics (*i.e.* dataset cardinalities, attribute cardinalities, attribute sizes) for simulating a query workflow execution.

In order to compare both cost estimations, we use the precision and recall measures [2]. The run-time cost is taken as benchmark. The query workflows reported by both functions are considered true positives (TP). The query workflows reported by the build-time function and do not reported by the run-time function are considered false positives (FP). The query workflows reported by the run-time function and do not reported by the build-time function are considered false negatives (FN). The precision is therefore given by $TP/(TP + FP)$ and the recall is given by $TP/(TP + FN)$.

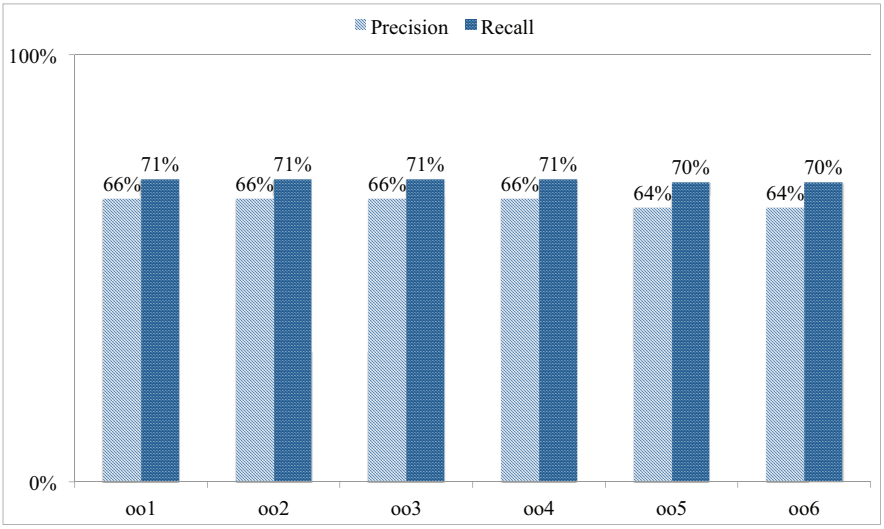


Figure 6.16 – Precision and recall of build-time cost estimation

Table 6.16 shows the charts for each hybrid query. Each chart represents the precision and recall found for every hybrid query with the different optimization objectives `OO1`,..,`OO6`. The charts show a tendence of the build-time cost estimation that suggests a liberal cost approximation. This is due to the absence of data statistics leading to find a high proportion of interesting results along with a high proportion of non-interesting results. On the other hand, the recall measure shows that an important fraction of the interesting results is not reported by the build-time function.

6.5 Conclusions

In this chapter we presented HyQoZ, the HYbrid Query OptimiZer. We introduced the hybrid query processing with the optimization performed by the components of HyQoZ. The HyQoZ components are described by REST interfaces and they exchange self-descriptive messages. The messages allow to instantiate different coordinations for implementing the hybrid query optimization. The interfaces and messages turn our optimization approach self-contained and enable the future work for defining optimization objectives for the HyQoZ execution itself.

2. Precision measures the retrieved results that are relevant. It is given by the relation between true positives and the total of retrieved results, *i.e.* $TP/(TP + FP)$. Recall measures the relevant results that are retrieved. It is given by the relation between true positives and the total of relevant results, *i.e.* $TP/(TP + FN)$.

For validating our proposal, we developed the testbed HyQoZTestbed that allows to characterize the hybrid query optimization implemented by HyQoZ components. We showed how HyQoZTestbed (1) generates synthetic hybrid queries, (2) accesses the QWGenerator for generating the search space of query workflows following a data-flow or control-flow, and (3) accesses the QWWeighter for estimating either the cost at build-time or the cost at run-time by means of a simulation using synthetic data statistics.

We compared the build-time and run-time cost estimations (*cf.* Section 5.1) using the precision and recall measures. Such measures give the proportion of interesting query workflows that are provided by the build-time formulation. The precision and recall are around 70% and 60% respectively which explains the absence of data-related parameters. In order to improve the accuracy of the cost estimation at build-time, it is necessary to incorporate data histograms adapted to the service-based environments. Nevertheless, our build-time cost estimation evokes the classical database heuristics in the absence of an algebra for rewriting expressions.

CHAPTER 7

Conclusions and perspectives

7.1 Main results and contributions

The hybrid query optimization problem addressed in this work combines data requirements and SLA contracts into the evaluation performed by query workflows. Our algorithm for generating query workflows aims to open the search space reflecting on the control-flow among activities. The idea behind this generation was to get a wider search space than the one provided by data-flow oriented approaches adopted by forerunner query optimizers. The cost estimation we proposed, allows to approximate the query workflow cost in the absence of data-related information, *e.g.* cardinality, selectivity, attribute size, stream data rate. For determining the dominance of query workflows w.r.t. the optimization objective of the SLA contract, we adopt a weighted distance metric that allows to treat with any combination of the potential cost attributes in service-based environments. The weighted distance metric is used as the scoring function in our top-k^{qw} algorithm that selects the solution space with the k most pertinent query workflows for the SLA contract.
The main results of this work are:

Control-flow based generation of query workflows. We proposed and implemented an algorithm for generating query workflows with sequential and parallel control-flows based on generation rules. The application of the generation rules produce well-constructed and equivalent query workflows with different grades of parallelism. The search space of query workflows is represented by a graph whose Hamiltonian paths lead to equivalent query workflows. This results in an NP-hard search as it traverse all the possible paths while some of them result in query workflows that are not interesting to satisfy the SLA contract.

Build-time cost function. We proposed a cost estimation that maps a query workflow to its cost formed by the attributes defined by the optimization objective of an SLA contract. Due to the absence of data-related parameters, we relaxed the cost estimation formulation in order to have a build-time cost estimation that evicts such an information. The build-time formulation leads to a partial order among query workflows and produces families of query workflows that get different costs at run-time. This approximation allows to estimate which query workflows are potentially optimal at run-time. Nevertheless, investigations are required to compute complex value data statistics that improve the cost accuracy at build-time.

Top-k^{qw} algorithm for selecting the best query workflows. We proposed top-k^{qw}, an adaptation of a top-k algorithm to obtain the best query workflows w.r.t. the optimization objective of an SLA contract. In order to get the score of query workflows we adopted a weighted distance metric that deals with conflicting cost attributes by applying penalties to get them closer or farther from the optimization objective. The optimization objective serves as a lower bound and avoids to make an exhaustive pair-wise comparison of the search space. Further searching strategies of mathematical optimization can be applied by leveraging on this bound. Besides, the weighted distance metric provides extensibility as it abstracts the cost attributes as *profit* and *loss* attributes. Thus, if new profit or loss attributes are incorporated, the top-k^{qw} algorithm stills working. The properties hold by the weighted distance metric (*i.e.* monotone, strict) allow to adapt other top-k algorithms such as No Random Access algorithms [UKl00, IBS08] that may result interesting for improving the pipelined choreography of HyQoZ (*cf.* Section 6.3.3).

7.2 Perspectives

The perspectives of this thesis arise from the key points of the hybrid query optimization.

1. **HyQoZ efficiency.** The complexity of the exhaustive generation of the query workflows behaves exponentially as shown in [LEVsZMC12]. Although the efficiency of the hybrid query optimization was out of the scope of this work, our framework gives the foundations for improuving the HyQoZ efficiency.

   — **Search space generation.** The query workflow generation is the hardest to perform among the optimization tasks. Our algorithm based on generation rules allows to incorporate several enhancements: (1) the cut of redundant paths into the search space graph, (2) the adoption of a join shape policy (*e.g.* left-deep, right-deep, bushy) depending on the query characteristics, and (3) the adoption of mathematical optimization strategies for traversing the Hamiltonian paths of the search space graph. Such enhancements have to be done regarding on the separation of the optimization aspects in order to keep the flexibility and portability of HyQoZ.

   — **Parametric optimization.** The optimization objective of the SLA contract defines a lower bound. Such a lower bound along with the separation of HyQoZ concerns, enable the execution of the optimization process in a pipelined fashion (*cf.* Section 6.3). Thus, the definition of parameters for adjusting the trade-off between efficiency and precision is an interesting problem. For instance, get a good enough query workflow with a time threshold, or the global optimal that implies more computation effort.

2. **Cost model.** In this thesis we have proposed a first rapprochement to the cost estimation for optimization purposes. In this sense, our optimization results are partially conclusive because our cost formulation is relaxed in the absence of data-related parameters. Next we enumerate research opportunities related to the cost model.

— **Concrete cost models for cost attribtues.** There are many cost attributes in service-based environments. For instance, quantitative measures of services [NLF99], network [ACH98], energy consumption [CS12, FP13], storage [LBN07, LBZ+11], privacy, and availability; and qualitative measures such as reliability and confidence of services, and data quality [MSV+02]. The adoption of concrete cost models for these attributes is necessary towards the generality of the HyQoZ cost model.

— **Meaningful statistics of data and QoS measures.** An interesting challenge is on the indexation of complex value statistics and QoS measures. The non-intrusive monitoring, the harvesting and selection of statistics, and the modeling of indexing structures for such an information are requirements towards an accurate cost model.

3. **Composite activities.** In this thesis, the activities were abstracted as black boxes regarding-less on the internal details such as (1) the implementation algorithm of the data transformation, and (2) the service operations used. This leads to an inter-operator optimization. Looking at state-full activities (*e.g.* activities combining data), they are service coordination themselves also modeled as query workflows with additional workflow operators such as iterations and control state. For such composite activities, it is necessary to investigate how to factorize or commute service invocations (*e.g.* hashing service, storage service) towards a intra-operator optimization.

   Besides, activities may be performed in parallel, either in independent or pipelined fashion. It depends on the logical relations with activities and on the logical dependencies within composite activities. In particular for independent parallelization, it is a challenge in the absence of knowledge about data organization. Moreover, it is also required to study how much is benefit the parallelization (*i.e.* scale-up, speed-up) regarding on the SLA contract.

4. **Service selection.** In this thesis we assumed static binding of an activity to a service instance. Nevertheless, the characteristics of the service-based environments invite to consider the dynamic binding of services. This enables the possibility to choose service instances whose QoS measures may improve the resulting query workflow cost. This leads to an assignation problem to be incorporated to the hybrid query optimization problem either in the search space generation or in a separate stage.

5. **Scheduling of query workflows and adaptability.** Hardware resources can participate in the query workflow execution by hosting activities in a Platform-as-a-Service fashion. This leads to the scheduling problem with the consideration of the costs related to the use of such resources. In particular, queries over stream data services require long-running query workflows. As long as a query workflow is being executed everything is subject to change (*e.g.* service instances, hardware resources,network) and thus the query workflow cost may fluctuate. This dynamics turns the query workflow cost a function of time and it introduces the problem of adaptability to continue meeting the SLA contract. Such an adaptation implies either the modification of the query workflow, the selection of different hardware resources, or the selection of different service instances. These problems are analogous to the site selection in peer-to-peer based query evaluation, and the

physical-operator selection in classical query optimization. In any case, this implies to be aware of the candidate query workflows plans for continuing satisfying SLA contracts.

6. **SLA design.** The design of the SLA contract [FSG+11, MSM09, Sch00] is a cognitive and negotiation challenge. The provision of tools for such a purpose is an interesting problem. For instance, *what-if* queries over histograms of query workflow executions can improve the SLA negotiations during the design. Besides, the derivation of the optimization objective from the SLA contract is an open problem. It is an instance of the lower-bound computation in minimization problems. Such derivation should consider a multi-query environment and the contention of service instances and network.
7. **Implementation of an extensible hybrid query processor with HyQoZ.** The implementation of an extensible hybrid query processor with HyQoZ should consider the possibilities to (1) include the participation of several service instances, (2) modify the optimization strategy regarding on the hybrid query signature and SLA characteristics, (3) adapt the query workflow execution as the service-based environment changes, and (4) incorporate QoS expectations of the hybrid query processing itself.

Looking further, the HyQoZ principle to combine the provision of data, and computing capabilities with the satisfaction of SLA contracts; can be applied beyond the hybrid query processing.

8. **Keyword-based data services.** The access of structured data requires technical knowledge on data-structures and query languages that makes difficult the democratization of such data. Keyword-based data services may facilitate the access to structured data. It is an interesting to investigate the problem of automatic construction of query plans for accessing structured data by means of keyword search services. The access must to be organized and measured towards the efficiency of query plans regarding on SLA contracts.
9. **HyQoZ approach for other data-centric applications.** The inclusion of the SLA notion as a combination of as many cost attributes as users require, along with our extensible optimization strategy, allows to apply our approach to other data-centric applications. It is the case of BigData where processing units are user-defined functions following parallel programming paradigms such as MapReduce[DG08], PACT[AEH11], or SCOPE [Jin10]. Scientific workflows is also an application area where our ideas may be applied with a quality of data perspective. In any case, our control-flow approach for generating execution plans can be applied to meet QoS requirements.
10. **Infrastructure for a collaborative and pervasive computing community.** The use of nowadays computing resources induces costs and QoS considerations. An alternative is to adopt the available cloud-based infrastructures assuming the economical costs and the technology adoption. Another alternative is to exploit the huge computing power available on users' computing devices. We believe that the massive collection of computing devices can be put together in a collaborative and pervasive computing community that exchange computing goods. With such a perspective, the users can participate through their home and mobile devices for exchanging goods into open data markets, query markets, computing markets, and storage markets.

Bibliography

[AAB+09] Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, Surajit Chaudhuri, AnHai Doan, Daniela Florescu, Michael J. Franklin, Hector Garcia-Molina, Johannes Gehrke, Le Gruenwald, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, Henry F. Korth, Donald Kossmann, Samuel Madden, Roger Magoulas, Beng Chin Ooi, Tim O'Reilly, Raghu Ramakrishnan, Sunita Sarawagi, Michael Stonebraker, Alexander S. Szalay, and Gerhard Weikum. The Claremont report on database research. *Communications of the ACM*, 52(6):56–65, September 2009.

[ACH98] Cristina Aurrecoechea, Andrew T. Campbell, and Linda Hauw. A survey of QoS architectures. *Multimedia Systems*, 6(3):138–151, May 1998.

[ACK+11] Anastasia Ailamaki, Michael J. Carey, Donald Kossman, Steve Loughran, and Volker Markl. Information Management in the Cloud (Dagstuhl Seminar 11321). *Dagstuhl Reports*, 1(8):1–28, 2011.

[ACP96] S. Adali, KS Candan, and Y. Papakonstantinou. Query caching and optimization in distributed mediator systems. *ACM SIGMOD*, pages 137–146, 1996.

[AEH11] Alexander Alexandrov, Stephan Ewen, and Max Heimel. MapReduce and PACT - Comparing Data Parallel Programming Models. In Holger Schwarz Theo H"arder, Wolfgang Lehner, Bernhard Mitschang, Harald Sch"oning, editor, *BTW*, pages 25–44, Kaiserslautern, Germany, 2011. LNI.

[AH00] R. Avnur and J.M. Hellerstein. Eddies: Continuously adaptive query processing. *ACM SIGMOD Record*, 29(2):261–272, 2000.

[AK98] Jos'e Luis Ambite and Craig A. Knoblock. Flexible and Scalable Query Planning in Distributed and Heterogeneous Environments. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 3–10, 1998.

[AKHLS13] Samer Al-Kiswany, Hakan Hac$backslash$ig$backslash$"um$backslash$"u$backslash$cs, Ziyang Liu, and Jagan Sankaranarayanan. Cost Exploration of Data Sharings in the Cloud. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 601—-612, Genoa, Italy, 2013. ACM New York, NY, USA.

[BAC+90] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez. Prototyping Bubba, a highly parallel database system. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):4–24, March 1990.

[Bat86] Don S. Batory. Extensible Cost Models and Query Optimization in GENESIS. *IEEE Database Eng. Bull.*, 9(4):30–36, 1986.

[Bat87] D Batory. Principles of Database Management System Extensibility. *IEEE Database Engineering Bull.*, 10(2):40–46, 1987.

[BCCR08] Daniele Braga, A Campi, S Ceri, and A Raffio. Joining the results of heterogeneous search engines. *Information Systems*, 33(7-8):658—-680, 2008.

[BCD08] Daniele Braga, S Ceri, and F Daniel. Optimization of multi-domain queries on the web. *VLDB*, 2008.

[BCV08] Christophe Bobineau, Christine Collet, and Tuyet-Trinh Vu. A strategy to develop adaptive and interactive query brokers. In *Proceedings of the 2008 International Symposium on Database Engineering and Applications*, pages 237—-247, New York, New York, USA, 2008. ACM New York, NY, USA.

[BE77] M. W. Blasgen and K. P. Eswaran. Storage and Access in Relational Data Bases. *IBM Systems Journal*, 16(4):363—-377, 1977.

[BEE+13] Jean Bacon, David Evans, David M. Eyers, Matteo Migliavacca, Peter Pietzuch, and Brian Shand. Enforcing End-to-End Application Security in the Cloud (Big Ideas Paper). In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, pages 293–312, Bangalore, India, 2013. Springer-Verlag, Berlin, Heidelberg.

[BGLJ10] Nicolas Bruno, C'esar A. Galindo-Legaria, and Milind Joshi. Polynomial Heuristics for Query Optimization. In *ICDE*, pages 589–600, 2010.

[BHS11] Magdalena Balazinska, Bill Howe, and Dan Suciu. Data Markets in the Cloud: An Opportunity for the Database Community. *VLDB*, 4(12):1482–1485, 2011.

[BKK+01] R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, S. Kreutz, A. and Seltzsam, and K. Stocker. ObjectGlobe: Ubiquitous query processing on the Internet. *The VLDB Journal*, 10(1):48–71, 2001.

[BN08] Jens Bleiholder and Felix Naumann. Data fusion. *ACM Computing Surveys*, 41(1):1–41, December 2008.

[BND+04] B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for Web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, May 2004.

[BS05] Christos Bouras and Afrodite Sevasti. Service level agreements for DiffServ-based services' provisioning. *Journal of Network and Computer Applications*, 28(4):285–302, November 2005.

[BT08] E. B"orger and Bernhard Thalheim. Modeling workflows, interaction patterns, web services and business processes: The asm-based approach. In *Abstract State Machines, B and Z*, pages 24–38. Springer-Verlag, Berlin, Heidelberg, 2008.

[Car04] J Cardoso. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281–308, April 2004.

[CBB+04] Christine Collet, Khalid Belhajjame, Gilles Bernot, Christophe Bobineau, Gennaro Bruno, Beatrice Finance, Fabrice Jouanot, Zoubida Kedad, David Laurent, Fariza Tahi, and Others. Towards a mediation system framework for transparent access to largely distributed sources. *Semantics of a Networked World*, pages 65–78, 2004.

[CCDS04] Fabio Casati, Malu Castellanos, Umesh Dayal, and Ming-Chien Shan. Probabilistic, context-sensitive, and goal-oriented service selection. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 316–321, New York, NY, USA, 2004. ACM.

[CD96] Kwok Chung T. and Weld Daniel S. Planning to gather information. *PROCEEDINGS OF THE NATIONAL ldots*, 1996.

[CGMH+94] S. Chawathe, Hector Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogenous information sources. In *Proceedings of IPSJ Conference*, pages 7—-18, Tokyo, Japan, 1994.

[CHS+95] M.J. Carey, L.M. Haas, P.M. Schwarz, M. Arya, W.F. Cody, R. Fagin, M. Flickner, A.W. Luniewski, W. Niblack, D. Petkovic, and Others. Towards heterogeneous multimedia information systems: The Garlic approach. In *Research Issues in Data Engineering, 1995: Distributed Object Management, Proceedings. RIDE-DOM'95. Fifth International Workshop on*, volume 95, pages 124–131. IEEE, 1995.

[CM77] AK Chandra and PM Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM ldots*, pages 77–90, Boulder, Colorado, USA, 1977. ACM Press.

[CM08] Andrea Cal'i and Davide Martinenghi. Querying Data under Access Limitations. In *ICDE*, volume 1, pages 50–59, 2008.

[CM09] Byung-Gon Chun and Petros Maniatis. Augmented Smartphone Applications Through Clone Cloud Execution. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, pages 8–13, Monte Verita, Switzerland, 2009. USENIX Association, Berkeley, CA, USA.

[Col12] Maria Colgan. Understanding Optimizer Statistics, 2012.

[CS93] Surajit Chaudhuri and K Shim. Query optimization in the presence of foreign functions. In *Proceedings of the 19th International Conference on Very Large Data Bases*, pages 529—-542. Morgan Kaufmann Publishers Inc., 1993.

[CS95] Surajit Chaudhuri and K. Shim. An overview of cost-based optimization of queries with aggregates. *Data Engineering Bulletin*, 18(3):3–9, 1995.

[CS96] Surajit Chaudhuri and Kyuseok Shim. Optimization of Queries with User-defined Predicates. In *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 87—-98, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[CS12] Cinzia Cappiello and Fabio A. Schreiber. Experiments and analysis of quality and Energy-aware data aggregation approaches in WSNs. In *10th International Workshop on Quality in Databases QDB*, pages 1–8, Istanbul, Turkey, 2012.

[CV11] Victor Cuevas-Vicenttin. *Evaluation of Hybrid Queries Based on Service Coordination*. Thèse de Doctorat, Grenoble Institute of Technology, 2011.

[CVVSC12] Victor Cuevas-Vicenttin, Genoveva Vargas-Solar, and Christine Collet. Evaluating Hybrid Queries through Service Coordination in HYPATIA. In *EDBT*, pages 602–605, 2012.

[CvVSCB09] Victor Cuevas-vicenttin, Genoveva Vargas-Solar, Christine Collet, and Paolo Bucciol. Efficiently Coordinating Services for Querying Data in Dynamic Environments. In *Proceedings of the 10th Mexican International Conference on Computer Science (ENC'09)*, Mexico City, Mexico, 2009. IEEE Computer Society.

[CYW96] Ming-syan Chen, Philip S. Yu, and Kun-lung Wu. Optimization Of Parallel Execution For Multi-Join Queries. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):416—-428, June 1996.

[DG92] D. DeWitt and Jim Gray. Parallel database systems: The future of high performance database processing. *Communications of the ACM*, 36(6):1–26, 1992.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce : Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):1–13, 2008.

[DGS+90] D.J. DeWitt, S. Ghandeharizadeh, D.a. Schneider, a. Bricker, H.-I. Hsiao, and R. Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, March 1990.

[DL97] OM Duschka and AY Levy. Recursive plans for information gathering. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997.

[DLP03] Asit Dan, Heiko Ludwig, and Giovanni Pacifici. Web service differentiation with service level agreements. Rapport technique, IBM Software group, May 2003.

[DSL+08] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: The Montage example. *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, November 2008.

[DT07] David DeHaan and Frank Wm. Tompa. Optimal Top-Down Join Enumeration. In *Proceedings of the 2007 ACM SIGMOD ldots*, pages 785—-796, Beijing, China, 2007. ACM.

[Ewe12] Stephan Ewen. Spinning Fast Iterative Data Flows. *VLDB*, 5(11):1268–1279, 2012.

[Fag99] Ronald Fagin. Combining fuzzy information from multiple systems. *ldots SIGART symposium on Principles of database systems*, 58(1):83–99, February 1999.

[FDBP01] Marie-Christine Fauvet, Marlon Dumas, Boualem Benatallah, and Hye-Young Paik. Peer-to-peer Traced Execution of Composite Services. In Fabio Casati, Dimitrios Georgakopoulos, and Ming-Chien Shan, editors, *Second International Workshop on Technologies for E-Services (TES '01)*, pages 103–117, London, UK, 2001. Springer-Verlag.

[Feg98] L Fegaras. A New Heuristic for Optimizing Large Queries. In *In 9th International Conference, DEXA'98*, pages 726—-735. Springer-Verlag, 1998.

[Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. Thèse de Doctorat, University of California, Irvine, 2000.

[FJK96] MJ Franklin, BT J'onsson, and Donald Kossmann. Performance Tradeoffs for Client-Server Query Processing onsson. *ACM SIGMOD Record*, pages 1–12, 1996.

[FK09] Daniela Florescu and Donald Kossmann. Rethinking cost and performance of database systems. *ACM SIGMOD Record*, 38(1):43, June 2009.

[FLMS99] Daniela Florescu, Alon Levy, Ioana Manolescu, and Dan Suciu. Query Optimization in the Presence of Limited Access Patterns. In *SIGMOD'99: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 311–322, 1999.

[FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, June 2003.

[FP13] AM Ferreira and Barbara Pernici. Using Intelligent Agents to Discover Energy Saving Opportunities within Data Centers. In *Second International Workshop on Requirements Engineering for Sustainable Systems*, pages 1–8, 2013.

[FSG+11] Ganna Frankovaa, Magali S'eguranb, Florian Gilcherb, Slim Trabelsib, J"org D"orflingerc, and Marco Aiellod. Deriving business processes with service level agreements from early requirements. *Journal of Systems and Software*, 84(8):1351–1363, 2011.

[GD87] Goetz Graefe and David J. DeWitt. The EXODUS optimizer generator. *ACM SIGMOD Record*, 16(3):160–172, December 1987.

[GDQ92] Shahram Ghandeharizadeh, David J. DeWitt, and Waheed Qureshi. A performance analysis of alternative multi-attribute declustering strategies. In *ACM SIGMOD Record*, pages 29–38. ACM Press, 1992.

[GG03] Sudipto Guha and D Gunopoulos. Efficient Approximation of Optimization Queries Under Parametric Aggregation Constraints. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, pages 778–789, Berlin, Germany, 2003. VLDB Endowment.

[GHK92] Sumit Ganguly, Waqar Hasan, and Ravi Krishnamurthy. Query optimization for parallel execution. In Michael Stonebraker, editor, *1992 ACM SIGMOD international conference on Management of data (SIGMOD '92)*, pages 9–18, New York, NY, USA, 1992. ACM.

[GI96] Minos N. Garofalakis and Yannis E. Ioannidis. Multi-dimensional resource scheduling for parallel queries. *ACM SIGMOD Record*, pages 365–376, 1996.

[GI97] Minos N. Garofalakis and Yannis E. Ioannidis. Parallel Query Scheduling and Optimization with Time- and Space-Shared Resources. *VLDB '97*, pages 296–305, 1997.

[GLR97] Cesar Galindo-Legaria and Arnon Rosenthal. Outerjoin Simplification and Reordering for Query Optimization. *ACM Transactions on Database Systems*, 22(1):43–74, 1997.

[GM93] G. Graefe and W.J. McKenna. The Volcano optimizer generator: Extensibility and efficient search. In *Data Engineering, 1993. Proceedings. Ninth International Conference on*, pages 209–218. IEEE, 1993.

[GMUW02] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems – The complete book*. Prentice Hall Press, second edi edition, 2002.

[GNnM+96] Manuel Barrena Garcia, Juan Hern'andez Nu nez, Juan Miguel Martinez, Antonio Polo Marquez, Pedro de Miguel, and Manuel M. Nieto Rodriguez. Multi-dimensional Declustering Methods for Parallel Database Systems. In Yves Bouge, Luc and Fraigniaud, Pierre and Mignotte, Anne and Robert, editor, *Euro-Par*, pages 866–871. Springer-Verlag, 1996.

[Gou09] Anastasios Gounaris. A Vision for Next Generation Query Processors and an Associated Research Agenda. In *Proceedings of the 2Nd International Conference on Data Management in Grid and Peer-to-Peer Systems*, pages 1–11, Linz, Austria, 1009. Springer-Verlag, Berlin, Heidelberg.

[Gra93] Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys (CSUR)*, 25(2), 1993.

[Gra95] Goetz Graefe. The Cascades framework for query optimization. *Data Engineering Bulletin*, 18(3):19–29, 1995.

[GSDB11] Ravindra Guravannavar, S. Sudarshan, Ajit a. Diwan, and Ch. Sobhan Babu. Which sort orders are interesting ? *The VLDB Journal*, 21(1):145–165, June 2011.

[GST83] Nathan Goodman, Oded Shmueli, and Y. C. Tay. GYO reductions, canonical connections, tree and cyclic schemas and tree projections. In *Proceedings of the 2Nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 267—-278, Atlanta, Georgia, March 1983. ACM Press.

[GW00] Roy Goldman and Jennifer Widom. WSQ/DSQ: A Practical Approach for Combined Querying of Databases and the Web. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 285–296, 2000.

[Had09] Marc Hadley. Web Application Description Language, 2009.

[HCL+90] L. M. Haas, W. Chang, G. M. Lohman, J. McPherson, P. F. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M. J. Carey, and E. Shekita. Starburst Mid-Flight: As the Dust Clears. *IEEE Trans. on Knowl. and Data Eng.*, 2(1):143—-160, 1990.

[Hel94] Joseph M. Hellerstein. Practical predicate placement. *ACM SIGMOD Record*, 23(2):325–335, 1994.

[HFC+00] JM Hellerstein, MJ Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M.A. Shah. Adaptive query processing: Technology in evolution. *Bulletin of the Technical Committee on*, page 7, 2000.

[HFLP89] L. M. Haas, J. C. Freytag, G. M. Lohman, and H. Pirahesh. Extensible query processing in starburst. *Proceedings of the 1989 ACM SIGMOD international conference on Management of data - SIGMOD '89*, pages 377–388, 1989.

[HKWY97] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing queries across diverse data sources. *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 276–285, 1997.

[HSMR09] Stavros Harizopoulos, Mehul A. Shah, Justin Meza, and Parthasarathy Ranganathan. Energy Efficiency : The New Holy Grail of Data Management Systems Research. In *In Proceedongs of the Fourth Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, USA, 2009. www.cidrdb.org.

[HSS+11] Martin Hirzel, Robert Soul'e, Scott Schneider, Bugra Gedik, Robert Grimm, and Yorktown Heights. A Catalog of Stream Processing Optimizations. Rapport technique, IBM Research, 2011.

[IBS08] Ihab F. Ilyas, George Beskales, and Mohamed a. Soliman. A survey of top- k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4):1–58, October 2008.

[IK84] Toshihide Ibaraki and Tiko Kameda. On the optimal nesting order for computing N-relational joins. *ACM Transactions on Database Systems*, 9(3):482–502, August 1984.

[IK90] Y. E. Ioannidis and Younkyung Kang. Randomized algorithms for optimizing large join queries. *ACM SIGMOD Record*, 19(2):312–321, May 1990.

[IK91] Yannis E. Ioannidis and Younkyung Cha Kang. Left-deep vs. bushy trees: an analysis of strategy spaces and its implications for query optimization. *ACM SIGMOD Record*, 20(2):168–177, 1991.

[IW87] Yannis E. Ioannidis and Eugene Wong. Query Optimization by Simulated Annealing. *SIGMOD Record*, 16(3):9—-22, 1987.

[JE07] Diane Jordan and John Evdemon. Web Services Business Process Execution Language Version 2 . 0. Rapport technique April, OASIS, 2007.

[Jin10] Ronnie Chaiken Jingren Zhou, Per-Ake Larson. Incorporating partitioning and parallel plans into the SCOPE optimizer. In *ICDE*, pages 1060–1071, Long Beach, CA, USA, 2010. Ieee.

[Jin12] Darren Shakib Jingren Zhou, Nicolas Bruno, Ming-Chuan Wu, Per-Ake Larson, Ronnie Chaiken. SCOPE: parallel databases meet MapReduce. *VLDB*, 21(5):611–636, June 2012.

[JMG05] Michael C. Jaeger, Gero M$backslash$"uhl, and Sebastian Golze. QoS-aware composition of web services: An evaluation of selection algorithms. In *On the Move to Meaningful Internet Systems*, pages 646–661, Berlin, Heidelberg, 2005. Springer-Verlag.

[JRGM04] Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Muhl. QoS Aggregation in Web Service Compositions. In *EDOC '04 Proceedings of the Enterprise Distributed Object Computing Conference*, pages 149 – 159. IEEE Computer Society Washington, DC, USA, 2004.

[KA01] Ken Kennedy and John R. Allen. *Optimizing compilers for modern architectures: a dependence-based approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[KB12] Mayuresh Kunjir and PK Birwa. Peak Power Plays in Database Engines. In *EDBT*, 2012.

[KBZ86] Ravi Krishnamurthy, Haran Boral, and Carlo Zaniolo. Optimization of Nonrecursive Queries. In *Proceedings of the 12th International Conference on Very Large Data Bases*, pages 128–137. Morgan Kaufmann Publishers Inc., 1986.

[KCC+11] Sudarshan Kadambi, Jianjun Chen, Brian F. Cooper, David Lomax, Raghu Ramakrishnan, Adam Silberstein, Erwin Tam, and Hector Garcia-Molina. Where in the world is my data ? *VLDB*, 4(11):1040–1050, 2011.

[KHAK09] Tim Kraska, Martin Hentschel, Gustavo Alonso, and Donald Kossmann. Consistency Rationing in the Cloud: Pay only when it matters. *Proceedings of the VLDB Endowment*, 2(1):253—-264, 2009.

[KIT10] Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Querying data provenance. In *Proceedings of the 2010 international conference on Management of data - SIGMOD '10*, pages 951—-962, Indianapolis, Indiana, USA, 2010. ACM Press.

[Kon11] Andreas Konstantinidis. Multi-objective query optimization in smartphone social networks. In *Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management*, pages 27–32. IEEE Computer Society Washington, DC, USA, 2011.

[Kos00] Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys (CSUR)*, 32(4):422–469, December 2000.

[KP09] Yannis Katsis and Yannis Papakonstantinou. View-based Data Integration. In *Encyclopedia of Database Systems*, pages 3332–3339. Springer US, 2009.

[KP13] Georgios Kellaris and Stavros Papadopoulos. Practical Differential Privacy via Grouping and Smoothing. *Proceedings of the VLDB Endowment*, 6(5):301—-312, 2013.

[KS00] Donald Kossmann and Konrad Stocker. Iterative Dynamic Programming: A New Class of Query Optimization Algorithms. *ACM Transactions on Database Systems (ldots*, 25(1):43—-82, 2000.

[KSTI11] Herald Kllapi, Eva Sitaridi, Manolis M. Tsangaris, and Yannis Ioannidis. Schedule optimization for data processing flows on the cloud. In *Proceedings of the 2011 international conference on Management of data - SIGMOD '11*, pages 289–300, Athens, Greece, 2011. ACM New York, NY, USA.

[KTG06] V. Santhosh Kumar, M. J. Thazhuthaveetil, and R. Govindarajan. Exploiting Programmable Network Interfaces for Parallel Query Execution in Workstation Clusters. In *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, Rhodes Island, Greece, 2006. IEEE Computer Society Washington, DC, USA.

[KTV97] Olga Kapitskaia, Anthony Tomasic, and Patrick Valduriez. Dealing with Discrepancies in Wrapper Functionality. In Jean Ferri'e, editor, *Base de Donn'ees Avanc'ees*, 1997.

[KUB+12] Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. Query-based data pricing. In *Proceedings of the 31st Symposium on Principles of Database Systems*, pages 167—-178, Scottsdale, Arizona, USA, 2012. ACM New York, NY, USA.

[KUB+13] Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. Toward practical query pricing with QueryMarket. In *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*, page 613, New York, New York, USA, 2013. ACM Press.

[LBN07] Kien Le, Ricardo Bianchini, and Thu D. Nguyen. A cost-effective distributed file service with QoS guarantees. In *ACM/IFIP/USENIX 2007 International Conference on Middleware*, pages 223–243, New York, New York, USA, 2007. Springer-Verlag New York, Inc.

[LBZ+11] Kien Le, Ricardo Bianchini, Jingru Zhang, Yogesh Jaluria, Jiandong Meng, and Thu D. Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 223–243, New York, New York, USA, 2011. ACM Press.

[LC00] Chen Li and E Chang. Query Planning with Limited Source Capabilities. In *Data Engineering, 2000. Proceedings. 16th ldots*, pages 401–426. IEEE Computer Society, 2000.

[LC01a] Chen Li and Edward Chang. Answering queries with useful bindings. *ACM Transactions on Database Systems*, 26(3):313–343, September 2001.

[LC01b] Chen Li and Edward Y. Chang. On answering queries in the presence of limited access patterns. In Jan Van den Bussche and Victor Vianu, editors, *Proceedigs of the International Conference on Database Theory*, pages 219–233. Springer, 2001.

[LEVsZMC12] Carlos Manuel L'opez-Enr'iquez, Genoveva Vargas-solar, Jos'e-Luis Zechinelli-Martini, and Christine Collet. Hybrid query plan generation. In *LANMR 2012*, pages 117–128, Mexico City, Mexico, 2012.

[Ley01] Frank Leymann. Web services flow language (wsfl 1.0). Rapport technique May, IBM Software Group, 2001.

[Li03] Chen Li. Computing Complete Answers to Queries in the Presence of Limited Access Patterns. *The VLDB Journal*, 12(3):211—-227, 2003.

[LNZ04] Yutu Liu, Anne H. Ngu, and Liang Z. Zeng. QoS computation and policing in dynamic web service selection. *Alternate track papers & posters of the 13th international conference on World Wide Web - WWW Alt. '04*, page 66, 2004.

[LP09] Willis Lang and Jignesh M. Patel. Towards Eco-friendly Database Management Systems. In *In Proceedongs of the Fourth Biennial Conference on Innovative Data Systems Research*. www.cidrdb.org, 2009.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 251—-262. Morgan Kaufmann Publishers Inc., 1996.

[LSW+12] Zhipiao Liu, Qibo Sun, Shangguang Wang, Hua Zou, and Fangchun Yang. Profit-driven Cloud Service Request Scheduling Under SLA Constraints. *Journal Parallel Distributed Computing*, 72(4):591–602, 2012.

[Mad96] O. Madani. Efficient information gathering on the Internet. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 234—-. IEEE Computer Society Washington, DC, USA, October 1996.

[Mah10] Sunita Mahajan. A Survey of Issues of Query Optimization in Parallel Databases. In *International Conference & Workshop on Emerging Trends in Technology*, pages 32–37, 2010.

[MBHT96] William J. McKenna, Louis Burger, Chi Hoang, and Melissa Truong. EROC: A Toolkit for Building NEATO Optimizers. In T. M. Vijayaraman And, Alejandro P. Buchmann, C. Mohan And, and Nandlal L. Sarda, editors, *Proceeding VLDB '96 Proceedings of the 22th International Conference on Very Large Data Bases*, pages 111–121, Mumbai (Bombay), India, 1996. Morgan Kaufmann.

[ME92] Priti Mishra and Margaret H. Eich. Join Processing in Relational Databases. *ACM Computing Surveys*, 24(1):63—-113, 1992.

[Men02] Daniel A. Menasc$backslash$'e. QoS issues in Web services. *IEEE Internet Computing*, 6(6):72–75, November 2002.

[ML86] Lothar F. Mackert and Guy M. Lohman. R* optimizer validation and performance evaluation for local queries. In *Proceedings of the 1986 ACM SIGMOD international*

conference on Management of data - SIGMOD '86, pages 84—-95, Washington, D.C., USA, 1986. ACM New York, NY, USA.

[ML88] Lothar F. Mackert and Guy M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In Wesley W. Chu, Georges Gardarin, Setsuo Ohsuga, and Yahiko Kambayashi, editors, *Readings in database systems*, pages 149–159, San Francisco, CA, USA, 1988. Morgan Kaufmann Publishers Inc.

[MN06] Guido Moerkotte and Thomas Neumann. Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, pages 930—-941, Seoul, Korea, 2006. VLDB Endowment.

[MN08] Guido Moerkotte and Thomas Neumann. Dynamic programming strikes back. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 539–552, New York, New York, USA, 2008. ACM Press.

[Mor64] William Thomas Morris. *The Analysis of Management Decisions*. R.D. Irwin, 1964.

[MSM09] FT Marques, JP Sauv'e, and JAB Moura. SLA design and service provisioning for outsourced services. *Journal of Network and Systems ldots*, 17(1-2):73–90, 2009.

[MSV+02] Massimo Mecella, Monica Scannapieco, Antonino Virgillito, Roberto Baldoni, Tiziana Catarci, and Carlo Batini. Managing data quality in cooperative information systems. *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, pages 486–502, 2002.

[Nas04] Bertram Nash, Alan and Lud"ascher. Processing first-order queries under limited access patterns. In *Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 307 —- 318, Paris, France, 2004. ACM Press.

[NK01] Zaiqing Nie and Subbarao Kambhampati. Joint optimization of cost and coverage of query plans in data integration. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, volume 2001, pages 223—-230, Atlanta, Georgia, USA, 2001. ACM New York, NY, USA.

[NL04] Alan Nash and Bertram Lud"ascher. Processing Unions of Conjunctive Queries with Negation under Limited Access Patterns. In *Advances in Database Technology - EDBT 2004*, pages 422–440. Springer Berlin Heidelberg, 2004.

[NLF99] Felix Naumann, U. Leser, and J.C. Freytag. Quality-driven integration of heterogeneous information systems. In *PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES*, pages 447–458. Citeseer, 1999.

[NSS86] Surendra Nahar, Sartaj Sahni, and Eugene Shragowitz. Simulated Annealing and Combinatorial Optimization. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pages 293—-299, Las Vegas, Nevada, USA, 1986. IEEE Press.

[OEtH02] Justin O'Sullivan, David Edmond, and Arthur ter Hofstede. What's in a Service ? Towards Accurate Description of Non-Functional. *DISTRIBUTED AND PARALLEL DATABASES*, 12:117–133, 2002.

[OL90] Kiyoshi Ono and Guy M. Lohman. Measuring the complexity of join enumeration in query optimization. In *VLDB '90 Proceedings of the 16th International Conference on Very Large*, pages 314–325. Morgan Kaufmann Publishers Inc., 1990.

[PGLK97] A Pellenkoft, CA Galindo-Legaria, and Martin Kersten. The complexity of transformation-based join enumeration. *VLDB*, pages 306–315, 1997.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 251–260. IEEE Computer Society Washington, DC, USA, 1995.

[PHH92] Hamid Pirahesh, Joseph M. Hellerstein, and Waqar Hasan. Extensible/rule based query rewrite optimization in Starburst. In Michael Stonebraker, editor, *1992 ACM SIGMOD international conference on Management of data (SIGMOD '92)*, pages 39–48, New York, NY, USA, 1992. ACM, New York, NY, USA.

[PY00] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of Web sources. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 86—-. IEEE Computer Society Washington, DC, USA, November 2000.

[PY01] C.H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 52–59. ACM, 2001.

[QRLCV11] Jorge-Arnulfo Quian'e-Ruiz, Philippe Lamarre, Sylvie Cazalens, and Patrick Valduriez. Scaling Up Query Allocation in the Presence of Autonomous Participants. In *Proceedings of the 16th International Conference on Database Systems for Advanced Applications: Part II*, pages 210—-224, Hong Kong, China, 2011. Springer-Verlag, Berlin, Heidelberg.

[RGL90] Arnon Rosenthal and Cesar Galindo-Legaria. Query graphs, implementing trees, and freely-reorderable outerjoins. *ACM SIGMOD Record*, 19(2):291–299, May 1990.

[RH86] A. Rosenthal and P. Helman. Understanding and Extending Transformation-Based Optimizers. *Database eeri*, 9(4):44, 1986.

[RLL+01] Jun Rao, Bruce G. Lindsay, Guy M. Lohman, Hamid Pirahesh, and David E. Simmen. Using EELs, a Practical Approach to Outerjoin and Antijoin Reordering. In Dimitrios Georgakopoulos and Alexander Buchmann, editors, *Proceedings of the 17th International Conference on Data Engineering*, pages 585–594, Heidelberg, Germany, 2001. IEEE Computer Society Washington, DC, USA.

[ROH99] Mary Tork Roth, Fatma Ozcan, and Laura M. Haas. Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 599—-610. Morgan Kaufmann Publishers Inc., 1999.

[RR82] Arnon Rosenthal and David Reiner. An architecture for query optimization. In *Proceedings of the 1982 ACM SIGMOD ldots*, pages 246–255, Orlando, Florida, 1982. ACM.

[RS97] Mary Tork Roth and Peter M. Schwarz. Don't Scrap It, Wrap It ! A Wrapper Architecture for Legacy Data Sources. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 266—-275. Morgan Kaufmann Publishers Inc., 1997.

[RSU95] A. Rajaraman, Y. Sagiv, and J.D. Ullman. Answering queries using templates with binding patterns. In *Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 105–112. ACM, 1995.

[RU93] Raghu Ramakrishnan and Jeffrey D. Ullman. A Survey of Research on Deductive Database. *Journal of Logic Programming*, 23:125—-149, 1993.

[SA80] Patricia G. Selinger and Michel E. Adiba. Access Path Selection in Distributed Database Management Systems. In Deen and Hammersly, editor, *Proceedings of the International Conference on Data Bases*, pages 204–215, 1980.

[SAB+95] V.S. Subrahmanian, Sibel Adali, Anne Brink, J.ames J. Lu, Adil Rajput, Timothy J. Rogers, and Charles Ward Robert Ross. HERMES: A Heterogeneous Reasoning and Mediator System, 1995.

[SAC+79] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *1979 ACM SIGMOD international Conference on Management of Data*, pages 23–34, Boston, Massachusetts, 1979. ACM.

[SAL96] Michael Stonebraker, PM Aoki, and Witold Litwin. Mariposa: a wide-area distributed database system. *on Very Large Data*, 5(1):48–63, January 1996.

[Sch00] Holger Schmidt. Service Contracts Based on Workflow Modeling. In *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management: Services Management in Intelligent Networks*, pages 132 – 144. Springer-Verlag London, 2000.

[SD89] Donovan a. Schneider and David J. DeWitt. A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment. *ACM SIGMOD Record*, 18(2):110–121, June 1989.

[SJ98] Fro lund Svend and Koistinen Jari. Quality-of-service specification in distributed object systems. In *COOTS*, 1998.

[SLE04] James Skene, D. Davide Lamanna, and Wolfgang Emmerich. Precise service level agreements. In *International Conference on Software Engineering*, pages 179–188, Washington, DC, USA, 2004. IEEE Computer Society.

[SMK97] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Heuristic and randomized optimization for the join ordering problem. *The VLDB Journal*, 6(3):191–208, 1997.

[SMWM06] Utkarsh Srivastava, Kamesh Munagala, Jennifer Widom, and Rajeev Motwani. Query optimization over web services. *VLDB '06, Proceedings of the 32nd International Conference on Very Large Data Bases*, 2006.

[SR07] Manivasakan Sabesan and Tore Risch. Web Service Mediation Through Multi-level Views. In *International Workshop on Web Information Systems Modeling*, pages 755–766, 2007.

[SR09] Manivasakan Sabesan and Tore Risch. Adaptive Parallelization of Queries over Dependent Web Service Calls. In *ICDE*, pages 1725–1732. Ieee, March 2009.

[SR11] Manivasakan Sabesan and Tore Risch. Adaptive parallelization of queries calling dependent data providing web services. In Divyakant Agrawal, K.Selccuk Candan, and Wen-Syan Li, editors, *New Frontiers in Information and Software as Services*, chapter Service Op, pages 132–154. Springer Berlin Heidelberg, 74 edition, 2011.

[SS94] Arun Swami and K.Bernhard Schiefer. On the estimation of join result sizes. In Matthias Jarke, Janis Bubenko, and Keith Jeffery, editors, *Advances in Database Technology - EDBT '94*, chapter Lecture No, pages 287–300. Springer Berlin Heidelberg, 1994.

[Swa88] Anoop Swami, Arun and Gupta. Optimization of Large Join Queries. *SIGMOD Record*, 17(3):8—-17, 1988.

[Swa89] A. Swami. Optimization of large join queries: combining heuristics and combinatorial techniques. *ACM SIGMOD Record*, 18(2):367–376, 1989.

[SY97] E.J. Shekita and H.C. Young. Iterative dynamic programming system for query optimization with bounded complexity, 1997.

[TAB+97] Anthony Tomasic, R'emy Amouroux, Philippe Bonnet, Olga Kapitskaia, Hubert Naacke, and Louiqa Raschid. The distributed information search component (Disco) and the World Wide Web. *ACM SIGMOD Record*, 26(2):546–548, June 1997.

[TD03] Feng Tian and David J DeWitt. Tuple Routing Strategies for Distributed Eddies. In *VLDB 2003, Proceedings of the 29th International Conference on Very Large Data Bases*, pages 333–344, 2003.

[TGT13] Efthymia Tsamoura, Anastasios Gounaris, and Kostas Tsichlas. Multi-objective optimization of data flows in a multi-cloud environment. In *Proceedings of the Second Workshop on Data Analytics in the Cloud - DanaC '13*, pages 6–10, New York, New York, USA, 2013. ACM Press.

[TKK+09] M Tsangaris, G Kakaletris, H Kllapi, G Papanikos, F Pentaris, P Polydoras, E Sitaridi, V Stoumpos, and Y Ioannidis. Dataflow Processing and Optimization on Grid and Cloud. *IEEE Data Engineering Bulletin*, 32(1):67–74, 2009.

[TRV95] A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databases and the design of DISCO. In *Distributed Computing Systems, 1996., Proceedings of the 16th International Conference on*, pages 449–457. IEEE, 1995.

[UKl00] G Ulrich and Werner Kiess ling. Optimizing Multi-Feature Queries for Image Databases. *VLDB '00 Proceedings of the 26th International Conference on Very Large Data Bases*, pages 419—-428, 2000.

[Ull88] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, December 1988.

[Ull97] D. Ullman, Jeffery. Information integration using logical views. *Theoretical Computer Science*, 239(2):189—-210, 1997.

[vdAtH05] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, June 2005.

[Ver04] D.C. Verma. Service level agreements on IP networks. *Proceedings of the IEEE*, 92(9):1382–1388, August 2004.

[VM96] Bennet Vance and David Maier. Rapid bushy join-order optimization with cartesian products. *ACM SIGMOD Record*, 25(2):35—-46, 1996.

[VRM04] Mar'ia-Esther Vidal, Louiqa Raschid, and Julian Mestre. Challenges in Selecting Paths for Navigational Queries : Trade-Off of Benefit of Path versus Cost of Plan. In *Proceedings of the 7th International Workshop on the Web and Databases: Colocated with ACM SIGMOD/PODS 2004*, pages 61—-66, Paris, France, 2004. ACM New York, NY, USA.

[VSIAP10] Genoveva Vargas-Solar, Noha Ibrahim, Michel Adiba, and Jean Marc Petit. Querying Issues in Pervasive Environments. In Apostolos Malatras, editor, *Pervasive Computing and Communications Design and Deployment: Technologies, Trends, and Applications*, pages 1–20. IGI Global, 2010.

[VVsC10] Cuevas-Vicenttin Victor, Genoveva Vargas-solar, and Christine Collet. Coordinating services for accessing and processing data in dynamic environments. In *COOPIS 2010*, 2010.

[WCSO08] Hiroshi Wada, Paskorn Champrasert, Junichi Suzuki, and Katsuya Oba. Multiobjective Optimization of SLA-Aware Service Composition. In *2008 IEEE Congress on Services - Part I*, pages 368–375. IEEE, July 2008.

[Wei09] Gerhard Weikum. Harvesting, searching, and ranking knowledge on the web: invited talk. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 3–4, New York, NY, USA, 2009. ACM.

[WH09] Florian M. Waas and Joseph M. Hellerstein. Parallelizing Extensible Query Optimizers. In *SIGMOD*, pages 871–878. ACM Press, 2009.

[Wie92] Gio Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38—-49, 1992.

[YB08] Qi Yu and Athman Bouguettaya. Framework for Web Service Query Algebra and Optimization. *ACM Transactions on Web*, 2(1):1–35, 2008.

[YKB99] Haiwei Ye, Brigitte Kerherv$backslash$'e, and Gregor V. Bochmann. Qos aware distributed query processing. In *Proceedings of the 10th International Workshop on Database and Expert Systems Applications*, pages 923—-. IEEE Computer Society Washington, DC, USA, 1999.

[YKB03] Haiwei Ye, Brigitte Kerherv'e, and Gregor v. Bochmann. Integrating Quality of Service Requirements in a Distributed Query Processing Environment. In *Database and Expert Systems Applications*, chapter 14th Inter, pages 803–812. Springer Berlin Heidelberg, 2003.

[YKB07] Jia Yu, Michael Kirley, and Rajkumar Buyya. Multiobjective planning for workflow execution on Grids. *2007 8th IEEE/ACM International Conference on Grid Computing*, pages 10–17, September 2007.

[YKvBO03] Haiwei Ye, Brigitte Kerherv'e, Gregor von Bochmann, and Vincent Oria. Pushing Quality of Service Information and Requirements into Global Query Optimization. In *IDEAS*, pages 170–179. IEEE Computer Society, 2003.

[YNGM00] R Yerneni, F Naumann, and H Garcia-Molina. Maximizing coverage of mediated web queries. Rapport technique, Stanford University, 2000.

[ZL98] Q Zhu and P Larson. Solving local cost estimation problem for global query optimization in multidatabase systems. *Distributed and parallel databases*, 421:373–420, 1998.

[ZPL08] Bin Zhou, Jian Pei, and WoShun Luk. A Brief Survey on Anonymization Techniques for Privacy Preserving Publishing of Social Network Data. *SIGKDD Explor. Newsl.*, 10(2):12—-22, 2008.

[ZRV02] Vladimir Zadorozhny, L Raschid, and ME Vidal. Efficient evaluation of queries in a mediator for web sources. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 85—-96, New York, NY, USA, 2002. ACM.

Appendix

APPENDIX A

Query workflow generation definitions

This appendix presents (1) the data dependencies among the activities of a query workflow described by *dt-functions*, and (2) the case of uncomposable activities and healthy relations for enabling the correct composition of activities during the query workflow generation.

A.1 Data dependencies

A data dependency is a binary relation between a couple of activities described by their corresponding *dt-functions* that share sets of tuples. We identify three data dependencies of activities taken from the literature of compilers optimization (*e.g.* [KA01]) and adapted to our context. The definitions go along with examples of the activities described by the *dt-functions* $\bar{F}$ derived from the hybrid query *Which of the common friends of Alice and Bob are interested in Art history ?* of Example 3.1.3.

$$
\begin{aligned}
\bar{F} = \{ & \bar{retr}_a(\{a\}, && \{a.nickname = \text{'Alice'}\}, && atts(a)),\\
& \bar{retr}_b(\{b\}, && \{b.nickname = \text{'Bob'}\}, && atts(b)),\\
& \bar{proj}_a(\{a\}, && \{\}, && \{a.nickname\}),\\
& \bar{proj}_i(\{i\}, && \{\}, && \{i.nickname, i.interest.tag\}),\\
& \bar{filt}_i(\{i\}, && \{i.interest.tag = \text{'Art History'}\}, && atts(i)),\\
& \bar{corr}_{a,b}(\{a,b\}, && \{a.friendship.with.friend = && \\
& && \quad b.friendship.with.friend\}, && atts(a) \cup atts(b)\ ),\\
& \bar{bind}_{a,i}(\{a,i\}, && \{i.nickname = && \\
& && \quad a.friendship.with.friend\}, && atts(a) \cup atts(i)\ )\ \}
\end{aligned}
$$

For readability purposes, we use symbols of *dt-functions* that suggest what the activities do.

A.1.1 Retrieval dependency

Two activities $\mathtt{f}_a$ and $\mathtt{f}_b$ have a retrieval dependency if $\mathtt{f}_a$ retrieves the tuples used by $\mathtt{f}_b$ to perform some transformation. Formally this dependency is defined as follows.

Property A.1.1 [Retrieval dependency] It is said that $\mathtt{f}_b$ has a **retrieval dependency** with $\mathtt{f}_a$ if $\mathtt{f}_b$ performs the retrieval of tuples required by $\mathtt{f}_b$. This is, given the *dt-functions* describing the activities $\bar{f}_a(\mathcal{A}_a, \mathcal{E}_a, \mathcal{P}_a)$ $\bar{f}_b(\mathcal{A}_b, \mathcal{E}_b, \mathcal{P}_b)$, $\exists exp \in \mathcal{E}_a \mid exp$ is a *R-exp* condition or a *B-exp* condition, and one of the following conditions holds

1. $\exists\, exp_b \in \mathcal{E}_b$ such that exp_b is of the form $name.att\,\theta\,term$ and $name \in \mathcal{A}_a \cap \mathcal{A}_b$;
2. $\exists\, name.att \in \mathcal{P}_b$ such that $name \in \mathcal{A}_a \cap \mathcal{A}_b$.

Notation: $f_a \xrightarrow{\text{r}} f_b$

Example A.1.1 Suppose the *dt-function* $\bar{bind}_{a,i}(\{a,i\},\, \{i.nickname = a.friendship.with.friend\},\, atts(a) \cup atts(i))$ describing the activity $\texttt{bind}_{a,i}$ that retrieves the tuple values of the type denoted by i by binding the input attribute $i.nickname$ with the value of the output attribute $a.friendship.with.friend$ of a. Also suppose the *dt-function* $\bar{filt}_i(\{i\},\, \{i.interest.tag =$'Art History'$\},\, atts(i))$ describing the activity $\texttt{filt}_i$ that filters the tuples of i which hold the expressions in $\mathcal{E}_i$. As $\texttt{bind}_{a,i}$ represents the retrieval of the tuples from i and $\texttt{filt}_i$ requires these tuples to perform the filtering, therefore $bind_{a,i} \xrightarrow{\text{r}} filt_i$ is true.

A.1.2 Anti-dependency

The anti-dependency happens when an attribute required by an activity $\texttt{f}_b$ is removed by another activity $\texttt{f}_a$. Formally this dependency is defined as follows.

Property A.1.2 [Anti-dependency] Given two activities $\texttt{f}_a$, $\texttt{f}_b$ described by the *dt-functions* $\bar{f}_b(\mathcal{A}_b, \mathcal{E}_b, \mathcal{P}_b)$ and $\bar{f}_a(\mathcal{A}_a, \mathcal{E}_a, \mathcal{P}_a)$ respectively, it is said that $\texttt{f}_b$ is in **anti-dependency** with $\texttt{f}_a$ if $\texttt{f}_b$ does not contain at least one attribute used by $\texttt{f}_a$ such that one of the following conditions holds

1. $\exists\, exp = (name.att\,\theta\,term) \in \mathcal{E}_a \;\mid\; name \in \mathcal{A}_a \cap \mathcal{A}_b,\; name.att \notin \mathcal{P}_b$. Intuitively, $\texttt{f}_b$ eliminates an attribute required for a condition expression of $\texttt{f}_a$;
2. $\exists\, name.att \in \mathcal{P}_a \;\mid\; name \in \mathcal{A}_a \cap \mathcal{A}_b,\; name.att \notin \mathcal{P}_b$. Intuitively, $\texttt{f}_b$ eliminates an attribute required for the projection expression of $\texttt{f}_a$.

Notation: $\texttt{f}_a \xrightarrow{\text{a}} \texttt{f}_b$

Example A.1.2 Suppose the *dt-function* $\bar{proj}_a(\{a\},\, \{\},\, \{a.nickname\})$ describing the activity $\texttt{proj}_a$ that removes all the attributes of a except $a.nickname$. Also suppose the *dt-function* $\bar{bind}_{a,i}(\{a,i\},\, \{i.nickname = a.friendship.with.friend\}, atts(a) \cup atts(i))$ describing the activity $\texttt{bind}_{a,i}$ that uses the $a.friendship.with.friend$ attribute to perform the retrieval of the a tuples. Therefore, the dependency $bind_{a,i} \xrightarrow{\text{a}} proj_a$ is true because $\texttt{proj}_a$ eliminates the attribute $a.friendship.with.friend$ required by $\texttt{bind}_{a,i}$.

A.1.3 Circular dependency

A circular dependency denotes a functional anomaly where there is a mutual dependency between a couple of activities by either retrieval dependency or anti-dependency. Stated another way, an activity $\texttt{f}_a$ depends on an activity $\texttt{f}_b$ and, in turn, $\texttt{f}_b$ depends on $\texttt{f}_a$. Formally this dependency is defined as follows.

Property A.1.3 [Circular dependency] The activityes $\mathtt{f}_a$ and $\mathtt{f}_b$ are said to be in a **circular dependency** if one of the following conditions holds.

1. $f_a \xrightarrow{\text{r}} f_b \wedge f_b \xrightarrow{\text{r}} f_a$;
2. $f_a \xrightarrow{\text{a}} f_b \wedge f_b \xrightarrow{\text{a}} f_a$.

Notation: $f_a \leftrightarrow f_b$

In the case of a circular dependency a pair of activities that perform retrieval by binding, it is a misunderstanding from the user that is stating a bidirectional binding pattern for a couple of service operations. This is, for two service operations $serv_1 : op_1$ and $serv_2 : op_2$, the $\mathcal{SC}$-$\mathcal{O}$ expression states that the values of the attributes in $oatts(serv_1 : op_1)$ are required by the attributes in $iatts(serv_2 : op_2)$ and the values of the attributes in $oatts(serv_2 : op_2)$ are required by the attributes in $iatts(serv_1 : op_1)$.

Example A.1.3 [Circular dependency by retrieval dependency] Suppose the hybrid query *Which persons have a friendship with themselves ?* For simplicity, we represent this hybrid query in a sql-like expression.

```
SELECT *
  FROM friends.friendsOf As a, friends.friendsOf As  b
 WHERE a.nickname = b.friendship.with.friend
   AND b.nickname = a.friendship.with.friend;
```

This query expresses a self-join that makes sense in a, for example, relational query expression. Nevertheless, in the presence of binding patterns it is not possible. The service operation `a` requires the value of `b.friendship.with.friend` to be invoked, and `b` requires the value of `a.friendship.with.friend`. This anomaly remains in the activities such that $bind_{a,b} \leftrightarrow bind_{b,a}$ because the condition $bind_{a,b} \xrightarrow{\text{r}} bind_{b,a} \wedge bind_{b,a} \xrightarrow{\text{r}} bind_{a,b}$ holds.

In the case of a circular dependency by anti-dependency, it happens if for a given couple of activities with shared type names, both eliminate attributes required by the other one. This is, for two activities $\mathtt{f}_a$, $\mathtt{f}_b$ described by the *dt-functions* $\bar{f}_a(\mathcal{A}_a, \mathcal{E}_a, \mathcal{P}_a)$ and $\bar{f}_b(\mathcal{A}_b, \mathcal{E}_b, \mathcal{P}_b)$ respectively, at least one attribute required by $\mathtt{f}_a$ is not contained in the projection expression $\mathcal{P}_b$, and at least one attribute required by $\mathtt{f}_b$ is not contained in the projection expression $\mathcal{P}_a$.

Example A.1.4 [Circular dependency by anti-dependency] A priori, these anomalies cannot be derived from a $\mathcal{SC}$-$\mathcal{O}$ expression but they must be taken into account during the activities composition. Therefore, we suppose two synthetic *dt-functions* $\bar{proj}_{a1}(\{a\},\{\},\{a.nickname\})$ and $\bar{proj}_{a2}(\{a\},\{\},\{a.friend\})$ both share the type name a and both have at least one attribute in their projection expressions that is not contained in the other one, *i.e.* $\mathcal{P}_{a1} \setminus \mathcal{P}_{a2} \neq \{\}$ and $\mathcal{P}_{a2} \setminus \mathcal{P}_{a1} \neq \{\}$. Thus, $\bar{proj}_{a1} \leftrightarrow \bar{proj}_{a2}$ is true.

A.2 Uncomposable activities and healthy cf-relations

There are activities whose composition gives a circular dependency that leads to an execution deadlock (*e.g.* cyclic stream-graph [HSS+11]). In order to guarantee the termination of the query workflow execution, we distinguish two cases, (1)the execution of an activity depends on the tuples retrieved by another

and this in turn requires the tuples retrieved by the first one, and (2)a couple of activities block each other by anti-dependency. Both cases are characterized by the deadlock cf-relation as follows [1].

Definition A.2.1 [Deadlock cf-relation] Given two activities $\mathtt{f}_a$ and $\mathtt{f}_b$, it is said that there is a deadlock cf-relation denoted by $\mathtt{f}_a \bowtie \mathtt{f}_b$ if there is a circular dependency $\mathtt{f}_a \leftrightarrow \mathtt{f}_b$. Observe that the deadlock cf-relation is commutative and thus $\mathtt{f}_a \bowtie \mathtt{f}_b = \mathtt{f}_b \bowtie \mathtt{f}_a$.

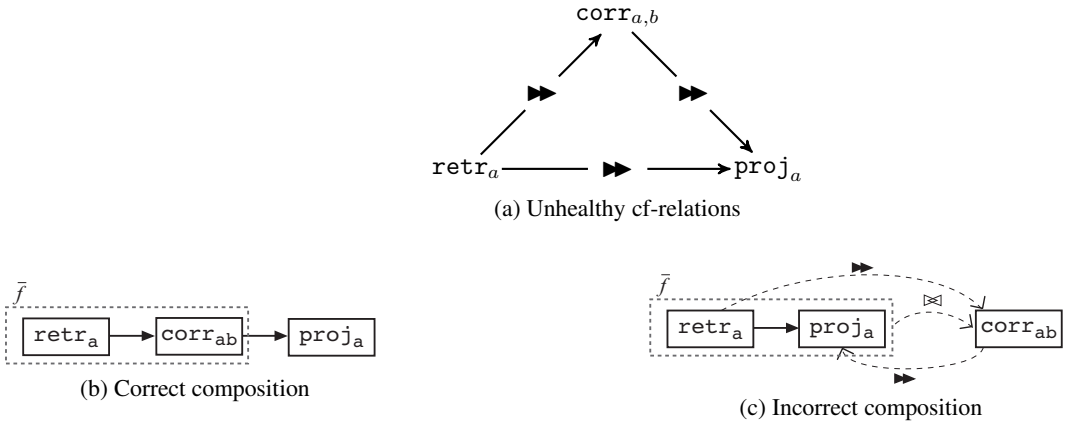
Example A.2.1 Suppose the activities $\mathtt{bind}_{a,b}$ and $\mathtt{bind}_{b,a}$ described by the *dt-functions* $\bar{bind}_{a,b}(\{a, b\}, \{a.nickname = b.friendship.with.friend\}, atts(a), atts(b))$ and $\bar{bind}_{b,a}(\{b, a\}, \{b.nickname = a.friendship.with.friend\}, atts(b), atts(a))$.

The activity $\mathtt{bind}_{a,b}$ retrieves the tuple values of a by using the values of $b.friendship.with.friend$. Meanwhile, the activity $\mathtt{bind}_{b,a}$ retrieves tuple values of b by using the values of $a.friendship.with.friend$. Thus, there is a deadlock cf-relation $\mathtt{bind}_{a,b} \bowtie \mathtt{bind}_{b,a}$.

This example is given by a mistake on the hybrid query expression. The user tries to bound the output values of a service operation with the input attributes of another and vise-versa. Aside this case, a priori all the activities do not have deadlock cf-relations. Nevertheless, there are combination of cf-relations whose activities composition can bring deadlock cf-relations. This combination of cf-relations is what we call unhealthy cf-relations.

A.2.1 Unhealthy cf-relations

The unhealthy cf-relations are groups of three related activities whose composition could brings to a deadlock if the activity composition is performed in certain order.



(a) Unhealthy cf-relations

(b) Correct composition

(c) Incorrect composition

Figure A.1 – Unhealthy cf-relation example

Example A.2.2 Consider the activities $\mathtt{retr}_a$, $\mathtt{proj}_a$, and $\mathtt{corr}_{a,b}$ with dependent cf-relations as shown in Figure A.1a. If such activities are composed firstly $\mathtt{retr}_a$ and $\mathtt{corr}_{a,b}$ and the resulting query workflow is composed with $\mathtt{proj}_a$, thus the result is a correct composition as shown in Figure A.1b. Never-

1. Note that the deadlock cf-relation is data related and it is different to the structural deadlock presented in Section 4.1.1

theless, the composition order can be different, for instance if $\texttt{retr}_a$ and $\texttt{proj}_a$ are composed first, then $\texttt{corr}_{a,b}$ cannot be composed because there is a deadlock cf-relation as shown in Figure A.1c.

This anomaly arises because $\texttt{proj}_a$ is on the right side of the dependent cf-relations with both $\texttt{retr}_a$ and $\texttt{corr}_{a,b}$. Therefore $\texttt{proj}_a$ should be always composed after $\texttt{retr}_a$ and $\texttt{corr}_{a,b}$ are composed. Thus, we define the unhealthy cf-relations as follows.

Definition A.2.2 [Unhealty cf-relation] Suppose a set of cf-relations $\vec{R}$ that contains the cf-relations among a set of activities A. It is said that $\vec{r}_u \in \vec{R}$ is an unhealthy cf-relation if there are two adjacent dependent cf-relations $\vec{r}_1, \vec{r}_2 \in \vec{R}$ such that if $\vec{r}_u = \texttt{a}\,\mathfrak{R}\,\texttt{b} \mid \mathfrak{R} \in \{ \blacktriangleright\!\blacktriangleright , \bowtie , \shortparallel \}$ and $\vec{r}_1, \vec{r}_2 \in \vec{R} \mid \vec{r}_1 = \texttt{a} \blacktriangleright\!\blacktriangleright \texttt{c}$, $\vec{r}_2 = \texttt{c} \blacktriangleright\!\blacktriangleright \texttt{b}$.

The Table A.1 shows the three cases where unhealthy cf-relations arise.

Unhealthy cf-relation	Adjacent cf-relations	Deadlock cf-relation
$\texttt{a} \shortparallel \texttt{b}$	$\texttt{a} \blacktriangleright\!\blacktriangleright \texttt{c}$ & $\texttt{c} \blacktriangleright\!\blacktriangleright \texttt{b}$	$\rightarrow \quad (\texttt{a} \shortparallel \texttt{b}) \bowtie \texttt{c}$
$\texttt{a} \blacktriangleright\!\blacktriangleright \texttt{b}$	$\texttt{a} \blacktriangleright\!\blacktriangleright \texttt{c}$ & $\texttt{c} \blacktriangleright\!\blacktriangleright \texttt{b}$	$\rightarrow \quad (\texttt{a} \blacktriangleright\!\blacktriangleright \texttt{b}) \bowtie \texttt{c}$
$\texttt{a} \bowtie \texttt{b}$	$\texttt{a} \blacktriangleright\!\blacktriangleright \texttt{c}$ & $\texttt{c} \blacktriangleright\!\blacktriangleright \texttt{b}$	$\rightarrow \quad (\texttt{a} \bowtie \texttt{b}) \bowtie \texttt{c}$

Table A.1 – Unhealthy cf-relations

A.2.2 Healthy cf-relations

In order to avoid uncomposable activities during the query workflow generation we define the healthy cf-relations set $\vec{R}_h$ that is a subset of the set $\vec{R}$ with no unhealthy cf-relations.

Definition A.2.3 [healthy set of activity cf-relations] Given a set of cf-relations $\vec{R}$, there is a function $heal : \vec{R} \rightarrow \vec{R}$ that eliminates the unhealthy cf-relations and produces a healthy set of cf-relations $\vec{R}_h$ such that $heal(\vec{R}) = \vec{R}_h = \{\vec{r} \in \vec{R} \mid \vec{r}$ is not an unhealty cf-relation$\}$.

Observe that in whichever case of the unhealthy cf-relations in Table A.1, the dropped cf-relation is reached by transitivity, *i.e.* $\texttt{a} \blacktriangleright\!\blacktriangleright \texttt{c} \blacktriangleright\!\blacktriangleright \texttt{b}$, and thus the three transformations are performed while deadlock cf-relations are avoided.

Example A.2.3 The set of cf-relations $\vec{R} = \{\texttt{retr}_a \blacktriangleright\!\blacktriangleright \texttt{proj}_{a,b}, \texttt{retr}_a \blacktriangleright\!\blacktriangleright \texttt{corr}_{a,b}, \texttt{corr}_{a,b} \blacktriangleright\!\blacktriangleright \texttt{proj}_{a,b}\}$ is transformed into the healthy set $\vec{R}_h = \{\texttt{rentr}_a \blacktriangleright\!\blacktriangleright \texttt{corr}_{a,b}, \texttt{corr}_{a,b} \blacktriangleright\!\blacktriangleright \texttt{proj}_{a,b}\}$ as shown in Figure A.2.

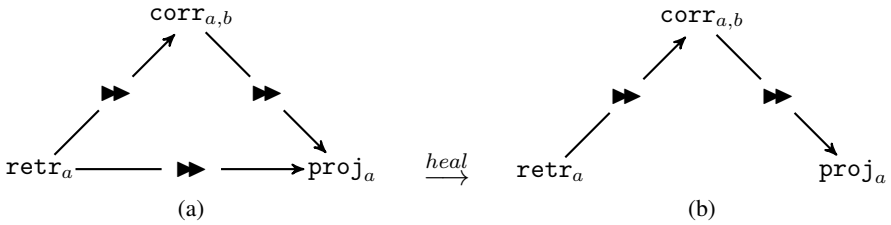


Figure A.2 – Healthy cf-relations example

The dropped cf-relation $\texttt{retr}_a \blacktriangleright\!\blacktriangleright \texttt{proj}_a$ is implicit by transitivity: $\texttt{retr}_a \blacktriangleright\!\blacktriangleright \texttt{corr}_{a,b} \blacktriangleright\!\blacktriangleright \texttt{proj}_a$ and thus the resulting composition implies the three data transformations.

A.2.3 Completeness of ss-$graph$

The $ss\text{-}graph$ $(QW_{\bar{f}}, \vec{R}_h)$ is complete if it has enough Hamiltonian paths that describe all the possible ways to compose activities. This is easy to see if $\vec{R}$ is computed by the Cartesian product
$\vec{R} = \{rel(qw_{f_1}, qw_{f_2}) \mid (qw_{f_1}, qw_{f_2}) \in QW_{\bar{f}} \times QW_{\bar{f}} \wedge qw_{f_1} \neq qw_{f_2}\}$.
Therefore, all the nodes in $QW_{\bar{f}}$ are connected to the $ss\text{-}graph$ and $\vec{R}$ contains all the possible paths that visit all the nodes.

As there are unhealthy cf-relations that lead to the uncomposable of activities, the $ss\text{-}graph$ is pruned by eliminating unhealthy cf-relations and producing then the healthy cf-relations $\vec{R}_h$. Even if this pruning eliminates alternative paths, the nodes still being accessible by transitivity through the reminding edges, therefore the $ss\text{-}graph$ remains complete.

APPENDIX B

HyQoZ specifications

This appendix presents the specifications of HyQoZ data structures, context messages exchanged among the HyQoZ components, REST interfaces of the API directory and QoS directory. Additionally, it presents the Command Line Interface (CLI) for accessing HyQoZTestbed.

B.1 Data structures syntax

Next we describe the syntax of data structures used for representing information through the HyQoZ's flow (See Figure B.1).

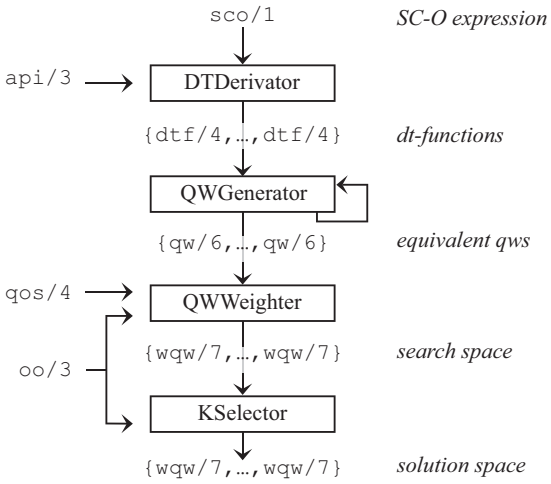


Figure B.1 – HyQoZ information flow

The syntax of data structures adheres to the Prolog syntax style and it is specified in BNF notation.

B.1.1 Complex value types

BNF:

```
 Type := Id : Def;
  Def := integer | string | real | boolean | Tuple | Set ;
Tuple := '[' Type [, Type]+ ']' ;
  Set := '[' Type ']' ;
```

Description:
The complex value types are described by nested functors where atoms denote type names and simple type definitions (*i.e.* `integer`, `string`, `real`, `boolean`). We profit of the list structure of Prolog (*i.e.* `[...]`) to represent tuple types and set types. Both types are distinguished from each other by the number of types contained in a single list. This is, a list representing the definition of a tuple type has at least two types and a list representing a set type has only one type. We adopt the double colon `::/2` for representing the dot operator and the simple colon `:/2` for representing the separator of type names and their definitions.

Example:
The following set type is formed by a single tuple type and this in turn by four simple types

$$\begin{aligned} friends:\{profile :< & \ nickname: String, \\ & gender: String, \\ & email: String, \\ & age: Integer >\} \end{aligned}$$

and it is represented as

```
friends:[profile:[nickname:string,
                  gender:string,
                   email:string,
                     age:integer]]
```

B.1.2 Input data structures

The input data of HyQoZ comes from the optimization requester (*e.g.* Hypatia, end-user) and from the service catalogs API directory and QoS directory. A requester submits hybrid queries represented by `sco/1` along with the optimization objective represented by `oo/3`. HyQoZ fetches the required service APIs represented by `api/3` and the QoS measures represented by `qos/4`.

APIs (`api/3`)

BNF:

```
   API := api '(' Id :: Id , Input , Output ')';
 Input := '[' Type [ , Type]* ']';
Output := Set;
```

Description:
The APIs describing service operations are provided by the API directory and fetched by the DTDerivator. The service operations are represented by `api/3`. We adopt the double colon `::/2` for representing the dot operator and the simple colon `:/2` for representing the separator of type names and their definitions.

The first argument of `api/3` corresponds to the service name and operation name (*e.g.* `service::op`), the second argument corresponds to the set of input attributes and the third one to the set of output attributes. Both input and output attributes are represented by complex value types described in Section 3.1.1.

Example:

For instance, the service $friends$ has two operations $profile$ and $friendsof$ and each one is defined by the types which follow:

Service:	$friends : \langle profile,\ friendship \rangle$
Data operation:	$profile$
Input:	$nickname : String$
Output:	$\{profile : \langle nickname : String, gender : String, email : String, age : Integer \rangle\}$
Data operation:	$friendship$
Input:	$nickname : String$
Output:	$\{friendship : \langle nickname : String, with : \{friend : String\} \rangle\}$

and they are respectively represented as:

```
api( friends::profile,
     [nickname:string],
     [profile:[nickname:string, gender:string, email:string, age:integer]])
api(friends::friendsof,
    [nickname:string],
    [friendship:[nickname:string, with:[friend:string]]])
```

$\mathcal{SC}$-$\mathcal{O}$ expressions (`sco/1`)

BNF:

```
    SCO := sco `(' OP `)';
     OP :=  RHO | PI | SIGMA | CORR | BIND;
    RHO := rho `(' Id :: Id as Id , Id :: Id `)';
     PI := pi `(' `[' Term+ `]' `,' OP `)';
  SIGMA := sigma `(' Cond `,' OP `)';
   CORR := corr`(' `[' CondExp `]' , OP , OP `)';
   BIND := bind`(' `[' CondExp `]' , OP , OP `)';
CondExp :=  Cond [, Cond]*;
   Cond := Term THETA Term;
  THETA := = | >= | <= | <> ;
   Term := Id [::Id]* | Constant;
```

Description:

The $\mathcal{SC}$-$\mathcal{O}$ expressions for denoting hybrid queries (*cf.* Section 3.1) are taken as input of the DTDerivator for producing the *dt-functions* `dtf/4`. A $\mathcal{SC}$-$\mathcal{O}$ is represented by `sco/1` whose single argument is a nested expression of data operators (*i.e.* renaming, projection, selection, theta-join, bind-join).

Example:

For example, consider the hybrid query *Which are the common friends of Alice and Bob ?* in Figure B.2 and its `sco/1` representation as follows. The unary complex value operators (*i.e.* `pi/2`, `sigma/2`, `rho/2`) have

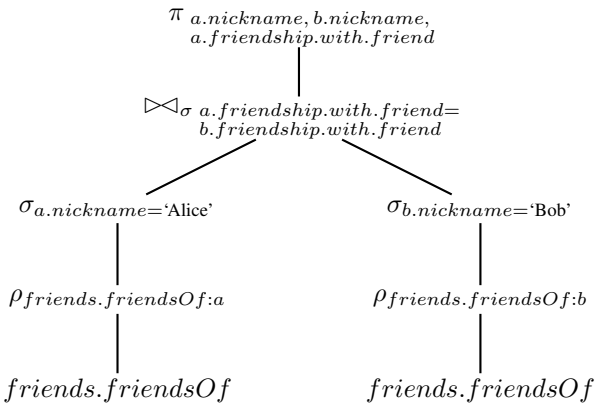


Figure B.2 – *Which are the common friends of Alice and Bob ?*

two arguments; the first corresponds to the expression that configures the operators, and the second one the operand. The binary complex value operator (*i.e.* `corr/3`) have three arguments; the first corresponds to the expression that configures the operator, and the last two to each operand.

```
sco(pi([a::friendship::with::friend],
       sigma(a::nickname='Alice',
             sigma(b::nickname='Bob',
                   corr([a::friendship::with::friend
                         =b::friendship::with::friend],
                        rho(friends::friendsof as a, friends::friendsof),
                        rho(friends::friendsof as b, friends::friendsof)
                   )
             )
       )
))
```

QoS Measures (`qos/4`)

BNF:

```
       QOS := qos ( Type , Measure , Units, Value);
      Type := time | price | energy;
   Measure := latency | throughput | price | response_time | energy;
     Units := seconds | euros | dollars | bits_per_second | kilojoules;
     Value := Number
```

Description:

The QoS directory provides up-to-date QoS measures fetched by the QWWeighter. The measures are associated to service operation instances and the network. Each measure is represented by the `qos/4`. The first argument indicates if it is about the service operation (*i.e.* `sop`) or the network (*i.e.* `net`). The second argument corresponds to the measure identifier. The third argument is the units in which the measure is expressed. Finally, the fourth argument is the value of the measure. For instance, the price of invoking a service operation is $0.2€$ is represented by `qos(sop, price, euro, 0.2)`.

Optimization objective (`oo/3`**)**

BNF:

```
      OO := oo ( Time , Price , Energy);
    Time := time ( V , W );
   Price := price ( V , W );
  Energy := energy ( V , W );
       V := Number;
       W := 1 | 2 | 3;
```

Description:

The Service Level Agreement is taken as input by the QWWeighter and by the KSelector. It is represented by `oo/3` that specifies the expectations of the user about the execution of the query workflow implementing the hybrid query. Each argument of `oo/3` represents the weighted cost attribute value for the time, price and energy. Nevertheless, this structure is modifiable according to the cost attributes interesting for the user which are subject to be implemented by a QWWeighter instance. Each argument specifies the name of the cost attribute, the expected value, and the weight between 1 and 3 (*cf.* Section 3.3).

Example:

For instance, the following Service Level Agreement $oo\langle(38.4, 3), (40.0, 1), (43.0, 2)\rangle$ is represented by `oo(time(38.4, 3), price(40.0, 1), energy(43.0, 2))`.

B.1.3 Intermediate data

The $\mathcal{SC}$-$\mathcal{O}$ expressions are derived into a set of *dt-functions* that we represent by `dtf/4` and are used for generating the query workflows represented by `qw/6`. Such query workflows are subject to a cost function that weights them with a vector `cost/3`.

dt-functions **(**`dtf/4`**)**

BNF:

```
      DTF := dtf `(' `[' TNList `]' , `[' CondList `]' , `[' ProjList`]' , Id `)';
   TNList := Id [, Id]*;
 CondList := Cond [, CondList]*;
     Cond := Term THETA Term;
    THETA := = | >= | <= | <> ;
     Term := Att | Constant;
      Att := Id [:: Id]*;
 ProjList :=  Att [, Att]*;
```

Description:

The *dt-functions* are produced by the DTDerivator and consumed by the QWGenerator. Their representation is done by `dtf/4` whose three first arguments correspond to each of the parameters of the *dt-function* signature. Besides, the fourth argument is an arbitrary and unique id represented by an `atom`. The uniqueness of the id's is transparently managed by the DTDerivator.

Example:

For instance, consider the following *dt-function* derived from a $\mathcal{SC}$-$\mathcal{O}$ expression that retrieves the interests of Alice from the data service operation denoted by a
$\bar{f}(\{a\}, \{a.nickname = \text{'Alice'}\}, \{a.nickname, a.interests\})$
and its corresponding representation

```
dtf([a], [a::nickname='Alice'], [a::nickname, a::interests],retr_a)
```

Query workflows (`qw/6`, `wqw/7`**)**

BNF:

```
              QW := qw'(' '[' VerticesList ']' ','
                    '[' VerticesList ']' ','
                    '[' VerticesList ']' ','
                    '[' ArcList ']' ',' in ',' out ')' ;
   VerticesList := Vertex [, Vertex]* ;
         Vertex := Id | ParID | in | out ;
        ArcList := Arc [, Arc]* ;
            Arc := '(' Vertex , Vertex ')' ;
```

Description:

The query workflows `qw/6` are produced by the QWGenerator and consumed by the QWWeighter. A query workflow is represented by a DAG through `qw/6` whose arguments allow the manipulation of the query workflow during the generation and correspond to the model presented in Section 4.1.

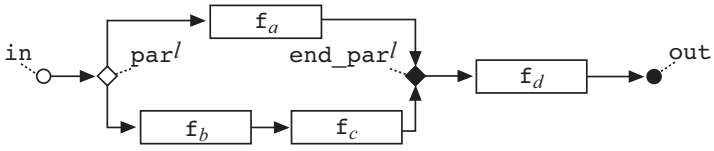


Figure B.3 – Query workflow example

Example: For instance, consider the query workflow in Figure B.3 whose representation is as follows.

```
qw([a,b,c], [par1, end_par1], [a,b,c, par1, end_par1, in, out],
   [(in,par1), (par1, a), (par1,b), (a, end_par1),
    (b, end_par1), (end_par1, c), (c,out)],
   in, out)
```

The QWWeighter produces the set of weighted query workflows `wqw/7` whose last argument corresponds to the cost `cost/3` of the query workflow.

Cost (`cost/3`)

BNF:

```
      COST := cost ( CTime , CPrice , CEnergy);
     CTime := time ( C );
    CPrice := price ( C );
   CEnergy := energy ( C );
         C := Number;
```

Description:
A query workflow cost is computed by the QWWeighter. It is a three attribute vector represented by `cost/3` where each argument denotes a cost attribute value. This is `cost(time(T),price(P),energy(E))` where `T` denotes the execution time cost, `P` denotes the execution price cost, and `E` denotes the energy consumption cost. This structure is compatible with the cost attributes specified by `oo/3`. The query workflows with its corresponding cost `wqw/7` form the search space used by the KSelector along with the `oo/3` for computing the solution space.

B.2 Context messages

The HyQoZ context messages are self-descriptive messages that enable the stateless client-server communication. These are formed by a series of attributes whose values belongs to specific domains. In particular, we consider the domain String, Long, Boolean, and the following proprietary domains:

— Functor: is an special case of a string with a specific format detailed by a functor syntax (*cf.* Section B.1)

— Nested messages: the messages may contain complex values in form of nested messages

There are two kinds of context messages: (1) **request messages** and (2) **response messages**. These messages are used into all the request/responses among HyQoZ components including the ones towards the external applications. The principle of this messages is to keep the context of responses through the hops of the optimization process or either for individual requests to HyQoZ components. The use of context messages enables either a service orchestration by an orchestration engine or a service coordination.

Each message is presented with a brief explanation and the message format which contains a set of fields where each is represented by an attribute name, its domain, a default value –if there is– , if the value is mandatory or not (*i.e.* true/false), and the description of the field.

B.2.1 Request context

The Table B.1 shows the fields of the request context message.

Attribute	Domain	Default	Manadatory	Description
rId	String		true	Identification of the requester
rToken	String		true	Token of the requester
role	'application' / 'end-user' / 'derivator' / 'generator' / 'weighter' / 'selector' / 'orchestrator'		true	Role of the requester into the optimization process
requestContext	Request context		false	Context of the preceding request
requestTS	Long		false	Timestamp assigned by the requester
requestID	String		false	Request identification managed by the requester
iRepresentation	String	'Functor'	false	Input representation
oRepresentation	String	'Functor'	false	Output representation
commPattern	'sync','async'		true	Communication pattern requested
callbackURL	String		false	URL of the sink for the response delivering via a callback
callbackVerb	'POST' / 'PUT'	'POST'	false	HTTP method to perform the callback
threshold	Long	'0L'	false	Time tolerated for performing the optimization. The value is added to the instant in which the request arrives to the service instance. If '0L', it is assumed no threshold

Table B.1 – Request context message format

The attributes **rId** and **rToken** enable the orthogonal identification and authentication of the requester in the server. These attributes are instantiated aside of the data elements for optimization purposes. The attribute **role** identifies the role of the requester and this one, along with the **requestContext**, enables the provenance of the optimization process. The attributes **requestTS** and **requestID** allow the requester to manage the invocations it performs and eventually take decisions about, for example, the validity of a response. The attributes **iRepresentation** and **oRepresentation** defines the underlining formats in which the data optimization elements are passed among services and the marshaling and un-marshaling of the requests/responses. The attributes **callbackURL** and **callbackVerb** allow to define which entity shall get the response in the case of asynchronous request in **commPattern**. This enable either a sequential multi-hop processing in which the response is returned directly to the requester, or a centralized coordination where all the responses are returned to the coordinator. The **threshold** attribute allows to define a permitted limit to perform the optimization phase. This attribute means that if the threshold is elapsed may the service instance has not satisfied the required quality of service.

B.2.2 Response context

The Table B.1 shows the fields of the request context message.

The attributes **sId** and **sToken** enable the identification and authentication of the service instance on the requester side. These attributes are instantiated aside of the data elements for optimization purposes.

Attribute	Domain	Default	Manadatory	Description
sId	String		true	Identification of the server delivering the response
sToken	String		true	Token of the server delivering the response
role	'application' / 'end-user' / 'derivator' / 'generator' / 'weighter' / 'selector' / 'orchestrator'		true	Role of the requester into the optimization process
responseContext	Response context		false	Response context of the preceding response
requestTS	Long		false	Timestamp assigned by the requester
requestID	String		false	Request identification managed by the requester
iRepresentation	String	'Functor'	false	Representation of the original request
oRepresentation	String	'Functor'	false	Representation of the response
timeEpalsed	Boolean		true	Indicates if the request threshold has been elapsed

Table B.2 – Response context message format

The attribute **role** identifies the role of the service instance and this one, along with the **responseContext**, enables the provenance of the optimization process. The attributes **requestTS** and **requestID** are the same than in the request message and allow the requester to manage the invocations/responses. The attributes **iRepresentation** and **oRepresentation** defines the underlining formats in which the data elements are passed among services and the marshaling and un-marshaling of the requests/responses. The **timeElapsed** attribute indicates if the request threshold has been elapsed at the moment of the current response is issued.

B.3 APIDirectory and QoSDirectory REST interfaces

B.3.1 APIDirectory

`api/catalog`

DESCRIPTION	Retrieves the catalog of service operations APIs that satisfy the query parameters below
URL STRUCTURE	**`http://<baseuri>/api/catalog`**
METHOD	**`GET`**
QUERY PARAMETERS	**`sNameFilter`** Filter for the service name represented as a regular expression
	`oNameFilter` Filter for the service operation name represented as a regular expression
INPUT	**`requestContext`** Context of the request
OUTPUT	**`requestContext`** Context of the request
	`responseContext` Context of the response
	`catalog` List of service operations in, for example, functor format

OUTPUT EXAMPLE

```
“catalogResponse” :{
    “requestContext” :{...},
    “responseContext” :{...},
    “catalog” :[“api” :“api(friends::profile,
                        [nickname:string],
                        [profile:[nickname:string, gender:string
                                  email:string, age:integer]]”,
              “api” :“api(friends::friendsof,
                        [nickname:string],
                        [friendship:[nickname:string,
                                     with:[friend:string]]]”,
              ... ]
    }
```

`api/{serviceName}/{opName}/choose`

DESCRIPTION Retrieves an instance of a service operation [1]

URL STRUCTURE `http://<baseuri>/api/{serviceName}/{opName}/choose`

METHOD `GET`

PATH PARAMETERS `serviceName` Name of the service

`opName` Name of the parameter

INPUT `requestContext` Context of the request

OUTPUT `requestContext` Context of the request

`responseContext` Context of the response

`instance` Service operation instance with the url of the machine-readable description of the service operation (*e.g.* WADL [Had09])

OUTPUT EXAMPLE

```
“instanceResponse” :{
   “requestContext” :{...},
   “responseContext” :{...},
   “instance” :{“api” :“api(friends::profile,
                        [nickname:string],
                        [profile:[nickname:string, gender:string
                                  email:string, age:integer ]]”,
              “wadl_url” :“http ://dataservices/instances/profile.wadl”},
   }
```

B.3.2 QoSDirectory

`/qos`

DESCRIPTION Retrieves the QoS measures of a service operation instance

URL STRUCTURE **`http://<baseuri>//qos`**

METHOD **`GET`**

QUERY PARAMETERS **`wadl_url`** URL of the description of the service operation instance

INPUT **`requestContext`** Context of the request

OUTPUT **`requestContext`** Context of the request

`responseContext` Context of the response

`wadl_url` URL of the wadl of the requested service operation instance

`timestamp` Instant at which the QoS measures are valid

`measures` State of service operation instance expressed as network and execution QoS measures

OUTPUT EXAMPLE

```
“measuresResponse” :{
    “requestContext” :{...},
    “responseContext” :{...},
    “wadl_url” :“http ://dataservices/instances/profile.wadl”
    “timestamp :” :“1379602397”
    “net” :{measure :{name :latency, units :seconds, value :},
             measure :{name :throughput, units :bps, value :},
             measure :{name :price, units :euros, value :},
             measure :{name :energy, units :watts, value :}
             },
    “ex” :{measure :{name :isize, units :kb, value :},
            measure :{name :osize, units :kb, value :},
            measure :{name :response_time, units :seconds, value :},
            measure :{name :price, units :euros, value :},
            measure :{name :energy, units :euros, value :}
            }
    }
```

B.4 Command Line Interface for HyQoZTestbed

`hyqoztb`	
`-buildtime`	Defines the cost estimation by build-time formulation
`-controlflow`	Defines the generetion by control-flow
`-dataflow`	Defines the generetion by data-flow
`-derive`	Derivation of the dt-functions
`-dtfs <arg>`	dt-functions in the list format `[dtf(A,E,P,ID),...]`
`-estimation_approach <arg>`	Defines the cost estimation approach at run-time (partialcf,fullcf,df)
`-forcegenshqs`	Force synthetic HQ generation
`-generate`	Generation of alternative query workflows
`-genshqs`	Generation of synthetic hqs
`-hqsignature <arg>`	Hybrid query signature
`-inputfile <arg>`	Input file
`-k <arg>`	'k' for the top-k selection
`-n <arg>`	Lower bound
`-N <arg>`	Upper bound
`-outputdir <arg>`	Output directory
`-outputfile <arg>`	Output file
`-qw <arg>`	Query workflow in functor sytax `qw/6`
`-runtime`	Defines the cost estimation by run-time formulation
`-sco <arg>`	SCO expression
`-select`	Selection of the solution space of query workflows
`-oo <arg>`	Optimization objective in functor syntax `oo/3`
`-typenames <arg>`	Type names in the list format `[type_name(Alias, S::M),...]`
`-weight`	Weighting of quwey workflows

HyQoZ – Optimisation de requêtes hybrides basée sur des contrats SLA

Carlos-Manuel LÓPEZ-ENRÍQUEZ