Evaluating Hybrid Queries through Service Coordination in HYPATIA

Víctor Cuevas-Vicenttín*, Genoveva Vargas-Solar*†, and Christine Collet*

*Grenoble Institute of Technology, CNRS, UJF
Grenoble Informatics laboratory (LIG),
681, rue de la Passerelle, BP. 72, 38402
Saint Martin d'Hères Cedex, France

†French Mexican Laboratory of Informatics and Automatic Control
Exhacienda Sta. Catarina Mártir s/n 72820
San Andrés Cholula, Puebla, México

{Firstname.Lastname}@imag.fr

ABSTRACT

The emergence of mobile and ambient computing technologies brings democratization in the access to information and data; services play a crucial role, thereby opening new research challenges for data querying. A promising method to access data within these novel dynamic environments in a convenient and efficient way is declarative querying. Such queries, which we characterize as hybrid queries, involve streaming and on-demand data originated from services, possibly with temporal and mobile properties. To evaluate these queries we propose an approach implemented in the HYPATIA system, which addresses two main aspects (i) using service coordination for query evaluation and (ii) an efficient and flexible mechanism that facilitates incorporating new capabilities.

1. INTRODUCTION

The advent of mobile computing and communication technologies lays a foundation for enabling timely and inexpensive access to information. We view declarative queries as the most convenient way of expressing information needs in an ambient computing environment. Such queries involve streaming and on demand data originated from services, which may have temporal and mobile properties; due to these characteristics we refer to them as *hybrid queries*.

Traditional techniques in conjunction with techniques used for evaluating queries over data streams could in principle be a solution for processing hybrid queries. Adopting such an approach, however, presents two major drawbacks; both of which are associated with traditional DBMSs, but that are particularly disadvantageous in highly dynamic environments. First, a query evaluation system based on such an approach will be expensive to develop and maintain. The addition of new capabilities in particular, would require that highly qualified system programmers develop the extensions in the required programming language, after which it is necessary to recompile and redeploy the system. Second, the system would require a heavy dedicated infrastructure, making it expensive to use and yielding it confined to a particular platform and location.

To our knowledge none of existing querying techniques tackle at the same time traditional, mobile and continuous queries by coordinating data services (on-demand and stream as well as static and nomad) along with computation services, which provide the computing (processing, calculation, indexing, etc.) functionalities that query evaluation requires. Related works either focus on services for data access like in [3], or address dynamic documents like in [1].

The main contributions of the HYPATIA system we propose for evaluating hybrid queries are: (i) the use of a service coordination approach for evaluating queries by mapping them to workflows. Activities of such workflows correspond to service calls, where services either enable access to data (e.g. Web-hosted DBMSs, search engines), or perform computing and data processing functions. (ii) the mechanism to construct and enact query workflows. Our approach is based on a workflow model and its corresponding evaluation infrastructure, providing flexible service coordination mechanisms that facilitate the addition of new capabilities (e.g., query operators). Our demonstration will show, by means of a concrete scenario, the feasibility of this approach for the evaluation of several types of queries without relying on a prebuilt and to a large degree inflexible DBMS. A more detailed description of our approach is given in [2].

2. DEMONSTRATION SCENARIO

The goal of HYPATIA is to provide an approach that integrates continuous, stream, snapshot, spatio-temporal, and mobile queries for accessing data in ambient computing environments. For this purpose, it introduces the notion of hybrid queries and a novel approach for flexible query processing based on service coordination.

Consider the scenario depicted in Figure 1. Multiple users are in an urban area carrying GPS-enabled mobile devices that periodically transmit their location; furthermore, they have agreed to share some of their personal information. A

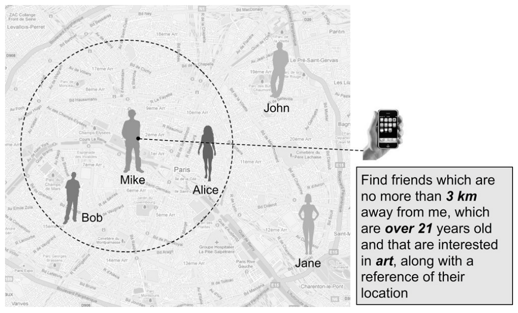

John
Mike
Alice
Bob
Jane
Find friends which are no more than **3 km** away from me, which are ***over 21*** years old and that are interested in ***art***, along with a reference of their location

Figure 1: Example scenario

user in this scenario may want to *Find friends which are no more than 3 km away from him/her, which are over 21 years old and that are interested in art, along with a reference point of their location.*

To answer this query three *data services* need to be accessed, which produce data in one of two ways: *on-demand* in response to a given request, or continuously as a *data stream.* In either case, the data service exposes an interface, composed of several operations and supported by standardized protocols. The JavaScript Object Notation (JSON) is used to represent the data. Accordingly, objects are built from atomic values, nested tuples, and lists.

The users' location is made available by a stream data service with the (simplified) interface

```
subscribe() → ⌈location:⟨nickname, coor⟩⌉
```

consisting of a subscription operation that after invocation, will produce a stream of `location` tuples with a nickname that identifies the user and his/her coordinates. A stream is a continuous (and possibly infinite) sequence of tuples ordered in time.

The rest of the data is produced by the next two on-demand data services, each represented by a single operation

```
profile(nickname) → person:⟨age, gender, email⟩
interests(nickname) → [s_tag:⟨tag, score⟩]
```

The first provides a single `person` tuple denoting a profile of the user, once given a request represented by his/her nickname. The second produces, given the nickname as well, a list of `s_tag` tuples, each with a tag or keyword denoting a particular interest of the user (e.g. music, sports, etc.) and a numeric score indicating the corresponding degree of interest.

The hybrid query just discussed can be expressed in our SQL-like query language, which we call HSQL, as follows

```
SELECT p.nickname, p.age, p.gender, p.email,
   nn('poi', l.coor)
FROM profile p, location l [range 10 min], interests i
WHERE p.age >= 21 AND l.nickname = p.nickname AND
i.nickname = p.nickname AND i.tag = 'art'
AND distance(l.coor, mycoor) <= 3
```

The `location` stream is bounded by a time-based window which will consider only the data received within the last 10 minutes. We rely on a special function `distance` to evaluate the spatial predicate of the query, which receives the current location of the user issuing the query as `mycoor`. As a location reference, the nearest point of interest (poi) is found by the `nn` service (nearest neighbor) which will find objects in the `poi` category.

2.1 Processing hybrid queries

In order to evaluate a hybrid query like the one presented in our example, we first need to give to it an executable form, which in our case is a workflow of activities. A *query workflow* specifies a service coordination in which activities will invoke services as required to generate the query result.

Two types of services take part in a query service coordination: data services and computation services. While data services provide the data, computation services are able to process these data as required to implement query operators. As we will see, these services can themselves be represented as workflows and therefore as service coordinations.

A query workflow corresponding to our example query is given in Figure 2. It is composed of data service activities such as the `location`, `profile` and `interests` activities, as well as computation service activities such as the `distance`, `nearest-neighbor`, `join`, `selection`, and `projection` activities. Services interoperate via workflow synchronization constructs such as parallel and sequential composition, which are defined using the Abstract State Machines [4] (ASM) formalism.

We describe next how a query workflow is generated from a given declarative query, how it is enacted, and the advantages that result from our approach.

Query workflow generation

The primary task to carry out for building a query workflow like the one depicted in Figure 2 from a given hybrid query is to determine appropriate joins. We use for this purpose an adaptation of the Graham-Yu-Ozsoyoglu (GYO) algorithm used in database theory to determine if a query is acyclic.

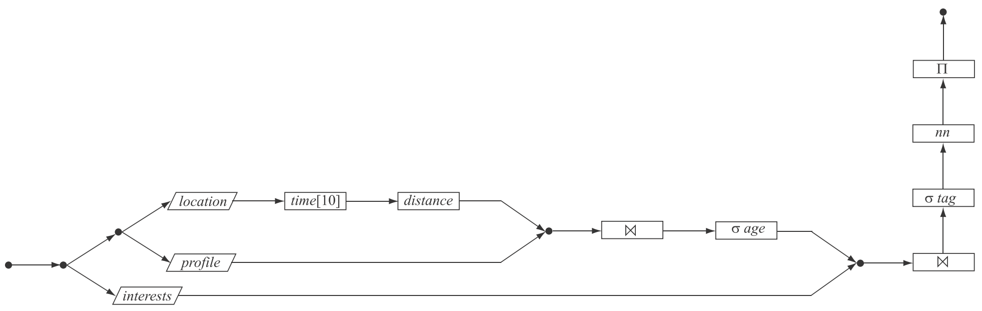
Our adaptation considers data service interfaces and the input and output parameter dependencies in their operations. For brevity, we just mention the three main phases of the algorithm. (i) Represent the join dependencies by a hypergraph. (ii) Process the hypergraph to yield a parse tree denoting the valid join orders. (iii) Derive a join order by traversing the tree and then add the remaining operators to produce the query workflow.

Query evaluation

Evaluating a hybrid query thus depends (i) on finding the adequate (data and computation) services, and binding each of the workflow activities to its corresponding service; (ii) on enabling the communication and interoperation between the activities. We briefly deal with these aspects next.

Service provisioning and invocation

As previously discussed, we have on-demand and streaming data services. Data is gathered from on-demand data services by invoking their data operations with the appropriate parameters, producing as a result a set of output tuples. This process is depicted in the figure below for the `profile` service of our example. Streaming data services are treated in an analogous manner, except that a single invocation is performed on their `subscribe` operation, after which they produce a data stream that is sent to the destination specified with the subscription parameters.



$$\pi_{p.nickname,\ age,\ gender,\ email,\ nn('poi',\ l.coor)} (\sigma_{'art' \in interests} (interests \bowtie_{i.nickname=p.nickname} (\sigma_{age < 21}(profile \bowtie_{p.nickname=l.nickname} (\sigma_{dist < 3} (\mathcal{W}_{t=10}(location)))))))$$

Figure 2: Hybrid query workflow example

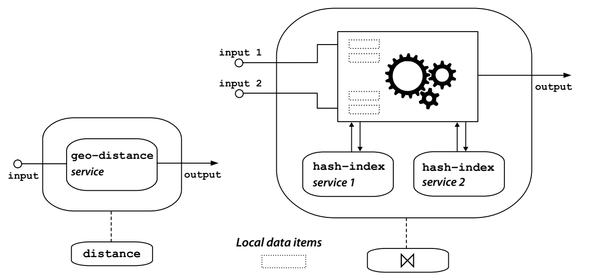
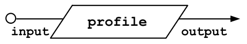


Figure 3: Simple (left) and composite (right) computation service



On the other hand, the computation services that support query operators can be simple or composite. Simple computation services are those whose execution is performed by a single service operation invocation. Figure 3 at the left illustrates such a service. The `distance` computation service relies on a `geo-distance` service, which provides the capability to calculate the geographical distance between two points. This is achieved by the invocation of its `distance` operation with the appropriate parameters.

In contrast, the execution of a composite computation service involves multiple operation invocations, possibly also from different services, as well as manipulation of local data. These tasks are organized in a service coordination specified as a workflow, following a model in which we add conditional and iteration constructs formalized via ASMs to our basic workflow model. The local data items, which are only visible within the composite computation service during its execution, maintain relevant information to guide the evaluation of the coordination appropriately.

Figure 3 at the right gives an example of a composite computation service, in the form of an overview of a service evaluating the join operator based on the symmetric hash-join algorithm and two instances of a stateful `hash-index` service. Several interrelated operation invocations on both service instances, as well as reads and updates on local data items, are used to find the tuple matches that form part of the join result.

Service interoperation and communication

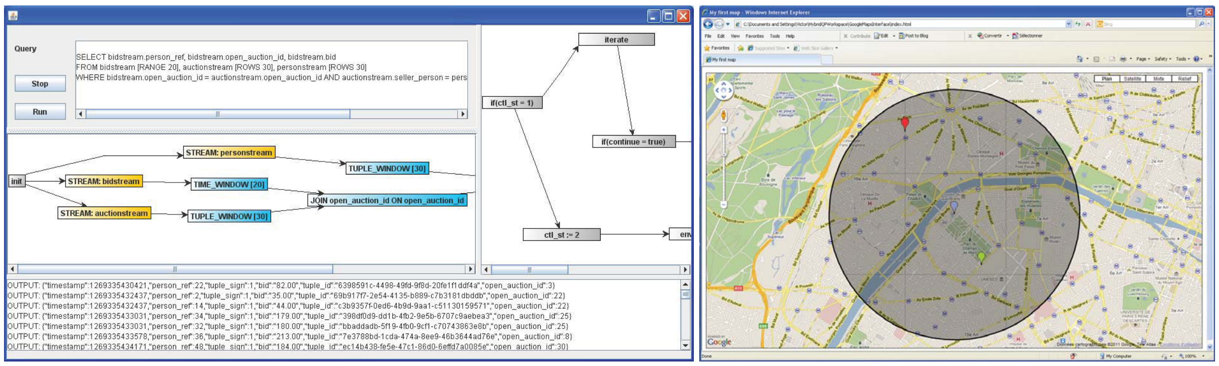
The computation services in a query coordination communicate via asynchronous *input operations* exposed by its executing environment. In order to take advantage of this mechanism, composite computation services must be designed in such a way that they access the input data sent to them by their preceding computation services. This data communication occurs when a service invokes the corresponding input operations of its successor, which in turn will store the input tuples as part of their state in order to process it accordingly.

Our query processing approach based on service coordination is geared towards flexibility, which is highly desirable in ambient computing environments. First, service coordination offers the capability to dynamically acquire resources that provide the required functionality by adopting a late binding approach, where additionally the best services available can be selected at query evaluation time. Furthermore, if some data or computation services become unavailable, evaluating the query is still possible by discovering other services that provide the same or similar data and functionality in the environment.

3. EXPERIMENT SETUP

In order to implement the Friend Finder scenario described in our example, and subject of our demonstration, we developed a test dataset using GPS tracks obtained from everytrail.com. Concretely, 58 tracks generated (by walking, cycling, or driving) within the city of Paris. We converted the data from GPX to JSON and integrated it to our custom stream server to create the location service. For the profile and interests services we created a MySQL database accessible via JAX-WS Web Services running on Tomcat. The profile data is artificial and the interests were assigned and scored randomly using the most popular tags used in Flickr and Amazon. For the nearest-neighbor (NN) points of interest we converted a KML file containing the major tourist destinations in Paris into JSON, this data is employed by the corresponding NN service in conjunction with an R-tree based spatial indexing service.

We also implemented an interface based on Google Maps that enables to visualize the query result, which is presented in Figure 4 right. This setup enables posing queries involving spatio-temporal aspects like in our example. Finally,



```
SELECT bidstream.person_ref, bidstream.open_auction_id, bidstream.bid
FROM bidstream [RANGE 20], auctionstream [ROWS 30], personstream [ROWS 30]
WHERE bidstream.open_auction_id = auctionstream.open_auction_id AND auctionstream.seller_person = pers
```

```
OUTPUT: {"timestamp":1269335430421,"person_ref":22,"tuple_sign":1,"bid":"82.00","tuple_id":"6398591c-4498-49fd-9f8d-20fe1f1ddf4a","open_auction_id":3}
OUTPUT: {"timestamp":1269335432437,"person_ref":2,"tuple_sign":1,"bid":"35.00","tuple_id":"69b917f7-2e54-4135-b889-c7b3181dbddb","open_auction_id":22}
OUTPUT: {"timestamp":1269335432437,"person_ref":14,"tuple_sign":1,"bid":"44.00","tuple_id":"c3b9357f-0ed6-4b9d-9aa1-c51130159571","open_auction_id":22}
OUTPUT: {"timestamp":1269335433031,"person_ref":34,"tuple_sign":1,"bid":"179.00","tuple_id":"398df0d9-dd1b-4fb2-9e5b-6707c9aebea3","open_auction_id":25}
OUTPUT: {"timestamp":1269335433031,"person_ref":32,"tuple_sign":1,"bid":"180.00","tuple_id":"bbaddadb-5f19-4fb0-9cf1-c70743863e8b","open_auction_id":25}
OUTPUT: {"timestamp":1269335433578,"person_ref":36,"tuple_sign":1,"bid":"213.00","tuple_id":"7e3788bd-1cda-474a-8ee9-46b3644ad76e","open_auction_id":8}
OUTPUT: {"timestamp":1269335434171,"person_ref":48,"tuple_sign":1,"bid":"184.00","tuple_id":"ec14b438-fe5e-47c1-86d0-6effd7a0085e","open_auction_id":30}
```

Figure 4: Hypatia GUI (left) and Friend Finder visualization GUI (right)

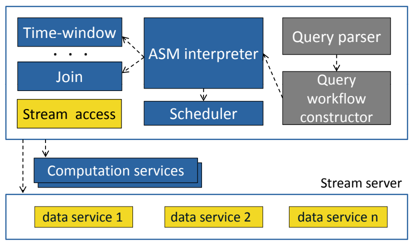


Figure 5: Architecture of Hypatia

an additional online auctions setup based on the NEXMark benchmark helps to demonstrate queries involving the rest of the implemented operators.

3.1 Implementation issues

The architecture of our system is presented in Figure 5, it was developed on the Java platform. Queries in HYPATIA are entered via a GUI (presented in Figure 4) and specified in our HSQL query language. When a query is provided to the system it is parsed and then its corresponding query workflow is generated by the query workflow constructor component, which employs our GYO-based algorithm. The parser was developed using the ANTLR (http://www.antlr.org/) parser generator.

The GUI also enables the user to visualize the query workflow (at the middle left of Figure 4 left), which is facilitated by the use of the JGraph (http://www.jgraph.com/) library. Data services are represented in yellow whereas computation services are represented in blue, both with their corresponding labels.

Two main components support the computation services corresponding to query operators. A scheduler determines which service is executed at a given time according to a predefined policy, while composite services coordinations are executed by the ASM interpreter that implements our workflow model (also developed with ANTLR). The computation service workflows can be visualized through the GUI, as shown in the right panel of Figure 4 (left). In turn, they are specified textually in a language based on the ASM formalism.

We developed a set of computation services that are used to build hybrid query operators. These services run on a Tomcat container supported by the JAX-WS reference implementation (https://jax-ws.dev.java.net/), which enables to create stateful services. The core operators currently implemented (either as simple or composite computation services) are join and bind-join, tuple and time based windows, grouping and aggregation, selection and projection.

During the evaluation of a query data tuples flow from the data services to various computation services, as determined by the query workflow. The end result is a data stream that denotes the tuples that are added and the tuples that are removed from the query answer, which is indicated by a special (positive or negative, respectively) numeric attribute on the tuples. The query result stream is presented in a textual form in the GUI (bottom of Figure 4 left). For the Friend Finder scenario, the results are also exported as a service and presented in our Google Maps GUI as markers within the (shaded) valid result area and that link to the user information.

Acknowledgment

This work has been supported by the ANR-08-SEGI-014 OPTIMACS Project (http://optimacs.imag.fr), financed by the French National Research Agency (ANR).

4. REFERENCES

[1] S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 527–538, New York, NY, USA, 2003. ACM.

[2] V. Cuevas-Vicenttín, G. Vargas-Solar, C. Collet, N. Ibrahim, and C. Bobineau. Coordinating services for accessing and processing data in dynamic environments. In *Proceedings of the 2010 international conference on On the move to meaningful internet systems - Volume Part I*, OTM'10, pages 309–325, Berlin, Heidelberg, 2010. Springer-Verlag.

[3] Y. Gripay, F. Laforest, and J.-M. Petit. A simple (yet powerful) algebra for pervasive environments. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 359–370, New York, NY, USA, 2010. ACM.

[4] Y. Gurevich. Evolving algebras 1993: Lipari guide. pages 9–36, 1995.