IGI PUBLISHING **ITJ3971**

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.igi-global.com

This paper appears in the publication, Journal of Electronic Commerce in Organizations, Volume 6, Issue 1
edited by **Mehdi Khosrow-Pour © 2008, IGI Global**

Context-Aware Query Processing in Ad-Hoc Environments of Peers

Nikolaos Folinas, University of Ioannina, Greece
Panos Vassiliadis, University of Ioannina, Greece
Evaggelia Pitoura, University of Ioannina, Greece
Evangelos Papapetrou, University of Ioannina, Greece
Apostolos Zarras, University of Ioannina, Greece

ABSTRACT

In this article, we deal with context-aware query processing in ad-hoc peer-to-peer networks. Each peer in such an environment has a database over which users execute queries. This database involves (a) relations which are locally stored and (b) virtual relations, all the tuples of which are collected from peers that are present in the network at the time when a query is posed. The objective of our work is to perform query processing in such an environment and, to this end, we start with a formal definition of the system model. Next, we formally define SQLP, an extension of SQL that covers the termination of queries, the failure of individual peers and the semantic characteristics of the peers of such a network. Moreover, we present a query execution algorithm as well as the formal definition of all the operators that take place in a query execution plan.

Keywords: *data integration; database management systems; information and communication; middleware; mobile technologies; technologies*

INTRODUCTION

Nowadays, the synergy between network and database management systems provides opportunities for the integration and querying of various heterogeneous sources of information, spread over an ad hoc network of peers. *The fundamental topic of this article is the context-aware processing of queries in ad hoc networks of peers through Web services.* We assume the existence of a set of peers who communicate with each other, thus forming a time varying ad hoc network of peers. For reasons of interoperability, we also assume that these peers use Web services for their interactions. Each peer has a database where (a) data can be

locally stored, or (b) descriptions of data are present, in a form that allows their collection from the appropriate peers and their subsequent querying with traditional database mechanisms. The querying and/or collection of these data is dependent on the state of the peer network and on the knowledge that the peer has about this state; therefore, each time a query is posed, its processing must be adapted to this state. *In other words, the state of the peer posing a query and, most importantly, the state of its surrounding network constitutes the context under which the query is processed.*

Assume the case where several kinds of vehicles are driving in a highway. Each vehicle is a part of a global pervasive computing environment where computations can be performed, data can be exchanged between computing devices of the environment and information is interactively requested and presented to the users. Cars interact with each other through Web services, providing dynamically changing information regarding the vehicle's location, velocity and fuel deposit. Moreover, each vehicle comprises services that offer static information concerning its type and technical characteristics. On the highway, there exist exits to parking areas, which may include facilities such as gas stations, fast food restaurants, medical help, and shopping centers. Each one of these facilities also comprises Web services, which range from simple ones, reporting the existence of the facility, to more complex ones providing information regarding, for instance, the price lists, the availability of certain goods or the number of patients waiting for medical help. The users of the facilities of the pervasive environment, for example, the drivers of the vehicles, can obtain information by posing queries to global information space of the environment. For instance, they may be interested in obtaining information like the closest gas station with a price of gasoline under 2€/gallon, the closest Italian restaurant, or notifications for the average speed of all the cars ahead.

To facilitate the smooth operation of peers within the aforementioned environment, specific technical challenges must be addressed. A significant problem is the fact that traditional query processing must be reconsidered to adapt to the particularities of our computing environment. In this article, we are specifically interested in the problem of formally defining a declarative query language that enables the posing of queries over an ad hoc network of peers as well as the introduction of a mechanism for the transformation of declarative database queries to query execution plans.

First, we start with the theoretic formulation of the problem. We construct a directed graph of peers, where each node corresponds to a peer and each edge to the physical connection among two peers. The graph of peers is time varying, since nodes and edges are added or invalidated as time passes. Apart from the possibility of communication, which dictates the structure of the graph, peers are further organized in *communities*, based on their semantic similarity, or *classes*, based on the interface of Web services they support. All our deliberations are based on the *principle of local scope*, that dictates that no peer has a global knowledge of the entire graph, and therefore, all its decisions must be made depending solely on the knowledge that this node has at a given time point. Specifically, the *viewpoint of a peer* is the subset of the graph known to this peer at a given time point and the communities of the peer are sets of peers whose publicized characteristics fulfill a logical condition that classifies

them into the appropriate community. The only classification that is not local is the class of each peer: we assume that a set of finite classes exists, each with an interface comprising a set of public Web service operations that all class instances support. Every peer is created as an instance of one of the globally known classes.

With respect to the relationships among peers, each peer plays both the role of the server and the role of the client in this environment. As a server, the peer implements and exports the interface of Web service operations prescribed by its class. The other peers of the system can invoke these Web services at runtime. At the same time, the peer is responsible for answering queries posed by its users. In our framework, we discuss traditional database queries and, therefore, the peer hosts a relational database where query processing takes place. The database includes different categories of relations. First, the database includes relations that obey the traditional assumption that a database hosts locally stored relations, whose extents are finite sets of locally stored tuples. In this article, we extend this implicit assumption and assume that the extent of a relation can be spread among the peers forming the context of a peer. Therefore, only the description of the schema (or intention) of such a *virtual* relation is locally available, along with the description of the necessary Web services that must be invoked in order to locally collect the relation's extent before continuing query processing as usual. This collection procedure practically dictates that a workflow of Web services has to be executed for each peer of the viewpoint of the querying peer. Finally, a third category of relations involves hybrid relations, whose extent is partly locally stored and partly needs to be collected from the other peers.

The processing of queries in such an environment is inherently different to the traditional one. We have already mentioned the context-aware aspect of data collection for the population of virtual relations. Moreover, due to the volatile character of the state of the peer's graph, it is quite probable that the viewpoint of a peer is an inaccurate reflection of the state of the peer graph. In other words, it is quite possible that the graph has changed since the last refreshment of the viewpoint of a peer. In fact, the graph can possibly change also during the execution of a query; therefore, the processing of a query must be inherently designed to tolerate failures (i.e., Web service invocations that do not respond) and continue operating regularly. Also, due to the possible vastness of the graph, it is necessary to be able to stop collecting answers after a certain, satisfactory amount of information has been collected. Based on these fundamental differences with traditional query processing, we introduce an extension of SQL, SQL^P that allows the user to exploit the context-dependent nature of the environment by specifying the peers of interest though abstract criteria that involve their location in the graph, their community, their class, or QoS characteristics, like, for example, their availability. The usage of virtual tables is transparent in SQL^P. We exploit the previously introduced model to formally specify the semantics of SQL^P.

The processing of the queries in this extended version of SQL requires also an extension of the mechanism of query execution. Traditional relational database management systems translate the declarative SQL queries to procedural, executable plans that are expressed in the form of left-deep trees of relational operators. Therefore, we introduce novel operators, specifically tailored for the support of Web

service invocation and composition, in order to populate the virtual tables. Then, query processing can continue as usual. We have also implemented a mechanism that allows us to determine the necessary set of peers that are supposed to participate in a query and to visually display the produced plans to the user.

This article is organized as follows: in the second section, we propose SQL^P, an extension of SQL for ad hoc P2P systems. To this end, we define a system model; we investigate language requirements and propose the syntax and semantics of SQL^P. In the third section, we extend the relational algebra with novel operators and algorithms in order to map SQL^P queries to query plans. In the fourth section, we discuss implementation issues. Finally, in the fifth section, we discuss related work and in the sixth section, we conclude our results and discuss topics for future work.

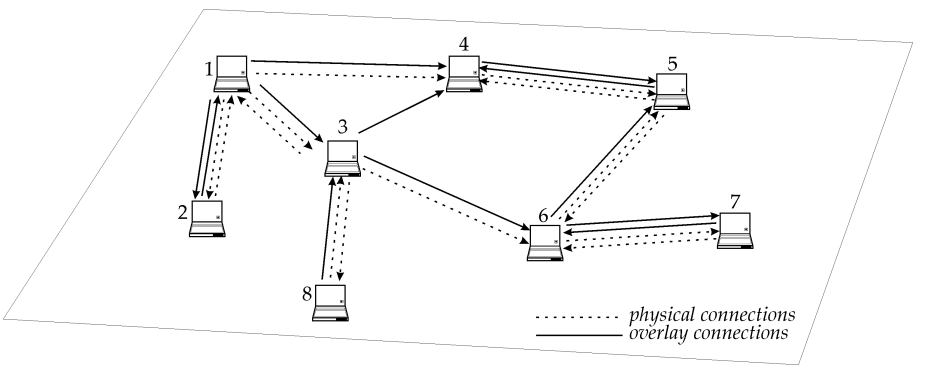
SQL FOR PEERS: SYSTEM MODEL, REQUIREMENTS, SYNTAX AND SEMANTICS

In this section, we formally define the system model. Then, we move on to formally define SQL^P, an extension of SQL for ad hoc P2P systems.

System Model

A bird's eye view of the system infrastructure is modeled by a graph $G(V,E)$, comprising a set of nodes V and a set of edges E (*Figure 1*). Each node in our system graph is a peer and each edge $e = <u,v>$ stands for the fact that node u can communicate with node v. The notion *'can communicate'* means that peer u can send data or make a request for data to v - in other words, the edge $<u,v>$ implies that peer u assumes (a) knowledge of existence and (b) network connectivity with node v. The edges are directed in the sense that although node u can communicate with v, the inverse does not hold (an edge $<v,u>$ would be required to demonstrate such a fact). This is quite frequent in modern ad hoc networks and deeply affects the design of efficient routing protocols (Abolhasan, Wysocki, & Dutkiewicz, 2004). In the sequel, we will also refer to an edge between two nodes as a *direct link*. To discriminate between different nodes, each node is characterized by a globally unique identifier, *peer id*.

Figure 1. A system's graph G(V,E)



As usually, a *path* between two nodes, say u_1 and u_2, is an acyclic sequence of consecutive edges belonging to E that connects these two nodes. The distance of two nodes, say u_1 and u_2, is the cardinality of the minimum set of edges required to reach node u_2 through a path starting at u_1. In other words, the distance of two nodes is defined by the number of hops involved in the connecting path, which is a typical assumption in ad hoc networks research. We will denote the distance of two nodes as $distance(u_1, u_2)$.

It is quite important here to stress the following properties of the system's graph:

- The graph is time varying. In other words, nodes leave or enter the system as time passes. Furthermore, nodes move randomly, causing the destruction of existent links and the establishment of new ones.
- No node has a full knowledge of the system's graph at any time point. On the contrary, it is important to design a system where each node has only a personal, restricted viewpoint of the graph. A fundamental principle in our deliberations is the *locality of peer scope*: Each peer must be designed to operate by exploiting its own knowledge of a subset of the system, without counting on some higher-level authority to provide a global viewpoint of the system.
- It is also important that each node is designed to operate under the assumption that its knowledge of the graph is both incomplete and (possibly) inaccurate. This is a disadvantage related to the current networking technology for ad hoc networks (Chlamtac, Conti, & Liu, 2003).
- The overall graph is not fully connected. In other words, it is not always possible to reach any node v of V, starting from another node u.

Context = Viewpoint of a Node

At every time instant T, a node u is aware of a subset of the system's graph, as it was configured at a previous time point $T' \leq T$. This subset of the graph is called viewpoint of node v at time T and denoted by $viewpoint(v,T)$. The subgraph $viewpoint(v,T)$ is connected. This property is recursively defined as follows:

1. $v \in viewpoint(v,T)$
2. All nodes u that are connected to a node x, $x \in viewpoint(v,T)$ through an edge (x,u) belong to $viewpoint(v,T)$. In other words, first, all nodes u that are connected to v through an edge (v,u) belong to $viewpoint(v,T)$. Then, the nodes that can be reached from these ones are also added. This is recursively continued.

Inaccuracy is inherent in this definition. Firstly, all the knowledge about direct links refers to a time point T' in the past. This means that whatever changes have happened between T and T' are obscure to v. The exact determination of time T' depends on the implemented routing protocol. Second, it is obvious that even if the overall set of nodes is finite (which is not an assumption that we have made so far), it is clear that it is impractical or even impossible to maintain all the knowledge for the graph for each node v. In fact this is the approach taken a large category of routing protocols known as on-demand routing protocols (Abolhasan et al., 2004).

Community

Apart from the physical connectivity among nodes, we can devise logical schemes for the connectivity of peers. In P2P terminology, the network of peers that share similar semantical properties is called an overlay network (Androutsellis-Theotokis & Spinellis, 2004). In our setting, a *community* of nodes is a subset of V who shares the same semantical properties. Each peer defines its own communities. Formally, semantical proximity is captured by a formula in a first-order predicate calculus. The principle of locality of a peer's scope imposes a design where each peer comprises a local set of communities, each defined as a subset of its viewpoint, upon fulfillment of the appropriate formula. Therefore, a community *comm_name* of a peer u is defined as:

$$community_{comm\_name}(u)=\{v|v\epsilon viewpoint(u,T) \text{ and } \varphi_{comm\_name}(v)=true\}$$

with φ being a formula in a first-order predicate calculus that returns true or false given the properties of a node v.

Clearly, a node u can have many communities and each node v in the viewpoint of u can belong to more than one communities of u. Moreover, assuming a simple community *Unclassified* that comprises all nodes that do not belong to any other community, the union of all communities of node u returns $viewpoint(u,T)$, at a time point T. An interesting observation here is that if two or more nodes agree for a correspondence of communities, a P2P overlay is formed.

Web Services

Each node is equipped with a set of Web service operations that it publishes, therefore giving the possibility to the rest of the nodes to invoke them. Formally, each node $u \in V$ possesses a finite set of Web service operations $WS^u=\{ws^{u1}, ws^{u2},.....,ws^{um}\}$ that are made public to the rest of the peers. In the sequel, we will not discriminate between the terms Web service operations and Web services.

Peer Classes

In the context of the integration of peers at a large scale, each peer has to resolve the problem of mapping the external interface of the other peers to its internal state. In other words, if a peer u is to invoke a Web service operation of another peer v, how does u decide the mapping of the operation's parameters or the operation's result to its internal state? Typically, there are two well-known extremes from the database community to handle this problem, as well as intermediate solutions.

- In the first extreme, a *global schema* is assumed. In distributed database systems (Ozsu & Valduriez, 1991), a global schema is assumed for the whole environment and each local database comprises a subset of the global schema. This approach requires a universal common agreement over a global schema (and the implicit semantics hidden behind it). We find this requirement too restrictive for a large-scale P2P environment that needs to be dynamically readjusted to novel peers that appear.
- An intermediate approach would be to hardcode all mappings among all peers. Still, this approach is too labor-intensive and clearly unable to scale up to the full extent of a P2P environment.
- In the second extreme, semi-automated techniques for schema matching have recently appeared in the literature. In

the context of the schema-mapping problem, where the mapping among two schemata must be discovered, semi-automated techniques have been proposed (Madhavan, Bernstein, Doan, & Halevy, 2005). Nevertheless, a certain degree of training and supervision is required for a mapping to be derived and --to the best of our knowledge-- there is no fully automated, fast method for this purpose. Therefore, although this technology would resolve the scalability problem and the ad hoc nature of the P2P environment, we cannot rely on its effectiveness for the moment.

To resolve the aforementioned problems of (a) scalability, (b) ad hoc nature of the environment and (c) schema mapping discovery, we resort to an intermediate solution that provides a reasonable balance to all the aforementioned issues. We classify peers to *peer classes*, with the members of each class exporting the same Web service operations. In other words, we assume a factory for each class, specifying the interface for each deployed instance.

We assume a traditional tree-based hierarchy of classes. Each subclass has a single superclass, whose interface it extends. All instances of the subclass are also instances of the superclass. Each node (a) *directly* belongs to exactly one class and (b) *indirectly* belongs to all the classes of the path that starts in the root and ends in its containing class in the tree of the class hierarchy. We call the set of nodes that directly belong to a class *immediate extent* and the set of nodes that indirectly belong to a class (due to its subclasses) the extended extent. Classes that do not have any descendants are called *base*, or *leaf classes*. We denote the interface of a class C by *interface*(*C*) and its immediate and extended extents as $extent_i(C)$ and $extent_e(C)$.

In *Figure 2* we can see the base classes *VW*, *BMW*, *TOYOTA*, *SHELL*, *BP*, *HOTEL*, and *RESTAURANT* with their respective nodes. In *Figure 3* we can also observe the superclass *CARS* on top of the classes *VW*, *BMW* and *TOYOTA* and a class *GAS STATION* as a superclass of *SHELL* and *BP*.

The aforementioned problems of integration are resolved in a balanced fashion. With respect to the scale-up of the environment, the integration problem is only dependent on the number of peer classes and not on the number of their instances. Although we anticipate a reasonably small number of peer classes, still the problem of integration is present. We assume a hard-coded, intermediate solution between pairs of classes. This does not necessarily require that all classes be mapped to each other; the only effect of the absence of a mapping would be that two instances belonging to non-reconciled classes could not query each other without a total failure of the system. Moreover, it is straightforward to devise mechanisms for incremental updates of class mappings for the deployed instances, so that, as new classes are added and the interfaces of old classes are updated, the deployed instances are informed on the new situation. With respect to the ad hoc nature of the P2P environment, the problem of class integration is orthogonal and not affected. The last problem, discovery of schema mappings is resolved at the factory level (although we recognize that we still need the same amount of coding effort as in traditional mediator-wrapper environments).

Figure 2. Base classes with their corresponding nodes

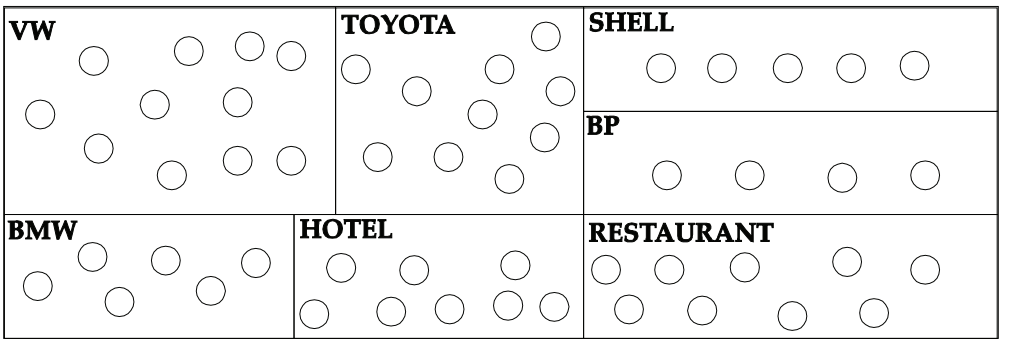
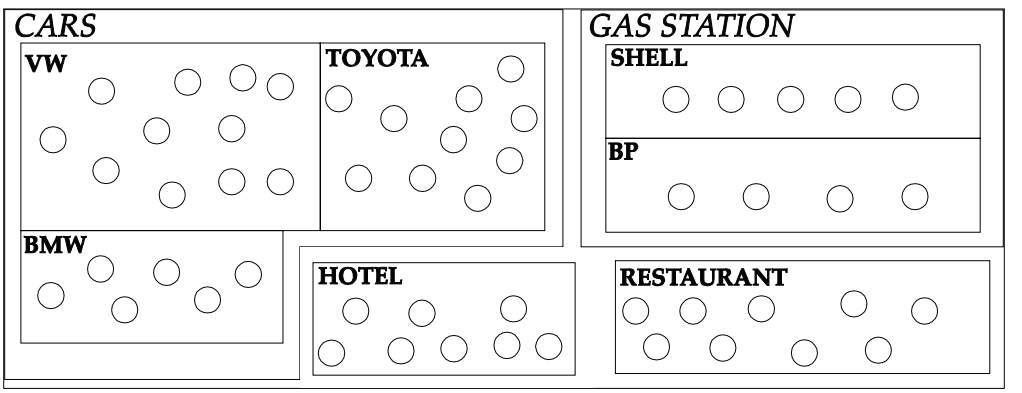


Figure 3. A hierarchy of classes with their corresponding nodes



Difference between Classes and Communities

The class of a node is an inherent property of the node, determined once and for all at the creation of the node, mainly for integration purposes, whereas the community (or communities) to which it belongs is a potentially time varying property that is determined individually by the other peers and is mainly used for querying purposes.

Clock

Each peer has its own clock. The clocks of the peers are not necessarily synchronized.

Peer Database

Each peer has a database, which comprises a set of relations. Each relation has a *schema* or *intention* comprised of a finite set of distinct attribute names. Also, each relation has an extension, which is a finite subset of the Cartesian product of the domains of the attributes of the relation's schema. The relations of a peer's database are classified in the following categories:

- **Locally stored (or local) relations:** Local relations are relations whose extension involves tuples that are locally stored at the peer that carries the relations' database. In other words,

local relations are exactly the same as in traditional relational databases.
- *Virtual relations*: Virtual relations are relations whose schema is fixed and locally known, but whose extension is not locally stored in the database of the peer. On the contrary, the extension of a virtual relation is collected from the appropriate peers at query time. Practically, this means that each time a user poses a query involving a virtual relation, the peer determines the set of peers who are to be contacted (along with the appropriate sequence of Web service operations of these peers that are to be invoked), collects the respective tuples, transforms them to the schema of the virtual relations, and, finally, stores (or "materializes") them. Then, query processing can be performed as usual.
- *Hybrid relations*: Hybrid relations are variants whose extension includes both locally stored tuples and tuples to be collected from other peers.

Each tuple collected for a relation belonging to the last two categories is tagged with a *timestamp*, produced by the clock of the node that receives the incoming tuple. The timestamp corresponds to the transaction time of the tuple, that is, the exact time point of its entrance to the receiver's database. A tuple's timestamp will be used for caching purposes.

Peer Characteristics

Each peer is characterized by several properties that can either be determined by the peer itself or by the class to which it belongs. Specifically, the characteristics that we adopt are:

- **(Average) Availability**, that is, the probability that the peer is operational at a given time instant.
- **(Average) Response Time,** that is, the average time needed for a Web service operation of the peer to execute.

Peer's System Catalog

Each node u needs a system catalog for its proper operation. The catalog includes useful information about the nodes known to u. Specifically, this information refers to:

- Class of the other nodes.
- Communities of the other nodes.
- Distance from other nodes.
- Node characteristics, like availability, and response time.

Results Collection from Other Peers

In this subsection we discuss issues of tuple collection for the virtual and hybrid relations. First, we formally introduce workflows of Web service operations. Next we discuss how the mapping of the workflow's result to a peer's relation is performed and finally, we formalize issues of result materialization.

Workflow $wf^{u.R}(u_i)$

Assume a peer u that poses a query and invokes Web service operations from a set of peers $u_1, u_2, ..., u_z$ in order to collect their tuples. In principle, it is quite possible that the requested information from a certain peer can only be obtained after the invocation of a workflow of Web service operations (rather than a single operation). For example, assume that a peer using the European metric system collects the velocities of other peers of class *CAR*, and a certain class of cars returns miles instead of kilometers. The conversion can be per-

formed through a simple BPEL workflow. We denote each of these workflows as $wf^{u.R}(u_i)$, with $1 \leq i \leq z$. Each such workflow w is an acyclic directed graph $G_w(V_w,E_w)$, with operations being modeled as nodes and edges being the representatives of control passing. Edges are tagged with the conditions under which they are fired at runtime. Each workflow has also a flat relational schema that includes a set of attributes that result from the possible un-nesting of the XML elements of the final message delivered by the workflow. Finally, the workflow has an extension, dynamically created at runtime, that instantiates the aforementioned schema.

Mapping of Other Peers' Web Services to Virtual Relations

In this paragraph, we formally discuss the mechanism that allows peers to collect tuples from the peers of their viewpoint. Assume a peer u that poses a query and invokes Web service operations from a set of peers $u_1, u_2,....,u_z$ in order to collect their tuples. The application of the workflow $wf^{u.R}(u_i)$ results to a set of tuples under the schema (B_1, B_2, ..., B_m), possibly after a set of XML un-nesting operations. Assume $R(A_1, A_2,...., A_n)$ to be the schema of R, the mapping between the two schemata is a function f_{map}, with f_{map}: $(A_1, A_2,...., A_n) \times (B_1, B_2, ..., B_m) \rightarrow \{true,false\}$. We impose the constraint that for each A_i, $1 \leq i \leq n$, there exists at most one B_j, $1 \leq j \leq m$, to which A_i is mapped. As usual, all attributes of the workflow schema that are not mapped to the schema of the target relation are projected out, whereas all the relation's attributes that are not populated by the workflow are filled with NULL values. The following example clarifies the aforementioned process. Assume the relation $R(E\_ID, E\_SALARY, E\_AGE)$ in the database of node u and let the workflow that is mapped to R for node v have the schema (ID,AGE,$NAME$). The workflow provides no information on salaries and the database does not store any data on names. Therefore, our mappings resulting to true are:

$$f_{map}(E\_ID,ID)=true,$$
$$f_{map}(E\_AGE,AGE)=true,$$

with the rest of all the other possible mappings of the Cartesian product of the two schema being evaluated to false. The transformation at an instance level is simple: (a) we project out all unnecessary workflow attributes, (b) we introduce NULL-valued attributes for the relation's attributes for which no workflow attribute exists, (c) we appropriately re-order the attributes of the workflow schema to match the relation's attributes and (d) we populate the target table.

Full-Partial Materialization

Whenever a workflow is executed for a certain peer and the produced results are successfully stored at the extent of the target virtual relation, we say that we have materialized these results. The fact that the results of a certain workflow for peer u_i have been materialized at the relation R of peer u is denoted as $(wf^{u.R}(u_i))$. Full materialization for a relation R of a peer u is the state of a query when all workflows for all the peers that have been selected to populate R have been successfully executed. We denote full materialization by $M(u.R)$. Assuming V^{all} as the set of these identified peers, we can formally define full materialization as $M(u.R)= U (wf^{u.R}(u_i))$, with $u_i \in V^{all}$.

Partial materialization

For a relation R of a peer u is the state of a query when the workflows for a clean

subset of the peers that have been selected to populate R have been successfully executed. We denote partial materialization by $M^p(u.R)$. Assuming V^{all} be the set of the peers that have been selected to participate in the population of R, and V^t as the set of the peers whose results have been successfully materialized, we can formally define partial materialization as $M(u.R) = U\ (wf^{u.R}(u_i))$, with $u_i \in V^t$, $V^t \subset V^{all}$.

SQL^P: An Extension of SQL for Ad Hoc P2P Networks

In this section, we discuss the extension of SQL that we introduce. The proposed language *SQL*^{P} (SQL for Peers) implements all the aforementioned requirements. *Figure 4* presents the general structure of an *SQL*^{P} query. We use [...] to refer to optional parts of the language and the expression **AND/OR** to signify that different clauses can be connected through one of these logical connectors.

Querying the Graph of Peers

Assume a query Q submitted at node u at the time point T. Let $\{R_1, R_2, \ldots, R_n\}$ be the relations that participate in the FROM clause of the query. Then, we can write the query as: $Q(R_1, R_2, \ldots, R_n)$. Without loss of generality, we can assume that the first k relations $R_1, R_2, \ldots, R_k$, $k \leq n$, are virtual or hybrid. In order to be able to define the semantics of the query properly we need to materialize these relations and then, execute the query over their collected extent as usually. Nevertheless, before specifying this semantics, we need to define the following concepts.

Figure 4. The generic syntax of a query in SQL^P

SELECT <projected columns>
FROM <relations involved in the query>
[WHERE <conditions to be satisfied>**]**
[GROUP BY <grouper attributes>**]**
[HAVING <post-grouping condition>**]**
[ORDER BY <sorter attributes>**]**
[WITH
  [HORIZON <peer selection condition: *LOCAL*, or *COMMUNITY* <*community name*> or *HOPS* θ value θ ∈ {< , > , = , ≤, ≥}, or *PEERS*=[$peer_1$,$peer_2$,…, $peer_n$]>
  ***AND / OR**]**
  [AVAILABILITY <minimum availability of peers to be contacted>
  ***AND / OR**]
  [RESPONSE_TIME <maximum tolerable response time>
  ***AND / OR**]**
  [CLASS <class name> ***AND or OR**]]
  [AGE <maximum allowed caching time for previously collected tuples > ***AND / OR**]
  [TIMING <query timing: *AD-HOC* or *CONTINUOUS* > < tuning clause>
    If *AD-HOC* tuning clause refers to successful termination:
      PEERS_PERCENTAGE, or *AMOUNT_TUPLES*, or *TIMEOUT*
    If *CONTINUOUS* tuning clause refers to update mode:
      PULL_BASED WITH PERIOD, or *PUSH_BASED*>**]**
]

Peers of Interest

The query Q, posed over peer u is divided in three parts. The first part is composed of the traditional SQL clauses, the second part comprises the clauses of our extension that occur after the keyword WITH that have the purpose of determining which peers are to be contacted, and the third part concerns the timing of the query.

The second part of the query depends on criteria like the horizon of the query of the graph of the viewpoint of peer u (HORIZON), QoS characteristics (AVAILABILITY, RESPONSE_TIME), the class of the peers (CLASS) and the age of the stored tuples in the virtual relations (i.e., if a peer has been recently contacted, as specified by the AGE clause, it is not necessary to contact it again). Remember that, due to the nature of the interaction among peers, it is not feasible to simply broadcast a request for tuples; on the contrary, specific Web service operations must be invoked on the specific port types of the peers.

In terms of semantics, we divide the second part into atomic conditions, logically connected through the connectors AND and OR. Assuming that these atomic conditions are C_1, C_2, …, C_r, the non-traditional part of the query can be rewritten in a disjunctive normal form, that is, a disjunction of conjunctive conditions.

The interesting aspect of this part is that a preparatory query must be performed over the system catalog to determine specifically which peers must be contacted in order to materialize the virtual relations. Contacting a peer means that for each virtual/hybrid relation in the FROM clause of the query the execution of the appropriate workflow must be initiated. In terms of semantics, each atomic condition specifies a set of peers of the viewpoint of u that qualify to be contacted. Given an atomic condition C, we define the set of peers of interest $V^u(C)$ to be the set of peers that belong to the catalog of peer u that fulfill C. Specifically, given a time point T for a query Q containing C,

$$V^u(C) = \{ v \mid v \in viewpoint(u,T): C(v) = true \}.$$

We do not involve timepoint T to avoid overloading the notation. Having defined the peers of interest for an atomic condition, it is straightforward to obtain the set of peers of a composite condition in disjunctive normal form: the intersection of the peers of interest of the atomic conditions produces the peer sets of each conjunct; these sets are subsequently ORed to produce the final set of peers of interest of the query, which are to be contacted.

Now, we are ready to define the semantics of each individual clause concerning the determination of the peers of interest.

- **HORIZON**: The condition of the HORIZON clause determines the peers of interest on the basis of the position in the graph, or their semantical characteristics. The clause allows several possibilities to the users. Assuming that the condition of the HORIZON clause is C_1, and $VH^u(C_1)$ is the resulting set of peers of interest, we can specify $VH^u(C_1)$ for each of the following possibilities that SQL^P offers:
  1. The only peer of interest is the local querying peer (C_1: LOCAL) $VH^u(C_1) = \{ u \}$
  2. The peers of interest are the ones of a certain community of the peer (C_1: COMMUNITY <C_NAME>) $VH^u(C_1) = \{ v \mid v \in viewpoint(u,T): v \in community(C\_NAME,u) \}$

3. A radius of a certain number of hops dictates the peers of interest (C_1: *HOPS* θ *value*, with $\theta \in \{ =, <, \leq, >, \geq \}$)
   $VH^u(C_1) = \{ v \mid v \in viewpoint(u,T): distance(u,v)\ \theta\ value$, with $\theta \in \{ =, <, \leq, >, \geq \} \}$
4. A set of peer ids, that is, a set of specifically requested peers, determines the peers of interest (C_1: $PEERS = \{peer_1, peer_2, ..., peer_n\}$)
   $VH^u(C_1) = \{ v \mid v \in viewpoint(u,T): v \in \{peer_1, peer_2, ..., peer_n\} \}$

All the necessary information for the evaluation of any of the aforementioned atomic conditions is found in the system catalog of u.

- **Quality of service:** The clauses concerning the AVAILABILITY and RESPONSE TIME of the peers of interest aim to guarantee a certain level of quality of service for the peer posing a query.
- **CLASS:** It is possible that we only need to query the peers of a certain class. Classes carry both structural typing information (as they statically define the interface of their instances), but also semantic information (as collections of semantically—therefore structurally—similar instances). In SQL^P, it is easy to specify an atomic condition that restricts the peers of interest to a certain class, by giving a condition of the form C_4: *CLASS* = *class_name*. Assuming $VC^u(C_4)$ the result set of peers of interest, and *class*(v) a function that returns the class of each peer from the system catalog of the querying peer, the resulting set of peers of interest is formally defined as:

  $VC^u(C_4) = \{ v \mid v \in viewpoint(u,T): class(v) = class\_name \}$

- **AGE:** Apart from the constraining of peers where their properties are taken as criteria for their inclusion in the resulting set of peers of interest, we can perform some form of caching in the extents of the collected tuples for virtual or hybrid relations. In other words, assuming that a peer is frequently queried, it is not obligatory to pay the price of invoking its Web service operations, executing the data transformation workflow and materializing the same results again and again, but rather, it is resource efficient to cache its previous results. The AGE clause of SQL^P provides the possibility of specifying a maximum caching age for incoming tuples in a virtual/hybrid relation.
- **Query Timing**: Having clarified the general mechanism for the determination of peers of interest, we move on to provide the specification for the timing of queries. Fundamentally, we have two modes of operation: ad hoc or continuous. Each mode has its own tuning parameters:
  - If the query is continuous, this means that the user is continuously notified on the status of the query result.
  - If the query is ad hoc, the query eventually has to terminate. Differently from traditional query processing (which operates on finite sets of always available, locally stored tuples), we need to tune the conditions that signify termination of a query that has been late to complete its operation, either due to peer failures, or the size of the peer's graph. To capture

these exceptions, we can terminate a query upon (a) the completion of a timeout period of execution, (b) the materialization of a certain amount of tuples that the user judges as satisfactory for his information, or (c) the collection of responses from a certain percentage of peers that were initially contacted. In all these cases, the execution of the workflows whose results have not been materialized is interrupted, the rest of the query is executed as usual and the user is presented with a partial—still, non-empty—answer.

Query Execution

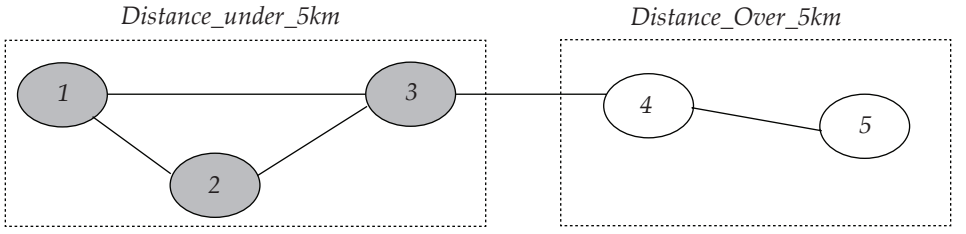
At this point we can describe the exact set of steps for executing a query. Suppose that at random time T, a query Q is performed by node u. Let $\{R_1, R_2, ..., R_n\}$ be the relations involved in query Q. Then the query can be written in the form: $Q(R_1, R_2, ..., R_n)$. We can assume that the relations R_1, R_2, ..., R_k, with $k \leq n$ are virtual or hybrid, without any impact on the generality. All tables R_1, R_2, ..., R_k must be filled with tuples. The procedure is the same for all tables; therefore we will present it only for table R_1.

The first step is to determine the set of target peers for node u that performs the query ($V^u(C)$), by evaluating C over the set of peers belonging the viewpoint of u (*viewpoint*(u)). C comprises of the conditions located at the clauses AGE, HORIZON, AVAILABILITY, RESPONSE_TIME and CLASS.

Let $V^u(C) = \{ u_1, u_2, ..., u_m \}$. For each node of $V^u(C)$ the appropriate Web services are invoked in order to require the appropriate tuples. Let also $wf^{a.R1}(u_1), wf^{a.R1}(u_2), ..., wf^{a.R1}(u_m)$, be the appropriate workflows of the peers belonging to $V^u(C)$.

The schema of each workflow is matched to the schema of relation R_1, which is the target relation. In the following, the clause TIMING is evaluated to determine the execution mode of the query (continuous or ad hoc) and the completion condition of the query. The next step is to attempt the execution of $wf^{a.R1}(u_i)$ and then perform a full or partial materialization of R_1, which is located in u, according to the query completion condition, which was mentioned before. Table R_1 is populated with the appropriate tuples and is ready to be queried. The same procedure is performed for all other virtual or hybrid tables. Therefore all tables of u are ready to be queried. At this point the query of u is performed over tables $R_1, R_2, ..., R_n$ based on traditional database methodology.

Figure 5. Graph configuration for query posing



Examples

In the rest of this section, we will present examples of SQL^P. Assume a peer network of the topology of *Figure 5*, consisting of 5 peers, each representing a car in the highway. Queries are posed to peer p_1, that classifies the rest of the peers in two communities, (a) the community of dark-shaded close peers (*Distance_Under_5km*) and (b) the community of light-shaded, distant peers (*Distance_Over_5km*). Peer p_1 is informed on the existence and connectivity of the rest of the peers through the underlying routing protocol that operates as a black box in our setting.

Peer p_1 carries a database consisting of two relations with the following schemata:

CARS(*ID*, *PLATE*, *BRAND*, *VEL*)
BRANDS(*BRAND*, *COUNTRY*, *METRIC_SYSTEM*)

The first relation describes the information collected from the peers contacted (and mainly serves queries about the velocity of the cars in the context of the querying peer). This relation CARS is virtual: each time a query is posed, tuples must be collected from the context of peer p_1 to populate it. The attribute BRAND is a foreign key to the relation BRANDS that is static and locally stored. Primary keys are underlined and the semantics of the attributes are the obvious ones. In the sequel, we give examples of SQL^P queries over the abovementioned environment.

Example 1

By this example we illustrate different situations where we can determine the peer nodes to which the query is addressed. Different strategies may be used for choosing the peers to query. In any case the decision is based on characteristics of the peers such as availability, response time, class of Web services implemented, and so forth. Peer p_1 wishes to know the license number, velocity and manufacturing country of all cars belonging to its community. Furthermore, the peer that poses the query wishes to limit it to those peers that: (a) are located no more than 5 Km away (**Distance_Under_5km**), (b) their availability is more than 60%, (c) their response time is less than 4 seconds and finally, (d) implement the European class of Web services. The syntax of the examined query is depicted in *Figure 6*.

Example 2

Peer p_1 wishes to know the license number, velocity and manufacturing country of all cars. The peer also wishes to complete the query when more than 70% percent of the target peers have replies successfully (*Figure 7*). To determine the target peers, the requesting peer selects the peers based on its catalog and according to their response time. The execution of the query stops when the requested percentage of 70% in our case is satisfied.

Example 3

Peer p_1 wishes to know the license number, velocity and manufacturing country of all cars. The peer also wishes to complete the query when more than 5 tuples have been collected for the relation CARS (*Figure 8*). The requesting peer contacts each peer that appears in its catalog. This procedure ends when the count of currently collected tuples becomes greater or equal to the posed limit.

Example 4

Peer p_1 wishes to know the license number, velocity and manufacturing country of all cars. The peer also wishes to complete the

Figure 6. Query 1

```
SELECT          CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM            CARS, BRANDS
WHERE           CARS.BRAND=BRANDS.BRAND
WITH
HORIZON         COMMUNITY Distance_Under_5km AND
AVAILABILITY    > 60% AND
RESPONSE_TIME   < 4.0 AND
CLASS           = 'european'
```

Figure 7. Query 2

```
SELECT      CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM        CARS, BRANDS
WHERE       CARS.BRAND=BRANDS.BRAND
WITH
TIMING      AD-HOC PEERS_PERCENTAGE > 70%
```

Figure 8. Query 3

```
SELECT      CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM        CARS, BRANDS
WHERE       CARS.BRAND=BRANDS.BRAND
WITH
TIMING      AD-HOC AMOUNT_TUPLES > 5
```

Figure 9. Query 4

```
SELECT      CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM        CARS, BRANDS
WHERE       CARS.BRAND=BRANDS.BRAND
WITH
TIMING      AD-HOC TIMEOUT > 7
```

query within a timeout of 7 seconds (*Figure 9*). The requesting peer contacts each peer that appears in its catalog. This procedure ends when the timeout is reached.

QUERY PROCESSING FOR *SQL*^{P} QUERIES

In this section, we deal with the problem of mapping the declarative *SQL*^{P} queries

to executable query plans. As already mentioned, the execution of traditional SQL queries relies on their mapping to left-deep trees whose leaves are database relations, internal nodes are operators of the relational algebra and edges signify pipeline of the results of a node to another. Clearly, since we raise fundamental assumptions of traditional database querying, such as the finiteness and locality of tuples as well as the conditions under which a query terminates, we need to extend both the set of operators that take part in a query and the way the query tree is constructed. In this section, we start by introducing the novel operators for query processing. Next, we discuss how we algorithmically determine the set of peers of interest and, finally, we discuss the execution of a query.

Novel Operators

In this subsection, we start with the operators that participate in SQL^P query plans. We directly adopt the *Project*, *Select*, *Group*, *Order*, *Union*, *Intersection*, *Difference* and *Join* operators from traditional relational algebra and move on to define new operators. First, we discuss operators that are used to construct the set of peers of interest. Then, we present the operators that actually take part in a query plan.

- **Operators applicable to the catalog of a peer:**
  - **Check_Tables:** Operator *Check_Tables* determines whether the tables belonging to the FROM clause of a query are virtual, hybrid or local. The input to the operator is the FROM clause of the query and the output is the same list of tables, each annotated with the category to which it belongs.
  - **Check_Peers:** This is a composite operator that applies the procedure mentioned in the second section for the determination of a set of peers out of a condition in disjunctive normal form. All clauses of the form HORIZON, AVAILABILITY, RESPONSE_TIME and CLASS are evaluated over the catalog through a *Check_Peers* operator and the set of peers of interest is determined by combining the results of these operators through the appropriate *Unions* and *Intersections*.
  - **Check_Age:** The *Check_Age* operator is also an operator used to determine the set of peers of interest. For each relation that hosts transaction time and producing peer attributes, an invocation of the *Check_Age* operator scans the extent of the relation, and identifies the appropriate tuples and their peers. The output is passed to the appropriate Difference operator that subtracts the identified peers from the previously determined set of peers of interest.
- **Operators that participate in query plans:**
  - **Call_WS:** This operator is responsible for dynamically determining which Web service operation, over which port type, of a specific peer must be invoked. Each Web service of a peer to be invoked is practically wrapped by this operator. The result is collected and forwarded to the operator managing the execution of a workflow of Web services.
  - **Wrapper_Pop:** This operator

is used in order to support the monitoring and execution of the workflow of Web services that populate a virtual or hybrid table. For each peer contacted in order to populate a certain virtual/hybrid relation, a *Wrapper_Pop* operator is introduced. Once the final XML result has been computed, its tuples are transformed to the schema of the target relation.

- **Fill:** A *Fill* operator is introduced for each virtual relation. The operator takes as input all the results of the underlying *Wrapper_Pop* operators (one for each peer of interest) and coordinates their materialization. Also, *Fill* checks the necessary conditions concerning the timing and termination of the query and, whenever termination is required, it signals its populating operators appropriately.
- **ExAg (Execute Again):** This operator is useful only in continuous queries and practically restarts query execution whenever the query period is completed.

Construction of the Query Tree

In this paragraph we discuss a simple algorithm to generate the tree of the query plan. Assume that a query is posed to peer p_1 and its viewpoint comprises n peers, specifically p_1, p_2, ..., p_n. The algorithm for the construction of the query tree is a bottom up algorithm that builds the tree from the leaves to the top and is described as follows:

1. We discover the virtual or hybrid relations that participate in the query. A specific sub-tree will be constructed for each of them.
2. We determine the set of peers of interest. For each peer that participates in the population of a certain relation, the leaves of the respective sub-tree are nodes representing the peer to be contacted. To keep the tree-like form of the plan, each peer can be replicated in each sub-tree to which it participates; nevertheless, each peer can also be modeled by a single node without any significant impact to the execution of the query.
3. We introduce a *Wrapper_Pop* for each peer that coordinates all the *Call_WS* operators that pertain to the operations of the peer. Between the peer node and the *Wrapper_Pop*, we introduce the appropriate *Call_WS* operators.
4. For each virtual or hybrid relations we introduce a *Fill* operator that combines the output of all the respective *Wrapper_Pop* operators; therefore it is their immediate ancestor.
5. Having introduced the *Fill* operators, the virtual or hybrid relations can be materialized and act as local ones. Therefore, the rest of the query tree is built as in traditional query processing.
6. If the query is continuous, we add an appropriate *ExAg* operator at the top.

Execution of a Query though the Query Tree

The execution of the query follows a simple strategy. First, we materialize the virtual/hybrid relations. Then, we execute the query as usual. Clearly, although this is not the best possible strategy for all cases (especially when only non-blocking operators are involved), we find that performing further optimizations is an orthogonal problem, already dealt in the context of blocking operators for streaming data (Babcock et al.,

2002). Therefore, in this article we consider only this baseline strategy since all relevant results can directly be introduced in the optimizer module of a peer. Specifically, the set of steps to follow for the execution of the query are:

1. All the *Call_WS* operators are activated and the appropriate services are invoked.
2. The *Wrapper_Pop* operators collect the incoming XML results and queue them towards the appropriate *Fill* operators that further push them towards the extents of the relations in the hard disk. This is performed in a pipelined fashion.
3. Once all virtual/hybrid relations have been materialized, the rest of the query plan is a traditional left-deep tree that executes as usual.

Example

In the following, we discuss the construction of the query plan for the query of *Figure 10*.

1. Step 1: The query involves two tables, CARS and BRANDS. The application of the operator CHECK_TABLES over the two relations results in the determination that the first is a hybrid one and the second a locally stored one.
2. Step 2: The operator CHECK_PEERS is applied to the catalog of peer p_1, in order to determine the peers of interest of the query. Taking into consideration the age of tuples found in relation CARS and the system catalog, the peer p_1 decides that the peers of interest are peers *2* and *8*.
3. Step 3: The operator CALL_WS is applied over each peer of interest.
4. Step 4: For each peer over which a CALL_WS is applied, we apply the operator WRAPPER_POP to coordinate the execution of its operations.
5. Step 5: The operator FILL is applied for the result of each WRAPPER_POP.
6. Step 6: The rest of the query plan is constructed as usual, with the only difference that the sub-tree of relation CARS is the one constructed in the previous steps.

Figure 10. Query for which the plan is to be constructed

```
SELECT          CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM            CARS, BRANDS
WHERE           CARS.BRAND=BRANDS.BRAND
WITH
AGE             < 5 AND
HORIZON         COMMUNITY DISTANCE_UNDER_5KM AND
TIMING          CONTINUOUS PULL_BASED_PERIOD = 7 AND
AVAILABILITY    > 60% AND
RESPONSE_TIME   < 3.0 AND
CLASS           = 'european'
```

Figure 11. Query plan for the aforementioned query of Figure 10

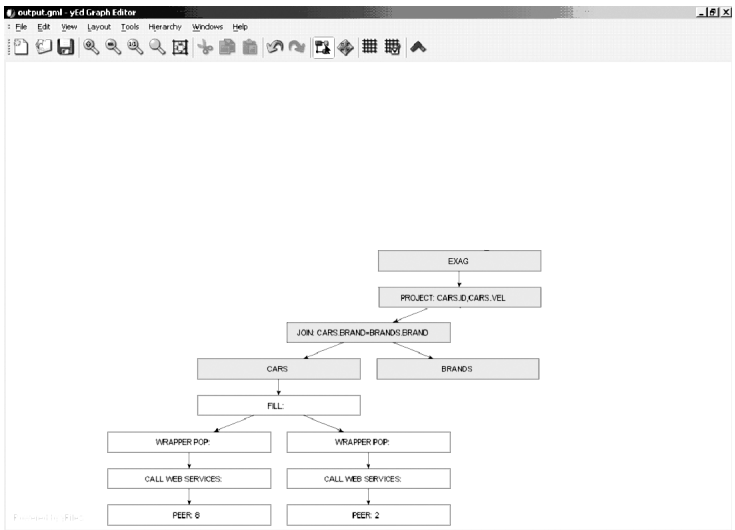
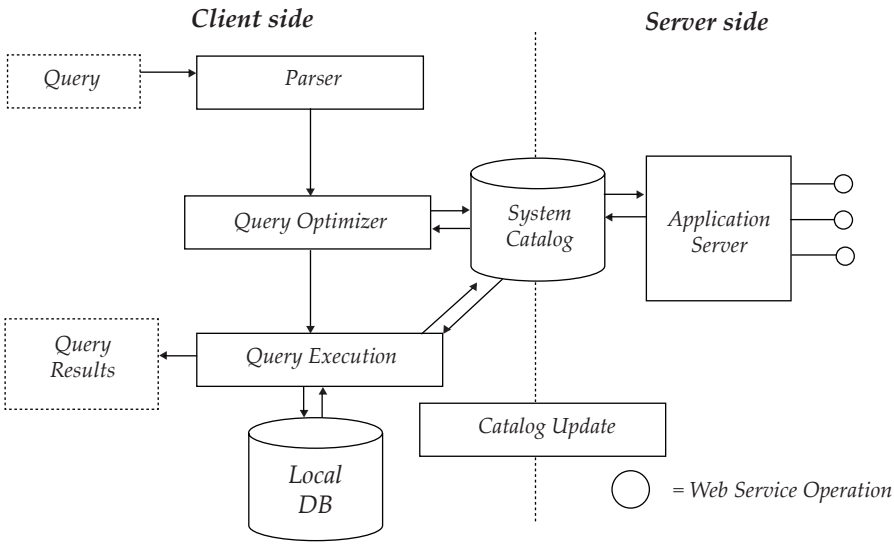


Figure 12. System architecture



IMPLEMENTATION

Figure 12 shows the full-blown architecture required to support our approach for context-aware query processing in ad hoc environments of peers. The elements shown in the figure are divided with respect to the client and the server roles played by peers. To play the client role, a peer comprises a traditional query processing architecture, involving a parser, an optimizer and a query processor. A local database and the system catalog complement the ingredients of the client part of a peer. Playing the server role amounts in publishing a set of Web services,

hosted by an application server, which is responsible for their proper execution. As usual, whenever a query is posed, the parser is the first module that is fired. The optimizer produces alternative plans out of which the best, with respect to a given cost model, is chosen. The query execution engine executes the query over the local database and returns the results.

Our first prototype implementation does not currently support the query optimizer subsystem. Instead, standard query plans are produced after parsing the user queries. The query execution subsystem includes a mechanism that allows visualizing the aforementioned plans. *Figure 11* gives a visualized execution plan through the Yed tool that graphically presents graphs.

Populating and updating the contents of the system catalog is done either statically, or dynamically. In the former case, the peer is responsible for updating the catalog through a catalog-specific API. The static update of the catalog takes advantage of the possible availability of peer-specific dynamic service discovery mechanisms. Such mechanisms may be exploited by the peer itself which takes further charge of updating the catalog accordingly.

The dynamic catalog update is realized by the catalog update subsystem, which relies on WSAMI, a middleware platform for mobile Web services (Issarny et al., 2005). WSAMI provides the naming & directory service that allows the dynamic discovery of Web services provided in mobile computing environments. Specifically, WSAMI is based on an SLP server – that is, an implementation of the standard SLP (http://www.openslp.com) protocol--for the discovery of networked entities in mobile computing environments.

RELATED WORK

The work that is closely related with the proposed approach for context-aware query processing over ad hoc environments of peers can be categorized into work concerning the fundamentals of heterogeneous database systems, context-aware computing and approaches that specifically focus on context-aware service-oriented computing. The prominent approaches that fall in the aforementioned categories are briefly summarized in the remainder of this section.

Heterogeneous Database Systems

Our approach for querying of ad hoc environments of peers bears some similarity with the traditional wrapper-mediator architectures used in heterogeneous database systems (Roth & Schwarz, 1997; Haas et al., 1997). Such systems consist of a number of heterogeneous data sources. The user of the system has the illusion of a homogeneous data schema, which is actually realized by the wrapper-mediator architecture. In particular, each data source is associated with a wrapper. The wrapper encapsulates the data source under a well-defined interface that allows executing queries. Each user query is translated by the mediator into data source specific queries, executed by corresponding wrappers. As opposed to traditional heterogeneous database systems, in the environments we examine the roles of users and data sources are not discrete. Each peer is a heterogeneous data source offering information to other peers that play the role of the user. Therefore, each peer may eventually serve as a data source and a user issuing queries. Analogous to the wrapper elements in our case is the Web services that give access to peers playing the role of data sources. Analogous to the mediator element is the hybrid relation mapping procedure

that executes workflows on Web services. In simple words, a traditional heterogeneous database system is a "1 mediator to N wrappers architecture." An ad hoc environment of peers in our case is an "N mediator to N wrappers architecture."

Another fundamental difference between the environments we examine and traditional heterogeneous database systems is that in our case the cardinality and the contents of the set of data sources may constantly change.

Context-Aware Computing and Infrastructures

In Dey (2001), context is defined as any information that can be used to characterize the interaction between a user and an application, including the user and the application. Several middleware infrastructures follow this definition toward enabling context reasoning and management (Fahy & Clarke, 2004; Chen, Finin, & Joshi, 2003; Chan & Chuang, 2003; Capra, Emmerich, & Mascolo, 2003; Gu, Pung, & Zhang, 2005; Roman et al., 2002). Amongst these approaches, CASS (Fahy & Clarke, 2004) bears some similarity with our approach, since context is modeled in terms of a relational data model. However, in our approach we do not assume centralized information management and virtual relations are dynamically compiled.

Context-Aware Service-Oriented Computing

In general, the integration of context-awareness and service-orientation just began to gain the attention of the corresponding research communities. In Keidl & Kemper (2004), for instance, the authors introduce ways for associating context to Web service invocations. In Maamar, Mostefaoui, & Mahmoud (2005) the authors go one step further by examining the problem of customizing Web service compositions with respect to contextual information. Web service execution is customized according to different types of context. Similarly, in Zahreddine & Mahmoud (2005) the authors propose a framework for dynamic context-aware service discovery and composition. Specifically, contextual information regarding the technical characteristics of user devices is used towards discovering services that match these characteristics.

CONCLUSION AND FUTURE WORK

In this article, we have dealt with context-aware query processing in ad-hoc peer-to-peer networks. Each peer in such an environment has a database over which users want to execute queries. This database involves (a) relations which are locally stored and (b) relations which are virtual or hybrid. In the case of virtual relations, all the tuples of the relation are collected from peers that are present in the network at the time when the query is posed. Hybrid relations involve both locally stored tuples and tuples collected from the network. The collaboration among peers is performed through Web services. The integration of the external data, before they are locally collected to a peer's database, is performed though a workflow of operations. We do not perform query processing in the traditional way, but rather, we involve context-aware query processing techniques that exploit the neighborhood of each peer and the Web service infrastructure that deals with the heterogeneity of peers. In this setting, we have formally defined the system model for SQL^P, an extension of traditional SQL on the basis of contextual environment requirements that concern the termination

of queries, the failure of individual peers and the semantic characteristics of the peers of the network. We have precisely defined the semantics of the language *SQL*^{P}. We have also discussed issues of data integration, performed through workflows of Web services. Moreover, we have presented an initial query execution algorithm as well as the typical definition of all the operators that can take place in a query execution plan. A prototype implementation that is implemented is also discussed.

ACKNOWLEDGMENT

This research is co-funded by the European Union - European Social Fund (ESF) & National Sources, in the framework of the program "Pythagoras II" of the "Operational Program for Education and Initial Vocational Training" of the 3rd Community Support Framework of the Hellenic Ministry of Education.

REFERENCES

Abolhasan, M., Wysocki, T., & Dutkiewicz, E. (2004). A review of routing protocols for mobile ad hoc networks. *Ad Hoc Networks*, *2*, 1-22.

Androutsellis-Theotokis, S., & Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, *36*(*4*), 335-371.

Babcock, B., Babu, S., Datar, M., Motwani, R., & Widom, J. (2002, June). Models and issues in data stream systems. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (PODS'02) (pp. 1-16).

Capra, L., Emmerich, W., & Mascolo, C. (2003). CARISMA: Context - Aware Reflective Middleware System for Mobile Applications. *IEEE Transactions on Software Engineering*, *29*(10), 929-945.

Chan, A.T., & Chuang, S.-N. (2003). MobiPADS: A reflective middleware for context-aware mobile computing. *IEEE Transactions on Software Engineering*, *29*(10), 1072-1085.

Chen, H., Finin, T., & Joshi, A. (2003). An ontology for context-aware pervasive computing systems. *Knowledge Engineering Review*, *18*(*3*), 197-207.

Chlamtac, I., Conti, M., & Liu, J. J.-N. (2003). Mobile ad hoc networking: Imperatives and challenges. *Ad Hoc Networks*, *1*(1), 13-64.

Dey, A.K. (2001). Understanding and using context. *Personal and Ubiquitous Computing*, *5*(1), 4-7.

Fahy, P., & Clarke, S. (2004, June). CASS - Middleware for mobile context-aware applications. In *Proceedings of the 2nd ACM SIGMOBILE International Conference on Mobile Systems, Applications and Services* (MobiSys'04).

Gu, T., Pung, H.-K., & Zhang, D.-Q. (2005). A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, *28*, 1-18.

Haas, L.M., Kossmann, D., Wimmers, E.L., & Yang, J. (1997, August). Optimizing queries across diverse data sources. In *Proceedings of 23rd International Conference on Very Large Data Bases* (VLDB'97) (pp. 276-285).

Issarny, V., Sacchetti, D., Tartanoglou, F., Sailhan, F., Chibout, R., Levy, N., & Talamona, A. (2005). Developing ambient intelligence systems: A solution based on Web services. *Journal of Automated Software Engineering*, *12*(1), 101-137.

Keidl, M., & Kemper, A. (2004, March). A framework for context-aware adaptable Web services. In *Proceedings of 9th International Conference on Extending Database Technology* (EDBT '04) (pp. 826-829).

Maamar, Z., Mostefaoui, S., & Mahmoud, Q. (2005, January). Context for personalized Web services. In *Proceedings of 38th IEEE Hawaii*

International Conference on System Sciences (HICSS'05) (pp. 166.2).

Madhavan, J., Bernstein, P.A., Doan, A., & Halevy, A.Y. (2005, April). Corpus-based schema matching. In *Proceedings of the 21st International Conference on Data Engineering* (ICDE 2005) (pp. 57-68).

Ozsu, T., & Valduriez, P. (1991). *Principles of distributed database systems*. Prentice-Hall.

Roman, M., Hess, C.K., Cerqueira, R., Ranganathan, A., Campbell, R.H., & Nahrstedt, K. (2002). Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, *1*(4), 74-83.

Roth, M.T., & Schwarz, P.M. (1997, August). Don't scrap it, wrap it! A wrapper architecture for legacy data sources. In *Proceedings of 23rd International Conference on Very Large Data Bases* (VLDB'97) (pp. 266-275).

Zahreddine, W., & Mahmoud, Q.H. (2005, March). An agent-based approach to composite mobile Web services. In *Proceedings of 19th International Conference on Advanced Information Networking and Applications* (AINA 2005) (pp. 189-192).

Nikolaos Folinas received his Diploma in computer science from the Univ. of Ioannina in 2003. He immediately joined the MSc program of the Department of Computer Science in the University of Ioannina and received his MSc in 2006.

Panos Vassiliadis received his PhD from the National Technical University of Athens in 2000. He joined the Department of Computer Science of the University of Ioannina as a lecturer in 2002. Currently, Dr. Vassiliadis is also a member of the Distributed Management of Data (DMOD) Laboratory (http://www.dmod.cs.uoi.gr/). His research interests include data warehousing, web services and database design and modeling. Dr. Vassiliadis has published more than 25 papers in refereed journals and international conferences in the above areas.

Evaggelia Pitoura received her BSc from the University of Patras, Greece in 1990 and her MSc and PhD in computer science from Purdue University in 1993 and 1995, respectively. Since June 2005, she is an associate professor at the Department of Computer Science of the University of Ioannina, Greece where she leads the distributed data management group. Her publications include more than 70 articles in international journals and conferences and a book on mobile computing. She has also co-authored two tutorials on mobile computing for IEEE ICDE 2000 and 2003. She is recipient of the best paper award of IEEE ICDE 1999 and two "Recognition of Service Awards" from ACM.

Evangelos Papapetrou holds a diploma and a PhD degree in electrical & computer engineering from the Aristotle University of Thessaloniki, Greece. He is currently a lecturer in the Department of Computer Science at the University of Ioannina, Greece. His research interests include routing in networks with periodic and/or stochastically varying topologies, IP networking, MANETs, QoS in wireless mobile systems. He has served as a reviewer in several journals and conferences. He has been involved in national as well as in European projects. He is a member of IEEE, ACM and the Technical Chamber of Greece.

Apostolos Zarras received his BSc in computer science in 1994 from the Department of Computer Science, University of Crete. From the same department he received his MSc in distributed systems and computer architecture. In 1999 he received his PhD in distributed systems and software architecture from the University of Rennes I. From 2004 until now he holds a lecturer position at the Department of Computer Science of the University of Ioannina. Apostolos Zarras has published over 20 papers in international conferences, journals and magazines. He is currently a member of the IEEE computer society. His research interests include middleware, model-driven architecture development, quality analysis of software systems and pervasive computing.