DBGlobe: A Service-Oriented P2P System for Global Computing

Evaggelia Pitoura
Computer Science Department,
University of Ioannina,
Ioannina, Greece

Serge Abiteboul
INRIA-Futurs and
Xyleme,
France

Dieter Pfoser
CTI,
Athens, Greece

George Samaras
Computer Science Department,
University of Cyprus,
Nicosia, Cyprus

Michalis Vazirgiannis[1]
Department of Informatics,
Athens University of
Economics and Business,
Athens, Greece

ABSTRACT

The challenge of peer-to-peer computing goes beyond simple file sharing. In the DBGlobe project, we view the multitude of peers carrying data and services as a super-database. Our goal is to develop a data management system for modeling, indexing and querying data hosted by such massively distributed, autonomous and possibly mobile peers. We employ a service-oriented approach, in that data are encapsulated in services. Direct querying of data is also supported by an XML-based query language. In this paper, we present our research results along the following topics: (a) infrastructure support, including mobile peers and the creation of context-dependent communities, (b) metadata management for services and peers, including location-dependent data, (c) filters for efficiently routing path queries on hierarchical data, and (d) querying using the AXML language that incorporates service calls inside XML documents.

Keywords

Peer-to-peer computing, peer-to-peer databases, global computing, services, pervasive computing, ubiquitous computing, metadata, services

1. INTRODUCTION

Peer-to-peer (p2p) computing refers to a new form of distributed computing that involves a large number of autonomous computing nodes (the peers) that cooperate to share resources and services [2]. The peers normally correspond to computers located at the fringe of the network. P2p computing has attracted much current attention spurred by the popularity of file sharing systems such as Napster, Gnutella and Kazaa.

However, the p2p challenge goes beyond simple file sharing. In our research, we view the conglomeration of interconnected peers carrying data as a virtual super-database with a dynamic schema. In the context of the DBGlobe project [1], our goal is to manage this super-database. In particular, we develop data management techniques for modeling, indexing and querying data hosted by open-ended networks of massively distributed peers.

To resolve heterogeneity and semantic mismatch problems, we employ a *service-oriented* approach. In DBGlobe, the data of each peer are made publicly available through services, that is, access to data stored locally at each peer is achieved by invoking an appropriate service. Direct querying of the structure and content of data is also supported by defining services that employ an XML-based query language.

At a second level, metadata, i.e., data describing peers and their services, are maintained to capture their behavior and state. A uniform XML-based representation for services and metadata is used to facilitate information exchange and sharing. Thus, our focus is on semantic matching of hierarchical service descriptions and data.

The collection of data on devices that exist around a specific context (e.g., location or user) forms a data sharing community that we call an ad-hoc database or simply *community*. The configuration of such communities varies over time. Communities are designed to support context-specific behaviors such as location-aware queries.

DBGlobe is an ongoing project covering all levels of a data management system for data and services hosted by massively distributed autonomous and possibly mobile peers. In this paper, we present our results along the following topics:

- infrastructure support including mobile peers and the management of communities (Section 2),

[1] Other DBGlobe contributors: Peter Triantafillou (Univ. of Patras), Ioannis Fudos, Georgia Koloniari (Univ. of Ioannina), O. Benjelloun, Tova Milo (INRIA), Nektaria Tryfona, Vassilios Verykios (CTI), George Polyzos, Efstratios Valavanis, Christopher Ververidis (Athens Univ. of Business and Economics) and Chara Skouteli (Univ. of Cyprus).

- metadata for services and peers, including location-dependent data (Section 3),
- filters for efficiently routing path queries on XML data (Section 4), and
- querying that incorporates service calls inside XML documents (Section 5).

This paper is intended to serve as a general overview of our research. For technical details and comparison with related work, please refer to the related publications.

2. ARCHITECTURE

DBGlobe is a global data and service management system. It connects a number of autonomous, mobile devices and provides support for describing, indexing and querying their data and services (Figure 1).

2.1 Overview

DBGlobe connects a number of peers, which in the DBGlobe context, are called mobile devices or *PMOs* (Primary Mobile Objects) to emphasize the mobility aspect. We employ a *service-oriented* approach in that data are wrapped in services. PMOs are the primary providers of services. A PMO may be viewed as an object providing a number of services (methods) that may be activated from anywhere on the network. Besides functioning as service providers (servers), PMOs may act as service requestors (clients) or both.

PMOs connect to the DBGlobe system (and possibly directly to each other) in order to exchange data and perform computations. They register by providing appropriate metadata information. Their number and location may change over time, as new PMOs join or leave the system and existing PMOs relocate.

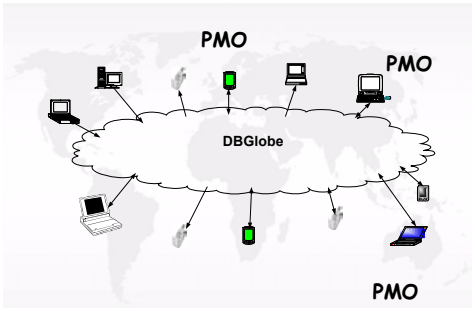

Fig 1: The DBGlobe Layer

DBGlobe components are grouped into infrastructure components and application-middleware components. Infrastructure components provide the basic functionality of the system, such as ubiquitous connectivity to devices, while middleware components support semantic and application related system services, such as the creation of communities. DBGlobe components are described in more detail later in this section.

Metadata information about PMO devices, users and their data is used to form communities, locate the appropriate services and build complex services. Metadata are described in detail in Section 3.

To discover the PMOs that offer an appropriate service, we use distributed indexes based on Bloom filters to route queries to matching PMOs. Indexes are described in Section 4.

Information exchanged between PMOs is expressed in the Active XML language [6]. An active XML document is an XML document that includes calls to services. AXML is presented in Section 5.

2.2 DBGlobe Infrastructure Components

Besides PMOs, the basic infrastructure components of DBGlobe are the *Cell Administration Servers* (CASs). Each CAS offers ubiquitous connectivity to devices, captures and stores contextual information and provides basic service publishing and semantic discovery features.

The overlay network of CASs is an essential layer of DBGlobe. The topology of CASs reflects the geographic layout of the system. In particular, we adopt a hybrid (partially ad-hoc) architecture where geographical 2-D space is divided into adjacent administrative areas (as in cellular communication systems) each managed by a Cell Administration Server. The overlay network of CASs constitutes the backbone that makes it possible for the PMOs to communicate and share data and services with each other. In our current design [3], each cell represents the coverage area of one WLAN access point. We assume that every PMO (including stationary devices) is associated with at most one cell at any given time (e.g., by keeping a live connection to the cell's defining network access point).

Each CAS can independently manage the PMOs which enter its area of authority. It keeps track of the PMOs that enter or leave the cell's boundaries. It stores metadata describing each PMO, the context and the resources offered and assists the user to locate services by semantically matching requests with existing service descriptions. It also provides basic services to visiting PMOs such as network addressing, session management and positioning. Each cell can support large numbers of PMOs moving inside its area and acting as sources or requestors of information.

2.2.1 PMO Components

A PMO is any autonomous, electronic device capable of communicating independently with the CAS via some communication channel. We assume that every PMO has built-in a globally unique identity (like Ethernet adapter addresses or IMEIs in GSM phones) and possibly incorporates components that can capture context (e.g., GPS receiver, digital compass, temperature sensor, etc.). In addition, it may host an application server (e.g, web server) for executing services. When a PMO is a device with

limited resources that cannot host a service but still contains resources to be shared, CASs may act as proxies. The corresponding CAS may host the service execution by retrieving the necessary data from the PMO and offering the service to requestors in lieu of the PMO that actually hosts the resources.

2.2.2 Cell Administration Server Components

CASs are interconnected through a network, e.g. the Internet. Although they can function autonomously, they are also aware of their neighbors that manage geographically adjacent cells). The CAS module consists of (Figure 2):

- A service ontology that has a hierarchical structure (starting from a universal concept).
- A service directory that lists all the services offered by PMOs in the cell.
- A service description repository of the local services.
- A CAS directory, containing addresses of other CASs.
- A device type and a PMO repository containing the list of device types and PMOs available in the cell and their profiles, and
- A temporal profile manager for storing the connection times of devices, discovering patterns and estimating probabilities of next appearance. A server can also keep historical data and computes statistics about their mobility habits to assist proactive behavior.

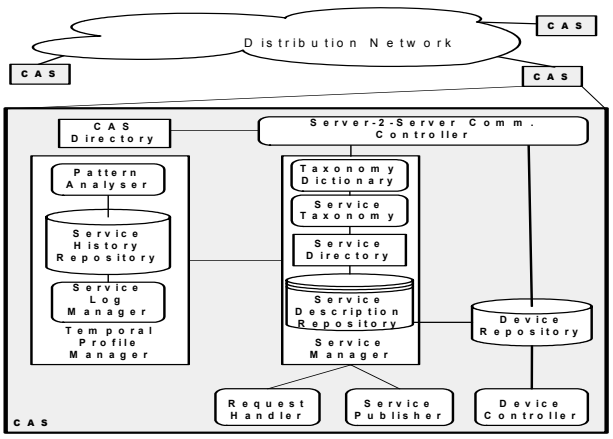

Fig 2: Cell Administration Server (CAS) components

2.3 Application Middleware

The basic responsibilities of the middleware are the management of communities and application support for disconnected operation. The major components of the middleware are the: (i) Communities Administrator Servers (CoAS) (ii) UserAgents, and (iii) dynamic query results database.

A *community* evolves around a semantic concept and may include any combination of spatial, temporal, or thematic characteristics that relate to that concept. The collection of PMOs with data or services related to the specific concept form the community. For example, all PMOs that provide weather services/data may form a "Weather" community. If a PMO needs to know whether it is raining in Athens, it will use the "Weather" community to find PMOs with appropriate services.

Each community is managed by a Community Administration Server (CoAS). A CoAS is the analogue of the CAS for communities. Similar to a CAS, a CoAS keeps track of the PMOs in its community and their related to the community services. It extracts relevant metadata information from the CAS servers and assists the user to locate services by semantically matching requests with existing service descriptions inside the community. Each CoAS keeps also track of other communities in the system.

Communities can be created proactively by users and administrators or in a dynamic manner by gathering (and grouping) metadata information on PMOs, services and users that have matching profiles and goals. CoASs are informed for changes on available services and metadata from the CASs. When a new service is available or a new PMO registers, the CASs propagates, for example, the appropriate metadata to all CoASs.

The *dynamic query result* database provides caching at the application layer. Intermediate results are stored in the form of views in this database. The goal of *UserAgents* module is to support network disconnections by cooperating with the dynamic query result database [7]. When a PMO gets disconnected, the UserAgent acts as its intermediary. During disconnection, the UserAgent remains active on the fixed infrastructure and works on the user's behalf (e.g. executing user requested queries). It stores query results at the dynamic query result database to be retrieved by the user upon reconnection. Currently, the dynamic query result database is used only for storing intermediate results during disconnections.

3. METADATA MANAGEMENT

Metadata stands for extracted structure and meaning of data. In the DBGlobe context, we encounter content metadata and profile data. The former describes the data contained in the PMOs, whereas the latter describes the user and the device itself [4].

3.1 Content Metadata

Content data are the actual data registered by the user on the PMO, which can be spatially-referenced and/or temporally-referenced information, indicating where and when the actual information was seen, or recorded. *Content metadata* describes these data. As these data are not

directly exposed but accessed through services, content metadata are in the form of ontologies relating to services.

Service discovery and *service creation* are essential tasks in a system such as DBGlobe. We introduce service ontology to support the structuring of the services and to aid service discovery. Further, the composition of services is a complex task and has to be supported by the system as well. Thus, parameter *ontology* is introduced to cover the parameters used in the various services.

3.1.1 Parameter Ontology

Services have signatures (e.g., described in WSDL) that provide the type of the arguments and results. Matching requests with service descriptions is a quite common problem in the literature. The most common approach is matching user requests expressed in keywords with service parameters (i.e., name, location, business, binding or tModel [12]). However, this kind of discovery mechanism does not include semantics. The parameter ontology aims at describing these types of all services in terms of a set of ontologies.

The example of a weather service illustrates our approach. Given a service that returns the weather for a given place and time,

(A) *Weather*: (location, time ➔ weather)

we want to extend this service to provide weather information along a route,

(A*) *Weather_en_route*: (route ➔ {location, time, weather}).

Assuming that a service *Route*(start, end ➔ route) exists that computes a route given a start and an end point, one can combine *Weather* to obtain *Weather_en_route*, if the parameters of the respective services match, e.g., *Route* returns a type route, which can be used as an input parameter for *Weather* if we know how to decompose a route into locations at times. These relationships can be exposed in parameter ontology.

3.1.2 Service Ontology

Once a service is defined, semantically it is more than the sum of its parts. Knowing the semantics of the parameters of the service is not sufficient to locate a service that fits user queries and reason about the semantics of the service (*service discovery*). Thus, what is needed besides an ontology relating the parameters is a means to locate or discover and relate services, a service tree (including cross links). The construct to realize such a structure can be in the form of ontology.

We propose a hierarchical structure in which semantically similar services are related to the same node in the tree, e.g., to the topic research. The tree forms a specialization relationship in that a child node contains more specialized services, e.g., a child of a "Research" node could be "Medical Research". Each node in the service tree has a set of attributes (keywords) that describes the domain of the referenced services. As these attributes are more abstract, they do not correspond directly to the types of the service interfaces (cf. parameter ontology). Searching is facilitated by a dictionary that contains lexicographically ordered keywords linked to the respective nodes in the service ontology.

3.2 Profile Data

Besides content data we have *profile data* characterizing the user and the PMO itself. Users have preferences with respect to what information they usually request, and considering mobility, as to when and to where they do this. Recording these data leads to the creation of a *user profile*. All data that characterizes the PMO will be stored in the *device profile*. We aim at capturing (i) the characteristics of the device itself, e.g., screen size and (ii) the characteristics of the device with respect to the DBGlobe system.

In connection with mobility and related applications, an important property of the device and the user (and thus part of the respective profiles) is their movement. It allows us to analyze spatial migration patterns, which leads to a better understanding of the PMO distribution in time and to the provision of better location-based services.

We provide a mobile ontology that is based on trajectories [8], interpolated samples of the position of the moving object. Given this representation, the ontology captures properties of the trajectory, e.g., speed, relationships to other trajectories, e.g., meet, and to its (spatial) environment, e.g., cross, can be derived the structural elements introduced by XML and ontologies to denote hierarchies and relationships.

4. FILTERS FOR HIERARCHICAL DATA

For a service request originating from a PMO, there may exist many sites (PMOs, CASs or CoASs) with matching services or documents. Thus, we need a mechanism to locate which sites contain relevant information efficiently. We adopt a decentralized approach. Each site maintains a data structure, called a *filter,* that summarizes all documents and services that exist locally. This is called a *local* filter. Besides its local filter, each site also maintains one or more *merged* filters that summarize the services and documents that exist in a set of its neighboring sites. These filters facilitate the routing of a query. When a query reaches a site, the site first checks its local filter and then uses the merged filters to direct the query only to those neighboring sites that may contain relevant documents. As our filters, we use Multi-level Bloom filters.

4.1 Multi-level Bloom Filters

Bloom filters are compact data structures for probabilistic representation of a set that supports membership queries.

Consider a set A = $\{a_1, a_2, \ldots, a_n\}$ of n elements. The idea is to allocate a vector v of m bits, initially all set to 0, and then choose k independent hash functions, $h_1, h_2, \ldots, h_k$, each with range 1 to m. For each element $a \in A$, the bits at positions $h_1(a), h_2(a), ..., h_k(a)$ in v are set to 1. A particular bit may be set to 1 many times. Given a query for b, we check the bits at positions $h_1(b), h_2(b), ..., h_k(b)$. If any of them is 0, then certainly b is not in the set A. Otherwise we conjecture that b is in the set although there is a certain probability that we are wrong. This is called a "false positive" and it is the payoff for Bloom filters' compactness. The parameter k and m should be chosen such that the probability of a false positive is acceptable.

Traditional Bloom filters cannot efficiently summarize hierarchically structured data (such as XML documents and XML-based service descriptions). We have proposed an extension of traditional Bloom filter, called multi-level Blooms appropriate for hierarchical documents [5]. We consider two types of multi-level Blooms based on the way XML documents are hashed: Breadth and Depth Bloom Filters.

Let an XML tree T with j levels, and let the level of the root be level 1. The *Breadth Bloom Filter* (BBF) for an XML tree T with j levels is a set of Bloom filters $\{BBF_0, BBF_1, BBF_2, \ldots, BBF_i\}$, $i \leq j$. There is one Bloom filter, denoted BBF_i, for each level i of the tree. In each BBF_i, we insert the elements of all nodes at level i. *Depth Bloom filters* provide an alternative way to summarize XML trees. We use different Bloom filters to hash paths of different lengths. The Depth Bloom Filter (DBF) for an XML tree T with j levels is a set of Bloom filters $\{DBF_0, DBF_1, DBF_2, \ldots, DBF_{i-1}\}$, $i \leq j$. There is one Bloom filter, denoted DBFi, for each path of the tree with length i (i.e., of i + 1 nodes), where we insert all paths of length i.

Our experimental results show that our multi-level Blooms outperform a same size traditional Bloom filter for processing path queries over hierarchical data

4.2 Semantic and Topological Distribution

Multi-level Bloom Filters are used for routing queries at the appropriate sites. Each site (PMO, CAS or CoAS) maintains a local Bloom filter summarizing the local services or documents. In addition, each site maintains a merged filter summarizing the documents for a set of its neighbors.

We consider two ways for determining the set of neighboring sites for which we maintain summaries. One approach is based on locality, and the other on filters similarity [11]. The locality based approach organizes the sites based on their proximity in the physical network (i.e., at the infrastructure level). The motivation behind this organization is to satisfy queries locally and minimize response time. The second approach organizes the sites based on their filter similarity, so as to group relevant sites together, thus supporting communities. The motivation for this organization is to minimize the number of irrelevant sites that process a query, by grouping together sites with relevant content. To achieve this we have defined a metric for deciding how "similar" two multi-level bloom filters are.

5. QUERYING

The information exchanged between PMOs is expressed in the Active XML (AXML in short) language. An AXML document is an XML document with embedded calls to services. The service calls are represented as particular XML elements. When calls included in an AXML document are fired, the document is enriched by the corresponding results. An AXML PMO contains AXML documents and offers (AXML) services, which can be used to enrich the AXML documents of the same or of other components. While documents with embedded calls have been used before, there is a fundamental difference between AXML and dynamic (XML) Web pages, since services may be invoked from anywhere on the network, calls embedded in an AXML page do not have to be evaluated before sending it.

In some sense, an AXML document can be seen as a (partially) materialized view integrating plain XML data and dynamic data obtained from service calls. As a simple example, consider an AXML document for the home-page of a local newspaper. It may contain some plain XML data, such as general information about the newspaper, and some dynamic fragments, e.g. one for the current temperature in the city, obtained from a weather forecast Web service, and a listing of current art exhibits, obtained from the local TimeOut guide service.

5.1 Controlling Service Call Activation

The service calls inside AXML documents are represented by special XML elements carrying the tag <sc> (for service call) which are interpreted as calls to services. An <sc> element encodes enough information to invoke the service on a fixed site. In case of a PMO that does not have a fixed URL, the <sc> element encodes an identification of the PMO that is needed to locate it on the network. Particular attributes of the <sc> element allow to specify when a service call should be activated (e.g. when needed, every hour, etc.), and for how long its result should be considered valid.

Continuous services. Simple services are similar to remote procedure calls. The service is called with some arguments, and eventually returns an answer. For the interesting class of continuous services, the interaction is more complex. Once a service call has been registered, a stream of answers is returned for this single service call. Streams of answers are encountered in many real-life applications: the stream of source updates for the maintenance of a data warehouse,

the readings of a temperature sensor or surveillance system, answers returned by continuous queries or publish-subscribe systems. In the case of such continuous services, the activation of the service call encapsulates a service subscription, and all the results received subsequently from the service are integrated in the caller document.

5.2 Intensional Parameters and Results

The parameters and result of a service call may themselves be AXML documents. A system component receiving a call with parameter containing service calls may have to activate the calls it includes before actually performing the service. Similarly, a service result may contain further service calls, which have to be activated by the site receiving the result. Thus, service call activations entail exchanging intensional data, and lead to a form of distributed computation.

The use of intensional parameters/results involves security issues. For instance, a malicious user may force a site to perform a dangerous action by calling one of the site's services with an intensional parameter that contains a call to a dangerous service. Besides security, the peer capabilities need also to be taken into consideration. We formalized the problem and developed algorithms to solve it [9].

5.3 Defining Services

The AXML framework allows to use arbitrary existing services as well as to define new services on top of the enriched AXML documents. The definition of AXML services relies on parameterized XML queries expressed in XQuery, the standard language for querying XML data, extended with updates. AXML services may query and update AXML or regular XML documents. The AXML service specification allows, in particular, the definition of continuous services, and of services with intensional input/output. One should note that an AXML service, when called by a non-AXML client, will detect the limitations of its caller and in such case will return only extensional answers.

5.4 Distribution and Replication

By the sole presence of PMO- and DBGlobe- services, AXML documents already include inherently some form of distributed computation. A higher level of distribution that also allows (fragments of) AXML documents and services to be distributed and/or replicated over several sites is highly desirable in a dynamic mobile architecture: a single XML document may need to be distributed on more than one PMO if the PMO does not have enough storage. Furthermore, replication of services, as well as the data they utilize, is needed, since a PMO will tend to use services "nearby". To address this, we extended the AXML data model with distribution and replication, and provided a location-aware extension to XQuery to handle distributed/replicated data and services [10].

6. CONCLUSIONS

DBGlobe is an ongoing project. Our future plans include among others appropriate notions of data consistency as well as providing a more systematic treatment of updates.

7. ACKNOWLEDGMENTS

This work was funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-32645 DBGlobe project.

8. REFERENCES

[1] The DBGlobe Project, http://softsys.cs.uoi.gr/dbglobe/

[2] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. "Peer-to-Peer Computing", HP Technical Report, HPL-2002-57

[3] E. Valavanis, C. Ververidis, M. Vazirgiannis, G.C. Polyzos and K. Norvag. "MobiShare: Sharing Context-Dependent Data and Services from Mobile Sources", 2003 IEEE/WIC International Conference on Web Intelligence (WI), 2003.

[4] D. Pfoser, E. Pitoura, N. Tryfona. "Metadata Modeling in a Global Computing Environment", 19th ACM International Symposium on Advances in Geographical Information Systems (ACM-GIS), 2002

[5] G. Koloniari and E. Pitoura, "Bloom-Filters for Hierarchical Data", Proceeding of the 5th Workshop on Distributed Data and Structures (WDAS), 2003

[6] The Active XML webpage, http://www-rocq.inria.fr/verso/Gemo/Projects/axml/

[7] C. Panayiotou and G. Samaras. "Personalized Portals for the Wireless User Based on Mobile Agents: Demonstration". IEEE 19th International Conference on Data Engineering, March 2003, Bangalore, India.

[8] D. Pfoser, C. S. Jensen. "Capturing the Uncertainty of Moving-Object Representations". SSD 1999

[9] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, F. D. Ngoc. "Exchanging Intensional XML Data". SIGMOD 2003.

[10] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, T. Milo**. "**Dynamic XML Documents with Distribution and Replication", SIGMOD 2003

[11] G. Koloniari and E. Pitoura. "Filters for XML-based Service Discovery in Pervasive Computing", submitted.

[12] Universal Description, Discovery, and Integration of Business for the Web,, http://www.uddi.org