UbiWare: Web-based Dynamic Data & Service Management Platform for AmI

Vasile-Marian Scuturici Sabina Surdu Yann Gripay Jean-Marc Petit
Université de Lyon, CNRS
INSA-Lyon, LIRIS, UMR5205
F-69621, Villeurbanne, France
{vasile-marian.scuturici,sabina.surdu,yann.gripay,jean-marc.petit@insa-lyon.fr}

ABSTRACT

The surrounding space is constantly augmented by a myriad of devices that expose heterogeneous data, like slower-changing data, dynamic data streams and functionalities. Handling data streams is a challenge by itself. Furthermore, developing applications that cope with heterogeneous data and diverse communication protocols is a tedious task. The success of such applications depends on the performance of data access and the easy management of available data. To address these challenges, we propose UbiWare, a middleware that facilitates application development for ambient intelligence. UbiWare is based on a distributed data access model. We abstract the surrounding space as a database-like environment and the heterogeneous entities and devices as data sources that produce data. To query the distributed data sources and access their data we introduce a declarative API that greatly simplifies application development and is compatible with the different operators used by query engines in Data Stream Management Systems or Pervasive Environment Management Systems. Queries written in declarative languages are submitted via an HTTP/REST-based protocol to data sources, where they are translated into the UbiWare's API commands.

Categories and Subject Descriptors

H.2.5 [**Database Management**]: Heterogeneous Databases

1. INTRODUCTION

The surrounding environment is constantly enriched with devices that provide the user with a digital representation of the world, yielding the so-called *pervasive environments*. In this context, computing systems evolve at a breakneck pace in order to fulfill the ambient intelligence vision (AmI), characterized by the heterogeneity, distribution and autonomy of devices, and by high data dynamics.

Applications developed for the AmI announce an easy to live in world for the user, who shouldn't even be aware of the presence of computing devices. The application developer, however, faces major challenges that significantly burden the development process. The myriad of environment-enhancing devices provides heterogeneous data, requiring different processing paradigms, like slower-changing data, similar to those found in classical databases, or potentially unending dynamic data streams. Some devices offer functionalities to the user (e.g., a camera that takes a panoramic picture of its surroundings). Furthermore, they present their data through various drivers, APIs and protocols. AmI applications need to understand and treat in a homogeneous way all these miscellaneous data and functionalities.

In a data-centric approach, the intensive interaction between the application and the environment data is performed via a data management layer, represented by a Data Stream Management System (DSMS) or by the more recently emerged Pervasive Environment Management Systems (PEMSs), like SoCQ [6]. The latter aim at easing application development over pervasive environments. The query engines of such systems use operators to translate SQL-like queries to relational algebra. Since these operators use data from the environment, access to the devices is a key factor in the performance of a DSMS or PEMS. Querying the diverse devices in a pervasive environment using a direct approach, without any middleware that can facilitate the communication between the query engine and the environment, is a tedious task. All the low-level details of the implementation have to be solved by developers.

We propose the Ubiquitous Data Middleware (UbiWare, for short), a middleware that aims at easing the interaction between query engines and miscellaneous devices in pervasive environments. UbiWare provides a homogeneous model for heterogeneous entities and data types, in a scalable framework, using a REST/HTTP-based protocol. Our goal is to provide a competitive model of organization of the environment data and mechanisms to efficiently access this data. The API we provide is compatible with the operators used by query engines.

2. THE UBIWARE PLATFORM

UbiWare is driven by the idea that the environment contains an ever growing number of miscellaneous entities that expose interesting, but heterogeneous data for the user. To tackle this problem, we abstract every entity as a data source, also called a *data service*, i.e., a producer as well as a supplier of data. We wrap each data service in a middleware nutshell that allows the external world (e.g., query engines) to query it via a declarative API. Moreover, data services can be connected in chains, in order to provide data flows.

We consider each data source to be a process equipped with computational capabilities, which provides various types of data, called *resources*. Consumers can access resources like static data, streams or functionalities. A data service is identified by an URI, it understands HTTP requests, in a REST manner, and delivers responses that respect the HTTP protocol. By using a HTTP/TCP protocol in a RESTful manner, UbiWare enables the integration of data services independently of the operating system and the used programming language [5]. HTTP/TCP is also the easiest way to bypass firewalls on the Internet or on different intranets.

The HTTP protocol presents one drawback: it was not designed to be used in a streaming context. Nevertheless, data services in the environment need to follow a push approach, where the consumer connects to the data service and waits indefinitely to receive the produced events. We achieve this behavior by using a permanently open HTTP connection between the consumer and the data source. The latter sends every newly produced event to the consumer and keeps the connection open [5].

The UbiWare model allows to filter interesting data directly at the source via projection and selection operators applied on data services. Data is accessed through the aforementioned streaming push technique, but also through pull mechanisms. The API we propose allows a consumer to obtain all the resources published by a data service, to get the structure (i.e., schema) of a resource, to perform a set of operations specific to each type of resource and to execute operations that manage resource composition. For managing data streams and resources, the API defines the PUT, GET, DELETE, UPDATE, INSERT and CALL operators. The names of the first 5 operators are self-explanatory, whereas the last one allows the developer to invoke a functionality on a given event from a stream or on a given resource and to produce a set of events over a specified schema.

UbiWare is built around a distributed architecture, where components have a dynamic behavior. The user is not concerned with handling distribution-specific aspects. The platform encompasses dynamic discovery functionalities. By using multicast announcement, data services reveal themselves when they become available or unavailable, and on a periodic basis. Consumers can also send discovery requests that receive immediate unicast responses [7]. Moreover, the discovery of data services producing data is facilitated by context awareness, i.e., through primitives using queries based on context values.

UbiWare achieves scalability by allowing clusters of data services to be processed by multiple management modules, called Data Service Managers. Such a module keeps information about discovered data services from the environment. It acts, in turn, as a data service that supplies information about the data sources in the environment. Consumers can directly interact with data services, without going through the Data Service Managers layer, but in this case developers also need to implement the data service discovery components.

With UbiWare as an abstraction layer between query engines and the environment, the developer is insulated from both heterogeneity aspects and specific data services implementation details. Application development under these circumstances is significantly simplified. Figure 1 describes the UbiWare architecture.

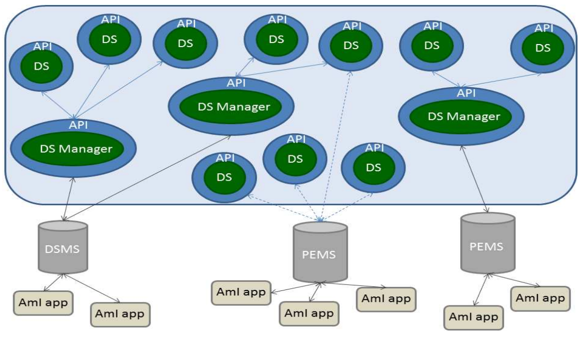

Figure 1: Data sources in the environment expose their resources via the UbiWare API and are managed by Data Service Managers. The latter interact with DSMSs/PEMSs, which process queries for AmI applications.

3. EXPERIMENTAL RESULTS

In the experimental study, we considered a pervasive environment comprising data services from the Linear Road benchmark [3]. We implemented a query engine on top of UbiWare. More precisely, we modified SQLite [2] in order to see the UbiWare data services as relational tables. We extended the standard SQL queries implemented in SQLite with continuous features. Moreover, every declarative query that accesses environment data and is submitted by the modified SQLite engine is translated into UbiWare API calls.

We ran a set of SQL continuous queries that describe various interactions between the query engine and the environment and we monitored system performance. Our experiments showed that the HTTP overhead incurred by using UbiWare depended on the representation of the transferred data. For binary data we used a base64 encoding, and the overhead in this case was around 33%. This overhead may be regarded as acceptable when considering the advantages derived from the middleware. Another possible drawback of UbiWare is the processing time at the query engine level in order to decode the HTTP strings to obtain the data. For data from the Linear Road dataset we observed a number of 140.000 decoded tuples/sec at the level of the query engine on a Windows XP machine with an Intel dual-core 3 Ghz and 3 GB RAM. This number is also acceptable, considering the time consumed by the query engine on various relational operators, like the join operations.

4. RELATED WORK

Our middleware is designed to handle heterogeneous data sources, providing not only static data, but also streams and functionalities. It can therefore be used as a data layer in DSMSs and PEMSs, smoothing communication between query engines and pervasive environments. We are not aware of existing solutions, that target both the streaming and the heterogeneity aspect. Using Web Services (that also use HTTP) is not appropriate in this context, as they don't allow efficient network streaming.

The LinkedData project ([1]) defines ways of sharing structured data on the Web. Data is represented using the RDF data model and each resource is identified by its URI. Access to data is realized using HTTP via GET requests. When compared to this approach, UbiWare stands out through its orientation towards streaming data.

The ExoEngine platform presented in [4] virtualizes components of streaming engines at different levels of granularity, to provide interoperability between SPEs and flexible deployment, but it doesn't tackle the heterogeneity encountered in pervasive environments.

5. REFERENCES

[1] Linked Data. `http://linkeddata.org/`.

[2] SQLite. `http://www.sqlite.org/`.

[3] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear Road: A Stream Data Management Benchmark. In *VLDB 2004*, pages 480–491, 2004.

[4] M. Duller, J. S. Rellermeyer, G. Alonso, and N. Tatbul. Virtualizing Stream Processing. In *Middleware 2011*, pages 269–288, 2011.

[5] Y. Gripay, F. Laforest, F. Lesueur, N. Lumineau, J.-M. Petit, V.-M. Scuturici, S. Sebahi, and S. Surdu. ColisTrack: Testbed for a Pervasive Environment Management System. In *'EDBT 2012*, pages 574–577, 2012 (demo).

[6] Y. Gripay, F. Laforest, and J.-M. Petit. A Simple (yet Powerful) Algebra for Pervasive Environments. In *'EDBT 2010*, pages 1–12, 2010.

[7] M. Scuturici. Dataspace API. `http://ds.liris.cnrs.fr/`.