A Service Architecture for Monitoring Physical Objects Using Mobile Phones

C. Frank, C. Roduner, P. Bolliger
Institute for Pervasive Computing
ETH Zurich
8092 Zurich, Switzerland
{chfrank, roduner, bolligph}@inf.ethz.ch

C. Noda
Communication Device
Development Department
NTT DoCoMo, Inc.
noda@nttdocomo.co.jp

W. Kellerer
Future Networking Lab
DoCoMo Communications
Laboratories Europe
kellerer@docomolab-euro.com

***Abstract*— Ubiquitous computing technology holds the promise of enabling comprehensive localization services for any physical object that is augmented with an electronic tag. Given an adequate object sensing infrastructure (e.g., based on RFID readers or sensor networks), the location of an arbitrary object could be obtained at the touch of a button. However, the large effort involved in installing a ubiquitous infrastructure of interconnected object sensing devices is one of the major obstacles for implementing such attractive services. In this paper we describe an object management system using mobile phones, so called ubiquitous gateways, as the infrastructure components mediating between object sensors and global network server infrastructure. As part of our system architecture and implementation, we describe query services which allow to set up mobile phones for local object tracking as well as to configure the behaviour of the global network to implement a number of use cases concerned with the monitoring of physical objects.**

I. Introduction

The attractiveness of a system for monitoring and locating personal objects has been frequently mentioned in literature. However, most existing systems which can be used for this purpose require an expensive infrastructure for *sensing* objects, for example, using Radio Frequency Identification (RFID) readers installed in the environment [1]–[3].

In this paper, we describe a prototype system for object monitoring and localization using mobile phones. Mobile phones combine two useful features: They are omnipresent in environments in which users live and are at the same time inter-connected by a homogeneous world-wide infrastructure. Given that important objects can be augmented with an electronic tag such that they can be detected when brought into the vicinity of a mobile phone, many new applications become possible revolving around managing, monitoring, or locating one's everyday items.

Various hardware could be employed for object tagging and sensing. For example, RFID tags are expected to be attached to various consumer products in the near future as they may realize significant cost savings in stock and supply chain management. In particular, passive RFID technology based on the Ultra-High Frequency (UHF) band [4], or similarly, *active* tags with a small autonomous power source [5], are expected to provide reading ranges of a couple of meters even with small reader modules. If improved variants of today's handheld UHF readers were integrated into mobile phones, a ubiquitous system could be deployed within few years using the short innovation cycle established through mobile phone sales. In addition to RFID, other upcoming radio communication systems, some even compatible with the phone's Bluetooth capability, could be used to identify objects in the phone's physical proximity in a similar way. If small inexpensive Bluetooth-discoverable tags [6] can be built (in fact, our prototype relies on BTnodes [7]) a ubiquitous object sensing infrastructure is already in place today.

Once object sensing technology can be integrated with mobile phones, various applications can be conceived concerned with the management and localization of tagged objects. Therefore, the focus of this paper is to devise a generic service architecture which allows for rapid implementation of end-user services concerned with the management of everyday objects. Further, we describe a demonstrator system which puts the architectural framework to work. The cornerstone of this architecture is the mobile phone which functions as a *ubiquitous gateway* mediating between local sensor information and the global service infrastructure.

We begin by describing the envisioned application use cases in Section II. After surveying related work in Section III, we describe the service architecture of our system in Section IV and, specifically, generic query services that allow configuring the interplay of system components for each of our use cases in Section V. We further describe the implementation of the devised service architecture and the associated demonstrator application in Section VI. We conclude with a discussion of future work in Section VII.

II. Application

In this paper, we will focus on the following four use cases revolving around the management of everyday items using mobile phones.

Remember Loss Context: The user can task his or her mobile device (ubiquitous gateway) to store the context in which an object left its range. This includes (among others) the following data: a trace of the user's location before and after the *loss* event, other people present, or other personal objects carried along when the object was lost. As there will be a large number of managed objects and users leave items behind on a regular basis (for example when leaving their home), we assume that it is unpleasant to issue a notification to the user each time a registered object goes out of range. Instead, the recorded data can provide the user with valuable hints regarding the location of a lost object as it helps recall the circumstances of the loss. Furthermore, the data can be used to perform automatic searches for the object (see *Find Object* use case).

Find Object: The user can send a query to many remote ubiquitous gateways (e.g., mobile phones of other users or ubiquitous gateways installed at lost and found offices) with the purpose of locating a given object that has been lost or misplaced. The query service should support queries that can be installed at remote ubiquitous gateways and trigger whenever the sought object comes in range of one of the object sensors.

Delegation: The user can delegate the care of a personal item from the personal ubiquitous gateway to other ubiquitous gateways [8], for instance to a hanger in a restaurant that can be tasked to guard a user's coat and send an alarm when it is removed without the user being present.

Lab Gate: A ubiquitous gateway device is installed next to a university or industry lab that uses an object sensor (such as an RFID reader) to record which objects leave with whom. It thereby provides an intuitive and non-intrusive check-in / check-out management for the equipment used in the lab.

We will use these four use cases to motivate the service architecture we describe in this paper. While we aim to provide generic support for the development of applications for the ubiquitous gateway, the above scenarios will be used as a test of the presented architecture's flexibility.

III. Related Work

Various work has argued for the relevance of reminding about personal objects or locating them. In a related prototype [2], RFID readers broadcast readings on carried tagged objects wirelessly to the passing user's personal device. The user is then notified on missing objects based on previously collected readings. Recently, a system called MAX [1] employed a wide-area infrastructure of base stations (e.g., in each room) and so called sub-stations (e.g., furniture) – for both of which a human readable name is assigned – to search for objects. Finally, the *delegation* use case was first mentioned in a different prototype [8]. Compared to these approaches, our system allows to set up a wide range of end-user services related to our application domain, which includes, but is not limited to, each of the prototypes described above. Moreover, compared to MAX [1], we do not rely on a pre-installed object-sensor infrastructure which is organized in a hierarchical fashion.

In addition, our system includes an implementation of supporting services centered around messaging, storage, and support for assembling applications from components. Using the presented framework, a user or application programmer can compose storage and messaging services into the communication pattern most suitable for the respective application scenario. For example, in the *lab gate* use case, the framework allows to set up sensors and storage components to proactively log all object-sensor readings in the user's database. Alternatively, in the *find object* use case, the framework allows to set up the system such that queries are disseminated reactively whenever an object is to be found. We will describe the query service in detail in Section V.

Note that the *find object* use case could also be implemented by proactively sending all sensor readings (e.g., object X seen by ubiquitous gateway A) to a centralized service which would then be used to locate a lost object. In this case our system would prominently face the challenge of a global data collection system, such as IrisNet [9] or Hourglass [10]. However, in our application we do not propagate sensor readings to the network core by default for various reasons. Firstly, the number of sensor readings (e.g., object X seen by ubiquitous gateway A) is by far larger than the number of queries (e.g., looking for a misplaced object) which makes querying for objects more efficient than propagating every sensor reading to a central server. Moreover, such aggregation of data in a centralized database would imply severe issues with protecting the personal privacy of system users. Last but not least, it is incompatible with a privacy enhancing feature of our system: Objects which have previously been associated with their owner will only be detected by a sensor *after* this sensor has received an explicit query for the given object (containing the owner's key, see Section IV for details).

Our current implementation of messaging services could be replaced by various frameworks that can provide message passing among gateways and to the server back-end [11]–[13]. However, most of these frameworks do not support the CLDC configuration of J2ME, which is also the reason why we could not make use of frameworks supporting component deployment such as [14]. From the frameworks that do support CLDC, note that Elvin's content-based messaging [11] is so far not beneficial to our application as in all our use cases the destinations of events are either known in advance (e.g., a certain user that is to be alerted) or provided by an application-specific service that determines where an object is to be searched. Analogously, we currently would not benefit from a peer-to-peer framework such as JXTA [13]. However, both [13] and [11] could provide a valuable extension for distributing information in our network at a later stage. Similarly, at a later stage an integration with the UMTS IP multimedia subsystem (IMS) [15] may prove valuable. So far, however, IMS's prominent features such as session management and enhanced handling of QoS requirements are not required by our application.

Finally, the authors of [16] describe a framework for using the mobile phone as a gateway to access heterogeneous health-related sensors pre-installed in the user's enviroment. Compared to [16], we focus on evaluating our architecture by applying it to heterogeneous use cases of our object management application. By using this approach, our architecture supports enhanced functionality, such as the wide-area distribution of queries and direct interaction between smart phones.

IV. Service Architecture

We will first give an overview of the core services of our architecture and the interplay of our system's components. Mobile phones acting as ubiquitous gateways are shown on the right side of Figure 1. They can sense tagged objects and their context either directly or through object sensors. Further,

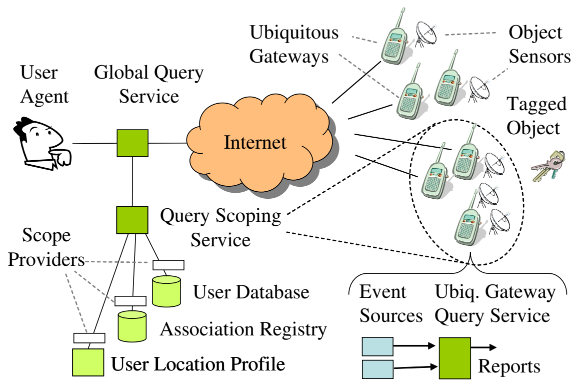


Fig. 1. Service architecture overview.

they are connected through a global packet-based network (e.g., a cellular packet network; shown as 'Internet') with a centralized server system. This server system hosts the *global query service* and the *query scoping service*, which manage queries for objects through the mobile phones. The user agent provides an interface for users triggering a query. This can be done from any user device including a user's mobile phone.

Query Services. The use cases of our application deal with context information provided by various types of sensors. Examples are the mentioned object sensors, the user's location, or various other context information that is required in the remember loss context use case such as a sensor identifying the presence of other persons. This variety of sensors motivates the abstraction of sensor information on the ubiquitous gateway through what we call *basic event sources*, which we will introduce in Section V-A.

Some use cases demand that, once a predefined condition is detected (e.g., when an object is sensed either in or out of range), a notification is generated. Such *reports*, notifications associated with additional context information, are issued by the *ubiquitous gateway query service* that specifies *which* conditions should be detected and *what* information to include in the report. It is described in detail in Section V-A.

A report is either sent to the local user carrying the ubiquitous gateway (i.e., the mobile phone), to a remote user carrying another mobile phone, or to an appropriate service (such as a storage service) in the infrastructure. The last two options require the *global query service* that provides both event routing and large-scale query dissemination. It is detailed in Section V-B.

Storage. As discussed, various data in the system need to be stored for later query. This involves mainly storing past events, i.e., keeping a log of which object was seen where and when. The different use cases require storing data both locally on the ubiquitous gateway (as is the case with *remember loss context*) or remotely in a user database provided by the back-end infrastructure if reliable storage is desired (as is the case with *lab gate*). The storage service is detailed in Section VI.

Association. Another core aspect of our application scenarios is *association*: The association service keeps track of associations between *users and objects* and is used when authorization decisions must be made. This is the case, for example, when a user asks the system to perform a wide-area search for a certain object whose whereabouts should only be visible to its owner. Second, *user to user* association enables more flexible handling of object access rights. Its main purpose is to maintain groups of users such as families or business departments in order to be able to grant them collective access rights to search an object. Third, *user to ubiquitous gateway* and *user to object sensor* association allows maintaining a set of favorite gateways and sensors that are particularly relevant to the user. A user may, for example, setup some ubiquitous gateways and object sensors in places that are of special interest to him or her. These could be dock gates in an industrial facility or a remote holiday home.

Location Profile. Further, users may choose to record statistics on their previous locations using a *location profile* service. This allows to send a *find object* query to locations where the user spends a large portion of his/her time, assuming that the objects are likely to be found there. For our purposes, we extended the concepts found in [17] by optional user interaction that allows the user to explicitly name locations.

Query Scoping. In the find object use case, the *query scoping service* assists the global query service by choosing an adequate subset of ubiquitous gateways which are likely to find the misplaced object based on application-specific heuristics. Such heuristics are implemented by means of *scope providers* – small adaptors that make use of data stored by other system services to specify which gateways should be queried. Specifically, our system includes three such scope providers. The first is based on the *user database* storing data on past object locations. This scope provider returns a list of gateways that are near locations where the object "was seen" in the past. A second scope provider based on the *association* service returns ubiquitous gateways that belong to users which are in some way associated with the object owner. Such a search strategy is useful when the object is used jointly by a group of colleagues or friends. Finally, a third scope provider returns ubiquitous gateways at relevant locations as determined by the *location profile* service. Note that a wide range of such search heuristics exists and we only list a small subset here. In a complementary paper on query scoping [18], we have described how a more comprehensive query scoping service could determine the likelihood that a ubiquitous gateway will find an object by integrating all kinds of history and profile data known to the system at the time of the query.

The global query service uses the ubiquitous gateway query interface we describe in Section V-A to install a query at the selected ubiquitous gateways, on which local storage services, basic event sources, and an event dispatcher, are used to implement the ubiquitous gateway query service as detailed in Section VI. Finally, at the network leaves, ubiquitous gateways may interface *object sensors* and, possibly, multi-hop sensor networks. For the latter, publish/subscribe approaches [19] could be employed to make the data available on the gateway in terms of a remote basic event source.

Privacy Considerations. For a system that handles a large variety of personal user data, protecting the users' privacy is of paramount importance. While not the focus of the paper, we made a few design choices to take privacy needs into account.

The first is that tagged objects can be hidden and remain visible only to their owner: Users may secure objects after purchase through the association service. A shared key is stored in the association registry, granting the owner exclusive rights to query for the associated object. Specifically, *secured objects* will not reply unless the respective object sensor polls with a packet containing the correct shared key [20]. A *find* query initiated by the owner will contain the key for accessing the object, as we describe in the next section. Further, users, whose presence may be stored as context in the *remember loss context* use case, can similarly choose to hide and only be visible to the devices of associated users.

Last but not least, the execution platform of services containing sensitive information can be re-configured within the application. That is, the association registry or the location profile, could be executed on the ubiquitous gateway device instead of the server back-end giving users full physical control of their data. Similarly, the location profile service which enhances object search functionality can simply not be deployed for some users. We will describe how our service implementation enables such transparent deployment of services in Section VI.

V. Query Services

The developed query services allow application programmers to specify the system's behaviour, in particular, to implement each of the use cases highlighted above. In this section, the interfaces of these query services will be described in detail.

The queries that an application can submit to our prototype system consist of two parts: First, a part for the *ubiquitous gateway query service* that specifies how an individual ubiquitous gateway shall use its object sensors when processing the query (e.g., if an object should be sensed using RFID, Bluetooth, or both). Second, a part for the *global query service* that specifies to which ubiquitous gateways a query shall be distributed, how it should be processed (e.g., time and cost constraints), and where the resulting events shall be delivered to.

A. Query Interface of the Ubiquitous Gateway

In this section we describe the query interface that each *gateway* exposes to the *global query service*. Before we can describe the interface, we need a concept for the context data that are available at the ubiquitous gateways (besides sensed objects) on which a query can operate. We call these basic sources of information *basic event sources*.

Basic Event Sources. Basic event sources provide a unified interface for all kinds of context data that are available on the ubiquitous gateway. They are custom-implemented components that are provided by the application developer. Each event source will generate events of a predetermined type and can be parameterized with an accuracy level to regulate the number of events it generates. Example event sources that are appropriate for our application include:

- A *Location* event source that provides information about a ubiquitous gateway's physical location. In an implementation that uses an integrated GPS receiver, this event source could generate an event as soon as the gateway's position changes by more than a programmer-specified amount. In a different implementation this source could generate an event whenever the GSM cell changes.
- *InRange* polls for tagged objects in periodic intervals specified by the programmer and, upon the detection of objects, generates an event that includes their identifiers. If secured objects should also be detected, the event source can incorporate access keys for these objects.
- Adversely, *OutOfRange* generates an event every time a specific, possibly secured, tagged object has been out of range for a given time. This event source is used as a trigger for remember loss queries.
- The *Persons* event source generates an event containing the identifiers of all persons that were detected in a gateway's vicinity. Again, the programmer can specify the frequency with which a scan for persons is performed.
- The *LabelReader* event source generates events whenever a context label is detected. Such context labels could be made available by places of interest (e.g., trains, buses etc.) that announce their symbolic name by broadcasting beacons.

It is clear that these examples of basic event sources are somewhat overlapping. However, as these components are custom-implemented to best suit a specific application, we believe that developers will come up with a wide range of differentiated basic event sources.

Query Formulation. Each query is specified by two parameters: The first parameter, is used to define when the query should produce a result. The programmer can specify this *trigger condition* by providing a (possibly custom-implemented) basic event source. The second parameter, defines the *report* that a query should generate, which is essentially a list of basic event sources. Typically, a report will have to include not just the data currently available from the specified basic event sources, but also observations prior and subsequent to the triggering event. In the report definition, we thus annotate each event source with an interval (a, b) in which the numbers a and b denote a range of events from a given source to be included in the report, relative to the time of the triggering event. Intervals can be time-based, e.g., $(-10s, 5s)$, where the report would include all events occurring within 10 seconds before and 5 seconds after the triggering event, or simply event based: $(-3, 3)$ would include 3 events that occurred prior to the triggering event and 3 events after the trigger. Event-based intervals are further annotated with an expiration timeout to allow for the generation of reports even if too few events arrive after the trigger. Note that the generated report is itself an event and thus the query itself is a new event source and can be treated on the same level as the basic event sources listed above.

Using such query definitions consisting of a *trigger condition* and a corresponding *report definition*, we formulate the described use cases as they would be issued at the ubiquitous gateway interface as shown in Figure 2.

The *remember loss context* query specification is defined to use the *OutOfRange* source (initialized for the specific object) as a trigger and, as soon as the object goes out of range, to

Query Spec.	**Remember Loss Context**	**Find Object**	**Lab Gate**
Trigger	OutOfRange(obj)	InRange(obj)	InRange(all)
Report	Location $(-120s, 60s)$ LabelReader $(-2, 0)$ Persons $(-3, 3)$	Location $(-5, 0)$	InRange $(-5, 5)$ Persons $(-5, 5)$

Fig. 2. Use cases formulated at the ubiquitous gateway query interface.

report the user's location observed during three minutes around the trigger event, the user's friends that were present at that time (at most 6 events from the persons source that occurred before or after the trigger), and recently read labels (the last two). Similar examples are given for the *find object* and *lab gate* cases (we omitted the analogous *delegate* use case).

The destination of the report and limits on the duration and the cost of the query are set via the global query service interface described in the next section.

B. Global Query Service

In order to facilitate the deployment of applications building upon a large-scale infrastructure as described above, our framework provides a *global query service*. It allows developers to easily formulate queries and pass them to the infrastructure that will then transparently distribute them both among the various logical components (middleware services) and physical devices (ubiquitous gateways). When the global query service receives a query, it uses a *scope provider* in order to relay it to the ubiquitous gateways that are most appropriate depending on the client application's needs. It also keeps track of active queries and removes them when the searched-for tagged object has been found.

Apart from shielding application developers from the complexity of the underlying service infrastructure, the global query service serves the purpose of isolating the party initiating a query from the party receiving it. A mobile network operator could for example act as a trusted intermediary between end-users who offer their terminals as sensing devices on the one hand and others who consume these services on the other hand. This allows for the effective enforcement of privacy and accounting policies. The interface also provides client applications with methods to monitor and control the costs incurred by their querying activities. It has to be noted here that the client application for invoking queries can reside on a user's mobile phone, which is acting as a ubiquitous gateway at the same time.

The global query service provides a simple method for client applications to bootstrap queries. This method returns a reference to a *query* object that clients use for subsequent communication with the global query service. The interfaces of the global query service and the queries created through it are shown in listings 1 and 2.

```
interface GlobalQueryService {
 Query createQuery(
   in QuerySpecification querySpec,
   in ScopeProvider scopeProvider,
   in long maxCost,
   in long maxEvents,
   in long maxTime ) };
```

Listing 1. Interface of global query service.

The semantics of the arguments passed to *createQuery* are defined as follows:

- The *querySpec* parameter specifies the details of the query that is to be executed at the ubiquitous gateways, including instructions on how the different sensors must be used (see Figure 2 for details).
- The *scopeProvider* argument is used by client applications to indicate the search heuristics that are best suited for the task at hand. An application can either point to an existing, predefined search scope provider or implement its custom one.
- Using the *maxCost* parameter, an application can limit the costs it is willing to incur during the execution of the query. As soon as a query has accumulated this cost, it is automatically terminated by the global query service. The dimension of costs is not defined. Depending on the setting the system is deployed to, this value could for example represent the charges a mobile network operator imposes on the use of its network or the energy needed to transmit a query. The *maxCost* parameter is optional. If it is not specified, the system will not constrain the query based on cost considerations.
- By the *maxEvents* argument, the client application defines how many events a query can generate at most. As soon as it has triggered this many events, it is automatically removed by the global query service. This parameter is optional. If it is not specified, the query will not be terminated based on the number of events it generates.
- Finally, *maxTime* can be used to limit the lifetime (in seconds) of the query. If the query should not be removed after a certain time, this argument can be omitted.

```
interface Query {
 addSink(in Sink sink);
 removeSink(in Sink sink);
 cancel(); };
```

Listing 2. Interface of *query* objects.

The events that are generated by a query are routed to *sinks* by the wide area infrastructure. A client application can add sinks to the query by calling the *addSink()* method.

C. Use Cases

In this section we will illustrate how the global query service abstraction can be used to formulate the four usage scenarios of our application.

Remember Loss Context. An application can implement the *remember loss context* use case as follows:

```
query = GlobalQueryService.createQuery(
  new QuerySpec(QTYPE_REMEMBER_LOSS_CONTEXT, ...),
  new SingleGatewaySP(gatewayInstance),
  null, /* maxCost */
  null, /* maxEvents */
  null /* maxTime */
);
query.addSink(localStorage);
```

In this example, the client application asks the global query service to use the *single gateway scope provider* to direct the query specification to the ubiquitous gateway referenced by *gatewayInstance*. In this case, *gatewayInstance* corresponds with the local ubiquitous gateway (i.e., the mobile phone currently executing the application). The indicated ubiquitous gateway then builds a query based on the primitives outlined in Section V-A and executes it locally.

In the *remember loss context* use case we assume that the user's mobile device should permanently monitor the presence of a tagged object. Therefore the limiting parameters *maxCost*, *maxEvents* and *maxTime* are omitted. As a second step, the application attaches the storage service running locally on the user's mobile device as an event sink where object loss events will be recorded.

Find Object. An application can request the global query service to locate a lost tagged object using the following code sequence:

```
query = GlobalQueryService.createQuery(
  new QuerySpec(QTYPE_FIND_OBJECT, ...),
  new LocationProfileSP(userId),
  10, /* maxCost */
  5, /* maxEvents */
  600 /* maxTime */
);
query.addSink(this);
```

In the *find object* use case, the client is unable to indicate the ubiquitous gateways the query specification should be distributed to. It therefore creates a *location profile scope provider*, which is then used by the global query service in order to obtain a number of gateways that are most likely to yield a search result. The list of promising gateways is calculated based on the user's historical locations as recorded in the location profile (as described in Section IV). It can be expanded gradually when a query does not yield a result after some time. Alternatively, the search algorithm we described in complementary work [18] could be used as a scope provider.

In this example, the application requests the global query service to terminate the query as soon as the accrued costs reach or exceed 10 units, 5 object events have been delivered, or the query has been running for 600 seconds. Finally, the client application makes sure that it receives the object events generated by the query so that it can inform the user when the object is found.

Lab Gate. The *lab gate* use case is similar to the *remember loss context* use case in that it is installed at a single, user-specified ubiquitous gateway. The following instructions are used:

```
query = GlobalQueryService.createQuery(
  new QuerySpec(QTYPE_LAB_GATE, ...),
  new SingleGatewaySP(labDoorGateway),
  null, /* maxCost */
  null, /* maxEvents */
  null /* maxTime */
);
query.addSink(myDatabase);
query.addSink(GLOBAL_MARKET);
```

In this use case, the global query service will send the query specification to a specific ubiquitous gateway installed at the lab's door. The query specification used in this example causes the lab gate to repeatedly look for tagged objects in its environment and return them organized in groups of objects that were observed at the same time. The query should not be limited by costs, number of resulting events, or time.

The application registers two event sinks in this example. First, the user's personal database that records the location history of his or her tagged objects. Second, a general event channel that we call *global market*. Any user participating in our system can subscribe to the global market in order to retrieve all events related to his or her tagged objects that are seen on the global market. Other users can decide to make their infrastructure available to whoever is interested by pushing events to the global market. The idea behind the global market is that owners of ubiquitous gateways might want to share with the community those events that are of no value for themselves and that are merely by-products.

VI. Prototype Implementation

The query interfaces described in the previous section can be used to setup each of our application use cases both at the global query service and the involved ubiquitous gateways. In this section, we give the details of our prototype implementation of the described service framework and associated query services. We describe the implementation of our system [21] from the bottom up.

A. Tagged Objects, Object Sensors, and Event Sources

In our prototype, we use BTnodes [7], tiny devices equipped with a processor, a battery unit and a Bluetooth radio, as *tagged objects*. Further, Nokia's Series 60 phones function both as *ubiquitous gateways* and as *object sensors*. We currently use Bluetooth discovery to implement object and person sensing, that is, the *InRange* and *OutOfRange* and *Person* event sources. We prototypically implement *secure* objects by optionally disabling Bluetooth discovery on BTnodes after association – from which point on, objects can only be detected when a connection attempt is made with the correct Bluetooth MAC address. When users specify that their presence should only be detected by associated users, their MAC addresses are similarly exchanged in the association process. The *Location* event source is implemented by reading the current cell id of the mobile phone. Additionally, the user can attach symbolic names (such as 'office') to each location, using our implementation of the location profile [17]. An additional event source may read data from a Bluetooth GPS device.

B. Component Management

Some of the services we described in Section IV require optional installation based on user preferences and/or device capabilities. For example, some users might prefer to turn off the location profile service due to privacy implications. Further, heterogeneous ubiquitous gateway devices might require different configurations of *basic event source* components to be deployed, based on the physical sensors they have available.

In order to provide a flexible means for application deployment to the back-end server and to heterogeneous ubiquitous gateways, we implemented every service in our system as an independent, replaceable and re-loadable component according to the OSGi specification [22]. This way, we can reduce the number of configurations that need to be developed and maintained for heterogeneous user devices. In addition, this allows to load only the required components to ubiquitous gateways and the server.

As supporting OSGi containers, we run an existing framework [23] on the server, and further provide our own lightweight implementation of the core parts of the OSGi framework on the J2ME platform. On the server, a pool of worker threads is provided that allow to handle many client requests concurrently.

C. Communication Channels

We can distinguish two different types of communication for inter- and intra-node communication, respectively.

Messaging. A simple point-to-point messaging system is used to realize communication between ubiquitous gateways and between a ubiquitous gateway and a centralized service on the server. For the former, we provide an SMS/MMS-based channel together with a second, Bluetooth-based channel, if the two devices are within ad hoc communication range. TCP sockets are used to communicate with the server.

Event Routing on the Ubiquitous Gateway. In order to distribute various sensor data to different services running on the ubiquitous gateway, we implemented an event router component, which enables an event source to send events to any number of event sinks, this way implementing the one-to-many communication pattern. In order to receive a specific type of event, a sink may indicate the desired type when subscribing. As event sources are threads which can be started and stopped any time, the router will make sure that sources are only running when at least one sink is subscribed to it. This helps save resources which are limited on the mobile phone.

D. Serialization and Storage

As discussed, we need a mechanism to persistently store data, for instance association data on users and objects or a potentially great number of events generated by queries. Depending on the global query service setup described in Section V-B, data can be stored using the Record Management System (RMS) of the local gateway or, more reliably, in a persistent data store in the back-end database.

For remote storage, we serialize (byte-encode) data records and send them to the server storage component, which updates the corresponding records in its database. As none of the software we reviewed supports the CLDC configuration of J2ME [24], we decided to implement a small serialization framework on our own. Conceptually similar to the Serialization API of J2SE [25], we offer a common serialization interface that is implemented by every class representing data to be stored. Instances of such classes can hence be serialized into a byte stream, the database, or RMS using the corresponding serializer. In order to save bandwidth while communicating with the gateways, we decided to serialize the objects into a compressed XML format.

By combining these storage services with the messaging system described above, we are able to offer a remote storage service. Consequently, storing gateway data in the server database becomes as easy as storing it in the local RMS.

E. Query Processing

As discussed in Section V, there are two components involved in processing a query. Each ubiquitous gateway runs a query processor that handles queries to an individual gateway. Further, the global query service is used to create a query and to determine the ubiquitous gateway the query must be sent to.

Query Processor. This component is used to validate, install, and remove queries on the ubiquitous gateway device and to execute multiple queries in parallel. Additionally, it is in charge of monitoring the queries' lifetime and thus stops and removes queries when necessary.

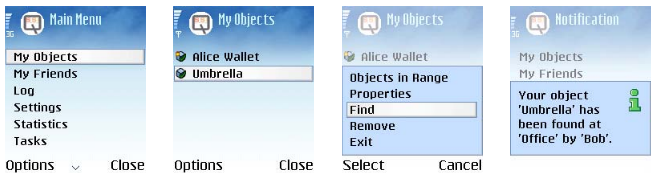
In a first step, the query processor will validate the query. This check includes the query's lifetime, user access rights and the availability of the required event sources. Consequently, the query will not be started should any of these checks fail.

Once a query is started by the query processor, it will subscribe to the basic event sources it requires at the local event router. Further the query stores a history of events relevant to this query using the local storage service. For example, given the *find object* query specified in Section V-A, the latest five events from the GPS receiver will be stored. Once the specified trigger event source, in this example the *InRange* source, has fired, the query will un-subscribe the trigger and generate a report as soon as all required events – that is, contents of the specified report definition – have arrived (a timeout is used to protect the system from waiting forever). In the *find object* example, no sources need to be monitored *after* the trigger, therefore the query immediately generates a report. Finally, the query will re-subscribe to its trigger event source to return to its original state.

Global Query Service. If the target gateway is known, as for example in the *delegation* use case, the global query service forwards the query to the specified target gateway.

If the target gateway is not explicitly stated, the global query service will invoke the given scope provider and query ubiquitous gateways in the order they are returned by the scope provider. In our current implementation, we use a scope provider based on association that returns a list of associated gateways (for example, office colleagues).

F. User Interface



Your object 'Umbrella' has been found at 'Office' by 'Bob'.

(a) Menu (b) Objects (c) Find (d) Report

Fig. 3. Screenshots of the prototype running on a Nokia 6630 phone.

To assess the feasibility of our query services in everyday use, we implemented a demonstrator application for J2ME mobile phones that would use the services described above. Additionally, this application allows to manage the user's object and friend associations as described in Section IV.

To create an initial association with a *tagged object*, the user selects an object currently in his or her vicinity from the list of *objects in range*. A similar procedure is used to create a *friend* association (not shown). If desired, this information can be propagated to the server back-end using the remote storage service described above. The user can edit such stored associations using the interface as presented in Figure 3(b).

Context menus attached to items in the list of objects allow to initiate a query to the infrastructure. For example, to find an associated object that has been lost, as described in the *find object* use case, the user would select the corresponding command as shown in Figure 3(c). This command causes a query to be sent to the global query service at the server back-end. As discussed, the global query service uses the association scope provider to distribute the query to office colleagues. When any of the ubiquitous gateways to which the query has been sent discovers the lost object, it will send a report message to the originating gateway. The user then receives a notification as shown in Figure 3(d).

VII. Conclusion

In this paper, we presented the design and implementation of a comprehensive service architecture that supports the development of various novel applications which draw on the ubiquitous availability of mobile phones. Core concept is the use of mobile phones as ubiquitous gateways that connect a ubiquitous sensing infrastructure with back-end services that provide global storage and query dissemination facilities.

The developed query service exports a pair of complementary interfaces which can be used to specify local and global behaviour, respectively. This two-layer abstraction allows for easy setup of the components involved in such large scale applications. By means of a demonstrator system for managing personal objects which includes various use cases ranging from finding misplaced personal objects to automated check in and check out of shared hardware in a university lab, we aimed to demonstrate the applicability and flexibility of the interface and of the underlying service architecture.

While our implementation of query processing services constitutes a pragmatic approach related to existing work [26], [27], its abstraction level which lies close to the application's logic allows for a lightweight and efficient implementation of the query service while at the same time remains generic enough to be applicable to a wide range of applications. Moreover, the global query service provides coordination functionality for wide-area queries (such as cost control and query dissemination), which is a specific challenge in our setting. Finally, our interfaces allow fine-grained tuning of the incurred communication overhead by means of an intermittent layer of storage services on the ubiquitous gateway device that can cache generated reports for later query.

Current work includes an experimental evaluation on the feasibility of searching objects using user-carried object sensor devices. Further, we will use our prototype to collect actual user data such as user associations and position traces which will allow to evaluate query scoping based on real user behaviour.

References

[1] K. K. Yap, V. Srinivasan, and M. Motani, "MAX: Human-centric search of the physical world," in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SENSYS'05)*, San Diego, CA, USA, November 2005.

[2] G. Borriello, W. Brunette, M. Hall, C. Hartung, and C. Tangney, "Reminding about tagged objects using passive RFIDs," in *Proceedings of the 6th International Conference on Ubiquitous Computing (UbiComp'04)*, Nottingham, England, September 2004.

[3] R. Want, K. Fishkin, A. Gujar, and B. Harrison, "Bridging Physical and Virtual Worlds with Electronic Tags," in *ACM Conference on Human Factors in Computing Systems (CHI'99)*, Pittsburgh, USA, May 1999.

[4] B. Bacheldor, "RFID vendors unite to promote UHF for items," *RFID Journal*, June 2006.

[5] RF Code, Inc., "Mantis^{TM} active RFID tags 433 MHz data sheet," www.rfcode.com/data_sheets/433_mantis_tags.pdf, 2006.

[6] Wibree Technology, www.wibree.com, October 2006.

[7] J. Beutel, O. Kasten, F. Mattern, K. Römer, L. Thiele, and F. Siegemund, "Prototyping Sensor Network Applications with BTnodes," in *Proceedings of the 1st European Workshop on Wireless Sensor Networks (EWSN'04)*, Berlin, Germany, January 2004.

[8] H. Shimizu, O. Hanzawa, K. Kanehana, H. Saito, N. Thepvilojanapong, K. Sezaki, and Y. Tobe, "Association management between everyday objects and personal devices for passengers in urban areas," Pervasive 2005, Demonstration, Munich, Germany, May 2005.

[9] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, "IrisNet: An architecture for a worldwide sensor web," *IEEE Pervasive Computing*, vol. 2, no. 4, 2003.

[10] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, Atlanta, GA, USA, April 2006.

[11] B. Segall and D. Arnold, "Elvin has left the building: A publish/subscribe notification service with quenching," in *Proceedings of the Australian UNIX and Open Systems User Group Conference (AUUG'97)*, Brisbane, Australia, September 1997.

[12] M. Caporuscio, A. Carzaniga, and A. L. Wolf, "Design and evaluation of a support service for mobile, wireless publish/subscribe applications," Department of Computer Science, University of Colorado, Tech. Rep. CU-CS-944-03, Jan 2003.

[13] L. Gong, "Project JXTA: A technology overview," 2002.

[14] A. Frei and G. Alonso, "A dynamic lightweight platform for ad-hoc infrastructures," in *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Kauai Island, Hawaii, USA, March 2005.

[15] J. F. Huber, "Mobile next-generation networks," *IEEE Multimedia*, vol. 11, no. 1, pp. 72–83, January 2004.

[16] D. Trossen and D. Pavel, "Building a ubiquitous platform for remote sensing using smartphones," in *Proceedings of the 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services (MobiQuitous'05)*, July 2005, pp. 485–489.

[17] K. Laasonen, M. Raento, and H. Toivonen., "Adaptive on-device location recognition," in *Proceedings of the 2nd International Conference on Pervasive Computing (Pervasive'04)*, Vienna, Austria, April 2004.

[18] C. Frank, C. Roduner, C. Noda, and W. Kellerer, "Query scoping for the Sensor Internet," in *Proceedings of the IEEE International Conference on Pervasive Services (ICPS'04)*, Lyon, France, June 2006.

[19] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, "Building efficient wireless sensor networks with low-level naming," *Operating Systems Review*, vol. 35, no. 5, pp. 146–159, 2003.

[20] S. J. Engberg, M. B. Harning, and C. D. Jensen, "Zero-knowledge device authentication: Privacy & security enhanced RFID preserving business value and consumer convenience," in *Proceedings of the 2nd Annual Conference on Privacy, Security and Trust (PST'04)*, October 2004.

[21] P. Bolliger, "Query services for the sensor internet," Master's thesis, ETH Zurich, January 2006.

[22] P. Kriens et al., "OSGi Service Platform Specification, Release 3," The Open Services Gateway Initiative, Tech. Rep., 2003.

[23] Knopflerfish OSGi, "http://www.knopflerfish.org," April 2006.

[24] Sun Microsystems Inc., "Connected limited device configuration specification 1.1," March 2003.

[25] "Java object serialization specification," Sun Microsystems Inc., 8 2001.

[26] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The design of an acquisitional query processor for sensor networks," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, San Diego, CA, USA, 2003, pp. 491–502.

[27] S. Li, S. H. Son, and J. A. Stankovic, "Event Detection Services Using Data Service Middleware in Distributed Sensor Networks," in *Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks (IPSN'03)*, Palo Alto, CA, USA, April 2003.