# The Sensor Internet at Work: Locating Everyday Items Using Mobile Phones

Christian Frank, Philipp Bolliger, Friedemann Mattern
Institute for Pervasive Computing
ETH Zurich, Switzerland

Wolfgang Kellerer
DoCoMo Communications Laboratories Europe
Munich, Germany

**Abstract**

We present a system for monitoring and locating everyday items using mobile phones. The system is based on phones which are enhanced with the capability to detect electronically tagged objects in their vicinity. It supports various functionalities: On the one hand, phones can store the context in which users leave registered items and thus help to locate them later on. On the other hand, object owners can search for their objects using the infrastructure of mobile phones carried by other users. We describe the design of our object location system and provide an algorithm which can be used to search for lost or misplaced items efficiently by selecting the most suitable sensors based on arbitrary domain knowledge. Furthermore, we demonstrate the practicability of such wide-area searching by means of user-held sensors in a series of simulations complemented by a real-world experiment.

## 1 Introduction

Inexpensive sensing devices are expected to play a major role in future computing systems that aim to make the daily life of their users easier by monitoring everyday physical processes and providing novel features based on the acquired data.

What currently hinders most of the conceived systems from becoming commercial applications, however, is a lack of adequate infrastructure of various types: First, a *sensing* infrastructure must be installed to perform the sensing task. Second, a *communication* infrastructure is required to distribute and aggregate sensor readings from multiple sensors. Finally, a *commercial* infrastructure is needed to manufacture and deploy sensing devices, and to generate revenue from the system.

The mobile phone system provides a unique opportunity to overcome these difficulties. *Sensing* technologies can be embedded into mobile handset devices or accessed from the handset via short-range wireless communication. Wide-area *communication* is a core property of the cellular network. It enables the integration of data from many

sensors and the support of applications with backend services such as data storage and dissemination. Finally, cellular network operators control an important *commercial* infrastructure as they can promote the large-scale production of sensor-enhanced devices and deploy such devices to a large subscriber base through their established sales channels (as we have already seen with camera phones).

With new devices and an adequate communication infrastructure connecting sensors to form a global *Sensor Internet*, the commercial deployment of powerful applications becomes a real possibility. Mobile phones can provide a unique sensing infrastructure for such applications: A single mobile phone, enhanced with appropriate sensors, can already provide almost full "coverage" of its owner's activities, the routes he follows, and the places he visits. Moreover, given the prevalence of mobile phones, multiple phones can achieve virtually ubiquitous geographic coverage: at any given moment, data could be retrieved from any place where there are phone users.

Making use of these unique properties of mobile phones and the cellular network, we present a system which is concerned with monitoring and locating everyday items by means of mobile phones. Mobile phones, with integral hardware that allows them to detect the presence of electronically tagged objects in their vicinity, fulfill the dual role of *object sensors* and of a *user interface* for managing personal items at the same time. Furthermore, data from many user-held object sensors can be aggregated to locate an arbitrary missing item.

This system allowed us to identify two particular challenges that are common to many applications which make use of the large people-centric infrastructure provided by mobile phones and the cellular network. The first is to define the *scope* of a sensing task: due to the large scale of the network, it would be inefficient to query all available object sensors or, alternatively, to store all sensor readings in a centralized database for any subsequent queries. Rather, it is sensible to involve only a small subset of all available sensors which are likely to cover the phenomenon one is interested in. To address this issue, we present in Section 3 an algorithm that incorporates arbitrary application-specific knowledge to select a subset of sensors for a given object search query.

The second challenge is to answer the question of whether a system based on sensors carried by people can provide sufficient sensor *coverage* in a relatively short period of time. In an extensive evaluation, which includes a real-world experiment with our object localization prototype, we therefore analyzed the properties of the coverage obtained given a wide range of different operational parameters such as the density of participants, their mobility, the range of the sensors used, and the time intervals within which a sensing task is performed. In addition to confirming the feasibility of object localization based on mobile phones, the study can provide valuable guidelines for the design of future people-centric sensing systems in general.

## 1.1 Object Sensors

Our application requires the integration into mobile phones of *object sensors*, which are able to detect the nearby presence of an electronically tagged item. Various technologies could be employed for this purpose. For example, passive RFID tags are expected to be attached to many consumer products in the near future, as they may realize signif-

icant cost savings in stock and supply chain management. In particular, passive UHF RFID technology [25] or active tags with a small autonomous power source [21] can provide reading ranges of a couple of meters even with small reader modules. If improved variants of today's handheld RFID readers were integrated into mobile phones, a ubiquitous system could be deployed within a few years using the short innovation cycle established through mobile phone sales. In addition to RFID, other upcoming radio communication technologies such as Zigbee, some even compatible with the Bluetooth capability of today's phones, could be used to identify objects in the phone's proximity in a similar way. If small, inexpensive Bluetooth-discoverable tags can be built (e.g., based on Wibree [29] or "Ultra Low Power (ULP) Bluetooth" [27]), this would mean that a ubiquitous object-sensing infrastructure is already in place today.

For each tagging technology, there is a certain trade-off between tag costs and size, the identification range achievable, and the costs of reader hardware. Irrespective of the technology used, we assume for our scenario that suitable object sensors can be integrated into mobile phones, as is already the case today with Bluetooth and NFC. Our current system prototype requires battery-powered tags (BTnodes [3]) on objects and uses the phones' built-in Bluetooth discovery for object sensing. While we also evaluate the benefit of an increased sensing range of possible future technology in this paper, the final choice of technology is based on costs versus range trade-offs, which remain to be explored in a concrete product's business plan.

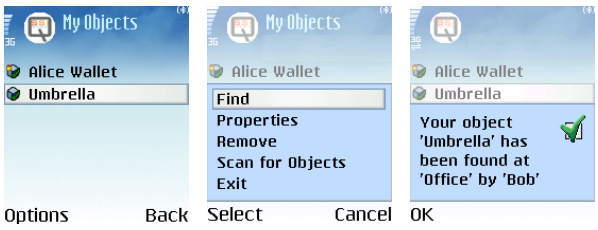
## 1.2 The Object Localization Application

Our object localization application involves various use cases concerned with managing everyday items by means of mobile phones.

**Remember.** The *remember* use case allows users to set up their mobile phone to store the context in which an object leaves the phone's local sensing range. This includes a trace of the user's location before and after the loss event and other people present or other personal objects carried at the time the object was left behind. As there will be numerous managed objects that users leave behind on a regular basis (for example when leaving their home), users will find it unpleasant to receive notifications each time objects go out of range. Instead, the relevant data is silently stored and can be used at a later time to help the user recall the circumstances of the loss or as a clue to the whereabouts of a lost object.

**Find.** In the *find* use case, the user can query the system for an item from the list of objects that have previously been associated with the user (Figures 1(a) and 1(b)). The system will then forward the query to a set of object sensors which, based on user preferences and system settings, are presumed good candidates to find the object. Object search strategies can be based on various heuristics, such as querying sensors near the location where the object was last within range of the user's device. Once a remote sensor has located the object, the user will receive a notification containing the object's location as shown in Figure 1(c).

**Delegate.** Our system also allows users to locally *delegate* care of a personal item from the personal mobile device to other object sensors installed in the environment.




(a) (b) (c)

Figure 1: User issues an object search query

For instance, a smart coat hanger in a restaurant can be tasked with guarding a coat and sending an alarm if it is removed.

**Gate.** Finally, users may add object sensors in places where these provide a particular benefit. As an example of such functionality, we support the *gate* use case, which involves installing an object sensing device to the door of a facility (such as an equipment room or industry lab) in order to record which objects leave with whom. Such functionality provides unobtrusive check-in and check-out management for the equipment used in the facility – and, like the *remember* use case, may contribute useful information for dealing with subsequent *find* queries.

In the remainder of this paper we present a system that can be flexibly adapted to implement each of the above use cases, while particularly focusing on the challenging *find* scenario. We first overview the system architecture in Section 2 and then detail our query services, which can be used to set up each use case, together with the methods developed for defining the scope of a *find* query in Section 3. We then discuss the privacy implications of the system in Section 4. In Section 5, we evaluate the practicability of mobile-phone-based object localization by means of a real-world experiment complemented by a series of simulations. We survey related work in Section 6 and provide conclusions on people-centered sensing in Section 7.

## 2 System Architecture

Figure 2 shows an overview of the system architecture. As mentioned above, mobile phones are used to link sensing functionality to users and the back-end infrastructure. Sensing functionality on mobile phones includes sensing the presence of tagged items, the phone's location, and other information relevant for remembering the context of an object's loss, as detailed in Section 2.1. Furthermore, our architecture involves *application-specific* services, such as associating objects and their owners, a user database, and profiling services that can be used to deduct heuristics for wide-area object searches. These services will be discussed in Section 2.2. All of the above are integrated using *query services* that support the implementation of the application's use cases. These include the *local query service*, used to set up individual mobile phones, the *global query service*, used to route queries on a global scale, and the *query scoping*

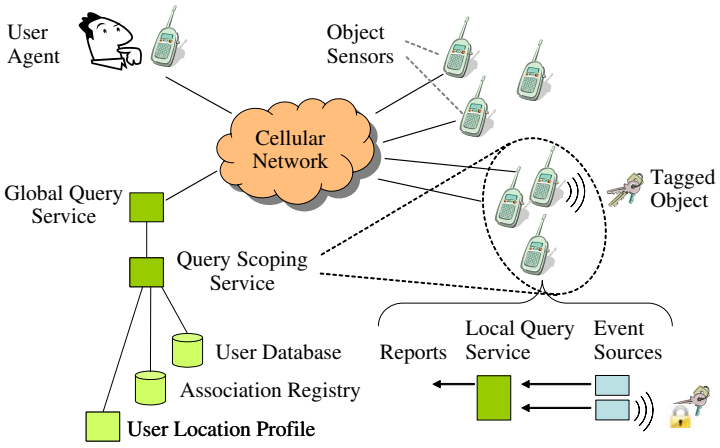



Figure 2: System architecture

*service*, which supports the latter in determining suitable receivers for a query (that are likely to produce a result) based on all kinds of history data stored by the application. These three query services will be detailed in Section 3.

## 2.1 Sensing Functionality

Apart from object sensors, which are integrated into each mobile phone, we presume that mobile devices have access to a variety of additional sensor information, in particular to sensors for determining the location of the user carrying the device.

**Event Sources.** All sensor information is encapsulated in application-specific software components which we name *event sources*. An event source generates events of a predetermined type, either in a periodic manner or when a certain condition is met. Our system uses event sources that generate an event whenever a tagged object comes *in range* or goes *out of range* of the device or when the device's *location* changes. It also uses an event source for *persons* who are within short wireless range of the user device.

In our prototype implementation, the *location* event source generates an event every time the mobile phone cell id changes. Object tagging is implemented using BT-nodes [3], tiny devices equipped with a Bluetooth radio, and object sensing is realized using Bluetooth discovery. The *in range* and *out of range* event sources generate an event every time a new object is discovered by Bluetooth or a previously seen object fails to respond to a connection attempt. Similarly, the *person* event source generates periodic events which contain a list of nearby users based on the Bluetooth names of their phones.

**Sensor Authentication.** Both object tags and devices representing people can restrict their visibility and allow only authenticated sensors to detect their presence. Such authentication is based on a shared secret that is exchanged between the actor initiating sensing and the sensed entity at an earlier time (the association service described below

is used for this purpose). After the initial key exchange, protected entities can be sensed only by sending an authenticated message. We discuss details of this concept, which is suited to tags of a small form factor [8], in Section 4.

Our Bluetooth-based prototype uses a simpler authentication principle. Protected people and objects are emulated by turning off visibility to Bluetooth discovery, while the MAC address takes on the role of a shared secret: sensing polls (for people or objects) consist of an attempt to connect to the correct address.

### 2.2 Application Specific Services

In our architecture, three system services have been designed to take on specific functionality mentioned in the object management application.

**Association.** The association service serves three main purposes. First, it keeps track of associations between users and objects (Figure 1(a) shows the list of Alice's associated objects). While objects can be detected by any object sensor in their initial state, once they have been associated they are detectable only within queries initiated by the object owner's device. Second, user to object sensor association allows users to maintain a set of object sensors that are particularly relevant for them (e.g., object sensors that have been installed by the user at home or at work; Bob's mobile device has previously been associated with Alice in the scenario depicted in Figure 1). Third, user-to-user association enables group access rights to certain objects, but is also used as a basis for disclosing the identity of associated users to each other (in Figure 1(c), Alice is shown the identity of Bob based on a previous association between them, otherwise the notification would only contain the location of the object). Similarly, users may choose to be visible only to *persons* event sources set up by associated users.

**Location Profile.** This optional service produces statistics on the locations (such as the physical location or the observed network cells) in which users spend most of their time. This makes it possible to implement a search strategy which gives preference to sensors at these locations, assuming that they represent places that contain personal belongings, such as a user's workplace or home.

Our prototype includes a rudimentary adaptation of a network-cell-based profiling system [16] enhanced with functionality for naming certain network cells that are particularly relevant (such as "Office" in Figure 1(c)).

**User Database.** We assume that the mobile network operator provides users with a database service in which application data such as previous reports of certain objects can be stored. The service can be used, for example, to record a log of objects reported by an object sensor that has been installed at a facility door – as mentioned in the *gate* use case above – for subsequent querying. Similarly, it can be employed to store details of the circumstances of an out-of-range event as required by the *remember* use case.

## 3 Query Services

The *query services* form the integrating element of our system, wiring the distributed components for the required application task. The back-end infrastructure hosts the

*global query service*, which exports a query interface allowing the desired application behavior to be set up according to the use cases *remember*, *find*, *delegate*, and *gate*. Each use case also contains some functionality that should be performed at the level of the individual mobile devices. Thus, each mobile device executes a *local query service* dealing with data requests at its local level.

To address the challenging *find* use case, the *query scoping* service supports the global query service by determining which sensors should be involved in a wide-area query, integrating information held within various application-specific services. For example, a query can be disseminated to a subset of associated users (based on information stored in the association registry), to locations that are particularly relevant for the user (based on location profile data), or locations near where the object was previously observed (based on the user database).

These query services are supported by a set of middleware services which are not detailed here. For instance, *storage services* are available both on the mobile device and in the back-end infrastructure – and are also used to implement the user database and the association service. Their implementations, including transparent serialization and database interaction for resource-constrained devices, are described in [1]. Furthermore, the means for local event forwarding, wide-area point-to-point messaging, and OSGi-based component execution platforms for both the mobile device and the back-end infrastructure have been implemented in our prototype. For details of these services, please refer to [12].

## 3.1 Query Service Interface

The interface of the query service we have developed consists of two parts. First, a *local part* specifies how an individual mobile device is to use its sensors when processing a query. Second, a *global part* specifies aspects related to query dissemination such as the mobile devices involved, and any time and cost constraints. Table 1 shows how the four use cases *remember*, *find*, *delegate*, and *gate* (introduced in Section 1.2) can be implemented using typical settings for the local and global parameters of a query.

| **Local** | **Remember** | **Find** | **Delegate** | **Gate** |
|---|---|---|---|---|
| Trigger | OutOfRange(obj) | InRange(obj) | OutOfRange(obj) | InRange(*any*) |
| Report | Location $(-120s, 60s)$<br>Persons $(-3, 3)$<br>Labels $(-2, 0)$ | Location $(-5, 0)$ | Location $(-1, 0)$ | Persons $(-20s, 20s)$<br>InRange $(-20s, 20s)$ |
| **Global** | **Remember** | **Find** | **Delegate** | **Gate** |
| Scope | local device | scope provider | delegate device | gate device |
| Sinks | local database | user interface | user interface | user database |
| $c_{\max}$ | $\infty$ | 10 | 0.5 | 50 |
| $e_{\max}$ | $\infty$ | 5 | 1 | $\infty$ |
| $q_{\max}$ | 1 | 100 | 1 | 1 |
| $t_{\max}$ | $\infty$ | 30 min | $\infty$ | $\infty$ |

Table 1: Parameter settings for the four use cases

The local part consists of two parameters. The first one is used to define when a query should produce a result. The programmer can specify such a *trigger* condition

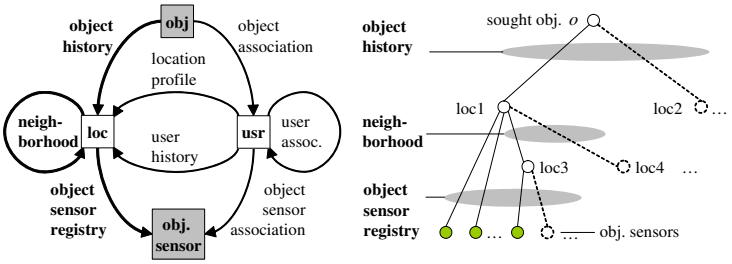
by providing an event source (this way, custom-implemented event source components can encapsulate event detection functionality). In our use cases, the trigger condition is based on properly parameterized *in range* and *out of range* event sources. These event sources can be parameterized to trigger when an arbitrary object comes within range, as in the *gate* example, or to trigger only with a certain object (as in the other examples).

The second local parameter defines the *report* that a query should generate. The report definition consists of event sources with annotated intervals $(a, b)$ stating the desired range of events relative to the time of the triggering event. Ranges can be based on time or on numbers of events. For example, the *remember* query specification is defined to report the user's *location* observed during the three minutes either side of the trigger event, the user's friends that were present at that time (a maximum of 6 events from the *persons* source), and the last two *labels* that have been observed (such *labels*, implemented by a technology similar to object tagging, may be attached to environments whose geographic location is irrelevant, for example, to identify the user's car or certain trains or busses). Similar examples are given for the remaining use cases. Event-based intervals are further annotated with an expiration timeout that enables reports to be generated even if not enough events arrive after the trigger.

The global part of the interface specifies who is to receive the query (its *scope*) and the resulting reports (the query's *sinks*), and also specifies the various limits on the query. The *scope* can either consist of a single mobile device (in the *remember* scenario the targeted device is simply the user's phone) or a custom-implemented *scope provider* component (as in the find use case). Scope providers, required to return a list of object sensors sorted by relevance, may be used to pass on a variety of search strategies to the global query service. For example, an association-based scope provider may simply return a list of previously associated object sensors. We discuss a more sophisticated scope provider component that can combine various search strategies in Section 3.2. The reports generated by the query, in turn, will be delivered to the specified *sinks*. Various system services may be wired as sinks: in the *remember* use case, reports will be delivered to the local database on the user's phone; in the *find* and *delegate* use cases, reports cause a direct notification on the user interface; and in the *gate* example, reports are collected in the backend user database.

Finally, different effort limits can be specified for a query: $c_{\max}$ limits the monetary costs billed to the user, $q_{\max}$ the number of object sensors that are involved in the query, $e_{\max}$ the total number of reports generated by the query, and, finally, $t_{\max}$ the total query time. In the *remember* scenario, the limits $c_{\max}$, $e_{\max}$, and $t_{\max}$ are irrelevant as the infrastructure involved is owned by the user. In the *gate* example, the user may employ $c_{\max}$ to cap costs incurred by data transmission to the backend database. In the *delegate* example, at most one object sensor should be queried ($q_{\max} = 1$) and at most one event, a notification, should be delivered ($e_{\max} = 1$). Moreover, monetary costs should be low.

The limits $q_{\max}$ and $t_{\max}$ are particularly relevant for the *find* example. Here, the global query service will distribute the query to the first $q_{\max}$ sensors returned by the scope provider. The term $q_{\max}$ is equivalent to the number of messages sent by the global query service in the dissemination phase and can be used to limit the communication costs. The parameter $t_{\max}$ may be used to specify the time that is




(a) Data model (b) Search tree implementing heuristics I

Figure 3: Data model and example algorithm execution

usually required for an adequate search strategy to find an object, and thus limits the search effort of inadequate strategies. Moreover, it allows for an iteration of different search strategies of increasing cost, where the next (more comprehensive) strategy is initiated after an unsuccessful timeout of the previous one. We will discuss suitable values for $t_{\max}$ in Section 5.

## 3.2 Query Scoping Service

To obtain suitable search strategies for wide-area queries of the *find* scenario is a challenge. Various heuristics can be used to distribute a query to a relevant subset of object sensors. Typically, sensible heuristics will be based on some kind of history data available in the system. For example, a query may be distributed to sensors near the location where the object was last observed.

**Data Model.** To elaborate on this, we show a simple data model of our application in Figure 3(a): *Objects* are associated with *users* (object owners) by the association service and also with *locations* (e.g., cells) where an object has previously been observed. Users may choose to record a history of their location on their mobile device (*user history*) or to enable the *location profile* service, which computes locations that a user visits frequently. In this simple model, locations are related to other locations via the *neighborhood* relation. Moreover, users can be associated with *object sensors* they often use (e.g., those they have installed in their office or car) and with other users who are family, friends, or colleagues. Finally, the mobile network operator maintains a database (*object sensor registry*) that stores the current location (e.g., the current network cell) of object sensors (including certain mobile phones that can be used as object sensors as well as stationary sensors).

For ease of description, we have omitted some details in the data model (in particular, a more refined location model). However, it is sufficient to demonstrate that typical search heuristics correspond to paths in Figure 3(a) from an entity of type *object* at the top to entities of type *object sensor* at the bottom, the latter representing those sensors that should be queried for the object. For example, we can query object sensors which:

I) Are near the location where the object was last seen.

II) Are near locations recently visited by the user.
III) Are near locations where the user spends a large amount of time.
IV) Are associated with the object owner (as in Figure 1).
V) Match the above strategies III and IV for a different (associated) user, such as a family member, or even for a friend of a friend, etc.

While, intuitively, none of these heuristics can guarantee success, they all incorporate sensible assumptions on where users keep personal belongings and where these are generally left. Note how each heuristic represents a path in the data model of Figure 3(a): Heuristic I corresponds to the path (*obj-loc-object sensor*) on the left, while heuristic V corresponds to the path *obj-usr-usr-loc-object sensor*.

Based on these considerations, a data model is a suitable means of expressing the real-world links between various types of data stored by the system, and can be used to generate a variety of search strategies – including all of the above. In particular, the system designer may assign weights $w(r)$ to each edge in the model, representing an estimate of whether exploring entities according to the relation type $r$ will be useful in the search, on a scale from 1 (very useful) to $\infty$ (not useful, hence very costly). For example, to implement heuristic I, the *object history*, *neighborhood*, and *object sensor registry* relation types would have a weight of 1, and all others a weight of $\infty$.

**Scoping Algorithm.** Parameterized with a weighted *data model*, a *source entity* $o$ (i.e., the sought object) and a *destination type* (i.e., object sensors), the scoping algorithm will "unfold" the data model into a search tree that contains entities stored by system services which are somehow related to the source entity $o$. The algorithm traverses the entities in the tree in order of decreasing (estimated) relatedness, essentially implementing a variant of the *uniform cost search* [22].

More concretely, the algorithm works as follows: It maintains a set of entities visited $V$ and a result list $Q$. Each entity $t \in V$ will be assigned a *relevance* measure $c(t)$, denoting how related $t$ is to the source entity $o$ given the data currently known to the system. Initially (line 1), $V = \{o\}$ and $c(o) = 0$, moreover $Q = \emptyset$.

**Algorithm 1**: Scoping algorithm based on uniform cost search.

**Input**: Data model $M = (E, R)$ with entity types $E$ and relation types $R$, weights $w(r)$ for relation types $r \in R$, source entity $o$ (of type $\in E$), destination type $d \in E$, entity limit $q_{\max}$.
**Output**: Result set $Q$ consisting of entities of type $d$

1 set $V = \{o\}$, $c(o) = 0$, $Q = \emptyset$
2 **while** $|Q| \leq q_{\max}$ **do**
3 pick relation $(u, v)$ between entity $u \in V$ and $v \notin V$ with smallest $c(u) + c(u, v)$ or exit loop if no such relation $(u, v)$ exists
4 add $v$ to $V$, set $c(v) = c(u) + c(u, v)$
5 **if** $v$ of type $d$ **then** add $v$ to $Q$
6 **return** $Q$

Based on the data referenced by the given data model, the algorithm in its core step (line 3) considers relations $\{(u, v)\}$ between entities $u \in V$ and $v \notin V$. If such

relations exist, the algorithm picks the relation $(u, v)$ with the smallest $c(u) + c(u, v)$. The costs $c(u, v)$ estimate the relevance of the relation $(u, v)$ based on two factors: $c(u, v) = w(r) \times g(u, v)$. The first factor, $w(r)$, is the relevance estimate which the system designer has assigned to the relation type $r$ corresponding to the relation $(u, v)$. Using the second factor, $g(u, v)$, the service that stores the relation may estimate the relevance of different $v$ for a given $u$. For example, the *location profile* could estimate $g(user, loc)$ for different locations using statistics on the amount of time the *user* spends at them. For the *object history* relation, it is intuitive that the *latest* location where the object was observed is the most relevant.

In the remainder of the loop, the algorithm adds $v$ to the set of visited entities $V$, updates the relevance estimate $c(v)$ of $v$ (line 4), and adds $v$ to the result set if it is of type *destination type* (line 5). The algorithm repeats these steps until up to *entity limit* related entities have been found, and then returns $Q$ (if the chosen destination type is *object sensor*, the entity limit corresponds to the limit on queried sensors $q_{\max}$).

Note that as $w(r)$ and $g(u, v)$ are issued on the same scale ranging from 1 (for very related) to $\infty$ (for unrelated), the computed relation costs $c(u, v)$ and the entity relevance measures $c(t)$ maintain the same semantics. Based on the costs $c(u, v)$, the algorithm explores the most relevant relations first, and the result list $Q$ contains entities $t$ of type *destination type* in order of increasing $c(t)$, that is, in order of decreasing relatedness to the start entity $o$. A sample execution implementing heuristic I (where $w(r) = 1$ for the *object history*, *neighborhood*, and *object sensor registry* relation types but $w(r) = \infty$ for all other types) is shown in Figure 3(b).

While based on the well-known *uniform cost search* method, the present approach lends itself well to distributed data sources: each system service that implements a relation type accessed by the algorithm is required to provide a single interface method *next(u)*, which allows an iteration through the entities $v$ related to $u$ in the order of increasing "unrelatedness" $g(u, v)$. Note that both the runtime and the space complexity of the algorithm depend on the data referenced in the given data model. For details of the scoping algorithm and its execution please refer to [13].

## 3.3 Global Query Service

We now discuss how the above scoping algorithm is used by the global query service to implement search strategies I-V in practice. For this, note that strategies I-III employ the *object sensor registry* relation, which associates a set of locations $L$ to a set of object sensors near them. Due to user mobility, however, the object sensors near set $L$ will change with time. Therefore query scoping is decoupled from the actual movements of users and their object sensors. That is, when heuristics I-III are implemented, *location* will be the *destination type* parameter passed to the scoping algorithm. The returned set of locations $L$, for example a set of cells, is then passed on to the global query service, which will distribute the query to sensors at these locations. The global query service is then concerned with maintaining a computed query scope over time while observing the cost control parameters ($q_{\max}$, $t_{\max}$, $e_{\max}$, and $c_{\max}$) specified in the query interface.

If search strategy IV is chosen, a set of object sensors is determined by the scoping algorithm. Here, a query will be distributed to the first $q_{\max}$ sensors returned by the scoping algorithm and be active for at most $t_{\max}$ time.

If search strategies I-III are chosen, the scoping algorithm will not directly return a set of object sensors, but a set of locations, as mentioned above. In the basic location model we employ, these can either be a set of cells (the most basic localization already available on the phone) or a set of geographic points (if phone localization is more precise) together with an associated measurement error. Because the set of object sensors associated with these locations may change over time, our system installs (or un-installs) a query at sensors which come close to (or depart from) these locations. Whether a sensor $s$ is close to the returned locations is defined by the implementation of a predicate $f$ (which maps $s$ to either *true* or *false*).

Depending on the way *locations* are modeled, we use two different implementations of $f(s)$. Given a set of cells $C$, $f(s)$ will be true if the mobile phone (with its object sensor $s$) is currently served by any of the cells in $C$. Note that this information is already available from the mobile network operator, that is, it can be accessed at the server end of our infrastructure without incurring additional communication costs.

If the mobile devices are equipped with a more accurate means of positioning, the locations returned by query scoping will instead be a set of geographic points $P$. Here, $f(s)$ will be true if the current position measured by a mobile phone's object sensor $s$ is less than a certain distance $d$ away from the points $P$. This distance $d$ will depend on the error incurred at the positioning sensor when the points in $P$ were measured (we will discuss a concrete implementation in Section 5). Note that such additional positioning information will only improve the efficiency of query dissemination if the positioning information of all object sensors is already known in a database on the server. Otherwise, it would be inefficient to propagate all object sensor positions to the server before query dissemination, and therefore a different approach is chosen. The query is distributed to object sensors in a set of cells $C$ which "cover" the whole area surrounding the points $P$ (the actual object sensor will be turned on only later, once the predicate $f(s)$ evaluates to true). Note that the total number of distributed queries is now the same as if locations were a set of cells $C$.

When installing queries for such location-based strategies I-III, the total number of object sensors at which a query will be installed ($q_{\text{total}}$) is made up of two parts, $q_{\text{total}} = q_{\text{init}} + q_{\text{mob}}$. Here, $q_{\text{init}}$ denotes the number of users queried initially at the time the query is issued – chosen from the initial query scope $S_{\text{init}} = \{s | f(s) = \text{true}\}$. In addition to $q_{\text{init}}$, the query will be installed at a second set of sensors $q_{\text{mob}}$ for which $f(s)$ becomes true while the query is active.

After a query is installed on an object sensor $s$, object sensing will be performed continuously until $t_{\max}$ expires. The mobile device associated with $s$ un-installs the query autonomously either when $f(s)$ becomes false or when $t_{\max}$ expires.

A query is declared successful if some object sensor $s$ reports having found $o$ at time $t_{\text{reply}}$ with $t_{\text{reply}} \leq t_{\max}$. The current position of $s$ represents the location at which the object was found and will be included in the reply issued to the user. A query is terminated without success once the query timeout $t_{\max}$ is reached.

# 4 Privacy Considerations

The query service, presented in the previous sections, makes use of a wide variety of personal data. It is therefore important to keep these data private and secure. In the following, we describe some of the privacy-enhancing features of our system.

Most prominently, tagged objects and persons can be protected from being sensed by unauthorized users with the help of a zero-knowledge authentication scheme as proposed in [8]. This protection is based on a shared secret $x$, which is known by an authorized user device $U$ and also by the sensed entity (i.e., the tagged object $o$). With query initiation, $U$ may issue a zero-knowledge authentication message (ZAM) in which the shared secret $x$ is made oblivious by means of a random session key $r$ and a current time stamp $d$. A ZAM $m$ has three parts: $m = (d; r \text{ xor } \text{hash}(d \text{ xor } x); \text{ hash}(r \text{ xor } x))$. A tagged object $o$ that receives $m$ can recover $r$ as it knows $x$ (using $d$ and part two, $o$ can compute $r$). Moreover, $o$ is provided with authentication proof that $U$ truly knows $x$ (using $r$, $o$ can check part three, which confirms that the sender truly knew the secret $x$). Note that for implementing this approach, tags or sensed objects are only required to evaluate *xor* operations and a simple hash function, capabilities which could even be implemented on passive tags [19]. For details on zero-knowledge authentication please refer to [8].

Based on this feature, if authentication fails, $o$ simply does not reply. Moreover, $o$ will only reply the first time it receives $m$. Therefore, an adversary $A$ cannot easily make use of forwarded queries to locate objects which have been lost by other users in public spaces. To be effective, $A$ must not only have been forwarded the find query containing $m$ but also be the first to *sense* $o$ using $m$. To achieve this, $A$ needs to control an infrastructure of object sensors that finds objects faster than the infrastructure provided by regular system users – in most cases an unrealistic assumption.

As noted, a ZAM represents a permit for sensing *once* only. For use cases in which sensing of an object or person $o$ must be performed *multiple times* within a given query (e.g., *delegate* or *gate*), there are three options. The first is to let an authorized user device $U$ re-issue a proper ZAM every time, which requires repeated end-to-end communication between $o$ and $U$. The second is to entrust the shared secret to the global query service (run by the mobile network operator on the back-end server), which may then issue authentication messages on its own. The third is to propagate the shared secret even further to the remote sensor $R$ (such as the object sensor employed in the *gate* scenario) for the time of query execution. The latter two options lower the communication overhead (the overhead of the third option is lowest) but require that the user is confident that the entrusted entity (e.g., the remote sensor $R$) does not leak the shared secret during the time of the query. After query termination, the shared secret $x$ may be changed – such a change could be propagated to multiple users $U$ previously associated with $o$ using the same authentication protocol.

A second privacy-enhancing feature of our system is the component deployment infrastructure, which permits a reconfiguration of the execution platform of services containing sensitive information within the application. That is, the association registry or the location profile, which should enhance object search functionality, could be executed on the user's mobile device instead of the server back-end, giving users full

physical control over their data. Similarly, deployment of the location profile service could be completely omitted – based on user preferences.

However, some potential privacy threats remain. One threat is shared by any system providing wide-area object localization – namely, that an adversary may attach a properly associated object to some person he or she wishes to track. Avoiding such threats is hard as they are related to the core functionality of the envisioned system. Possible approaches would be to limit the frequency of user queries, or make repeated consecutive queries for an object expensive. Alternatively, adversaries may leave an object at a known location, and then check whether associated users arrive at this location. Here, the required transparency can be obtained by making association services symmetrical, such that an object sensor carrier is allowed to see which associated users have sent queries to his or her device – while un-associated users are kept oblivious of each other by the mobile network operator.

Several extensions of these concepts are conceivable. For example, users could be offered a feature to protect private spaces [30], such as their home, from queries issued by unauthorized users. While dependent on accurate and verifiable location sensors for user devices, such protection of spaces could be enforced by the mobile network operator, which exerts the role of a gatekeeper in both directions, when forwarding user queries and when forwarding replies. This is similar to the way, in the present system, the network operator will ensure that a user's identity is only disclosed to associated users.

# 5 Evaluation

In the following, we examine whether our object search system can perform well enough to be a useful application. To achieve this, we evaluate the coverage of the system and the reply time for search queries in a real-world experiment and in simulations.

## 5.1 Real-world Experiment

In this section, we return to the scenario introduced in Figure 1, where the user is at home and is trying to verify the whereabouts of a given object that was left at the office. The mobile phones of the user's officemates (e.g., Bob) are registered with the association service and thus are considered relevant object sensors.

Our experiment was performed with four users working on the same floor. The users were given mobile phones running the object search prototype already tasked to perform continuous object sensing for all objects (using repeated Bluetooth discovery) and to report their findings at regular intervals to the back-end database. Similarly, 10 BTnodes[1] representing tagged objects were distributed in various rooms on the same floor. Figure 4 shows the experimental setup (tagged objects are shown as numbered circles while the offices of the four participating users are shaded).

[1] BTnodes [3] are class 2 Bluetooth devices which can be detected by regular phones over distances of a few meters.

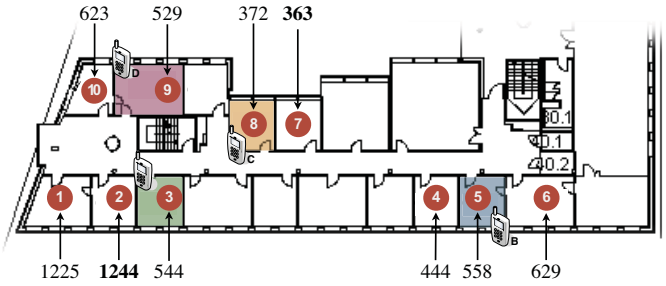



Figure 4: Experiment setup: Average reply times (in seconds) for the 10 tagged objects in different rooms

Note that Bluetooth may be too expensive and battery-intensive to be used as an object tagging technology in a practical system. Nevertheless, it allows us to test whether, given a better-suited future technology with similar radio characteristics, the mobility of a few office colleagues suffices to detect a given object within a reasonable time.

Over four consecutive days, all sensor readings entered the database as *(user, time, obj_id)* tuples. We considered only core office hours, that is, readings reported after the fourth user arrived in the morning and before the first left at night. This resulted in 30 hours of data. Based on these data, the reply time for a search query for a given object $o$ issued at an arbitrary time $t_q$ to the four office colleagues can be computed as $t_r = t_{\mathrm{DB}} - t_q$, where $t_{DB}$ is the time of the next database entry on the object $o$. This allows us to compute the average reply time for each object, assuming that queries for an object were distributed uniformly. In order to save messaging costs, user devices cached seen objects and only re-reported them to the database 10 minutes after their last report on the same object. This way, even if an object sensor has seen the object continuously, the resulting reports will yield an average query reply time of 5 minutes instead of zero.

For each object, Figure 4 shows the average reply time in seconds. Intuitively, we expect to obtain low values for objects with a participating user in the same room (objects 3, 5, 8, 9). Furthermore, note that the best results were obtained for objects close to the printer and the coffee machine (objects 7 and 8), while the worst results are for objects in rooms that were not visited by the participants during the experiment.

We show a cumulative density curve of the reply times observed for object 2 (with worst results), object 7 (with best results), and the average over all objects in Figure 7(a) later in this paper. In all cases, reasonable success rates (about 80%) could be obtained with a maximum query time $t_{\max}$ of 30 minutes. Note that results are satisfactory in spite of the latency commonly involved in Bluetooth discovery: as the discovery procedure involves a relatively long frequency-hopping sequence (in our experiments, roughly 8 seconds), a phone could theoretically "miss" objects if it is moved past them too quickly while scanning the wrong frequency band. However, this case rarely occurred in our experiments – while carried along the hallway past a room, phones would almost always detect objects located inside it.

## 5.2 Simulation Model

In the experiments described above, we focused on a small and confined search area and query scope. Here and in Section 5.3, we use simulations to investigate the characteristics of an object search system operating in a wide area with a larger user base, to provide design guidelines for a future realistic system.

Note that adequate models of a large-scale execution environment are difficult to obtain, as these must consider many aspects of daily life. To provide an accurate basis for system design, models must include the number of participating users, the frequency at which these users lose or search for certain objects, the number of tagged objects owned by each user, and particular scenarios determining when and where objects are lost. Intuitively, such a model contains many parameters which cannot be influenced by the system designer. We refer to such parameters as *environmental parameters*. Our approach to these parameters, of which there are vast numbers, is to investigate a significant proportion of them.

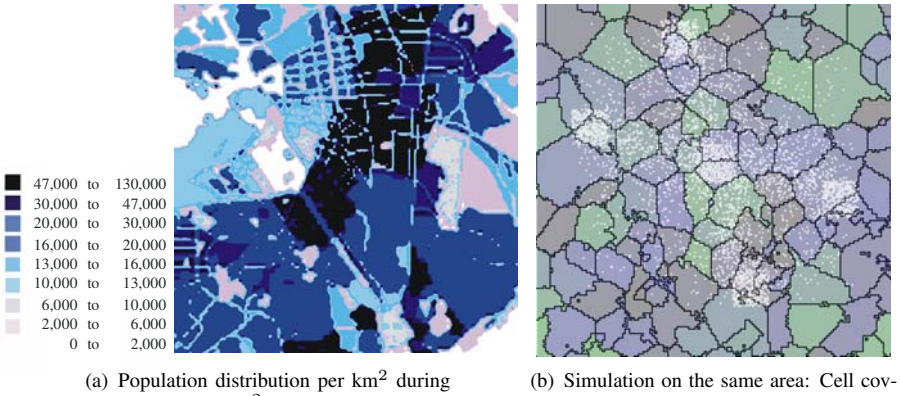
In addition to this, there are some *design parameters* that determine the system's performance and can be set and varied more or less directly by the system developer. These include the size of the search scope (the number of users participating in the search for an object), the range of an object sensor (which can be influenced by using more expensive tag and object sensing hardware), and the timeout period used for queries. For these *design parameters* we aim to find the most appropriate values, i.e., the parameter settings that can implement object searches with the lowest communication overhead for a given success rate.

**Scenario.** In the evaluated scenario, a user misplaces an object $o$ and later issues a search query to the global query service. We assume that at the time the object left the range of the object sensor, the user's mobile device recorded its location $p$. This location $p$ is used as a clue in the search (implementing heuristic I presented in Section 3.2). We evaluate two versions, a *cell-based* version in which $p$ is a cell, and a *position-based* version in which $p$ is a geographic point measured with a certain error. Note that if a user's local object sensor is not operated continuously but instead with a certain sampling frequency, the absence of an object could be detected late. This may increase the error of $p$ (to consider this in our simulations, we use high positioning errors, see "Sensor Model" below). In both versions, query scoping is performed as described in Section 3.3.

**Metrics.** It is the *success rate* of our system that we are most interested in. This corresponds to the proportion of queries for which a notification from some object sensor is received within the query timeout $t_{\max}$. Furthermore, we examine the overhead for query distribution $q_{\text{total}}$ including the part $q_{\text{mob}}$ which is caused by user mobility.

In our simulations, we do not examine object sensing costs explicitly, as we expect wide-area query dissemination to dominate the total cost due to the object sensor's shorter wireless range and the potential energy-efficient implementations of object sensing (e.g., it is usually sufficient to briefly activate an object sensor every time it is moved).

**Environment Model.** We assume that the object is left in a densely populated urban environment. In this setting, we study how an object can be found by pedestrians




(a) Population distribution per km$^2$ during the day of a 10 km$^2$ square area of downtown Lisbon [10]

(b) Simulation on the same area: Cell coverage model [6] and users (dots) distributed according to the hotspot mobility model

Figure 5: Environment models

moving within a square area. In the default case, the size of the simulation area is 1 km$^2$. The choice of the user density $u_d$ is derived from the total daytime population as estimated by the Momentum project [10] (a downtown Lisbon example which we cite from [10] is shown in Figure 5(a)). In our experiments, we only model those users who are *pedestrians* (for details of this rationale, see "Mobility Model" below) and are associated with *a single* mobile provider. Making a conservative estimate based on these factors, the user density range $u_d$ is 100-2000 users/km$^2$, values which represent only a small fraction of the estimated daytime population shown in Figure 5(a). Our default participant density of $u_d$=500 users/km$^2$, for example, corresponds to only *one* pedestrian user per 2000 m$^2$ of office space.

As mentioned, in some settings we rely on cell identifiers for positioning. To study such scenarios, we use actual position and orientation data from UMTS antennas together with a detailed model of land use types (e.g., buildings, highways, open, water) to compute the strongest-signal cell for each point in the simulation area [6, 18]. In Figure 5(b), we show an example of a resulting cell-coverage map computed for a UMTS network of downtown Lisbon. For cell-based scenarios, the simulation area is enlarged to 10 km$^2$ to let it contain a significant number of cells. While we are aware that in reality several cells may be observed at a given location at different points in time, we assume for our study that the object can be found in the cell where it was last seen. Results for scenarios in which several cells need to be searched to cover a certain location could be extrapolated from the results we provide.

**Mobility Models.** In our experiments, we assume that an *object* will not move once its owner has left it somewhere. In contrast, *user* mobility is a crucial aspect of our evaluation, as it has a high impact on the performance of the system in question. It is therefore particularly important that we make only conservative assumptions about user mobility. In this regard, we model only the slowly moving "pedestrian" proportion of the total population. This is based on the assumption that typical usage scenarios

involve locating objects inside buildings (e.g., at home, at work, in malls, or on campus) where users tend to follow pedestrian mobility patterns. Note that we consider far fewer users than the 50%-70% proportion of pedestrians mentioned in [10, p. 37], thus taking into consideration users who have not yet made their object sensors available to other users, are associated with a different cellular operator, are stationary, or are not moving on foot. Adding these users would improve coverage results, but only slightly: stationary users contribute less to coverage than mobile users, and fast users such as drivers would only help in the infrequent case where objects are left on the street.

In the most basic setting, we use a random waypoint mobility model parameterized for pedestrians. Users pick a random destination and start moving towards it at a speed drawn uniformly from the interval (2,4) km/h. (The average speed of 3 km/h is chosen according to the ETSI guidelines [9].) We choose trip destinations within 200 m of the user's current position and include pause times of 2 to 5 minutes in-between trips. We assume that such short trips and subsequent pauses capture the envisioned usage scenarios (e.g., in offices or malls) more realistically than the standard version of the model according to which users would follow straight paths for several kilometers. Moreover, such short trips result in a higher ratio of pausing users, and can therefore be considered a conservative assumption.

We also use a second mobility model which was derived from WLAN traces from the Dartmouth campus [15]. The model includes *hotspot* regions that represent central points on the campus (for example, a hotel, a library, and a cafeteria), which tend to contain many users and also represent popular destinations chosen by the campus population. In our adaptation, we model a typical campus environment using five hotspot regions, one in the middle and four shifted to each side of our simulation area. The size of each hotspot region is one hundredth of the simulation area. Half of the trips by a given user are made inside the current hotspot and half are directed to another arbitrary hotspot on the campus. The remaining (non-hotspot) area is called the *cold* region. In our simulation, users never choose a destination in the cold region, but only travel through it. As hotspot regions have a higher density, their positions and sizes are apparent in Figure 5(b), which shows 5000 users in a 10 $km^2$ area together with a cell-coverage model derived from downtown Lisbon. The chosen trips include 2 to 5 waypoints. The speed and pause times follow log-normal distributions parameterized according to [15, table 3]. The pause time distribution has a mean of 0.71 hours with a high standard deviation of several hours, as [15] found that users tend to stay in a hotspots for longer periods.

To avoid an initial transient period, we used initializations of user trips according to the perfect simulation method [17]. In the hotspot mobility model, however, some distributions had to be estimated, and thus a transient period of 1000 s remains. Object search queries are issued only after this period.

**Sensor Model.** A mobile device operates its object sensor (that is, polls for objects within range) once per time unit (usually a second) as long as a query is installed and running. This models the fact that some sensing technology (like Bluetooth) could miss objects if a sensor is only briefly in contact with the object (because, for example, it is moved too quickly). In the default case, we assume that the object sensor has a sensing

range of 5 m. A mobile device therefore sends a "found" notification if its sensor is active while within 5 m of the object sought.

Moreover, in some simulations we assume that the user has a position sensor available (e.g, GPS). To model the sensor's localization error, the position $p$ returned by the sensor is drawn uniformly from a disk centered around the actual position of the user. We refer to the radius of this disk as the *positioning error* $e_p$ used in the simulation ($e_p$ is set to 100 m unless otherwise stated; we use conservative values for $e_p$ as detecting the absence of an object late may increase the total error of the last known position of an object, even if the actual positioning technology is more precise). Note that with this error distribution, the density of observing an actual error, say $e$, is proportional to the circumference of a circle with radius $e$, and therefore the mean error is $(1/\sqrt{2})e_p$.

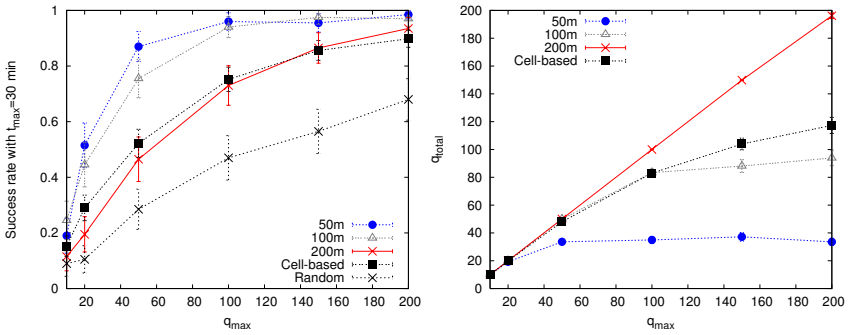
Alternatively, we also model a scenario in which the positioning sensor simply returns the cell to which the mobile phone is currently connected. The benefit of cell-based positioning is that all information on cell associations is also available at all times at the server end of our infrastructure, thus permitting the efficient implementation of query scoping. Using cell information for scoping does not prevent phones from activating a more advanced positioning technology (such as GPS, or combined signal strength information from multiple nearby cells) in situations where precise positioning is particularly important, for example when they have found an object or when they are storing a loss position $p$. However, even if $p$ is more exact than a cell identifier, query scoping is still based on cells – unless more exact sensor positions are available at the base station (cf. Section 3.3).

## 5.3 Simulation Results

Using the simple scenario and the environment models described above, we aim to investigate several aspects of a future object sensing system. Foremost, given some scope, we want to confirm whether it is possible to find objects with reasonable success rates and a small enough overhead. Furthermore, we aim to investigate how cell-based scopes compare to position-based scopes and to a random query dissemination strategy which queries a certain proportion of all users. Moreover, we aim to gain an insight into the sensitivity of the system's performance with regard to parameters such as user mobility, object sensing range, and chosen query timeouts.

**Success rate.** In the first set of simulation runs, we investigated the query success rate observed with *position-based* scoping and *cell-based* scoping. These scopes are implemented according to Section 3.3, based on the location $p$ where the object was last within range of the user's device. In the position-based version, the scope $S_{\text{init}}$ consists of sensors within a disk around $p$ of radius $r = s_r + e_p$ where $s_r$ denotes the range of the user's object sensor and $e_p$ the maximum positioning error. In both versions, if $|S_{\text{init}}| > q_{\max}$, then $q_{\max}$ sensors are randomly chosen from $S_{\text{init}}$.

Figure 6(a) shows the proportion of successful queries (in which the sensors queried have located the object within 30 minutes) when the user-imposed limit $q_{\max}$ on the number of queried sensors is varied. Five different graphs show the results obtained with different positioning errors $e_p$ (from $e_p$=50 m to $e_p$=200 m), cell-based scoping, and a random strategy where we distributed the query to a fraction of $q_{\max}/500$ of all




(a) Success rate (b) Sent messages

Figure 6: Success rate and overhead with different positioning technologies

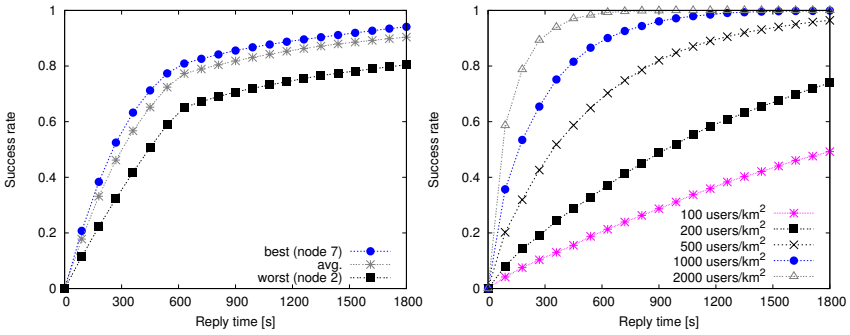
users. For all graphs, the success rate obtained can be increased by raising $q_{\max}$ and reaches acceptable levels with $q_{\max}$=200.

The communication effort involved in the same runs, determined by the number of sensors $q_{\text{total}}$ which were actually queried, is shown in Figure 6(b). As, by the definition of our protocol, the search area becomes larger with an increased positioning error, the required effort increases as well. Similarly, searching the whole coverage area of the cell where the object was left requires more messages to be sent before reasonable success rates are obtained. Note, however, how in Figure 6(b) the number of sensors queried $q_{\text{total}}$ at some point stops growing with the user-imposed limit $q_{\max}$. This is because with small enough scopes, the object is found before the limit $q_{\max}$ is reached. Observe also how the performance of cell-based scoping is comparable to a 200 m positioning error and even outperforms the latter in terms of communication effort $q_{\text{total}}$. This is because, with position-based scoping and large positioning errors, many ineffective queries are sent to mobile devices which erroneously measured a position that was close to the position clue $p$.

Finally, as Figure 6(a) shows, any scoping performs better than a random strategy. Even if 40% of all users are queried (i.e, $q_{\max} = 200$), the success rate of the random strategy is still only around 60%. Needless to say, its communication effort is worst as it is proportional to the total number of users (not shown).

**Timeout, sensing range, and different mobility models.** Apart from scoping, several other parameters may significantly influence the performance of the system.

The first is the timeout used for queries. Here, the question is whether a more adequate choice of timeout (previously set to $t_{\max}$=30 min) can be made when waiting for successful replies. Note that choosing an adequate timeout is particularly relevant when *object sensing itself* is considered a significant cost. Particularly since in reality the object might be outside the chosen scope, it is important not to continue futile sensing for too long, but at the same time to issue a confident "not-found" reply. Furthermore, system performance is expected to vary with user density. We show the interplay of these two parameters with position-based scoping in Figure 7(b). Each graph represents the




(a) Real-world measurements (b) Random waypoint simulations

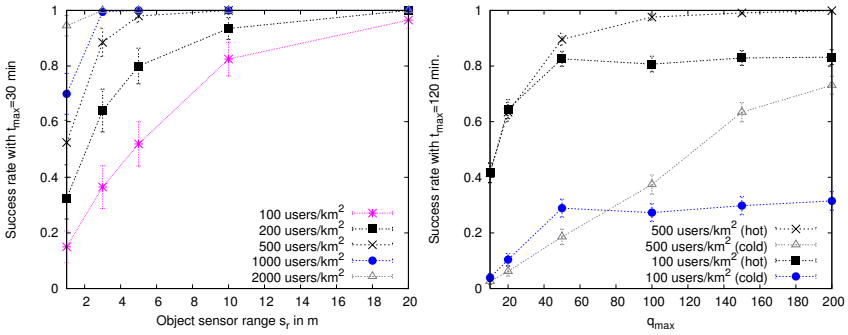
Figure 7: Cumulative density functions of reply times

cumulative density function of the reply time obtained after 5000 repeated simulation runs (each data point represents the proportion of requests answered within the given timeout period) in which no limit was set on communication effort $q_{\max}$. As expected, the likelihood of finding the object increases with a longer timeout, but for high user densities, short timeouts (5 to 10 minutes) are sufficient. Moreover, high success rates can be obtained with $t_{\max}$=30 min even with user densities as low as 500 users/km$^2$. For lower densities, longer timeouts must be used.

Observe that the graphs of Figure 7(a) from our office floor experiments (see Section 5.1), in which the actual user density was greater than 4000 users/km$^2$, are comparable to user densities of 500 to 200 users/km$^2$ in Figure 7(b). This is compatible with our earlier conjecture that the random waypoint simulation only models the "pedestrian" proportion of all users, and confirms that the approach of looking at user densities that are smaller than in reality is valid.

A second important parameter, which is expected to have a major impact on performance, is the range of the object sensors used. In the runs shown in Figure 8(a), we demonstrate the impact of the sensing range on the success rate of position-based scoping. With 2000 users/km$^2$, even a range of 1 m yields acceptable results. As expected, however, the sensing range has a high impact. When designing a practical system that is to be robust at low user densities, it seems worthwhile investing in object sensing technology with a higher range.

Finally, a third important parameter is the mobility of the system's participants. Figure 8(b) shows the success rate observed with the hotspot mobility model when raising the message limit $q_{\max}$. We show four graphs for the cases in which the object was left at a hotspot or in the cold region with two different user densities. Because user pause times in this model are quite long, we extended the query timeout $t_{\max}$ to 2 hours. Note, however, that the total number of queries remains limited to $q_{\max}$ and therefore the results remain comparable to simulation runs based on the pure random waypoint model shown in Figure 6(a). Here, for very low user densities, the success rate cannot be improved by raising $q_{\max}$ as the timeout remains the predominant constraint. For




(a) Sensing range (b) Hotspot mobility

Figure 8: Varying sensing range and mobility

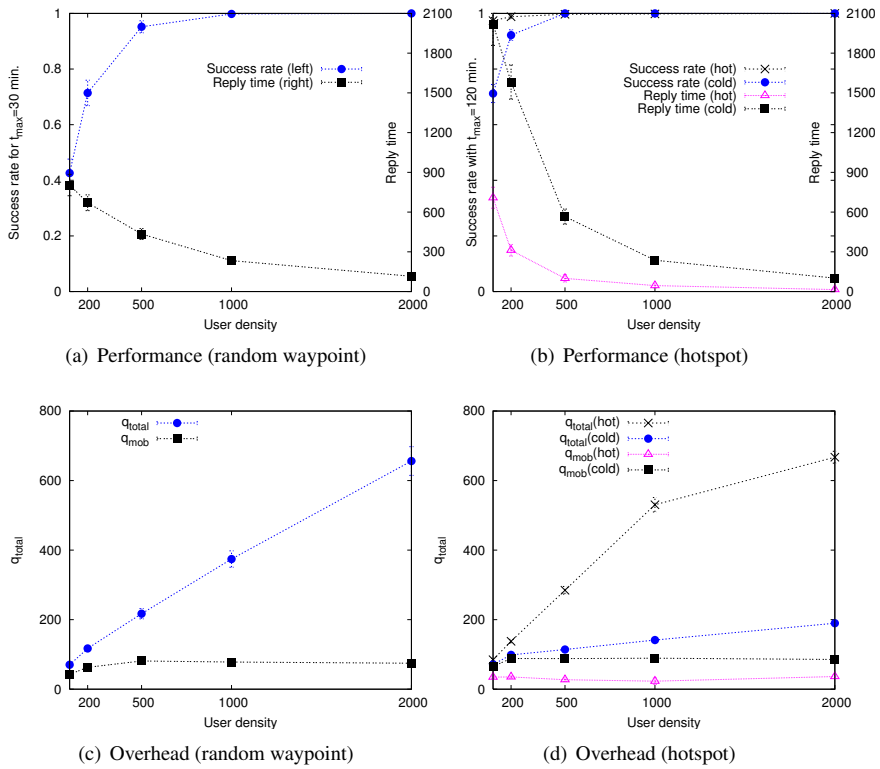
500 users/km$^2$, however, the object can often be found with at most 200 messages, even if it is located within the cold region.

**Effects of increasing user density.** Additional insight can be gained when the user density is varied. Such experiments were performed with cell-based scoping and are shown in Figure 9. We show the success rate and the query reply time while varying the user density in Figure 9(a), and similarly the results for the hotspot mobility model in Figure 9(b). The corresponding overheads are shown in Figures 9(c) and 9(d) respectively. Note that for these runs no limit $q_{\max}$ was set.

Both overhead figures show the total overhead $q_{\text{total}}$ and the overhead due to user mobility $q_{\text{mob}}$ included in the total. Observe that $q_{\text{mob}}$ does not increase with higher user densities. One reason for this is that the query reply time decreases with increased user density and therefore compensates for the expected increase in the mobility-based overhead. Quite differently, $q_{\text{init}}$ (equal to $q_{\text{total}} - q_{\text{mob}}$) increases proportionally to the user density as the number of queries is not limited by a particular $q_{\max}$.

The main result here is that once the success rate is good, querying additional users does not improve results apart from lowering the reply time. In other words, waiting for users to move is more efficient than simply querying more users. As a consequence, if a higher reply time is acceptable, then the protocol can manage with far fewer queries by choosing a smaller initial number of queries $q_{\text{init}}$.

**Summary.** Summing up, we observed that high success rates can be obtained with a range of different mobility patterns and scoping variants. Cell-based scoping, which is free from the additional overhead of propagating object-sensor position information, proved to be particularly valuable. Finally, in certain circumstances the system may even work reasonably well with very low participant densities representing a hundredth of the expected daytime population in an urban area.




(a) Performance (random waypoint)

(b) Performance (hotspot)

(c) Overhead (random waypoint)

(d) Overhead (hotspot)

Figure 9: Cell-based scoping when the user density is varied

# 6 Related Work

Various research papers argue for the relevance of locating everyday objects, monitoring the presence of items, or avoiding their loss. Many such systems [2, 5, 24, 28, 30], however, suggest a specific pre-installed object sensing infrastructure, which is costly to deploy and to maintain. Reminder systems [2, 24] focus on notifying users before a loss takes place. As mentioned, we generally assume that tagged objects will often be intentionally left behind and therefore avoid immediate notification. If desired by users, however, the query service supports immediate notifications as well (as in the *delegate* use case, which is similar to [24]).

An object search system could also be implemented by proactively sending all sensor readings to a centralized database (subsequently queried when an object needs to be located). Such a system would face the formidable scalability challenge of a global data collection system, such as IrisNet [14] or Hourglass [20]. In contrast to this, we are using a reactive (query-based) approach, as the number of sensor readings (e.g., *object X seen by object sensor A*) is expected to be much larger than the number of queries.

Some research on distributed sensing systems also advocates the use of mobile phones and the mobile network infrastructure [4, 7, 23, 26]. Apart from the different application focus, we address two central challenges of large-scale applications based on user-held sensors: *Query scoping* (determining which sensors should be queried from a large array of sensors) and the properties of the wide-area *sensor coverage* obtained by user-held sensors. These challenges are likely to re-appear in other applications [4, 7] as well.

# 7 Conclusion

We have presented a comprehensive system for managing and finding everyday objects relying on mobile phones as omnipresent object-sensing devices. We have discussed the architecture, design, and expected performance of this system, together with a flexible means of generating object search heuristics from application data. Based on the ubiquitous mobile network infrastructure which is already in place, wide-area searches for everyday objects become possible without incurring the high costs involved in equipping a larger environment with an object-sensing infrastructure.

Our system makes use of an unconventional approach, which relies on the participants' mobility in order to cover an essential portion of the search space. We therefore spent significant effort on modeling and testing the circumstances in which such an object search system would be used on a large scale. The results are encouraging. In all our experiments, we were able to observe a high rate of successful queries, that is, of objects being found. While the time until a reply can be obtained varies with user mobility and density, our conjecture – that most of the time an object will eventually be found – was confirmed. Moreover, we could show that even in settings with relatively high positioning errors or which rely solely on the observed cell identifier for localization, the total overhead for distributing an object search query remains acceptably low. While this does not change the basic fact that objects left in deserted places will not be found, we have shown that, for objects left within the users' space, such a system is feasible.

In a broader context, this paper has analyzed the properties of the coverage obtained from user-held sensors. By means of the average query reply time (e.g., 30 minutes) that we observed with a particular sensing range, participant density, and mobility pattern, we quantified the time that must elapse before a point-shaped phenomenon has been sufficiently covered. Therefore, the query reply time can be interpreted as the (reciprocal of the) maximum sampling frequency that our user-centric infrastructure realizes for a particular spot within the area under observation.

In this regard, the results are applicable to other wide-area sensing applications. For example, recent work has mentioned measuring air quality or average noise levels in urban areas [4]. In such systems, the trade-offs examined between sensing range, maximum sampling frequency, and participant density are likely to re-appear. These observations indicate that studying Sensor Internet applications that make opportunistic use of mobile phones and their infrastructure from a more general point of view is a worthwhile research issue.

**Acknowledgments.** The work presented in this paper was supported by DoCoMo Euro-Labs and partially by NCCR-MICS, a center funded by the Swiss National Science Foundation. We would like to thank Michael Fahrmair and Daisuke Ochi for their thoughtful comments and suggestions on draft versions of this paper, Christof Roduner and Chie Noda for their contributions to the presented work in earlier phases of our research collaboration, and Hans-Florian Geerdes for his valuable advice on the Momentum dataset [18]. Parts of this paper were published earlier in a preliminary form [11, 12, 13].

# References

[1] Philipp Bolliger and Marc Langheinrich. Distributed persistence for limited devices. In *System Support for Ubiquitous Computing Workshop (UbiSys'06) at UbiComp'06*, Orange County, CA, USA, September 2006.

[2] Gaetano Borriello, Waylon Brunette, Matthew Hall, Carl Hartung, and Cameron 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] BTnodes. `www.btnode.ethz.ch`, 2006.

[4] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava. Participatory sensing. In *SENSYS'06 Workshop on World-Sensor-Web: Mobile Device Centric Sensor Networks and Applications (WSW'06)*, Boulder, CO, USA, November 2006.

[5] Christian Decker, Uwe Kubach, and Michael Beigl. Revealing the retail black box by interaction sensing. In *3rd International Workshop on Smart Appliances and Wearable Computing (SAWC'03) at ICDCS'03*, Providence, RI, USA, May 2003.

[6] Andreas Eisenblätter, Hans-Florian Geerdes, and Ulrich Türke. Public UMTS radio network evaluation and planning scenarios. *International Journal on Mobile Network Design and Innovation*, 1(1):40–53, 2005.

[7] Shane B. Eisenman, Gahng-Seop Ahn, Nicholas D. Lane, Emiliano Miluzzo, Ronald A. Peterson, and Andrew T. Campbell. MetroSense project: People-centric sensing at scale. In *SENSYS'06 Workshop on World-Sensor-Web: Mobile Device Centric Sensor Networks and Applications (WSW'06)*, Boulder, CO, USA, November 2006.

[8] Stephan J. Engberg, Morten B. Harning, and Christian 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.

[9] ETSI. Selection procedures for the choice of radio transmission technologies of the UMTS. Technical Report 3.2.0, European Telecommunications Standards Institute, April 1998.

[10] Lucio Ferreira, Luis M. Correia, David Xavier, Allen Vasconcelos, and Erik Fledderus. Deliverable d1.4: Final report on traffic estimation and services characterisation. Technical Report IST-2000-28088, Momentum Project, 2003.

[11] Christian Frank, Philipp Bolliger, Christof Roduner, and Wolfgang Kellerer. Objects calling home: Locating objects using mobile phones. In *Proceedings of the 5th International Conference on Pervasive Computing (Pervasive'07)*, Toronto, ON, Canada, May 2007.

[12] Christian Frank, Christof Roduner, Philipp Bolliger, Chie Noda, and Wolfgang Kellerer. A service architecture for monitoring physical objects using mobile phones. In *Proceedings of the 7th International Workshop on Applications and Services in Wireless Networks (ASWN'07)*, Santander, Spain, May 2007.

[13] Christian Frank, Christof Roduner, Chie Noda, and Wolfgang Kellerer. Query scoping for the Sensor Internet. In *Proceedings of the IEEE International Conference on Pervasive Services (ICPS'06)*, Lyon, France, June 2006.

[14] Phillip B. Gibbons, Brad Karp, Yan Ke, Suman Nath, and Srinivasan Seshan. IrisNet: An architecture for a worldwide sensor web. *IEEE Pervasive Computing*, 2(4), 2003.

[15] Minkyong Kim, David Kotz, and Songkuk Kim. Extracting a mobility model from real user traces. In *Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'06)*, Barcelona, Spain, April 2006.

[16] Kari Laasonen, Mika Raento, and Hannu Toivonen. Adaptive on-device location recognition. In *Proceedings of the 2nd International Conference on Pervasive Computing (Pervasive'04)*, Vienna, Austria, April 2004.

[17] Jean-Yves Le Boudec and Milan Vojnović. Perfect simulation and stationarity of a class of mobility models. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'05)*, Miami, FL, USA, March 2005.

[18] Momentum. Models and simulation for network planning and control of UMTS. momentum.zib.de/data.php, May 2006.

[19] Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. RFID privacy issues and technical challenges. *Communications of the ACM*, 48(9):66–71, 2005.

[20] Peter Pietzuch, Jonathan Ledlie, Jeffrey Shneidman, Mema Roussopoulos, Matt Welsh, and Margo 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.

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

[22] Stuart Russell and Peter Norvig. *Artificial Intelligence: A modern approach*, chapter 3, pages 75–76. Prentice Hall, 1995.

[23] SensorPlanet. www.sensorplanet.org, 2007.

[24] Hirobumi Shimizu, Osamu Hanzawa, Kenichiro Kanehana, Hiroki Saito, Niwat Thepvilojanapong, Kaoru Sezaki, and Yoshito Tobe. Association management between everyday objects and personal devices for passengers in urban areas. Demonstration Abstract in Adjunct Proceedings of Pervasive'05, Munich, Germany, May 2005.

[25] SkyeTek. SkyeModule M9 embedded UHF RFID reader data sheet. www.skyetek.com/Portals/0/Documents/Products/SkyeModule_M9_DataSheet.pdf, 2007.

[26] Dirk Trossen and Dana 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)*, pages 485–489, July 2005.

[27] Ultra Low Power Bluetooth Technology. bluetooth.com/Bluetooth/Learn/Technology/lowpower.htm, 2007.

[28] Roy Want, Kenneth P. Fishkin, Anuj Gujar, and Beverly L. Harrison. Bridging Physical and Virtual Worlds with Electronic Tags. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'99)*, pages 370–377, Pittsburgh, PA, USA, May 1999.

[29] Wibree Technology. www.wibree.com, 2006.

[30] Kok Kiong Yap, Vikram Srinivasan, and Mehul 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.