

Query Scoping for the Sensor Internet

Christian Frank, Christof Roduner
Department of Computer Science
ETH Zurich, Switzerland
{chfrank,roduner}@inf.ethz.ch

Chie Noda, Wolfgang Kellerer
Future Networking Lab
DoCoMo Communications Laboratories Europe
{noda, kellerer}@docomolab-euro.com

Abstract—In a large scale distributed system with a large number of sensors interconnected through a wide-area network infrastructure, it is advantageous not to disseminate a query to all available sensors, but only to a subset of the most-relevant ones. We target an application scenario where mobile phones trace users’ objects (which are equipped with small identification tags), distribute useful context information related to these objects, and are able to locate them when lost or misplaced. For locating a lost object, the proposed algorithm – parameterized with a data model of the application domain – is able to explore a wide range of heuristics based on history data present in the system (on objects, users, and their past location), similarly to the way a human user would re-iterate all that she/he knows about a lost object in order to locate it. As the proposed algorithm uses the data present in the system to parameterize its execution, it is generic enough to be applied to other application domains.

I. INTRODUCTION

In our everyday life we often search for personal items such as keys, wallets, umbrellas or sports bags that we have either lost or misplaced. Emerging technologies, such as RFID systems and wireless sensor networks, properly integrated in traditional mobile infrastructure, have a great potential to allow finding these objects quickly and effectively when needed. We envision that daily-life objects will be equipped with small identification tags that will allow detecting them via radio communication. This will allow mobile phones to sense objects (or act as gateways to devices with object sensing capability, which we term *object sensors*) and thus both record and distribute useful context information related to users and their belongings [1].

Mobile network providers could leverage their wide-area infrastructure to integrate both: all kinds of installed *object sensors* and a potentially high number of mobile network customers carrying handhelds that include object sensing capability. Such a platform would provide a (nearly) ubiquitous infrastructure for locating tagged objects and enable owners to locate misplaced, missing, or lost items. Central to this platform is what we term the *ubiquitous gateway*: A terminal to the mobile network that acts as a mediator between *object sensors* and the wide-area infrastructure (e.g., cellular, Internet).

In such a wide-area infrastructure, however, it is required to disseminate a query only to a subset of all available sensors in order to perform efficient and scalable search. As 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), sending all readings to and storing them in a central database is inefficient. On the other hand, mobile resources are too scarce than to simply flood queries to all ubiquitous gateways. In particular, by modeling the way how a usual user would search for a misplaced item, we have data available to prioritize search to certain gateways. In this paper, we propose an algorithm which, based on the application’s data model together with various data stored by different system services, is able to follow a wide range of heuristics – on objects, their past location, the whereabouts of their owners and social relationships among users.

Moreover, the proposed algorithm is independent of the application domain. Its data exploration technique finds association paths by combining related facts and thereby computing a measure of *relatedness* between different entities of the system. The application programmer only needs to specify *which* entity to begin with (in our case, the lost object) and the type of entity to arrive to (in our case, an entity of type *ubiquitous gateway*, which will then forward the query to available object sensors). In addition to that, the application programmer can annotate the system’s data model with weights estimating the (user-perceived) importance of individual relations. The algorithm will then generate an ordered list of *ubiquitous gateways* that – according the given estimates – are most likely to find the lost object.

II. SERVICE ARCHITECTURE

Various auxiliary services are required to implement the scenario outlined above. We briefly overview these in this section to delineate various information available in the system which may later be used to locate a lost object.

Association: The association service serves three main purposes: First, it 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 associations maintain groups of users such as families in order to facilitate access management. Third, user to object sensor association allows maintaining a set of “favorite” object sensors that are particularly relevant to the user. A user may, for example, setup some object sensors in places that are of special interest to him or her, such as a holiday home.

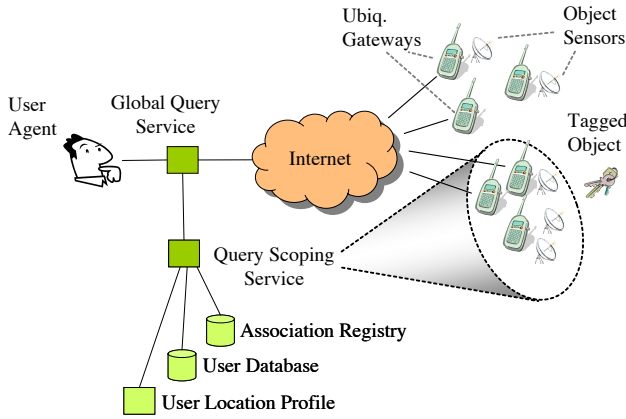


Fig. 1. System Architecture

Gateway and Global Storage: Our application requires persistent storage services. One scenario includes saving suitable context information each time a personal item leaves the communication range of the user’s gateway (specifically storing a location trace of the user around the “loss” event) that can help finding it later on. In a different scenario, user-installed object sensors may simply report a stream of sensed objects that pass by them. For these cases, the service infrastructure provides users with a *user database* service that may be used as a sink for events generated by object sensors and ubiquitous gateways. Moreover, all association relationships are kept in a storage component called *association registry*. Storage services are available both on the user gateway and in the backend infrastructure [2].

Localization: For remembering the context of an object when it goes out of range and to provide location information for found objects, we need to have location information available on the ubiquitous gateway. While the current implementation is based on GSM cell information, GPS can be added if increased accuracy is desired.

Location Profile: In our prototype, we adapted [3] in order to obtain an ordered list of GSM cells in which the user spends most of his/her time. This lets us implement a search heuristic which mainly considers locations where the user spends much time.

Figure 1 shows an overview of the system architecture: As mentioned, *ubiquitous gateways* are used to link *object sensors* to the backend infrastructure. The backend infrastructure hosts the *global query service*, which provides adequate dissemination support, cost control, and validity management for user queries. The query dissemination is based on the *query scoping service*, which implements application specific heuristics for retrieving the most appropriate subset of gateways based on history data provided by other system services. Such history data could, for example, consist in information on past object locations and be used to implement a heuristic that starts searching where the object was last seen. Similarly, the association registry could be used to disseminate a query to gateways that are associated to the user (i.e., gateways that

the user commonly uses, e.g. at the office or in his/her car). A third example data source could be based on the *user location profile* (based on [3]), which is able to extract the user’s most frequented locations.

III. A QUERY SCOPING SERVICE

So far we’ve mentioned several heuristics that could be used to distribute a query to a large number of gateways and object sensors. Specifically, we could query sensors which:

- 1) Are near locations where the object has been seen in the past.
- 2) Are near locations recently visited by the user.
- 3) Are near locations where the user spends a large amount of her/his time.
- 4) Are associated with the object owner.
- 5) Match the above strategies 3 and 4 for a different (associated) user, such as a family member, or even for a friend of a friend, etc.

While, intuitively, these heuristics cannot guarantee success, each incorporates particular application-level assumptions on where users keep personal belongings and where these are generally left.

However, it is not clear how many such strategies exist and which ones to combine. Most strategies employ history knowledge and try to leverage the fact that users, objects, and sensors are somehow *linked* in the real world through common locations, associations, friends, or points in time. We will show in Section III-A how these real-world *links*, expressed in terms of a data model, can be used as a basis to generate a variety of search strategies – including all of the above.

Because each strategy could – depending on the particular history of the user and/or object – yield a large number of relevant ubiquitous gateways, it is particularly important to allow applications to limit the number of gateways that are queried in a particular stage of the search and allow the user to expand the search (involving additional costs) later on.

We briefly summarize these points: We require a query scoping algorithm which is able to implement a wide range of heuristics depending on the given application, further, allows the user or application programmer to express preference for various heuristics while not disregarding the others completely, and finally, returns a list of ubiquitous gateways sorted by success probability according to the user’s estimates.

A. Data Model

An example data model of our application is shown in Figure 2: *Objects* are associated to *users* (object owners) by the association service and also to *locations* (e.g., cells) in which an *object* has been observed. Users may choose to record a history of their location on their mobile device (*user history*) and further to have the *location profile* service running, which computes the *locations* that are most relevant for a given user. In this simple model, locations are related to other locations via the *neighborhood* relation. Moreover, *users* can be associated to certain *gateways* which they often use (e.g., which they have installed in their office or car) and

to other *users* which are family or friends. Finally, the mobile network operator keeps a database (*GW Registry*) which stores the current location (e.g., the current cell) of certain *gateways*.

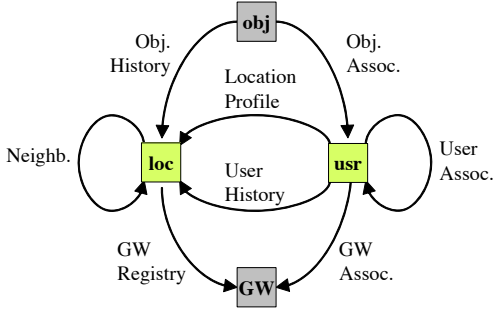


Fig. 2. Data model. Boxes represent entity types and arrows represent relationship types.

Note that we omitted many details in the data model (most prominently a more refined location model and the fact that *ubiquitous gateways* interface *object sensors*) for ease of exposition. However, it is sufficient to show that each employed heuristic (from the heuristics listed at the beginning of Section III) corresponds to a path from an entity of type *obj* at the top to an entity of type *gateway* at the bottom of Figure 2. As an example, heuristic 1 (*query sensors that are near locations where the object was seen in the past*) corresponds to the path *(obj-loc-GW)* on the left, while the heuristic 5 corresponds to the path *obj-usr-usr-loc-GW* traversing the *location profile* relationship type.

The algorithm is thus parameterized with a data model – as an example with the one shown above – and further with weights $w(r)$ for each edge in the model, representing the application programmer’s estimate whether exploring entities according to this relationship type will be useful in the search, on a scale from 1 (very useful) to ∞ (not useful). For example, to implement heuristic 1, the *obj. location history*, *neighborhood*, and *GW registry* relationship types would have weight 1, and all others weight ∞ .

B. Unfolding the Data Model: The Scoping Algorithm

We can now proceed to employ such a parametrized data model to generate a growing search scope for a given object.

1) *Relationship Adaptors*: The different relationship types depicted in Figure 2 are implemented by different system services (described in Section II), which can be executed both in the background infrastructure or on the personal mobile device. Each of these services thus implements the interface of a *relationship adaptor*, which is used by the query scoping algorithm.

The *relationship adaptor* interface allows, for a given entity u , to iterate through a list of entities $\{v\}$, which are related to v according to this relationship. The method $next(u)$ returns the next entity v from the list of entities related to u . Entity v is returned together with a measure of *relatedness* $g(u, v)$ between the entities u and v . Specifically, $g(u, v)$ denotes how relevant v will be for a successful search compared to

the other target entities related to u . The measure $g(u, v)$ is on the same scale as $w(r)$ ranging from 1 for very related to ∞ for unrelated. Subsequent calls to $next$ will retrieve entities ordered from the most to the least *related* target entity. Moreover, a second interface method $peeknext(u)$ is analogous to $next$ but simply returns the current destination entity v without advancing to the next.

Note that most relationship types in our data model have an attribute that allows for a simple implementation of the *relatedness* estimate $g(u, v)$: Regarding *obj. history*, it is intuitive that the *latest* location where the object was observed is the most relevant. *Locations* can thus be ordered by the most recent time value. The same order can be applied to the *user history* relation. Similarly, *location profile* can rank *locations* by the amount of time the user spends in them and use this amount of time to calculate $g(u, v)$. The *neighborhood* relation can estimate g between two *locations* based on physical proximity.

2) *Algorithm*: The algorithm, given the data model and a start entity s calls the $next$ method at the relevant relationship components present in the system to generate a *search graph*, in which nodes correspond to entities (somehow related to entity s through one or a series of relations). The costs $c(u, v)$ of each edge (u, v) are based on the two relevance estimates, made by the relation adaptor and by the application programmer: $c(u, v) = w(r) \times g(u, v)$.

Let V denote the set of entities already visited. Each entity $t \in V$ will be assigned a *relatedness* measure $c(t)$, which basically corresponds to the length of the shortest path from s to t with respect to the local edge weights $c(u, v)$. This way, $c(t)$, maintains the same semantics: a low value represents a high measure of *relatedness* between s and t .

Initially, $V := \{s\}$ contains the sought object and $c(s) := 0$. At each step, the algorithm picks the edge (e, e') with $e \in V$ and $e' \notin V$ with smallest $c(e) + c(e, e')$.

Given such an edge, the algorithm then sets $V := V \cup \{e'\}$, queries e in case e is of type *gateway*, and further sets $c(e') := c(e) + c(e, e')$. The algorithm repeats this step until a fixed number of gateways has been queried (the user may set this number and is then offered to continue to search if the search was not successful).

This is a variant of *uniform cost search* [4] in which we do not expand all children of a node e at a time as there are potentially too many, due to several relationships relevant for e and a potentially huge list of related entities provided by each one. Therefore, at each step, we instead only add *one* child entity e' to the search tree.

3) *An Example*: Consider an example execution in which the application programmer has set the weight values $w(r)$ to 10, 2, and 3, respectively, for the *obj. location history*, *neighborhood*, and *GW registry* relationship types – while for all other types $w(r) = \infty$ – in order to implement the proposed heuristic to search “where the object has been seen in the past”. In this example, we simply implement $g(u, v)$ to return the position of v in the (sorted) list of entities related to u .

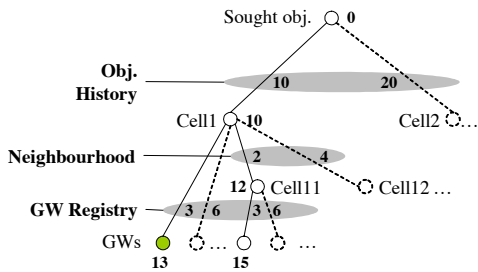


Fig. 3. Search tree, edges and nodes labelled with relatedness estimates

Algorithm execution is shown in Figure 3: Starting with s , only the *obj. history* relation is relevant (given the weights $w(r)$), and a call to *next* generates the *location* where the object has been seen last, in this example *cell1*. Given a cell entity, *GW registry* may generate child entities representing *ubiquitous gateways* in that cell, while *neighborhood* generates children which are other, neighboring, cells. The other relations will never contribute any child entities because of their high weight.

In this example, entities s , *cell1*, *cell11*, and two entities of type *gateway* have been visited. Each of them may be used to generate additional child entities according to the shown relations (the potential child entities are shown as dashed lines). From these candidate entities, the algorithm picks the edge (*cell1*, *cell12*), adds *cell12* to V , sets $c(\text{cell12})$ to 14, and continues with the next step (in which it would query the gateway marked in Figure 3).

4) *Discussion*: Through its main element, a *uniform-cost search* based on the combined relatedness values $c(u, v)$, at any given point in time, the algorithm will have explored *all* entities that are within a given level of *relatedness* to the start entity s . It will therefore also query gateways in the order of their *relatedness* to s , which is the property we desire.

Note that while our search strategy inherits the exponential space complexity present in any breadth-first search, this is not a significant disadvantage in our application domain: Firstly, the costs of an object search billed to the user will most likely be proportional to the number of queried gateways (due to communication costs). As only a constant (user-defined) budget is granted, at most a *constant* number of gateway entities will be in the search tree (as any visited gateway is queried during search). Secondly, the number of intermediate entities (like locations) in the tree can also be limited by raising the weights of individual relations, which will re-configure the algorithm to explore “deeper” paths earlier and reach gateways faster. In the example shown in Figure 3, a high weight $w(r)$ of the *obj. history* relation would cause the algorithm to explore many gateways at already known locations before generating additional ones.

IV. RELATED WORK

Our work is related to various fields. Recent work has argued for the relevance of different aspects of our application [5], [6], [7]. While [7] explicitly includes object search, the authors do not address the query scoping problem. Query

scoping itself is required in various contexts: In wireless sensor networks, generic frameworks for selecting eligible nodes based on their local properties (e.g., position) have been presented [8], [9]; in mobile ad hoc networks, message forwarding based on the last-known encounter between two mobile nodes has been implemented [10].

Note also that our algorithm is a variation of standard graph exploration techniques [4]. However, as querying a gateway is associated with physical costs, we cannot benefit from search strategies that examine large portions of a solution space (e.g., Tabu search [11]). We instead need to obtain a relatively small set of likely solutions (gateways likely to find the object). To achieve this, we described a generic way to parameterize the search with a data model of the application domain.

V. CONCLUSION AND OUTLOOK

We presented an approach for query scoping which explores chains of relationships between entities of the application’s data model. It is able to integrate local estimates of *relatedness* between entities in the data model together with user-assigned weights representing estimates of the relevance of individual *relationship types* present in the model. This way, it is able to incorporate *any* data present in the system as algorithm input, while exploring flexibly many search strategies at once and obeying user preferences for the search.

Future work includes evaluation of the algorithm with respect to actual user data (group and object associations, user profiles) collected using our prototype implementation [2] and a reformulation of the weights used in the presented approach in a probabilistic context.

REFERENCES

- [1] C. Frank, C. Roduner, C. Noda, M. Sgroi, and W. Kellerer, “Poster abstract: Interfacing the real world with ubiquitous gateways,” in *Adjunct Proc. of EWSN 2006*, Zurich, Switzerland, Feb. 2006.
- [2] P. Bolliger, “Query services for the Sensor Internet,” Master’s thesis, ETH Zurich, Jan. 2006.
- [3] K. Laasonen, M. Raento, and H. Toivonen., “Adaptive on-device location recognition,” in *Proc. of Pervasive 2004*, Vienna, Austria, Apr. 2004, pp. 287–304.
- [4] S. Russel and P. Norvig, *Artificial Intelligence: A modern approach*. Prentice Hall, 1995, pp. 75–76.
- [5] H. Shimizu, O. Hanzawa, K. Kanehana, H. Saito, N. Thepvilajanapong, 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.
- [6] G. Borriello, W. Brunette, M. Hall, C. Hartung, and C. Tangney, “Reminding about tagged objects using passive RFIDs,” in *UbiComp 2004*, Nottingham, England, Sept. 2004.
- [7] M. M. Kok Kiong Yap, Vikram Srinivasan, “MAX: Human-centric search of the physical world,” in *SenSys 2005*, San Diego, CA, USA, Nov. 2005.
- [8] J. Steffan, L. Fiege, M. Cilia, and A. Buchmann, “Scoping in wireless sensor networks: a position paper,” in *Proce. of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing*, Toronto, Ontario, Canada, 2004, pp. 167–171.
- [9] C. Frank and K. Römer, “Algorithms for generic role assignment in wireless sensor networks,” in *SenSys 2005*, San Diego, CA, USA, Nov. 2005.
- [10] M. Grossglauser and M. Vetterli, “Locating nodes with ease: last encounter routing in ad hoc networks through mobility diffusion,” in *Proc. of INFOCOM 2003*, vol. 3, San Francisco, CA, USA, Apr 2003.
- [11] F. Glover, “Tabu search: A tutorial,” *Interfaces*, vol. 20, no. 5, 1990.