Toward Ubiquitous Searching

Miaomiao Wang[1,2], Jiannong Cao[2], Yan Sun[2], Jing Li[1]
[1] *Department of Computer Science,*
The University of Science and Technology of China, Hefei, Anhui, China
[2] *Internets and Mobile Computing Lab, Department of Computing,*
The Hong Kong Polytechnic University, Kowloon, Hong Kong
{csmmwang,csjcao,csysun}@comp.polyu.edu.hk

Abstract

In this paper, we propose a novel concept called "ubiquitous searching", which allows people to organize and search the desired information about the objects in the physical world, navigating from one object to others through their contextual links, just like what we do in the web searching. How to realize such an exciting idea poses many challenges, and a new approach is needed to provide scalable system abstractions and infrastructures. We identify the design principles and propose the framework toward ubiquitous search. The system architecture and the key algorithms for the proposed framework are developed. We also describe a proof-of-concept prototype and a simulation study used to experiment with our framework and algorithms.

Keywords: Ubiquitous Computing, Wireless Sensor Network, Searching, Physical World

1. Introduction

Two trends in networking and computing technologies motivate the work described in this paper. First, recent years have witnessed the rapid advances in embedded devices [1], wireless sensors networks (WSNs) [2], and mobile communication technologies [3], as well as their fast growing applications. The widespread deployment of integrated sensing, computing, and communication systems is transforming the physical world into a ubiquitous computing platform. Sensing tags, memory, computing and communication capabilities are immersed into our living environments, appearing on motion detectors, door locks, light bulbs, alarms, cellular phone, vehicles, and possibly in person's wallet or even key rings [4]. It is foreseeable that, in the near future, we will be offered the opportunities to access and search the information about the physical objects directly, in a way much the same as searching the virtual world on the web using Google or Yahoo.

Another trend is the rapid spread of the concept and development of techniques of information searching. Nowadays, people can search the information on the Internet about web-pages, pictures, music, and even satellite maps of the Earth and the Mars. To extend the searching from the cyber world to the physical world will be an exciting application which is not far away from us.

In this paper, we propose a novel concept called *ubiquitous searching*. The key idea of ubiquitous searching is to acquire, organize, and browse the desired information about objects in the physical world, navigating from one object to others through their contextual links, just like what we do in the web searching.

An application scenario of ubiquitous searching is that, a person named Jack can start the search using the keywords "dog, black", as well as his personal information such as "Jack, password", and then obtain all the available information (with appropriate privacy restrictions) around the world about black dogs, such as "belonging to whom" "location", "near to what", etc., ranked by some pre-defined order. The most desired dogs will be presented the first. Jack can then select a particular dog for further information. He may find that the dog is near a cat and, if he is interested in the cat's information, he can simply click on the link for that "cat" object.

However, how to realize the exciting idea of ubiquitous search faces many challenges. An immediate question is how to enable people to directly get the timely information about, and related to the desired objects, given the huge amount of diverse, dynamic, and heterogeneous information available in the large scale physical environment. A new approach is needed to provide scalable system abstractions and corresponding algorithms.

This paper contributes to ubiquitous searching in three ways. First of all, we first time propose the

978-1-4244-1890-9/07/$25.00 ©2007 IEEE

concept. Second, we identify the design principles, and propose the ubiquitous searching framework (USF), which includes the system model with the definition of the abstract information type UIO (Ubiquitous Intelligent Object), and the searching model with the searching interface and two key algorithms. One algorithm is the mobile agent-based crawling algorithm for information gathering. Another algorithm is the ranking algorithm for information re-organizing and presentation. Third, we present a proof-of-concept prototype built to demonstrate our framework and describe simulation results to evaluate our algorithms. However, as mentioned before, there are many challenges to address for realizing ubiquitous search. This paper serves as the very first step, providing insights into the problem, discussing possible solutions with the hope to inspire more related research on the topic.

The rest of the paper is organized as follows: Section 2 briefly reviews the related work. Section 3 presents the design of the ubiquitous searching framework (USF) and its core algorithms. In section 4, we present the design of a system prototype with an application scenario and a simulation study of the proposed crawling algorithm. Finally, we conclude this paper by discussing the future works in Section 5.

2. Related Work

To the best of our knowledge, this paper is the first one that originates the integration of the concepts of web searching and ubiquitous computing. In this section, we briefly review the existing works with similar motivations or contributing to the realization of this concept.

Researchers recently have made efforts on designing systems, both in concept and implementation, for acquiring information from the ubiquitous computing environment. Several works have addressed the issue of how to get the pre-defined, homogeneous information from a small scale, close, and static environment [5, 6, 7, 8]. Some works [15] use RFIDs for physical object identification. Other works, e.g., Cougar [6] and TinyDB [5], use wireless sensor network (WSN) for querying the physical object values. They abstract the WSN as a database, and make use of the distributed query processing techniques to obtain the data by using SQL-like statements.

Techniques have been proposed for obtaining the location information of dynamically moving objects [9, 10, 11]. The techniques involve the use of radio frequency, ultrasound, and video capturing.

Combining the above two concerns, research has been done [9] to address the problem of "Human-Centric Search of the Physical World". It is similar to the database approach in query processing, but the main concern here is how to obtain the location information about a specific object in a small scale network.

SensorWeb [12, 13, 14] shares the similar motivation with our work. It is an emerging trend to make various types of web-resident sensors, instruments, image devices, and repositories of senor data, discoverable, accessible, and controllable via the WWW. However, the main objective of SensorWeb is resource sharing. In contrast, our work is oriented to information searching and browsing.

In terms of information searching, global context aware service discovery [19, 20] also shares some part of motivations with our work. These works focused on searching desired services using pre-defined protocols by organizing pervasive data using some global index. The works in [21, 22], proposed "Context-aware browsing of the world", but did not touch the *ubiquitous searching* concept, and no algorithm is provided for ranking the search results according to their relevance.

In summary, our work relates to and differs from the existing works in the following ways. First, our proposed USF is developed as an overlay, built upon the supporting techniques available now and in the future. Second, compared with the works sharing the similar motivations, our work provides a systematic approach with abstractions and algorithms for ubiquitous searching. We address the issues of searching and browsing the objects, not only the services provided by them. In addition, we address the issue of how to rank the information about the objects people are interested in searching.

3. Ubiquitous Searching Framework

In this section, we first describe the design principles and then introduce the USF. We propose the key algorithms and mechanisms for implementing the framework.

3.1. Design principles of USF

As mentioned, from both the architecture and the algorithm aspects, to design a system to realize the ubiquitous searching concept is not an easy task, we have identified the following principles that should be followed in the design of the framework.d

Openness and scalability: the system should be able to address the objects in a large scale ubiquitous computing environment, supporting the dynamically changing sets of objects and their relations. Standardized and scalable abstractions should be defined, including the information metadata, the system model, the system structure, and the operation interfaces.

Privacy: physical objects are more privacy sensitive than the traditional Internet files (e.g. documents or web pages). Information about the objects should be protected by appropriate privacy mechanisms.

Appropriate algorithms: the algorithms designed for traditional cyber-world searching are not suitable for ubiquitous searching, mainly for two reasons. First, the supporting techniques (e.g. embedded devices and wireless communications) for ubiquitous searching are different from those used in the web searching environment. Second, the information about physical object is different from the WebPages in terms of the metadata, information dynamics, etc. New searching algorithms are needed for gathering, extracting and organizing the information.

3.2 System architecture of USF

The system architecture consists of the abstract object model, the system model, the system structure, and the searching interface.

We define the *physical objects* in the real world, equipped with the sensing tags, computing, storage, and communication capabilities, as *ubiquitous intelligent objects* (UIOs). Some UIOs are stationary, while others are mobile. UIOs can be always or intermittently connected to the network. Today much progress has been made for realizing such concept. The devices that can be used to implement the UIOs may be electronic labels/tags, RFIDs, MEMS devices, tiny sensors, and embedded software, etc.

The abstract object model (meta data) defines the data description of the UIOs. The meta-data of a UIO contains three parts: *Self_description, Ability_description,* and *Relationship_description.*

The pre-defined *Self_description* describes the features of the UIO, which specifies what or who the UIO is, and usually will not always change. The structure of *Self_description* is a tuple **[UIO_ID, attributes-list (name, value, privacy)]**. UIO_ID is globally unique and organized hierarchically reflecting UIOs class inheritance relationship. Every attribute has a name, value, and privacy class, which can be *public_RW, restricted_W_public_R, or restricted_R*. The UIO_ID reflect relationships the between all the UIOs.

Ability_description describes the information that specifies what UIO has "perceived" and "derived". This information can be the raw data captured from the sensors in a UIO or processed results by the UIO. In particular, we define UIO that can get the temporal and spatial information as *self-conscious* UIOs. The structure of *Ability_description* is a tuple **[Information_type, Values, Report_interval, Privacy]**. Here, the privacy class can be *public_R or restricted_R*. Writing is not allowed here, because the information should just reflect the fact themselves as obtained. Report_interval indicates the interval for value reporting, and the setting of the value depends on the Information type and the status of the UIO. For example, a UIO equipped with a thermometer sensor can perceive the temperature information, and if the sampling rate of the temperature information is low, the corresponding Report_interval should be long. In addition, the status of the UIO will affect the Report_interval. An example is a UIO equipped with a GPS sensor. When the moving speed of the UIO is fast, the Report interval should be short.

Relationship_description describes the relationships between the UIOs, which can be used for navigating from one UIO to another. For large and static UIOs (e.g. hill and the lake), their relationships can be pre-defined. For other UIOs, their relationships are dynamically obtained by using the supporting mechanisms. The structure of *Relationship_description* is a tuple **[Out_set, In_set]** Out_set contains all the other UIO_IDs that the UIO are aware of, while In_set contains all the other UIO_IDs that "know" this UIO. Note that, "being aware" can be logical and spatial. There are two kinds of logical UIOs relationships. One is the structural relationship, for example, "composed by". Another kind is the behavior relationship, for example, "used by" or "play with". The spatial relationship can be "near" or "up to", etc.

The system model of USF is cluster-based with two tiers. The first tier is a mesh network consisting of all the UIOs, which are clustered into sub-systems. The second tier is an overlay composed of the master UIOs of the subsystems. Client (As a specific UIO) can search the information about the UIOs using different devices through the master UIOs. Figure 1 illustrates the system model.

In general, the UIOs are clustered according to their physical locations. The master UIOs are usually static and of a large scale, with the abilities of handling the searching requests. For example, a campus is defined as a master UIO, because it is aware of all the buildings and students on the campus and can support the searching of them. A subsystem can have one or more master UIOs, which can also be organized hierarchically. One subsystem has one top layer UIO. The global information about the physical world is shared by all top layer master UIOs. The search can span a number of relevant subsystems, e.g., by flooding.

Figure 2 shows the structure of USF in one subsystem (for simplicity, only one master UIO is shown). The structure is composed of several parts. Among them, the *search engine*, *ranking*, *index and*

analysis, *utilities*, and *crawl control* are associated with the mater UIO.

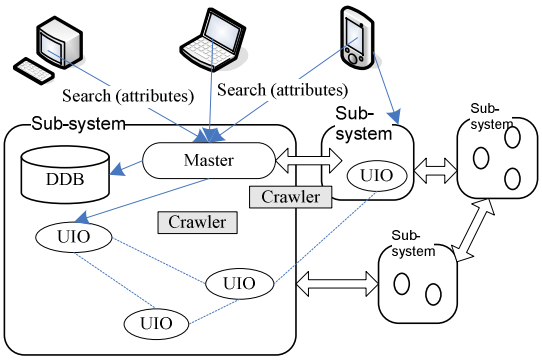

Figure 1. USF system model

Search Engine is responsible of receiving the search requests from the clients, and sending the ranked results to the client. *Crawlers* are a small program implemented by using mobile agents. They "browse" the physical world on behalf of the search engine. *Crawler control* is responsible of controlling the actions of the crawler agents. *UIO information repository* is a distributed database for storing the information gathered by the crawler agents. *Indexer and analysis* is responsible of parsing the UIO information, and making the inverted files for later search. *Ranking* is used to re-arrange the UIO information and present the most relevant information to the user. *Utilities* contain policies used by the related algorithms.

The interface of USF is of the following format:

Search (Result_information_list, Keywords, Search_process type, Validation)

Result_information_list contains the search results, sorted by the object ranking algorithm. *Keywords* are provided by the client for the search. The Searching engine will map the keywords to the meta-data of the UIO using the *Utilities*. *Validation* is a set of values that denote the client's identity. USF will map the client identity to the roles in using the *Utilities*. The relationships between the roles and the UIO information privacy are also maintained in the *Utilities*.

USF provides two types of search processes: *Common search* and *Fresh-focused search*, depending on whether the search is initiated on demand in response to an incoming request. *Common search* has three steps. In the UIO-information-acquiring step, a master UIO periodically gathers information about the UIOs in the corresponding subsystem using mobile agents controlled by *Crawlers control*. In the UIO-information-reorganizing step, the *index and analysis* module will scan all the gathered information, construct reverse sorting index table, and store the results for the next step. The inverted files are periodically rebuilt. Finally, in the Searching-response step, *Result_information_list* is sent back to the client by *Search Engine*.

On the other hand, *Fresh-focused search* aims at searching the most recent information about the UIO. Because of the large number of UIOs, the newest information about the desired UIOs may not be available in the distributed database. For searching the UIOs with high information dynamics, the client can use this kind of search. Whenever the fresh-focused search is initiated, a mobile agent is sent out immediately to find the related information.

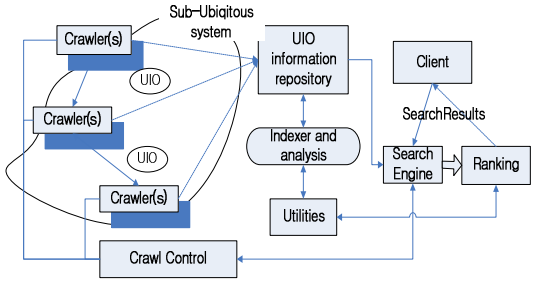

Figure 2. USF system structure

3.3 Key Algorithms for USF

Related works in web-search are involved with developing necessary algorithms, such as crawling, indexing, and ranking. To support our USF, we also need to consider the key algorithms. Here, we propose two core algorithms for USF: a mobile agent-based crawling algorithm for UIO information gathering and an UIO ranking algorithm for UIO information re-organization. The two algorithms are similar in function but different from the traditional corresponding versions used in Cyber search in terms of the implementation approach and strategies.

The basic scheme of our crawling algorithm is shown in Figure 3. The master UIO first initializes a crawler agent. (We explain later why a mobile agent approach is used). Then the crawler agent *replicates* itself and *dispatches* the copies to the UIOs selected in the OUT_set of the master UIO by the *Object_selecting* function.

When the crawler arrives at a required-crawled UIO, it executes the *Gathering* function to inspect the *meta data tuples* of the UIO, stores the extracted information to the data structure called *GatheredInfo*, send the *GatheredInfo* back, and inspects the *Relationship_description* to select the UIOs to crawl next. Then, the crawler executes the *Forward* function to replicate itself and dispatches the copies to the selected UIOs. To avoid redundancy, the *Replicate* function first checks whether a remote UIO needs to be crawled.

Each crawler is associated with a timer which defines the scope of crawling. The crawler will stop when the timer expires or cannot find more

required-crawled UIO, then it runs the *Notify* function to inform the parent UIO that task has been done. Due to the recursive replication of the crawler to all UIOs in the network, the master UIO will eventually end up with messages containing all the information about the whole network. Note that, the above crawling algorithm is used in Common search. In Fresh-focused search, the crawler agent will bring the *keywords* and *Validation* information during crawling. In the *Gathering* function, validation will be performed and only validated information will be gathered.

Now, let us explain the design concerns and principles behind this algorithm, and discuss the potential problems and possible optimizations.

First, the crawler should be as flexible and light-weight as possible. Here we use the mobile agent approach for its flexibility and scalability. It allows not only the convenient update of the crawler algorithm and strategies by creating a new version of crawler in the Master UIO, but also easy handling of adding new UIOs or deleting old ones, because there is no need to maintain the data gathering route. In contrast, the data base approach needs to update the distributed query processing function of all UIOs in case of code change, and needs to update the query dissemination and results gathering routes according to the dynamics of UIOs.

The mobile agent can migrate or clone itself. Here we use clone because the copies of the agents are parallelized crawlers in nature. This not only helps to achieve better performance but also facilitates potential remote cooperation between the copies. There are many possible cooperation problems here deserving investigation. For example, we have assumed that the Out_set and In_set do not change during one crawling process. If they change, however, the crawlers should cooperate to proceed with the process. Another example is that, if we allow the crawlers to cross the boundary of the sub-systems, then the crawlers in different sub-systems should be able to cooperatively control the information gathering process. The crawler of our system is running on an overlay, optimizing the supporting network mechanism will improve the performance of the crawler.

```
Algorithm on master UIO:
1.  initialize a crawling agent A;
2.  A Replicates itself;
    Dispatch the copy of A to
    Object_selecting(Master_UIO,
    Selected_UIO_ID)
3.  Wait for reply_message or timeout;

Algorithm on UIO being visited:
1.  Gethering (GatheredInfo, Refresh_strategy,
    Privacy strategy) send back GatheredInfo;
2.  Object_selecting(This_UIO,Selected_UIO_ID)
3.  set num of children=num (Selected_UIO_ID);
4.  set num_replies to 0;
5.  Forward();
Forward() {
    A Replicates itself;
    Dispatch the copy of A to (Selected_UIO_ID);
    while {1} {
    wait for reply_message or timeout;
    incr num_replies;
    if (num_replies = num_children or timeout)
        { Notify(); Exit; }
    }
}
```

Figure 3 Mobile agent based crawling algorithm

Second, the crawler should not depend on or interfere with the UIOs much. So using the event based method for UIOs to report the refreshed information is not practical. Thus, the crawler has to decide how frequently to re-visit the UIOs it has already crawled, in order to keep the Master UIO informed of the changes on the UIOs. The refresh strategy is implemented in the *Gethering* function. It is dependent on the meta data because different meta data have different changing characteristics. For *Self_description* the crawler will get the information according to an adaptable pre-defined frequency stored the *Unities*. For *Ability_description,* the crawler will get the information according to the *Report_interval*, and for the *Relationship_description* the crawler will get the information according to the historical log or analysis results stored in the *Utilities*. For analysis of the relationships, some promising techniques in the literatures can be adopted. For example, techniques proposed for social network and social networking [22, 23] can be used to address the relationship of UIOs and their changing characteristics by means of either mathematical models and/or experiments.

Third, the crawler should provide the mechanisms to protect the privacy of the UIOs. The privacy strategy is also implemented in the *Gethering* function. For the *public_RW* and *restricted_W_public_R* information, the crawler will gather the information without pre-processing, while for the *restricted_R* information, it will encrypt the information.

Fourth, the crawler should provide efficient results in terms of the defined metrics. The crawler should carefully decide which UIOs to crawl and in what order. This is because 1) the master UIO may have limited capacity and may not be able to get all the information; 2) crawling is always time consuming, and crawlering all the UIOs will consume too much time; and 3) not all the UIOs are necessarily of equal

interests to the client. *Object_selecting* is responsible of implementing this design principle, with the goal of visiting the UIOs of more importance before visiting the UIOs of less importance. The *Importance Rank of UIO* will be discussed later in the Information-dependent UIO ranking algorithm.

The basic scheme of Information-dependent UIO ranking algorithm is shown in Figure 4. The algorithm may affect the USF in two aspects. One is to help the *Object_selecting* function in the mobile agent based crawling algorithm as mentioned before, and another is to order the *Result_information_list* that will be returned by the USF interface.

The basic scheme of the Information-dependent UIO ranking algorithm is shown in Figure 4. The algorithm ranks the UIOs based on their importance and is used in USF in two ways. One is to help the *Object_selecting* function in the mobile agent based crawling algorithm as mentioned before, and another is to order the *Result_information_list* that will be returned by the USF interface.

The importance of an UIO, I(UIO), is defined as:

$$I(UIO)=a_1 \cdot S(UIO,Q)+a_2 \cdot R(UIO)+a_3 \cdot M(UIO,Q)+a_4 \cdot P(UIO)$$

S(), R(), M(), and P() are the importance assignment functions to be explained below. The coefficients a_1, a_2, a_3, a_4 are their adaptable weights.

S(UIO, Q) is the importance function in terms of information similarity. Given a query Q, it assigns a higher importance value to a UIO whose information better matches that requested by Q. Many information similarity algorithms exist and can be adopted for S(UIO, Q) [23]. Here we use a simple function what is based on counting the matched keywords. It is information-type aware, i.e., it takes into consideration the importance factor of different meta data, which is pre-defined and stored in *Utilities*. We assign more weight to keyword matching in *Self_description* than keyword matching in other parts of the meta data, because the information in *Self_description* reflects directly the object itself rather than what being observed by the object. Furthermore, keyword matching for UIO information with *public_RW* or *restricted_W_public_R* privacy has more weight than keyword matching for UIO information with *restricted _R* privacy. This is because information with more strict privacy has less use to the user.

R(UIO) is the importance function in terms of object relationship. It assigns higher importance to a UIO that knows, and is known by, more other UIOs. This is because this kind of UIOs can provide more context information for the user's further navigation or can speed up the crawling process by dispatching more copies of crawlers once. We adopt the page rank algorithm used in web search [16, 24], replacing web pages by UIOs for ranking.

$$R(U_c)=(1-d)+d((R(U_1)/C(U_1)+\cdots+R(U_n)/C(U_n))$$

where $R(U_c)$ is the importance of U_c, the UIO under consideration, $R(U_i)$ is the importance of UIO U_i which knows U_c, $C(U_i)$ is the size of U_i's *In_Set*, and d is a damping factor with a value between 0 and 1.

M(UIO, Q) is the importance function defined in terms of object characteristics. One main characteristic of an UIO is its location. We assign higher importance to a UIO closer to the searching user than that further away. This is because usually the user concerns more about reachable objects.

P(UIO) is the importance function in terms of object popularity. It assigns a higher importance to a UIO being searched more frequently in the history. This information is maintained in *Utilities*.

For crawling process:

UIO-List$_{ranked}$ = UIOs in Out_set;

1. Set a_3=a_4=0
2. Determine coefficient a_2
3. If (similarity-based_crawling), determine coefficient a_1;
4. Rank the UIOs in UIO-List$_{ranked}$ by I(UIO)

For results ordering:

1. Determine the weight coefficients of I(UIO) where is a_1 is assigned much more weight than other co-efficients.
2. Rank the UIOs by I(UIO) and put the results to *Result_information_list*

Fgure 4. Information-depended UIO ranking algorithm

4. Prototype and Simulation

To demonstrate the idea of ubiquitous searching we have developed a prototype. Although, the underlying support for ubiquitous searching can use various kinds of devices and networks, for a proof-of-concept prototype, we used a wireless sensor network. The hardware used in our prototype is shown in Figure 5. Following the design, the prototype also uses a two-tier architecture. In the first tier, we use Berkeley's Motes [17, 18] as UIOs, and in the second tier, we use our custom-made TFAD-901 node as master UIOs which has higher capability for processing and communication.

(a) Berkeley's Motes (b) TFAD-901

Figure 5. Hardware components of our prototype system

For prototype implementation, the *Self_description* of the UIOs are pre-defined and stored in the EEPROM of Motes. The values in the *Ability_description* are the sensed data, and are stored in a flash. There are three kinds of "normal sensors": temperature, light, and sound, and a "location sensor" that senses its own location by using the RSSI technique [11]. The *Out_set* and *in_Set* of *Relationship_description* are maintained as a neighbor list of the Motes. The Motes are always listening to the beacon signal of each other. If any change occurs, a Mote will refresh its neighbor list. The protocol used for the communication between UIOs is IEEE 802.15.14, while the protocol for the communication between the master UIOs is AODV.

We use a PC as the system for interfacing with the user. Apache Tomcat is used as the web-sever to get the search instruction from and post the result back to the user. The request is wrapped into a message and sent to the serial port, and then to the TFAD-901 node to initiate a searching process. First, the request message will be unwrapped, and *Validation* information will be used to acquire a privacy role. Key words will be used to perform mapping. Then, the TFAD-901 either performs a Common search, maping the key words using index, and returning ranked UIO information to the client–end, or initializes a new mobile agent to do the Fresh-based search.

The part of the mobile agent crawling algorithm running on the TFAD-901 node is programmed using C, while the part running on the Motes is programmed using nesC compatible with the TinyOS operating system.

For a demonstration, the following application scenario is used. Jack wants to search a black dog by using USF. He enters the key words (Dog, Black) and his identity (name, password), and selects the common search process. Results are shown in Figure 6. Jack can then further click the related objects found, e.g., the room or the cat, if he wants. But for privacy reason he can't browses detailed information of Marry's dog. We used the Linux file system to store the UIO information and the index information.

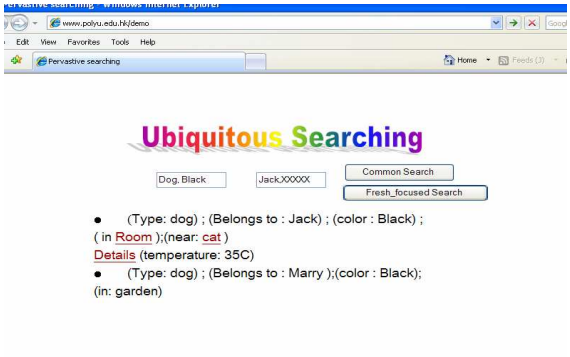

Figure 6. USF searching result demo

The main purpose of the prototype is to demonstrate our idea. However, due to the limited scale, it cannot be used to effectively evaluate the performance of the proposed algorithms. Therefore, we also conducted simulations.

The simulation program is written in Java. We simulate the UIO crawling process in one subsystem. We initialized 20000 UIOs in a 100m*100m area. Among them, 80% UIOs are static and random distributed in the area, while 20% UIOs are mobile using the mobile model of random walk, at the speed of 0.01m/s. The relationship between the UIOs is the spatial "near" – one UIO is in the pre-defined range of another UIO. Range is defined as a circle area with a radius randomly selected from (0, 10). All the UIOs have the ability of getting their own locations (coordinate (x,y)), and the master UIO is placed in the middle of the area. For mobile UIOs, the reporting interval is set according to the moving speed.

We define useful UIOs as those whose importance is higher than the pre-defined threshold. The metric for evaluating the performance of the crawling algorithm is the crawling efficiency defined by K_u/H, where K_u is the number of useful UIOs that have been crawled, and H is the total number of useful UIOs.

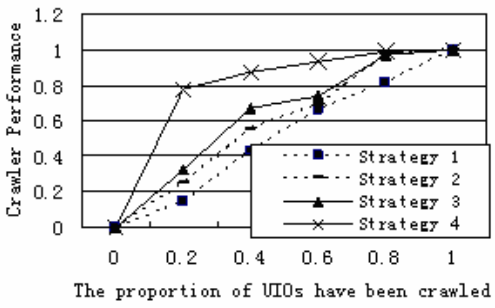

Figure 7. Performance of crawling algorithm

We conducted four experiments, corresponding to the four different crawling strategies: 1) without ranking, and using the same information refresh; 2) without ranking, and using discriminatory information refresh; 3) using ranking with the same information refresh; 4) using ranking with discriminatory information refresh. The simulation results for the four different cases are shown in Figure 7. We can see that, for each case, as the crawling portion of the UIOs increases, the crawling efficiency is also increased. Comparing the different strategies, we can observe that, crawling using ranking will increase the crawling efficiency. On the other hand, using discriminatory information refresh according to the UIO information type will also help improve the performance.

5. Conclusions and Future work

It has been believed that with the continuous advances in the supporting technologies, our living environment is being transformed into a ubiquitous computing platform. Acquiring desired information directly from the physical world will be an exciting application. In this paper, we have proposed a novel concept called "Ubiquitous Searching", and presented the USF framework for realizing the concept. We have also described a prototype using Crossbow Micaz nodes and our tailor-made TFAD-901 nodes.

In the future, we will enhance our work in several aspects. First, we will further develop the prototype as a test bed for the framework and the applications. More kinds of sensing devices as heterogonous information sources, such as RFID and mobile cell phone with sensors, will be considered. Second, we will do more evaluation and improve the key algorithms in terms of the time-delay and information accuracy, the users' satisfactory of the ranking etc. Third, presently, the ubiquitous searching is based on syntax matching. We will investigate the semantic ubiquitous searching. The feature level data fusion and in-network data mining techniques will be studied.

There are some other works that can be done on this topic. For examples, how to use the social networking theory to model the behaviors of UIOs, so as to optimize the data gathering, and how to crawl hidden UIOs, etc.

Acknowledgement

This work is supported by Hong Kong Polytechnic University under the ICRG grant G-YE57 and the large equipment fund G.61.27.D01B

Reference

[1] Cecilia Mascolo, Stephen Hailes, "Survey of Middleware for Networked Embedded Systems", FP6 IP "RUNES" - D5.1

[2] I.F. Akyildiz, et al., "Wireless Sensor Networks: A Survey", *Computer Networks*, Vol.38, 2002, pp. 393-422.

[3] Licia Vapra, Wolfgang, Emmerich, "Middleware for Mobile Computing" Department of Computer Science University College London.

[4] D. Estrin, R. Govindan, J. Heidemann (Editors):"Embedding the Internet", Communications of the ACM, Volume 43, Number 5. May 2000.

[5] S.R. Madden, M.J. Franklin, and J.M. Hellerstein, "TinyDB: An Acquisitional Query Processing System for Sensor Networks," ACM Trans. Database Systems, vol. 30, no. 1, 2005, pp. 122–173.

[6] P. Bonnet, J. Gehrke, and P. Seshadri, "Towards Sensor Database Systems," Proc. 2nd Int'l Conf. Mobile Data Management, 2001, pp 314–810.

[7] Bonnet P, Gehrke J, Seshadri, "Querying the Physical World", IEEE personal Communications, 2000

[8] Boulis, C.C. Han, and M. B. Srivastava. "Design and Implementation of a Framework for Programmable and Effi-cient Sensor Networks". In MobiSys 2003

[9] KokKiong Yap, Vikram Srinivasan and Mehul "Motani, MAX: HumanCentric Search of the Physical World" SenSys'05, November, 2005, USA.

[10] P. Bergamo and G. Mazzini, "Localization in sensor networks with fading and mobility," in IEEE PIMRC, September 2002, pp. 750

[11] J. Hightower and G. Borriello, "Location systems for ubiquitous computing", Computer, vol. 34, pp. 57-66, August 2001.

[12] Nickerson BG, Sun Z, Arp JP (2005) "A Sensor Web Language for Mesh Architectures". 3rd Annual Communication Networks and Services Research Conference, May 16-18, 2005, Halifax, Canada.

[13] Tao V, Liang SHL, Croitoru A, Haider Z, Wang C, "GeoSWIFT: Open Geospatial Sensing Services for Sensor Web". In: Stefanidis A, Nittel S (eds), CRC Press, pp.267-274. 2004.

[14] Reichardt M (2005) Sensor Web Enablement: "An OGC White Paper". Open Geospatial Consortium (OCG), Inc.

[15] http://www.wavetrend.net/

[16] Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd. "The PageRank Citation Ranking: Bringing Order to the Web".

[17] http://www.xbow.com

[18] http://www.tinyos.net

[19] Paul Castro, Richard Muntz "An adaptive approach to indexing pervasive data", MobiDE 2001.

[20] Gabriella Castelli, Alberto Rosi, Marco Mamei, Franco Zambonelli, "A Simple Model and Infrastructure for Context-aware Browsing of World", in the proceeding of Pervasive computing.

[21] G. Castelli, A. Rosi, M. Mamei, F. Zambonelli, "Browsing the World: Briding Pervasive Computing and the Web", 2nd International Workshop on Ubiquitous Geographical Information Systems, Munster (D), 2006.

[22] Elizabeth F.Churchill, Christine A. Halverson, "Social Networks and Social Networking", IEEE internet computing.

[23] Vikram Srinivasan, Mehul Motani, Wei Tsang Ooi, "Analysis and implications of Student Contact Patterns Derived from Campus Schedules" Proceedings of MobiCom'06, September 23-26,2006, Los Angeles, California, USA.

[24] J.Cho, H.Garcia-Molina, L.Page. "Efficient Crawling Through UR: Ordering". Seventh International Web Conference (WWW 98). Brisbane, Australia, April 14-18,1998