GPS-based Location Extraction and Presence Management for Mobile Instant Messenger[1]

Dexter H. Hu and Cho-Li Wang

Department of Computer Science, The University of Hong Kong,
Pokfulam Road, Hong Kong
{hyhu,clwang}@cs.hku.hk

Abstract. Location is the most essential presence information for mobile users. In this paper, we present an improved time-based clustering technique for extracting significant locations from GPS data stream. This new location extraction mechanism is incorporated with Google Maps for realizing *cooperative place annotation* on *mobile instant messengers* (MIM). To enhance the context-awareness of the MIM system, we further develop an *ontology-based* presence model for inferring the location clues of IM buddies. The GPS-based location extraction algorithm has been implemented on a Smartphone and evaluated using a real-life GPS trace. We show that the proposed clustering algorithm can achieve more accurate results as it considers the time interval of intermittent location revisits. The incorporation of location information with the high-level contexts, such as mobile user's current activity and their social relationship, can achieve more responsive and accurate presence update.

1 Introduction

Instant Messenger (IM), characterized by its instantaneous message delivery and presence awareness, has become an important part of our everyday life. With the advancement of cellular technology (*e.g.*, GPRS), it has become possible to get the instant messenger services through mobile phones. As the mobile IM users could potentially move from place to place, location information becomes the most essential contextual cue among the mobile IM users. Location awareness among the communicating buddies makes collaboration more effective in field-work or on-site business as the communication attempts can be initiated without being "blind" or intrusive.

In recent years, owing to the low-cost and lightweight Global Positioning System (GPS) solutions, more and more mobile devices are equipped with GPS functionality. Geospatial data can be obtained by mobile users in real time. To make the GPS-based location service be truly useful for the mobile IM users, we address three issues in this research. First, the solution must be able to filter out useless GPS location data on the fly, as user's mobility is highly dynamic and

[1] This work is supported by National Natural Science Foundation of China (NSFC) Grant No. 60533040.

evolving. Moreover, the GPS-based location service should have an accurate and efficient location model to extract most of the places that users deem important in real life.

Second, after detecting the significant locations, raw location data (longitude, latitude, etc.) should be translated to more symbolic, personally meaningful place annotations. With the location extraction unit built on the mobile IM, all IM users become the location information providers. Users potentially can acquire location information from their buddies through relating new locations to some existing annotations. As thus, mobile users can rapidly access spatial information through such location knowledge sharing. So far this feature is not available in most mobile IM solutions.

Third, most instant messengers support presence management which checks the presence of the user's buddies (*e.g.*, "busy", "off line", "away") and provides a visual indication of each buddy's presence status on the IM client. Location awareness could potentially enable the development of more advanced presence management scheme for MIM. For example, the presence management in MIM should further consider what sort of presence information should be available to whom, and under which circumstances according to the given location information. It also requires to incorporate other context information (*e.g.*, activity status, people's relationship) to infer each buddy's presence status.

In this paper, we present a *mobile instant messenger* (MIM) with three new features: (1) *Improved location extraction algorithm*, an on-line clustering algorithm to extract significant locations more accurately from raw GPS data. We assume there is no location knowledge (*e.g.*, GPS trace) priori the execution of the clustering as the location service is usually needed in a new or partial familiar environment where not every place or path is known. (2) *Cooperative place annotation*. Google Maps is integrated to allow mobile users to share place markers among IM buddies for creating his/her personal map. (3) *Context-aware presence management.* Web Ontology Language (OWL) [3] is used to model buddies' relationship, locations, and activity for automatic presence management. We rely on public Web services like Google Calendar as sources of user contexts.

The rest of this paper is organized as follows. Section 2 explains the *i-Cluster* location extraction algorithm and cooperative place annotation. Section 3 discusses the context-aware presence management in MIM. Section 4 highlights the design of the MIM system. Section 5 reports the implementation details of of MIM and evaluation of its features. Conclusions are discussed in Section 6.

2 Location Extraction and Place Annotation

Identifying significant locations from user's trace is basically a clustering problem [4]. In the past, various location extraction solutions have been proposed based on different sources, such as GPS coordinates [2], GSM cell transition data [5], and Wi-Fi (or Bluetooth) beacons [1]. Intuitively, significant places are usually location visits having a recognizable duration [1] [2]. In some cases, we are also interested in places which may not have a long stay duration, but are revisited

shortly [5] [6]. This type of places include entrance of a parking lot, main gate of a university, junctions of street, etc. There are also situations, where a user's on-going task is disrupted unexpectedly and the user returns to the same place shortly afterwards to finish the task. These are all strong indicators of meaningful locations. We propose an improved time-based clustering algorithm (named *i-Cluster*), which can further extract these types of places.

The original time-based clustering algorithm [1] (called *TBC* afterwards) determines significant places where the user stays longer than a given time threshold t. A new run of clustering is started when distance between the new location and the centroid of the current cluster is larger than a threshold d. In general, it is difficult to tune the two parameters in order to extract all significant places aforementioned in real life.

Our algorithm takes additional consideration to the junction area of user's trace. We introduce a third parameter t_{intv} and use an auxiliary data structure *Tempplaces*. *Tempplaces* keeps track of those visited places with a duration of stay less than t, which are temporarily not qualified as significant places by the TBC algorithm. t_{intv} is a given threshold value that specifies the tolerable time interval of intermittent location revisits. Two temporary clusters in *Tempplaces* will be merged if user moves away from a cluster and returns within t_{intv} time.

The pseudo code of the *i-Cluster* is shown in Algorithm 1. We follow the same definition of parameter d and t as in the TBC algorithm. There are additional variables used in *i-Cluster*. The input to *i-Cluster* is *loc*, which is the new reading of GPS location data. *cl* is the current cluster, which records the centroid coordinate, first timestamp, last timestamp, and the size (number of GPS points) of the cluster. *Places* is used to record the extracted significant places. Function *Distance*() calculates the distance from a given point to the centroid of a cluster. Function *Duration*() measures the time duration of a user staying in the clustered area. To cater for the GPS positioning error and make sure the user is really moving away, *plocs* is used to temporarily keep a small number of pending location data. We report departure of user from current cluster if at least l samples are collected in *plocs*.

We explain the *i-Cluster* algorithm as follows. In line 1-3 (also line 29-30), we add the *loc* to current cluster *cl* if its distance to *cl* is shorter than d. *plocs* is cleared whenever *loc* falls within *cl* (line 3, 28). In line 6-7, a significant place is added to *Places* if *cl*'s duration is longer than t. Otherwise, *cl* will be inserted to *Tempplaces* (line 9-25) for potential merge. The merging process scans through *Tempplaces* in reverse time order (line 11), and tries to merge *cl* with a most recent temporary cluster created within t_{intv} time earlier than *cl* (line 13), which satisfies (1) the summation of the duration is no less than t, and (2) the distance between the centroids of the two clusters is no longer than d (line 16). Other temporary clusters beyond the t_{intv} time window are removed (line 22). The time gap (line 13) is the difference of the later cluster's first timestamp and the earlier cluster's last timestamp. After these steps, the algorithm starts a new cluster from *plocs.end* (line 26-28), as the user is moving away. We make sure there are at least l location data received in between two consecutive clusters.

Note that we merge two nearby clusters found in $Tempplaces$ by measuring the distance between the centroids of them. We set the merge distance threshold d (line 16) instead of a small one (*e.g.*, 2 meters), as the centroid of cluster does not timely reflect the current position of the user. User's current position may in fact be very close to the centroid of the cluster to be merged with the current cluster, while the distance between the two centroids is still in a distance of d.

Algorithm 1 *i-Cluster* (loc)

```
1: if Distance(cl, loc) < d then
2:     add loc to cl {/*Add the new data to current cluster if it's within distance range*/}
3:     clear plocs
4: else
5:     if plocs.length > l then
6:         if Duration(cl) > t then
7:             add cl to Places {/*A significant place found*/}
8:         else
9:             merged ⇐ false {/*Add the temporary cluster to Tempplaces for potential merge*/}
10:            add cl to the end of Tempplaces
11:            for j = Size(Tempplaces) − 2 to 0 do
12:                tc ⇐ jth cluster in Tempplaces
13:                if (Firsttimestamp(cl) − Lasttimestamp(tc)) < t_intv then
14:                    dist ⇐ Distance(tc, cl_centroid)
15:                    sum ⇐ Duration(cl) + Duration(tc)
16:                    if dist ≤ d and sum ≥ t and merged = false then
17:                        merge cl, tc to a single cluster added to Places
18:                        remove cl, tc from Tempplaces
19:                        merged ⇐ true
20:                    end if
21:                else
22:                    remove tc from Tempplaces
23:                end if
24:            end for
25:        end if
26:        clear cl
27:        add plocs.end to cl
28:        clear plocs
29:        if Distance(cl, loc) < d then
30:            add loc to cl
31:        else
32:            add loc to plocs
33:        end if
34:    else
35:        add loc to plocs
36:    end if
37: end if
```

The *i-Cluster* algorithm has several merits. First it is space-efficient as we do not keep the GPS data belonged to a cluster. Besides, the memory size of $Tempplaces$ is bounded by the intermittent time value t_{intv} and the average speed v of user, since we only keep clusters within a time window of t_{intv}. In a worst case when the user keeps moving, the expected number of clusters nc in $Tempplaces$ can be estimated as:

$$nc = \frac{t_{intv}}{\frac{2d}{v}} = \frac{t_{intv}v}{2d} \tag{1}$$

In a typical case, where $d = 40$ meters, $t_{intv} = 1200$ seconds (20 minutes), $v = 5$ km/h (the average walking speed of pedestrians), nc is roughly 20. So the space overhead induced by *i-Cluster* algorithm is not large, and the time complexity of merging clusters $O(nc)$ is thereby tolerable in resource-restricted mobile devices. As the cluster merging step is quite efficient, the performance of *i-Cluster* is as good as the TBC algorithm.

After detecting the significant locations, raw location data (longitude, latitude, etc.) should be converted to meaningful place labels. We implement a place annotation function similar to those used Google Maps [7] and GeoNote [8] on our MIM. Users can either enter their annotation (i.e., "points of interest") manually, then upload to a central map server, or they can select a place label among those created by their buddies whoever visited the same place before. The place annotation selection is usually subjective and it depends on user's focus of interest on the spot. We call this *cooperative place annotation*. More details are discussed in Section 4.

3 Context-aware Presence Management

The co-awareness of buddy's location context can avoid unnecessary conversations (*e.g.*, "Where're you?", "What're you doing now?"), and achieve more efficient collaboration. For example, we can schedule a meeting at a place where each person is most close to, and when every member is available.

We further consider the *connectedness implications of presence* [9] in terms of MIM user's social networks and community. Given the useful location annotations generated by all users, the presence management subsystem determines what sort of location data be revealed to user's buddies. In our design, user's presence is represented by the triple "Status:Activity@Place". The subsystem will derive customized location indicators according to the activity they are currently involved and the social relationship between users and their buddies (or their roles in a group activity). In other words, a user's physical location is interpreted differently and different place labels could be displayed on his/her buddies' mobile phones.

We use the Web Ontology Language (OWL) [3] to model buddy relationship and domain knowledge involved in IM communications. By inferring on the ontology-based framework, we update buddies' presence in the buddylist accordingly. This method also helps to rank location recommendations while performing the cooperative place annotation. For example, the location recommendations by buddies to participate the coming group activity could be ranked higher.

Figure 1 shows the ontology model used in our MIM. The *presence* ontology is incorporated with *location*, *activity*, and *status* ontologies. The *hasPeoplerelation* property of *user* captures the social relationship between people. Currently, we define three sub-properties to reflect the common buddy relationship, including family, colleagues, and friends. These properties appear as *hasFamilyRelation*, *hasWorkRelation*, and *hasFriendRelation* in the ontology model.

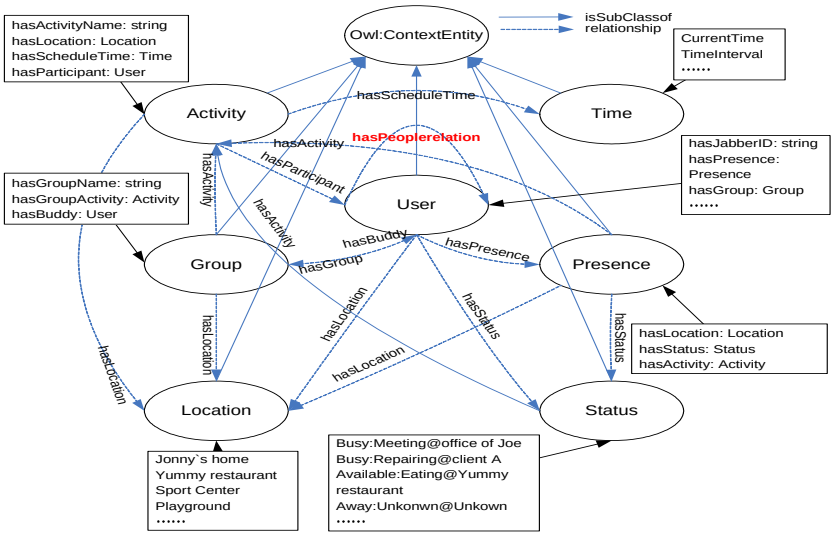


Fig. 1. Diagrammatic view of the MIM ontology model.

The buddylist on the client is automatically refreshed whenever any salient context changes happen to buddies. The process is divided into three steps:

1. Decide the current activity of the user (if any), which is triggered by location update of *i-Cluster*, starting of a scheduled event's time, etc.
2. Determine user's new presence to buddies based on the rules defined.
3. Generate each buddy's new buddylist pushed to client according to the update priority: (1) the buddylist showing members involved in the current activity, (2) the buddylist recording people nearby user's current location, and (3) buddy relationship.

4 The MIM System Design

The design of MIM is extended from *Smart Instant Messenger* (SIM) system [11], which was developed atop of the Jabber IM platform [13]. Figure 2 shows the MIM system architecture and communications between each component. The MIM client communicates with other buddies via the Jabber Server using the instant messenger protocols *XMPP*. It performs *i-Cluster* for detecting significant places as a background task on a GPS-connected mobile phone. It also handles the map download, rendering, and display of various types of buddylist. These are related to the GUI design.

On the server side, the Jabber server serves as a gateway to mediate the communications between MIM client and other server-side components. There is a *packet listener* in the Jabber server to parse the *XMPP* packet type, and activate the corresponding back-end service. There are four other components

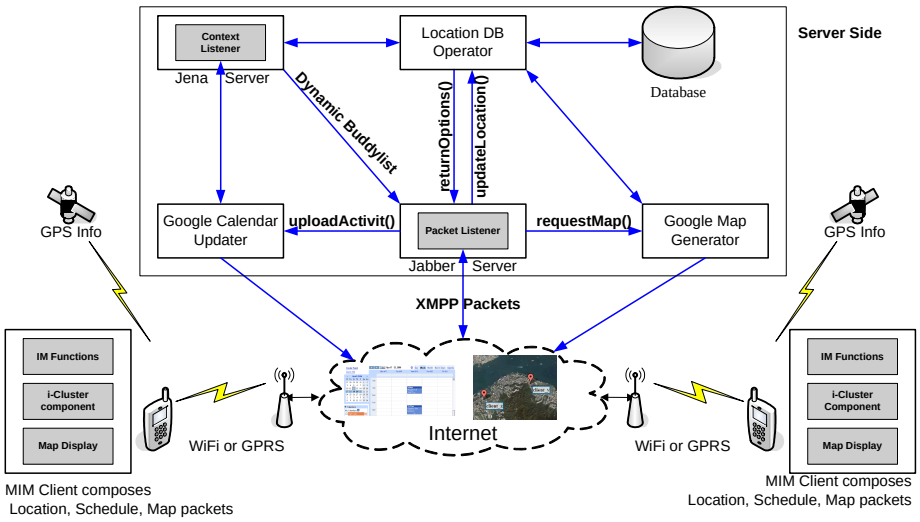


Fig. 2. Overview of the MIM System.

in the server side. Among them, the Jena server is the most complicated one. It realizes the context-aware presence management scheme based on the new ontology model. The Jena server employs a *context listener* to monitor each buddy's location changes and calendar events. Based on these high-level contexts, Jena server infers the current activities of all users. Upon detecting a location change from a MIM client, Jena server determines the "Status:Activity@Place" triple to be sent to his/her buddies and when to deliver. The decision of location annotation and delivery time is user-dependent as it is based on the relationship between the user and his buddy, and the activity they are currently involved. To facilitate calendar entry creation and event query, the *Google calendar updater* is added.

The main function of the *location database operator* (LDBO) is to perform insertion, deletion, and query operations on the location database. LDBO stores various annotated location information produced by users through *i-Cluster* component in the MIM client. It records user's id registered in Jabber server, place's GPS coordinates and its semantic label, creation time of place data, and *hit number* which is the number of times a place's semantic label ever chosen by others MIM users. LDBO can reply queries with location recommendations that satisfy a given distance criterion or time range. It also provides the initial ranking of the location recommendations according to the hit number. These location recommendations will be further analyzed by the Jena server to derive more accurate recommendations based on other reasoning rules discussed in the previous section. To deal with the map generation, a *Google map generator* is designed to resize the map images and add location markers. It also caches the map images downloaded from Google Maps.

For supporting communication between MIM client and various server-side components, the XMPP protocol of Jabber is extended by defining three new types of *custom* packets [13]: *location*, *google-calendar-event* (*schedule*), *google-map-request* (*map*).

When MIM client detects that a user has stayed at a place for a short period of time, it automatically sends a location request to Jabber server. The request packet containing the latitude and longitude of user's current location is first handled by the Jabber server, which parses the type of the XMPP packet and forwards it to LDBO. LDBO will reply with the requested place's semantic labels filtered by Jena server. In case it is a new location (*i.e.*, no recommendation available), or the user decides to annotate himself (*i.e.*, rejecting all recommendations), the *i-Cluster* is resumed. Once it detects a significant place, it alerts user to enter a semantic location label. The location information is then sent to the Jabber server with a *google-calendar-event* packet containing centroid's GPS coordinates, starting time, ending time, user id, and the added semantic location label.

In case the received XMPP packet is a *map* packet, for requesting location information of other user(s) on the buddylist, the Jabber server will forward the request to the Google Map generator to create maps with location indication of buddies. It can either reply to client with an online map URL or transfer the converted map image with place markers.

5 Implementation and Evaluation

5.1 MIM Client and Map Display

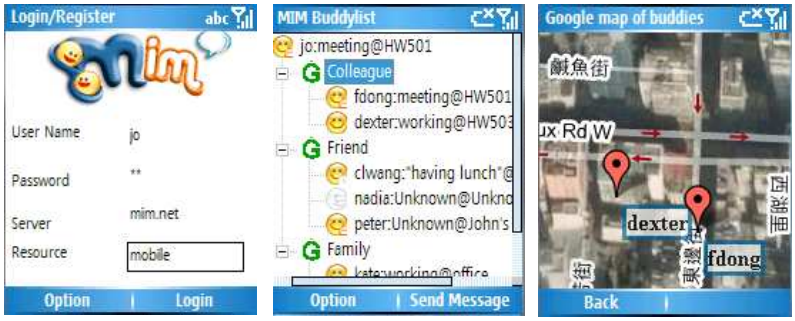
We implemented two versions of MIM client on a Dopod C720W Smartphone running Windows Mobile 5.0 operating system in both C# and Java in the J2ME (MIDP 2.0) platform.

Figure 3 (a) shows the GUI of MIM client produced from Windows Mobile 5.0 Smartphone emulator. Figure 3 (b) shows the presence of Jo's friends, colleagues, families, including status icon, activity and location. Note that this test used the RFID reader to detect users' indoor location. The activity information was retrieved from the Google Calendar. Upon selecting a group/buddy, the user can choose to fetch and view google map image with location makers of the buddy/group members. The image size parameters sent to MIM server is determined by the screen size of the smartphone.

5.2 Evaluation of *i-Cluster* Algorithm

In this experiment, the Smartphone reads GPS data from a Holux GPSlim236 GPS receiver [14] via Bluetooth connection, as the C720W Smartphone does not have a built-in GPS receiver.

A sample user trace was collected by walking around the Sai Wan area of Hong Kong Island for a total time of 2.6 hours. The GPS receiver reports GPS



(a) MIM Login GUI (b) Buddylist of Jo (c)Map with buddy locations

Fig. 3. MIM Client GUI and Map Display

position reading one per second. A total of 9373 GPS data points are collected. The MIM client only made use of longitude, latitude, and timestamp of the GPS position reading. Figure 4 (a) shows the area the mobile user has visited.

We set the parameters of the *i-Cluster* algorithm as $d = 40$ meters, $t = 300$ seconds, $t_{intv} = 1200$ seconds, and $l = 10$. The values of d and t are determined according to the knee point in [1]. We set $l = 10$ for keeping the pending locations. The value of t_{intv} is set to 20 minutes as we believe when an important task is disrupted unexpectedly, the user usually will come back to the same place soon. To evaluate the accuracy of the *i-Cluster* algorithm, we disable the stationary detection function, *i.e.*, the MIM client is not allowed to query location recommendation from MIM server.

Figure 4 (a) shows the seven extracted significant places by *i-Cluster*. They are plotted by the GPS visualizer [15] as *waypoints* in Google Maps format. We found the reported locations are very close to the places we visited by viewing the street map.

We further investigate those clusters detected by *i-Cluster*. Figure 4 (b) shows the GPS data points nearby place b. Figure 4 (c) shows the centroids of the corresponding clusters in the $Tempplaces$. They are numbered from 1 to 14 in time order. In the experiment, we stayed at the 7-Eleven convenience store shortly (about 3.2 minutes), left for an ATM machine, then came back again through another block after 6.5 minutes, and stayed another 3.4 minutes at the store. As shown, cluster 4 and cluster 10 are merged into a single cluster as place b (the 7-Eleven convenience store in Figure 4 (a)), which would be simply ignored if TBC algorithm is used.

5.3 Presence Reasoning Logic

We use Jena's Rule Java object [12] to define reasoning rules for determining user's new presence. The basic rules are in a human readable form of "*antecedent* => *consequent*".

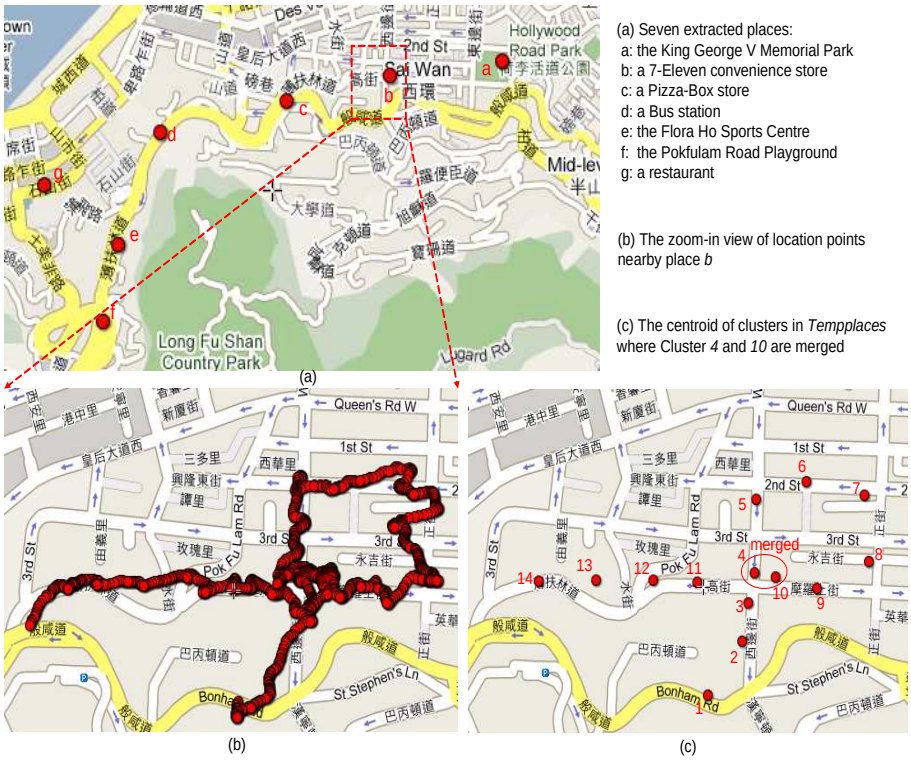


Fig. 4. The experiment shown in Google Maps format

Table 1. Basic rules for inferring user's presence.

Cases	Antecedents	Consequents
Determine the presence shown to a colleague	hasTime(CurrentTime, "Work time") hasWorkRelation(?x1, ?x2) hasSameActiviy(?x1, ?x2)	hasAvailableStatus(?x1, ?x2) hasSameGroup(?x1,?x2)
Determine the presence shown to family member	hasTime(CurrentTime, "Work time") hasFamiliyRelation(?x1, ?x2)	hasBusyStatus(?x1,?x2) hasActivityHidden(?x1,?x2) hasLocationShown(?x1,?x2)
Determine the presence shown to friend	hasTime(CurrentTime, "Work time") hasFriendRelation(?x1, ?x2)	hasAwayStatus(?x1,?x2) hasActivityHidden(?x1,?x2) hasLocationHidden(?x1,?x2)
Determine the presence shown to colleague when it's off-duty	hasTime(CurrentTime, "Off-duty") hasWorkRelation(?x1, ?x2)	hasAwayStatus(?x1,?x2) hasActivityHidden(?x1,?x2) hasLocationHidden(?x1,?x2)
Determine the presence shown to friends when it's off-duty	hasTime(CurrentTime, "Off-duty") hasFriendRelation(?x1, ?x2)	hasAvailableStatus(?x1,?x2) hasActivityShown(?x1,?x2) hasLocationShown(?x1,?x2)
Determine the presence shown to family member when it's off-duty	hasTime(CurrentTime, "Off-duty") hasFamilyRelation(?x1, ?x2)	hasAvailableStatus(?x1,?x2) hasActivityShown(?x1,?x2) hasLocationShown(?x1,?x2)

Table 1 shows the rules we used in MIM for automatic update of user's presence. The first three rules infer user's availability to his co-workers during work time, but avoid exposure to friends and family members. The last three show user's presence to his friends and families, whereas hide it from colleagues to avoid disturbance during his spare time.

We evaluated the responsiveness of MIM presence model performed by the Jena server. In this evaluation, the MIM client was connected to a desktop server via Wi-Fi connection. We found the average processing time for a typical reasoning on context changes is around 2-4 seconds. We will investigate more time-efficient reasoning solutions in the future.

6 Conclusion and Future Work

In this paper, we present a GPS-based location extraction system to support MIM. We designed the *i-Cluster* algorithm to better locate the significant places that would be ignored by previous time-based clustering algorithms. We found the cooperative place annotation scheme can greatly reduce the computing cost of the GPS-based location extraction method and produce more accurate location information. With the wide adoption of low-cost GPS receiver built on mobile devices, the co-awareness of the location information has a good potential to develop more powerful context-aware applications. We modeled IM-related concepts and people's relationship using ontologies to automate the presence inference. We believe this is an emerging trend in the development of IM. In the future, we shall further extend our MIM presence model and study more intelligent and faster reasoning solutions.

Lastly, we found the powerful Web services have made it very convenient to build the proposed MIM functions. We suggest that the location-aware presence management should make use of public Web services like Google Maps and Google Calendar. The "attachment" of user's context information to the public Web with more reliable and stable services is a viable solution to realize "pervasive" context-aware computing, especially for mobile users.

References

1. J. H. Kang, W. Welbourne, B. Stewart, and G. Borriello. Extracting Places from Traces of Locations. In Proc. WMASH, pages 110–118, New York, NY, USA, 2004. ACM Press.
2. D. Ashbrook and T. Starner. Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users. In Personal and Ubiquitous Computing, 7(5):275–286, October 2003.
3. OWL Web Ontology Language. `http://www.w3.org/TR/owl-features`
4. H. Zha, C. Ding, M. Gu, X. He and H.D. Simon. Spectral Relaxation for K-means Clustering. In Neural Information Processing Systems vol.14. pp. 1057-1064, Vancouver, Canada. Dec. 2001.

5. P. Nurmi, J. Koolwaaij. Identifying Meaningful Locations. In The 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services, 17-21 Jul 2006, San Jose, CA.
6. F. Schmid, K. F. Richter. Extracting Places from Location Data Streams. In A. Zipf (Eds.), Workshop Proceedings (UbiGIS), Münster, Germany.
7. Google Maps API. http://www.google.com/apis/maps
8. F. Espinoza, P. Persson, A. Sandin, H. Nyström, E. Cacciatore and M. Bylund. GeoNotes: Social and Navigational Aspects of Location-Based Information Systems. In Ubicomp 2001: Ubiquitous Computing, International Conference Atlanta, Georgia, September 30 - October 2, Berlin: Springer, p. 2-17.
9. R. Rettie. Connectedness, Awareness and Social Presence. In Proc. PRESENCE 2003, online proceedings.
10. Google Calendar Data API. http://code.google.com/apis/calendar/overview.html
11. C. F. Law, X. Zhang, M. S. M. Chan, C. L. Wang. Smart Instant Messenger in Pervasive Computing Environments. In The First International Conference on Grid and Pervasive Computing, May 3-5, 2006, Taichung City, Taiwan.
12. Jena: a Semantic Web Framework for Java. http://jena.sourceforge.net
13. Jabber Instant Messenger. http://www.jabber.org
14. GPSlim236 GPS Receiver. http://www.holux-uk.com/Products/gpslim236/index.shtml
15. GPS Visualizer. http://www.gpsvisualizer.com/map