A Mobile State Detector for Complex Social Environments: Applications to Tunelbana Wagons

Nikos Korfiatis and Koray Duhbaci
Interactive Systems Engineering Programme
Department of Computers and Systems Sciences
Royal Institute of Technology (KTH), Stockholm
Stockholm, March 30, 2005

[1] Introduction and Social Background

Very often we have been faced with circumstances where we mix with a crowd of people who share the common environment with us, have a face contact every time but we don't have established a social relationship. Consider for instance the train wagons that we use to see the sometimes the same people for a significant amount of time, we feel that we know each other but we haven't exchanged even a few words. In most of the cases this has to do with behavioral barriers that interfere between us thus not allow us to establish a social relationship. Shyness or circumstantial cases (eg. We feel tired, we are talking with someone else) are also factors that are responsible for such cases. Culture is also something that affects this type of communication (eg. Different societies react different in such cases, eg. The societies in the North and the South of Europe).

On the other hand the public transport (eg the tunelbana or the busses) is something that makes us getting boring when we don't have something to do (eg. Have a mate, reading etc). Most of the times this case has been addressed by mobile phone companies where the games in the Mobile Phone can find a usage. As the trip lasts longer it is usually more boring for the persons to be sitting without having something to do. *In our case we make the assumption that socializing with someone is something that is much effective than self entertainment activities such us playing games with the mobile, listening to music etc.*

What we foresee in our application scenario is a scenario where the social barriers will be dropped and communicating and socializing with the others is not something difficult. If we consider for instance the internet and how easy is to meet someone you don't know before we want to use this metaphor to a pragmatic social communication case. In the following paragraphs we present the main scenario which runs inside a tunelbana wagon for sake of simplicity and deployment of the ***Spot*** and ***Eventheap*** platforms that we used for the development of a proof-of-concept prototype.

[2] Scenarios of the proposed application

"Thomas after a very tough day of work in Erricson in Kista he is returning home in Hasselby Strand. Thomas uses the public transport to go to work so he needs to take the tunelbana everyday from Hasselby Strand to T-Centralen and reverse. He feels boring and he doesn't have his mp3 player to listen to some music or the radio. Instead of this he has his ipaq with the meetwagon application. He opens it and starts beaming available mates in the wagon. Because of the SPOT system he can figure where the available chatting mates are available by using the label on their spot location. He finds that there are other two ladies available for discussion. He uses the eventheap to send a notification to their devices and after some exchange of tokens they decide to seat altogether in an empy sit in the wagon". As the stations change the time passes very fast when he realizes that he is already in the centralen. Now in the trip to Hasselby Strand he will use again his device to meet some other interesting people.

The above scenario represents a successful deployment of our application. In that case the SPOT system is being used to mark and identify the wagon sits or positions. The person can express with a label about his/her status and publish it using the Spot.

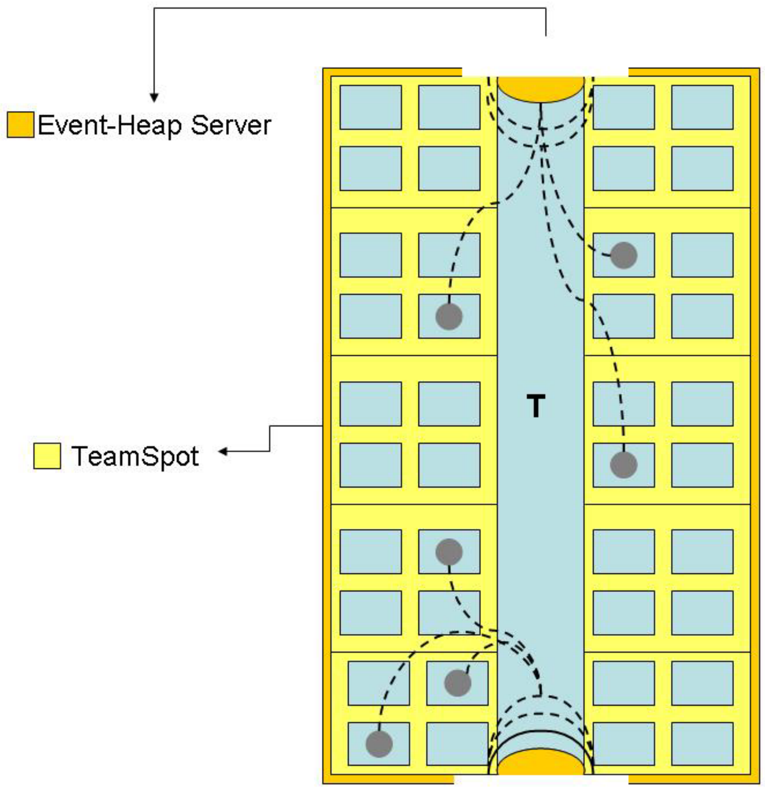


Figure 1: The Spot system is surrounding the wagon providing the web service with the id and the place where each client is deployed. The clients use the eventheap service to notify the others where they want to communicate.

After the sit is labeled with spot each client can access the labels using the provided web service. A search can be done to the label stream in order to identify the people that are available and where they are sitting. After the filtering of the people that have an "accept" status in socializing an event is being sent to their clients to notify them.

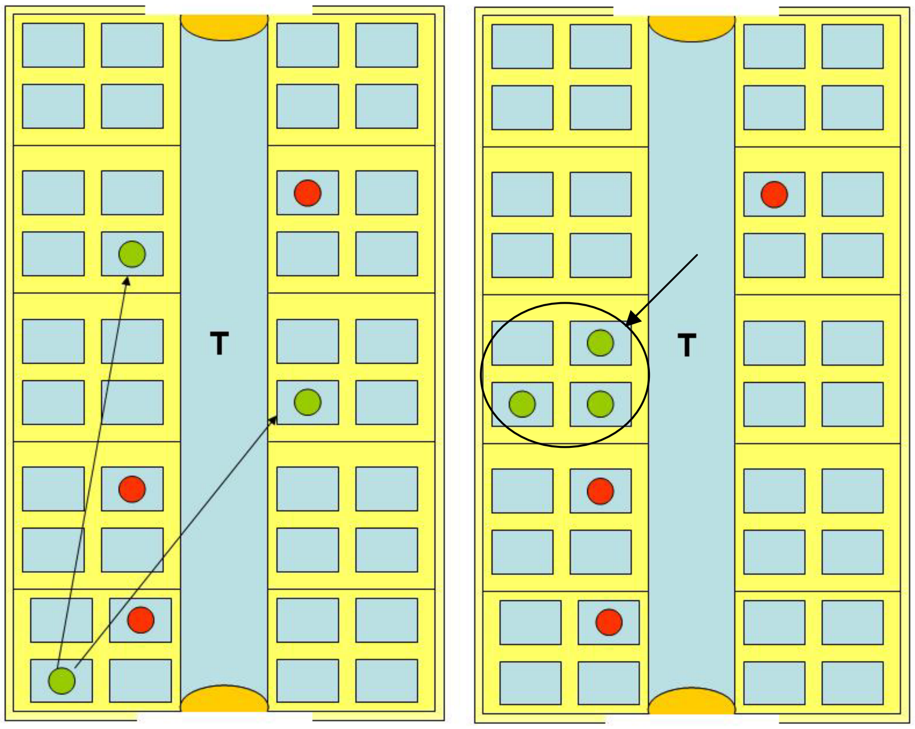


Figure 2: The client sends an event with the EventHeap to the other clients available for discussion (green tags). Although in the system are registered other client not all of them are available for chatting (indicated with the red label). The final stage of the scenario is the interested mates to sit together

After the identification is being arranged the clients can sit together or search for other available mates in the wagon as the passenger change.

[3] Technical Implementation

In order to provide a proof of concept implementation of the aforementioned scenario we used the SPOT and EventHeap systems acordingly

Identifying location context using SPOT

For this purpose we have extended the client.java in order to read information based on the wagon spot. In our architecture each wagon has been labeled before as a spot , identified by the spot client with information such us the sit and the wagon.

After the wagon and the sit is labeled as we change position we receive a stream as follows:

```
<SpotLabel>;labelname=no;label=7518;userid=4;spotid=49;isPublic=false;timest
amp=1109087062594;</SpotLabel>;<SpotLabel>;labelname=Name;label=near the
door not available ,
lt;userid=4;spotid=49;isPublic=false;timestamp=1109087055472;</SpotLabel>;<S
potLabel>;labelname=window sit wagon
availiable;label=something;userid=4;spotid=49;isPublic=false;timestamp=11090
94635010;</SpotLabel>;</SpotProfile>
```

We then use the following method to get information about the available clients and their location.

```
public vector availiableclient ;
public void checkAvailiableMates(String message)
	{
		SpotLabel[] clientsConnected = findlabel(message) ;
		for (int k = 0; k<clientsConnected.length; k++)
		{
			System.out.println("List of availiable clients")
			if("Availiable".indexof(clientsConnected[i]))
			{
				getclientid(clientsConnected[i]) ;
			}
		}
	}

	public void getclientid(String currentMessage)
	{
		Vector = new Vector() ;
		StringTokenizer   tok   =   new   StringTokenizer(currentMessage,
";=");
		while(tok.hasMoreTokens()){
		String s = tok.nextToken();
		if(s.equals("<SpotLabel>")){
			SpotLabel spotlabel = new SpotLabel();
			String str = "";
			while(!(str = tok.nextToken()).equals("</SpotLabel>"))
			{
				if(str.equals("userid")
				{
				  availiableclient.add(tok.nextToken().toString())
;
				}
			}
		}
	  }
	return availiableclient ;
	}
```

After we obtain the list with the available clients we need to notify them if are accepting chatting now.

Notifying Mates using the EventHeap

After we get the array with the available mates we make a multicast type event sending to the available clients for chatting. We access the previously loaded

```
public void notify clients
{
	ListIterator mates = availiableclient.listIterator();
	while (mates.hasNext()) {
try{
      // Connect to EventHeap
      EventHeap myHeap=new EventHeap(args[0]);

      // Create event, set the event type, and add a field
      Event myEvent=new Event();
      myEvent.setEventType("SimpleEvent");
      myEvent.addField("Chatting from client "+availiableclient[k,
String.class, Event.VIRTUAL, Event.FORMAL);

      // Loop getting and printing events
      while(true) {
        Event retEvent=myHeap.waitForEvent(myEvent);
        System.out.println("**\n**\n**\n**\nGot message: " +
                      retEvent.getPostValue("Interest for chatting ?") +
                        "**\n**\n**\n**");
       }

     }
     catch(EventHeapException e) {
       e.printStackTrace();
     }

        }
        catch (ArrayIndexOutOfBoundsException f)
        {
        System.out.println("No value at element");
        statLbl.setText(" error ");
        }
    }
}
```

Each client listens for an event that will be sent from the multicasting client and can respond also.

[4] Further Implementation and Future Directions

One significant disadvantage of our application is that we have to mark each time the seat and the label that we are situated. Also the accuracy of the spot system in the current state is not enough for a small place like the tunelbana wagon. A more professional approach could use Bluetooth to construct an adhoc wireless network and establish a communication chanell between all the clients without demanding a significant infrastructure to be implemented since Bluetooth is available in a wide range of platforms and devices.

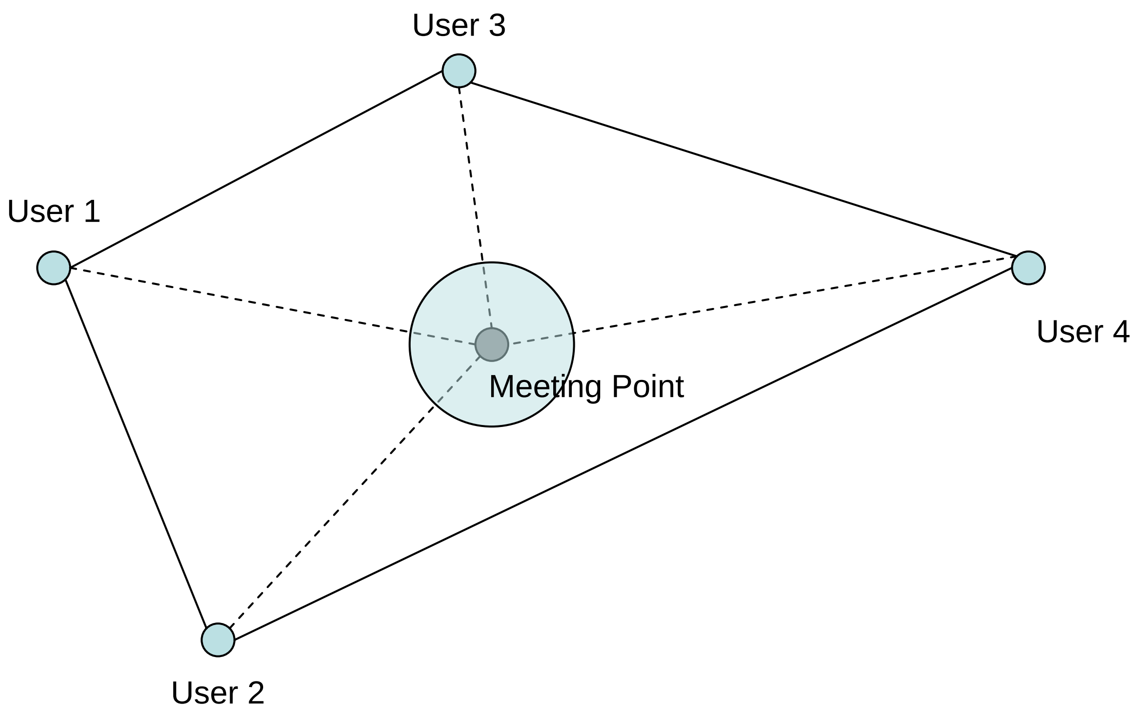
Appendix: Deployment of Alternative Scenarios

Alternate 1

A group of students is going to meet for lunch. Every student is using IPAQ and connected to one of the hotspots in the campus. All students are studying on different buildings on a very big campus. Our application detects each student's current location and suggests a meeting place based upon criteria such as traveling time. So application suggest most efficient meeting place.

We can use Spot application to detect other user's location. OthersLocation.class (provided by Li Wei) helps us to get the latest location of someone and when someone connected to system.

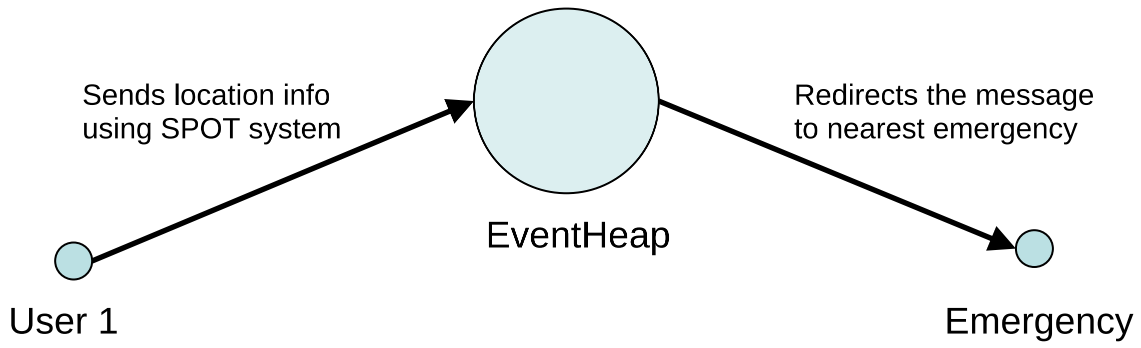
When we get username and locations of the users, we post this information to EventHeap server. We assume that EventHeap has a database to store all spot locations. This database will help us to write a code for calculating distance between two spots. In our scenario there will be more than one user, so server application will calculate the middle point of the shape which is formed by connecting spots of users. This middle point will be the most effective meeting point for users. But we should add a proximity value to extend our meeting points. If there is no place to go for lunch at the point defined by the system, application should offer an alternative place. When meeting point is decided by one user, other participants will receive a message for the meeting place through EventHeap



Alternate 2

Emergency Call is a service that is activated by pressing an emergency button on the application. It establishes a call to the emergency service center and gives the exact location of the user. This will increase the speed and accuracy of the service by avoiding wrong address descriptions and avoiding losing time for telling the address.

Spot application will give the users location and we will post this information to EventHeap to redirect this message to nearest Emergency Server. When Emergency server receives the message, they will also see the exact location of the person who needs help. This communication will be one way, from a person to emergency server.


Sends location info using SPOT system
Redirects the message to nearest emergency
EventHeap
User 1
Emergency