XMPP based Data Abstraction Layer for Smart Phone Sensors

J. Heymendran, and T.N.K De Zoysa

***Abstract*-Recent explosion in smart phone usage has led to sensor applications that take advantage of various sensors and computing power of smart phones. In building such applications architectural wise three things needed to be addressed, they are; a) how to capture data b) how to communicate the captured data c) how to process the captured data. A typical smart phone sensor application may address these questions by following ways; The data capture architecture may involve native smart phone Application Programming Interface (API) code call to capture the data, the communication architecture may based on client-server model where phones collect and post data to a central server or a hybrid client-server model where co-located phones make ad-hoc network connections and then send the aggregated data to the central server and the processing may occur on the central server. Our approach involves, a data capture abstraction layer where we abstract out data capture logic from native API, and provide Structured Query Language (SQL) like language syntax to query data, we utilize the user's social contacts to connect to nodes and to communicate data using chat. The proposed data abstraction layer for smart phone sensors enables nodes to communicate and process data in near real-time without necessity of owning and operating any middleware servers. In this paper we discuss about basic architecture of the system and provide a discussion on prototype implementation.**

Index Terms— Data Abstraction Layer, Peer-to-Peer Sensor Applications, Social Networks

I. INTRODUCTION

Wireless sensor network frameworks such as TinyDB [12] and TikiriDB [10] provide high-level abstraction layer with SQL like query languages to the developer making implementation and deployments of data collection applications much easier to handle. Our research proposes similar, SQL Query like data abstraction layer architecture on top the phone sensors, providing a higher level abstraction and a user-friendly query language for application programmer to transparently access and query sensor data from one device to another device using existing infrastructure and technologies.

A. *Smart Phone Applications with sensor processing*

Smart phones with sensor applications generally monitor or track some smart phone sensor data and then apply processing and finally derive some use full information.

J. Heymendran was with University of Colombo School of Computing, No 35, Reid Avenue, Colombo 7, Sri Lanka (phone: 0722119597; e-mail: janagan.h@gmail.com).

T.N.K De Zoysa is with University of Colombo School of Computing, No 35, Reid Avenue, Colombo 7, Sri Lanka (e-mail: tnk@ucsc.cmb.ac.lk).

Currently there are applications where environmental data is collected from co-located smart phones which are mainly used for scientific studies. In the future we may see applications that collect levels of pollution or noise level or traffic information and even collect weather data, then the collected data may be queried by another phone for some processing on the data. Using smart phone sensors and accessing their data for processing opens up new and interesting application areas. One of the challenges of building such a network is to transparently access and query sensor data among phones. Currently to access data, each phone has to implement custom code to retrieve data.

B. *Social Networks in Smart Phone Sensor Applications*

Since the inception of smart phones and apps, there have been various research and products available that are community based sensor monitoring and reporting applications. These Applications use network of users participating in their application community to collect sensor data and extract interesting information value to the user. For example the Google Latitude application allows user to display their location to certain people.

The WAZE [14] application is community based traffic app, where its collects traffic information from the community and gives advice on the best route.

What these applications have in common as mentioned earlier is they employ client-server model where phones collect and post data to a central server or a hybrid client-server model where co-located phone communicate and post the aggregated value to server.

Our approach to design is to use the already available social network to be our nodes and use their messaging infrastructure to communicate.

C. *Research Objective*

The proposed research outlines a data abstraction layer for phone sensors based on Extensible Messaging and Presence Protocol (XMPP) [13]. We discuss the architecture for the three questions 1) how to capture data 2) How communicate data and 3) How to process data.

II. RELATED WORK

Mobile wireless sensor networks (WSN) such as TinyDB [12], TinkiriDB [10] incorporates the declarative query approach to querying and capturing sensor data. These systems use acquisitional query processing techniques [12] to optimize query processing and communication between the nodes of the network.

Our approach uses similar concept of declarative queries in the context of smart phones.

A smart phone sensor network a system called mTikiriDB [3] approaches the problem, by mapping the WSN architecture to smart phones. The network is a peer-to-peer (P2P) system based on a distributed hash table. The main problem with this approach is, this needs known IP addresses to communicate, one needs to be in the local network to do this. This may well work few dozen phones, all co-located in the same network, but in a global scale this approach won't work. Our approach uses XMPP [13] used in social networking services such GTalk [5] chat for messaging. This approach provides global coverage of nodes in different locations

In the context of smart phone sensor applications there are many papers that have discussed various architectures fulfilling specific needs of a domain. The approach taken by [8], [9] and most systems in general handle data capture by writing custom native code or have declarative querying capabilities, and the communication is based on client-server model where phones collect and post data to a central server or a hybrid client-server model where co located phones make ad-hoc network connections and then sends the aggregated data to the central server for processing. Our approach too involves, a data capture abstraction layer where we abstract out the data capture logic from the native API, and provide SQL like language syntax to query the data. The user's social contacts acts as nodes and they communicate using XMPP [13] chat messages.

III. Architecture

In this section we discuss the design features of the system. The basic idea is that the user's social network becomes the sensor nodes and the social network's messaging system is the communication medium where users query and send the sensor data among participating nodes.

As in Fig. 1 the device and the user accounts in the social network represent a node in the network. Using the social network's messaging system we can design a P2P system that does not need any external public servers or middleware.

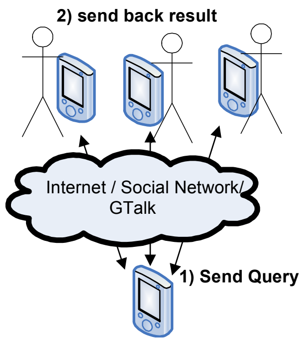


Fig. 1. Overall view of the system

Fig. 1 illustrates system's basic usage scenario. As user types in the query for example "select temp from sensors" and sends it via GTalk [5]. The receiver phone receives, executes the query and returns the results to the sender.

Since there is no central server involved, all the software components are self-contained in the phone. The software components are layered according to their functionality Fig. 2 illustrates the basic component layers of the system. At the bottom the sensors acquire and provide the sensor data. It is then passed onto the query processing system, where it is processed and query operations are run. The processed data is then passed onto the messaging module, this module uses the basic messaging primitives to wrap the processed data and forwards to the social network.

A. Query Processing

This module is responsible for parsing and executing queries. The Fig. 3 shows the main query processing components: The query controller is the interface that other sub modules interact with and also it is responsible for controlling and coordinating its sub components. The other subcomponents help in parsing, executing and periodically running the queries.

When a new query comes in, it is first sent to the query controller. The query controller synchronizes and executes the query requests sequentially. A query is first parsed by the query parser which checks for any syntactic errors in the query and also creates and initializes data structures needed for query processing. The query processing module is responsible for applying the query operators and processing the data. It takes the data from the data collection subcomponent, processes it, and returns the resultant data to the query controller. The data collection subcomponent is responsible for acquiring the sensor data from the native sensor APIs. The query daemon is a background process that periodically runs the query processing subcomponent. The query controller finally sends resultant data from query processing to the messaging component.

We use the following SQL query structure to specify queries.

1. Query Structure:

SELECT {attributes}
FROM [sensors] [WHERE {<expr>}]
[ACTION_INTENTS {commands}]

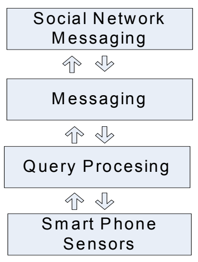


Fig. 2. The core software layers of the system

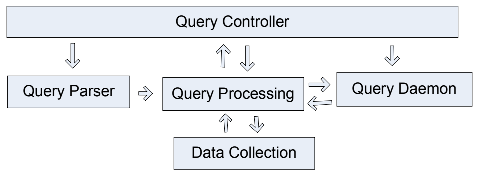


Fig. 3. Primary sub modules of the query processing system

In the query syntax; attributes represents a specific sensor property. We provide basic support for conditional, relational and ACTION queries. ACTION queries provide mechanism for triggering any native actions in the phone if the query conditions are met. E.g. opening camera.

B. *Messaging*

Since we use social network messaging, we needed to design a protocol that enables nodes to exchange messages autonomously. We designed a very simple set of messaging primitives that enabled the communication among nodes. These message primitives followed a specific format. The data that needed to transfer were first converted to our format and then sent using social network's messaging mechanism.

We developed five message primitives according to their message content and tasks. Each message primitive is asynchronous and does not acknowledge received messages. Messages are simply fired into the network and forgotten, if the receiving node receives the message, it processes the message and takes appropriate action.

The message primitives have format as in Fig. 4. Message is simple string starting with primitive type followed by a delimiter colon character and followed by the payload data.

Below each message primitive explained in detail.

1) Query Primitive: The Query primitive carries the typed in query text to the target node. Once the message reaches the target node, the query text is extracted, parsed and the query processing is initiated.

2) Result Primitive: Once the query processing ends, the resultant sensor data is ready to be sent to the source node. The resultant sensor data is put into the result primitive and sent over the network. When result message reaches the source node, the content is extracted and displayed on the phone screen.

3) Exception Primitive: Query processing is immediately stopped if error occurs during parsing or, query processing. An exception message is created with the error message as its content and sent to the source device. The source device displays error message when it receives it.

4) Stop Primitive: The stop primitive is an empty message that tells the target device to stop query processing, and sending sensor data. The current system does not support "duration" operator, which specifies the duration of the query processing. So to get around this problem we developed the stop primitive.

5) Display Primitive: We developed this primitive for debugging purposes. We can check whether a node receives messages by sending this primitive to that node. The node that receives the display primitive, displays the contents of the message.

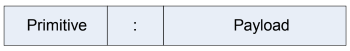


Fig. 4. A Primitive's message format

C. *Routing*

The node's network structure is similar to user's social network structure as we are using user's social network contact list, the node routing is solely based on the user's contact list accounts. When routing we can either specify to route to all the accounts in the contact list, or specify certain people.

If the contact list looks like as in Fig. 5, the nodes of the network will also look the same. If we consider "UserA" wants query from the network, "UserA" node will first fetch the contact list of "UserA" and simply routes messages to the people are who in the contact list. One point to be noted here is that the current system is not designed to forward the query to "friend-of-friend nodes. For example if we take the above network "UserA" forwards the query message to "UserB", "UserE" and "UserD" but "UserB" does not forward the message down to "UserC". This is a limitation of the current system, but in the future we would consider the friends-of-friend nodes.

IV. Implementation Architecture

This section describes the implemented components of the architecture. We chose Google's GTalk [5] as our social network. GTalk users with Gmail Accounts could participate in the network since Gmail users have access to GTalk messaging. We implemented the system in Android phones using Java language. In following sections we explain the main system components.

A. *Query Processing Sub System*

The query processing sub system is responsible parsing, processing queries and collecting sensor data. The query service component acts as a coordinator between other systems and all query requests go through the query service component. The query parser uses ANTLR [2] parser generator. Initially we developed the query grammar according to the query syntax discussed earlier. From the grammar we generated the ANTLR's Lexer and Parser classes using ANTLR tools. The generated classes were then integrated into query processing module.

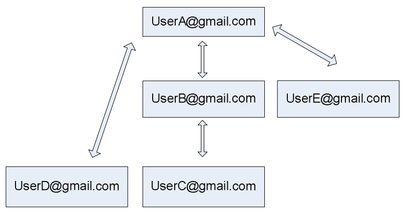


Fig. 5. User's contact list network structure

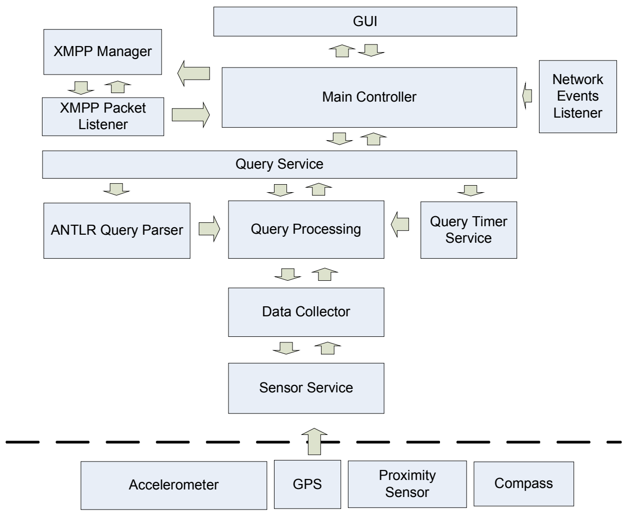


Fig. 6. Illustrates the overall implementation architecture of the system

The query processing module runs the query operators over the sensor data. We implemented query operators using java classes. The query timer service periodically runs the query processing module to process the sensor data. This is because in Android phones the sensor information is not directly accessible, so the sensor data is collected first then it is operated on. The data collector system coordinates with sensor service module to collect and store sensor data temporarily before being processed by the query processor. The sensor service module manages the smart phone sensors in the system. This module when initiated, registers with the sensors to collect sensor data. When sensor data is available the API framework runs the data collector methods and stores the data.

B. *XMPP Messaging Sub System*

a XMPP [13] based messaging system is used. The messaging system manages the network's incoming and outgoing XMPP messages. XMPP is an open standard and widely supported by industry heavy weights such as Google and Facebook. XMPP is a near real-time messaging protocol based on XML. ASmack [4] XMPP client libraries are used to communicate with Google's GTalk [5] servers. The XMPP Packet Listener module listens for incoming XMPP messages. When it receives a message, it is forwarded to the main controller, where it is processed and appropriate action is taken. The XMPP manager is responsible for maintaining the XMPP connections and implementing various utility functions needed for sending and receiving messages

C. *Main Controller*

Any request to sub system needs to be routed to this module. This helps Android phone to run smoothly and function without freezing or not responding, all requests are queued and sent accordingly when resources are available

D. *Others*

The network event listener observes any network related events and informs the main controller to take appropriate action. For example if the Internet connection is lost, the main controller is informed, which then shuts down the query processing running currently. The Graphical user interface (GUI) is the Android screen abstraction; any result that needs to displayed is sent by the main controller to the display area

V. PROTOTYPE

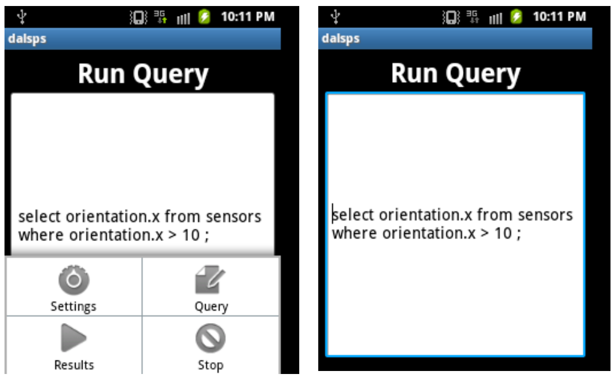
In this section we discuss the details of the implementation. The prototype was implemented with three Android smart phones. All had working sensors and connection to the Internet using 3G wireless network. Also For each device, a Gmail account was created and added to each other's contact list. To join the network user had to install the application in an Android smart phone and set the connection settings.

In Fig. 7, first one represents the first window displayed to user. This features a text box to enter the query. By pressing the menu button on the phone the user can view the menu option items of the application.

Here it displays four options. The first option settings: have the configurations that need to be set up before joining the network. Here the User Id is Gmail e-mail address of the user.

The query menu option item presents the user with list of pre-filled list of query operators that are supported by the system. To run the query, the user has to type in the query text and click the menu option item "Results".

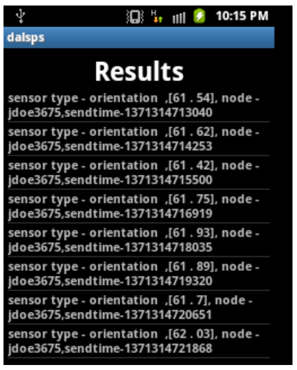
The Fig. 8 shows the results from the nodes in the network. Here each record shows the sensor type, the value of the sensor data, the user account (node) from which the sensor data was sent. And finally it displays the data sent time from origin node



select orientation.x from sensors where orientation.x > 10 ;

select orientation.x from sensors where orientation.x > 10 ;

Fig. 7. Initial display window and the menus available in the system



Results

```
sensor type - orientation  ,[61 . 54], node -
jdoe3675,sendtime-1371314713040
sensor type - orientation  ,[61 . 62], node -
jdoe3675,sendtime-1371314714253
sensor type - orientation  ,[61 . 42], node -
jdoe3675,sendtime-1371314715500
sensor type - orientation  ,[61 . 75], node -
jdoe3675,sendtime-1371314716919
sensor type - orientation  ,[61 . 93], node -
jdoe3675,sendtime-1371314718035
sensor type - orientation  ,[61 . 89], node -
jdoe3675,sendtime-1371314719320
sensor type - orientation  ,[61 . 7], node -
jdoe3675,sendtime-1371314720651
sensor type - orientation  ,[62 . 03], node -
jdoe3675,sendtime-1371314721868
```

Fig. 8. Window displaying results from querying

A. *Basic queries and special features*

This section we will explore some basic queries that can be executed and discuss some advanced features that were incorporated into the system.

1) Relational Queries: The system supports relational operators such as < , >, =,<=, >= , !=

Below we present sample basic query :

select orientation.x from sensors where orientation.x > 1;

Here the ".x" is orientation x's coordinate value, similarly user can fetch y, z coordinate values also. For example, the query below retrieves the coordinate values of y, z:

select orientation.y, orientation.z from sensors where orientation.x > 1;

If the attribute does not mention any coordinate value, then the x is taken as the default attribute. Also note that delimiter semi-colon is needed to mark the end of query.

2) Conditional queries: Following is a sample conditional query

select orientation.x from sensors where orientation.x > 1 and orientation.y < 0;

3) Action Queries: As mentioned earlier the system supports Action queries, where the user can invoke a native procedure in the smart phone. This is similar TinyDB's [1] "OUTPUT ACTION" query. Here we present sample ACTION query:

select orientation from sensors where orientation.x > 1 action_intents display

In the above query, the syntax "action_intents display" specifies that, call the native procedure "display" when the result satisfies the condition orientation.x > 1.

When the conditions are satisfied the system extracts the actions, in this case it's "display"; and creates an Android Intent [1] object and sends to Android Intent filter system. If any method is mapped to this action, then it calls the specific method.

Also the system supports multiple actions in an ACTION query. Below given query executes multiple actions:

select orientation from sensors where orientation.x > 1 action_intents display,print.

VI. DISCUSSION

In this section we describe critical issues with the system, the relative differences from other similar projects and explore areas in which our architecture can be used

A. *Scalability and Bandwidth usage*

XMPP [13] is scalable depending on its deployment architecture. GTalk [5] which provides XMPP based chat messaging is used by millions of users; to support such scale, Google uses its vast messaging infrastructure. XMPP has been compared with another competing presence protocol SIMPLE [15]. The performance analysis indicated, XMMP performed better in handling presence and bandwidth [16]. So therefore our system is inherently scalable due to the fact that we use GTalk messaging and XMPP.

The interconnection of nodes via WiFi and Blutooth while inexpensive, it cannot connect nodes beyond its local network range. We want our system connect to large number of nodes and in a wider area.

Scaling issues arising from friend-to-friend mechanism may be mitigated, by using network optimization techniques used in TinyDB [12]. One such technique is the use of aggregate queries, where sensor data results are aggregated at each node, while data traverses up to the root node. Using this approach the root node won't get inundated with responses from all the nodes in the network. The response messages will be only coming from the nodes that it had originally sent. Because the intermediate and leaf nodes in the network wont be able to send messages directly to the root node, but only to its immediate parent node. This approach reduces the number messages need to be sent (only aggregate values are sent). Also compression might be applied further reducing size of the message, so there by reducing bandwidth usage.

B. *Security and Privacy issues*

For current implementation we can introduce a white list of people who may query. And also enable user to opt out of the network. We can introduce access permissions for each sensor. Screen alerts when a user tries querying the phone, at that point user can accept the query or reject the query. Also we can incorporate triggers which can shutdown the application when power is low, or user uses the Internet for some other use.

For the future friend-to-friend implementation, we can use Oskar Sandberg's distributed routing mechanism [11]. This routing mechanism is the basis for Freenet [6], which is a P2P platform for sharing files. It is an ananonymising network where connection is made only to trusted users. This approach may reduce the security and privacy concerns.

C. *Other Issues related to system*

The current implementation only the immediate friends in the contact list can participate in the query network, more effective way to build the network might be to use the "friends-of-friend" nodes in the social network, so our intention to move in that direction in future implementations. The evaluation was done only with three phones. The ideal case would be to have large community of users who would use the application. Since this is not viable, we would need to develop a simulator to model social network and it's mobile users. This way routing mechanism, bandwidth usage, scale and other various properties can be validated before deploying it into the real-world.

D. Architectural comparison to other similar systems

In this section we try to compare broadly, our architecture with similar systems. One obvious difference is that we use an existing Social network as opposed to creating a community from ground up. Also as mentioned earlier,In most community based sensor application systems, the communication architecture may based on client-server model where phones collect and post data to a central server or a hybrid client-server model where co-located phones make ad-hoc network connections and then send the aggregated data to the central server. Our architecture does not have a central server or central processing. One advantage our architecture has is that we can deploy social based application very quickly since the social network is already in place and also our architecture does not need any central server support.

On the dis-advantage side, since we rely on the social network provider for infrastructure support, they may take way the free access in the future. We cannot do any post processing or information mining on data since we do not collect data in a central server. Also since we cannot implement any custom message formats, the current way of passing commands in chat messages isn't an elegant way to pass data. Also there is danger being flagged as spam.

E. Possible Applications

We believe that any social network based sensor applications, which need to be deployed very quickly can be implemented with this architecture, especially if its need to be every cost-effective and open source solution.

F. Possible Hybrid Architecture Models

We can build hybrid models of this architecture, which we can mitigate dis-advantage listed earlier. For example we could employ one smart phone as sort of base station, modify its code to log the results, this phone can directly connect to pc or laptop for further processing or logging data. This base station phone can put the query into network and log the resultant data, which later can processed offline for information extraction.

VII. Future

In this section we suggest our future directions in extending the system. Here we propose general enhancements to system in query processing and optimization. The messaging protocol needs to be revamped to handle node failures and synchronization. We propose to use the "small world phenomenon" [7] in social networks to form the sensor network, to complement this we also suggest new query operator "location" which shall collect sensor from particular region in the social network.

VIII. Conclusion

We have described XMPP based data abstraction layer, its components in social network setting, where we exploit the existing user profiles used as nodes and social network infrastructure to route messages. This kind of data abstraction layer can be used in variety social applications that use the collective sensor data. We showed that this kind architecture is possible by creating prototype using Android phone. The prototype mainly consisted of query module and a messaging module. The query module parsed and processed queries while the messaging handled routing and communication between nodes. Also we developed messaging primitives which is a very basic protocol that handled the messaging between nodes. Using this prototype we gained valuable insight and we have noted several areas in which we are moving forward in this research area as we mentioned in the future section.

We believe smart phone sensor applications will tightly integrated into social networks in the future and will play a key in the social based environment monitoring applications, and believe our work contributes to this emerging field classified as participatory sensing.

Acknowledgment

I would like to express my sincere thanks all the people who provided valuable guidance during my work.

References

[1] Android Intents. (2013, Jan 10). [Online]. Available: http://developer.android.com/guide/components/intents-filters.html.

[2] ANTLR. (2013, Jan 10). [Online]. Available: http://www.antlr.org/.

[3] A. Sayakkara and T. N. K De Zoysa, "A Declarative Interface for Smart-phone Based Sensor Network Systems," In 2nd Int. Workshop on Mobile Sensing, Beijing, China, 2012.

[4] ASMACK. (2013, Jan 10). [Online]. Available: https://github.com/Flowdalic/asmack.

[5] Google GTalk. (2013, Jun 10). [Online]. Available: http://google.com/talk

[6] I. Clarke, O. Sandberg, M. Toseland, and V. Verendel., "Private communication through a network of trusted connections: The dark freenet." (2013, Nov). [Online]. available: https://freenetproject.org/papers/freenet-0.7.5-paper.pdf.

[7] J. Kleinberg, "The Small-World Phenomenon: An Algorithmic Perspective," In Proc of the 32nd ACM Symposium on Theory of Computing, Portland, OR, May 2000.

[8] Miluzzo et al, "Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application," In Proc. 6th ACM Conf. Embedded Network. Sensor Systems, Raleigh, NC, 2008, pp. 337–350.

[9] N. D. Lane, "Community-Aware Smartphone Sensing Systems," IEEE Internet Comput, 2012, vol. 16, no. 3, pp. 60-64.

[10] N. M. Laxaman, M. D. J. S. Goonatillake, and K. D. Zoysa, "Tikiridb: Shared wireless sensor network database for multi-user data access," IITC, 2010.

[11] O. Sandberg, "Distributed routing in small-world networks," In 8th Workshop on Algorithm Engineering and Experiments(ALENEX06), Miami, FL, 2006.

[12] S. R. Madden, "The Design and Evaluation of a Query Processing Architecture for Sensor Networks," Phd dissertation, Dept. Comput. Sci., Univ. of California Berkeley, USA, 2003.

[13] XMPP. (2012, Dec 15). [Online]. Available: http://xmpp.org.

[14] WAZE. (2013,Jun 10). [Online]. Available: http://www.waze.com/

[15] SIMPLE (2013, Nov, 10). [Online]. Available: http://tools.ietf.org/html/draft-ietf-simple-simple-07

[16] Presence scalability (2013, Nov, 10). [Online]. Available: http://xmpp.org/2007/04/presence-scalability/