Publish/Subscribe Communication for Crowd-sourcing Based Smart City Applications

Róbert L. Szabó*† and Károly Farkas*‡
*Inter-University Centre for Telecommunications and Informatics, Debrecen, Hungary
†HSNLab, Dept. of Telecommunications and Media Informatics,
‡Dept. of Networked Systems and Services,
Budapest University of Technology and Economics, Budapest, Hungary
Email: szabo.robert@etik.hu, farkask@hit.bme.hu

Abstract—**Collecting data and monitoring our environment give the basis for smart city applications which are getting popular today. However, the traditional approach to deploy a sensing and monitoring infrastructure is usually expensive and not always practical. Mobile crowd-sourcing can open new ways for data collection and smart city services. In this case, mobile devices with their built-in sensors and their owners are used to monitor the environment. For instance, the timetable of a smart travel planner service can be updated in real-time based on the continuously monitored time gap by passengers between consecutive buses on a public transportation route. This requires a common communication model facilitating crowd-sourced data collection. In this paper, we overview the design considerations of crowd-sourcing based smart city applications, propose data collection via using the publish/subscribe communication model and investigate the possible use of the Extensible Messaging and Presence Protocol (XMPP) for such applications.**

Keywords-Crowd-sourcing; Publish/subscribe; XMPP; Internet of Things

I. INTRODUCTION

In the 21st century, environmentally sustainable living style more and more comes to the front. Cities attract an increasing population with their greater political and economical power. These cities also develop rapidly in technology as their core systems are getting interconnected and getting more intelligent. To seize this opportunity intelligent applications are needed which – based on the core infrastructure of the cities – can make our cities even smarter by enriching our everyday life.

For introducing such applications the surrounding environments are to be sensed and monitored. Traditionally, some pre-established sensing infrastructure should be in place to collect data which can be an inflexible and costly solution. Fortunately, the proliferation of mobile smartphones enables smart sensing via mobile crowd-sourcing [1]. Thus, the built-in and ubiquitous sensors of the phones are used for sensing either in participatory or opportunistic way depending on whether data collection happens with or without user involvement. Moreover, the mobiles can be used as an interface to the smart applications.

Handling the crowd-sourced data collection calls for a common platform and communication model. For managing resources and handling the communication in resource constrained networks a number of protocols were proposed. For instance, the Constrained Application Protocol (CoAP) [2] or the Message Queuing Telemetry Transport (MQTT) [3] are application layer solutions. They require the use of gateways or proxies introducing unnecessary complexity. Moreover, CoAP is based on the request/response scheme which assumes synchronous communication.

In a crowd-sourcing based smart city application scenario, a publish/subscribe (pubsub) communication scheme is more suitable where information sources publish events autonomously to the pubsub service whenever something happens. Users can subscribe to event contents and the pubsub service delivers the events to all the interested parties. We believe that the Extensible Messaging and Presence Protocol (XMPP) [4] suits well for such communication.

In this paper, we overview the design considerations of crowd-sourcing based smart city applications and investigate the possible use of the XMPP protocol for communication and data collection functions. XMPP is already established, standardized, freely available and provides the primitives which can be used as building blocks in such applications.

The rest of the paper is structured as follows. In Section II, we overview related work. Our design considerations are presented in Section III. We discuss the possible use of XMPP as the communication platform in Section IV. Finally, we give a short summary and sketch our future plans in Section V.

II. RELATED WORK

In a recent overview paper [1] Ganti et al. reviewed the current state of the art and challenges of mobile crowd-sourcing. They identified mobile smartphones as possible bridges of everyday objects to our enriched world. They envisioned a plethora of Internet of Things (IoT) applications that will be available in the future through community based sensing and monitoring. Among the many challenges like resource limitations, aggregate analytics, privacy, security and data integrity, they identified the unifying architecture as a major need for easy development and deployment of crowd-sourcing based IoT applications. We believe that XMPP is a viable candidate for that purposes.

Kirsche and Klauck in [5] have already proposed to bring XMPP into the Internet of Things. For the same reason to have a unifying protocol architecture, they have investigated whether XMPP can be used in embedded systems. They

concluded with experiments that XMPP can be minimized to run on resource constrained devices and being able to communicate with full fledged clients. Based on their results it will be possible to not only bring smartphone based sensing into the common platform of XMPP, but also smart objects.

Buddycloud [6] and OneSocialWeb [7] are two social network projects using XMPP. The former is an open source distributed social network which provides an efficient way for servers on different domains to connect and synchronize updates. In buddycloud, the communication model is based on XMPP. The latter is a standardization effort which is aimed at defining the 'language' to bridge different social networks and make it easy for them to join and share information across social network boarders. In the OneSocialWeb initiative, XMPP is used as the core engine.

On the other hand, there are approaches to create open frameworks from the scratch for crowd-sourced data collection and smart city services. For example, the Funf Open Sensing Framework [8] is an extensible sensing and data processing framework for mobile devices. It provides an open source, reusable set of functionalities, enabling the collection, uploading and configuration of different data types. Funf is using probes as basic data collection objects. These probes can be remotely configured and collect on-phone sensor and many other types of data, such as application usage or browsing history.

Another crowd-sourcing tool is the Ushahidi platform [9] which uses multiple channels to crowd-source information including Twitter, email, SMS and the Web. This open source software platform enables volunteers to map everything from local events to globally relevant incidents. When an event occurs a volunteer sends a brief report via the Web or text message and the software annotates it with time and location information. Thus, the platform provides powerful geographical mapping tools, but unfortunately it does not handle automatically the built-in smartphone sensors.

III. DESIGN CONSIDERATIONS

We outline our design considerations of crowd-sourcing based smart city applications following the capabilities categorized by Mattern and Floerkemeier in [10].

A. Services

The goal of smart city applications is to deliver information about objects to users. However, if it is not known in advance who needs when what information, then it is very difficult to design tailor made and useful communication functions. Therefore, first we must investigate the communication pattern of crowd-sourcing based smart city applications.

In a crowd-sourcing based smart city application scenario, user devices sense the environment and associate it with surrounding objects. Each such measurement and association can be considered as an event created by the user. These events could have common attributes like creation times, associated objects and locations (absolute or relative) besides additional varying and unknown further event contents. Such events describe the context of objects but originate from users. The collecting user in a crowd-sourcing scenario is willing to share these events with others. On the receiving side, any user shall be able to receive events he/she is interested in.

Such communication architecture resembles a publish/ subscribe (pubsub) communication model, where sources of information publish events to the pubsub service and users can subscribe for event streams of interest. At the pubsub service users can register for event streams by subscribing to specific event contents. In turn, the pubsub service is responsible for checking the received event against all of its current subscription and delivering the event to all the interested users whose subscription match the event. The pubsub model allows asynchronous and multicast communication between users in a dynamic environment [11].

B. Communication and Cooperation

In line with the idea presented in [5], we believe that the communication and cooperation between clients and the server could be realized through the standard protocol of XMPP [4]. XMPP could bring a uniform communication model into the mobile crowd-sourcing world. It is an open technology for real-time communication using Extensible Markup Language (XML) [12] message format. XMPP allows sending of small pieces of information in XML format from one entity to another in quasi real-time.

Furthermore, XMPP has several extensions, like multi-party messaging [13] and a notification service [14]. The notification service realizes a publish/subscribe communication model, where publications sent to a node are automatically multicast to the subscribers of the node. Moreover, [14] introduced collection nodes to more easily manage subscriptions through aggregate of notifications. Since XMPP is widely used in instant messaging services, it is well established and proven both in technology and in scalability. We detail the proposed use of XMPP in Section IV.

C. Addressability

In our approach, the things will not be directly addressable, hence will not directly be involved in the communication. Instead, information collected about the surrounding things will be reported by the user's equipment (smartphone), which are already connected to the Internet through the standard IP protocol stack with public or private addresses. In either case, they can initiate connection setups to the servers.

D. Identification

In order to be able to assign context information to surrounding objects, they must be identifiable. Identification can be done directly or through indirection. Direct identification could include real world physical object IDs (e.g., a licence plate number). The advantage of real world IDs is that they are already deployed but they are usually difficult to be automatically scanned by user devices. On the other hand, if identification helpers (smart tags) are deployed to the objects, then automated or semi-automated recognition becomes feasible. Such smart tags could be optically readable codes (e.g., QR codes), radio frequency identifiers (RFID/NFC tags) or even WiFi network identifiers (SSIDs). These smart tags could directly carry the real world IDs, however, this would need careful programming and deployment of such tags. That would just simply fail in a community based effort.

Therefore, we propose to use indirection with smart tags, i.e., smart tags will carry their own (possible) unique IDs, whose context will identify the physical object itself. This will allow late binding of smart tags and objects by the community, when the smart tag is first read. Such indirection would also allow the reuse of such tags, e.g., a magnetic tag could be assigned to different objects in time.

Another issue is how the identifier name space is constructed. To allow for maximum flexibility in the deployment, we propose to use a flat ID space. Such flat ID could be created from any identifier by a hash function. The benefit would be that even real world identifiers can be easily mapped to such hash IDs and that physical objects can have several, independent tags associated with them. Any change in one of these tags will not affect the others.

So, we propose an architecture where flat digital IDs indirectly identify physical objects. The resolution of the flat ID space to physical objects shall be done in a scalable way, e.g., by distributed hash tables (DHTs) [15].

E. Sensing

The goal is to enrich our living with the information related to the physical objects surrounding us. However, before we come to the point when objects themselves will be able to sense their surroundings, we all have to do our part in collecting these information. Therefore, the actual sensing will be done by the user or the user's smartphone device. The collected information will be reported to the identified smart tags nearby, which in the server side will be analyzed and assigned to the corresponding objects. The user's device can also request an object identification based on the identified mart tag to modify the sensing process (frequency, collected information, etc.). Alternatively, sensors and sampling strategy of the user's device could also be remotely controlled based on the identified objects associated by the detected smart tags.

F. Actuation

The goal here would be to manipulate the objects' environment through actuation. However, assuming passive objects, this is not feasible. Nevertheless, indirect control of the objects' environment might be possible based on interfaces at the server side. Such example could be to notify the traffic center about traffic jams, who in turn could initiate correcting actions. Such feedback loop could involve several parties with the final goal of manipulating the objects' environment.

G. Embedded Information Processing

The demand for embedded information processing is twofold: i) we want to have real time information about the objects' state; and ii) we would like to disseminate and present meaningful information to the users. Both demands embedded information processing at the client side. Another issues is to give incentives to people to contribute to the sensing of the environment. This is best achieved, if the individual user herself could also benefit from the collected and processed information.

H. Location and Status

In order to enrich our physical world, we want to be aware of the location and status of the objects that are out of our physical reach. In our architecture, location of objects can be estimated by the reporters' location. Such location information can come from GPS signals or from proximity sensors like 3G Cell-ID, WiFi access points or even short range radios like Bluetooth, Zigbee, or RFID/NFC tags. Similarly, the status of the objects can be estimated from the context of the reporters, e.g., being in a traffic jam or meeting the run schedule. It can be noticed, that due to the diversity and uncertainty of possible status information of objects it is much more difficult to conclude such information autonomously at the sensing (client) side.

I. User Interfaces

Keeping in mind that we want to offer information about objects of our interest, the goal is to filter and visualize relevant information to the user. Since we rely on users' smartphones as primary sensors this enables us to use these smartphones for user interactions. Fortunately, today smartphones offer familiar or even well received interfaces to the users. Our goal must be to design appropriate visualizations and non disturbing interaction interfaces that will be accepted and used by the community.

IV. COMMUNICATION THROUGH XMPP

XMPP defines a standardized way for event based messaging, which besides others also supports the publish/subscribe communication model. Therefore the possibility to reuse the existing services of XMPP seems appealing. In the following, we discuss how XMPP can be used to facilitate communication and data collection.

A. Publish/Subscribe Nodes

In order to realize the publish/subscribe communication model objects of interest must have their own pubsub nodes at an XMPP server. Such object related pubsub nodes could be the leaf pubsub nodes at the server.

If the XMPP's default *open access* model is used, then without any further interactions users can subscribe and receive events from the pubsub node. However, publishing to a pubsub node requires special *affiliation*, like a `publisher` or a `publisher-only` [14]. These can only be granted through authorization requests. For convenience, either this authorization process shall be automated to allow easy user publication to pubsub nodes or an extension to XMPP to handle special access rights might be added. See XEP-0060 draft standard [14] for the elaboration on different affiliations and access rights.

Assuming appropriate affiliation of the user, when he/she wants to publish some information to the object's node, he/she sends an XMPP `info/query` (*iq*) message to the node with an item containing the event's payload. The payload of the item can be flexibly defined according to the standard of Atom notifications [16].

At this point, all subscribers of the pubsub node will receive a notification `message` from the XMPP service containing the published `item`.

B. Collection Nodes

In our everyday life, we get used to hierarchies and ordering. When we want to know the timetable for a specific bus run, we start looking for it at the operator, at the bus lines and at the bus itself sequentially. Such ordering among objects seems even more important when not only the group of objects will carry information but also the individual ones. Therefore, we propose to group and order object related leaf pubsub nodes into hierarchies.

In such a model, only the leaf pubsub nodes could receive events from primary sources, who directly sense and report the context of the object. On the other hand, the grouping nodes would contain all their child nodes' events (aggregates). This can be achieved by XMPP's item aggregation via collection nodes [14]. In XMPP, when a leaf node is associated with a collection node, the items published to a leaf node are also pushed to entities subscribed to the collection node. Following the public transportation scenario, this would result in an aggregation hierarchy from cars to lines to buses to the operator as shown in Fig. 1. Such grouping of objects would allow easy reception of primary events in a hierarchy.

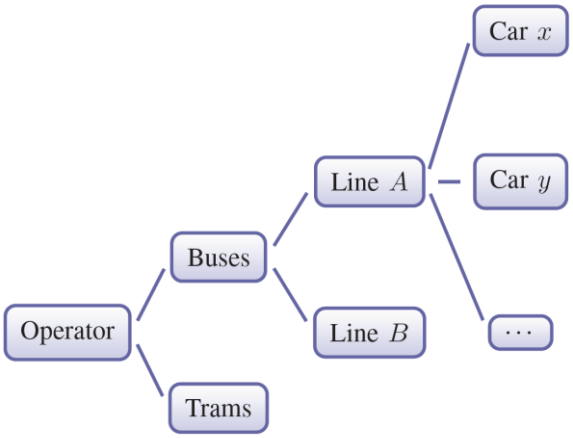

Figure 1. Collection example for a transportation operator

C. Service Discovery

Users of the system will be interested in information related to specific objects and would like to avoid flooding of their client with unsolicited data. As we have seen the grouping of objects into hierarchies allows easy event aggregation, but not easy separation. In order to constrain the reception of events to objects of interest we need a discovery mechanism for existing collection and leaf nodes.

Fortunately, the core XMPP protocol supports service discovery. In our case, clients shall issue an *info/query* `disco#info` stanza for querying services and *info/query* `disco#items` to discover the top-level nodes available at the discovered service. Next, information about the nodes must be queried by *info/query* `disco#info` specifying the node of interest. The response will identify the node's type as `collection` or `leaf` [17]. A `collection` node can further be queried for its items (children), which might be `leaf` or `collection` nodes themselves. Last but one, an *info/query* `disco#items` to the node can reveal detailed information about the node, like its ID, creation time, owners, etc. (see [18] for further details) to decide on subscription. At last, the user can decide whether to subscribe to a specific `leaf` or `collection` node based on the information discovered about the service.

Fig. 2 shows an example, where a client discovers two pubsub nodes (*A* and *B*) at the XMPP server. In the example, node *A* is a `collection` node for node *B*. The client subscribes to node *B* wit publish-only affiliation. With publish-only, the client is able to publish events to the node but it will not receive the published events. However, the client subscribes to the `collection` node *A*, where he/she will receive the events published to node *B* through the aggregation.

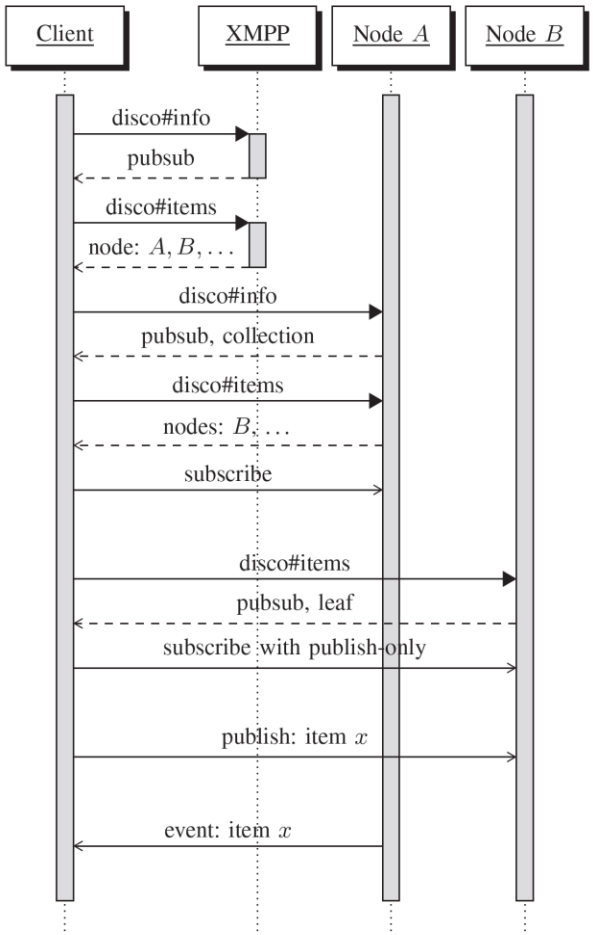

Figure 2. XMPP discovery, publish and receive

D. Analytics

We have seen how users can discover and subscribe to low level primary or aggregated events of objects or object hierarchies. However, most users will be interested in value added information related to these objects. This information can be deduced by third parties by analyzing raw event streams and publishing value added information (e.g., status, presence, location) related to these objects. Results of analytics can either

be published directly to the object's node or to a separate node or a separate node hierarchy.

For example, a 3rd party analytics provider can remap the original object hierarchy to his/her own pubsub node hierarchy. This would allow users to subscribe only to these value added services but in the same time to contribute with publishing to the object's node. This scheme would suit well a crowd-sourcing based model, where individual users would contribute to the sensing of objects in their proximity, but would be mostly interested in receiving information about objects further away. Therefore, the goal of analytics would be to provide value added reports to these objects, which could not easily be derived from the individual event stream.

E. Access Management

It might be desirable for users to limit access to information related to their objects. For instance, the value of a household's power meter might be too sensible to be published openly. However, the user might be interested in an energy consultancy service, which needs access to the daily energy consumption to be analyzed for rationalization. Such service provider could be easily granted access to the events of the power meter to perform analytics. When the service period or the contract with the analytics provider is over, the user can revoke these grants.

V. SUMMARY

In this paper, we presented our design considerations for crowd-sourcing based smart city applications and investigated the possible use of the XMPP protocol for communication functions. We believe, that the publish/subscribe communication scheme suits well the crowd-sourcing approach. XMPP is already established, standardized, freely available, supports the publish/subscribe communication model, which make it an ideal candidate for crowd-sourcing based application scenarios.

As future work, we plan to implement an experimental platform based on the XMPP protocol together with some simple crowd-sourcing based smart applications, such as smart travel planner or smart parking.

ACKNOWLEDGMENTS

The publication was supported by the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project. The project has been supported by the European Union, co-financed by the European Social Fund. Károly Farkas has been partially supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

REFERENCES

[1] R. Ganti, F. Ye, and H. Lei, "Mobile Crowdsensing: Current State and Future Challenges," *IEEE Communications Magazine*, pp. 32–39, Nov. 2011.

[2] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained Application Protocol (CoAP)," Internet Engineering Task Force, Nov. 2011. [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-coap-08

[3] IBM, "MQ Telemetry Transport." [Online]. Available: http://mqtt.org

[4] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 6120 (Proposed Standard), Internet Engineering Task Force, Mar. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6120.txt

[5] M. Kirsche and R. Klauck, "Unify to bridge gaps: Bringing XMPP into the Internet of Things," in *2012 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, Mar. 2012, pp. 455–458.

[6] Buddycloud Team, "buddycloud." [Online]. Available: http://buddycloud.com

[7] OneSocialWeb Team, "OneSocialWeb." [Online]. Available: http://onesocialweb.org

[8] MIT, Media Lab, "Funf - Open Sensing Framework." [Online]. Available: http://www.funf.org

[9] Ushahidi Team, "Ushahidi Platform." [Online]. Available: http://www.ushahidi.com

[10] F. Mattern and C. Floerkemeier, *From the Internet of Computers to the Internet of Things*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Jan 2010, pp. 242–259. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-17226-7 15

[11] Y. Huang and H. Garcia-Molina, "Publish/Subscribe in a Mobile Environment," *Wireless Networks*, vol. 10, no. 6, pp. 643–652, Nov. 2004.

[12] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (XML) 1.0 (fifth edition)," W3C, W3C Recommendation REC-xml-20081126, Nov. 2008. [Online]. Available: http://www.w3.org/TR/2008/REC-xml-20081126/

[13] P. Saint-Andre, "XEP-0045: multi-user chat," XMPP Standards Foundation, Standards Track XEP-0045, Feb. 2012. [Online]. Available: http://xmpp.org/extensions/xep-0045.html

[14] P. Millard, P. Saint-Andre, and R. Meijer, "XEP-0060: Publishsubscribe," XMPP Standards Foundation, Draft Standard XEP-0060, Jul. 2010. [Online]. Available: http://xmpp.org/extensions/xep-0060.html

[15] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up Data in P2P Systems," *Communications of the ACM*, Feb. 2003.

[16] M. Nottingham and R. Sayre, "The Atom Syndication Format," RFC 4287 (Proposed Standard), Internet Engineering Task Force, Dec. 2005, updated by RFC 5988. [Online]. Available: http://www.ietf.org/rfc/rfc4287.txt

[17] P. Saint-Andre, R. Meijer, and B. Cully, "XEP-0248: pubsub collection nodes," XMPP Standards Foundation, Standards Track XEP-0248, Sep. 2010. [Online]. Available: http://xmpp.org/extensions/xep-0248.html

[18] P. Saint-Andre, K. Smith, and R. TronCon, *XMPP: The Definitive Guide: Building Real-Time Applications with Jabber Technologies*, 1st ed. O'Reilly Media, May 2009.