A Simple M2M Overlay Entity Discovery Protocol

Teemu Väisänen
VTT Technical Research Centre of Finland, Oulu, Finland
Teemu.Vaisanen @ vtt.fi

***Abstract*—This paper deals with discovering M2M overlay entities in Machine-to-Machine (M2M) service networks. The eXtensible Messaging and Presence Protocol (XMPP) is used as a basic building block for the M2M communication. XMPP does not offer mechanisms for discovering unknown entities from unknown contacts, and this paper's goal is to provide a protocol enabling this. The presented protocol does this by using asynchronous remote procedure calls (RPCs), unicast messages and friend-to-friend type of communication. The paper proposes new XMPP subscription statuses to enable exchange of roster items in the M2M overlay entity discovery protocol without compromising privacy, and presents the protocol for discovering unknown M2M overlay entities from unknown M2M overlay entities.**

Keywords - distributed systems, service discovery, privacy

I. INTRODUCTION

Machine-to-machine (M2M) is a buzzword meaning a bagful of technologies that allow devices to communicate with one another using different communication channels. Terms such as M2M, Internet of Things (IoT), Smart Objects (SO), and Web of Objects basically all mean the same, including e.g., remote management of devices. Technologies used commonly in M2M include at least naming and identification of entities, service discovery (SD), security services such as authentication, and communication technologies. End-to-end M2M communication can be established with one or more protocol conversion gateways running between the actual M2M devices, but the trend is to build systems where end-to-end communication is possible for example using IPv6 [1]. Overlay networks in M2M are sometimes called middleware and they are often used on top of other networks to make naming easier, to improve routing, and/or to improve Quality-of-Service (QoS). In M2M service networks, a SD protocol can be thought of as a comprehensive discovery protocol including at least functionalities of M2M device, overlay entity and service discovery mechanisms, and mechanisms for selecting discovered M2M services that are to be used. The selection can be based on discovery order, location, etc.

The M2M service network presented in this paper is based on the eXtensible Messaging and Presence Protocol (XMPP) [2]. M2M overlay entities are identified as XMPP Jabber Identifiers (JIDs) running in XMPP clients. M2M services are running in M2M overlay entities. XMPP's SD mechanisms do not offer the possibility for discovering unknown overlay entities from unknown rosters, which is discussed more in details in Section II. The protocol presented in this paper provides a solution to this problem. Later in this paper this M2M overlay entity discovery protocol is called only service discovery (SD) protocol.

The rest of this paper is organized as follows: Section II gives information about XMPP and existing related SD protocols, Section III introduces the main contribution, the SD protocol and how XMPP is used, Section IV gives evaluation and presents some of the use cases, which were used to test the protocol, while Section V concludes the paper and gives proposals for future work.

II. RELATED WORK

XMPP is a set of open XML technologies for presence and real-time communication. It is continuously extended through the standardization process of XMPP Standards Foundation. XMPP was originally created for near-real-time messaging, presence, and request-response services [3][4], but it has been used to build e.g., Smart Grids [5], M2M architectures [6][7], and sensor networks [8]-[13].

XMPP offers built-in publish-subscribe ("pubsub") functionality [14], so polling is not necessarily required between clients and services. The specification of pubsub is long, but the idea is simple: 1) An entity publishes information to a node at pubsub service, and 2) the pubsub service pushes a notification to all entities that are authorized to learn about the published information.

XMPP uses Transport Layer Security (TLS) to secure server-to-server and client-to-server connections [2] and offers end-to-end signing and object encryption [15]. In addition to these, security extensions exist or are proposed, including e.g., privacy [16]. XMPP is distributed; anyone can have their own servers with several clients. Client-server architecture commonly enables connectivity through firewalls, because clients initiate sessions. XEPs such as [17][18] exist to help with firewalls.

XMPP has four presence subscription statuses. In 'none' state the user does not have a subscription to the contact's presence information, and the contact does not have a subscription to the user's presence information. In other words, you are not interested in the item's presence, and the item is not interested in yours. In 'to' state the user has a subscription to the contact's presence information, but the contact does not have a subscription to the user's presence information. In 'from' state the contact has a subscription to the user's presence information, but the user does not have a subscription to the contact's presence information. In 'both' state both the user and the contact have subscriptions to each other's presence information [2]. These statuses do not tell anything about how (if at all) your presence information should be told by your contacts to their contacts or contacts

 ISBN: 978-1-61208-202-8

unknown to you and/or to your contacts. This paper discusses how these kinds of situations with existing XMPP statuses should be handled and how new presence subscription statuses could be used to improve privacy wishes.

XMPP creates long-lived sessions between communicating entities. Sessions might shut down because of many reasons and repeating session initialization might be problematic and/or slow. XMPP uses XML stanzas, and XML parsers may become a bottleneck in embedded devices and in networks that have limited bandwidth capabilities. However, Binary XML might become more common in the future: XMPP is presented as one use case in WC3's XML Binary Characterization Use Cases [19] and it has been proposed to be used with Efficient XML Interchange (EXI) [20]. If XMPP is used in mobile devices in the same way as in PCs with fixed power, problems such as battery running out will certainly arise. Because of this, an extension providing knowledge of mobile handset behavior has been described [21]. In addition, other XEPs to decrease bandwidth exist, e.g., Stream Compression [22]. Although XML takes resources, the smallest interoperable XMPP client implementations work in embedded devices such as sensors [8]-[13], and there are client and server implementations for mobile phones.

XMPP has two different types of SD protocols [23][24]. In the basic XMPP SD protocol [23] entities are servers, clients or gateways. The protocol provides methods for 1) discovering entities (disco#items) and for 2) discovering features supported by a given entity (disco#info). An XMPP client knows at least one other entity, its server. The client is able to discover services (multi-user chatrooms (MUC) [25], pubsub, etc.) offered by the server and features that are supported in those services. XMPP's serverless messaging specification [24] defines mechanisms that enable working without servers, e.g., in body area networks, or LANs: Principles of zero-configuration networking (Zeroconf) [26] are used. Zeroconf uses multicast DNS (mDNS) [27] and DNS-Based Service Discovery (DNS-SD) [28]. In XMPP roster item exchange can be done e.g., with service administration [29] or roster-item exchange [30].

In social networks, discovery protocols have been used for finding friends, groups, links, etc. "Google's Search plus Your World" (earlier "Google Social Search") [31] has features that allow your friends to affect your search results. The SD presented in this paper lets the XMPP client answering the SD request to build the answer. Facebook social search [32] includes information about the frequency of clicks on the search results by members of the social network who are within a predetermined degree of separation from the member who submitted the request. This degree can be compared to a hop limit in the presented SD protocol.

III. THE SD PROTOCOL

This section presents the main contribution of this paper, the SD protocol. The purpose of the SD protocol is to discover M2M overlay entities from unknown M2M overlay entities without compromising privacy. Protocol must work without centralized naming services, which keep track of M2M overlay entities, it must not broadcast huge amount of messages and flood the network, and it must work over XMPP, but also in XMPP clients without support for service administration [29] or roster-item exchange [30]. This paper presents a SD protocol that meets these requirements, using ideas coming from the following real life examples:

A tap in Eemil's bathroom has started to leak and has caused moisture problems in the bathroom. Eemil wants to find a person who could fix these problems. If he already knows someone who has fixed bathrooms or taps before, he will probably ask these people to help first. If Eemil does not know anyone who is able to fix his bathroom, he might ask from his friends if anyone has a friend who has fixed bathrooms before. If one of his friends knows such a person, it is likely that he or she gives this information to Eemil. If none of Eemil's friends know such a person, they can ask from their friends. If any of Eemil's friends' friends know such people, they can send this information directly to Eemil (if Eemil's friends have told who is the original requester), or to the last requester (Eemil's friend who asked it) and they can forward the reply to Eemil. When thinking the example further, Eemil can select friends from whom he wants to ask the fixer. If Eemil thinks that some of his friends should certainly know about people working with bathroom fixing, or that some of his friends know more people than an average person, he will probably ask from them first.

Based on this real life example and to enable discovering unknown entities from unknown M2M overlay entities, the hop limit was selected to be two. Hop limit is analyzed in Section IV.

XMPP offers basic building blocks listed in Section II, such as message transmission, naming of M2M overlay entities (nodeid@domainid/resourceid) and rosters, for the protocol. In used M2M service network, every XMPP JID in XMPP client has its own private roster, which is called a private M2M overlay. Rosters are stored to XMPP servers. XMPP servers can be clustered. Roster includes other XMPP JIDs. The overlay can be constructed in several ways: by adding roster items after registration, using different presence subscription statuses [2], or accepting all XMPP client subscriptions from the XMPP server it is registered to. These are mainly implementation and configuration issues. MUC [25] rooms can be thought of as second type of a M2M overlay. Describing usage of MUCs in the SD protocol is out of the scope of this paper. A M2M device runs one or more XMPP clients. An XMPP client has one or more JIDs registered, which can be registered to one or more XMPP servers.

The following list contains required features of a M2M overlay entity, which are not offered by XMPP:

1. The entity can ask from its contacts to parse a string presenting the wanted M2M overlay entity name.
2. The entity can forward the parsing request to its contacts.
3. If the entity finds a wanted string from its roster list, and contacts in the answer accept forwarding their information, the entity answers to the requester, who might be the original requester or forwarder of the parsing request.

 ISBN: 978-1-61208-202-8

4. The entity can parse rosters with additional information such as reputation.

The SD protocol works at the application layer inside an XMPP client implementation. It can be categorized as a unicast protocol, as it sends direct discovery messages to known receivers. Its purpose is to discover M2M overlay entities presented as JIDs.

Our proposal is that when using XMPP's four existing presence subscription statuses "none", "from", "to", and "both", by default the entities should not advertise the existence or the presence of one another to anyone else.

Because some, but not all, M2M overlay entities or their owners might want to share their JIDs to unknown entities, there is a need for new XMPP presence subscription statuses. They could be such that the entity replying to a SD request knows if the JID in the answer allows giving its name to other entities. At least the following new additional presence subscription statuses to XMPP are needed:

subscription='from-anyone': This means the same as the XMPP state 'from' but also that your contact can advertise you and your presence information to anyone. Notice that this does not mean that anyone who tries to subscribe to your presence information is necessarily accepted by you.

subscription='from-contacts': This means the same as the state 'from', but also that your contact can advertise you and your presence to any of her contacts.

subscription='both-anyone': You and the contact are interested in each other's presence information and can advertise it to anyone.

subscription='both-contacts': You and the contact are interested in each other's presence information and can advertise it to your own and your contact's contacts.

Adding more new subscription statuses such as advertising presence only to certain server is possible, and instead of using new subscription statuses, new fields can be used to tell about these privacy wishes.

The SD protocol uses one-way RPC. It is a variant of asynchronous RPC in which the client continues immediately after sending the request to the server without waiting for the server's acknowledgement [33]. The SD protocol messages are formatted as JSON-RPC [34]. JSON-RPC is a remote procedure call protocol following the same principles as XML-RPC [35]. JSON-RPC's "notification" provided asynchronous RPC for the SD protocol. Notification is a special request which does not have a response, it has same properties as request object except the id must be null. In the prototype, the SD messages are transmitted and processed as XMPP Instant Messaging (IM) messages in XMPP clients.

Some XMPP clients might have service administration [29] or roster-item exchange [30] support, and they can be used with the SD protocol. For instance, if there is an entity authorized to get all rosters from the server, it can give more comprehensive answers for queries.

The SD protocol's JSON-RPC messages' params field has to include at least information about the searched string. A forwardedParseRoster method call's params field includes also information about the original sender. This enables XMPP clients to reply directly to the original parseRoster message sender. Examples of JSON-RPC formatted notification messages are presented in Table 1.

TABLE 1. JSON-RPC FORMATTED NOTIFICATION MESSAGES

{"method": "parseRoster", "params": [{"value": "weather"}], "id": null}
{"method": "forwardedParseRoster", "params": [{"value": "weather", "originalsender": "tempsensor.313@vtt.fi/kaitovayla1"}], "id": null}
{"method": "reply", params": [{"value": weatherservice@vtt.fi"}], "id": null}

In a pseudo code, the SD protocol works as presented in Table 2.

TABLE 2. THE SD PROTOCOL IN PSEUDO CODE

STRING wanted_service; XMPP ROSTER own_roster; JSON-RPC MESSAGES parseRoster, forwardedParseRoster, reply;
IF (own_roster includes wanted_service) Service is discovered; ELSE Send parseRoster request to selected contacts in own_roster;

IF (proper reply is received) {
IF (reply includes wanted_service) Service is discovered;
}

IF(proper parseRoster function request is received) {
IF (own_roster includes wanted_service) Service is discovered and reply is sent to the requester.
ELSE Send forwardedParseRoster request to selected contacts in own_roster;
}

IF (proper forwardedParseRoster function request is received) {
IF (own_roster includes wanted_service) Service is discovered and reply is sent to the original requester;
}

IV. EVALUATION

The SD protocol was tested in different use cases with 1-3 modified XMPP clients. Each client had 1-3 XMPP accounts (JIDs) registered to 1-3 XMPP servers. Three unmodified XMPP clients were used for debugging.

A. *Hop limit*

This section presents two use cases used to test the SD protocol and to get information about the hop limit. Figures 1-2 present use cases, where users A, B, C, D, and E are M2M overlay entities, and boxes next to them represent contacts in their rosters. Each JID has their own different M2M overlay (JIDs in their roster).

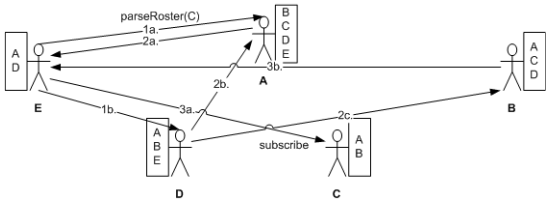

Figure 1. E tries to find C, hop limit is unlimited.

Figure 1 presents an example, in which E tries to find C, or a string that is in C's JID.

1. E sends a roster parsing requests to its contacts A (1a) and D (1b). It could be possible to send them only to the ones who are currently available, not e.g., in Do-Not-Disturb (DND) status.
2. A and D parse their own rosters. A's roster matches and it sends a reply (2a) to E. A and B have already done a presence subscription. Same way D checks its roster and it does not find C. It forwards the request to its contacts A (2b) and B (2c), except E, who is the original requester.
3. B receives the request from D, it parses its own roster, match is found, and it replies to E. The reply includes B's parsed roster and information about C. At this point B could also subscribe to E, and/or exchange roster item [30]. At the same time E already subscribes to C (3a), because it got information about C from A (2a).

Optimizing the hop limit is a complex problem. Using unlimited hop limit was not possible because without proper timeouts it can flood the network and jam devices. One requirement was that the protocol must be able to discover unknown entities from unknown M2M overlay entities. This means that the hop limit must be at least two. If the hop limit is one, the protocol enables discovering unknown entities only from known M2M overlay entities (your contacts).

If thinking about real world, sharing things with or borrowing them to your friends is usually ok. Section III presented an example of Eemil finding a fixer, in which a maximum of two hops was used. Then again, two hops might be too much because borrowing things to or sharing them with friends of you friends might be something most people do not want to do.

When moving these thoughts of human behavior from real-life to M2M and to the SD protocol, it was decided that a limit of two hops would be used: If a wanted M2M overlay entity is not found from contacts, the entity can ask it from its contacts (the first hop). If contacts do not know it, they can forward a message to ask the wanted entity from their contacts (the second hop). If they do not know it, the wanted M2M overlay entity is not found. Selecting two also keeps the protocol as simple as possible but still fulfills the discovery requirement.

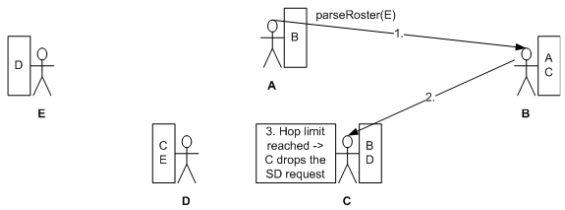

Figure 2. A tries to find E, hop limit is 2.

Figure 2 presents an example in which the M2M overlay entity E is not found, because of the hop limit is two, and because C does not know E. As it can be seen, if M2M overlay entities have only few contacts in their rosters, discovery process is short, may not succeed and only few messages are sent.

B. Replying mechanism

After selecting the hop limit, the amount of transmitted messages was calculated. In the worst case situation, the maximum amount of SD and forwarded SD messages (with considering neither XMPP nor TCP/IP acknowledgements etc.) can be calculated using (1)-(8).

M = *Maximum amount of messages.*
α = *Number of unique contacts in the roster of the SD request message sender A.*
β = *Number of unique contacts in the roster(s) of unique contact(s) of A.*

Case A: All XMPP clients are registered to the one and same XMPP server. Replies are sent directly to the original sender (1) or through the SD request forwarder (2):

$$M = 2\alpha + 4\alpha\beta. \quad (1)$$

$$M = 2\alpha + 6\alpha\beta. \quad (2)$$

Case B: The SD request sender XMPP client is the only client registered in the first XMPP server and two other XMPP clients are registered in the one and same XMPP server. Replies are sent directly to the original sender (3) or through the SD request forwarder (4):

$$M = 3\alpha + 5\alpha\beta. \quad (3)$$

$$M = 3\alpha + 7\alpha\beta. \quad (4)$$

Case C: 2 XMPP clients (the SD request sender and the first contact) are registered in the one and same XMPP server, and the third XMPP client in another XMPP server. Replies are sent directly to the original sender (5) or through the SD request forwarder (6):

$$M = 2\alpha + 6\alpha\beta. \quad (5)$$

$$M = 2\alpha + 8\alpha\beta. \quad (6)$$

Case D: Every XMPP client is registered in its own separate XMPP server. Replies are sent directly to the original sender (7) or through the SD request forwarder (8):

$$M = 3\alpha + 6\alpha\beta. \quad (7)$$

$$M = 3\alpha + 9\alpha\beta. \quad (8)$$

 ISBN: 978-1-61208-202-8

Using α=β=16 in (1)-(8) a chart in Figure 4 has been drawn. It presents total amount of SD messages (requests and responses). Y axis presents the amount of transmitted SD messages. X axis presents cases A, B, C, D and (1)-(8). In cases C and D the amount of SD messages are approximate when answering directly to the requester (5) and (7). In fact, (5)/(7) approaches 1 when α and β approach infinity.

The difference between direct replies and sending replies through the forwarder can be calculated by subtracting answers in each case A, B, or C. For example, when using three servers with α=32 and β=32 (7) gives M = 6240 and (8) gives M=9312, so the difference is 3072. When α and β approach infinity, result of (2)/(1) approaches the ratio 1,5. (4)/(3) approaches the ratio 1,4, (6)/(5) approaches the ratio 1+1/3 and (8)/(7) approaches the ratio 1,5.

If communication between clients and servers is not taken into consideration, in the simplest case when α=β=1, only three messages are transmitted: 1) from the SD request sender A to its contact B, 2) B forwarded the SD request to its contact C, and 3) C replies to A (9). If C would answer through B, there would be 4 messages instead (10). These are presented in Figure 5.

$$M = \alpha + 2\alpha\beta. \quad (9)$$

$$M = \alpha + 3\alpha\beta. \quad (10)$$

When α=β=1 and all clients are using only one server, the number of messages increases to 6 (1), with two servers to 8 (3) and (5) and with three servers to 9 (7) respectively. If the contact C would send the reply through B, numbers would be with one server 8 (2), with two 10 (4) and (6) and with three 12 (8). These cases are described in Figures 6-8. S1, S2, and S3 are servers.

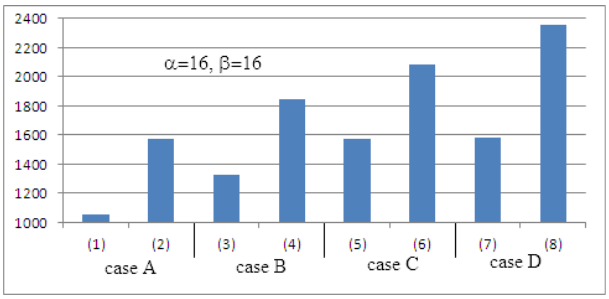

Figure 3. Number of SD messages, α=β=16.

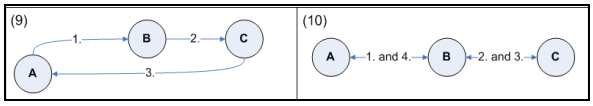

Figure 4. Number of SD messages, no servers, α=β=1.

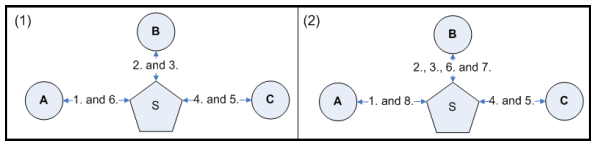

Figure 5. Number of SD messages, one server, α=β=1.

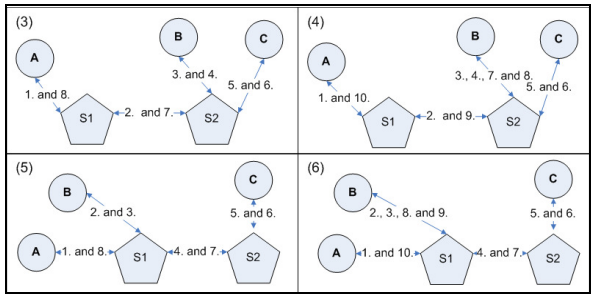

Figure 6. Number of SD message, two servers, α=β=1.

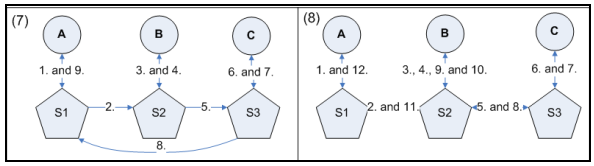

Figure 7. Number of SD messages, three servers, α=β=1.

To save messages and bandwidth, the first approach was selected: XMPP clients reply directly to the original SD requester.

C. *Answer handling*

When the M2M overlay entity tries to find strings that are too commonly used in JIDs, the SD request sender is likely to get answers including several JIDs, from several different senders. In some cases, this can also be thought of as an advantage if entities known by several other entities are thought to be more popular, and as such also more suitable. The SD request message sender could use this suitability information to categorize found JIDs. Currently the JID in the first reply is selected.

D. *Security*

XMPP offers security services for the M2M service network, but this section includes information about new security issues coming from the SD protocol. In one-way RPC the original sender cannot know for sure whether its request will be processed if the reliability is not guaranteed, but this also allows the requester to be able continue its work without the need to wait for the reply. Direct replies to the original SD requester generate threats, related to forging the original sender. This makes flooding and Denial of Service (DoS) attacks possible. Risk of the threats can be decreased in servers, by asymmetric cryptography and processing only certain messages (including proper information or coming from certain domain, for example). In real life, when answering directly to the original requester, request forwarders would not necessarily get information if the answer is found.

 ISBN: 978-1-61208-202-8

V. CONCLUSION AND FUTURE WORK

XMPP offers several building blocks, such as naming, rosters, message transmission and remote commands, pubsub, SD and security for M2M service networks, but no mechanism for finding roster entities from unknown XMPP entities. Therefore, this paper presented a simple M2M overlay discovery protocol for discovering XMPP JIDs behind several hops, in an XMPP based M2M service network. Hop limit was two, which was selected based on real life examples and in order to keep the protocol simple, but so that it also fulfilled the discovery requirements. The amounts of messages with synchronous or asynchronous RPC were analyzed. Asynchronous one-way RPC and direct replies to original requester were selected to decrease the amount of messages. Each entity or owner of the entity should be able to choose whether its presence and/or JID are shared to unknown entities, or not. The proposed four new XMPP subscription statuses enable describing when information can be advertised only to contacts or to anyone.

Future work includes applying some ideas of the paper to be submitted to XEPs. Mechanisms for handling different JIDs received in different SD replies must be designed and implemented. The SD protocol presented in this paper uses JSON-RPC formatting [34], but the format of the SD messages can be changed to Jabber-RPC format [35][36]. Distributed Hash Tables (DHT) could be used to enable serverless communication between XMPP nodes. Xeerkat [37] is one example implementation of a P2P computing framework that utilizes XMPP as a communication protocol.

ACKNOWLEDGMENT

Author wants to thank UseNet project [38] members and QXmpp [39] and ejabberd [40] developers. In the prototype, QXmpp XMPP client was modified, and Ejabberd was used as an XMPP server.

REFERENCES

[1] IPSO Alliance's webpage, http://www.ipso-alliance.org, 16.04.2012

[2] IETF RFC 6120, P.Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core", March 2011

[3] P. Saint-Andre, K. Smith, and R. Tronçon, "XMPP: The Definitive Guide, Building Real-Time Applications with Jabber Technologies", O'Reilly, 2009

[4] P. Saint-Andre, "XMPP: lessons learned from ten years of XML messaging," Communications Magazine, IEEE, vol. 47, no. 4, pp. 92-96, April 2009, doi: 10.1109/MCOM.2009.4907413

[5] M. R. Lavelle, (2010), "Micro-Grid Applications - Leveraging XMPP Short Messaging", http://www.lavelleenergy.com/documents/Micro-grid%20Applications%20Paper.pdf, 16.04.2012

[6] M. Kuna, H. Kolaric, I. Bojic, M. Kusek, and G. Jezic, "Android/OSGi-based Machine-to-Machine context-aware system," Telecommunications (ConTEL), Proceedings of the 2011 11th International Conference on, pp. 95-102, 15-17 June 2011

[7] QEES Open Software Platform website, http://qees.dk/da/services/open-software-platform, 16.04.2012

[8] T. Parkkila, (2005), "Application and platform management of an embedded system". Smart Systems 2005, Conference Proceedings. Seinäjoki, 3-4 May 2005.

[9] A. Hornsby, P. Belimpasakis, and I. Defee, "XMPP-based wireless sensor network and its integration into the extended home environment," Consumer Electronics, 2009. ISCE '09. IEEE 13th International Symposium on, pp. 794-797, 25-28 May 2009, doi: 10.1109/ISCE.2009.5156807

[10] A. Rowe, M. Berges, G. Bhatia, E. Goldman, R. Rajkumar, L. Soibelman, J Garrett, and J. M. F. Moura, "Sensor Andrew: Large-Scale Campus-Wide Sensing and Actuation", Carnegie Mellon University, 2008

[11] xbee-xmpp website, http://code.google.com/p/xbee-xmpp/, 16.04.2012

[12] A. Hornsby and E. Bail, "µXMPP: Lightweight implementation for low power operating system Contiki," Ultra Modern Telecommunications & Workshops, 2009. ICUMT '09. International Conference on, pp. 1-5, 12-14 Oct. 2009, doi: 10.1109/ICUMT.2009.5345594A.

[13] P. Saint-Andre, "XEP-xxxx: Sensor-Over-XMPP", XEP proposal, V 0.0.18

[14] P. Millard, P. Saint-Andre, and R. Meijer, "XEP-0060: Publish-Subscribe", V 1.13

[15] IETF RFC 3923, P. Saint-Andre, "End-to-end Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP), October 2004

[16] P. Millard and P. Saint-Andre, "XEP-0016: Privacy Lists", V 1.6

[17] I. Paterson, D. Smith, P. Saint-Andre, and J. Moffitt, "XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH)", V 1.10

[18] I. Paterson and P. Saint-Andre, "XEP-0206: XMPP Over BOSH", V 1.3

[19] XML Binary Characterization Use Cases, http://www.w3.org/TR/xbc-use-cases/#xmpp, 16.04.2012

[20] P. Saint-Andre, "XEP-xxxx: Stream Compression with Efficient XML Interchange", XEP proposal, V 0.0.1

[21] D. Cridland, "XEP-0286: XMPP on Mobile Devices", V 0.1

[22] J. Hildebrand and P. Saint-Andre, "XEP-0138: Stream Compression", V 2.0

[23] J. Hildebrand, P. Millard, R. Eatmon, and P. Saint-Andre, "XEP-0030: Service Discovery", V 2.4

[24] P. Saint-Andre, "XEP-0174: Serverless Messaging", V 2.0

[25] P. Saint-Andre, "XEP-0045: Multi-User Chat", V 1.25

[26] Zero Configuration Networking (Zeroconf) webpage, http://www.zeroconf.org, 16.04.2012/

[27] S. Cheshire and M. Krochmal, "Multicast DNS", IETF Internet-Draft: draft-cheshire-dnsext-multicastdns-15, Dec 9, 2011, Expires: June 11, 2012.

[28] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery", IETF Internet-Draft: draft-cheshire-dnsext-dns-sd-11, Dec 9, 2011, Expires: June 11, 2012.

[29] P. Saint-Andre, "XEP-0133: Service Administration", V 1.1

[30] P. Saint-Andre, "XEP-0144: Roster Item Exchange", V 1.0

[31] Google Search Plus Your World website, http://www.google.com/insidesearch/plus.html, 16.04.2012

[32] US Patent 7890501, C. Lunt, N. Galbreath, & J., "Visual tags for search results generated from social network information"

[33] A. S. Tanenbaum and M. van Steen. 2006. "Distributed Systems: Principles and Paradigms (2nd Edition)". Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[34] JSON-RPC specification, http://json-rpc.org/wiki/specification, 16.04.2012

[35] XML-RPC specification, http://xmlrpc.scripting.com/spec.html, 16.04.2012

[36] DJ Adams, "XEP-0009: Jabber-RPC", V 2.2

[37] Xeerkat webpage, https://code.google.com/p/xeerkat/, 16.04.2012

[38] UseNet project website, https://usenet.erve.vtt.fi, 16.04.2012

[39] QXmpp website, https://code.google.com/p/qxmpp/, 16.04.2012

[40] ejabberd community site, http://www.ejabberd.im/, 16.04.2012

 ISBN: 978-1-61208-202-8