Session Mobility for Collaborative Pervasive Apps Using XMPP

István Koren
Advanced Community Information Systems (ACIS)
RWTH Aachen University
Ahornstr. 55, 52056 Aachen, Germany
koren@dbis.rwth-aachen.de

Daniel Schuster, Thomas Springer
Computer Networks Group
Technische Universität Dresden
Dresden, Germany
{*daniel.schuster*|*thomas.springer*}*@tu-dresden.de*

***Abstract*—In pervasive computing scenarios, user sessions often involve multiple devices like smartphones, tablets, large public displays and smart objects. In such settings, session mobility allows users to split sessions across several devices, to dynamically include or remove devices from sessions and to handover sessions from one device to another. While there are some good SIP-based solutions for session mobility in ubiquitous audio/video scenarios, there is yet no working approach for collaborative apps with event-like communication patterns. We describe and evaluate a solution for session mobility based on XMPP offering discovery of personal devices, discovery and authorization of public devices, and ad-hoc discovery of smart objects. All this can be done with little or no modification of existing XMPP technology, thus creating a standard-based toolset for session mobility. This eases development of collaborative apps and finally allows users to seamlessly continue their task in pervasive settings.**

***Keywords*-pervasive computing; mobile computing; task continuity; session mobility; session transfer; multi-device sessions; XMPP**

I. Introduction

In pervasive computing scenarios, users can profit from the ability to dynamically involve various types of devices like smartphones, tablets and large public displays to fulfill their tasks. This user- and task-centric perspective calls for rather device-transparent applications, or so-called pervasive apps.

A prominent use case for pervasive apps are collaborative location based games where users may catch up certain goodies at their current location. Users may seamlessly interact with a public display in front of or inside a building using their smartphones while others are taking part of the game using a Web application on their tablets. Another scenario is the use of a Smart TV to display the pile of discards in a collaborative card game.

As can be seen from these and other examples, collaborative pervasive apps call for a new level of session mobility. Session data has to be transferred instantly from one device to another. Multiple devices may be used in a split session in parallel (especially with public displays). All this should work with one-touch configuration for the user as the smartphone is the primary and coordinating device for such apps.

There are already quite good solutions for mobility of audio/video sessions using the Session Initiation Protocol (SIP) [1]. But these methods were designed for data streams with continuous characteristics. In contrast, the communication patterns of collaborative pervasive apps follow a fundamentally different approach as they are rather event-based with a large number of small messages (location updates, shared object modifications, etc.).

As the eXtensible Messaging and Presence Protocol (XMPP) is explicitly designed for exchanging short XML-structured messages in real-time on a global scale, it appears to be a quite good candidate to realize session mobility for such collaborative apps. We suggest a concept based on XMPP core technology and current community extensions that provides unified ways to identify other devices, move sessions across them and adapt communication flows.

Our main contribution is thus developer support for session mobility based on XMPP. This includes methods for discovery of personal devices, actual transfer of sessions, split sessions with public displays and ad-hoc discovery of smart objects. The main focus is on collaborative pervasive apps, but other pervasive apps without collaborative functionality but similar communication patterns may profit from these findings, too. While we implemented and evaluated our approach inside the Mobilis framework, it could be easily applied to any existing or newly developed pervasive apps. Our approach only requires an XMPP backend and appropriate XMPP client libraries. Both are freely available for heterogeneous platforms including smart objects.

II. Requirements and Related Work

Session mobility is a rather overloaded term defining such diverse things as operating system migration, IP mobility protocols, multimedia session mobility, or application migration. Our work on collaboration session mobility is most closely related to multimedia session mobility as defined in [2]. We share the notion of a permanent connection or bi-directional data stream between different devices that has to be transferred to a new device. The difference lies in the communication patterns which call for slightly different requirements.

A. Requirements

[2] defines *Device Discovery* and the actual session mobility as the two essentials of multimedia session mobility. Session mobility is further distinguished in moving a session to a different device (*Session Transfer*) and running a session split over multiple devices (*Session Split*).

We rely on these three basic requirements with some modifications. Firstly, while for audio and video communication it is often sufficient to tear down the communication link and to reestablish it on another device, collaborative apps have to transfer their shared application state. Hereby, previously exchanged artifacts like messages, shared objects or files need to be serialized.

Secondly, we add a fourth requirement which originates in the communication patterns of collaborative apps. If run in a split session, a collaborative app may show different aspects of the collaboration workspace. E.g., the collaborative game mentioned in the introduction may show the map on a public display while showing the member list and chat on the mobile device. Such split scenarios in server based settings call for packet filtering to only route the messages actually needed to the respective device.

B. Session Mobility with SIP

As already mentioned above, SIP inherently offers a session transfer procedure to migrate an active session from one device to another. After having established an RTP session between two devices, one of these endpoints may send a REFER message to its communication partner that contains a new SIP URI. In return, the partner may then trigger an INVITE to the new endpoint to initiate a new session between them. Finally, the connection to the original client is closed.

Besides the built-in session mobility features, further SIP extensions have been presented for more advanced scenarios. CINEMA [1], Mobility Header [3], and Multi-Device System [4] are systems and approaches dealing with SIP-based split sessions across multiple devices. Session split can only be provided by extending SIP with proprietary client and server extensions or using a proxy for all communication. The latest result of these efforts is the informational RFC 5631 [2], defining the current state of the art in SIP-based Session Mobility including split sessions. But device discovery is outside the scope of this specification.

C. Single User Application Session Migration

In contrast to pure multimedia session transfer solutions, TWIST [5] is an architecture that inserts a middleware layer into conventional client/server systems. The main goal of this layer is to enrich existing software with a session transfer technology. Sessions can only be transferred as a whole including all application data.

Deep Shot [6] is a framework for capturing a user's work state on a desktop client to move and continue it on a mobile device. To initiate the session transfer, a picture of the display content has to be shot with the target device. The framework then analyses the application on the screenshot that in turn is asked to bundle the current state to a URI that gets sent to the target device. The system's message channel is based on XMPP; all information including the screenshot is transferred in-stream.

Table I
SESSION MOBILITY FEATURES OF RELATED WORKS

	Device Discovery	Session Transfer	Session Split	Packet Filtering
XMPP	–	–	✓	✓
SIP	–	✓	–	–
CINEMA [1]	✓	✓	–	–
Mobility-Header [3], Multi-Device System [4], RFC 5631 [2]	–	✓	✓	–
TWIST [5], Deep Shot [6]	✓	✓	–	–
Session Mobility with XMPP	✓	✓	✓	✓

D. Discussion

Table I shows the requirements compared to what existing approaches already deliver. XMPP supports session split natively as users may login to an XMPP server with different resources like *user1@xmpp.org/iphone* and *user1@xmpp.org/pc* simultaneously. XMPP servers can be configured to send messages to both resources if the Bare JID *user1@xmpp.org* is used. Packet filtering is supported by extension XEP-0273 [7].

SIP on the other hand works quite good for session transfer (with the REFER method discussed above), but other functionality such as device discovery and session split has to be added by proprietary extensions like in the SIP-based works discussed above. Approaches for single user application session migration offer ideas for device discovery and session transfer but fail on session split and packet filtering.

As can be seen in the last row of Table I, our approach combines the features of existing session mobility approaches with the discovery and transfer features of session migration approaches. As XMPP is yet missing appropriate device discovery and session transfer functionality, we focus on these two requirements in Section IV and Section V. The overall architecture fulfilling all the requirements mentioned above is depicted in the following section.

III. SESSION MOBILITY ARCHITECTURE

The architecture of the session mobility system is portrayed in Figure 1. We support native clients like Android or desktop Java clients, HTTP-based clients like the Kiosk

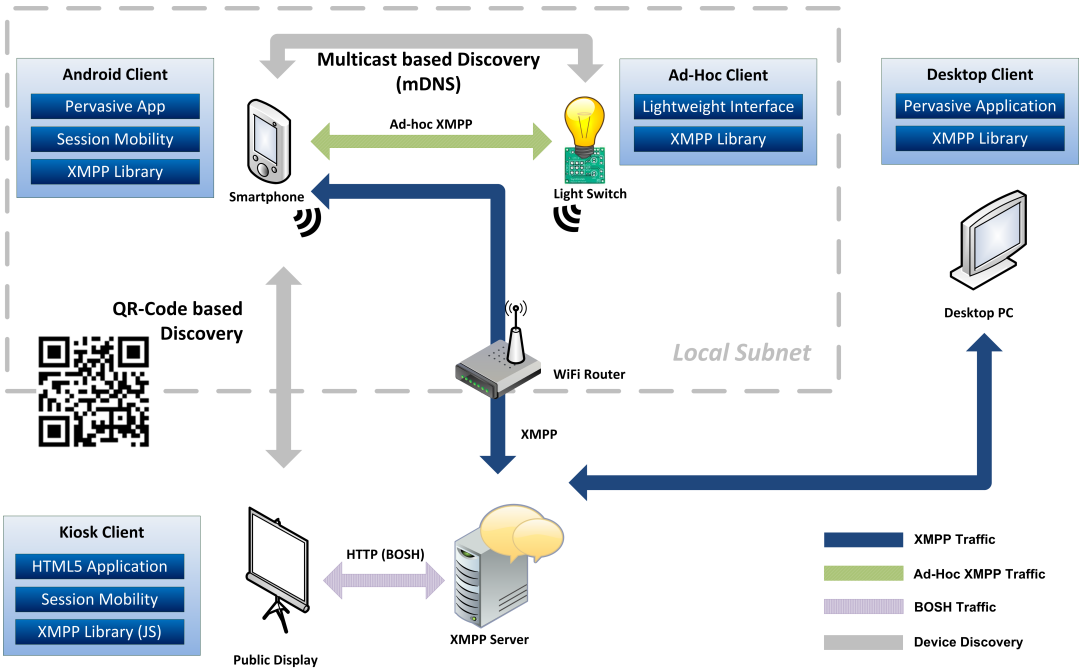

Figure 1. Session Mobility Architecture using XMPP

client (public display), and ad-hoc clients represented as the smart light switch. Native clients directly connect to an XMPP server and communicate to each other via XMPP. This works across network borders and even on a global scale by XMPP inter-server communication.

HTTP clients use a BOSH proxy for communication. BOSH stands for Bidirectional Streams over Synchronous HTTP and is standardized as XEP-0124 [8]. Ad-hoc clients advertise presence via DNS service discovery. Clients wishing to connect to such a client need to be in the same subnet.

Discovering devices for session transfer and split sessions can be done between all different device types. Based on the affiliation of devices to users and networks we consider three modes of device discovery which will be discussed in the following.

IV. Device Discovery

The session mobility system offers the following methods for device discovery: Personal device discovery for transferring a session to another device of the same user, kiosk mode device discovery for moving a session to a publicly available device and ad-hoc mode device discovery for seamlessly integrating devices not directly connected to an XMPP server.

A. Personal Device Discovery

As stated above, the XMPP standard inherently provides the possibility to login on multiple endpoints with identical user credentials. For discovering personal devices, we take advantage of this concept as a user may connect all his devices to the XMPP server with a single username/password combination. Built-in functionality of XMPP allows to query a list of parallel resources to let the user select a target device. Alternatively, a QR code may be displayed on the originating endpoint that encodes its JID. A target device may then scan the code to initiate the session transfer negotiation. A similar scanning approach is possible with an NFC tag that reveals the endpoint's address. Generally, JID advertising means are limited by device capabilities.

B. Kiosk Mode Device Discovery

Kiosk mode device discovery enables public displays to be used for private user-based sessions. Kiosk devices are standalone devices available for public use, such as terminal computers in hotel lobbies and presentation hardware in meeting rooms. In a basic scenario, they could provide a login interface to enable users to enter their user credentials to start a session.

In our approach, we use a secure session transfer procedure that does not require passwords to be entered on kiosk devices. Figure 2 shows the authentication flow. A daemon on the Kiosk Client is connected to an XMPP server over a device-specific connection. Its JID may be advertised by a QR code or NFC tag as stated above, or a service repository similar to the location-based database described in [1].

When a user wants to take over the session, the Kiosk initiates a second XMPP connection with the requesting user's JID. In the following, the server's SASL challenge is forwarded by the Kiosk's inherent connection and answered on the user device. After the second server link has been

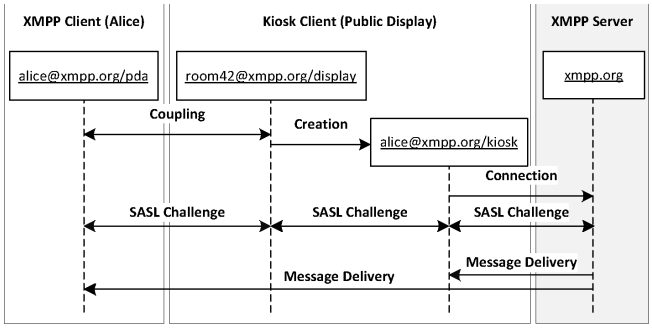

Figure 2. Kiosk mode authentification flow

successfully established, the device may keep the daemon's connection open or temporarily disconnect it while the user-specific session is running.

We have successfully implemented and tested the solution above in our proof-of-concept. In the meantime, as XMPP is a living standard, further scenarios realizing secure authorization for third-party clients have been presented. Google Talk and Microsoft Live Messenger now support XMPP authorization using OAuth2 [9] although an official XMPP extension protocol is not yet existing. The OAuth2 based login procedure is executed on the requesting device, typically by showing a web frontend with login screen and authorization checks. This solution is compatible with our approach; in our system the resulting token would be sent to the Kiosk Client to use it for the user-specific connection negotiation.

C. Ad-hoc Mode Device Discovery

Finally, the ad-hoc mode device discovery enables devices to be integrated into an application session that are not connected to an XMPP server but advertise their presence on a local network instead. The "Serverless Messaging" extension of XMPP (XEP-0174) is a standard allowing endpoints to communicate with each other without an intermediate server [10]. The concept is fundamentally based on the specifications of multicast DNS and DNS service discovery for performing autoconfigure queries in a local subnetwork.

To listen to entities supporting ad-hoc device discovery, their local mDNS daemon has to publish a DNS record to the multicast address 224.0.0.251 (or FF02::FB in IPv6 networks resp.). Furthermore, a DNS TXT record as in Listing 1 is published including a key-value based list of strings representing presence information and connection information such as the port the device is listening on.

Listing 1. A Multicast DNS TXT Record

```
jid=lightsensor@xmpp.org
port.p2pj=5562
status=avail
```

An entity that wishes to connect to another entity in the local network is now able to initiate a direct TCP connection. Subsequently, similar to a standard XMPP connection to a server two XML streams are opened, one in each direction.

V. Session Transfer

The session transfer takes place after a device discovery and establishment of a communication link between two devices; regardless of whether the packets are routed via an XMPP server or sent over a direct peer-to-peer connection.

A. Capturing Application State

As opposed to stream-based audio/video communication typically managed by SIP, event-based pervasive apps can have various types of application state. Instant messaging based applications require most of the previously received and sent packets, therefore a packet logging and replay mechanism is a possible means for collecting the current application state. This catch-all approach benefits of the packet-based character of XMPP. Other applications may only depend on the current snapshot; in the case of document sharing apps the latter is in fact the product of all delta packets before. Generally we offer a hybrid approach that lets the application developer decide what data the new device should be equipped with.

B. Transferring a Session

For transferring the session to another device, the application state has to be serialized. The actual session transfer is preceded by a transfer method negotiation that consists of an offer/answer model similar to SDP in SIP and the SASL authentication during the XMPP login phase. Currently, we offer the mechanisms 'in-band XML', 'file-based' and 'compressed bundle'. With in-band XMPP the application state is encoded as XML and transferred via a plain IQ message. The file-based mechanism uses XMPP extensions for file transfer to deliver the application state data files out-of-band via a direct TCP connection. The last mechanism is comparable to the file-based transfer except that all files are bundled in a single compressed ZIP file. As certain capabilities may not be feasible for every XMPP client, the app decides which mechanism to use.

C. Packet Filtering for Split Sessions

For efficient communication when certain parts of the application are running on another device, we propose to also split the communication at a central point when messages are routed over an XMPP server. This may be applicable for location-based instant messaging apps, where the chat history is shown on one device while another display shows the location of the user's contacts. Packet Filtering enables the communication endpoints to specify a list of packet filters that is sent to the server.

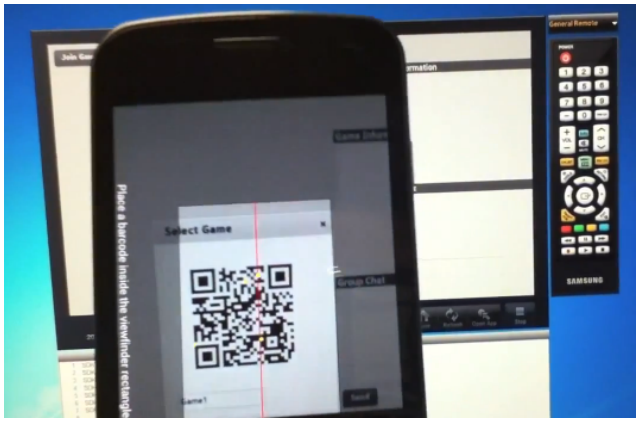

Figure 3. Discovering a Smart TV from a Mobile Client

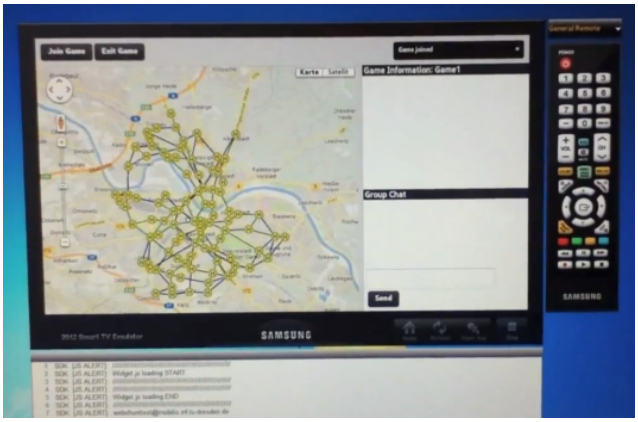

Figure 4. Successful Transfer from a Mobile Client

The ”Stanza Interception and Filtering Technology” (XEP-0273) [7] gives the XMPP endpoint fine-grained control over the delivery of inbound packets. The filtering is performed on the server upon rules that the client has to supply in advance. Packet-based filters in XMPP are filters applying to either IQ, Message or Presence packets or a combination of these. On the other hand, namespace-based filters leverage that all XMPP traffic is equipped with a namespace that identifies packets based on their functionality.

VI. Evaluation

To evaluate the session mobility approach, we present two show cases we implemented, one for kiosk mode and one for ad-hoc mode. The kiosk mode show case demonstrates all four requirements mentioned in Section II, while device discovery is solely based on a QR code. The ad-hoc mode prototype additionally shows the feasibility of ad-hoc device discovery.

Both show cases were implemented inside the Mobilis framework [11], which already offered an XMPP-based infrastructure to build collaborative Android and HTML5 apps. But as mentioned above, the approach can also easily be implemented without Mobilis using any combination of XMPP server and client libraries.

A. Kiosk Mode Evaluation

To demonstrate the Kiosk Client, we extended our existing urban location-based game XHunt [12] with a control system that shows the current position of all players on a map. The mobile client running on Android was extended with the possibility to invite public displays to the game that present the location of all players. The functionality is available as soon as an active game session was started.

The Kiosk Client itself is implemented as an HTML5 application running in state of the art desktop and mobile browsers. We employ Strophe.js as XMPP library. In conjunction with jQuery it offers convenient methods to build and send custom XMPP packets.

The web app is connected to the XMPP server over a dedicated device-specific login. During standby mode, the webpage shows a QR code that encodes its JID. After the mobile device performed the scanning and decoding of the bar code as shown in Figure 3, an IQ message is sent to the XHunt server application that in turn invites the Kiosk Client to the session.

HTML5 offers maximum portability to possible platforms for public displays. We implemented an app for the Samsung Smart TV platform (see Figure 3) which simply opens an HTML5 viewer capable of the discovery and game logic through JavaScript. Figure 4 shows the completed session transfer.

B. Ad-hoc Mode Evaluation

The ad-hoc stack for serverless XMPP communication was developed with the support of the jmDNS library. Our prototype includes a simple solution for both an ad-hoc client and server embedded into an Android application bundle. As described above, the server publishes a multicast DNS record on the local link. The client on the other side maintains a multicast daemon that listens to published records. We also provide a contact list with the IP addresses and ports of available ad-hoc devices.

The prototype application built upon the ad-hoc stack simulates a small lighting system on Android devices. It connects to all devices that publish an mDNS TXT presence record on the local network. An IQ request/reply interface allows to set the background color of the view.

In our trials we have discovered that the prototype runs very well in small home networks but fails in scenarios like a campus-wide wifi network. This resembles Apple’s problems with multicast DNS in larger university networks that have recently lead to an Internet-Draft at the IETF [13].

C. Discussion

The two show cases demonstrate the feasibility of XMPP-based session mobility, especially considering the four main requirements defined in Section II. The two missing points currently not yet fully integrated in XMPP were mechanisms for device discovery and session transfer.

Based on our experiments, device discovery based on QR codes seems to be the most intuitive and reliable mechanism from the user's point of view. QR codes shown on notebook displays or even a TV were recognized by mid-prized Android smartphones instantly without any errors. Within a second, the public display joined the session and the user was able to use the larger display to continue the game.

It even outperformed native discovery of personal devices based on XMPP service discovery. It was easier and faster for users to scan the QR code than to select the personal device based on XMPP service discovery. Ad-hoc mode discovery on the other hand takes a couple of seconds to proceed and as mentioned above does not work in all networks. Thus, we recommend ad-hoc mode discovery only as a fallback method in case QR code discovery is not available.

Besides the discovery aspect, we also did some initial experiments on state transfer with a chat client. On session transfer, chat history was sent to the new device and the user could instantly continue his session. But as this functionality is rather application-specific, we did not further evaluate it.

VII. Conclusion

We presented an architecture for session mobility of pervasive apps between mobile devices, stationary kiosk devices and ad-hoc smart objects. The system is built on XMPP and a subset of its extension protocols.

The main contributions of this work include developer support with mechanisms for device discovery and session transfer. Though solutions for SIP-based session transfer exist, they mainly allow for the transfer of audio/video streams and lack support for event-based interaction and data like message and file exchange.

As part of future work, we plan to fully demonstrate interoperability with the internet of things using real sensor and actor devices based on the lightweight XMPP implementation presented in [14]. Integrating ad-hoc with server based communication offers further opportunities for a detailed evaluation of the system.

Further work needs to be done on the problem of state transfer. To reach an application-generic solution, state transfer could be handled similar to late join or temporary connection failure events. If pervasive apps are designed to exchange application state on a publish-subscribe basis rather than a push-based protocol, they are able to easily transfer application state in session mobility situations.

Acknowledgment

The research work described in this paper is partially funded through the LAYERS FP7 ICT Integrated Project (grant agreement no 318209) of the European Union.

References

[1] S. Berger, H. Schulzrinne, S. Sidiroglou, and X. Wu, "Ubiquitous computing using SIP," in *Proc. NOSSDAV*, 2003.

[2] R. Shacham, H. Schulzrinne, S. Thakolsri, and W. Kellerer, "Session initiation protocol (SIP) session mobility," RFC 5631, IETF, 2009.

[3] M. Chen, C. Peng, and R. Hwang, "SSIP: Split a SIP session over multiple devices," *Computer Standards & Interfaces*, vol. 29, no. 5, pp. 531–545, 2007.

[4] R. Shacham, H. Schulzrinne, W. Kellerer, and S. Thakolsri, "An architecture for location-based service mobility using the SIP event model," in *Mobisys Workshop on Context Awareness*, 2004.

[5] T. Phan, R. Guy, J. Gu, and R. Bagrodia, "A new twist on mobile computing: Two-way interactive session transfer," in *Internet Applications, 2001. WIAPP 2001. Proceedings. The Second IEEE Workshop on*, 2001.

[6] T. Chang and Y. Li, "Deep shot: a framework for migrating tasks across devices using mobile phone cameras," in *Proceedings of the 2011 annual conference on Human factors in computing systems*, 2011.

[7] J. Hildebrand, J. Moffitt, and P. Saint-Andre, "Stanza interception and filtering technology (SIFT)," 2011.

[8] I. Paterson, D. Smith, P. Saint-Andre, and J. Moffitt, "XEP-0124: Bidirectional-streams over synchronous HTTP (BOSH)," http://xmpp.org/extensions/xep-0124.html, 2010.

[9] D. Hardt, "The OAuth 2.0 authorization protocol," http://tools.ietf.org/html/draft-ietf-oauth-v2-31, 2012.

[10] P. Saint-Andre, "XEP-0174: Serverless messaging," http://xmpp.org/extensions/xep-0174.html, 2008.

[11] A. Schill, D. Schuster, T. Springer, S. Bendel, and R. Lübke, "Mobilis - pervasive social computing using XMPP," http://mobilis.inf.tu-dresden.de, 2012.

[12] D. Schuster, D. Kiefner, R. Lübke, T. Springer, P. Bihler, and H. Mügge, "Step by step vs. catch me if you can - on the benefit of rounds in location-based games," in *3rd IEEE Workshop on Pervasive Collaboration and Social Networking (PerCol)*, Lugano, Switzerland, 2012.

[13] K. Lynn and S. Cheshire, "Requirements for DNS-SD/mDNS extensions," *IETF draft-lynn-mdnsext-requirements-00*, 2012.

[14] M. Kirsche and R. Klauck, "Unify to bridge gaps: Bringing XMPP into the internet of things," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, 2012.