Federated Access to Smart Objects Using XMPP

Daniel Schuster
Computer Networks group
TU Dresden, Germany
daniel.schuster@tu-dresden.de

Ronny Klauck and Michael Kirsche
Computer Networks group
BTU Cottbus - Senftenberg, Germany
(klaucron, kirscmic)@tu-cottbus.de

Dominik Renzel and István Koren
Advanced Community Information Systems group
RWTH Aachen University, Germany
(renzel, koren)@dbis.rwth-aachen.de

***Abstract*—Communication with smart objects currently only works in isolated, sometimes even proprietary islands. This limits the value of smart objects connected to the Internet of Things, as only a limited number of other devices may actually communicate with them. We propose to use the eXtensible Messaging and Presence Protocol (XMPP) to connect IoT islands as it is inherently federated and collaborative as well as secure and globally scalable. Our idea is to use XMPP Multi-User Chat (MUC) to build a secure and accessible platform for sensor data exchange between organizations. We demonstrate a scenario of three distributed and interconnected XMPP-driven sites accessing sensor data from all sites with different types of clients. The architectural pattern from this paper can easily be used in any XMPP-based system without the need to enhance or to extend the standards.**

I. Introduction

The value of the Internet of Things will only grow with the number of devices able to communicate to each other. This is related to Metcalfe's law, stating that the value of a network is proportional to the square of the number of devices connected to the system (n^2). While it is unrealistic assuming that each object is likely to communicate with each other one in the Internet of Things, its value will be at least proportional to something between n and n^2.

But unfortunately, there is currently no real *Inter*net of Things yet. Each smart object comes along with its own proprietary smartphone or tablet app which is needed to access its full functionality. This creates thousands of smart object islands isolated from each other. Furthermore, this "one-app-per-object" approach [1] does not even scale with the number of devices one single user would like to interact with.

Integrating diverse device types into a comprehensive net of smart objects thus remains a grand challenge. With particular regard to interoperability, a true net of things would allow users to access smart things and resources from any vendor with their mobile devices from a possible different vendor without considering how interconnection and communication eventually works.

What often comes into mind here is the analogy to the World Wide Web. But while the protocol to access information from websites is simple and efficient (HTTP), it only works if the device hosting the website is registered in the DNS, which takes some administrative effort. Thus a solution for interoperability of smart objects has to take both into account: Easy and decentralized discovery of objects as well as a way to securely access the information from the device.

We believe that the *eXtensible Messaging and Presence Protocol (XMPP)* is the right basis to realize this. It was explicitly designed as a platform, where users are registered and authenticated and can own an arbitrary number of devices which are able to be discovered and to communicate directly using short XML messages. It is thus the only open standards family available today that offers inherently extensible support for global-scale communication among different equivalent entities.

This work proposes a simple, light-weight approach: XMPP Multi-User Chat (MUC) can be used as an efficient way to model smart objects or places in the real world where sensors push information and receive commands. They are ready to use as they can be discovered and protected with fine-grained access control. Federated access is simple and can be restricted to certain domains. This approach is discussed in Section IV.

To demonstrate the feasibility of the solution, we implemented a scenario where we connected three sites which were formerly developed independently from each other. Applications at all three sites are able to discover and access sensor information using our approach. Technical details and lessons learned are described in Section V.

Before discussing our solution, we will first have a broader look at the problem and existing solutions in the next sections.

II. Problem Definition

The problem definition seems to be quite simple: Every device in the Internet of Things should be able to access every other device and retrieve information as well as send commands. When looking into detail at this task, there are many facets of it, which are not to be solved in a single piece of work.

First we have to differentiate the types of devices. Smart objects often send their data to cloud services or may be directly accessed by UI devices like smartphones, tablets, or notebooks. Thus, there are three main communication patterns that should be supported:

1) Any UI device is able to access information collected by cloud services from smart objects.
2) Any UI device is able to access smart objects directly.
3) Smart objects are able to identify and communicate to each other to coordinate.

The first level seems to be easily solvable by adding REST-like APIs to current cloud services. Fine-grained access control is possible, but has to be repeatedly set up for every provider of cloud-controlled smart objects. The information flow is n:1:m,

which means an arbitrary number of devices is able to access the information using an intermediate node (the cloud service).

The second level calls for easy discovery mechanisms, like showing all smart objects in a room or house. Access control is harder to achieve here, but still bound to a user. Communication flow is 1:n, i.e., multiple UI devices may access a smart object simultaneously.

For the third level, discovery seems to be the same than for level 2 while access control is harder as it loses the user metaphor. Smart objects can be both attached to a user as well as simply attached to the room or larger movable object they are in. Information flow can upgrade to n:m, i.e. each object possibly sending individual messages to each other object belonging to the same group (place).

Regarding the communication patterns, we typically expect a manageable amount of small messages to be exchanged at the last mile while the flow of sensor data can grow to large streams in between communication domains.

III. XMPP and the Internet of Things

This is the point where XMPP comes into play. While it was designed for Instant Messaging, the choice of XML as the message format makes it extensible to transport any type of small messages between nodes in near real-time.

A. Introducing XMPP

Messages are sent via an XMPP server to so-called JIDs (XMPP URIs). There are two variants of it: *user1@mydomain.org* is a Bare JID representing a user (normally a person). The URI *user1@mydomain.org/phone* is a Full JID representing an entity owned by the user.

Messages may now be sent either to a concrete entity (using the Full JID) or to a user (via the Bare JID). In the latter case, the actual receiving entity is determined by priorities which can be set during login phase.

XMPP as a distributed system is designed much like the e-mail system as there would usually be an XMPP server per organization as well as some independent servers which offer free or paid use of their services for private persons. These servers interconnect via DNS to relay messages to the target domain. Thus, XMPP is inherently capable of federated communication.

More information about XMPP can be found in [2] or at the website *xmpp.org*. An active community is constantly extending the capabilities of XMPP as well as providing free server and client implementations and libraries for a large number of platforms. Extensions are specified within and managed by the XMPP Standards Foundation (XSF). These extensions enable additional functionality like multi-user chat, file sharing, or serverless messaging. A full list of specifications can be found at *xmpp.org/extensions*.

B. IoT-related Activities

Part of these activities are a number of works to use XMPP in IoT scenarios. Current approaches can be split in two contrary strategies: (i) integrating smart objects and sensing devices with XMPP-enabled gateways and (ii) deploying XMPP itself on resource-constrained devices.

For (i), approaches like [3]–[5] use the Publish Subscribe extension of XMPP in a mediated way: non-constrained devices (e.g., desktop, smartphone, netbook) act as gateways who translate sensor-specific data into XML messages, while the smart object itself is not directly connected to the XMPP network.

[3] covers context management for sensors and introduces an XML privacy scheme to protect sensor data in XMPP networks. Gateways are used here to connect sensor devices. Sensor Andrew [5] is a large-scale framework to deploy sensors across Carnegie Mellon University and to prototype scalable applications. [4] integrates sensor data from Android smartphones directly in an XMPP network by deploying a smartphone-based XMPP client and by coupling XMPP and the Java-based OSGi framework to realize context-aware data processing in industrial M2M scenarios.

In addition, dedicated XMPP extensions for IoT scenarios are currently developed by Peter Waher [6]. They focus on large-scale M2M communication for industrial environments. Sensors are accessed using a Concentrator node that acts as an XMPP client. Fine-grained access control can be reached by involving a so-called Provisioning Server as part of the XMPP network.

Approaches of (i) break the Internet's end-to-end principle by relying on gateways. Approaches from (ii) therefore focus on the direct deployment of XMPP on constrained devices and the adaption of XMPP and its standard libraries for use in scenarios with resource-constrained devices.

A first step in this direction was μXMPP [7], which demonstrated that XMPP can run on constrained devices. μXMPP included a reduced set of XMPP core functionalities and enabled a direct communication between smart objects and non-constrained devices. Chatty Things [8] improves μXMPP with an API for developing IoT applications, support for essential extensions, optimizations for small RAM/ROM footprints and a reduction of XMPP message exchange. The custom XMPP extension Temporary Subscription for Presence (TSP) [8, Sec. VI] reduces the number and size of messages for joining a MUC room. The constrained object joining the MUC room only announces its presence and does not receive MUC presence vice versa.

Chatty Things first introduced the idea to send sensor data to an XMPP Multi-User Chat (MUC) room. This inherently simple concept is powerful enough to be extended to a concept for federated and discoverable smart object communication which we will show in the next section.

C. Discussion

As could be seen above, there is already some work regarding the use of XMPP in IoT scenarios. We can see the current approaches and the approach refined here as different levels for different application needs.

Thus we define the following levels of communication with smart objects in XMPP:

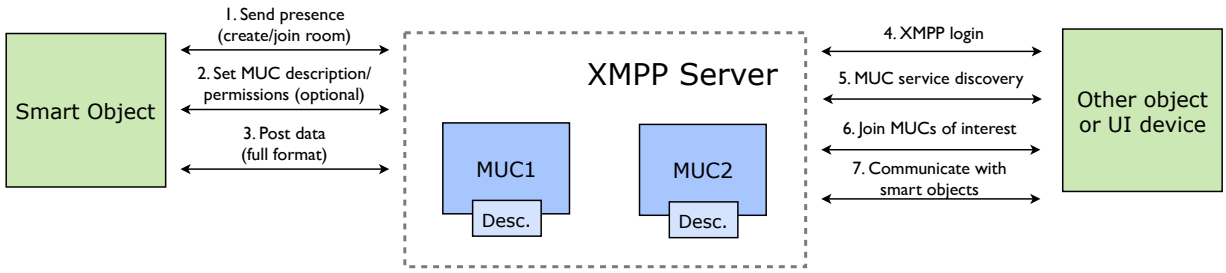

Fig. 1. MUC-based concept

1) Direct communication (discovery by other means).
2) Communication via the extensions Multi-User Chat and Service Discovery.
3) Communication via the Publish Subscribe extension (with integrated Service Discovery).

Each approach has its pros and drawbacks. If the JID of the device is already known, the communication can be realized using direct messages to the smart object either via the XMPP server or using the Serverless Messaging extension of XMPP if both devices are in the same wireless network. Koren [9] describes how the JID can be determined in a number of scenarios. If the user is the owner of both devices, XMPP presence already does the job quite well. If both devices are nearby, a QR code with the JID of the device can be attached to it or the JID can be transported via NFC.

The IoT extensions of Peter Waher discussed above also belong to this category as there is currently no means defined for discovery of smart objects.

Level 2 is the easiest approach if discovery is needed. Multi-User Chat and Service Discovery are part of all established XMPP libraries and can even be run on constrained devices as has been proven by Chatty Things. We refine Level 2 in this paper as shown in Section IV.

Level 3 adds more fine-grained discovery and notification mechanisms but based on our own experience it is not supported on all platforms properly. Furthermore, Publish Subscribe requires more complex communication and application logic. Thus there is additional effort needed in realizing a Publish-Subscribe-based solution, which should only be done if the additional features of Publish Subscribe are actually needed for the use case.

IV. MUC-BASED CONCEPT

Our concept of MUC-based sharing of sensor data can be seen in Figure 1. The XMPP server needs to have the extensions Multi-User Chat (XEP-0045) and Service Discovery (XEP-0030) enabled. This should be the case for almost all XMPP servers in productive use. MUC rooms each carry a description field which qualifies them as sensor MUC rooms. The description field is also already standardized in XMPP.

A. Discovery and Communication

A smart object announces itself by sending presence to a Multi-User Chat room. If this room doesn't exist yet, it is created by the MUC extension when receiving the presence stanza. Multiple smart objects may join the same room, so it is possible to create a room that represents a place or other real-world object with multiple sensors or smart objects.

The owner of the MUC room (usually the smart object that created it) is able to change the description as well as the permissions of the MUC room. The description is later used by other objects to discover the sensor MUC room. The permissions are used to restrict access to the MUC room on a coarse-grained (password protection) or fine-grained basis (both whitelisting and blacklisting of individual JIDs is possible).

Finally, the smart object sends its sensor data within the body of a message stanza to the MUC room. We propose to use JSON as data format as it enables easy parsing and simple representation of data structures. Mixing up XML and JSON seems to be strange in the first place but is totally reasonable as the XML markup works as the data envelope while the more native data format contains the actual payload. Once received, this payload string can be handed over to logging and other processing where no reference to any XMPP-related stuff is needed.

To be able to discover sensor MUC rooms at an XMPP server, an XMPP entity fist has to login to its XMPP server. It is now able to retrieve a list of MUC rooms at the XMPP server using the disco#items request of XEP-0030. It has to check whether the description of a MUC room marks it as a sensor MUC room and finally join the MUC rooms of interest. It will then not only receive the actual sensor data via messages but a small configurable window of past messages, too. This leverages the full capabilities of MUC rooms.

The discovery described above not only works for the XMPP server the user is connected to. It can also issue a Service Discovery request to another XMPP server thus asking for sensor MUC rooms. The same holds true for joining MUC rooms. Thus discovery and actual access of sensor data is possible crossing organizational borders.

B. JSON descriptions

We propose the following self-describing format for JSON descriptions of a MUC room:

```
{"sensormuc":
 {"type":"MULTI",
  "format":"full",
  "location":
```

```
   {"countryCode":"US",
    "cityName":"Pioneer Village",
    "latitude":40.98520,
    "longitude":-111.9020}}}
```

If a MUC room carries a JSON string in its description field with key = "sensormuc", it should be identified as a MUC room carrying sensor data. The type "MULTI" in the example states a MUC room that represents multiple sensors. In this case, the different sensors pushing values to the MUC room have to provide their type in each sensor event which can be seen in the next example:

```
{"sensorevent":
 {"type":"AMBIENT_TEMPERATURE",
  "values":[20.34432],
  "timestamp":"2013-09-18T18:31:38+01:00"}}
```

The format is based on the Android API for sensor events. For the sake of simplicity, we prefer such a format over more complex formats like SensorML.

If there is only one sensor per MUC room, the description looks like in the following example:

```
{"sensormuc":{"type":"AMBIENT_TEMPERATURE","
    format":"short","location":{"countryCode
    ":"DE","cityName":"Cottbus","latitude
    ":51.076834,"longitude":13.772586}}}
```

In this case, a shorter format for the actual sensor data can be used which does not contain the type of the sensor.

V. Example Solution - ACDSense

To evaluate the feasibility and federation capabilities of the solution described above, we developed an inter-organizational sensor scenario (called ACDSense) spanning our three universities, RTWH Aachen University, BTU Cottbus - Senftenberg and TU Dresden.

In the scenario, weather data is contributed at each site with different types of sensors and is consumed at each site with different types of clients. Thus, ACDSense enables different views on sensed data by supporting standard XMPP clients as well as XMPP-driven apps and websites.

As depicted in Figure 2, each organization runs its own XMPP server. The three servers are federation-enabled and thus accept incoming messages from other XMPP servers. They are able to route outgoing messages by locating the target XMPP server via DNS. In the following, we describe the three subsystems which are all able to provide sensor data to the other subsystems and to discover and read data sources from other organizations as well.

A. High-Performance Sensors and Dashboard in Aachen

The sensor part at RWTH Aachen University represents a class of low-cost, high-performance sensors. It is implemented using a commercially available Raspberry Pi single-board computer with an affiliated USB thermometer and automatic WLAN and XMPP connections to a sensor MUC room established at boot time. On system startup, the sensor's main script running in the Python runtime environment connects to a previously defined XMPP server, enters a configured MUC sensor room, and then periodically fetches data from the thermometer and pushes JSON sensor event payloads into the MUC room.

The second component of the RWTH Aachen University ACDSense infrastructure is a collaborative Web application providing a well-defined collaboration context for technician communities to jointly monitor sensors and coordinate support action, e.g. replacement of broken sensors. The application is built on top of the ROLE SDK (cf. [10]), providing so called spaces as collaboration contexts. For supporting the ACDSense scenario, we built a set of three widgets and deployed them to a space (cf. Figure 2, lower left screenshot).

B. Mobilis-based Sensing Apps in Dresden

The second subsystem was developed at TU Dresden and realizes a native sensing app (iOS) on top of the Mobilis framework [11] with basic sensor discovery features. Mobilis enables to write compounds of native mobile apps and cloud services which are dynamically deployed in a runtime environment. The communication is based on XMPP and thus compatible with other XMPP-based approaches.

The actual sensor data is accessed via an iPad App written in Objective C, which is shown in the lower middle of Figure 2. The app allows to browse sensor domains and select sensors of interest within each domain. All data is provided by the Sensor Service.

The second ACDSense component at the Dresden site is the Sensor Data Generator. This component was built to test the sensing infrastructure with a large number of sensor data. It processes historical hurricane weather data from 5000 US weather stations available at [12] and sends it to multiple MUC rooms. One MUC room is used per weather station while there are always multiple sensor types per MUC room like wind, humidity, or ambient temperature.

C. Integration of Constrained Devices in Cottbus

The implementation at Cottbus shows the integration of constrained devices using the Chatty Things approach with the XMPP extension Temporary Subscription for Presence (TSP) in our architecture. Zolertia Z1 nodes running a minimized and modular XMPP stack on the Contiki OS, push their sensor data directly to MUC rooms of an XMPP server.

The Chatty Things use TSP to avoid presence overhead when joining a MUC room. To further reduce the XMPP message size, redundant data is avoided as meta data is only transmitted once. The topic of the room represents the type of the sensor, the first message after joining the MUC room transmits the geo-location and the unit of the sensed data while all other messages only transmit the sensed value (i.e., short format). The remote commands `'int++'` and `'int--'`, sent via a chat message, allow users to interactively adjust the interval between the pushes of sensor data by 10 seconds.

The information can be accessed by ordinary XMPP clients like Adium or Pidgin. The rightmost lower screenshot in Figure 2 shows the output of a sensor MUC room. The information is human-readable and thus no additional software is necessary to monitor Chatty Things sensors if the JIDs of all sensor MUC rooms are known in advance.

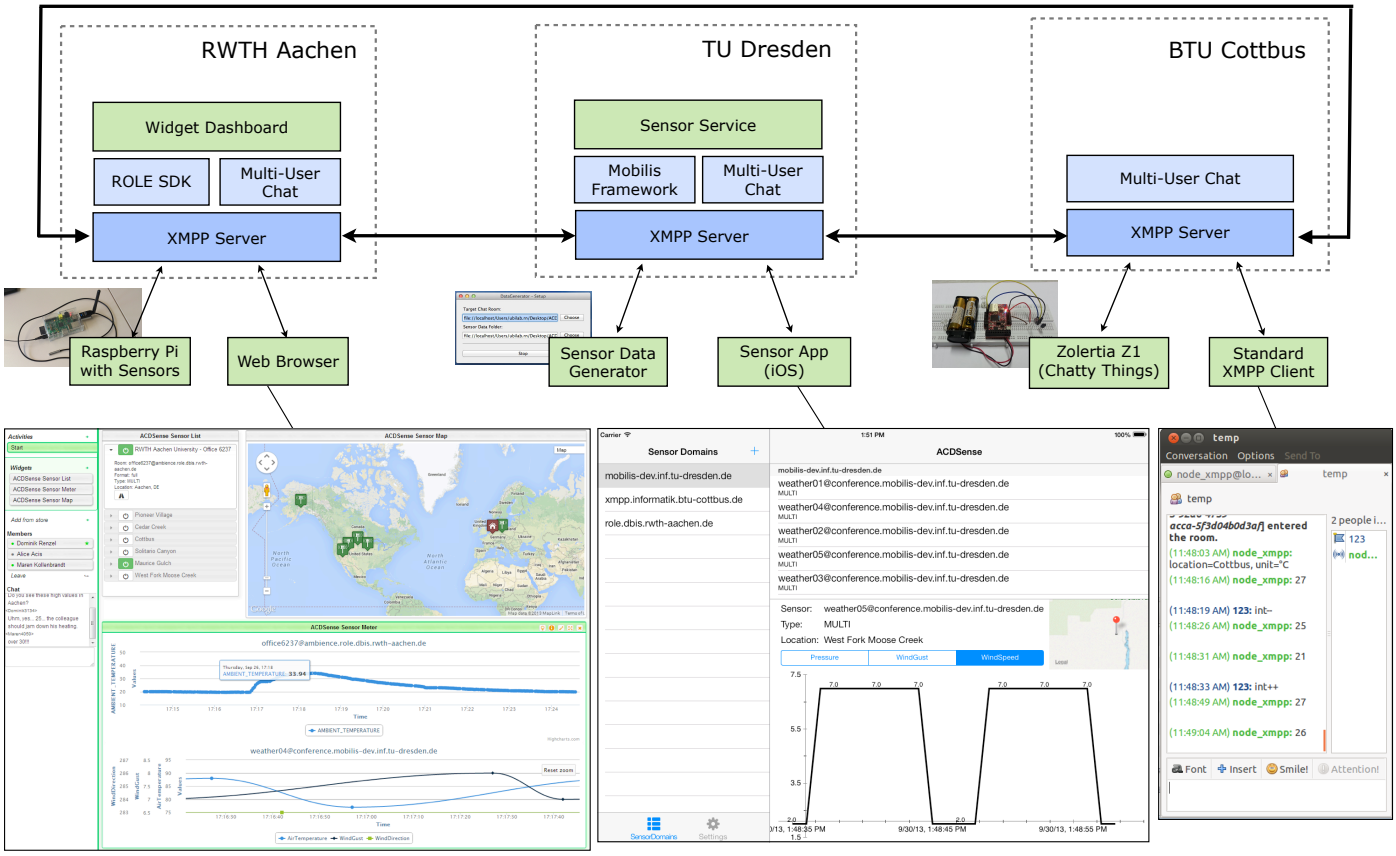

Fig. 2. Architecture and screenshots of example scenario ACDSense

D. Discussion

The whole setup described in Section V was realized within a time period of four weeks and tested over several weeks. Besides developing the actual components, there were administrative challenges inherent of the XMPP design to realize the inter-organizational communication.

First, we needed to configure the XMPP servers to accept server-to-server connections. As these connections run over port `5269` and not the standard XMPP client port (`5222`), a new firewall filter had to be added at all three sites. In addition, a second DNS SRV record (`_xmpp-server._tcp`) per XMPP server is needed in theory. In practice, most self-hosted XMPP servers use their URL as their XMPP service domain. In this case, an ordinary DNS A or AAAA record is sufficient for clients/servers to connect to the XMPP server, which worked for our setup.

Another challenge emerged from the use of the Multi-User Chat extension. It uses its own sub-domain like `conference.xmpp-server.org`. As administrators normally forbid to resolve all sub-domains with a single DNS A record, the MUC sub-domain needs to be added as a DNS alias as well. If it is not possible to modify DNS entries for administrative reasons, a simple workaround is to modify the `etc/hosts` file of each machine taking part in the federation. This worked well for our small federation scenario described above. Large scenarios certainly require DNS entries for each server and MUC sub-domain.

Fixing these administrative issues and sticking to the JSON formats described above was sufficient to enable inter-organizational sharing of sensor data. Each of the three clients running at its own site was able to discover and see the sensor data from all three sites.

We thus proved that our concept is indeed capable of realizing lightweight infrastructures for federated communication in the Internet of Things based on XMPP. Other systems currently not running XMPP may be easily extended as there are a number of XMPP libraries available for all sorts of platforms.

VI. Non-XMPP Approaches

XMPP is only one of the main competitors to act as the discovery and access protocol for the Internet of Things. The other main approaches are REST-like communication (HTTP/CoAP) and MQTT.

The intuitive approach for accessing sensors is to use REST-like HTTP GET requests and return the value of the sensor in JSON or pure text format. While this is one of the most prominent approaches, the smart objects need to provide a mini web server or need a proxy doing this on their behalf. A new TCP connection needs to be established and closed for each request. XMPP requires the node only to run a client library opening an outgoing TCP connection to its XMPP server. This connection is reused for many requests and responses.

The Constrained Applications Protocol (CoAP) [13], [14]

defines a UDP-based access protocol for smart objects, especially constrained devices with approximately 10 Kbytes of RAM and roughly 100 KBytes of code space (class-1 devices). It uses HTTP-like pull semantics to access information on smart objects, but request and responses are transported using a compact encoding on top of UDP. Thus, smart objects do not need to manage TCP connections nor an HTTP stack. Push semantics can be reached by setting the Observer option, where the smart object will continue sending results to the requester as soon as the requested value changes. Discovery of sensors has to be done in a Web-like way by requesting a well-known URI (`GET /.well-known/core`) offering a collection of links.

While both HTTP and CoAP offer cross-domain access based on the DNS, access control has to be handled by other means. XMPP offers the advantage that all users are authenticated and MUC rooms can be used to do fine-grained access control. Furthermore, discovery of resources is more flexible with our MUC-based approach as the smart object itself is able to modify the registered information by managing the MUC room. Regarding the focus on constrained devices, Chatty Things has proven that a "lightweight" XMPP is even able to run on such class-1 devices. Finally, one advantage of our approach is the simple support of multiple users accessing the same sensor information. The sensor only needs to push this once to the MUC room regardless of the number of receivers.

MQTT [15] is a publish-subscribe architecture where clients connect via TCP to a Broker and subscribe to topics. A hierarchical topic scheme is used. Sensors may be modeled as topics and thus distribute their data. MQTT uses an efficient binary message format and offers different quality levels for message delivery (at-most-once, at-least-once, exactly-once). A modified version designed for wireless sensor networks is available as MQTT-SN [16] and uses UDP instead of TCP as well as shorter topic identifiers to optimize the use on constrained devices.

While MQTT is the most efficient approach especially in one-to-many scenarios, it lacks federation capabilities. MQTT and other PubSub architectures are most useful in isolated application scenarios, where one central message broker is sufficient and federation not necessary.

VII. Conclusions

We have shown a concept and showcase of inter-organizational data sharing in the Internet of Things using XMPP. Our MUC-based concept complements other approaches and is especially easy to implement and deploy.

While our small showcase was sufficient to show the feasibility of the approach, it would be interesting to compare the different approaches (XMPP, CoAP, MQTT) on a quantitative level. This requires a large experimental setup with different types of sensor data logs for different usage scenarios as well as a working implementation of each approach. We plan to realize this in future work.

But whatever the results will be, it is most likely that we will see at least these three approaches in parallel in future IoT systems or even a mixture of approaches in the same system. E.g., [17] tries to bring together REST and MQTT using a Broker powered by a No-SQL database. Such pragmatic solutions will power the Internet of Things of tomorrow enabling interoperation among communication concepts where it is necessary or advantageous.

Acknowledgements

The authors wish to thank Markus Wutzler and Martin Weissbach for implementing the TU Dresden part of ACD-Sense.

References

[1] S. Jensen, "Mobile Apps Must Die," http://jenson.org/mobile-apps-must-die, 2011.

[2] P. Saint-Andre, "XMPP: Lessons Learned from Ten Years of XML Messaging," *IEEE Communications Magazine*, vol. 47, no. 4, pp. 92–96, 2009.

[3] Z. Jaroucheh, X. Liu, and S. Smith, "An Approach to Domain-based Scalable Context Management Architecture in Pervasive Environments," *Springer Personal and Ubiquitous Computing Journal*, pp. 1–15, 2012.

[4] M. Kuna, H. Kolaric, I. Bojic, M. Kusek, and G. Jezic, "Android/OSGi-based Machine-to-Machine Context-aware System," in *Proc. of the 11th IEEE Conference on Telecommunications (ConTEL)*, 2011, pp. 95–102.

[5] A. Rowe, M. Berges, G. Bhatia, E. Goldman, R. Rajkumar, J. Garrett, J. Moura, and L. Soibelman, "Sensor Andrew: Large-scale Campus-wide Sensing and Actuation," *IBM Journal of Research and Development*, vol. 55, no. 1.2, pp. 6:1–6:14, 2011.

[6] XMPP Standards Foundation, "Tech pages/IoT XepsExplained," http://wiki.xmpp.org/web/Tech%20pages/IoT%20XepsExplained, January 2014.

[7] A. Hornsby and E. Bail, "μXMPP: Lightweight Implementation for Low Power Operating System Contiki," in *IEEE Conference on Ultra Modern Telecommunications & Workshops (ICUMT)*, 2009, pp. 1–5.

[8] R. Klauck and M. Kirsche, "Chatty Things - Making the Internet of Things Readily Usable for the Masses with XMPP," in *Proceedings of the 8th Conference on Collaborative Computing: Networking, Applications & Worksharing (CollaborateCom)*. IEEE, Nov. 2012.

[9] I. Koren, D. Schuster, and T. Springer, "Session Mobility for Collaborative Pervasive Apps Using XMPP," in *Proc. of the 4th IEEE Workshop on Pervasive Collaboration and Social Networking (PerCol)*, 2013.

[10] D. Kovachev, D. Renzel, P. Nicolaescu, and R. Klamma, "DireWolf - Distributing and Migrating User Interfaces for Widget-Based Web Applications," in *Proc. of the 13th Int. Conference on Web Engineering (ICWE)*, ser. LNCS, no. 7977, 2013, pp. 99–113.

[11] D. Schuster, R. Lübke, T. Springer, and A. Schill, "Mobilis - Comprehensive Developer Support for Building Pervasive Social Computing Applications," in *Networked Systems (NetSys)*, Stuttgart, Germany, 2013.

[12] Kno.e.sis Project Consortium, "Linked Sensor Data," [Online] http://wiki.knoesis.org/index.php/LinkedSensorData, 2010.

[13] Z. Shelby, K. Hartke, and C. Bormann, "Constrained Application Protocol (CoAP)," IETF, Internet-Draft, June 2013. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-coap/

[14] C. Bormann, A. P. Castellani, and Z. Shelby, "CoAP: An application protocol for billions of tiny internet nodes," *Internet Computing, IEEE*, vol. 16, no. 2, pp. 62–67, 2012.

[15] Eurotech, International Business Machines Corporation (IBM), "MQTT v3.1 Protocol Specification," http://www.ibm.com/developerworks/webservices/library/ws-mqtt/, 2010.

[16] A. Stanford-Clark and H. L. Truong, "MQTT for sensor networks (MQTT-SN) protocol specification version 1.2," http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf, 2013.

[17] M. Collina, G. E. Corazza, and A. Vanelli-Coralli, "Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST," in *Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on*. IEEE, 2012, pp. 36–41.