XMPP - The Heartbeat of Global-scale Pervasive Computing

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*—Modern pervasive computing has extended its scope to include interaction with smart objects, cloud services and other people as well. Realizing seamless communication among all mentioned entities on a global scale is the next big frontier in pervasive computing. In the last years, the eXtensible Messaging and Presence Protocol (XMPP) and its numerous extensions have gained momentum as a multi-purpose lightweight middleware for social computing, multi-device interaction, and communication with sensors. This paper surveys the current state of research and standardization in the field of XMPP-empowered pervasive computing. We demonstrate a proof of concept with a use case of three distributed and interconnected XMPP-driven pervasive systems.**

I. Introduction

The vision of pervasive computing empowers people to use computing and networking resources anywhere and anytime. It was originally described by Mark Weiser [1] in 1991 as an experience *"refreshing as taking a walk in the woods"*. When Saha and Mukherjee [2] revisited Weiser's vision in 2003, major technology advances were still required at four levels: devices, networking, middleware, and applications.

Ten years later, we review the state of technology again and evaluate what parts required for global-scale pervasive computing are available and what issues remain open. In the last decade, we faced a fast growth and wide coverage of network connectivity (e.g., WLAN, UMTS, LTE) and a resulting ecosystem of diverse Internet-enabled (mobile) devices. In particular, we saw the advent of "smart things": Internet-connected small-scale embedded computing systems enabling us to grasp and interact with the environment through sensors and actuators. Looking at the application level, we saw the rise of complex scenarios such as environmental monitoring, ad-hoc car sharing, monitoring of cardiac patients, and adaptive lighting in road tunnels. Pervasive computing use cases like these are impressive by themselves. However, there are yet no means of global-scale communication and coordination to connect such isolated islands.

Integrating diverse device types into a single platform for global-scale pervasive computing thus remains a grand challenge. With particular regard to interoperability, true pervasive computing on a global scale 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. Unfortunately, the current "one-app-per-object" approach and data collection in proprietary clouds lead exactly to the opposite direction (cf. [3]) of isolated pervasive computing islands.

Furthermore, our knowledge of pervasive computing grew over the last decade. The tremendous success of online social networks has coined the phrases *Pervasive Social Computing* [4] and *Socially Aware Computing* [5]. These systems are both pervasive and social, thus comprising *Human-to-Human (H2H)*, *Human-to-Machine (H2M)* and *Machine-to-Machine (M2M)* communication. With the inclusion of cloud services, we gain pervasive communication patterns as shown in Figure 1. We suggest that one open standard baseline platform should handle all these types of communication and provide necessary services. On the one hand, such a platform should adopt and integrate with successful Web concepts to benefit from increased interoperability and uptake. On the other hand, it should enhance the Web by communication patterns and services beyond traditional request-response.

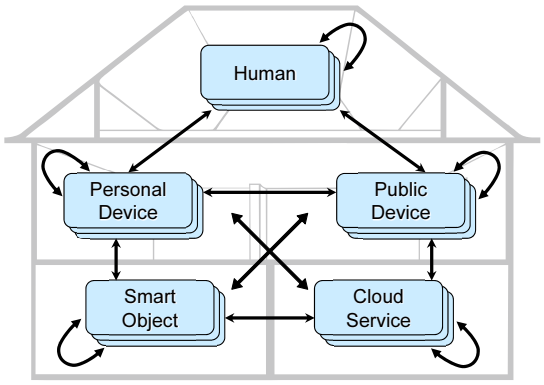

Fig. 1. Communication patterns in *Pervasive Social Computing*

The *Web of Things* approach attempted to fully integrate smart devices into the Web. However, it suffered from a centralized client-server architecture, lacking federation and being limited in terms of available communication patterns. Instead, we pursue a vision of humans, personal and public devices, smart objects and cloud services, able to interact with each other directly in real-time and in a controlled, but decentralized way. We believe that the *eXtensible Messaging and Presence Protocol (XMPP)* is the basis to realize this vision. It is the only open standards family available today that offers inherently extensible support for global-scale direct communication among human users, machines, sensors, etc. in real-time pervasive computing environments.

Therefore, this paper explores the possibility of using XMPP and its extensions to form the heartbeat of global-scale pervasive computing. The motivation for this work originates from independent XMPP-related research at our institutions and the common interest to build bridges between our previously independent "research islands" with the help of XMPP federation. Based on both literature review and practical experience from our joint initiative, we claim that XMPP can be the baseline platform for pervasive computing on a global scale.

This work delivers two main contributions: a survey of current research on XMPP-powered pervasive computing and a proof-of-concept study in federation of pervasive systems between organizations. Altogether, we provide background knowledge and hands-on experiences to foster discussion in the pervasive computing community and to assess if XMPP is suitable as a common baseline platform.

The rest of this paper is structured as follows: Section II briefly introduces the reader to XMPP. Section III surveys XMPP-related research in different areas of pervasive computing. Section IV presents our joint scenario as a small-scale, but scalable instance of our vision. Section V and Section VI comment on related surveys and approaches and conclude this work with an outlook to future research directions.

II. Introducing XMPP

XMPP was introduced as the Jabber open source project in 1999. It intended to create an open standard for instant messaging and bridge proprietary instant messengers existing at that time, a situation similar to today's social computing and Internet of Things. XMPP was approved by the IETF in 2002. A revised version from 2011 is now defined in RFC 6120 and RFC 6121 as the core of XMPP ([6], xmpp.org).

A. Core XMPP

The XMPP core facilitates XML-based communication between arbitrary entities in near real time based on long-living TCP connections. Entity addressing is provided by a powerful URI scheme: the *Jabber Identifier (JID)*. Syntactically, a JID consists of three parts: a *node* part, a *domain* part, and a *resource* part.

The node part identifies a specific entity account hosted on an XMPP server. The domain part identifies the server of the node in compliance with DNS, whereby message routing is enabled across multiple servers in a federated network setting. The resource part enables multiple devices to run under one user account (e.g., all personal devices of a user). For client addressing, XMPP differentiates between a *Full JID*, i.e. JID with node, domain and resource, and a *Bare JID* without resource identifier. Given a Bare JID, an XMPP server uses priorities to decide which resource should be addressed.

To be able to communicate in the XMPP network, bi-directional XML streams are initiated between an XMPP client and its server. During connection establishment, authentication and encryption are activated, the session (TCP connection) is bound to a resource identifier, a contact list (*roster*) is retrieved from the server, and initial presence information is sent and received by the entity. After successful completion

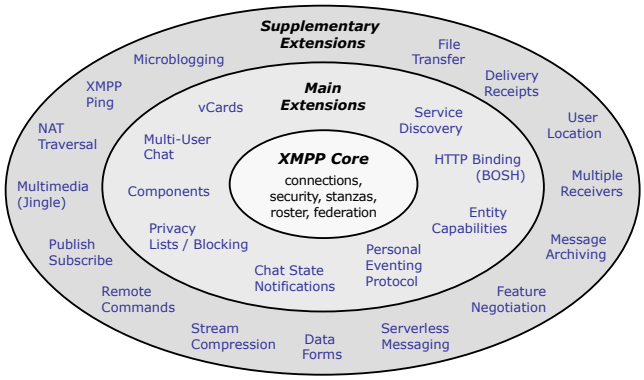

Fig. 2. Overview and classification of XMPP extensions

of these steps, XML fragments called *stanzas* are exchanged in near real time. XMPP provides three basic stanza types for the actual communication. A *message* stanza realizes a push mechanism, allowing an entity to send information to another entity with fire-and-forget semantics. A *presence* stanza realizes a broadcast mechanism where entities subscribed to the sender receive the published information. Presence stanzas usually handle presence subscriptions and the actual exchange of presence information between entities. An *IQ (Info/Query)* stanza realizes a reliable request-response mechanism with mandatory response. IQs can be sent from client to server and vice-versa, as well as directly between clients. All stanza types share a set of basic attributes, in particular the `from` and `to` attributes, specifying source and target JIDs. As such, stanzas serve as envelopes for sending arbitrary, yet well-formed and qualified XML data payloads.

B. XMPP Extensions

Another powerful feature of XMPP is its extensibility, inherited from the extension mechanisms of XML. Figure 2 gives an overview of the whole XMPP platform as it exists in 2013. The inner circle represents the XMPP core as described above. The second circle embraces the main extensions supported by most popular existing client and server implementations. These offer exchange of enriched presence information (Personal Eventing Protocol) and means to cache them (Entity Capabilities), Chat State Notifications, mechanisms to block unwanted communication, vCards and a mechanism to write own server extensions (Components).

More relevant for pervasive and social computing scenarios are the extensions for Service Discovery, Multi-User Chat, and HTTP Binding. Service Discovery offers a simple mechanism to retrieve the items of any resource and ask for its local features. It is also used to figure out support for certain XMPP extensions. The Multi-User Chat extension supports any kind of open or closed group communication including group presence, roles, access control, and late join support. The HTTP Binding protocol defines a proxy element facilitating an integration of browser-based clients like Smart TVs in an XMPP network.

A selection of supplementary extensions is shown in the third circle in Figure 2. These extensions serve specific use cases starting from a simple ping protocol leading to a comprehensive publish/subscribe service for awareness or context management.

One extension worth to mention separately is Serverless Messaging. If supported, multicast DNS is used to discover XMPP clients in the same local network. Once the JIDs and IP addresses are known, the clients exchange stanzas via peer-to-peer connections.

XMPP extensions are named XEPs (XMPP Extension Protocols)[1]. Every XEP has an assigned number like XEP-0060 for Publish-Subscribe. Some of the extensions in Figure 2 like File Transfer even consist of several XEPs. To avoid confusion, we intentionally omit the XEP numbers in this paper.

III. XMPP for Pervasive and Social Computing

XMPP's characteristics as a lightweight event middleware and its extensibility, described in Section II, inspired numerous researchers to use XMPP for challenges in the wide area of pervasive and social computing.

To classify these works accordingly, we take a step back to the communication patterns as depicted in Figure 1. The figure shows a house, where the ground floor depicts data gathered by smart objects and managed by cloud services. The first floor adds personal and public devices with a UI and therefore enables humans to access the information. Finally, the top floor adds humans (and communities thereof) in terms of social computing. In the following, we elaborate XMPP-related research according to these three (floor) levels.

As there are often comparable patterns of using XMPP and its extensions, we build up an overview architecture in Figure 3, where all different communication patterns and their mapping to XMPP and its extensions are depicted.

A. Ground Floor – XMPP for the Internet of Things

The ground floor covers the acquisition of data and the interconnection of the user and the real world. It reflects Kevin Ashton's [7] fundamental idea behind the *Internet of Things* (IoT) concept: make the physical world accessible from the Internet by embedding computing technology in everyday objects and use sensor devices to grasp the environment. By using XMPP to integrate smart objects (i.e., sensors and actuators) in the Internet, a pervasive environmental awareness and a bridge over the current gap between the virtual and the physical world is hence possible.

Current approaches of using XMPP in IoT scenarios can be split in two contrary strategies: (i) integrate smart objects and sensing devices with XMPP-enabled gateways and (ii) deploy XMPP itself on resource-constrained devices.

For (i), approaches like [8]–[10] use the Publish Subscribe extension 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. [8] 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 with Wave-enabled[2] OpenFire[3] servers. Sensor Andrew [10] is a large-scale framework to deploy sensors across Carnegie Mellon University and to prototype scalable applications. [9] 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 XEP extensions for IoT scenarios are currently developed by Peter Waher[4]. 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. The IoT XEPs currently in development rely heavily on the *Efficient XML Interchange (EXI)* format to transmit detailed XML messages containing additional, yet redundant meta data for the raw sensor data. This meta data is needed to interpret the raw sensor data once, but it is also included in each message, thus increasing message size. Such optimizations are required to compete with constraint-specific protocols like CoAP [11].

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 [12], 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. The next step was Chatty Things [13]. Several improvements like an API for developing IoT applications, support for essential XEPs like Multi-User Chat, optimizations for small RAM/ROM footprints and a reduction of XMPP message exchange were implemented. The integration of Chatty Things in Publish/Subscribe environments is presented in Section IV together with a discussion on how to avoid redundant data in networks of smart objects with limited IP payload size.

B. Ground Floor – XMPP for Cloud Computing

As for cloud services, it is still uncommon for smart objects to push their data to the cloud via XMPP. Other methods for pure data push like REST interfaces seem to be sufficient and more practical in this case. Thus, XMPP for object-to-cloud communication is only preferable if an XMPP infrastructure is already in place. XMPP is more useful when smart objects are controlled autonomously by cloud services. However, we are unaware of current research doing this without user intervention.

The most profound use of XMPP in cloud computing can be found for communication between cloud services. Agent-based systems such as SPADE [14] or Kestrel [15] use XMPP presence to signal agent availability. Tasks are

[1] xmpp.org/extensions

[2] Wave Federation Protocol = XMPP extension for concurrent federated editing of XML originally developed by Google

[3] http://www.igniterealtime.org/projects/openfire/

[4] http://wiki.xmpp.org/web/Tech_pages/IoT_XepsExplained

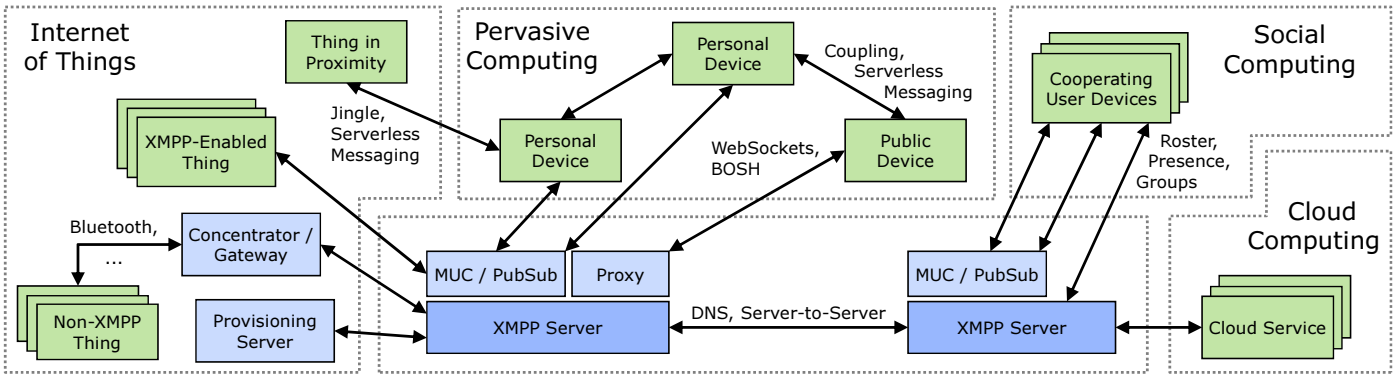

Fig. 3. Overview of XMPP use in pervasive and social computing

delegated directly to agents using XMPP messages. The Multi-User Chat XEP is used for multiple agents to perform a common task. Kestrel has recently even been used to build a virtual organization cluster to simulate proton collisions using 800 virtual machines [16]. The XMPP-based agent approach has been extended by [17] to smartphones in order to use them as one cloud computing resource. The Serverless Messaging XEP is used to discover nearby devices.

Consequently, this approach of cooperating agents also works for intercloud communication as shown in [18] and [19]. Cooperating cloud services act as XMPP clients and use presence to share availability information. The authors proposed a new XMPP extension called IO Data for remote service execution that has not been widely accepted in the community and got deferred from the standardization process.

C. First Floor – XMPP for Pervasive Computing

While we have gathered the necessary information from sensors and optionally sent them to cloud services, we want to have a mixture of personal and public devices to access the information and present it to users in a convenient way.

Gathering sensor information with personal devices soon refers to classic context management architectures which enable users to subscribe to certain information relevant for their current situation and the apps they use. The XCoA architecture [20] describes such a framework using XMPP. Context agents act as XMPP clients posting their context information to context providers. Context providers are realized as server Plug-ins using the Components XEP. Furthermore, the Publish/Subscribe resp. HTTP Binding XEPs are used to access information from personal devices resp. to visualize context information in websites. Other systems such as MACK [21] use XMPP only as a notification infrastructure connected to a central context management server.

Furthermore, personal devices can also be used to control smart objects [22]–[24]. The Remote Commands XEP is used to send commands to sensor-equipped robots. Both the smartphone as a controller and the robot run an XMPP client library. Roster information is used to discover smart objects the user is authorized to control. IQ stanzas are used to directly query information from the smart object. Event notifications can be realized using message stanzas or the Publish/Subscribe XEP. Audio/video sessions between the smart object and the personal device can be set up using the Multimedia (Jingle) extension. [25] shows the integration of this H2M communication in a standard XMPP client software. Machines and their presence information are displayed in the roster and can be activated or shut down.

Communication between cloud services and personal devices defines the research field of Mobile Cloud Computing, where tasks are delegated to the cloud and mobile devices get notified about the results. The focus here is on the proper interaction mechanisms to provide awareness about task execution status. [26] proposes to use XMPP as a push notification infrastructure for arbitrary JSON content comparing it to existing proprietary solutions like Google Cloud Messaging or Apple Push Notification Service.

Hornsby [27] demonstrates the use of XMPP for coupling multiple (personal) devices with a remote Model View Controller (MVC) approach. For 1:1 coupling, the communication between model and view builds upon customized message and IQ stanzas. If multiple views are connected to a model, Multi-User Chat or Publish/Subscribe XEPs are used.

DireWolf [28] takes a similar approach by distributing a widget-based application over multiple personal devices. UI elements are synchronized by the Publish/Subscribe XEP.

At the next level of pervasive computing, public devices like large public displays come into play. Koren [29] shows means for coupling public with personal devices using a challenge-response authentication scheme. The public device runs an XMPP client session with its own user account, but enables the launch of a user session that runs under the account of the requesting user. Communication between personal and public device is restricted to XMPP messages in this scenario. However, XMPP can also be used to initiate external direct communication between the devices as shown by [30]. A VNC connection between a tablet and a projector is initiated via XMPP. The Serverless Messaging XEP is useful in these scenarios for device discovery, but the JID of the public device can also be encoded in a QR tag on or near the display.

A deficiency of XMPP in these scenarios is the lack of advanced Service Discovery. The Service Discovery XEP only allows to discover components and features running at an XMPP server or to query entities with a known JID. There are yet no means to discover services realized as XMPP clients, e.g. an XMPP client run by a public display.

D. Top Floor – XMPP for Pervasive Social Computing

To complete our model, pervasive computing increasingly embraces social computing applications. Especially in H2H communication and collaboration, concepts such as dynamic contact and group management, organizational structure, collaboration context, awareness, security, privacy, and trust play important roles. Many of these concepts are already supported with the XMPP core standard and XEPs to support individuals and communities in their particular practices. XMPP is thus inherently well-suited for pervasive social computing, as we point out in this section.

Dynamic Contact & Group Management: The management of personal contacts and groups of persons is perhaps the main purpose of social computing. Contact management is well supported by core XMPP roster lists and in most cases used without changes [23], [31]–[33]. However, several authors highlight the need for better support of dynamic group management and sharing in distributed settings [32], [34]. In the context of dynamic patient-focused group management in e-Healthcare, Griffin et al. [34] find limitations in different approaches. Roster lists provide too loose and informal support for group management only for single persons. Group management based on the Publish/Subscribe extension requires administrative efforts not in control of the user. As a solution, various authors propose own extensions for closing the existing gap [32], [34]. Nevertheless, most authors do not take into account the intuitive and shared group management support in the Multi-User Chat XEP.

Organizational Structure: We find common agreement that inter-organizational collaboration must favor decentralized approaches and that XMPP federation fosters scalable modeling of intra and inter-organizational network structures. Franke et al. [32], [35] emphasize the need for decentralization in disaster management, solvable with XMPP federation. Woo et al. [23] describe XMPP federation as enabler for the large-scale construction of a hierarchically clustered presence network. The construction of an XMPP federation-based scalable media distribution network is described in [33].

Collaboration Contexts: Exchange of contact information, group formation and organizational structure are mostly precursors for collaboration among participants of social pervasive computing platforms. However, collaboration requires well-defined collaboration contexts. Dodson et al. [36] use XMPP Multi-User Chat rooms as collaboration context for participants in their Junction network. Franke et al. [35] employ Google Wave, where waves serve as collaboration contexts. DireWolf [28] provides collaboration contexts for the orchestration of widgets to complete Web applications, including XMPP Multi-User Chat rooms and Publish/Subscribe channels. Mobilis [37] takes a different approach by providing an environment for dedicated collaboration services running as XMPP clients. These services form the collaboration context for mobile native multi-platform apps. An own service discovery extension is used to discover these services based on app identifiers.

Awareness: XMPP inherently supports real-time awareness in multiple facets (cf. [4]) with a generic presence mechanism and various Personal Eventing XEPs. Woo et al. [23] emphasize the importance of awareness with their design of a large-scale XMPP presence network. According to [32], real-time awareness is essential for the synchronization of inter-organizational collaboration in disaster management and achievable with XMPP. Mayrhofer et al. [31] employ XMPP presence for building a location sharing platform, deliberately avoiding the use of a Personal Eventing XEP for geo-location changes in favor of increased interoperability.

Security, Privacy & Trust: Communication and collaboration in pervasive social computing should be secure, protect privacy, and manage trust among participants to create confidence in the whole system. XMPP widely supports security by TLS/SASL channel encryption and authentication across federated networks. End-to-end encryption is currently under active research [6]. Only few authors explicitly mention XMPP security features (e.g. [31], [33]). XMPP provides privacy protection with roster list management in conjunction with asymmetric presence subscriptions. However, as soon as presence subscriptions are exchanged, there is no support for managing trust among participants [33]. Thus, the authors realized an XMPP roster list contact-based trust management and inference extension.

Connection to Web: Social software of today is mostly Web-based and accessed via browser-based interfaces available on most devices. In the last years, XMPP gained momentum with its integration in online social networks such as Facebook, Google Talk and Skype. HTTP Binding extensions enabled the connection of XMPP from within regular browsers. Dodson et al. [36] employ this technique in their Junction protocol. Franke et al. [35] built their work upon Google Wave and its browser-based front end. Recently, the availability of WebSockets in browsers gradually replaces XMPP HTTP Binding with more stable and performant XMPP over WebSocket connections [28], [33], [38].

XMPP and its public extensions provide a coherent set of generic building blocks for Pervasive Social Computing. Many of these building blocks can be applied as-is in common scenarios and serve as solid foundation to own extensions for more specialized scenarios. However, some proposals for custom extensions would also be more generically applicable, e.g., group management and trust.

IV. Example Solution - ACDSense

To evaluate the interoperability and federation capabilities of the XMPP protocol, we developed an inter-organizational pervasive computing scenario (called ACDSense) spanning our three universities, RTWH Aachen University, BTU Cottbus - Senftenberg and TU Dresden. The findings of this approach can be further generalized to realize global-scale pervasive computing. The scenario is a classic scenario of sensor data analysis where technicians check for relevant sensor data at various sites. While this has already been demonstrated using XMPP before, the peculiarity of our approach is the design of three independent systems, each with data source and sink, which are enriched by the federated access to sensor data. ACDSense enables different views on sensed data by supporting standard XMPP clients as well as XMPP-driven apps and websites.

As depicted in Figure 4, each organization runs its own XMPP server as the core of the infrastructure. The three servers

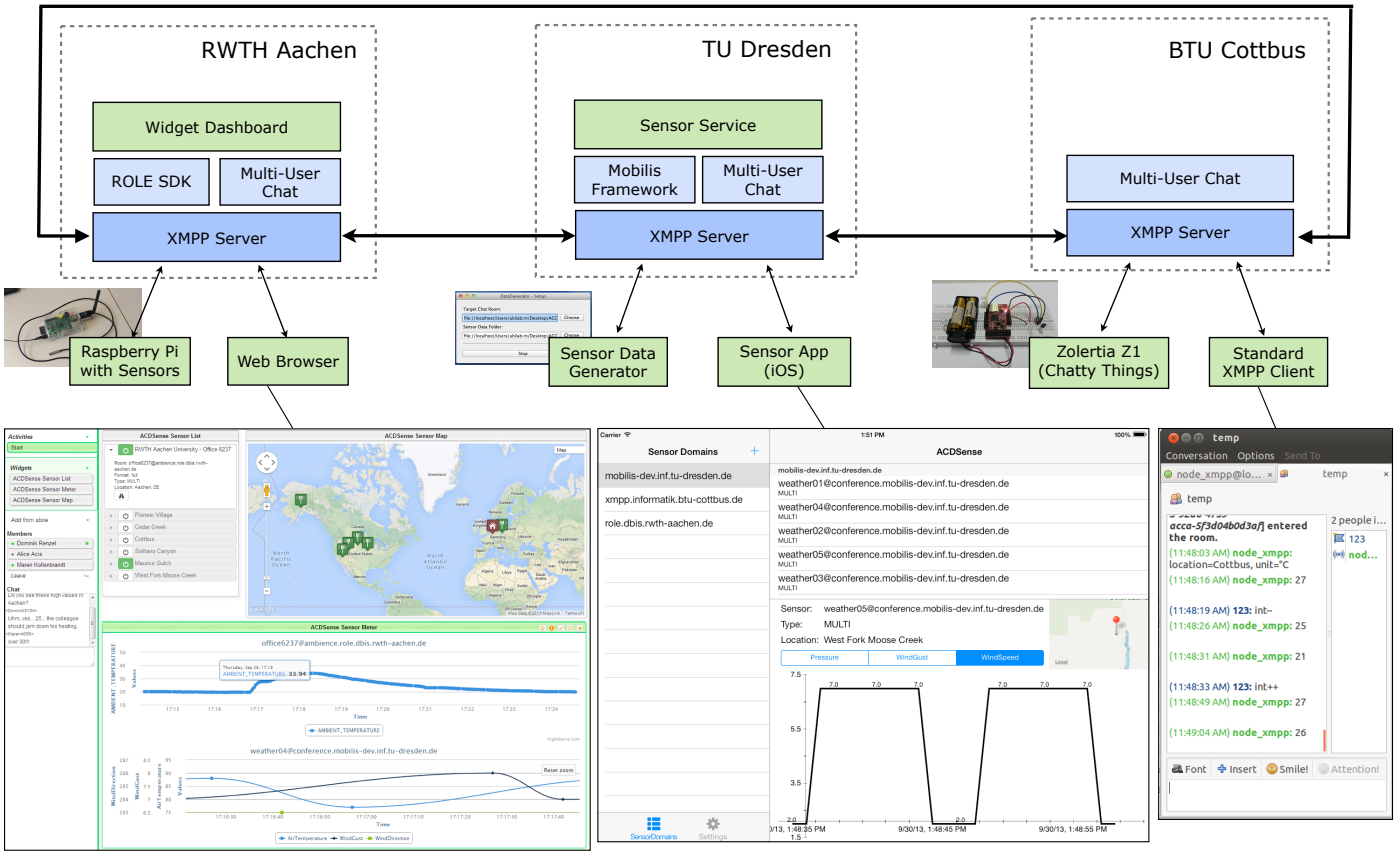

Fig. 4. Architecture and screenshots of example scenario ACDSense

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. Seamless Integration of Smart Objects (IoT) in Cottbus

Chatty Things itself implements XMPP as the underlying communication protocol for smart objects running the Contiki operating system to provide a set of services, e.g. discovery, identity management, authentication, user interaction via remote commands, support for hybrid smart object networks, and information filtering. Constrained and non-constrained devices can directly communicate with each other without relying on protocol gateways, device-specific code or special data representations. Push notification (i.e., Publish/Subscribe) is required for smart objects, as it provides a bandwidth- and energy-efficient event distribution (cf. [39]). While the Publish/Subscribe extension can broadcast complex information and provide content filtering, access control, and a definition of affiliations (e.g., decouple publishers and subscribers), it is not supported by all XMPP clients and it requires structural complex and large messages (i.e., heavy use of IQ stanzas) to push information to interested XMPP clients.

Chatty Things hence uses *Multi-User Chat (MUC)* as a alternative push notification: joining a MUC room is comparable to a "subscription", sending a message to a MUC room is comparable to a "publication", while the MUC room itself represents the topic (e.g., temperature). The XEP approach *Temporary Subscription for Presence (TSP)* [13, Sec. VI] combines the advantages of Multi-User Chat (i.e., supported by the majority of XMPP clients in contrast to Publish/Subscribe) and presence notification to realize a lightweight PubSub alternative extended with a topic-based filter and the decoupling of publishers and subscribers. To reduce the XMPP message size in smart object networks, 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 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.

Using this MUC-based approach, the information can be accessed by ordinary XMPP clients like Adium or Pidgin. The rightmost lower screenshot in Figure 4 shows the output of a sensor MUC. The information is human-readable and thus no additional software is necessary to monitor Chatty Things sensors if the JIDs of all sensor MUCs are known in advance.

B. Mobilis-based Sensing Apps in Dresden

The second subsystem was developed at TU Dresden and realizes a native sensing app on top of the Mobilis framework [37] with basic sensor discovery features. As already mentioned in Section III, 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.

We developed a Sensor Service in Java that connects to the XMPP server using the Smack client library. The Sensor Service accepts subscribe requests from clients for multiple XMPP servers. Once a server is added to the list, the Sensor Service requests for all Multi-User Chat rooms using the Service Discovery extension. To differentiate between standard MUCs and MUCs used for sensor data, we added a JSON description to the MUC properties of each sensor MUC. The following example shows the description property of one of our sensor MUCs representing weather data:

```
{"sensormuc":
 {"type":"MULTI",
  "format":"full",
  "location":
   {"countryCode":"US",
    "cityName":"Pioneer Village",
    "latitude":40.98520,
    "longitude":-111.9020}}}
```

This type of sensor MUC is an extension to the topic-based sensor MUCs described above, as they allow for multiple sensors to post to a MUC. Here, the MUC represents a room or other geographic area with information available from multiple sensors.

Using this description, the Sensor Service is also able to provide some additional meta data like location. The actual sensor data is accessed via an iPad App (Sensor App) written in Objective C, which is shown in the lower middle of Figure 4. The app allows to browse sensor domains and select sensors of interest within each domain. All data is provided by the Sensor Service.

While the Sensor App could access the sensor MUCs directly, the indirect access via the Sensor Service provides some added value: The data is delivered in a structured XML format via IQ stanzas already including all the meta information available from the MUC properties. Furthermore, other formats like the short format provided by the Chatty Things sensors can also be read without adding additional logic to the app. However, the main advantage of the indirect approach is the ability to access past sensor data as well as to limit the number of sensor updates using pre-filtering at the Sensor Service.

The third 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 [40] and sends it to multiple MUCs. One MUC is used per weather station while there are always multiple sensor types per MUC like wind, humidity, or ambient temperature.

Actual data is embedded in MUC messages as JSON string:

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

C. High-Performance Sensors and Dashboard in Aachen

The sensor part at RWTH Aachen University represents a class of low-cost, high-performance sensors that is capable of using the full JSON format described in Section IV-B. 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. [28]), providing so called spaces as collaboration contexts. Each space manages a list of members and enables the orchestration of multiple widgets to complete Web applications with the help of an *Inter-Widget Communication (IWC)* approach. Additionally, each space controls an XMPP MUC room and a Publish/Subscribe node for remote IWC across browser instances. For supporting the ACDSense scenario, we built a set of three widgets and deployed them to a space (cf. Figure 4, lower left screenshot).

A Sensor List widget displays a list of available sensors, retrieved via XMPP service discovery on MUC services located at our different institutions and filtered in the same manner as described in Section IV-B. Each sensor item in the list features a toggle button and additional sensor meta data, again retrieved via service discovery on the respective MUC room. With the toggle button the user can switch on/off the flow of sensor data to the client, realized by entering/leaving the respective MUC room. The Sensor List widget furthermore distributes sensor data and meta data to two other widgets.

An additional Sensor Map widget visualizes sensor geo-locations on a map. Together with the current location of the user, accessible in modern browsers, this widget supports navigation of service personnel to a specific sensor's location. Finally, a Sensor Meter widget shows dynamic sensor data graph plots for all sensors toggled on in the Sensor List widget, thus allowing sensor monitoring in real-time.

D. Discussion

The whole setup described in Section IV 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.

XMPP servers need to be configured 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 must be added. 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/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 works well for small federation scenarios like the one described above. Large scenarios certainly require DNS entries for each server and MUC sub-domain.

Apart from these administrative issues, it was a memorably easy experience to interconnect our systems as described above. We merely had to agree on a common format for sensor data and meta data, which will eventually become superfluous with upcoming standard definitions. Everything else was provided by XMPP and a handful of widely supported XEPs. We even did not have to agree on a common programming environment, given the rich ecosystem of XMPP libraries available in many programming languages. More fine-grained access control to sensor MUC rooms is possible with role and white/blacklisting concepts available in the MUC XEP.

V. Related Work

There have been several previous attempts to survey and classify XMPP-related work. [6], [41], [42] focus on a review of XMPP standardization work and offer a good overview in this direction. However, research work or actual system designs are not covered by these surveys. [43] reviews XMPP research with a focus on consumer electronics and defines six application classes. We built upon this work and elaborate on XMPP work from the pervasive computing perspective.

Another category of related work lies in the cross-cutting support for global-scale pervasive computing. The Web of Things solution of Philips for example, couples HTTP with the Constrained Application Protocol (CoAP) [11] to build a REST-based global-scale pervasive computing. The integration of CoAP nodes has the disadvantage that smart object-specific code and application protocol gateways are used, introducing additional complexity in terms of single point of failure, protocol mapping, and protocol version updates.

The IBM smarter planet approach uses the Message Queuing Telemetry Transport (MQTT) [44] protocol, which focuses on a highly scalable Publish/Subscribe, but not on a standardized and extensible communication scheme.

In order to estimate XMPP's performance in smart object networks, our performance test reconstructed a test case (e.g., temperature reception) of RESTful Web services [45, Sec. 6.3]: one-hop network (IPv6) while one Zolertia Z1 runs the 6LoWPAN border router and is connected via SLIP to the computer; ContikiMAC disabled. The Chatty Thing pushes its sensed temperature data to the MUC room with the sensor-specific topic `'temp'`. For CoAP, the provided REST example (e.g., `get 'hello world'`) of Contiki OS was used. The completion time (summarized in Table I) for the REST-based approaches was measured as the time between requesting a service and getting its sensed data. As Chatty Things uses the Publish/Subscribe paradigm, dedicated request messages are not needed to retrieve updates for the sensed data.

TABLE I. REST vs XMPP in smart object networks

Approach	Request Size	Response Size	Completion Time
RESTful Web Services	85 Bytes	141 Bytes	440 ms
CoAP	15 Bytes	19 Bytes	54 ms
XMPP (Group Chat)	-	144 Bytes	201 ms
XMPP (One-to-One Chat)	-	96 Bytes	122 ms
XMPP (Presence)	-	35 Bytes	38 ms

The measurements of XMPP also include the performance for presence status and one-to-one chat message exchanges to give a general performance overview of the typically used XML message stanzas. The very low message size of CoAP is a clear advantage in smart object networks, but XMPP can achieve a comparable performance when the XML message fits in a single TCP/IP packet (i.e., max. 48 Bytes). Thus, implementations of XML data compressions (e.g., EXI) have to be available on smart objects to use XMPP more efficiently.

VI. Conclusions

Federating pervasive computing systems, as described in this work, is only a first step towards global-scale pervasive computing. We presented the current state of the art in using XMPP as infrastructure for pervasive computing, accompanied by a showcase for XMPP-based inter-organizational sharing of sensor data.

XMPP is currently revitalizing as a pervasive middleware because its multiple-resources-per-user approach fits ideally the vision of pervasive computing. Young developers and big companies are joining the game while becoming more and more active in XMPP-based research and standardization. Recent developments of peer-to-peer Web technologies like WebRTC and XMPP over WebSockets further promote the use of XMPP as a signaling protocol.

Nevertheless, there are several challenges still to be mastered. As mentioned previously, discovery in XMPP is still very server-centered and needs to be extended to fit the needs of global-scale peer discovery. Access control needs to be more elaborated to secure sensitive sensor data. The federated approach also calls for mechanisms for end-to-end security like Off-the-Record (OTR) messaging, which are not yet widely used in the XMPP community. Furthermore, full exchange of redundant meta information is often unnecessary for IoT scenarios and needs to be reworked as well.

Despite these challenges, XMPP already offers much more than any other comparable solution. Standardization is absolutely necessary to overcome our isolated pervasive computing islands and to build a solution where billions of pieces of computing seamlessly disappear in their natural surroundings.

References

[1] M. Weiser, "The Computer for the 21st Century," *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[2] D. Saha and A. Mukherjee, "Pervasive Computing: A Paradigm for the 21st Century," *Computer*, vol. 36, no. 3, pp. 25–31, 2003.

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

[4] J. Zhou, J. Sun, K. Athukorala, D. Wijekoon, and M. Ylianttila, "Pervasive Social Computing: Augmenting five Facets of Human Intelligence," *Journal of Ambient Intelligence and Humanized Computing*, vol. 3, no. 2, pp. 153–166, 2012.

[5] P. Lukowicz, A. Pentland, and A. Ferscha, "From Context Awareness to Socially Aware Computing," *IEEE Pervasive Computing Journal*, vol. 11, no. 1, pp. 32–41, 2012.

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

[7] K. Ashton, "That 'Internet of Things' Thing," *RFID Journal*, July 2009. [Online]. Available: http://www.rfidjournal.com/article/view/4986

[8] 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.

[9] 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.

[10] 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.

[11] 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/

[12] 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.

[13] 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.

[14] M. E. Gregori, J. P. Cámara, and G. A. Bada, "A Jabber-based Multi-Agent System Platform," in *Proc. of the 5th ACM Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2006, pp. 1282–1284.

[15] L. Stout, M. A. Murphy, and S. Goasguen, "Kestrel: an XMPP-based Framework for Many Task Computing Applications," in *Proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS)*. ACM, 2009, pp. 11:1–11:6.

[16] L. Stout, M. Walker, J. Lauret, S. Goasguen, and M. A. Murphy, "Using Kestrel and XMPP to Support the STAR Experiment in the Cloud," *Journal of Grid Computing*, vol. 11, no. 2, pp. 249–264, 2013.

[17] G. Huerta-Canepa and D. Lee, "A Virtual Cloud Computing Provider for Mobile Devices," in *Proc. of 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, 2010, pp. 1–6.

[18] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow, "Blueprint for the Intercloud-Protocols and Formats for Cloud Computing Interoperability," in *Proc. of the 4th IEEE Conference on Internet and Web Applications and Services (ICIW)*, 2009, pp. 328–336.

[19] F. Tusa, M. Paone, M. Villari, and A. Puliafito, "CLEVER: A Cloud-enabled Virtual Environment," in *Proc. of the IEEE Symposium on Computers and Communications (ISCC)*, 2010, pp. 477–482.

[20] D. Gomes, J. M. Gonçalves, R. O. Santos, and R. Aguiar, "XMPP based Context Management Architecture," in *Proc. of the IEEE GLOBECOM Workshops (GC Wkshps)*, 2010, pp. 1372–1377.

[21] F. Schmitt, J. Cassens, M. C. Kindsmüller, and M. Herczeg, "Mental Models of Ambient Systems: A Modular Research Framework," in *Proc. of the 7th Int. and Interdisciplinary Conference on Modeling and Using Context (CONTEXT)*. Springer, 2011, pp. 278–291.

[22] Q. Wang, S. Liu, and Z. Wang, "A New Internet Architecture for Robot Remote Control," in *Proc. of the IEEE/RSJ Conference on Intelligent Robots and Systems*, 2006, pp. 4989–4993.

[23] H. Woo, H. Kim, K. Kim, and D. Kim, "A Large Scale Presence Network for Pervasive Social Computing," in *Proc. of the IEEE Conference on Pervasive Computing and Communications (PERCOM Workshops)*, 2013, pp. 145–150.

[24] S. Bendel, T. Springer, D. Schuster, A. Schill, R. Ackermann, and M. Ameling, "A Service Infrastructure for the Internet of Things based on XMPP," in *Prof. of the IEEE Conference on Pervasive Computing and Communications (PERCOM Workshops)*, 2013, pp. 385–388.

[25] K. Kurowski, A. Oleksiak, M. Witkowski, and J. Nabrzyski, "Distributed Power Management and Control System for Sustainable Computing Environments," in *Prof. of the IEEE Green Computing Conference*, 2010, pp. 365–372.

[26] H. Flores and S. Srirama, "Mobile Cloud Messaging supported by XMPP Primitives," in *Proc. of the 4th ACM Workshop on Mobile Cloud Computing and Services (MCS)*, 2013, pp. 17–24.

[27] A. Hornsby, "XMPP Message-based MVC Architecture for Event-driven Real-Time Interactive Applications," in *Proc. of the IEEE Conference on Consumer Electronics (ICCE)*, 2011, pp. 617–618.

[28] 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.

[29] 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.

[30] F. Chang and D. Chen, "The Design of an XMPP-based Service Integration Scheme," in *Proc. of the 7th IEEE Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP)*, 2011, pp. 33–36.

[31] R. Mayrhofer, C. Holzmann, and R. Koprivec, "Friends Radar: Towards a Private P2P Location Sharing Platform," *Proc. of the 13th Conference on Computer Aided Systems Theory (EUROCAST)*, pp. 527–535, 2012.

[32] J. Franke, C. Ulmer, and F. Charoy, "Pervasive Emergency Response Process Management System," in *Proc. of the 8th IEEE Conference on Pervasive Computing and Communications (PERCOM Workshops)*, 2010, pp. 376–381.

[33] D. Renzel, K. A. N. Rashed, and R. Klamma, "Collaborative Fake Media Detection in a Trust-Aware Real-Time Distribution Network," in *Proc. of the 2nd Workshop focusing on Semantic Multimedia Database Technologies (SMDT)*, ser. CEUR-WS Proc., vol. 680, 2010, pp. 17–28.

[34] L. Griffin, E. de Leastar, and D. Botvich, "Dynamic Shared Groups within XMPP: An Investigation of the XMPP Group Model," in *Proc. of the IFIP/IEEE Symposium on Integrated Network Management (IM)*, 2011, pp. 634–637.

[35] J. Franke, F. Charoy, and P. El Khoury, "Framework for Coordination of Activities in Dynamic Situations," *Enterprise Information Systems Journal*, vol. 7, no. 1, pp. 33–60, 2013.

[36] B. Dodson, A. Cannon, T.-Y. Huang, and M. S. Lam, "The Junction Protocol for Ad Hoc Peer-to-Peer Mobile Applications," [Online] http://mobisocial.stanford.edu/papers/junction.pdf, 2011.

[37] 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.

[38] K. A. N. Rashed, D. Renzel, R. Klamma, and M. Jarke, "Community and Trust-aware Fake Media Detection," *Multimedia Tools and Applications Journal*, pp. 1–30, May 2012.

[39] T. Eugster, A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many Faces of Publish/Subscribe," *ACM Computer Survey 35 (2)*, 2003.

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

[41] P. Saint-Andre, "Jingle: Jabber Does Multimedia," *MultiMedia, IEEE*, vol. 14, no. 1, pp. 90–94, 2007.

[42] O. Ozturk, "Introduction to XMPP Protocol and Developing Online Collaboration Applications using Open Source Software and Libraries," in *Prof. of the IEEE Symposium on Collaborative Technologies and Systems (CTS)*, 2010, pp. 21–25.

[43] A. Hornsby and R. Walsh, "From Instant Messaging to Cloud Computing, an XMPP Review," in *Proc. of the 14th IEEE Symposium on Consumer Electronics (ISCE)*, 2010, pp. 1–6.

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

[45] D. Yazar and A. Dunkels, "Efficient Application Integration in IP-based Sensor Networks," in *Proc. of the 1st ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)*, 2009, pp. 43–48.