# User-Hosted SOA Infrastructure over XMPP

João M. Gonçalves
PT Inovação e Sistemas
Universidade de Aveiro
Aveiro, Portugal
Email: joao-m-goncalves@ptinovacao.pt

Diogo Gomes
Instituto de Telecomunicações
Universidade de Aveiro
Aveiro, Portugal
Email: dgomes@av.it.pt

***Abstract*—The proliferation of user-owned connected devices has brought value to mobile application developers, which can make use of locally-available sensors and capabilities and send their information to the web, centralizing the data flows. A more distributed approach would have device capabilities offered directly on the network as services hosted by the user. These pervasive user-hosted services could be made discoverable and available over a public federated service infrastructure. The infrastructure would provide transport over an identity layer, where endpoints are addressed by their identities instead of network identifiers, and on top of which services can be exposed to be consumed by trusted friends or anonymous users, as the hosting user prefers. The work presented in this paper explores the possibility of implementing such a distributed social Service-Oriented Architecture (SOA) over Extensible Messaging and Presence Protocol (XMPP). This SOA, which would expose re-usable coarse-grained software components in a service ecosystem, differs from traditional SOA because it attempts to counter the centralization existing services, in favour of a fully-distributed service ecosystem where each peer can behave both as service consumer and provider. Finally, an analysis is done on how suitable XMPP is to serve as a base protocol for such infrastructure.**

***Keywords*—Software architecture; Distributed computing; Middleware; Web services**

## I. Introduction

The Service-Oriented Architecture (SOA) paradigm advocates for systems to be composed of re-usable coarse-grained software components, which consume and provide services in a service ecosystem in an attempt to promote loose coupling, increasing re-usability, reducing technology lock-in and ease extension. Although SOA is commonly mentioned in an enterprise context, the architectural principles that it advocates are very present in the web today, as most web applications expose some of their data via Representational State Transfer (REST) APIs and Really Simple Syndication (RSS) feeds [1].

Despite this landscape, the current web is still very centralized, with most of the user data and most popular services being hosted or controlled by a few dozens of companies. Moreover, the proliferation of user owned networked devices has untapped potential that can be capitalized by allowing device capabilities to be offered on the network as services. These pervasive user-hosted services could be made discoverable and available over a public federated service ecosystem with a social twist. This exercise of bringing together social and pervasive computing into a user-hosted SOA was one of the key research challenges tackled in the SOCIETIES project, and the concept behind the work presented in this paper.

The Extensible Messaging and Presence Protocol (XMPP) is an IETF protocol with numerous open extensions built directly on top of TCP, which can provide an extensible messaging infrastructure. It has built-in federation mechanisms, endpoint authentication, resolution and presence, message routing, asynchronous messaging and some degree of reliability. In recent years XMPP has been adapted in applications other than instant messaging, and even as a lightweight approach to Message Oriented Middleware (MOM).

This paper describes the efforts done under the SOCIETIES project to leverage the principles of SOA and the advantages of using XMPP as a messaging bus by establishing a parallelism between XMPP services and the generic services that SOA describes. XMPP was used as a session-layer protocol where authentication and federated Identinet [2] functions are provided, abstracting the network and the wire-protocol as much as possible. Native APIs are exposed to services with only identity-level and service-level concepts which eases discovery and transparently addresses transport, network and data representation interoperability for those services.

The underlying concepts and technologies are initially described in Section II, followed by the analysis of the distributed user-hosed SOA vision in Section III. In Section IV the design effort to realize the requirements using the XMPP concepts are described, and in Section V the implementation and results are presented. Finally, in Section VI, the benefits towards other approaches and the problems and value of the proposed approach are discussed.

## II. Background

SOA has been one of the most referred concepts in software engineering in the first decade of the new millennium. As an architectural paradigm SOA advocates that a software system should be designed having service-orientation in mind. Business-aware coarse-grained software components relate with each other by being either service providers or service consumers - they are often both, relatively to different services. Service discovery mechanisms are made available by a service broker for the consumers to find suitable providers. This approach aims to enable:

- loose coupling: service consumers only depend on the

service interfaces and not on particular provider implementations;
- increased re-usability: a business process can be decomposed in several component services, that can be reused across all business processes;
- reduced technology lock-in: services consumed are platform independent and all communication is based on open standards;
- easier extension: new logic can be added to business processes by the addition of a new service being consumed.

The rise of SOA as a major architectural trend was accompanied by the ascension of a few technological solutions for implementing SOAs, such as SOAP and WSDL, which had the twisted effect of binding the technology and the architectural paradigm together in the minds of many software engineers. That bond started being severed with the popularization of Representational State Transfer (REST), a minimalist style for providing data services over HTTP, and with the increasing use of raw XML and JavaScript Object Notation (JSON) to represent the service data payloads.

Message-Oriented Middleware (MOM) is another well known architectural paradigm with relatively successful implementations and wide dissemination. A MOM infrastructure, typically deployed internally in an enterprise, allows diverse software components to communicate asynchronously. A coordinating component usually called message broker will handle endpoint resolution, message persistence and routing. The most striking difference between MOM and traditional Web Services is the messaging model: web services normally use the request-response model, while MOM support more flexible and generic asynchronous messaging, which enables the publish-subscribe model. A MOM usually does not define any payload format. Java Message Service (JMS) is a well known specification of a MOM API and some implementations of it are used in enterprise-grade software deployments. AMQP is another well-known MOM specification which, unlike JMS, includes the definiton of a wire-level message format, aiming to enable interoperability between different MOM vendors.

XMPP is an IETF protocol which has open extensions, dubbed XMPP Extension Protocols (XEPs), which are supervised by the XMPP Standards Foundation (XSF). It is a bi-directional XML messaging protocol, originally aimed for IM, built directly on top of TCP, which can provide an extensible messaging infrastructure, with built-in federation mechanism, supporting endpoint authentication, resolution and presence, message routing, asynchronous messaging and some degree of reliability. In recent years XMPP has been adapted by applications other than IM, such as VoIP [3] and microblogging [4]. More recently there have been proposals to be used for sensor control [5]. Also, it can be used as a lightweight approach to MOM [6], representing a more interoperable alternative which dodges technological lock-ins of proprietary systems. However, unlike a typical enterprise MOM, it does not provide transaction management and is likely less scalable.

The XSF approved an extension in 2005, XEP-0072 [7], which defines the transport of SOAP messages over XMPP.

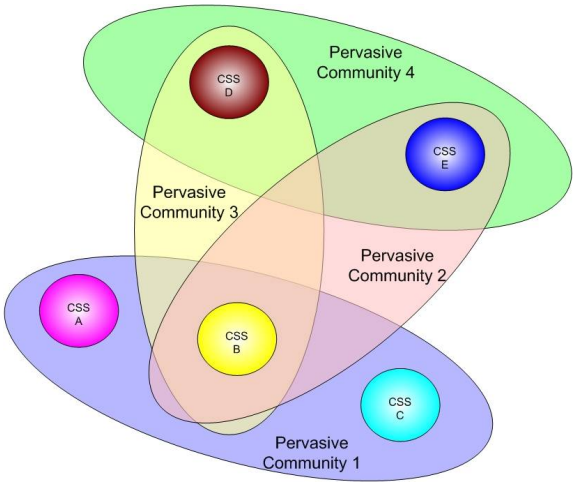


Fig. 1. Representation of CSSs and CISs

The extension efectively defines a SOAP binding to XMPP, as previous specs did to HTTP and SMTP. However, as expected from a direct binding, the advanced capabilities of XMPP such as presence and service discovery are ignored and the protocol is used simply as transport. Furthermore, the differences between IQ and Message are ignored in favour of the interaction pattern being defined at the SOAP level.

## III. Requirements of the SOCIETIES Vision

SOCIETIES is an European project with a consortium of 16 partners, active from October 2010 to March 2014. It is the largest integrated project out of the fifth call for project submissions of FP7, and aims to bring together social and pervasive computing into one integrating platform. In the project's vision, the proliferation of user owned networked devices has untapped potential that can be capitalized by allowing device capabilities to be offered on the network as services. These devices would together form a Cooperative Smart Space (CSS), a digital representation of a user or organisation, enabling the sharing of user-owned services. CSSs constitute the users' bridge between the physical world and the digital social communities the user is a part of. A community is a collection of CSSs and/or supporting infrastructure services, who wish to collaborate for mutual agreed purpose for which the community formed. The community's digital representation is called a Community Interaction Space (CIS), through which users can access and make available services. Figure 1 shows five CSSs, each of which represents an individual, that have formed themselves into four communities. These services would be accessible on top of a trust-enabled identity layer, designed with strong privacy concerns. An open federation of identity domains would be in place where anyone can create his domain. On top of this identity layer, ad-hoc trust relationships can be then formed, and access control and privacy policies should condition the visibility and accessibility of the user-owned services, enabling effective control of the user's digital exposition.

This vision is similar to Sarma and Girão's, that envision the Future Internet of Things as an Identinet where each endpoint, independently of being a person, a service or software, is represented by an identity. This Identinet provides some device independence as what matters is the entity that the device is operating on behalf of. Also, by employing identity management techniques, it has the potential to enable enhanced security and privacy for users [2]. Unlike in a traditional SOA where all services are exposed within an ecosystem, such as an enterprise, in this social-based approach where services are offered by individual users it makes more sense to enable discovery of services on top of an identity-layer.

In order to realize the social re-usability vision of SOCIETIES, user services have to be exposed both locally, within a device's internal communication and discovery mechanisms, and remotely, over the network. Given the dynamic and distributed nature of the envisioned architecture, an extensible and decentralised messaging framework with service announcement capabilities is required for supporting the remote remote communication. Also, we aim to support not only traditional RPC-type request-response interactions but also asynchronous messaging and publish-subscribe patterns. The communication payload is not defined by interface method signatures, as it happens in RPC and SOAP, but in a data-oriented manner using a standard data representation.

By defining the payloads formats in this way we support more interaction patterns and create a platform-independent wire-protocol which can be understood by each of the specific technologies used in each node, thus allowing for multiple platform implementations while maintaining technical interoperability in the system as a whole. In the project, Java on OSGi (Eclipse Virgo) was chosen as the base technology for the unconstrained nodes (desktops, notebooks and cloud deployments) while Android was the base technology chosen for mobile, constrained, nodes. Thus, this technical interoperability was required also in practice, for the reference implementation of the SOCIETIES architecture and its user trials [8], [9].

The semantic interoperability of the platform communication is addressed at two levels: at the core level, where communicating network nodes need to convey information about identity, network and services, and at the payload level, where the service message semantics are understood in an extensible way. The required flexibility and functionality can be achieved by generically supporting data structure definitions and enabling each service to define different structure and respective semantics - a kind of domain specific language.

## IV. SOCIETIES Service and Communications Design

In the SOCIETIES European Project an identity-enabled communication layer was implemented relying on XMPP [10]. The protocol was used as a session-layer protocol where authentication and federated Identinet [2] functions are provided, supporting application-layer control and data on top of it. The client-server architecture of XMPP not only enables the

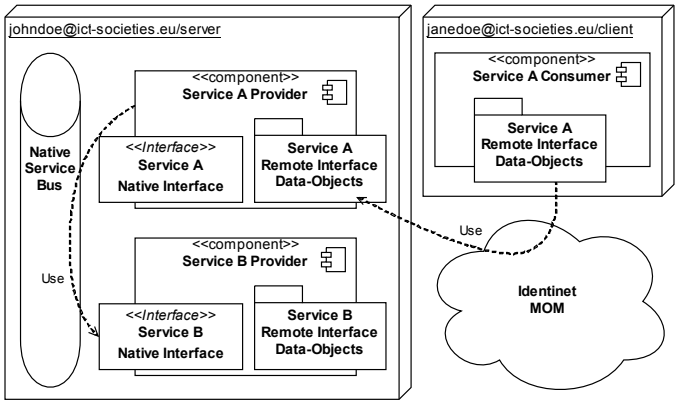


Fig. 2. Service re-use in SOCIETIES

Identinet through theuse of servers and DNS as a resolution mechanism, and enables servers to act as an anonymizing trusted-third-party for most communications [11]. The implementation, dubbed Communication Framework, abstracting the network and the wire-protocol as much as possible by exposing an API with only identity-level and service-level concepts and transparently addressing transport, network and interoperability in data representation.

The payloads carried in XMPP's IQs and Messages have custom namespaces that are under the responsibility of one domain-specific service. They are formally defined using XML Schemas, an interoperable format, which then can be mapped to data structures used in many different languages. These schemas define common data types and semantics ensuring consistency between local and remote interactions. When the Communications Framework implementation receives one such message it routes it locally to the appropriate service based on the specified XML namespace.

The XML namespace (i.e. xmlns) of the payload indicates the payload format to be assumed for parsing purposes, a technical interoperability feature, also defines the semantic meaning of the message. Each service can publish its interface data structures under a namespace, indicating both format and semantics. These specifications could be published in a central registry similar to the XMPP Registrar [12]. Such registry should contain at least reference to the semantic and functional description of that namespace, similarly to what is done with existing XMPP Extension Protocols (XEP) [13] in XMPP. However in SOCIETIES we simply tried to enforce and ad-hoc description of the structures in the XML schemas themselves.

The actual normative binding between service and payload namespace occurs locally in each node. Services that are started in a SOCIETIES node attempt to locally register their payload namespaces in the Communications Framework. If no collisions are detected, the Communications Framework will from then on route incoming messages with the defined payload to the referred service, with no explicit service reference being required.

These services are both accessible locally in the SOCIETIES node and remotely over the network, as illustrated in

Figure 2. Locally, they are exposed in technology-dependent way because there are looser interoperability constraints: within the same node we assume the same software stack is used to implement all services. For the OSGi node the OSGi service discovery mechanisms were used, while in Android the Intent mechanism mimicked that functionality. However, remotely, the services need to be exposed in a more agnostic way.

The binding of services and namespaces is already a common practice in XMPP, as defined in the Service Discovery extension [14], despite some differences between XMPP services and generic SOA services. As advised in the XMPP Extension design guidelines [15], the Service Discovery extension was re-used by the SOCIETIES Communications Framework, making remote services discoverable via this XMPP standard protocol. Any node can query any other node regarding the services it supports, and the response will carry the namespaces of the supported services.

Thanks to the SOCIETIES Communications Framework, only a few lines of code need to be written in order to make a native local service accessible this way through a network. First, an XML Schema version of the data structures needs to be specified, defining also a namespace and describing the service's remote functionality. The Communications Framework provides tools to generate native objects representing those data structures which are used to build a service wrapper. This wrapper basically consists of generic payload reception methods which, depending on the received objects, call specific methods with underlying parameters of that service. All the XML parsing and XMPP packet building and routing is done transparently.

Besides the original IQ and Message communication patterns supported natively by XMPP, the SOCIETIES Communications Framework also supports Publish-Subscribe (PubSub), piggybacking on an existing XMPP extension [16]. A special service within the SOCIETIES node, the Pubsub Client Service, handles PubSub-type interactions. Any service can subscribe and publish to as well as receive notifications from a remote node by interacting with the Pubsub Client Service local API.

## V. Implementation and Results

The SOCIETIES Communications Framework became a core part of the SOCIETIES project, with its API being used both in OSGi and Android by most of the project from the initial stable release in February 2012 until the User Trial #2 in November 2013 [9]. More than 15 developers defined over 40 payload schemas through which they exposed their SOCIETIES services over the network [10]. One of the first steps of the work was to draft an example remote service along with an XEP-like document describing it, which was made available under the Seed initiative [10]. The goal was to drive adoption inside SOCIETIES and to ease the learning curve for developers. While the Communications Framework enjoyed wide adoption in the project, the XEP-like documentation practices, which would form the bases for a SOCIETIES Registrar, unfortunately weren't adopted.

The services were mostly implemented in Java, by implementing an interface called *IFeatureServer* which defines the following request handling methods: *getQuery*, *setQuery* and *receiveMessage*. Inside that method the developer only needs to verify if the received object is of the expected type, dynamically generated from the XML schema, and used it as a data object to execute the necessary business logic. In the case of the *getQuery* and *setQuery* methods an Object can be returned, which is serialized to XML and included in the IQ result. A client that accesses a remote service is required to implement *ICommCallback*, which defines an IQ result reception method and an IQ error reception method. In order to initiate an IQ the client can use the SOCIETIES Communications Framework, and whenever the response arrives the respective callback method is executed.

**Example 1** SOCIETIES Node Service Discovery Response

```
<iq from='user1.societies.local' type='result'
 id='info1' to='client@societies.local/device'>
 <query
  xmlns='http://jabber.org/protocol/disco#info'>
  <identity category='component' type='generic'
   name='Societies Communication Manager'/>
  <feature var='http://jabber.org/protocol/disco#info'/>
  <feature var='urn:xmpp:ping'/>
  <feature var='jabber:iq:last'/>
  <feature var='urn:xmpp:time'/>
  <feature var='http://jabber.org/protocol/pubsub'/>
  <feature var='http://societies.org/api/schema/
       privacytrust/trust/broker'/>
  <feature var='http://societies.org/api/schema/
       privacytrust/trust/model'/>
  <feature var='http://societies.org/api/schema/
       privacytrust/trust/evidence/collector'/>
 </query>
</iq>
```

The services implementing *IFeatureServer* register themselves with the SOCIETIES Communications Framework and reference the Java package containing the XML-bound data objects. The SOCIETIES Communications Framework uses the XML namespace to which this Java package is bound to advertise the support of this service through XMPP Service Discovery [14], as shown in Example 1. One a service has been remotely discovered, remote consumers (or clients, in a more XMPP-oriented terminology) can send XML encoded payloads to them.

**Example 2** SOCIETIES wire-protocol of a CIS Join Request

```
<iq type='set' from='css1@societies.local/device'
 to='cis1.societies.local' id='join1'>
 <community xmlns='http://societies.org/community'>
   <join/>
 </community>
</iq>
```

Example 2 shows a simple XMPP-based SOCIETIES request from CSS *css1* to join CIS *cis1*. Upon the reception of a message, the SOCIETIES platform takes these XMPP identifiers, JIDs, and converts them to an agnostic identity data

structure, abstracting XMPP specifics from the services. Similarly, when a message is to be sent, the inverse happens. This process implements a correspondence between the conceptual SOCIEITES identity model and the underlying communication identifiers. This correspondence also allows SOCIETIES components to be interoperable, at the identification level, with pure XMPP clients.

For implementing the Communications Framework itself, two XMPP open source libraries were used: Smack and Whack, both provided by Ignite Realtime [17], [18]. Also, the open source SimpleXML [19] serialization library was used. A key factor that contributed to the choice of these libraries over others was their compatibility with Android. In order to keep the implementation simple, a significant part of the code is platform-agnostic, running both in Android and standard Java.

Another significant technical detail of the Communications Framework implementation has to do with the modular nature of OSGi. Because the Communications Framework (de)serializes XML payloads transparently to other OSGi services, it would require to dynamically import all of these services data objects, which is seen as somewhat of an OSGi anti-pattern. It caused, beyond performance problems, OSGi to re-resolve the Communications Framework package imports each time a service would no longer be available. For that reason, we decided to make Communications Framework OSGi aware, and to explicitly handle class-loading contexts. This enables the Communications Framework to transparently transform XML to and from Java Objects in the class-loading context of the interested service, effectively maintaining the benefits of OSGi while abstracting the (de)serialization code.

## VI. Discussion and Conclusion

The work presented in this paper explores the possibility of implementing a distributed social SOA over XMPP. It differs from traditional SOA because it attempts to counter the current centralization *status quo* of the web in favour of a service ecosystem composed by user-hosted peers that can behave at the same time as service consumers and providers. However these peers need coordination both in the service domain, where a discovery and routing infrastructure is necessary, and on the identity domain, so that they can establish social and trust relations. In SOCIETIES we attempted to implement such an infrastructure by relying in XMPP, and by adopting as much of the core protocol and its extensions as possible. The result was an open source implementation used by the project partners to implement the services that provided the functionality for the user trials.

As Roy Fielding created REST by re-using the HTTP verbs and enabling a minimalist alternative to SOAP, in SOCIETIES we tried to re-use the rich functionality set of XMPP for a similar goal. Compared to XEP-0072 [7], which defines SOAP over XMPP, the SOCIETIES approach wins in a number of evaluation criteria. First, as recommended in the XMPP Extension design guidelines [15], it re-uses existing extensions relevant for the use case, namely Service Discovery [14] and Publish Subscribe [16]. Second, it makes use of XMPP's pre-existing support for asynchronous messaging and Publish-Subscribe while XEP-0072 leaves interaction patterns for the business logic layer, blindingly executing them using XMPP IQs or Messages. Finally, the resulting wire-protocol is more compact and *XMPP-like*.

TABLE I
Comparison of Architectural-Style Tecnhologies

| | SOAP | REST | AMQP | XMPP User--hosted SOA |
|---|---|---|---|---|
| Typical Deployment Scope | intranets | Internet | intranets | Internet |
| Service Discovery | dynamic, via polling | static, via convention and documentation | out of scope | dynamic, via polling and notification |
| Wire-Protocol Format | XML over HTTP | JSON over HTTP | AMQP over TCP | XML over TCP |
| Payload Structure Definition | machine-readable, integrated with endpoint | human-readable, via convention and documentation | out of scope | machine-readable, via convention |
| Messaging Patterns | request-response | request-response | message routing by key | message routing by identifier |
| Transaction Support | add-on | no | optional | no |
| Typical End-User Role | consumer | consumer | not involved | provider and consumer |

A part of work that was designed but not implemented was the support for a more flexible service discovery mechanism. Beyond having a typical *polling-based* service discovery protocol, we aimed to re-use the mechanism of capability advertisement over presence defined in the Entity Capabilities XEP [20] to advertise client-hosted services. That way, the services exposed by a network node would be propagated along with the node's presence across the user's contact network, enabling an event-driven service discovery environment. Also, Entity Capabilities propagate service information in a very network-efficient way, by passing client fingerprints around. An explicit service discovery request is only required in case the fingerprint hasn't been previously cached.

Table I illustrates the differences in service infrastructure functionality for key architectural-style technologies - SOAP, REST and AMQP - and for the solution proposed in this paper. AMQP clearly differentiates from the other technologies because it does not define a full stack and service ecosystem functionality. Instead it focuses on providing flexible, high-performance messaging with transaction support for enterprise environments. Its use for communication with end-user applications is unheard of, despite technologically feasible, probably because the such application would require an out-of-band method for queue configuration.

The approach presented in this paper clearly distinguishes itself from SOAP and REST in three dimensions: support of reactive presence-based service discovery, flexible messaging patterns and enabling clients to behave as providers. Regarding payload structure definition, our approach is a middle-ground between SOAP and REST: while it sticks with a machine-readable format, it privileges convention over querying for the format on runtime.

The most relevant issue identified throughout the work presented in this paper relies in a mismatch between XMPP and SOCIETIES. In SOCIETIES we'd like that users host services themselves, forming a distributed social service ecosystem. This was a key orientation in the project, drawing from the prevasive-social vision and supported by the adopted privacy architecture. On the other hand, the XSF aims to keep XMPP clients (read *user-side software*) as simple as possible [15], preferring that services are generally provided by XMPP components [21]. While XMPP doesn't restrict the ability for clients to behave as service providers - IQ queries get routed to them just as it happens for components - there are underlying client-server assumptions in a number of XEPs. Most relevantly, the mechanisms defined in Service Discovery [14] and Entity Capabilities [20] don't establish a difference in feature announcement when the client is advertising capability to consume a service versus providing it: clients are always assumed to be consumers while servers/components are the providers. The compromise solution found within SOCIETIES was to treat XMPP components as if they were user-owned, despite lacking the rich identity features that XMPP clients have, and forcing us to drop the idealized presence-based service advertisement previously described.

## ACKNOWLEDGMENT

We would like to thank the SOCIETIES project colleagues which gave ideas and feedback regarding the design of the Communications Framework, namely Micheal Crotty for vision input, Miquel Martin for the technical and philosophical discussions, and Alec Leckey for the outstanding coordination and support. Also, thanks to the rest of the SOCIETIES development team which used the Communications Framework and gave implementation-level feedback, namely: Nikos, Nicolas, Pavlos, Sancho, Eliza, Patrick, Sarah, Stuart, Maria, Liam, Thomas and Olivier.

The work presented in this paper was supported by the European Commission via the ICT FP7 SOCIETIES Integrated Project (No. 257493).

## REFERENCES

[1] C. Schroth and T. Janner, "Web 2.0 and SOA: Converging concepts enabling the internet of services," *IT Professional*, vol. 9, no. 3, pp. 36–41, 2007. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4216107

[2] J. a. Girão and A. C. Sarma, "Identities in the Future Internet of Things," *Wireless Personal Communications*, vol. 49, no. 3, pp. 353–363, 2009.

[3] S. Ludwig, J. Beda, P. Saint-Andre, R. McQueen, S. Egan, and J. Hildebrand, "XEP-0166: Jingle," 2009. [Online]. Available: http://xmpp.org/extensions/xep-0166.html

[4] P. Saint-Andre, J. Hildebrand, S. Dobrov, and V. Saliou, "XEP-0277: Microblogging over XMPP," 2012. [Online]. Available: http://xmpp.org/extensions/xep-0277.html

[5] P. Waher, "XEP-0323: Internet of Things - Sensor Data," 2013. [Online]. Available: http://xmpp.org/extensions/xep-0323.html

[6] L. Johansson, "XMPP as MOM," in *Greater NOrdic MIddleware Symposium (GNOMIS)*, 2005. [Online]. Available: http://www.gnomis.org/presentasjoner/oslo2005/xmpp.pdf

[7] F. Forno and P. Saint-Andre, "XEP-0072: SOAP Over XMPP," 2005. [Online]. Available: http://xmpp.org/extensions/xep-0072.html

[8] K. Doolin, "SOCIETIES completes Enterprise User Trial," 2013. [Online]. Available: http://www.ict-societies.eu/2013/04/23/societies-completes-enterprise-user-trial/

[9] ——, "SOCIETIES Enterprise User Trial #2 Complete RELEVANCE @ ICT2013," 2013. [Online]. Available: http://www.ict-societies.eu/2013/11/12/societies-enterprise-user-trial-2-complete-relevance-ict2013/

[10] SOCIETIES, "GitHub: SOCIETIES," 2011. [Online]. Available: https://github.com/societies/

[11] M. Hansen, P. Berlich, J. Camenisch, S. Clauß, A. Pfitzmann, and M. Waidner, "Privacy-enhancing identity management," *Information Security Technical Report*, vol. 9, no. 1, pp. 35–44, Jan. 2004.

[12] XMPP Standards Foundation, "XMPP Registrar," 1999. [Online]. Available: http://xmpp.org/resources/registrar/

[13] ——, "XMPP Extensions," 1999. [Online]. Available: http://xmpp.org/extensions/

[14] J. Hildebrand, P. Millard, R. Eatmon, and P. Saint-Andre, "XEP-0030: Service Discovery," 2008. [Online]. Available: http://xmpp.org/extensions/xep-0030.html

[15] P. Saint-Andre, "XEP-0134: XMPP Design Guidelines," 2004. [Online]. Available: http://xmpp.org/extensions/xep-0134.html

[16] P. Millard, P. Saint-Andre, and R. Meijer, "XEP-0060: Publish-Subscribe," 2010. [Online]. Available: http://xmpp.org/extensions/xep-0060.html

[17] Ignite Realtime, "Smack." [Online]. Available: http://www.igniterealtime.org/projects/smack/index.jsp

[18] ——, "Whack." [Online]. Available: http://www.igniterealtime.org/projects/whack/index.jsp

[19] N. Gallagher, "Simple XML Serialization." [Online]. Available: http://simple.sourceforge.net/

[20] J. Hildebrand, P. Saint-Andre, R. Tronçon, and J. Konieczny, "Entity Capabilities," 2008. [Online]. Available: http://xmpp.org/extensions/xep-0115.html

[21] P. Saint-Andre, "XEP-0114: Jabber Component Protocol," 2012. [Online]. Available: http://xmpp.org/extensions/xep-0114.html