Introduction to XMPP Protocol and Developing Online Collaboration Applications using Open Source Software and Libraries

Ozgur Ozturk
ozturk@gatech.edu
Georgia Institute of Technology, Atlanta, GA

INVITED TUTORIAL PAPER

ABSTRACT

Extensible Messaging and Presence Protocol (XMPP) is an open, XML-based protocol aimed at near-real-time, extensible instant messaging (IM) and presence information. It has been expanded into the broader realm of message-oriented middleware. Built to be extensible, the protocol has been extended with features such as Voice over IP and file transfer signaling. XMPP protocol has been used by many social networking platforms including gtalk, and facebook; collaborative services like google wave, and gradient; geo-presence systems like Nokia Ovi Contacts; multiplayer games like chesspark, and by many online live customer support and technical support services. In this manuscript, I will introduce XMPP and its extensions. In the tutorials I will demonstrate how you can leverage some of the available open source software and libraries for rapid development of XMPP enabled services and communication platforms. The tutorial slides and supplementary materials will be available on my website: http://drozturk.com/talks.

KEYWORDS: Distance-Learning, Platforms for Collaboration, Social Multimedia and Networks, Soft Computing Solutions for Collaboration Systems, Web- and Internet-Enabled Collaboration, Collaboration Technologies in Industry and Businesses

1. INTRODUCTION

XMPP is an open standard formalized by IETF and continuously extended through the standards process of the XMPP Standards Foundation. Other strengths of XMPP include:

- XMPP servers form a decentralized network similar to e-mail (unless it is used for intranet communication with server-to-server communication deactivated); anyone who has a domain name and a suitable Internet connection can run their own XMPP server and connect their users and services to the XMPP network.
- There are many open source and free server and client implementations implementing XMPP and its many standard extensions (XEPs).
- XMPP protocol is extensible by design. It is easy to extend the protocol for communication of new kinds of information for new services.
- Available open source implementations strive to be modular and extensible as well. It is relatively easy to add new functionality by writing your own protocol extension, implement it as component or module and add it to an existing server or the client implementation.

2. XMPP vs. SIMPLE

XMPP has been competing with SIMPLE (SIP for Instant Messaging and Presence Leveraging Extensions) to be the dominant presence protocol. SIMPLE is a set of extensions to the established SIP protocol (Session Initiation Protocol) that initiate, set up, and manage a range of media sessions, including voice and video. SIMPLE extensions define SIP signaling methods to handle the transport of data and presence. Proponents of XMPP argue that an XML-based data-transport technology is better suited than a signaling technology to handle IM and presence. According to its designers, one major benefit of XMPP is that it can be extended across disparate applications and systems because of its XML base [1]. XMPP has been gaining ground especially in the social networking and instant messaging applications domain.

3. ACKNOWLEDGEMENT

This tutorial is heavily based on the book “XMPP: The Definitive Guide, Building Real-Time Applications with Jabber Technologies” [2] with permission from Peter Saint-Andre.

4. BASICS OF XMPP

XMPP technologies use a decentralized client-server architecture with a direct federation model. Every XMPP entity needs an address, called a JabberID (JID). JabberIDs for users look like email addresses, composed of username and the domain (e.g., ozgur7@gmail.com). XMPP developers usually call this a bare JID, since a full JID, formed by adding a resource identifier to the end of your account address (e.g., ozgur7@gmail.com/web-9z2), is used for routing traffic to a certain connection you have (e.g., one of your connections through the web client) instead of any other connections you might have open at the moment (e.g., via your desktop client or your mobile client).

4.1. Communication Primitives

Client-server communications in XMPP are made over XML streams. Basic unit of communication in XMPP is called a stanza. Existing code libraries abstract away from the raw XML layer; XML stanzas are constructed programmatically. Three types of XML stanzas are exchanged: *message*, *presence*, and *iq* (Info/Query). This section briefly describes them on examples.

The *message* stanza is the “push” method for getting information from one place to another. A simple *message* stanza could look like this:

```
<message    from="ozgur7@gmail.com/wz2"
            to="stpeter@jabber.org"
            type="chat">
  <body>How are you?</body>
  <subject>Query</subject>
</message>
```

It contains *to* and *from* addresses and can have *type* and *id* attributes. The *from* address is not provided by the sending client, but instead is stamped by the sender’s server to avoid address spoofing. *Body* and *subject* are payload elements used in person-to-person chat messages.

Presence advertises the network availability and status messages of entities. It is a specialized publish-subscribe method; people who requested subscription to your presence and authorized by you receive updated presence information when you come online, and go offline, and change your status. A simple *presence* stanza could look like this:

```
<presence from="ozgur7@gmail.com/pda-z2">
  <show>do not disturb</show>
  <status>in a meeting</status>
</presence>
```

In a typical IM application, *presence* information of your contacts is displayed in your *roster*. When you come online, your client software announce your presence to your server and the server handles the rest - both notifying your contacts that you are online and fetching their current presence for display in your client interface.

The *iq* stanza provides a structure for request-response interactions and simple workflows. *iq* stanzas include only one payload. Requests and responses are tracked using the *id* attribute, which is generated by the requesting entity and then included by the responding entity. The *type* attribute included in the exchanged *iq* stanzas helps in establishing a structured IQ interaction between two entities. The following example of my client software getting my roster and then updating it illustrates this kind of interaction. First, it asks my server (gmail.com) for my contact list which is stored on the server, by sending an *iq-get* with empty payload qualified by the relevant namespace:

```
<iq     from="ozgur7@gmail.com/wz2"
        id="rr82a1z7"
        to="ozgur7@gmail.com"
        type="get">
  <query xmlns="jabber:iq:roster"/>
</iq>
```

The server sends back an *iq-result* with a payload containing an *item* element for each contact in her roster:

```
<iq     from="ozgur7@gmail.com"
        id="rr82a1z7"
        to="ozgur7@gmail.com/wz2"
        type="result">
  <query xmlns="jabber:iq:roster">
    <item jid="frnd1@gmail.com"/>
    <item jid="spouse@facebook.com"/>
  </query>
</iq>
```

The client starts a new transaction to add a new contact. Since this is an update operation, client uses an *iq-set*:

```
<iq     from="ozgur7@gmail.com/wz2"
        id="ruZ61vd7"
        to="ozgur7@gmail.com"
        type="set">
<query xmlns="jabber:iq:roster">
<item jid="stpeter@jabber.org"/>
</query>
</iq>
```

Each *iq* request must get a reply. The server acknowledges the update by returning an empty *iq-result*:

```
<iq     from="ozgur7@gmail.com"
        id="ruZ61vd7"
        to="ozgur7@gmail.com/wz2"
        type="result"/>
```

Another important *iq* type is error, which is returned when a get or set request cannot be processed.

5. EXTENSIBILITY AND AVAILABLE EXTENSIONS

XMPP is an extensible protocol as its name implies. The XMPP Standards Foundation (XSF) standardizes extensions to XMPP through a process centered around XMPP Extension Protocols (XEPs) [3]. All XEP documents are listed on XSF's website (http://xmpp.org/extensions/). Many features are available through using these standard extensions. Objectives of some of these extensions will be briefly mentioned here.

Data Forms XEP specifies how a server sends the information necessary for a client to render a form to display to the user and the related semantics for forms processing (such as request, response, submit, and cancel). It defines several common field types (boolean, list options with single or multiple choice, text with single line or multiple lines, single or multiple JabberIDs, hidden fields, etc.), provides extensibility for future data types [4]. A further extension is CAPTCHA Forms XEP where entities send a challenge question to discover whether the sender of an XML stanza is a human user or a robot [5].

In-Band Registration XEP defines an XMPP protocol extension for communicating with XMPP-based instant messaging servers and other services for in-band registration, password change or cancellation of an existing registration. It is extensible via use of data forms, thus enabling services to gather a wide range of information during the registration process [7].

Multi-User Chat XEP defines how multiple XMPP users can exchange messages in the context of a room or channel, similar to Internet Relay Chat (IRC). In addition to standard chat-room features such as room topics and invitations, it defines a strong room control model, including the ability to kick and ban users, to name room moderators and administrators, to require membership or passwords in order to join the room, etc [8].
Publish-Subscribe XEP [9] and *Personal Eventing Protocol (PEP)* XEP [10] and further extensions of PEP (like *User Tune* [11], *User Location* [12], *User Activity* [13] XEPs…) presents a more generalized form of the publish/subscribe model than *presence*. These are used for communicating "rich presence" such as moods, and exchanging "lifestreaming" data, such as microblogs, but they are also being applied to storing personal data, such as bookmarks and client preferences.

Basic XMPP communication itself is not conducive for heavy bandwidth transfers. *Jingle XEP* extension is developed to standardize the negotiation and management of any kind of media session, including voice and video chat, file transfer, and screen sharing. This makes Jingle yet another powerful building block in the XMPP toolkit. Jingle uses XMPP as the signaling channel to set up, manage, and terminate media sessions, whereas the media data itself is sent either peer-to-peer or mediated through a dedicated media relay. Setting up a voice communication includes negotiating media codecs, bitrates and other parameters related to the voice format to be used, deciding whether TCP or UDP will be used for communication, what IP addresses and ports will be used, etc [13].

High bandwidth transfers like the ones mentioned in the previous paragraph bring a heavy load to the servers, if the servers act as a relay between the peers. Such communication is usually made p2p (peer-to-peer), taking the server out of picture after setting up the initial connection. However, firewalls and Network Address Translators (NATs) require the use of advanced mechanisms in setting up the p2p connection. Interactive Connectivity Establishment (ICE) is a powerful methodology for figuring out how to set up media sessions (such as voice and video calls) over the Internet. *Jingle ICE-UDP Transport* XEP defines how a transport method, negotiated via ICE methodology, sends media data using raw datagram associations via the User Datagram Protocol (UDP) [14].

Remote Procedure Call (RPC) is a technology that allows a computer program to cause a subroutine or procedure to execute in another address space, commonly on another computer on a shared network. *Jabber-RPC* XEP defines how to transport XML-RPC payloads, which is a method of encoding RPC requests and responses in XML [6]. Ad-

Hoc Commands XEP [15] defines and extension for advertising and executing application-specific commands, such as those related to a configuration workflow. Typically the commands uses *Data Forms* XEP [4] in order to structure the information exchange. *Remote Controlling Clients* XEP [16] specifies recommended best practices for remote controlling clients using Ad-Hoc Commands. Simple Object Access Protocol (SOAP) is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. *SOAP Over XMPP* XEP [17] defines methods for transporting SOAP messages over XMPP. These extensions can be used in a wide variety of scenarios, including game management, remote instrument monitoring, machine-to-machine communication, cloud computing, integration with enterprise resource planning (ERP) systems, and other applications..

If none of more than two hundred extensions published by XSF suits your needs, it is not very difficult to design your own extension protocol. For detailed recommendations about protocol design, you may consult the *Protocol Design Guidelines* [18].

Of course not all of the extensions can be available in all libraries or all client or server software. How do you find out which features are supported for example by a certain deployment of a certain XMPP server implementation. *Service Discovery* XEP [19] helps us at this point with its two basic discovery methods. One enables you to discover what entities are out there on the network, for example XMPP servers typically host additional entities such as pubsub topics and multi-user chat rooms. The other enables you to discover which features a given entity supports.

6. SOME OPEN SOURCE IMPLEMENTATIONS

XMPP software and libraries have been implemented in most major programming languages and operating systems. Most of them are available as open source projects. XMPP foundation website keeps a list of clients [20], servers [21] and libraries [22]. In my tutorial demonstrations, I will use Eclipse environment to use and/or modify OpenFire Server, Spark Client, and Smack XMPP library [23] which are all Java implementations, and Visual Studio Environment for Jabber-net XMPP library [24] which is a C# .Net implementation.

BIOGRAPHY

OZGUR OZTURK is currently a postdoctoral fellow at the Georgia Institute of Technology. He has been building a collaboration platform using XMPP components to start a company with support from GATech Venturelab. He received his PhD from the Ohio State University in CSE (Computer Science and Engineering) in 2007, following his Master Degree from Oregon Health and Science University also in CSE in 2002. He worked as a Postdoctoral Research Associate in the Department of Plant Biology at Carnegie Institution for Science, Stanford, CA in 2007-2008 and as a Senior Software Developer at Oracle Corporation, Atlanta, GA in 2008-2009. His research interests include Databases, Data Mining, Data Warehousing, Information Retrieval, Information Representation, Natural Language Processing, and Software Engineering.

REFERENCES

[1] http://www.infoworld.com/t/platforms/xmpp-vs-simple-race-messaging-standards-295

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

[3] http://xmpp.org/extensions/

[4] R. Eatmon, J. Hildebrand, J. Miller, T. Muldowney, P. Saint-Andre, XEP-0004: Data Forms, http://xmpp.org/extensions/xep-0004.html

[5] I. Paterson, P. Saint-Andre, XEP-0158: CAPTCHA Forms, http://xmpp.org/extensions/xep-0158.html

[6] D.J. Adams, XEP-0009: Jabber-RPC, http://xmpp.org/extensions/xep-0009.html

[7] P. Saint-Andre, XEP-0077: In-Band Registration, http://xmpp.org/extensions/xep-0077.html

[8] P. Saint-Andre, XEP-0045: Multi-User Chat, http://xmpp.org/extensions/xep-0045.html

[9] P. Millard, P. Saint-Andre, R. Meijer, XEP-0060: Publish-Subscribe, http://xmpp.org/extensions/xep-0060.html

[10] P. Saint-Andre, K. Smith, XEP-0163: Personal Eventing via Pubsub, http://xmpp.org/extensions/xep-0163.html

[11] P. Saint-Andre, XEP-0118: User Tune, http://xmpp.org/extensions/xep-0118.html

[12] R. Meijer, P. Saint-Andre, XEP-0108: User Activity, http://xmpp.org/extensions/xep-0108.html

[13] S. Ludwig, J. Beda, P. Saint-Andre, R. McQueen, S. Egan, J. Hildebrand, XEP-0166: Jingle, http://xmpp.org/extensions/xep-0166.html

[14] J. Beda, S. Ludwig, P. Saint-Andre, J. Hildebrand, S. Egan, R. McQueen, XEP-0176: Jingle ICE-UDP Transport Method, http://xmpp.org/extensions/xep-0176.html

[15] M. Miller, XEP-0050: Ad-Hoc Commands, http://xmpp.org/extensions/xep-0050.html

[16] R. Tronçon, P. Saint-Andre, XEP-0146: Remote Controlling Clients, http://xmpp.org/extensions/xep-0146.html

[17] F. Forno, P. Saint-Andre, XEP-0072: SOAP Over XMPP, http://xmpp.org/extensions/xep-0072.html

[18] P. Saint-Andre, XEP-0134: XMPP Design Guidelines, http://xmpp.org/extensions/xep-0134.html

[19] J. Hildebrand, P. Millard, R. Eatmon, P. Saint-Andre, XEP-0030: Service Discovery, http://xmpp.org/extensions/xep-0030.html

[20] http://xmpp.org/software/clients.shtml

[21] http://xmpp.org/software/servers.shtml

[22] http://xmpp.org/software/libraries.shtml

[23] http://www.igniterealtime.org/projects/

[24] http://code.google.com/p/jabber-net/