International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 5, May - 2014

Multi Protocol Cross Platform Communication Middleware

Dhruv Sangvikar
Department of Computer Engineering
AISSMS's Insitute of Information Technology
Pune, Maharashtra, India

Vikas Tekale
Department of Computer Engineering
AISSMS's Insitute of Information Technology
Pune, Maharashtra, India

***Abstract*— With the introduction of multiple communication platforms available, the internet messaging user community is fragmented into users of different platforms and services. Each service provider has it's own protocol and client implementation. Thus a standard client for instant messaging is lacking. There is a need to standardize communication platforms so that the fragmentation can be reduced and ultimately reducing the data and storage redundancy. XMPP is an open technology for real-time communication[1]. At present the user has to rely on completely different clients, platforms and technologies. This leads to extra usage of computing resources like storage and processing. In short, heterogeneous systems for the same task are present. A common solution to this is a Message Oriented Middleware (MOM). This architectural framework for interoperability can be applied to ease the communication and bridge the gap between the users and service providers. Thus a cross platform and multi protocol middleware can be used.**

Keywords—Multi Protocol, Communication, Middleware, XMPP

I. INTRODUCTION

The internet offers fantastic opportunities to communicate with others. Apart from the modern services like video chat, audio chat and other interaction services, instant messaging has been one of the most useful communication means. As a result there has been a tremendous rise in the number of service providers. Internet giants like Facebook, Google, Microsoft, Yahoo!, etc. each have their own services, implemented in their own proprietary technologies.[19] Because of these many choices, most of the users making use of these services are registered with every different service provider. So every time they have to communicate or chat using instant messages, they have to hop from one service provider to the other. This results in many problems. A few of them may be stated as follows:

1. Every service provider has a different client, which have to be obtained from the respective service providers' portals.

2. Users have to get acquainted with different interfaces and mechanisms for the same task of communication.

3. Different clients also results in extra resource usage like extra computing and storage.

4. User has to keep a tab of the different clients required to use a particular service.

5. In large scale environments, when installation on multiple devices is necessary, the whole process of setting up becomes time consuming and redundant.

6. Functionality provided by one platform may be absent in the other client. These results in user confusion.

7. Completely different interface on the smart phone and tablet devices also leads to the same problems stated above.

8. On resource restricted machines, multiple clients cannot be run (systems like thin clients, etc).

9. With more and more services, and each having separate protocol and client, it is not feasible to use them separately.

Normally protocol building is done with one of the following strategies:

1. Custom protocols

2. Protocol frameworks

3. Horizontal protocols

Now, with most of the service providers having used the custom protocols, it has resulted in multiple protocols for the same task of real-time instant messaging. Thus, due to multiple independent and non-inter-operable protocols, it is necessary to apply a abstraction layer above these services. This middleware will effectively hide the mechanism of each service provider by giving the same working mechanism and interface to work with every service. This solution is basically based upon the MOM architecture. [2]

Some clients exist which provide multi protocol communication, by giving access to the different service providers. But they exist on a single platform – either on desktop computers or for phones. Thus a pure multi protocol and cross platform client is absent which can provide the same functionality everywhere.

The main goal of this work is to provide an abstraction layer above the service layer so that the user is unaware of the underlying platform and service provider without any loss in core functionality, and abstract the underlying protocols and mechanism, thereby implementing the MOM model in SOA systems. Also resource consumption can be minimized by using the same client for different service providers.

The rest of this paper is organized as; section 2 highlights the related works along with their downsides, section 3 discusses our proposed model to overcome the downsides of

the existing work. Section 6 gives the conclusion followed by references.

II. RELATED WORK

With the last decade putting many service providers on the communication map, protocols have been developed at rapid speed. Our review shows that some solutions do exist to partially tackle the above stated problems.

Pidgin is an open source software which provides clients for the desktop platforms Windows and Linux. But it does not provide cross platform solution. Also, due to it be being written in C primarily, its direct porting to other platforms is not possible directly. [3]

Miranda IM is an open source multi protocol instant messaging client designed only for Windows. Thus other platforms and even operating systems are not supported at all. Also, being written in C and C++, it poses portability problems. [4]

Empathy is the default chat application in many Linux distributions. Written in C, supporting multiple protocols, it poses portability issues. [5] Also, the messages are not encrypted using even the basic UTF-8 encoding systems, and is not recommended for enterprise and high security environments. [6]

IBM SameTime is a proprietary solution from IBM, which supports IBM-internally developed multiple protocols and acts as a middleware for other real-time services. This does not support other service providers and is restricted to enterprise environments.[7]

naim is a console client which supports limited number of protocols, with the notable omission of XMPP. Lack of common protocols and graphical interface, makes it highly restricted in its usage. Also, only the desktop platform is supported. [8]

Kopete is a KDE based application with only the desktop client. It supports multiple protocols, but being written in C++ and Qt, poses the problem of being non-cross platform. [9]

Kaidu is multi protocol but not cross platform. Also, written in C++, it supports only the Gadu-Gadu protocol compliant protocols.[10]

InstantBird is another application based upon Pidgins libpurple library and Mozilla's XULRunner runtime environment. It exists only for the desktop platform.[11]

Trillian is cross platform and multi protocol client which has native applications for desktop and smart phone platforms. But it is a proprietary software and has to be paid for. [12]

Imo.im is a web based service which allows to connect multiple services through its interface. But a native client for the desktop does not exist, with the smart phone client also only in development. Also as of 2 March, 2013, it has discontinued its services to develop a proprietary messaging solution.[13]

Jitsi is a multi protocol client written in Java. It can be ported to other Java supported platforms, but till date no such client exists.[14]

Thus, a truly cross platform solution does not exist. Trillian is the closest it gets, but still they have the some limitations in the way they are implemented.

The given implementations are the most commonly and easily available software solutions. Also, solutions which have been discontinued from development have been discarded.

The significance of our development approach is to resolve each of these above cited problems. The development for interoperability will be based upon REST architecture. The development will be targeted such that the solution will be cross platform and target at least two or more protocols. The GUI for different platforms will be as similar as possible in the constraints of the different interfaces of the platform

III. PROPOSED SYSTEM

In the previous section, we have reviewed the existing development work. In this section we will mainly see the architectural framework and technologies which can be used to develop a truly cross platform and multi protocol communication application.

Instant messaging service providers are mostly cloud based and provide services through the REST architecture. Research without common implementation has been done about real-time communication systems using REST.[15] The other commonly used architecture for cloud based services is SOAP. SOAP can be used to transfer XML based messages for events from one-to-many nodes or one-to-one nodes. Implementation details have been researched upon for a SOAP based instant messaging system.[16]

Specifically for instant messaging, Extensible Messaging and Presence Protocol (XMPP) was developed. An advantage of XMPP is that in its very nature it tackles security issues that are not straight forward to solve in the web world and the SOAP and REST protocols of web services. Thus, we narrowed in on the XMPP technology and decided to use the core protocols as described by the Internet Engineering Task Force (IETF).[17]

In short, a Service-oriented architecture will be required to implement a middleware for the said purpose. Service-oriented architecture (SOA) aims to interconnect distributed, loosely coupled, and inter operable components of software owned or provided by different domains. For example, many applications and heterogeneous platforms require a process flow of communication to solve interoperability problem in cross-platform systems. Thus, insuring an inter operable communication between cross-platform systems over Internet is the main problem for Service Oriented Architecture. [18]

The said solution can be divided in following modules: the core module - which will contains the core functionality as described in the RFC 6120, the function call module- which will make use of the core functions written to implement the service and GUI module – which acts as the interface to the user while abstracting the other two modules. The GUI-module will be developed for the desktop and the mobile platform independently, while the function call module will be slightly altered as required.

A. Core Module

Here the basic implementation of the specification as given the RFC 6120 of XMPP is provided. It will provide the basic functionality for the XMPP client to connect with any standard XMPP based service provider.

1. *RFC 6120 Core Functionality*

The purpose of XMPP is to enable the exchange of relatively small pieces of structured data (called "XML stanzas") over a network between any two (or more) entities. XMPP is typically implemented using a distributed client-server architecture, wherein a client needs to connect to a server in order to gain access to the network and thus be allowed to exchange XML stanzas with other entities (which can be associated with other servers).

An XMPP based messaging system works as follows:

Step 1: Open stream

Client: Clients send an open stream packet to server to request a new session.

<stream:stream to='example.com' xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>

where "example.com" is domain name of XMPP server connected to.

Server:Server sends back a XML stream starts with <stream:freatures>, includes requirements of either TLS or SASL negotiation, or both.

<stream:features><starttlsxmlns='urn:ietf:params:xml:ns:xmpp-tls'> <required/> </starttls> <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'><mechanism>DIGEST-MD5</mechanism> <mechanism>PLAIN</mechanism><mechanism>EXTERNAL</mechanism></mechanisms> </stream:features>

Step 2: Encryption and Authorization.

2.1 If server needs TLS negotiation. Client: Clients send a STARTTLS request to server.

<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>Server:Server sends back a message shows the TLS is allowed:<proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>or failed:<failure xmlns='urn:ietf:params:xml:ns:xmpp-tls'/> </stream:stream>

In case of failure, the server closed the TCP connect. Client:If TLS is processed, then clients request a new session:

<stream:stream xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' to='example.com' version='1.0'>

Server:Server responses an XML stream indicating the needs of SASL negotiation.

<stream:stream xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' from='example.com' id='c2s_234' version='1.0'> <stream:features> <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'><mechanism>DIGEST-MD5</mechanism> <mechanism>PLAIN</mechanism> <mechanism>EXTERNAL</mechanism> </mechanisms> </stream:features>

2.2 SASL negotiation

Client needs to choose anauthenticationmethod available from server to carry out SASL negotiation. In case above, "DIGEST-MD5", "PLAIN" and "EXTERNAL" are options.

The "PLAIN" authorization method is the simplest among them. It works as following.

Client:Client send a stream with selected authorization method with base64 encoded user name and password. The user name and password are allocated in format of "\0UserName\0Password". Then, the client sends the following stream to server.

<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='PLAIN'>AG1iZWQAbWlycm9y</auth>

Server:If the server accept the authorization, it sends back a stream with "success" tag.

<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>

or

Server: If the password does not match the user name, or there is an error on encoding, the server will sends a failure stream.

<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>

Round 3 Resource binding (Optional)

Client:Client asks server to bind a resource:

<iq type='set' id='bind_1'><bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/></iq>

or

Client:Client binds a resource:

<iq type='set' id='bind_2'>

<bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>

<resource>someresource</resource>

</bind>

</iq>

Server:Server sends back another <iq> stanza, if the "type" tag is "result", that means the binding is successful, otherwise, it is failed.

<iq type='result' id='bind_2'>

<bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>

<jid>somenode@example.com/someresource</jid>

</bind>

</iq>

Step 4: Request a new session

Immediately after SASL negotiation and/or optional resource binding, clients must establish a session to start instant messaging.

Client:Client request session with server:

<iq to='example.com' type='set'id='sess_1'><session xmlns='urn:ietf:params:xml:ns:xmpp-session'/></iq>

Server:Server sends back a iq stanza showing whether session has been created successful or not.

The successful message will be like:

<iq from='example.com' type='result'id='sess_1'/>

If the server failed to create a session, it will reply a message like below or other type of error messages.

<iq from='example.com' type='error' id='sess_1'><session xmlns='urn:ietf:params:xml:ns:xmpp-session'/><error type='auth'><forbiddenxmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/></error></iq>

Step 5: Client and server exchange XMPP stanzas

If all steps above are successful, then client can send XMPP stanzas to server and receive XML streams. Client can send <iq> stanzas to request roster or other information from server, and use<presence>stanzas to change its presence status. Instance message and other payload can be send via<message>stanzas.

Step 6: Close stream

Finally, if clients want to finish the talk and close the XMPP session, it needs to send a close stream to server.

<presence type='unavailable'/></stream:stream>

Then, server will change client's presence to "Offline" and close TCP connections with clients.

B. Functional Module

The application will be able to provide the following basic functionality:

- Connect and authenticate to the service providers,
- Send and receive messages,
- Login to multiple service providers at a single time,
- See a unified roster from multiple service providers.

C. Graphical User Interface Module

The user will interact with the application using the graphical interface provided using this module. This module will be developed on per platform basis keeping in mind the platform constraints and interfacing mechanisms.

D. Technologies Targeted

Our developement will focus on the following platforms:

- Desktop: Linux and Windows
- Mobile: Android

Thus multiple platforms can be catered and provided for by giving an application on these platforms. Thus the primary problem of multi platform solution solved.

The following service providers use their proprietary protocols in their provided chat services. But they are either compliant with XMPP or provide an XMPP interface to use their services. Thus, using the services through the XMPP core functions is possible for these services. Thus the targeted service providers are:

- Facebook, Inc.
- Google Inc.

Thus multiple proprietary protocols are targeted and unified under a single application. Also, XMPP interfacing is used to deal with these providers

1) Development Language: Java

Java is a "write once, run anywhere" development language. It allows for the source to be compiled once and the byte code generated can be used anywhere on any platform with a Java Virtual Machine (JVM).

This helps in giving a solution which can be easily ported from one platform to the other with much dependancies. Also, the GUI will be designed independently for the targeted platform. This will allow for the abstraction of the core functions from the end user, and provide a unified mechanism and functionality for multiple service providers

Thus, the required Cross platform multi protocol can be implemented using the above technologies.

IV. CONCLUSION AND FUTURE SCOPE

The current implementation caters only to the core functionality of instant messaging. Also, the focus has been on providing a cross platform solution. The extension is to check whether the same solution can be easily ported to the Android platform without writing the whole code from the scratch. For scaling this further, future enhancements and changes can be added as and when needed.

With better connectivity and access to the internet, video and audio conferencing has become quite feasible. With better speeds insant messaging is no longer the de facto of communication. Instead, for easier and more direct approach, multimedia has been used more and more commonly these days. This can be implemented in the future.

Moreover, more service providers can be brought under the same umbrella by implementing their protocols to solve the above said shortcomings.[20] Protocols not directly providing any XMPP interfacing over their proprietary protocols can be targeted. Thus, this can also be targeted as a further development course for the same implementation.

REFERENCES

[1] http://xmpp.org/about-xmpp/
[2] Agent-based MOM Interoperability framework for integration and communication of different SOA applications Najhan M.Ibrahim, Mohd Fadzil Hassan, Zain Balfagih Department of Computer and Information Sciences
[3] https://developer.pidgin.im/wiki/WikiStart
[4] https://code.google.com/p/miranda/
[5] https://wiki.gnome.org/action/show/Apps/Empathy
[6] https://wiki.gnome.org/action/show/Apps/Empathy#Project_Resources
[7] http://www-03.ibm.com/software/products/en/ibmsame

[8] http://naim.n.ml.org/, https://code.google.com/p/naim/

[9] http://kde.org/applications/internet/kopete/

[10] https://gitorious.org/kadu, http://www.kadu.im/w/English:GetInvolved

[11] https://jitsi.org/Main/About

[12] http://en.wikipedia.org/wiki/Instantbird

[13] https://imo.im/about

[14] https://jitsi.org/Documentation/UserDocumentation

[15] Research on web instant messaging using REST web services, Yishan Song; Beijing Univ. of Posts and Telecommunication, Beijing, China, Ke Xu, Ke Liu, Published in Web Society (SWS), 2010 IEEE 2nd Synopsium

[16] A Cross Platform Web Service Implementation Using SOAP By Nan-Chao Huang Submitted in partial fulfillment of the requirements For The Degree of Master of Science in Computer and Information Science

[17] RFC 6120, By P. Saint-Andre, IETF,, Category: Standards Track, ISSN: 2070-1721, March 2011

[18] Agent-based MOM Interoperability framework for integration and communication of different SOA applications Najhan M.Ibrahim, Mohd Fadzil Hassan, Zain Balfagih Department of Computer and Information Sciences

[19] A Study of Internet Instant Messaging and Chat Protocols Raymond B. Jennings III, Erich M. Nahum, David P. Olshefski, Debanjan S

[20] Research Article : Research on Effectiveness Modeling of the Online Chat Group Hua-Fei Zhang, 1 Li-Gang Dong, 1 Jia-wei Sun, 2 and Ying Li 1 1 2 School of Information & Electronic Engineering, Zhejiang Gongshang University, Hangzhou, 310018, China Information Management Center,Training Department, Information Engineering University, Zhengzhou, 450002, China