Name of system: Jitsi
Reviewer: Daniel Kozimor
Date: Monday, November 14th, 2011
Author of software: Emil Ivov
Author of book chapter: Emil Ivov
Five star rating of book chapter: 2
Purpose of system:

Jitsi aims to be a full-fledged telephony client with support for VoIP, Internet video conferencing, instant messaging, and desktop sharing. It achieves this goal in context of two primary constraints. The designers wanted Jitsi to be cross-platform, and thereby were driven to use Java, as a primary implementation language. And the designers needed a robust, modular architectural in order to support many disparate protocol formats, with an easy option to add many more in the future. The latter was a key driver behind using Apache Felix, an open source implementation of **Open Services Gateway initiative framework,** or OSGi, a component management framework which promotes loosely coupled architectures. Unlike Skype, Jitsi is solely a client-side application with no proprietary network. It currently supports video conferencing and telephony via SIP (in fact, it was originally called *SIP Communicator*) and XMPP Jingle protocol. Its Instant Messaging feature supports most major IM protocols, like XMPP used byJabber, Facebook chat and Google Talk, OSCAR, and .NET Messenger Service. At the same time it is architectured to support a robust plugin framework which makes additions easy to include and distribute.

Basic metrics

KLOC: 700KLOC
Project start-up: 2003
Number of major releases:

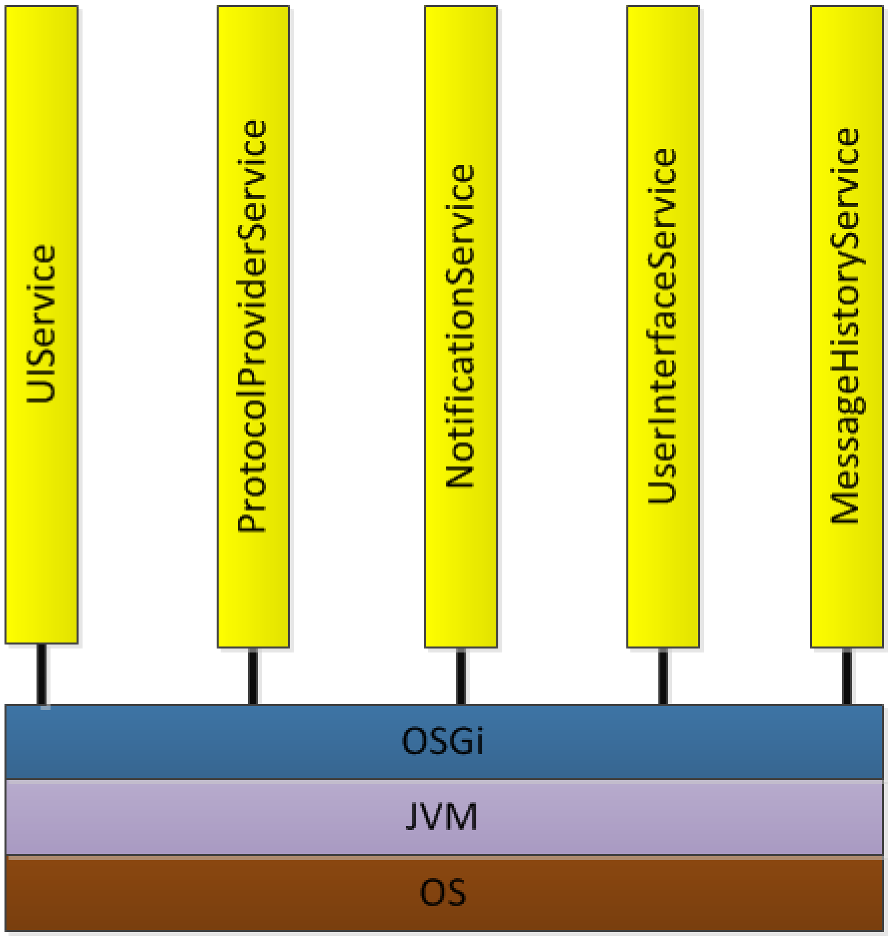
- "Jitsi" has several 'stable' releases since name re-branding. It hasn't hit 1.0 yet.
- 5 major alpha releases since 2006.

Number of developers: 5 core - 20 other contributors
Size of user community or number of installations:
Major stakeholders:
BlueJimp, BlueTone, Greenpeace France, ippi.fr, University of Strasbourg
Use of concurrency:
Concurrency is certainly used throughout the application. UI needs to be responsive while a web conferencing session is active (video and audio). The OSGi framework itself maintains a thread pool in order to do event handling and dispatching. In addition, individual bundles (protocol-related or otherwise), implement their own threads to handle bundle-specific tasks.
Implementation language:
The primary implementation language is Java. There is quite a bit of native code in order to support platform specific features (e.g. Growl for Mac). In addition, to support a number of media-related features (e.g. video codecs, audio handling) third party native libraries are used.
Supporting software: Apache Felix OSGi, JAIN-SIP, Smack library for XMPP

High level architecture

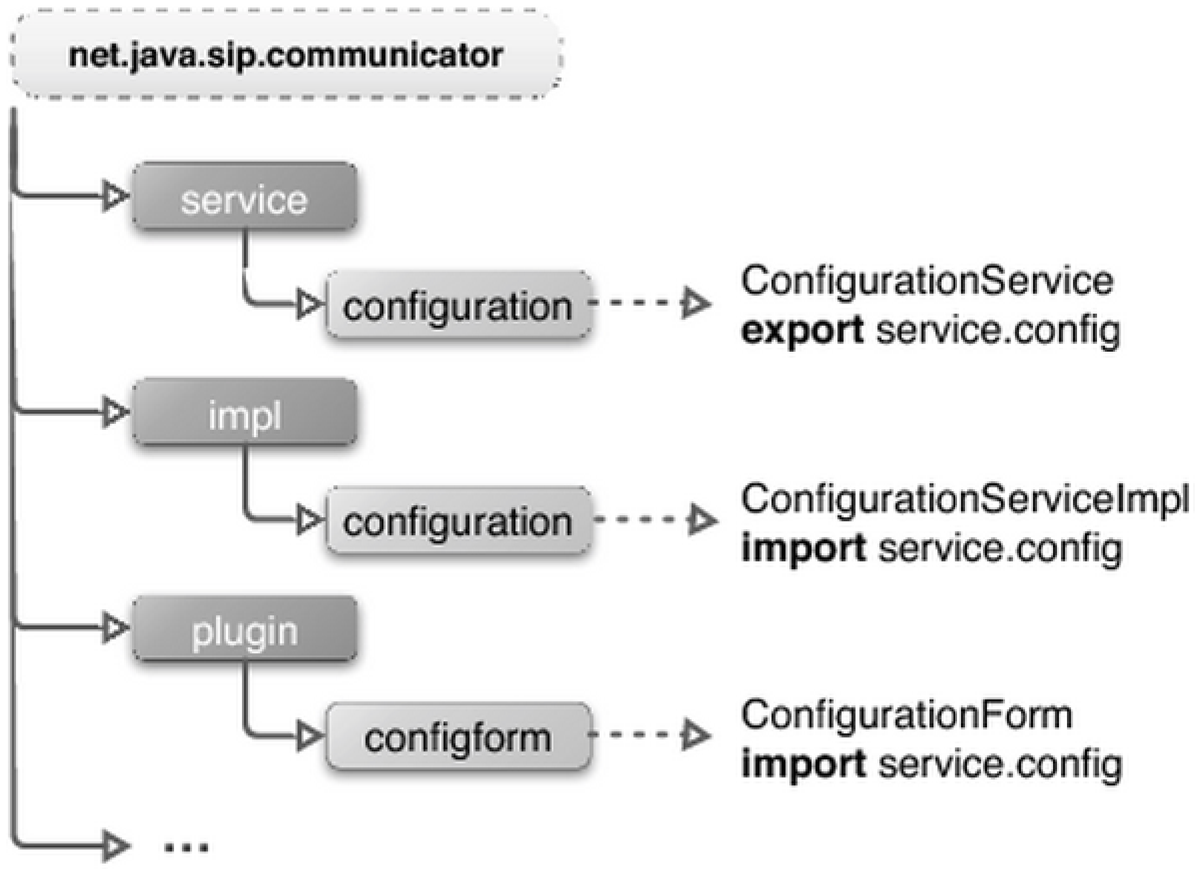
Diagram of software architecture

Jitsi architecture



As a Java based application, Jitsi runs inside the Java Virtual Machine, however business logic is guided by the OSGi framework, depicted as the blue layer sitting on top of the JVM/JRE. In this diagram Individual components are depicted in yellow and only a subset of them are represented in this diagram. Components and plugins are only tightly-coupled to the framework which is why the are shown connect to OSGi layer with a thick line. They are loosely-coupled to all other components because should they need to work with another bundle, they use OSGi query mechanism to grab a reference to another bundle. In this way the framework acts as a notification and messaging broker between component bundles.

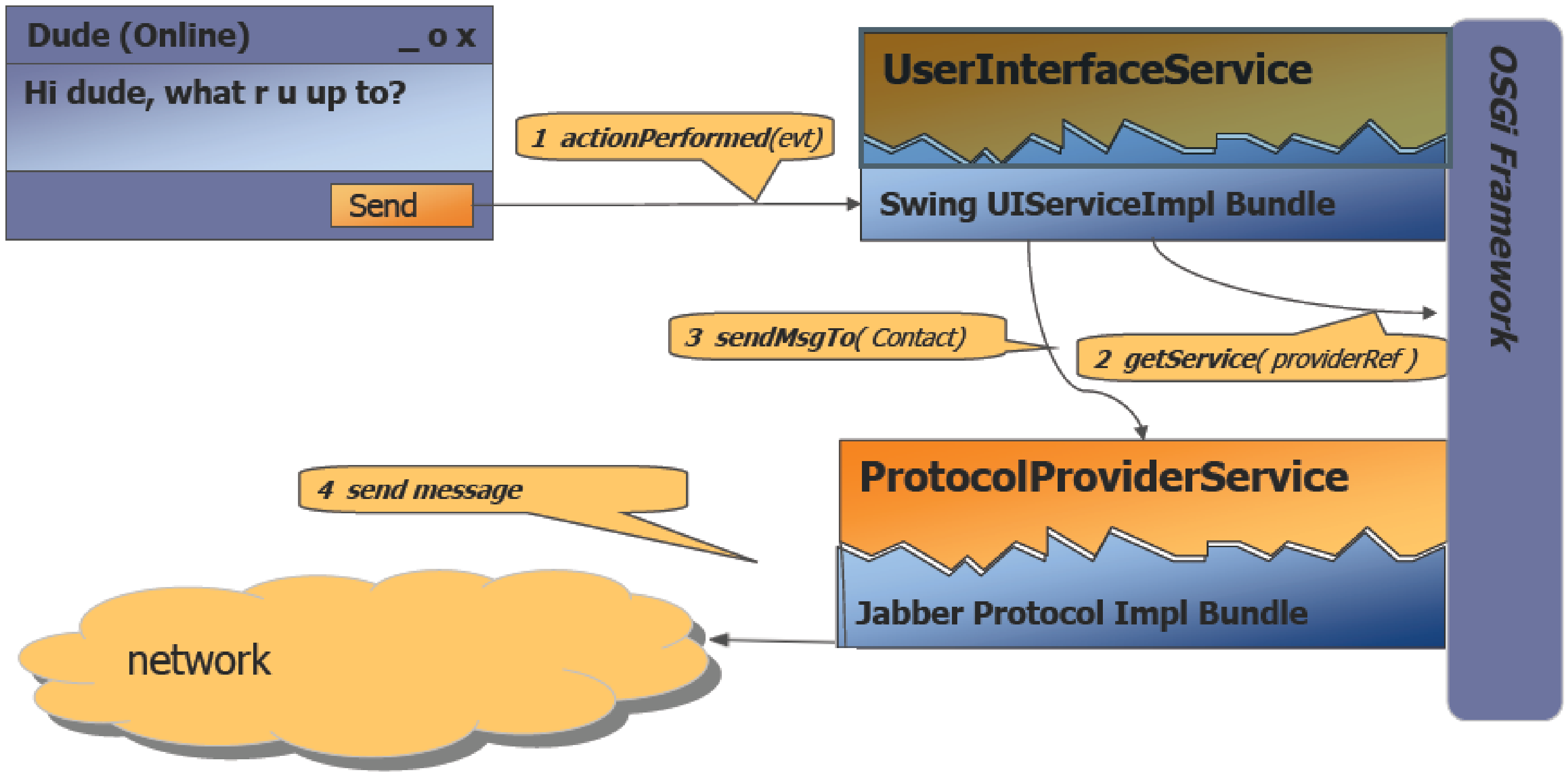
Jitsi Service structure



The diagram depicts the package structure of an individual component bundle. In this example, it is the ConfigurationService. The only classes visible outside the bundle are the interfaces contained within service node (hence the “export” label). The actual implementation of the service is under the impl node.

High level scenarios

- Sending an XMPP message. (Diagram from http://2007.jres.org/planning/slides/132.pdf)



1. User enters a message in a UI Textfield and presses 'Send'.
2. Event is dispatched and caught by the UserInterfaceService component. The UserInterfaceService itself is implemented by SwingUIServiceImpl class.
3. In order to fulfil the action the UserInterfaceService queries OSGi Framework for the ProtocolProviderService for the current working protocol (in this case XMPP/Jabber messaging protocol).
4. sendMsgTo(Contact) is called on the interface and Jabber Protocol Impl implementation class dispatches it across the network.

Data structures or algorithms [Any that are important to the overall architecture]
There aren't any key data structures that are important to the overall architecture of the system.

Control flow and/or data key to the architecture if any

Architectural style:

The architecture of Jitsi is driven by OSGi Framework. Specifically Apache Felix implementation of OSGi. OSGi manages the life-cycle of each plugin and component, and is responsible for starting and disposing them. It also explicitly prohibits bundles from seeing and grabbing other bundles' implementing classes. In addition, it acts as library of all currently loaded bundles, and provides a query api fro any bundle to query for any other service

Major evolutionary changes:

Some changes which Jitsi went through since the initial project started.

- ○ JavaSound used in the early years, but it had major deficiencies, such as no ability to choose audio device (can only use whatever the default device is). Linux implementation of JavaSound uses deprecated OSS drivers. Therefore an alternative sound library was used, PortAudio, a widely supported C library to handle sound.

- ○ Similar issues with Java (specifically Java Media Framework) and Video Capture and Rendering. After JMF, a video caputre framework called LTI-CIVIL was used, and when that proved suboptimal for real-time collaboration, it was re-written from scratch by using native renderes (e.g. Video4Linux, QTKit, DirectShow/Direct3D)

- ○ JMF only supports H.263 and 176x114 CIF format which is a substandard format for video chat. FFmpeg libraries were used instead.

- ○ Support was added for native OS features (e.g. Growl notifications on Mac OS X, as well as Outlook and Apple Address Book integration, desktop capture).

- ○ In 2005 Sip Communicator was completely rearchitectured to use OSGi.

Performance bottlenecks:

There aren't major performance bottlenecks currently. Java Media Framework was an issue in the past given its luck-luster support for real-time audio and video, but Jitsi developers have since moved to native libraries.

Real time:

Jitsi aims to deliver high quality real-time communication, video and audio, hence real time is paramount to proper function.

Notation for architecture:

When describing OSGi based architectures, a layered architecture diagram is used. This is apt because OSGi framework controls every stage of the lifecycle of the managed services, which are loosely coupled, so individual componets sit ontop of the OSGi framework. Components in Jitsi are connected to each other through framework code, so every major component sits independently from each other. OSGi itself sits on top of the JVM layer.

Methodology:

Agile / Open Source.

Appendix:
Kruchten's eight context attributes applied to Brown/Wilson systems

1. **Size:** L = 700KLOC
2. **Criticality:** Med=Company critical
3. **Age of system:** L (since 2003)
4. **Rate of change:** Hi- One major rewrite in 2005. New Protocols added often. New functionality added to existing features.
5. **Business model:** Open Source - Founder runs a company for custom development
6. **Stable architecture:** Lo
7. **Team distribution:** Med=Occasional physical meetings
8. **Governance:** Lo **-** 5 developers account for about 90% of source code contributions, almost all employed by BlueJimp.