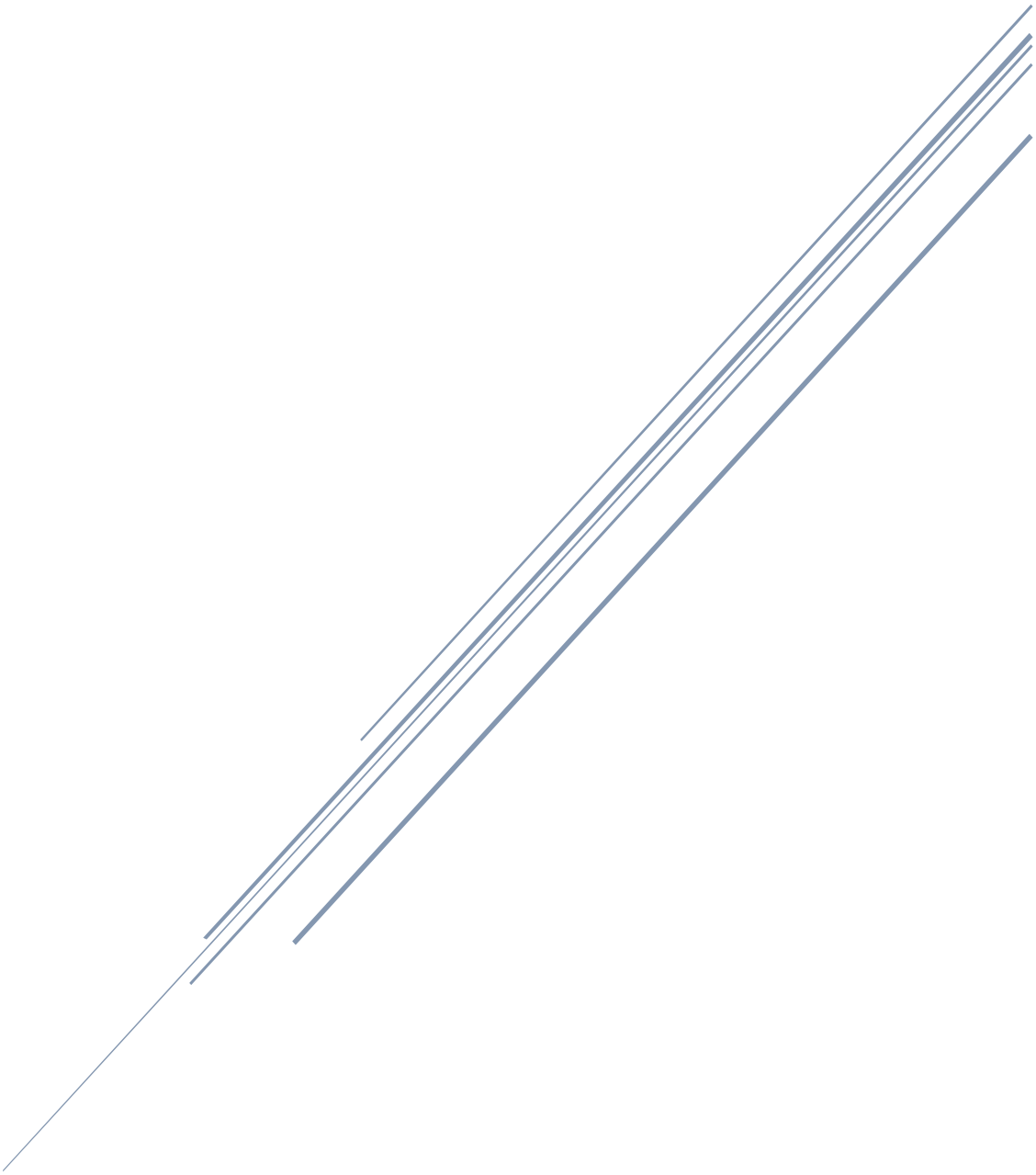
PROXY DEMO

Software Design Document



Maciej Kaniewski

Version history

Version #	Implemented by	Revision Date	Reason
1.0	Maciej Kaniewski	04/07/2014	Initial Design Definition draft

Table of Contents

Introduction

Proxy Server demo project was created based on specification:

Write a proxy server application that can accept requests over a socket, send the request to another server, and then send the response from that server back to the originating client via the proxy.

The functionality of the server should be something simple like adding two numbers, appending a string to the request, etc.

The application should be able to handle multiple concurrent requests

To implement such a Server I have decided to base network communication on well proven and tested network communication library netty.io and Java NIO Channels.

Design Considerations

Design requirements

Main requirement for this project is that a proxy server is always configured to pass communication only to one server specified in configuration file it is not a part of design for client to send to proxy any configuration messages which would change proxies remote server.

- Proxy server should not do any modifications to data forwarded and received from both server and client.
- Proxy server is not interested in any data it is not parsing, this data and is not taking any action other than forwarding this data either to server or client.
- Proxy server can handle multiple concurrent connections

Design Overview

Introduction

The Design Overview is section to introduce and give a brief overview of the design. The System Architecture is a way to give the overall view of a system and to place it into context with external Systems. This allows for the reader and user of the document to orient themselves to the design and See a summary before proceeding into the details of the design.

Dependencies

Project is using four open source libraries they all have a safe open source licences and are widely used across industry.

Netty

First one is netty it is well established network framework designed for creating advanced network projects it has multiple protocols already implemented and it is very simple to create new ones. It has a well proven Architecture and has well established position on the market, used by companies like Facebook, Avast, Red Hat, and Twitter...

Licence - Apache 2.0

SLF4J

Is a logging facade created for java serves as a simple abstraction for various logging frameworks (e.g. java.util.logging, logback, log4j) allowing the end user to plug in the desired logging framework at deployment time.

Licence - MIT

Below are two libraries which are used only for testing purposes and are not a part of final release bus still have a licence which is safe for company use.

Mockito

> *"Mockito is a mocking framework that tastes really good. It lets you write beautiful tests with clean & simple API. Mockito doesn't give you hangover because the tests are very readable and they produce clean verification errors."*

Licence - MIT

jUnit

Well known java unit testing framework

Licence - CPL

Version Control

Project is using Git as a Version Control System, it is well proven distributed VCS it is hosted on Github:

https://github.com/firegnom/proxy-demo

Build

Maven is a build automation tool used primarily for Java projects. Maven addresses two aspects of building software: First, it describes how software is built, and second, it describes its dependencies. Contrary to preceding tools like Apache Ant it uses conventions for the build procedure, and only exceptions need to be written down. An XML file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins. It comes with pre-defined targets for performing certain well-defined tasks such as compilation of code and its packaging. Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the Maven 2

Central Repository, and stores them in a local cache. This local cache of downloaded artifacts can also be updated with artifacts created by local projects. Public repositories can also be updated.

Project is structured in to parent POM project and 4 sub projects which are modules of parent project.

parent – is a maven parent project which is holding all other modules

common – this module stores protocol classes decoder's encoders and shared classes

server – this module stores server classes and its tests

client – this module stores Client classes and creates executable jar file for client.

proxy - this module stores Proxy project, configuration for this project and tests, it is compiled to executable jar file

Continuous integration and code quality

Project is configured to use Jenkins as a CI server it is accessible via link:

http://jenkins.firegnom.net

CI server is a very important part of the project all the builds are stored on CI server there is a history of previous builds , Jenkins is also integrated with SonarQube previously known as Sonar which is a great tool to verify code quality and is accessible here :

http://sonar.firegnom.net/dashboard/index/1

Communication protocol

Message sent between client and server has well defined structure it contains two fields of information:

1. int number
2. String message in UTF-16 encoding.

When message is sent to the server a ServerMessadeEncoder is used to encode POJO in to ByteBuf used by netty to transfer Byte stream of data. When message is received from the server ServerMessageDecoder is used to convert data in ByteBuf passed by netty to Java Object Server Message.

Message transferred using network is as follows:

4 bytes - int number

4 bytes - int message size

Message size* 2 bytes - String message

A sample message

This is a sample message which sends number **255**

And message "**test**"

00	00	00	FF	00	00	00	04	00	74	00	65	00	73	00	74
Int number				Int size				String message							

Hexadecimal dump from logger:

```
         +-------------------------------------------------+
         |  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f |
+--------+-------------------------------------------------+----------------+
|00000000| 00 00 00 ff 00 00 00 04 00 74 00 65 00 73 00 74 |.........t.e.s.t|
+--------+-------------------------------------------------+----------------+
```

Use Cases

I will describe here three different execution scenarios, the first one is a default client server communication and no proxy server running in-between. Second scenario is a communication between client and server which is going via proxy server. Third Scenario is proxy server which is configured to proxy all connections to my remote server firegnom.net on port 80

Scenario 1

This is a default client server communication there is no proxy server running and message is sent directly from client to server.

Step 1

Start server listening on default port that is **49001**:

```
java -jar server-1.0-SNAPSHOT.jar
```

Step 2

Start Client and sending to number **123** and message **test123** directly to server on port **49001**:

```
java -jar client-1.0-SNAPSHOT.jar localhost 49001 123 test123
```

Scenario 2

In this scenario communication between client and server is passed via proxy server and message is sent from the client to proxy server running on port 49000 and from proxy server it is forwarded to server running on port 49001. Message is later processed by the server and response is returned to the client via proxy server.

Step 1

Start server listening on default port that is **49001**:

```
java -jar server-1.0-SNAPSHOT.jar
```

Step 2

Start proxy server listening on default port that is **49000** and forwarding all traffic to server running on port **49001:**

```
java -jar server-1.0-SNAPSHOT.jar
```

Step 3

Starting Client and sending to number **123** and message **test123** to proxy server listening on port **49000** which will forward this connection to server running on port **49001**:

```
java -jar client-1.0-SNAPSHOT.jar localhost 49000 123 test123
```

Scenario 3

In this scenario proxy server acts as a bridge to remote server running on host firegnom.net and port 80 which is a default http server there is a Jenkins server running on this machine and all connections will be forwarded to that Jenkins server. Direct link to Jenkins server is: http://jenkins.firegnom.net

Start proxy server listening on default port that is **49000** and forwarding all traffic to server **firegnom.net** running on port **80**:

```
```

```
java -jar server-1.0-SNAPSHOT.jar config.properties
```

To test this just open browser and visit address http://localhost:49000/ and connection done from the browser will go through the proxy server

Unfortunately proxy server is not configured with its own proxy configuration so it won't be possible to set proxy to reach firegnom.net from inside some companies' intranet, but firegnom.net can be replaced with any other server accessible in the intranet it can be changed in config.properties file:

```
#Properties file responsible for proxy configuration

local.port=49000
remote.host=firegnom.net
remote.port=80
timeout=60
threads.boss=1
threads.worker=16
```

References

1. Wikipedia.org