Message system X

Table of contents

1. Project Overview

2. Architecture Requirements
   2.1 Overview of Key Objectives
   2.2 Architecture Use Cases
   2.3 Stakeholder Architecture Requirements
   2.4 Constraints
   2.5 Non-functional Requirements
   2.6 Risks

3. System Definition
   3.1 Architecture Overview
   3.2 Logical view
   3.3 Development view
   3.4 Process view
   3.5 Physical view

4. Key Decisions

5. Architecture Analysis
   5.1 Scenario Analysis
   5.2 Approach Analysis

1 Project Overview

General

We are developing Message System X, which is a chat application where users can connect to a server and send messages to everyone connected or private messages to specific user utilizing peer-to-peer connection.
The application supports both TCP and UDP communication and is developed using only Java with no external dependencies.
It will be possible to configure the application to emulate communications delay, message loss and failure rate per 100 messages. It will be easy to configure these different settings will inside the application.

Guidance for stakeholders

These are simple guidance for each stakeholder which points out most important parts for each one. We still recommend that you read through the whole documentation.

Developers: Chapter 2 - Risks, Chapter 3 section 3.2 & 3.3.

Users: Chapter 2 - Use cases, Chapter 2, Chapter 3 section 3.2.

2 Architectural requirements

2.1 Overview of Key Objectives

During the review of Message system X's feature and quality list a few key objectives were uncovered.

These are:

1. Performance: *As required in the feature list, with a system load of up to 100 messages that are sent per second, the delay between request and response must be less than 1ms.*

2. Modifiability: *The user should be able to easily configure the application to emulate certain behaviours in the communication infrastructure; such as communication delay and message loss.*

2.2 Architecture Use cases

Stakeholder list

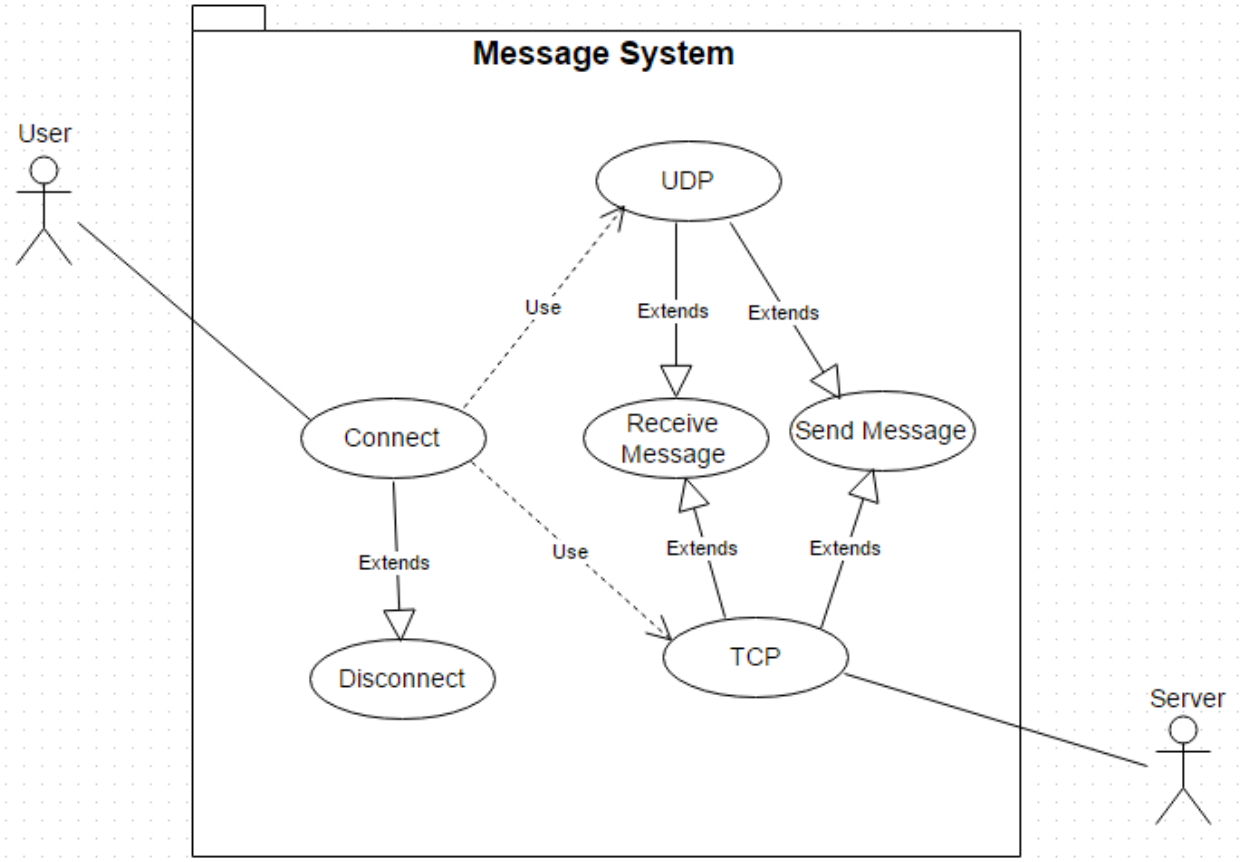
Stakeholders involved in the Message system		
Stakeholder	**Role**	**Main interest**
Software architect	Responsible for architectural design. Architectural decisions are tradeoffs among competing quality requirements.	Feedback on architectural design. Improve insight in the stakeholders´ quality concerns.
Designer/Developer	Responsible for downstream design and implementation of the system.	Clarity and completeness of the design.
User representative	Represents a candidate end-user of the system.	Functionality of the system.

User concerns

Mainly described by what the user wants to be able to do with the system. We present use cases from the above stated requirements.

Use case

In this model you can overview use cases. Each use case is defined below the model for more specific information.



UC1. User wants to connect to the message system

Precondition

The message system is installed and running.

Postcondition

The message system presents a message that the user has successfully connected.

Level

Functional requirement #1, #4

Main scenario

1. Starts when the user wants to connect.
2. The user starts the message system.
3. The user presses the connect button.
4. The message system presents a success message.

Alternative scenarios

4a. The message system could not connect. Presents error message.

UC2. User wants to disconnect

Precondition

The user is connected to the message system.

Postcondition

The message system presents a message that the user has disconnected.

Level

Functional requirement #2

Main scenario

1. Starts when the user wants to disconnect.
2. The user presses the disconnect button.
3. The message system presents a success message.

UC3. User wants to send a message

Precondition

The user is connected to the message system.

Postcondition

The message sent by the user is presented to all services connected by the message system.

Level

Functional requirement #3, #4

Main scenario

1. Starts when the user wants to send a message.
2. The user writes in a message into the message box and presses send.
3. The message system receives the message and sends it to all connected services.
4. The user is presented with the message they send by the message system.

Alternative scenario

3a. The message system could not receive the message. An error message is presented to the user.

4a. The user was not presented with it's own sent message. The message system had an internal error. User is presented with an error message.

UC4. User wants to send a message to a specific user

Precondition

Two users.

Postcondition

The message sent by the user is presented to both the receiving user and sending user.

Level

Functional requirement #3, #4

Main scenario

1. Starts when the user want to send a message to a specific user.
2. Both users set the peer2peer property in their properties file to true.
3. They then start the application and fill out incoming and outgoing ports that match each other (eg. First user receives on 1234 and sends on 1235, while the second user has the opposite).
4. One of the users enter and message and press send.
5. Both users are presented with the sent message.

Alternative scenario

5a. The message was not presented. Error has occured on the other users side.

UC5. User receives a message

Precondition
Two users, one to receive and one to send. Or just one user that will receive from the message system.

Postcondition
The message sent by the sending user or message system is presented.

Level
Functional requirement #3, #4

Main scenario

1. Starts when the user wants to connect.
2. The user starts the message system.
3. The user presses the connect button.
4. The message system presents a success message.

Alternative scenario

1a. The user is already connected and receives a message from an existing user. The message is presented to the receiving user.
4a. The message system server is offline and the user presented with an error message.

Developer concerns

Developers typically use the architecture as a reference for developing the system and/or assembling system components. These will therefore be their main concern.

2.2 Stakeholder Architecture Requirements

Functional requirements

1. A service can register to listen to incoming messages.
2. A service can de-register.
3. A service can send a message.
4. A service can receive a message.
5. Incoming messages are handled on a first come first served basis.
6. It should be possible to configure the communication infrastructure to emulate communication delay.
7. It should be possible to configure the communication infrastructure to emulate message loss.
8. Different types of configurations should be supported (e.g, messages sent by a particular service, between particular services, all communication).
9. It should be possible to define specific profiles for sequences of messages (e.g., the failure rate of the message should switch between 1% and 5% every 100 messages sent).

2.3 Constraints

Technical constraints

- Programming language - Java exclusively is a requirement enforced by the stakeholders.
- User of external libraries or framework - Using external libraries or frameworks are prohibited by the stakeholders.
- System behavior: First come first served basis when receiving messages is a functional requirement.

Business constraints

- Schedule: The time frame is limited to only three weeks.

2.4 Non-functional requirements

1. For a load of up to 100 messages that are sent per second, the real delay of transmitting a message should be less than 1 ms.
2. It should be easy to configure the extra behavior of the communication infrastructure.

Table 1: Quality Attribute Scenario 1: For a load of up to 100 messages that are sent per second, the real delay of transmitting a message should be less than 1 ms.

Element	**Statement**
Stimulus	A Service sends a message to the Message system.
Stimulus source	Message system receives the message.
Environment	Many services uses the Message system which has a load over 100 messages per second.
Artifact	System
Response	Message sent
Response measure	Transmitting message under 1 ms.

Table 2: Quality Attribute Scenario 2: It should be easy to configure the extra behavior of the communication infrastructure.

Element	**Statement**

Stimulus	A requirement regarding the extra behaviour of the communication infrastructure changed.
Stimulus source	The process of development.
Environment	The system is currently under development and new decisions were made regarding target requirement.
Artifact	System
Response	The design of the system as a whole change according to fulfill the newly changed requirement.
Response measure	Cost and time it takes to reconfigure.

2.5 Risks

Table 3: Architectural Driver Priorities

#	Architectural Drivers	Section Discussed In	Importance	Difficulty
1	**Scenario 1** *Load of up to 100 messages*	**1.3**	**High**	**High**
2	**Scenario 2** *Easy to configure*	**1.3**	**High**	**High**
3	**Requirement 1** *Service Register*	**1.1**	**Medium**	**Low**
4	**Requirement 2** *Service Deregister*	**1.1**	**Medium**	**Low**
5	**Requirement 3** *Service*	**1.1**	**Medium**	**Low**
6	**Requirement 4** *Message Manager*	**1.1**	**Medium**	**Medium**
7	**Design Constraint 1** *Java*	**1.2**	**Low**	**Low**
8	**Design Constraint 2** *Message order*	**1.2**	**High**	**Medium**

3 System Definition

3.1 Architecture Overview

In general the system is based on socket communication between two endpoints. There are two different types of setup for the Messenger system X. First one is all communication which follows a Client-Server pattern. In this case a server must be provided first in order for consumers, eg client to connect and establish a binding between these points. When the connection is established the server is the host and serves all clients which are connected to it with messages sent from clients registered to the server.

Second one is a specific single client to another single client communication, which uses the pattern called Peer-to-Peer and is configured in a way that both clients acts both server and client.
To separate these two type of strategies for communication the system will use different service providers which will set up specific configurations depending on which one is chosen.

3.1.1 Alternative designs

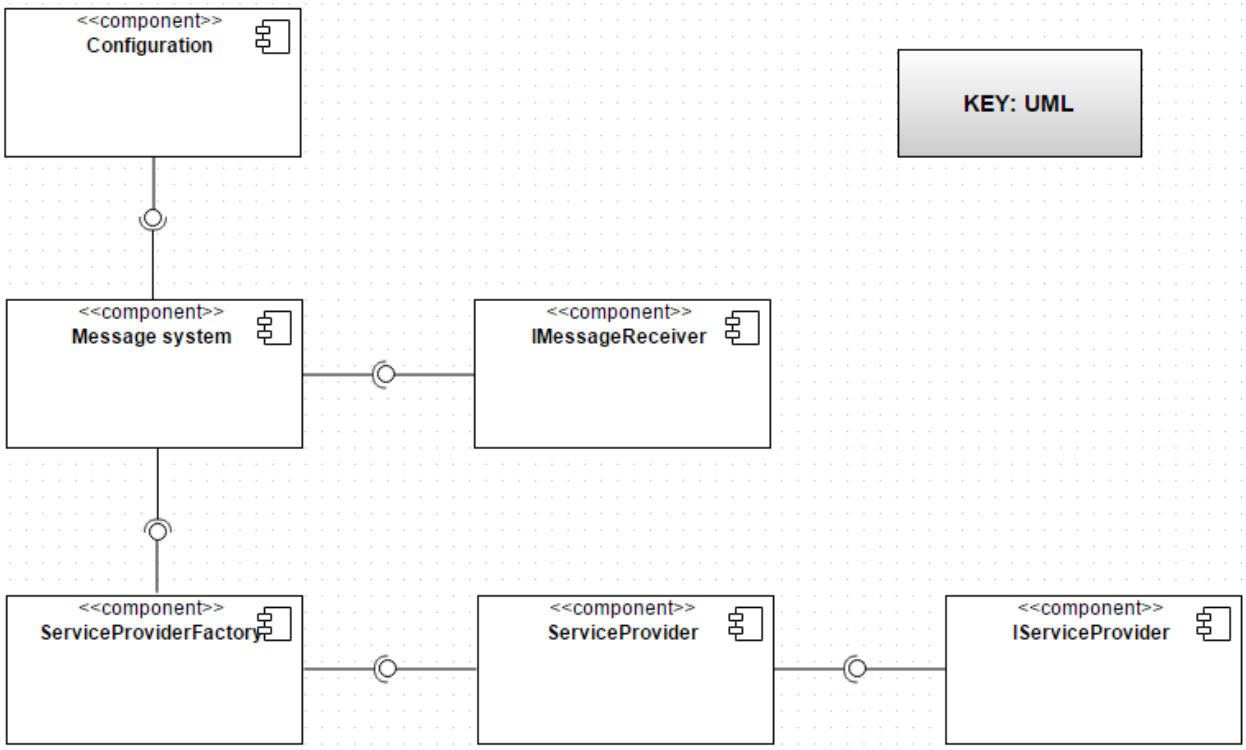
- Client/Queue/Client
- Service-oriented

3.1.2 Decision

We have considered our options and decided to use two different patterns.
For the all communication we are using a Client-Server pattern with TCP socket and in single specific to single specific communication we have chosen Peer-to-Peer pattern utilizing UDP socket.

3.2 Logical view

The logical view is concerned with the functionality that the system provides to end-users. The view is mainly produced for designers/analysts showing the structure.



Rationale

This illustration of an UML sequence diagram fulfills all the functional requirements of the application. It gives a general overview for the system as a whole.

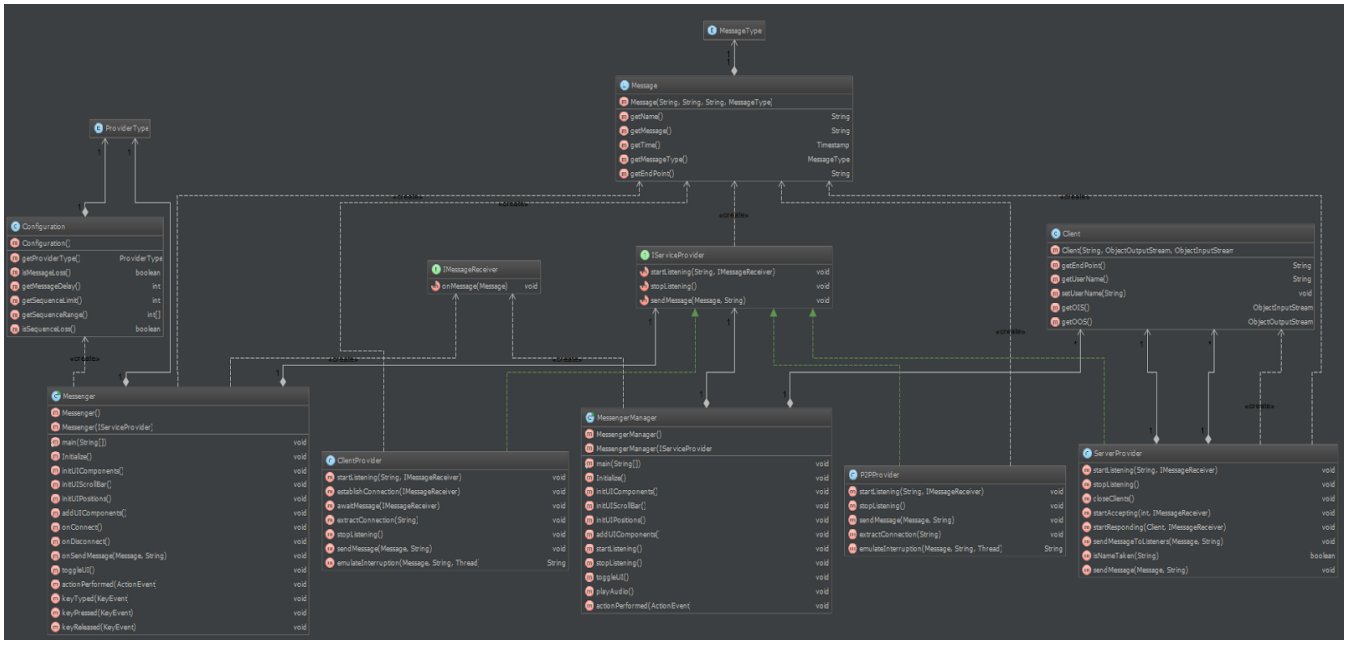
The top-level decomposition separates functionality for Service from functionality for Server. The main motivation for the decomposition are the quality requirements modifiability.

By providing each client with a own service it is easy to configure communication from a particular service, between particular services or even all communication.

We have chosen to utilize a client-server pattern to illustrate the various ways of communication between service in client and server.

3.3 Development view

The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view.



See fullsize image here: https://drive.google.com/open?id=0By

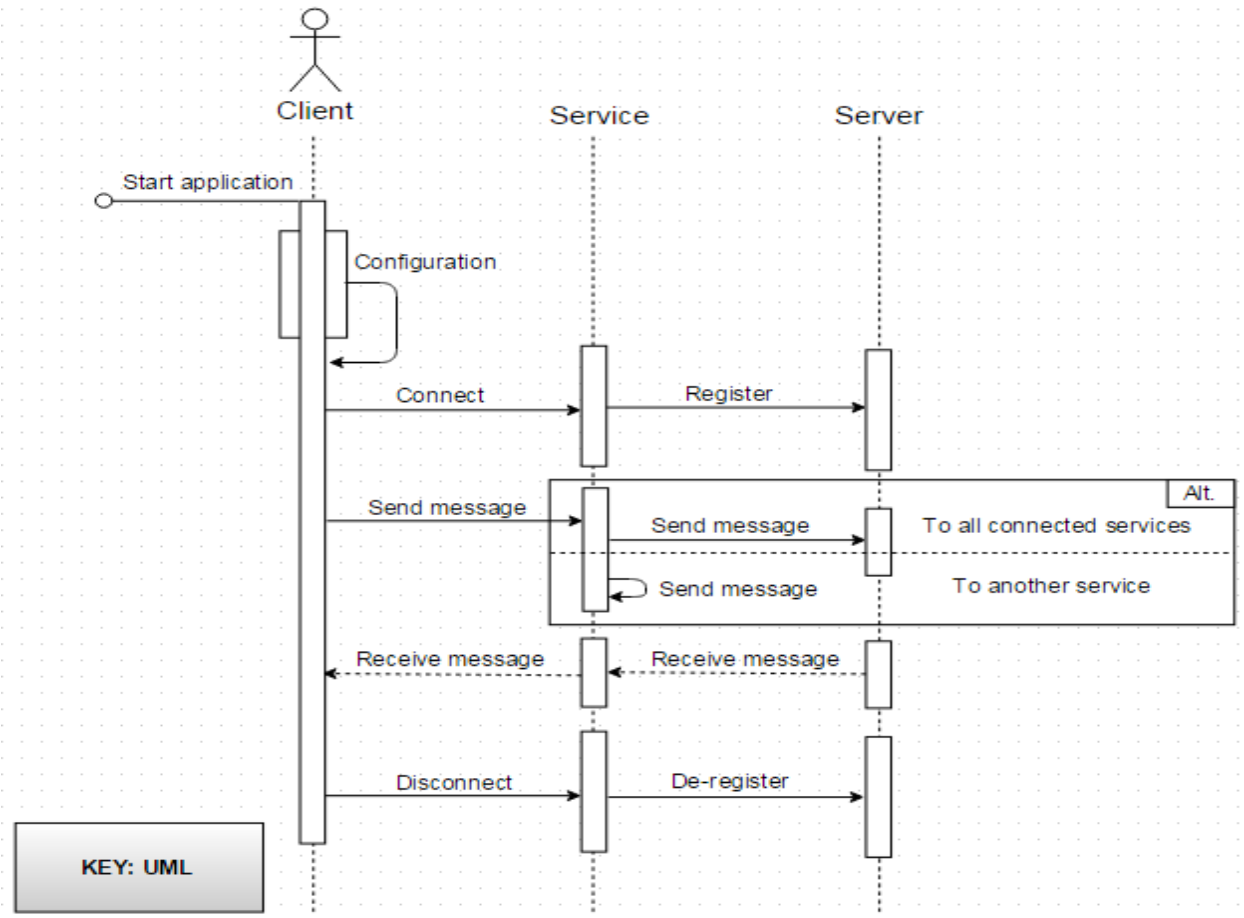
Rationale:

This UML component diagram represents the subsystems and their dependencies that are derived from the system as a whole.

While decomposing the system as a whole we got five sub systems which all are given own responsibilities. Each sub system satisfies one or more requirements from the requirement list found in chapter 2 within this documentation.
The message system component is one of the core components which will allow functional requirement 1, 2, 3, 4 and 5 stratified. Configuration is the second separation of concern which we applied as a dependency on the Message system. It gives the user the ability to configure the system in such a way that we fulfill quality requirement 2, functional requirement 6, 7, 8 and 9.

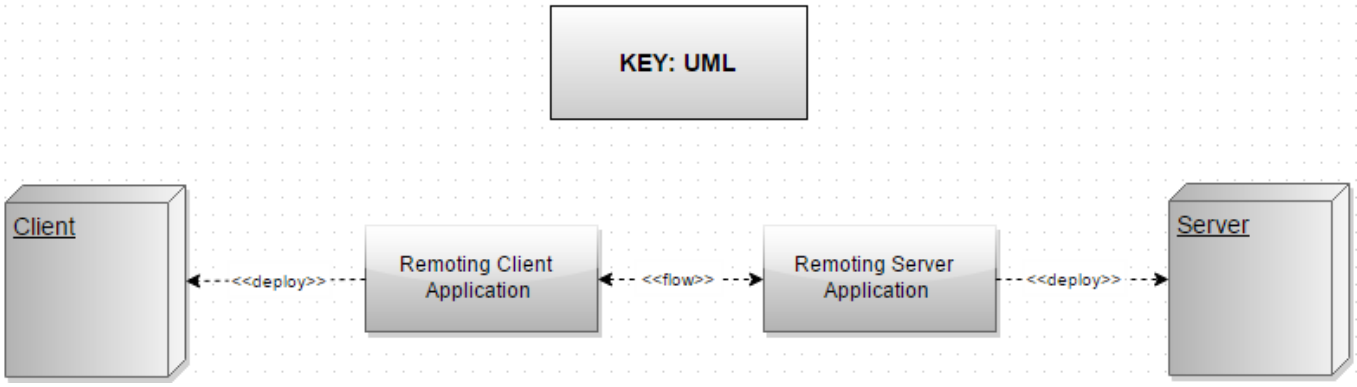
3.4 Process view

The process view shows the processes/workflow rules of a system and how those processes communicate with each other.



3.5 Physical view

The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. This view is also known as the deployment view.



4 Key decisions

Key Decision: <<Configuration>> Decomposition of the system into two socket types and construction of a component named Configuration.

Description
By using a properties file that stores variables in it we are able to easily change the configuration and there of easily modify the behaviour of the message system.

Business Drivers
The key business requirements that drive the decision Quality requirement 2 in section 2.4

Technical Drivers
The key technical requirements that drive the decision Functional requirement 6, 7, 8 and 9 in section 2.2

Approach	We chose to put each major function in a component and assign it requirements from the list. This to make it easier to configure and modify.
Benefits	Modifiability, easy to configure to a low cost of money and time.
Drawbacks	At possible re-installation or re-location of the application, loss of previous configuration is a drawback. This is because the application will be storing the configuration in a configuration file or something alike that is not part of the actual code at startup.
Drivers Realized	Drivers that have been realized by this approach is #1 Easy to configure, #4 Message Manager.
Notes	**N/A**

Issues/Considerations	The configuration file follows a set of key=value pair that is essential to how the settings are being stored. There is also no way to update the configuration in real time as the application is running. If any changes are made they will take effect after the application has

	restarted.

Conclusion	The plain text key=value approach offers easy configurability to users that lack the code/data-structure knowledge of java or xml for example.

Key Decision: <<Communication types>> Providing two different types for communication.

Description
We chose to decompose all communication, single specific to single specific into two separate types of communication. TCP socket with client-server pattern for all communication and UDP socket with Peer-to-Peer pattern for specific communication between two endpoints.

Technical Drivers
The key technical requirements that drive the decision Functional requirement 1, 2, 3, 4 and 8 in section 2.2

Approach	We chose to split communication into two types. First one TCP and second UDP. For this we created two different service providers which provided needed configurations for these techniques.
Benefits	We are working on the lowest network level which give us high modifiability and better control of our program.
Drawbacks	Consuming a lot more time than using a web api for the same purpose.
Drivers Realized	Drivers that have been realized by this approach is #3 Service register, #4 Service de-register, #5 Service, #6 Message manager
Notes	**N/A**

Issues/Considerations	Since we are not using any already optimized way of socketing, we have to try and realize the issues that we may or may not encounter as we go along.

Conclusion	Benefits with using both protocols in our application, gives us the ability to determine the usefulness of our configuration and provider adaptability. For

	example, providing us with information regarding possible changes/issues with connection (TCP) vs connectionless (UDP).

5 Architecture Analysis

5.1 Scenario Analysis

When we developed our design we had to perform a critical trade-off regarding the quality requirements. In order to make our application easy to configure as required in quality requirement 2 we chose a component based model which will ensure the requirement is fulfilled.

5.2 Approach Analysis

Analysis of Architectural Approach			
Scenario #: 2.4.1	For a load of up to 100 messages that are sent per second, the real delay of transmitting a message should be less than 1 ms.		
Attributes	Performance		
Environment	Normal operation		
Stimulus	A Service sends a message to the Message system.		
Response	Message sent		
Architectural decisions	**Sensitivity**	**Tradeoff**	**Risk**
KD4.1			**R1**
KD4.2			

Risks

- **R1.** Since this requirement haven’t been fulfilled it doesn't puts a risk on the system per se but as a risk towards itself as it’s not accomplished as required.

Analysis of Architectural Approach			
Scenario #: 2.4.2	It should be easy to configure the extra behavior of the communication infrastructure.		
Attributes	Modifiability		
Environment	Normal operation		
Stimulus	The configuration file changed.		
Response	The configuration is updated on next application startup.		
Architectural decisions	**Sensitivity**	**Tradeoff**	**Risk**
KD4.1		**T1**	
KD4.2			

- Trade-offs

  T1. Achieve an easy configuration and thereby fulfilling not only this important design requirement but also many functional requirements, was a massive trade-off. We had to put our full focus on the configuration aspects and thus we could not place any importance on **KD4.1**.