Programming of Distributed Applications

Laboratory work nr. 1

Message-based integration that would allow for asynchronous communication between the distributed components of a system

FAF 141 student:
Ana-Maria Brinza

Supervisor :
Mihai Pecari

Chisinau 2017

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to illustrate the work done upon the 1st laboratory work on Programming of Distributed Applications. The main objectives of this work would be:

- Study of Messaging Agents
- Developing a messaging agent communication protocol;
- Concurrent treatment of messages;
- Choosing the transport protocol
- Choosing and developing the message storage strategy;

1.2 Tasks

This laboratory work has more tasks. The basic task is to implement a Message Queue:

- Development of the communication protocol.
- Implementing the queue of messages

The other task for this work would be:

- Implementing the message storage mechanism
- Implementing the routing mechanism of messages
- Implementing the publisher-subscriber

In this work, not all of them where implemented. The message broker was implemented and also a message storage mechanism was implemented.

1.3 Definitions/Abbreviations

- **Message Broker** an intermediary program module that translates a message from the formal messaging protocol of the sender to the formal messaging protocol of the receiver.

2 Application Architecture

The architecture of my application is illustrated in figure 1. So basically we have a sender application, which has the send messages function; Receiver application - has the get message function and basically it get's the messages from the queue.

The Message Chanel is represented by the queue based on JSON which are implemented by notibroker protocol.

Also, there are two external files: *data.txt* and *received.txt* which initially where designed in order to store the data externally.

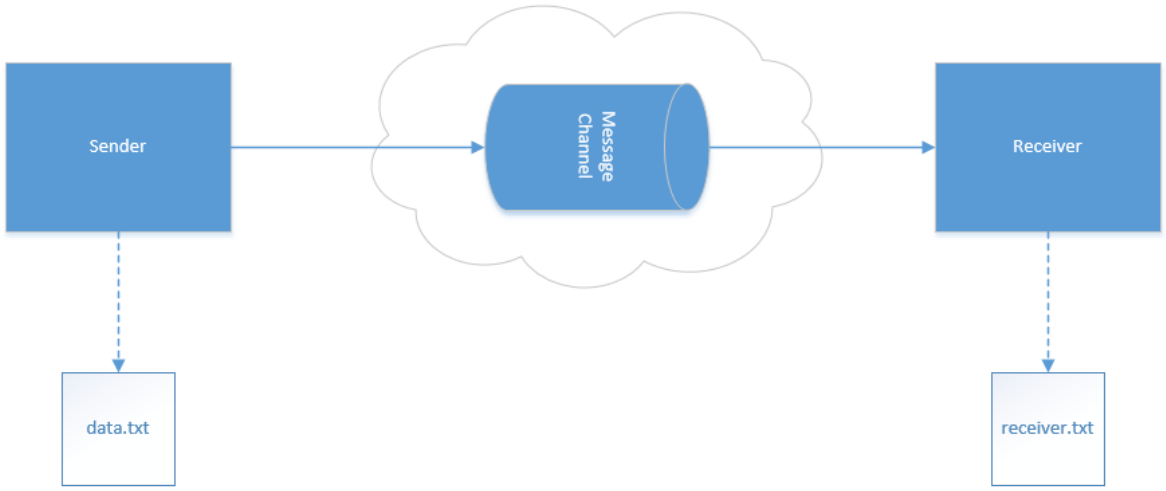

Figure 1: The application architecture

The implementation is described in the next chapter of this work.

3 Implementation

This project work is based on 3 main components:

- Message Broker
- Client-producer (the sender)
- Client-consumer (the receiver)

The Message Broker *(manage.py)* listens on localhost:14141 for messages from clients. Client-producer *(sender.py)* sends each second a message, which is defined by user of the application and also has an id which is incremented, to the broker. Client-consumer (receiver.py) polls each second for a new message from the broker.

In order to start the application/the broker we have to run *manage.py*. This script will access the *run_server* function from notibroker, where host name and port is specified.

Next, to send messages *sender.py* is ran. It has the *send_message* function and also it sends the data to a external .txt file *data.txt*

In order the receiver to have the messages the *receiver.py* is ran. It has the *get_message* function and receives the messages from the queue.

All the implementation and the source code is on the link on the git hub repository, indicated in the *References*

4 Conclusion

This Laboratory Work is a good start to get familiarized with Message Broker and how it actually works. A message broker is an intermediary program module that translates a message from the formal messaging protocol of the sender to the formal messaging protocol of the receiver. Message brokers are based on message queues and it stores some messages.

Nowadays there are a lot of message brokers' software, for example Apache Kafka, Apache Qpid, RabbitMQ(in python), etc; all of these have a lot of functionalities and can be helpful. The one implemented in this work is a basic one, with basic functions - being an intermediary between the sender and receiver and storing the bunch of messages that have to be sent. As it had learning purpose it's not very advanced and there aren't a lot of functionality.

5 References

[1] https://en.wikipedia.org/wiki/Message_broker

[2] https://github.com/anamariabrinza/PAD_labs