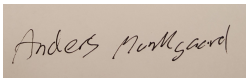
System Integration: ChatX

Anders Munkgaard, Jackie Engberg Christensen, Michael Dolatko, Patrick Petersen
December 17, 2015

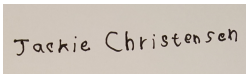
University College Nordjylland
Teknologi og Business
Sofiendalsvej 60
Phone: +45 7269 8000
Fax: +45 7269 8001
http://www.ucn.dk/

Theme:
System Integration: ChatX
Project period:
Spring 2014

Author(s):

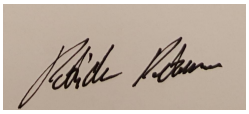


Anders Munkgaard



Jackie Engberg Christensen

Michael Wit Dolatko



Patrick Rasmus Petersen

Supervisor: Gianna Bellé & Michael Holm Andersen
Printed Copies: 0
Total Pages: 20
Appendix: 1
Completed: December 17, 2015

Synopsis:

This report documents the process that has been used to create ChatX which is a chat program with a client-server structure. The purpose of ChatX was to work with system-integration, and that has been done using principles from large scale development. The report explains which methodology has been used, what thoughts and considerations were made during the design phase, which technologies have been utilized and how they have been implemented. Even though the program does not implement all its features, ChatX is extensible and able to integrate and be integrated with further systems.

Contents

1 Overview

1.1 Introduction

This report has been worked on as part of the system integration course, with the purpose of learning to develop an application that makes use of multiple technologies taught in the course. The application is known as ChatX, an instant messaging system that provides client-side RSA encryption for its users and will make it difficult for unwanted third party listeners to understand the messages. This report will describe how the process was planned, how the application was designed, the implementations of our solutions and lastly conclude on the project, alongside with thoughts and considerations of what could be improved on in the application and the process.

2 Process

This part of the report will cover the process used whilst developing ChatX. An agile method was adopted, and Pair Programming was added here for harder tasks.

2.1 Kanban

Due to time pressure and considering how agile Kanban is, it was deemed the best methodology to follow for ChatX. Kanban has few rules and encourages changing flow of process if it helps a development team to deliver more value to the software. It also puts emphasis on how important visualization of work flow is. Figure 2.1 shows an example of a the Kanban board used.

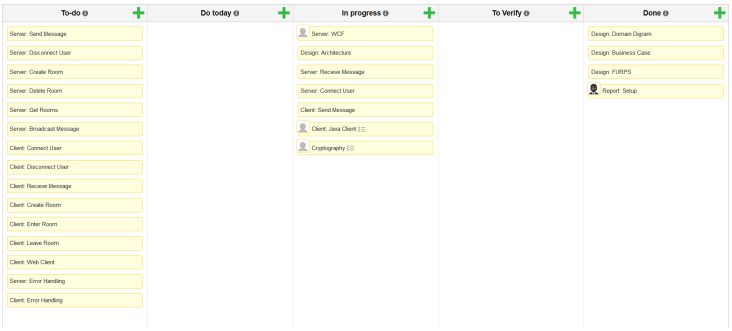


Figure 2.1: Kanban board

This Kanban Board has five columns: To-do, Do Today, In Progress, To verify and Done. The To-do column on the Kanban Board is used to keep track of the tasks that are left to be completed in order to complete the entire project. The next column is Do Today, tasks from To-do are picked and then put over to Today. This helps keep track of how fast the work flow is in the team and whether it can handle more tasks each day or needs to take a step back and do fewer. The In Progress column shows which tasks are being

currently worked on, it helps to show which tasks are already taken so that code work will not be duplicated and hours wasted. Once a task is completed it is then put over to the To Verify column, where other team members have to take a look at the results of the task and verify it has been made properly. Once a task has been verified, it is then moved to the final column which is named Done. This column shows all tasks which have been completed.

2.2 Reflection

Unfortunately, the Kanban rules and board that were set have barely been followed, due to time pressure and how frequently the system design was changed before a final design was decided on. In retrospect it would have been better if a methodology was decided upon earlier rather than as late as it happened in this project and putting more effort into following the rules. Instead of communicating through the Kanban board, ideas and tasks were handled by direct communication.

2.3 Code Standards

Since this project makes use of two different programming languages it would be unwise to not set a code standard. So a code standard was set for each programming language.

2.3.1 Java

Class names must start with a capital letter. Variables must start with lowercase letters, private ones should be refereed by using the ”this” keyword. Curly brackets are to be on their own lines at all times. Method names must start with a lowercase letter.

2.3.2 C#

Class names must start a capital letter. Variables must start with lowercase letters, private ones should be refereed by using the ”this” keyword. Curly brackets are to be on their own lines at all times. Method names must start with a capital letter.

3 System Design

3.1 Business case

ChatX is a chatting system which will allow users to communicate with each other from where ever they are and without fear of the messages being read by unwanted users, as long as they use a platform that supports the application and have a Internet connection.

3.1.1 FURPS

FURPS is a architectural checklist. It helps describe what areas needs to be focused on, and an overall view of what ChatX's purpose is.

3.1.1.1 Functionality

Functionality deals with requirements the customer has set for the project. To help the customer determine the requirements, a question to ask is: What problem is the system going to solve? And with ChatX, users that are in need of communication with other users from a long distance on a secure connection, making sure that messages are private, and only seen by the intended users.

3.1.1.2 Usability

Usability describes how accessible ChatX is for the users. Is it easy to use? Is the GUI design intuitive? A way to check if the users like the aesthetics of the application, and more importantly, can figure out how to use it, would be to prepare usability tests and alpha/beta tests as they can give crucial feedback on how user friendly the system is.

3.1.1.3 Reliability

Reliability describes how reliable the system should be: How much downtime can it handle? Are failures predictable? How is the system going to recover in case of failure?

Due to ChatX's architecture it is possible to setup several servers at once and if one fails, then the other servers can be used instead. In the worst case scenario - i.e. all servers shut down - then it would not take too long to install the server software on a different machine and ChatX would continue working.

3.1.1.4 Performance

Performance describes how fast an application must be, what the highest allowed response time is and how much memory is needed.

ChatX is a chatting system, meaning that messages has to be sent and received as quickly as possible, however we can only assure a fast reliable service at the server end.

3.1.1.5 Supportability

Supportability, the last of the FURPS principles, describes how maintainable an application is after deployment. It also describes how easy it is to install and configure.

For ChatX, two installations are required: the server, and the client. The machine running the client also has to install java, and the machine(s) running the server and service has to have access to a supported message queue system.

3.2 Architecture

ChatX's architecture will be a client-server architecture. The client will call the server whenever a user requests to use the system and the server will keep track of how many users are online and where each message is supposed to be sent to.

3.2.1 Domain Model

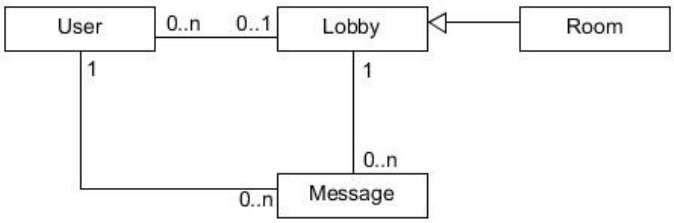


Figure 3.1: Domain Model over ChatX

The domain model consists of four classes. The user class is where users are identified and if they are regular users or administrative users, a user can only be in one lobby at a time but a lobby can have multiple users. A user can also send zero to many messages. The message class is used for sending messages between users, a user can have zero to many messages, whilst a message can only belong to one user. The lobby is where all users start. From the lobby a user can create a room and choose which users are to be included in it, Or join a room. The room class inherits the functionality of the lobby class.

3.2.2 Communicationdiagram

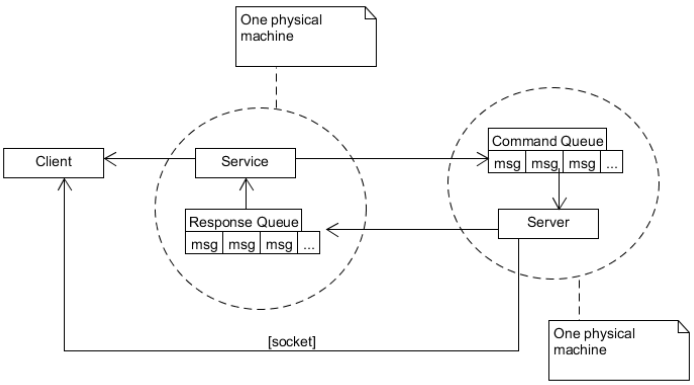


Figure 3.2: Communication-diagram

As seen in the diagram above, figure 3.2, the client asks for information at the service. The service then communicates with the queue system, in this case RabbitMQ, that then communicates with the server. The server always listens to queues that is in RabbitMQ, and as soon as a new message is in the command queue, which is the implemented way of saying messages that has not been dealt with yet, then it sends the messages out to those people who is in the room using the socket connection which is open constantly from the server to each of the clients.

3.3 Cryptography

To make the communication secure the clients implements an RSA encryption to encrypt all messages sent. RSA is a Public- Private key (asymmetric) encryption algorithm. RSA encryption is a standard for encrypting and is practically impossible to decode due to prime factorization. The general idea is that it is easy to find the product of two large primes, but it is hard to factor a large product and find the primes.

RSA was originally made by Ronald Linn Rivest along with Adi Shamir and Len Adlemanand published in their paper A Method for Obtaining Digital Signatures and Public-Key Cryptosystems[1].

To be able to apply RSA encryption, 6 variables are needed; d, e, n, p, q and φ(phi). d, p, q and φ(phi) should be kept secret at all time to keep this encryption secure. Let p and q be a prime number, lets for simplicity say

$p = 7$ and $q = 13$. In practice these 2 numbers would be larger depending on the numbers of bits used for encryption. n is defined as in equation 3.1.

$$n = p \times q \tag{3.1}$$

In this case $n = 91$. Additional n have the bit-length dependent on security of the encryption. If the encryption should be 1024 bit, when n would be a 1024 bit number. φ or $\varphi(n)$ is found by using equation 3.2.

$$\varphi = (p - 1)(q - 1) \tag{3.2}$$

For this example $\varphi = 72$ and it is now possible to find e which is the public exponent. e is a random prime and have two requirements which must be met; e must be a relative prime of φ, ie. e and φ have no common factors and e must be an integer such that $1 < e < \varphi$ and $\gcd(e, \varphi) = 1$. For this example let $e = 7$. The public key is now available as n and e is known and can be given out.

Lastly we calculate d which is the private exponent using extended euclidean algorithm. Equation 3.3 is used for finding d.

$$d = e^{-1} \bmod \varphi \tag{3.3}$$

From equation 3.3 $d = 31$ and it is possible to further check if this is correct by using equation 3.4 which in this case it is, and the private key is now defined by n and d.

$$e \times d \bmod n = 1 \tag{3.4}$$

Now where both the public and private keys are assigned, they can be used to encrypt and decrypt messages. If the message "Hi" were to be decrypted it would first need to be translated into a number, eg. as 8 and 9. It is now possible to encrypt both letter individually by using equation 3.5 which gives the output cipher text C, and as well as decrypt it again with equation 3.6.

$$C = M^e \bmod n \quad \text{Where M} < \text{n} \tag{3.5}$$

$$M = C^d \bmod n \tag{3.6}$$

In the computer world when encrypting data, a padding is often used. Padding is often used to fill out byte blocks. For instance, if the message "Hi!" would had the byte block [48 69 21]. This block size is only of 3 bytes, but the encryption algorithm might read a block of 8 at a time which would mean the block needs 5 more bytes. There is multiple ways of padding a

message, such as zero-padding [48 69 21 00 00 00 00 00], padding with the same value as the number of bytes [48 69 21 05 05 05 05 05][2]. When the padding is done a block can be sent to the encryption algorithm to produce the cipher text.

4 Implementation

The implementation section below will explain what technologies were used and how they were implemented into ChatX.

4.1 Server

The Server is written in C#, it is the part of the system that handles the users and which rooms they are in. The server is split up with the three layer architecture. Model, Controller and GUI.

The Model layer consists of two classes, User and Room. The User class is used to handle the users in ChatX. The Room class keeps track of which Users are in it.

The server has an interface which contains the necessary methods to implement any messaging queue system if RabbitMQ should ever be replaced. For now, RabbitMQ is the message queue system that listens to commands from the service and, depending on what the service requests from the server, sends messages back to the service.

The Controller is where the servers logic has been implemented. It has one class: ServerController. It keeps track of how many users are online in the system, which rooms those users are in, how many rooms there are and also which usernames are already taken. ServerController uses the singleton pattern, meaning there can only be one instance of ServiceController on the server. This has been done because otherwise it would be difficult to handle which users and rooms belong to which instances. Service Controller has a event handler that checks what commands it recieves from Service, the event handler looks into what command the message has and sends it to the appropriate method. ServerController consists of seven methods, JoinRoom, SendMessage, RequestRooms, LoginRequest, VerifyAndLockUsername, and ReleaseUsername. Whenever a user logs in, a socket connection is created between the Server and users logging in their rooms, which allows users to communicate with eachother. Lastly ServerController makes use of the same

Config file used in the Service. It helps showing how the messages from RabbitMQ should be understood for both sides.

The GUI for the server is to control the server. It can turn on the Server, so it starts listening for any incoming messages from the Service, and it can be turned off. It can also connect to a specified service IP.

4.2 Webservice

The webservice is written in C#, and is implemented using Windows Communication Foundation (WCF).

The function of webservice is to be a broker between the client and the server(s); whenever a client interacts with the system, be it sending a message or requesting a list of rooms, it needs to go through this service. The service will then contact the server(s) through a queue system, currently RabbitMQ, through which the servers also sends their response. The communication can be seen in figure 3.2.

The service consists of:

- The exposed functionality available to the clients. This is done through the usual way with an interface as the service contract, and an implementation.
- A server distributor. The idea of this is to handle what servers are available, what their IP's are and what port-numbers the socket-connections should be created to go to.
- A message queue driver. This handles all functionality related to the messaging queue system.
- A config class file. This file statically contains the formats of the actual messages send and received through the messaging queue system.

The first three of the above items all inherit from the interfaces IChatXService, IServerDistributor and IMQDriver respectively. The first interface is implemented, as this is common, good practice for services. The next two has interfaces to make it easier to replace the actual implementations at a later state of development and even deployment. At the moment, the implementation of the server distributor only distributes to one server, and the only messaging system implementation uses RabbitMQ. But since the implementations are coded against an interface, the only change needed to switch to another implementation is at the initialization of the object. Below, in figure 4.1, the service is illustrated:

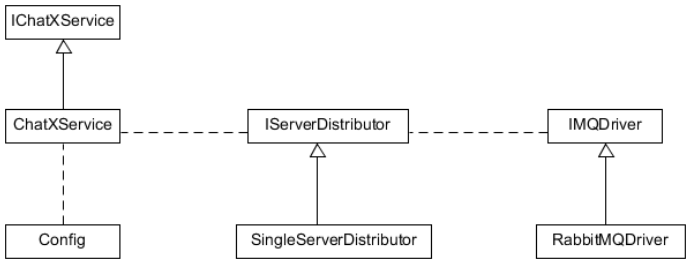


Figure 4.1: ChatX Service Architecture

The lines between ChatXService, IServerDistributor and IMQDriver are dotted to illustrate the statelessness of the service.

5 Conclusion

The purpose of this project was to make a system, which integrates different technologies. In the process of developing this system, the design went through a lot of phases, each making the system more flexible and scalable. Even though scalability is not a focus point of this project, making the system scalable has helped making different "moving parts" interchangeable, thereby making integration with other systems possible in the future.

New technologies have been researched, mainly RabbitMQ and RSA encryption. RabbitMQ is what makes the servers scalable through copying, and it also opens up doors for further integration of systems with and by ChatX. A lot of work has gone into the RSA encryption, and encryption in general, but because so much time went with designing and redesigning the system, the encryption was never implemented.

5.1 Future Work

There was a larger amount of features which did not get implemented due to the scale of the project, but it is possible to extend the system with these features in the future with minimal changes and additions to the current system.

One possible addition to the system would be another client, e.g. a web client. This has not been a high priority, as the current solution requires a socket connection which is difficult to implement in standard HTML. Furthermore, the integration of the used technologies proved to be more difficult and time consuming than anticipated. Lastly, SSL would need to be implemented to make the solution secure and encrypted.

Voice over IP(VoIP) and video chat was another technology that was considered to be implemented. A few changes could make this possible on the current clients, but for a web client either a plugin and/or a cloud service would be necessary.

A thought was also at the client side to be able to embed e.g. instance

media, such as videos and images directly into the chat and make it possible to view without opening external tools or programs.

Lastly an idea was to make a multi platform encryption system. Such a system could have simplified some of the work, as some systems does not communicate well with each other, but making such system fell out of scope of the course.

Bibliography

[1] A. Shamir R.L. Rivest and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems, April 4, 1977.

[2] www.di mgt.com.au. Using padding in encryption. http://www.di-mgt.com.au/cryptopad.html.

Apendix

Git Repository

The full source code for this project can be found on the following link:

https://github.com/istbarp/ChatX

Additional a git repository can be checked out with the folowing link:

https://istbarp@github.com/istbarp/ChatX.git