INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Distributed chat system

Design and development of a distributed, interactive, multi-user and real-time chat system

Object-Oriented Design / Object-Oriented Programming

4IR-SI-C

Design report

16/11/2020

v0.1.0

MOINE Robin

SMANIOTTO Nathan

Based on 2019 specifications from Thales Alenia Space

ThalesAlenia Space
a Thales / Leonardo company

TABLE OF CONTENTS

Warning: some diagrams may be too wide to be read correctly in this report. Therefore, every diagram is available in the '***/doc/diagrams***' folder.

1 - System features

During the life cycle of this system which we refer as "Agent", we can distinguish multiple phases :

- Agent deployment by the administrator or a technician
- Agent used by common users and agent upgrades by the administrators
- Agent removal by the administrator

These phases highlight three actors, each of them with specific abilities and needs as shown in the following use case diagram.

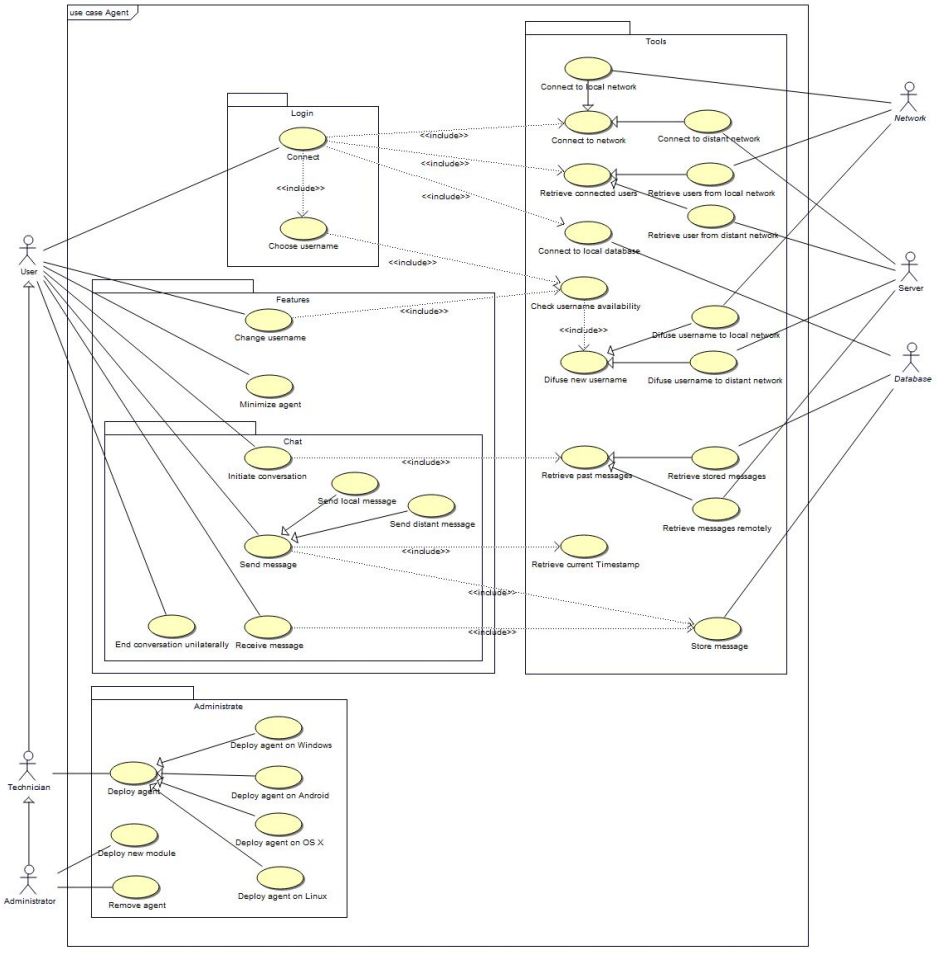

Figure 1 - Agent's use case diagram

This representation has allowed us to summarize every action performed by all the users, actions which will then be temporally described in a following non-exhaustive list of sequence diagrams.

2 - System structure

In order to represent our vision of how this system will operate, we first had to think about what will be in it. Therefore, using classes and interfaces along with their own attributes and methods, we managed to imagine the different components of the system from an object-oriented point of view in order to display them in a class diagram.

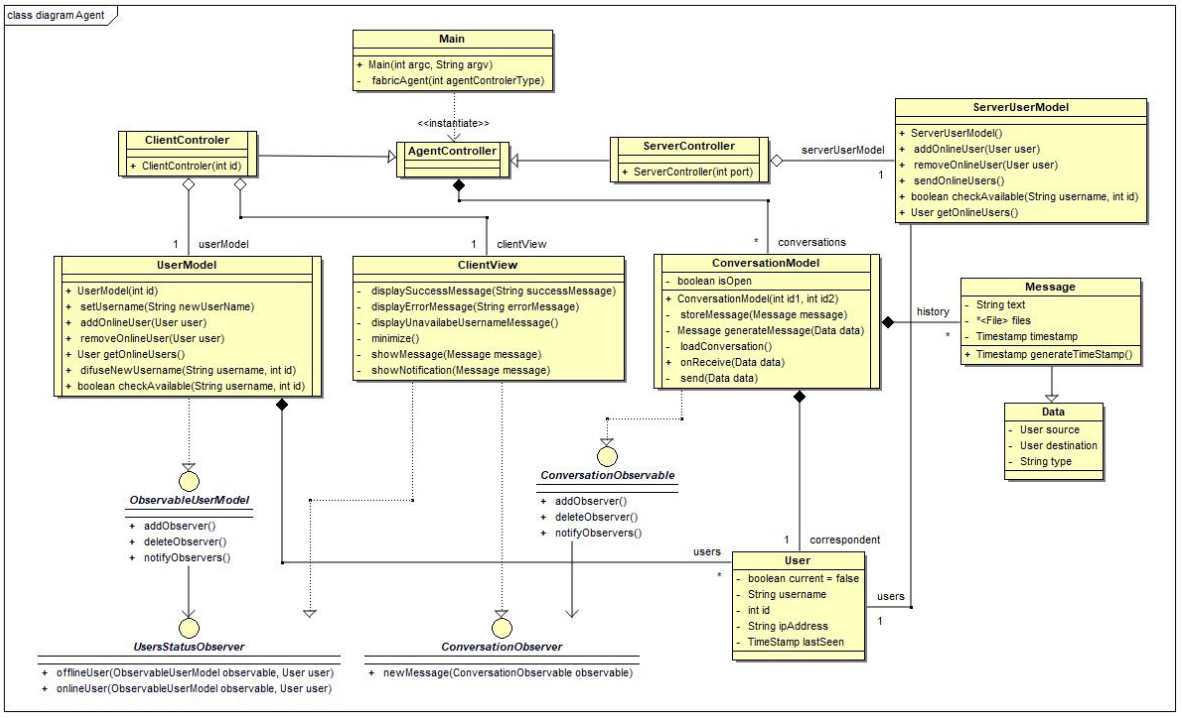

Figure 2 - Agent's class diagram

Our architecture is based on the Model-View-Controller design pattern. It allows us to manage the logins, the conversations and the graphical user interface concurrently. In order to communicate events modifications we decided to use the observer design pattern. In this diagram we represent only two groups of observers but in reality we will have more of them especially to react to user interface events.

3 - Objects interactions

3.1 - Connect

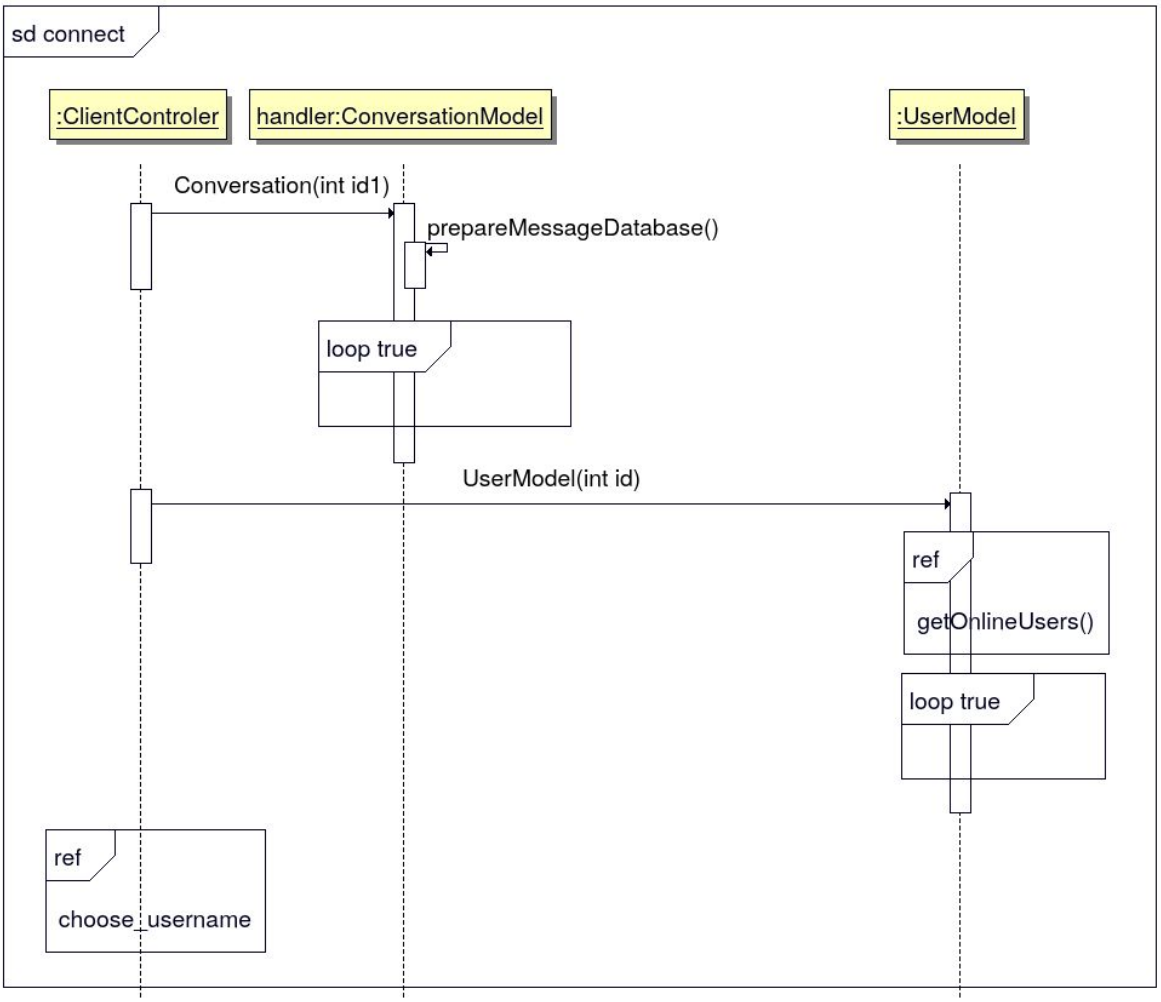

Figure 3 - Connect sequence diagram

In order to connect to users we chose to use an active class Conversation which will act as a server for conversation : as shown in the above diagram, we start by creating a ConversationModel which will be the handler. Whenever a new client wants to start a communication, it will send a message to the handler which will create the Conversation between the client and the distant user.
After that, the ClientController starts UserModel which will give access to online users. Thanks to that, the ClientController will ask the model to choose its username.

3.2 - Retrieve connected users

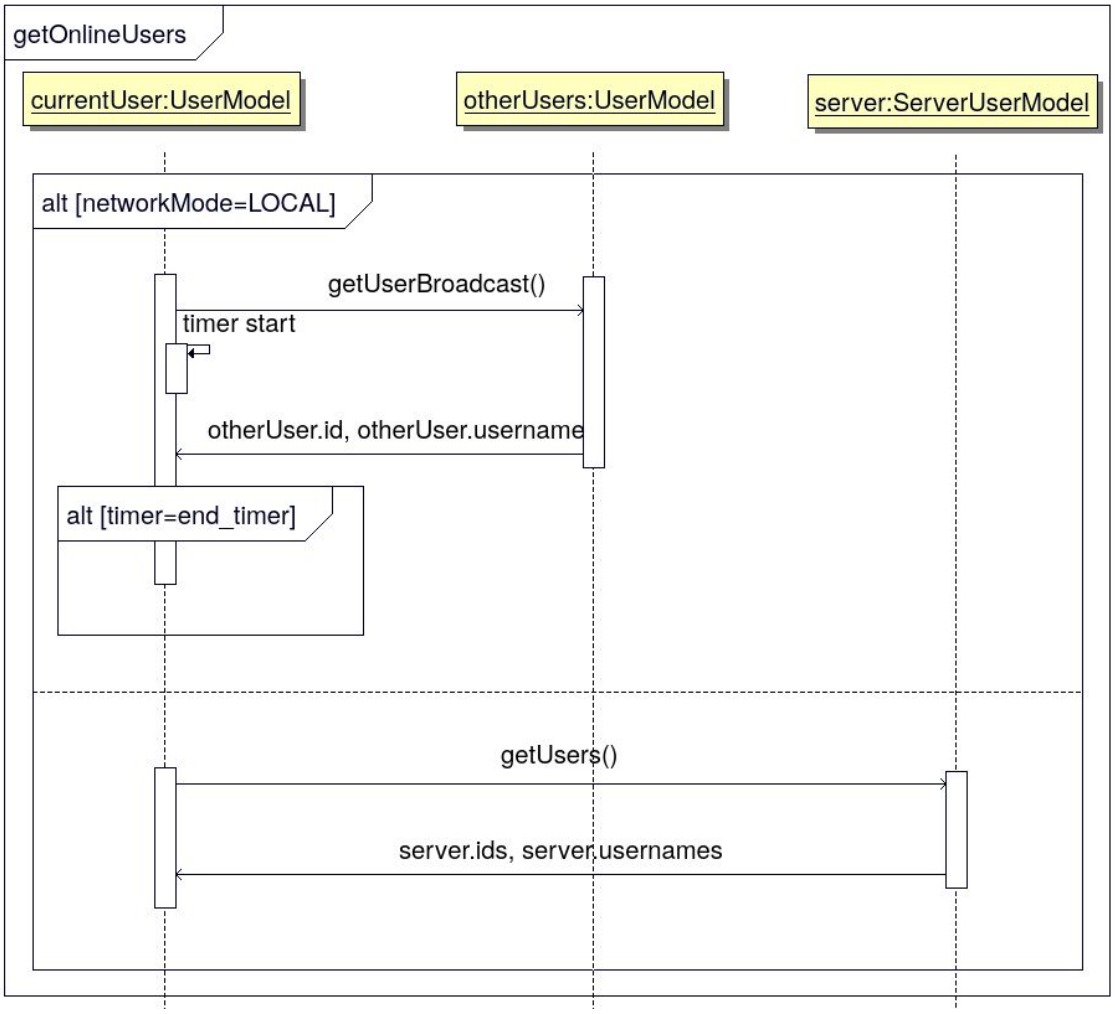

Figure 4 - Retrieve connected users sequence diagram

As previously, we split the getOnlineUsers' functionality according to the networkMode :

- in the LOCAL mode the UserModel is in charge of the logins and the username, id link (unique number for each client)
- in the centralized mode, the ServerUserModel gathers all ids and usernames

In this context we have to take into account the [CdC-Bs-25] requirements expressing a maximum delay of 5 seconds to show all users currently logged.

- In the case of the local mode, because we do not know how many users are correctly connected, we need to use a timer to guarantee the duration requirement of 5 seconds.
- With the centralised architecture, we consider that the network has enough capacity to allow us to have our response in less than 5 seconds.

In the future we might add timers on both modes to launch a popup if the answer does not come.

3.3 - Choose username

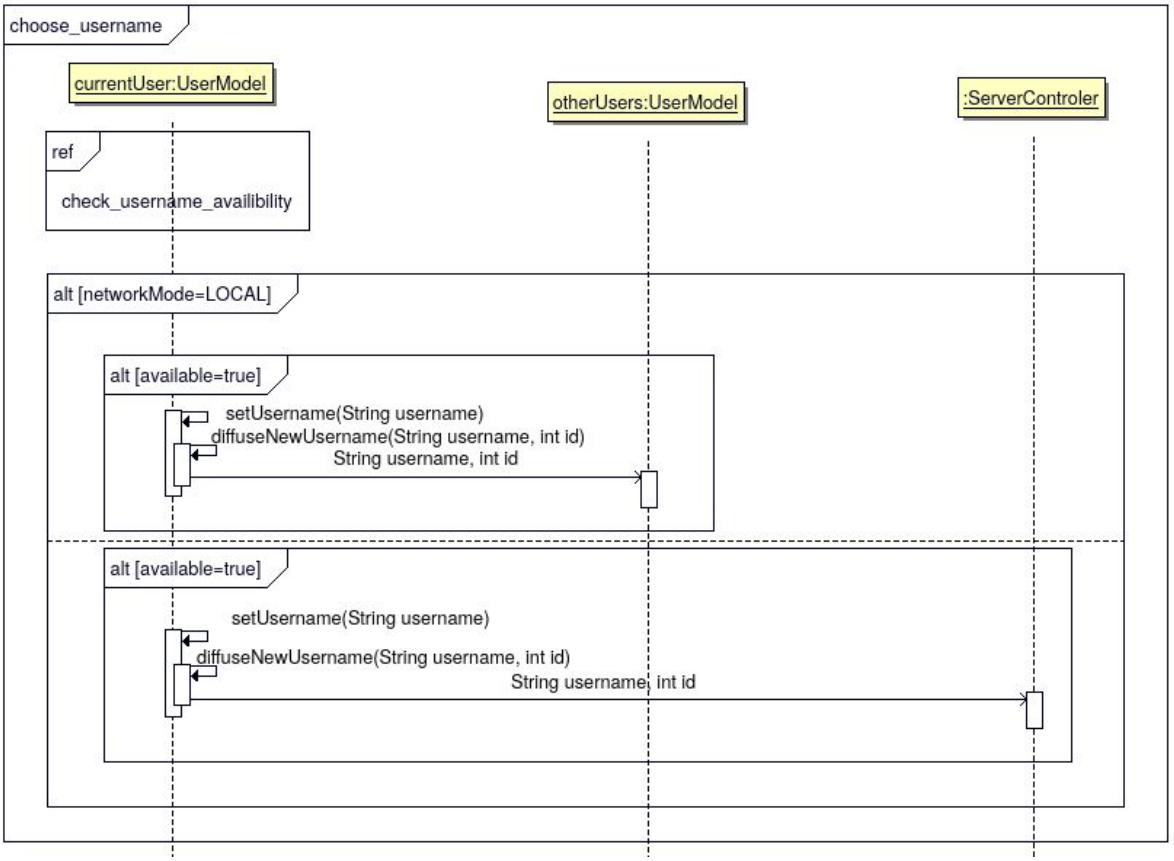

Figure 5 - Choose username sequence diagram

To choose our username, the specifications required that :

1. the username must be unique from all other connected logins [CdC-Bs-10]
2. all other users must be notified if a client changed his login [CdC-Bs-17]
3. this modification must not end the running clavardage sessions [CdC-Bs-18]
4. the username can be changed at all time by the user [CdC-Bs-16]

In our implementation we satisfied requirement 1 with check_username_availibility which tells us whether the username is already used.

Furthermore, we used the diffuseNewUsername to send the logins associated with the unique id to the logins manager (UserModel or ServerController linked to ServerUserModel)

We used the Observer design pattern to dynamically notify changes to the view and not interrupt any conversation.

As we will use this design pattern for the interface to notify models, we will be able to also satisfy the requirement 4.

3.4 - Check username availability

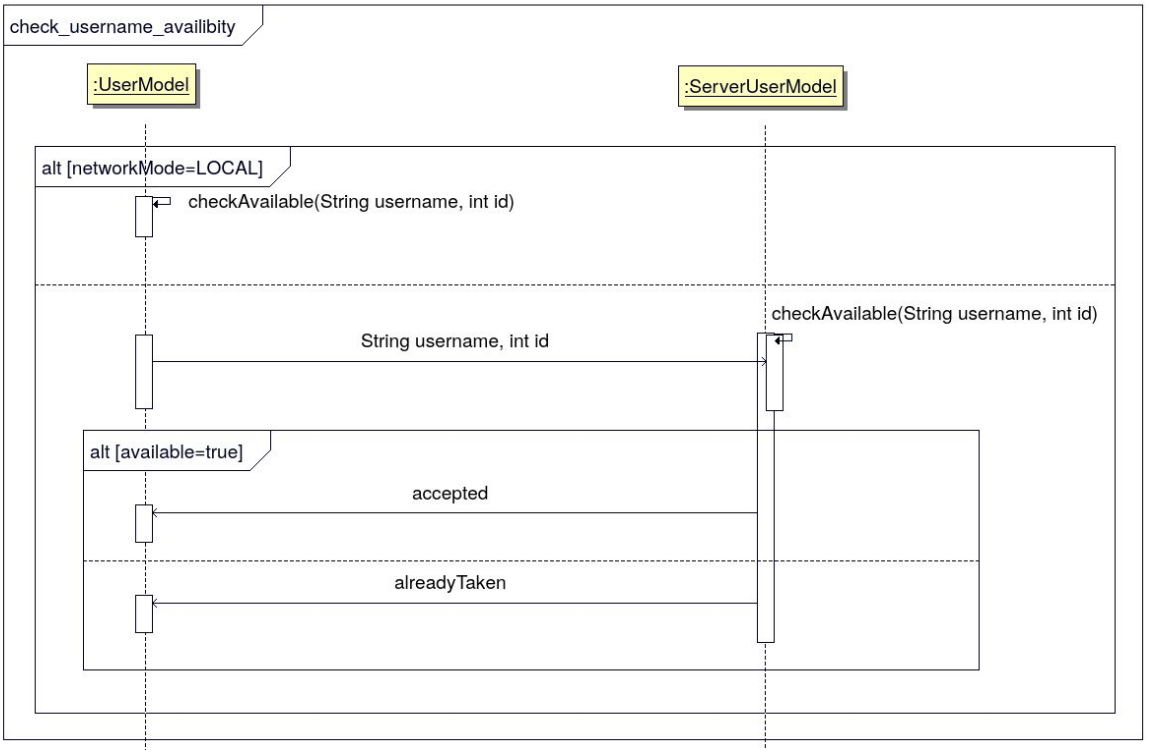

Figure 6 - Check username availability sequence diagram

The possibility if one person was using one login was required to attribute a new login or modified the current username of the client. This functionality is strongly linked with the requirements of choose_userame. We can note that in the first mode as usernames and ids are only stored in UserModel variables, it uses one of its methods to perform the operation. But in the other case we asynchronously ask the ServerUserModel to check if this combination is available.

3.5 - Initiate conversation

After login in, the user, thanks to a retrieved list of all connected agents on the network, is able to either wait for an incoming message or to simply initiate its own conversation with the desired correspondent.

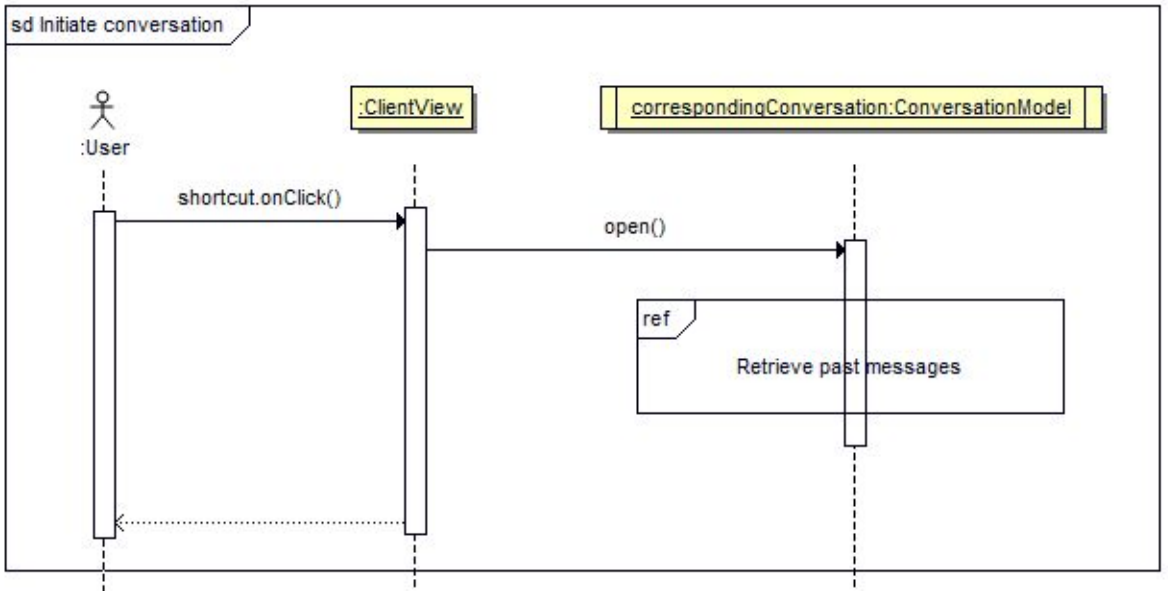

Figure 7 - Initiate conversation sequence diagram

By opening a conversation, previous messages will be retrieved in order to satisfy the [CdC-Bs-14] requirement which specifies that previously exchanged messages must be accessible and shown.

3.6 - Send a message

In order to communicate with its correspondent, the user must validate the sending of its message and attached files within the graphical interface.

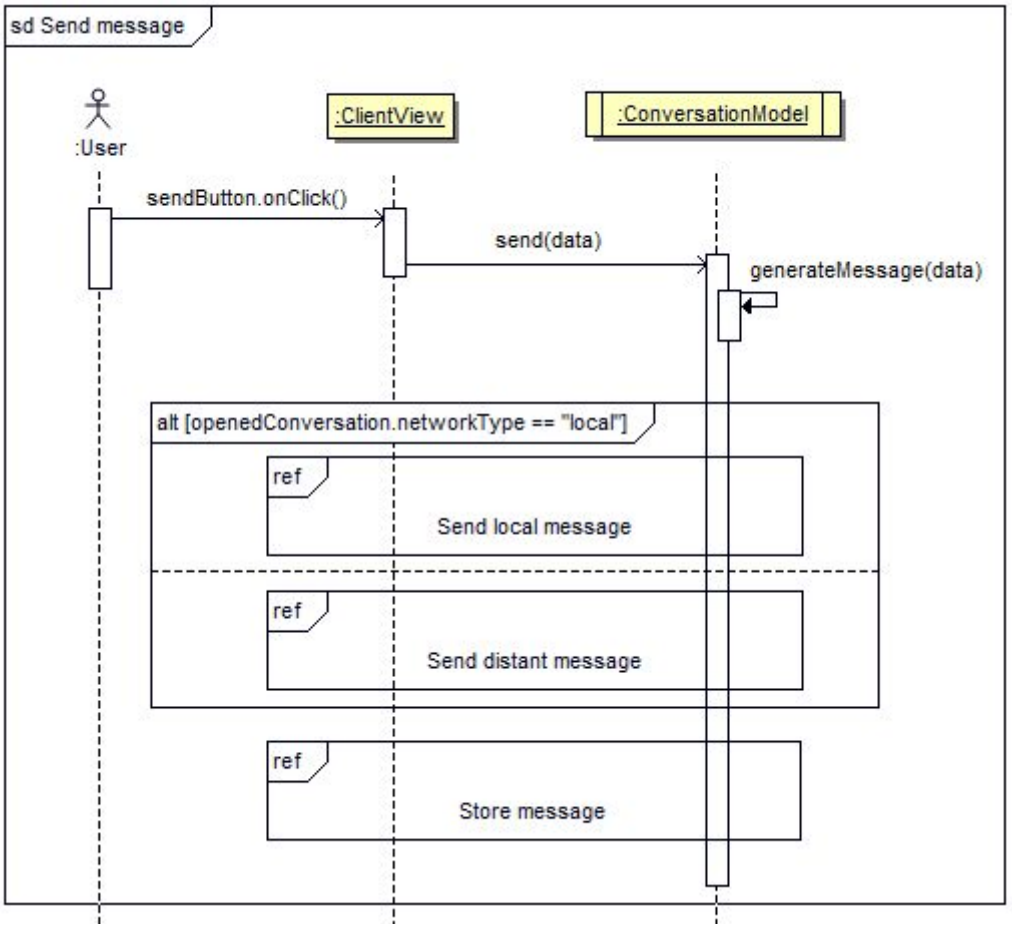

Figure 8 - Send message sequence diagram

This action will cause an event which will be received and treated in order to generate a proper message based on a specific structure. This message will then be sent, depending on where the correspondent is located (within the company's local network or in another one. Finally the message will be locally stored in order to be retrieved later.

3.7 - Receive a message

As a chat system implies, the user must be able to see incoming messages. To do so, the agent, thanks to active conversations will be able to receive messages and treat them according to its described behaviour.

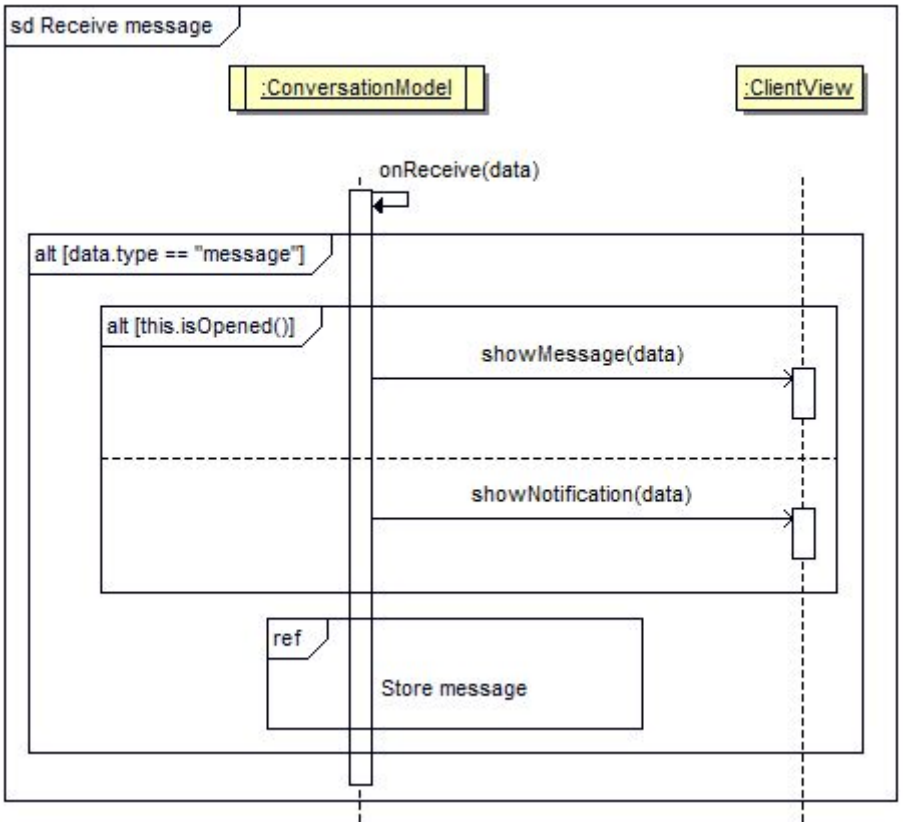

Figure 9 - Receive message sequence diagram

Indeed, the conversation will firstly filter messages in order to show them only if the conversation is opened. Otherwise, it means that the conversation is minimized but still running in the background. Therefore, a notification will be displayed to the user in order to alert him that he has received a new message.

3.8 - Change username

During its experience and as the [CdC-Bs-16] prerequisite requires, the user must be able to change its username without impacting the proper functioning of the system.

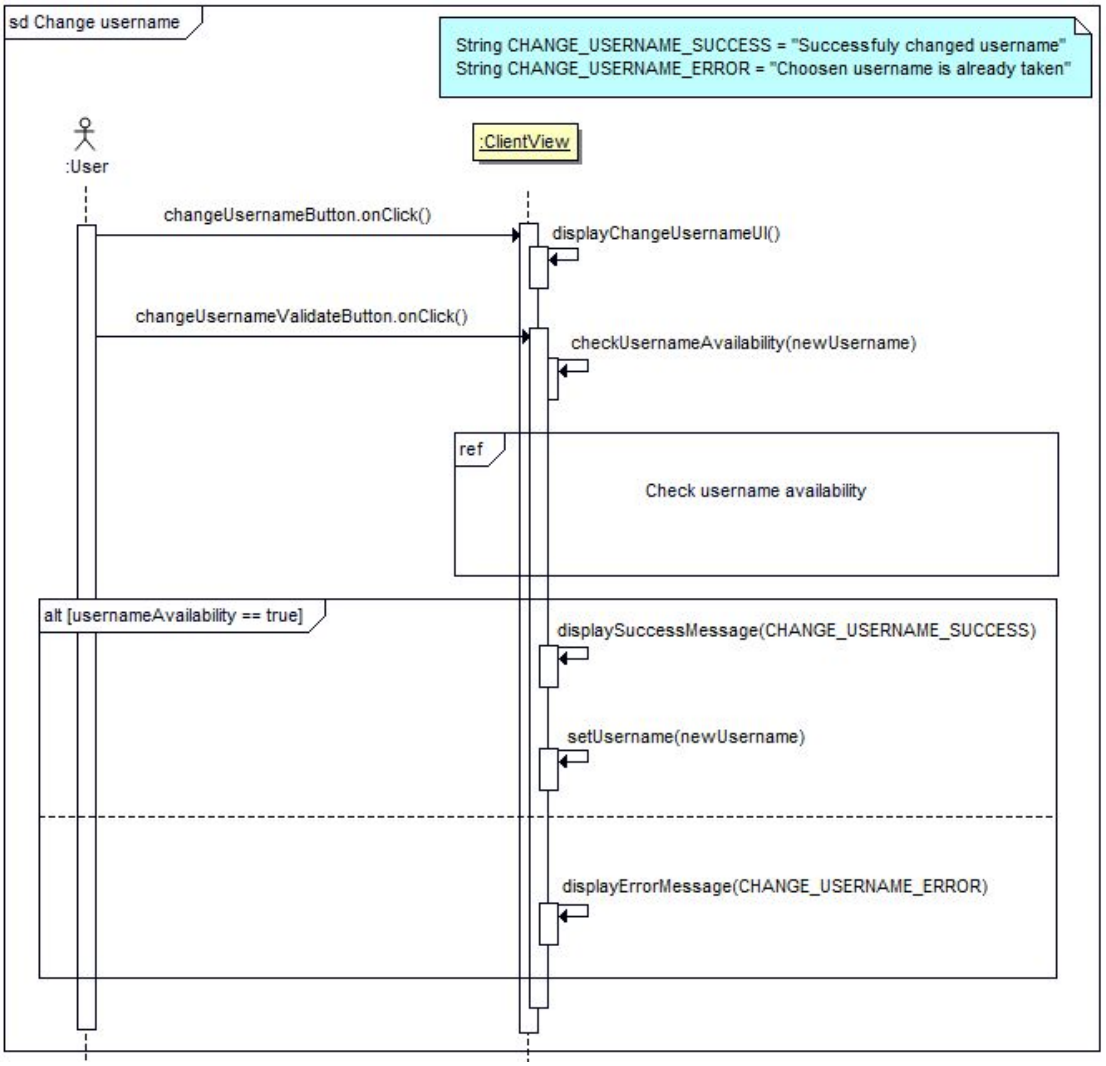

Figure 10 - Change username sequence diagram

Clicking on the right button of the graphical interface, the user will be asked to type its new desired username in a popup window. By validating its new username, the agent will first check if it is available and then update it locally if possible. Note that the spread of this new information is contained in the referenced “Check username availability” process.

3.9 - Iconify the agent

According to the [CdC-Bs-15] specificity, when minimized, the agent must be iconified within the task bar. Behaviour which will be performed inside the minimize method.

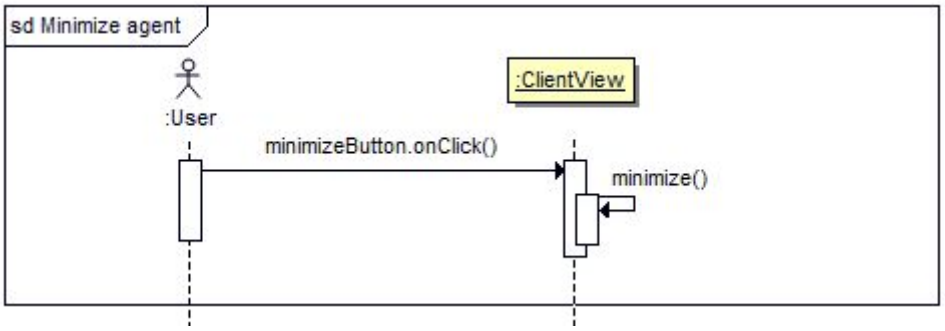

Figure 11 - Minimize agent sequence diagram

3.10 - Deploy the Agent

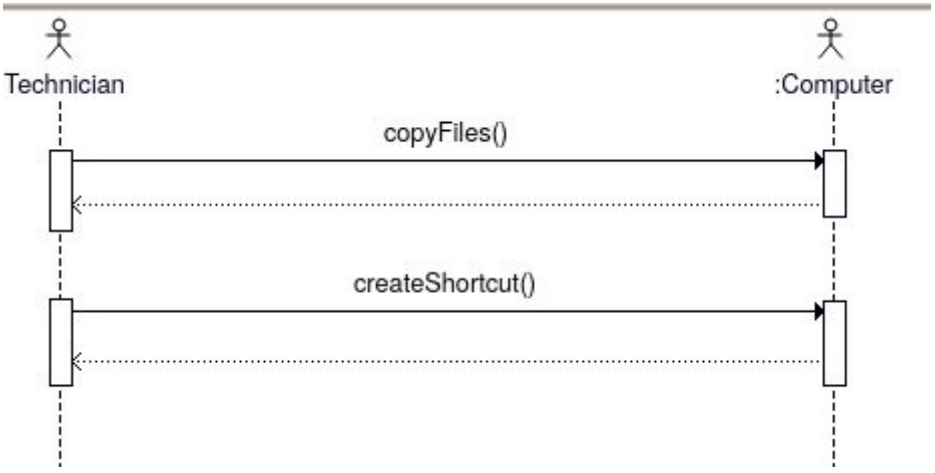

Figure 12 - Deploy agent sequence diagram

4 - State machines diagrams

4.1 - ClientController

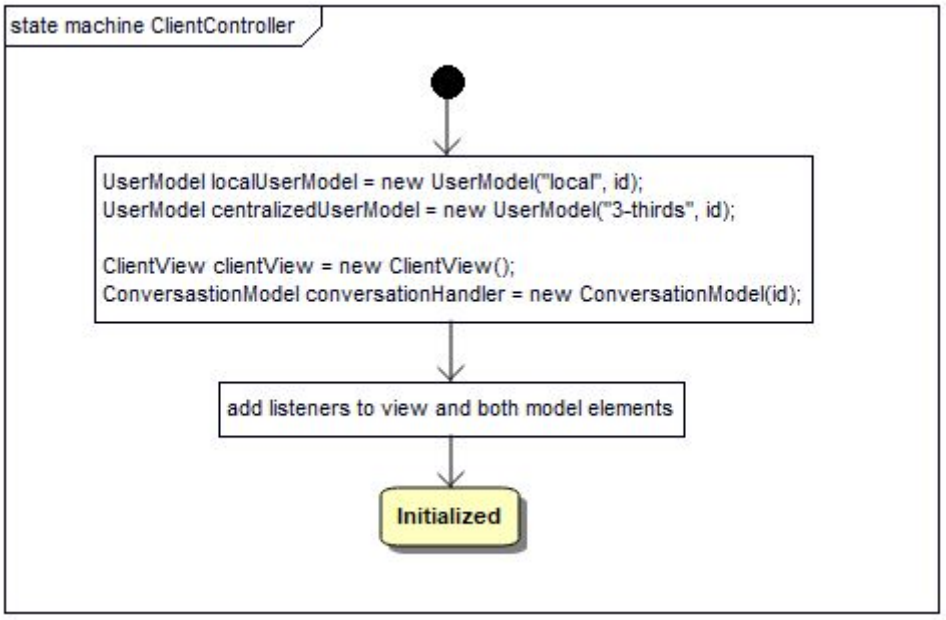

Figure 12 - ClientController state machine

The controller will make the link between all the elements of the system : he will especially set up the observers attached to all of the components in order to dynamically adapt the view to the models and inversely.

4.2 - UserModel

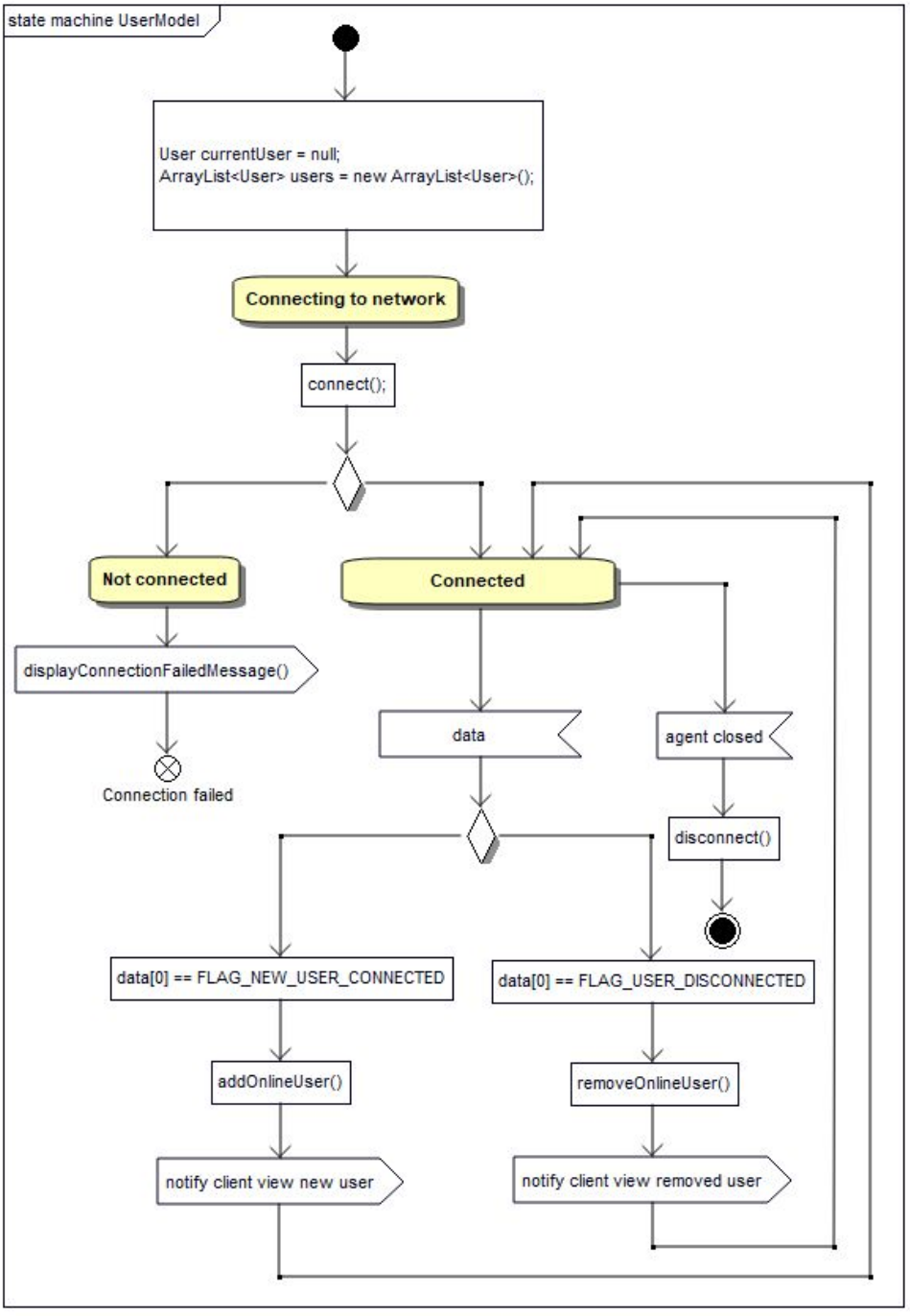

Figure 13 - UserModel state machine

This state diagram machine points out how external updates of the online users in the case of the local network mode are detected by each user. In the centralized mode the Server is in charge of this functionality and the local UserModel will only be notified when a login modification occurs.

4.3 - Conversation

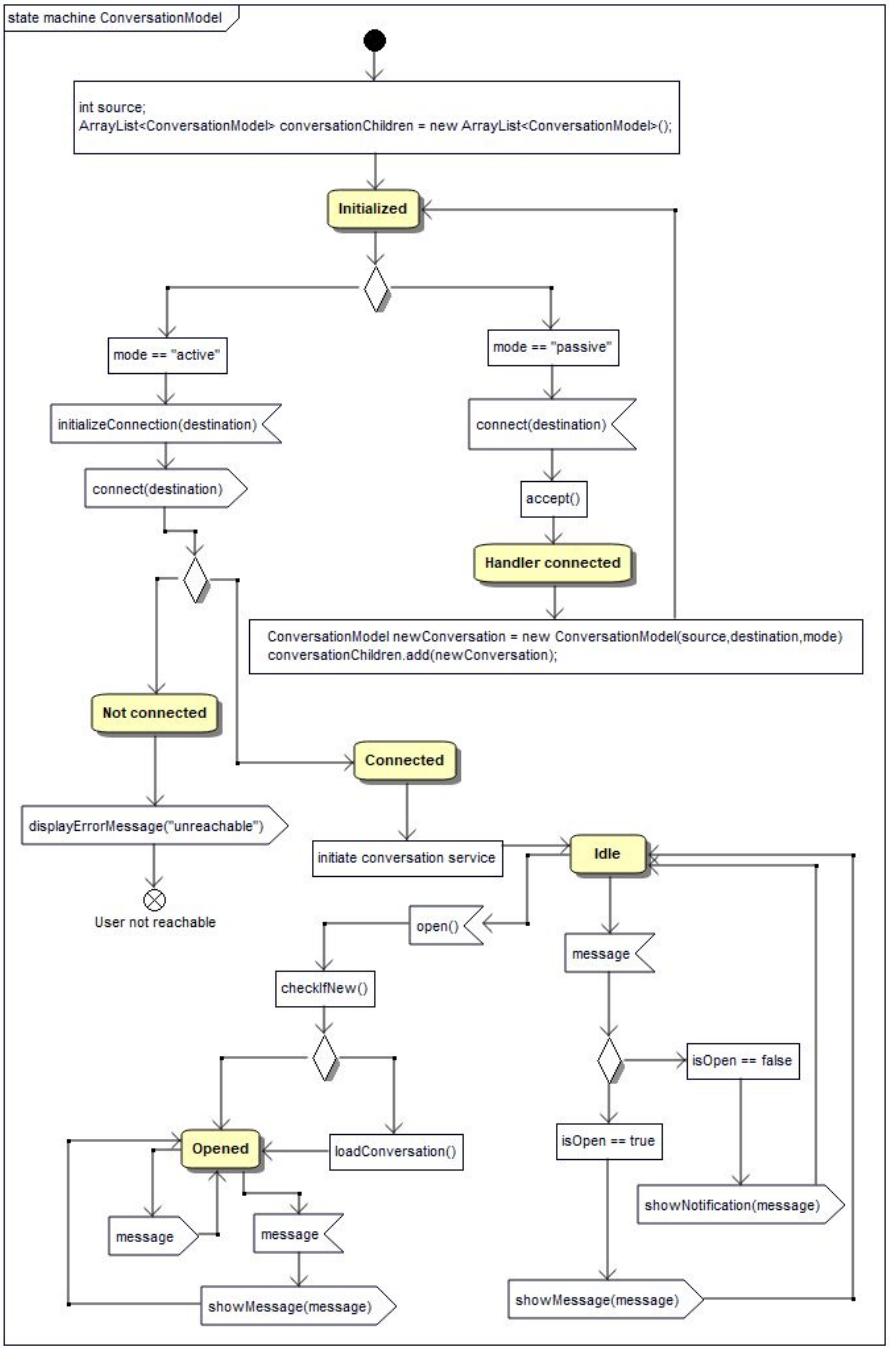

Figure 14 - ConversationModel state machine

Once a conversation is initialized within the conversationHandler previously instantiated, its mode attribute will predict its near future behaviour. Indeed, a passive will only accept the connection request and wait in its own thread since it is another user who has made the decision to initiate the conversation. However, an active conversation is the result of the current user wanting to initiate the conversation, by opening it first for example. Once connected to the correspondent a conversation is able to be opened in the user interface or

receive messages. If it receives messages it will display them only if the conversation is open. Otherwise it will display a notification. If opened, then the user can directly send messages.

4.4 - Server

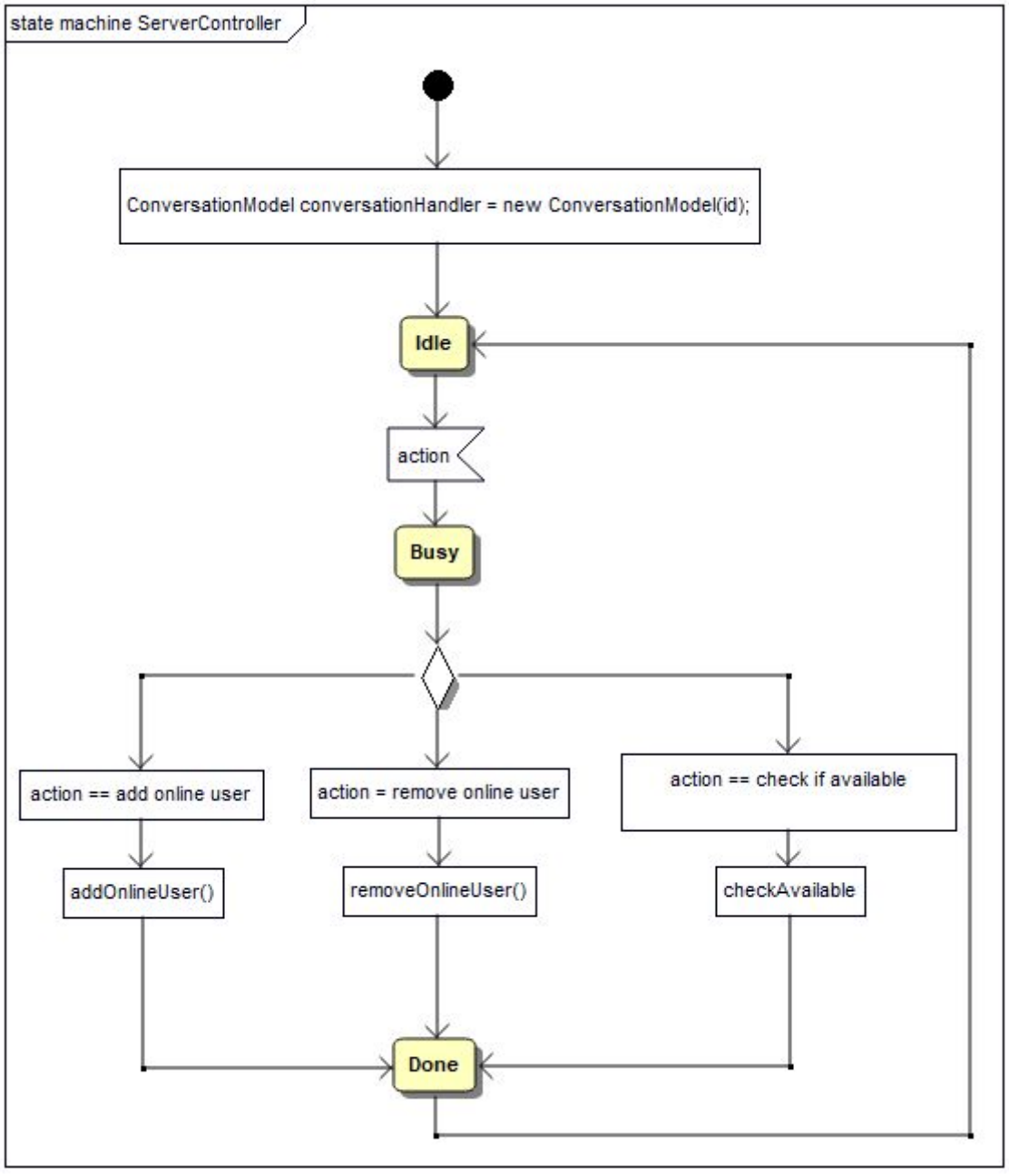

Figure 15 - ServerController state machine

As explained before, in the centralized mode the server will be in charge of maintaining logins. Moreoverit will provide services to the clients to check their logins.

ANNEX

(currently in French, will soon be translated to English)

5 - Exigences Fonctionnelles

5.1 Fonctionnalité relatives à l'agent

5.1.1 Fonctionnalités d'administration de l'agent

[CdC-Bs-1] Le système doit pouvoir être déployé sur un poste de travail fonctionnant sur le système d'exploitation Windows
[CdC-Bs-2] Le système doit pouvoir être déployé sur un poste de travail fonctionnant sur le système d'exploitation Linux
[CdC-Bs-3] Le système doit pouvoir être déployé sur un poste de travail fonctionnant sur le système d'exploitation OS X
[CdC-Bs-4] Le système doit pouvoir être déployé sur un poste de travail fonctionnant sur le système d'exploitation Android
[CdC-Bs-5] Le déploiement du système devra se limiter à la copie sur le poste de travail d'une série de fichiers et la création d'un raccourci pour l'utilisateur
[CdC-Bs-6] La taille globale de l'ensemble des ressources composant le système ne devra pas excéder 50 Méga-Octets sous une forme non compressée quel que soit le système d'exploitation sur lequel il est déployé

5.1.2 Fonctionnalités d'utilisation de l'agent

[CdC-Bs-7] Le système doit permettre à l'utilisateur de choisir un pseudonyme avec lequel il sera reconnu dans ses interactions avec le système
[CdC-Bs-8] Le système doit permettre à l'utilisateur d'identifier simplement l'ensemble des utilisateurs pour lesquels l'agent est actif sur le réseau
[CdC-Bs-9] Le système doit permettre à l'utilisateur de démarrer une session de clavardage avec un utilisateur du système qu'il choisira dans la liste des utilisateurs pour lesquels l'agent est actif sur le réseau
[CdC-Bs-10] Le système doit garantir l'unicité du pseudonyme des utilisateurs pour lesquels l'agent est actif sur le réseau
[CdC-Bs-11] Tous les messages échangés au sein d'une session de clavardage seront horodatés
[CdC-Bs-12] L'horodatage de chacun des messages reçus par un utilisateur sera accessible à celui-ci de façon simple
[CdC-Bs-13] Un utilisateur peut mettre unilatéralement fin à une session de clavardage

[CdC-Bs-14] Lorsqu'un utilisateur démarre un nouvelle session de clavardage avec un utilisateur avec lequel il a préalablement échangé des données par l'intermédiaire du système, l'historique des messages s'affiche
[CdC-Bs-15] L'utilisateur peut réduire l'agent, dans ce cas, celui-ci se place discrètement dans la barre des tâches sous la forme d'une icône lorsque le système d'exploitation permet cette fonctionnalité
[CdC-Bs-16] Le système doit permettre à l'utilisateur de changer le pseudonyme qu'il utilise au sein du système de clavardage à tout moment
[CdC-Bs-17] Lorsqu'un utilisateur change de pseudonyme, l'ensemble des autres utilisateurs du système en sont informés
[CdC-Bs-18] Le changement de pseudonyme par un utilisateur ne doit pas entraîner la fin des sessions de clavardage en cours au moment du changement de pseudonyme

6 Autres Exigences

6.1 Exigences opérationnelles

6.1.1 Exigences de performances

[CdC-Bs-19] Le déploiement du système doit être réalisable en 2 heures à partir de la prise de décision de déploiement.
[CdC-Bs-20] Le changement de pseudonyme d'un utilisateur doit être visible de l'ensemble des autres utilisateurs dans un temps inférieur à 20 secondes
[CdC-Bs-21] Le temps écoulé entre l'envoi d'un message par un utilisateur et la réception par un autre utilisateur ne doit pas excéder 1 seconde
[CdC-Bs-22] Le système doit permettre la mise en place de 1000 sessions de clavardage simultanées au sein de celui-ci
[CdC-Bs-23] L'agent doit permettre la mise en place de 50 sessions de clavardage simultanées
[CdC-Bs-24] Lorsque la vérification de l'unicité du pseudonyme de l'utilisateur échoue, l'utilisateur doit en être informé dans une période ne dépassant pas 3 secondes
[CdC-Bs-25] Le temps d'apparition des utilisateurs au sein de la liste des utilisateurs pour lesquels l'agent est actif ne doit pas excéder 5 secondes
[CdC-Bs-26] Le système doit permettre un usage simultané par au minimum 100 000 utilisateurs

6.1.2 Exigences de ressources

[CdC-Bs-27] Le système doit avoir une empreinte mémoire inférieure à 100Mo

[CdC-Bs-28] Lors de son exécution, le système ne doit pas solliciter le processeur plus de 1% du temps lorsque la mesure est réalisée sur un intervalle de 5 secondes
[CdC-Bs-29] Le système doit présenter une réactivité normale pour une application de clavardage

6.1.3 Exigences de sûreté de fonctionnement

[CdC-Bs-30] Le système doit garantir une intégrité des messages supérieure à 99,999%
[CdC-Bs-31] Une utilisation normale du système ne doit pas avoir d'impact sur le reste du système

6.1.4 Exigences particulières

[CdC-Bs-32] Le système doit permettre une extension simple des fonctionnalités par l'utilisation d'un système de modules qui fera l'objet d'une standardisation

6.2 Exigences de développement

6.2.1 Management et gestion du projet

[CdC-Bs-33] Les normes suivantes sont applicables aux travaux relatifs au projet :

a. ISO 15288 (Ingénierie Système)
b. ISO 9001 (Gestion de la Qualité)
c. ISO 14001 (Environnement)
d. ISO 10007 (Gestion de la Configuration)
[CdC-Bs-34] Un plan d'Assurance Qualité devra être fourni par le fournisseur de solution
[CdC-Bs-35] L'ensemble de la documentation produite ainsi que les artefacts devront être gérés en configuration pendant le cycle de vie.
[CdC-Bs-36] Le fournisseur devra fournir la preuve des procédures définies dans son plan d'assurance qualité.
[CdC-Bs-37] Le fournisseur devra fournir un plan de développement du système.
[CdC-Bs-38] Le fournisseur devra faire état de ses vérifications et tests.
[CdC-Bs-39] Le cycle de développement doit contenir des jalons auxquels le client doit être impliqué afin de valider les activités effectuées.
[CdC-Bs-40] Le fournisseur devra proposer une analyse du soutien logistique lors de la réponse à l'appel d'offres.

6.2.2 Exigence d'ergonomie

[CdC-Bs-41] Le système doit répondre aux standards d'ergonomie de chacune des plateformes pour lesquelles il est conçu.

6.2.3 Exigence de qualification

[CdC-Bs-42] Une version future du système pourra faire l'objet d'une certification en vue de son utilisation dans le milieu aéronautique.