Carlo Abi Chahine
Sylvain Archenault
Yves Houpert
Martine Wang

RAPPORT DE CONCEPTION UML :

Bamboo Ch@t

Projet GM4
Juin 2006

Table des matières

1 Introduction **2**

2 Présentation du logiciel **3**
2.1 Précisions sur le protocole de communication . . . 3
2.2 Précisions sur les threads . . . 3

3 Étude de conception UML du logiciel **4**
3.1 Schéma entrée-sortie . . . 4
3.2 Business model . . . 5
3.3 Les cas d'utilisations . . . 6
3.4 Diagrammes d'activité . . . 7
3.4.1 Inscription et connexion à BambooCh@t . . . 7
3.4.2 Ajout d'un contact . . . 8
3.4.3 Suppression d'un contact . . . 9
3.4.4 Envoi d'un message . . . 10
3.5 Diagrammes de séquences . . . 11
3.5.1 Connexion (Partie Cliente) . . . 11
3.5.2 Connexion (Partie Serveur) . . . 12
3.5.3 Envoi d'un message (Partie Cliente) . . . 13
3.5.4 Envoi d'un message (Partie Serveur) . . . 14
3.5.5 Réception d'un message . . . 15
3.5.6 Ajouter un contact (Partie Cliente) . . . 15
3.5.7 Ajouter un contact (Partie Serveur) . . . 16
3.6 Diagrammes de classes . . . 17
3.6.1 Les diagrammes de classes . . . 17
3.6.2 Les paquetages . . . 22
3.7 Diagrammes de collaborations . . . 23
3.7.1 Envoi d'un message . . . 23
3.7.2 Réception d'un message (Partie Serveur) . . . 24
3.7.3 Réception d'un message (Partie Cliente) . . . 25
3.8 Diagrammes d'états-transitions . . . 26
3.8.1 Émission du message . . . 26
3.8.2 Serveur/Client . . . 27
3.8.3 Réception du message . . . 27

4 Conclusion **28**

Chapitre 1

Introduction

Ce rapport illustre l'étude de conception UML de l'un de nos projets réalisés au cours de cette année. Dans un premier temps, nous présenterons ce projet et ses fonctionnalités pour bien définir la problématique. Ensuite, nous exposerons les différents travaux UML que nous avons effectués et qui permettront de bien structurer la réalisation du projet. Ces diagrammes seront accompagnés d'explications pour une meilleure compréhension. Enfin, nous conclurons ce rapport en détaillant les bénéfices que l'ont peut tirer d'une telle étude et les raisons qui font que UML est un langage très souvent utilisé par les développeurs lors de la conception de logiciels complexes.

Chapitre 2

Présentation du logiciel

Dans le cadre de l'option Java, nous avons souhaité développer un système de messagerie instantanée cryptée . Voici une présentation générale du projet BambooCh@t sur lequel nous avons réalisé notre étude de conception UML.

Le logiciel est composé de 2 applications distinctes : l'application Client et l'application Serveur.

L'application Client consistera en une petite fenêtre dans laquelle l'utilisateur pourra communiquer avec les autres utilisateurs. De plus, il aura à sa disposition la liste des personnes connectées. Il aura également la possibilité d'utiliser quelques options liées par exemple au cryptage des communications.

L'application Serveur aura, quant à elle, le rôle de connecter les utilisateurs entre eux, de gérer les connexions (par mot de passe). Elle permettra également de récupérer les clefs publiques des autres utilisateurs. De plus, elle implémentera un mini-site permettant l'administration, la gestion des utilisateurs, la recherche des clés publiques, l'inscription, la génération des clefs,...

Il y a une troisième application importante qui est incorporée au serveur, il s'agit d'un Entreprise Java Bean (EJB). Celui-ci est en charge de gérer la base de données.

2.1 Précisions sur le protocole de communication

Il est également important de détailler le mode de communication entre le client et le serveur afin de faciliter la compréhension des diagrammes, ceux-ci étant souvent allegés des détails des communications réseaux. Afin de communiquer, les deux applications utilisent des objets. On peut distinguer trois type d'objets principaux :

- Message : Message d'un client à un de ses contacts.
- Requêtes : Une demande d'un client pour ajouter un contact, changer son statut, ...
- Réponse : Réponse du serveur suite à une requête.

Pour transiter sur le réseau, ces objets ont besoin de subir une transformation. On les écrit en XML avant de les envoyer, puis on les récrée à la réception. Ce mécanisme a souvent été omis dans les diagrammes.

2.2 Précisions sur les threads

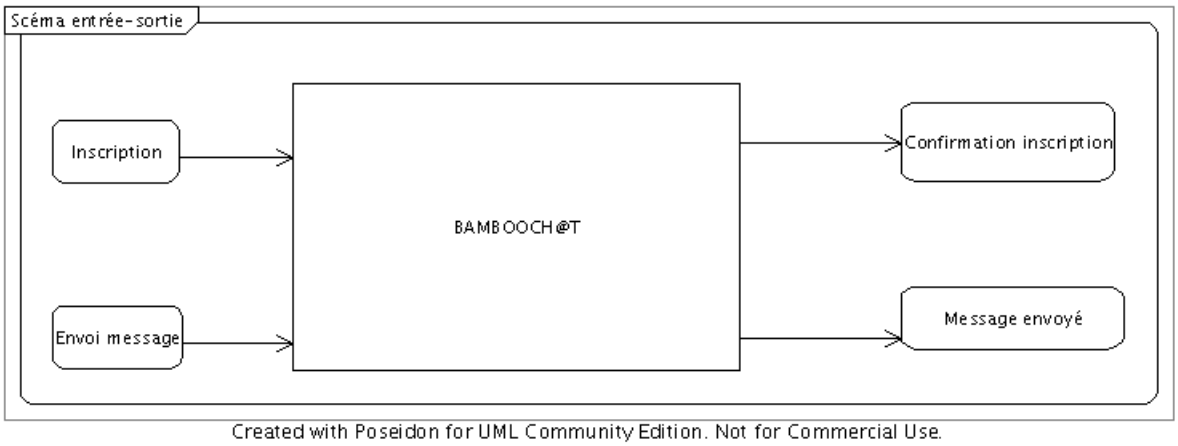
Afin d'avoir un serveur performant pouvant traiter plusieurs requêtes à la fois, nous avons utilisé des threads. Ceux-ci ne se sont pas faciles à représenter dans les scénarios. C'est pourquoi nous avons précisé à l'aide d'un commentaire lorsqu'une méthode est exécutée dans un thread.

Chapitre 3

Étude de conception UML du logiciel

3.1 Schéma entrée-sortie

Le schéma d'entrée-sortie est le diagramme le plus général du langage UML. Celui présente uniquement les fonctionnalités de l'application ainsi que le résultat que l'on obtiendra. BambooCh@t permettra donc à l'utilisateur de s'inscrire (et ultérieurement de se connecter) puis d'envoyer et de recevoir des messages.



3.2 Business model

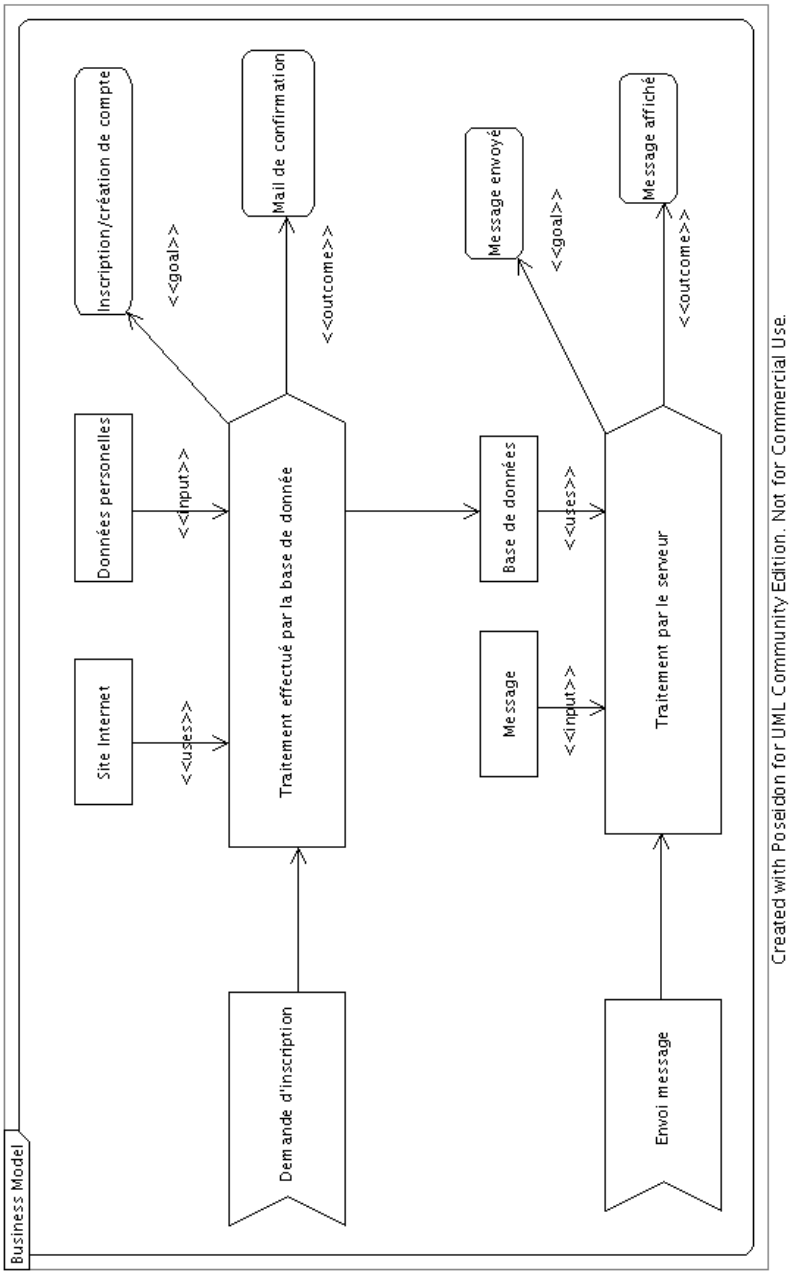


FIG. 3.1 – Business Model

3.3 Les cas d'utilisations

Les cas d'utilisations permettent de décrire le comportement d'un système du point de vue de l'utilisateur.

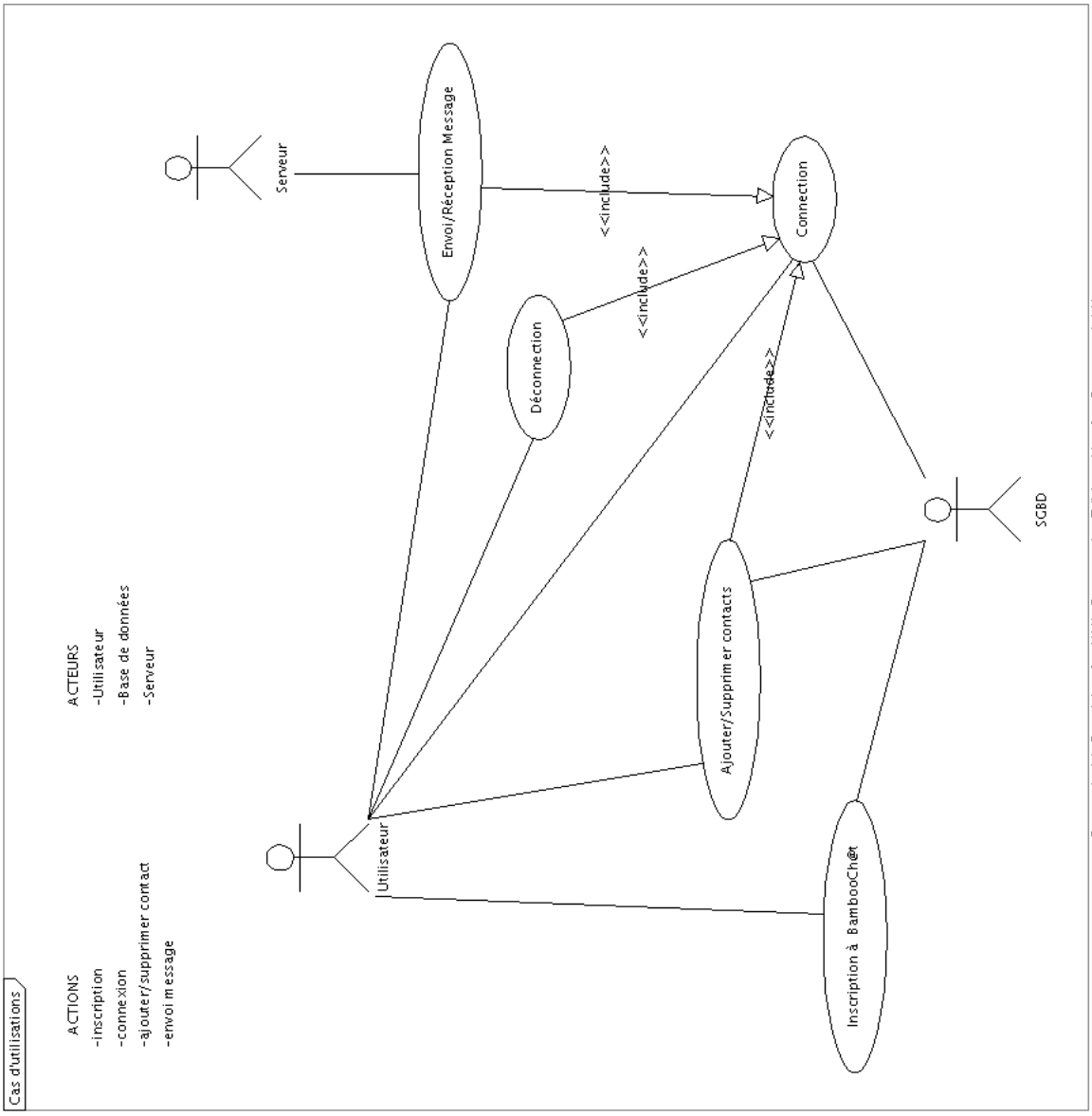


FIG. 3.2 – Les différents cas d'utilisation

3.4 Diagrammes d'activité

3.4.1 Inscription et connexion à BambooCh@t

La connexion à BambooCh@t est bien sûr différente si l'utilisateur utilise le logiciel pour la première fois ou s'il est déjà répertorié dans la base de données. Dans le premier cas, la personne devra suivre les différentes démarches pour s'inscrire. En revanche, si l'utilisateur possède déjà un compte, il aura accès directement à sa liste de contacts.

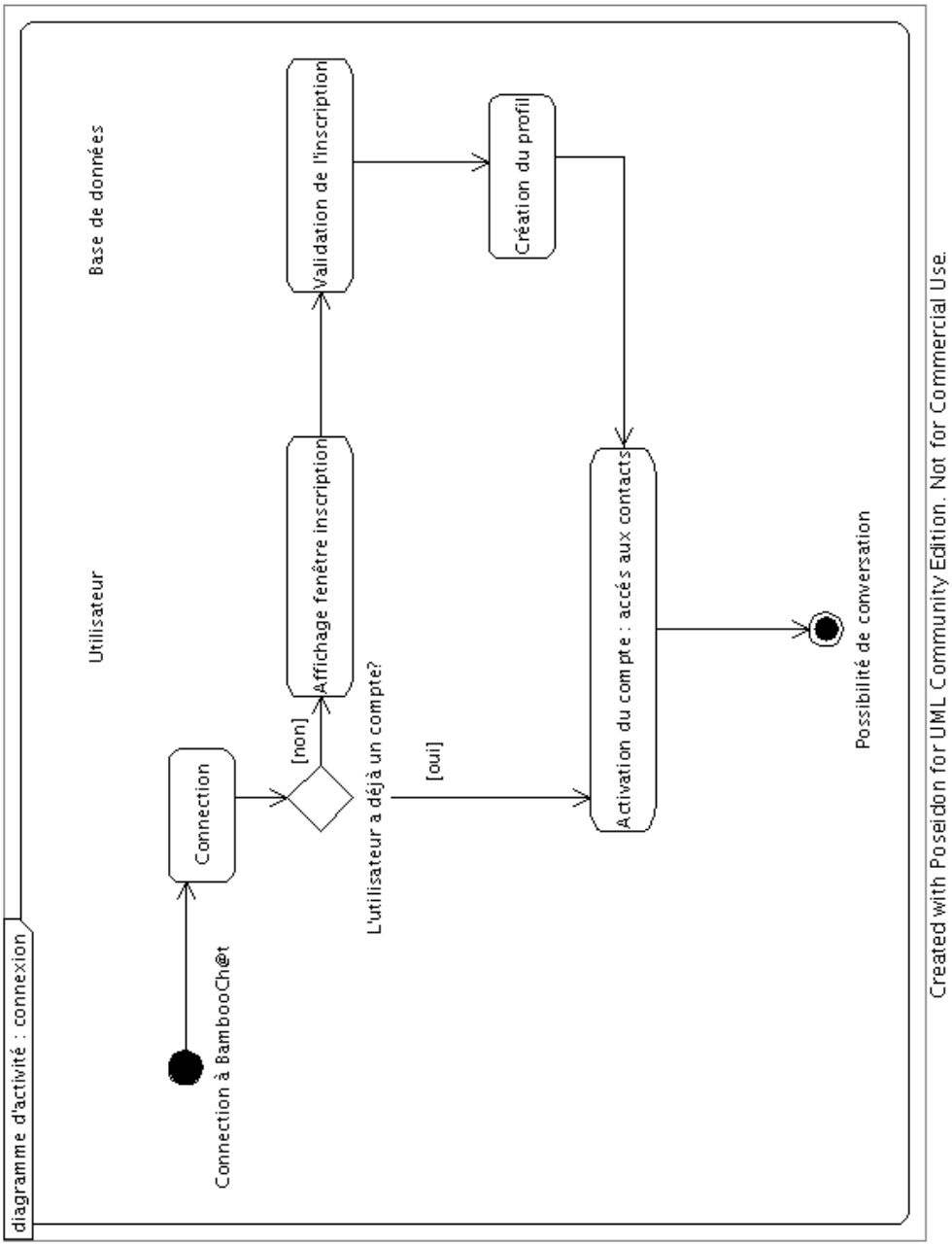


FIG. 3.3 – Connexion à BambooCh@t

3.4.2 Ajout d'un contact

L'application permet à l'utilisateur de gérer ses contacts. Lorsque celui-ci désire ajouter un contact dans sa liste, une fenêtre « Ajouter Contact » apparaît. L'utilisateur saisit les coordonnées du nouveau conctact pour que celui-ci soit enregistré dans la base de données. Ensuite, le contact ajouté apparaît sur la liste de contacts de l'utilisateur.

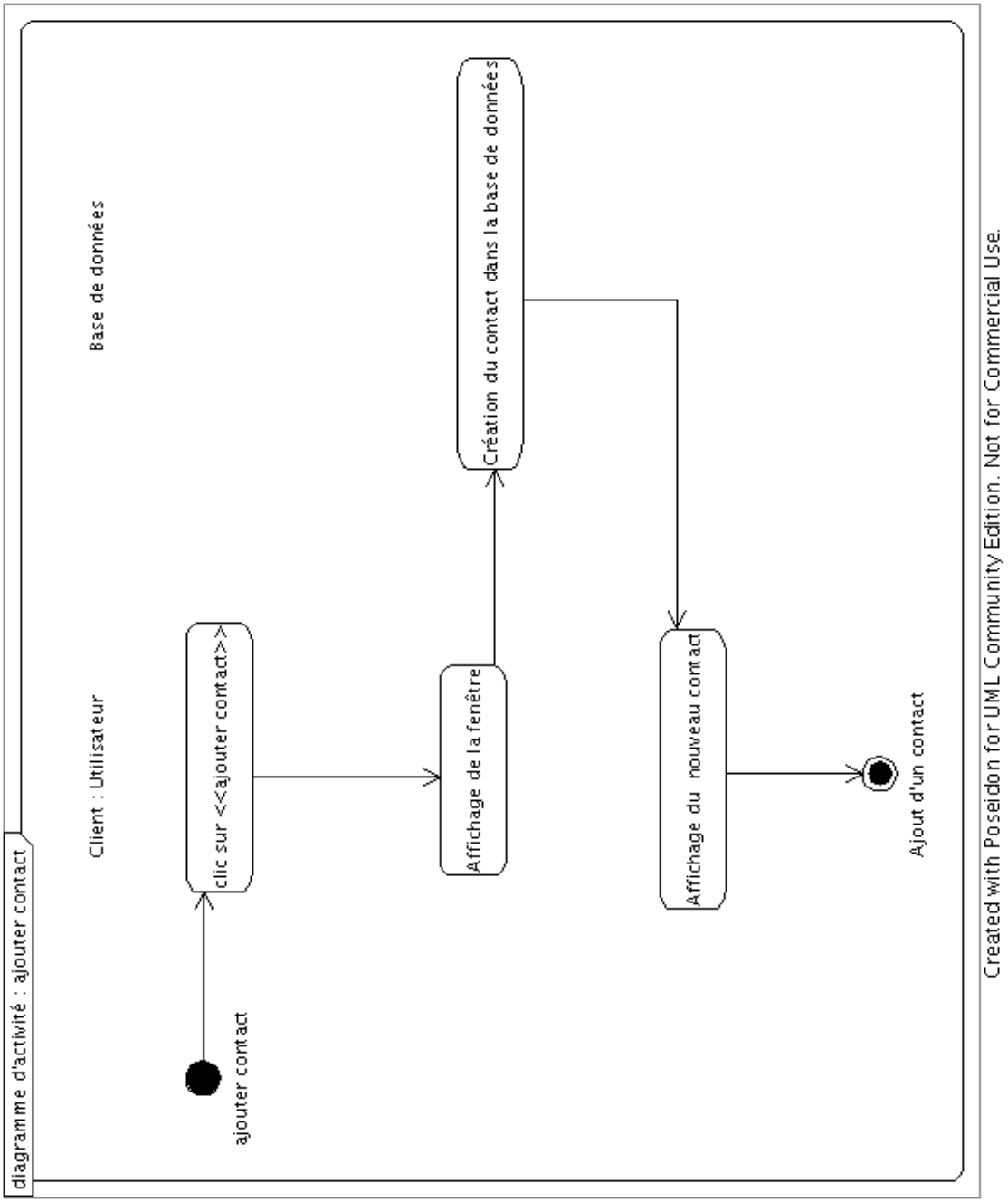


FIG. 3.4 – Ajouter un contact

3.4.3 Suppression d'un contact

La possibilité de supprimer un contact est la 2ème grande fonctionnalité qui permet à l'utilisateur de gérer sa liste. Celle-ci est très simple puisqu'il suffit de le sélectionner puis de cliquer sur « Supprimer contact ». Celui-ci sera également supprimé dans la base de données.

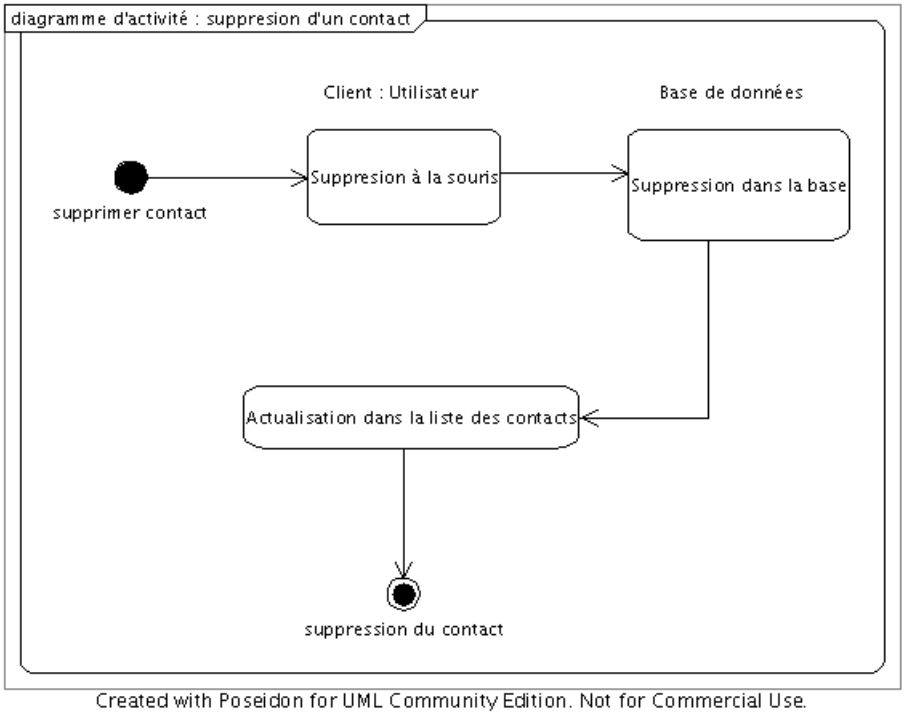


FIG. 3.5 – Supprimer un contact

3.4.4 Envoi d'un message

L'envoi de message est la fonctionnalité principale de BambooCh@t mais également la plus complexe. En effet, l'utilisateur peut tout d'abord choisir s'il désire ou non crypter le message. Dans un deuxième temps, lorsque le serveur aura récupéré le message, l'application devra savoir si la conversation est déjà entamée (dans ce cas, on affiche le message dans la fenêtre correspondante) ou non. Dans ce cas précis, on affectera un numéro de fenêtre à la future conversation. On ouvrira celle-ci et le message s'affichera à l'intérieur.

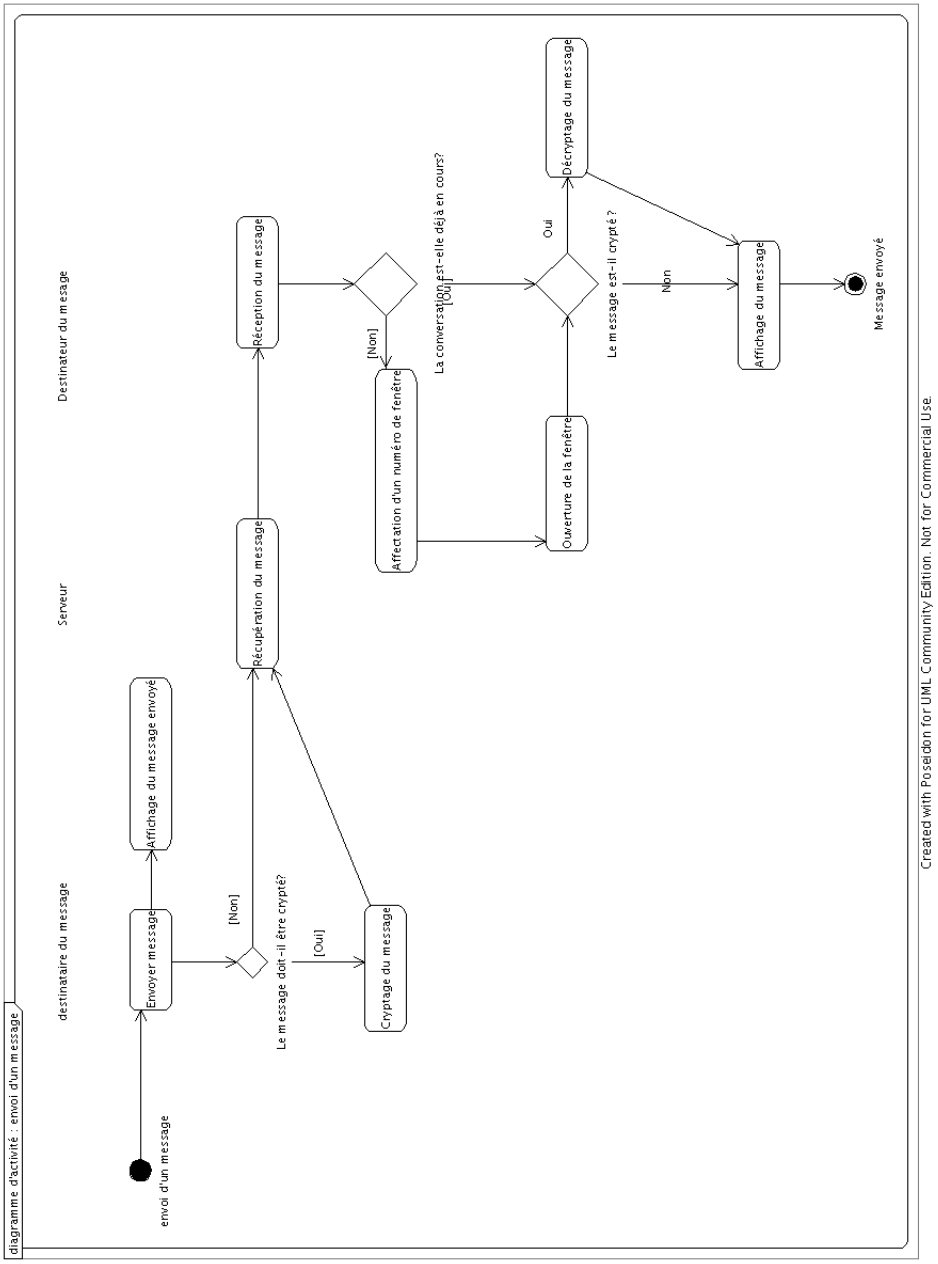


FIG. 3.6 – Envoyer un message

3.5 Diagrammes de séquences

Les diagrammes de séquences permettent de représenter la séquence des interactions entre plusieurs objets impliqués dans une fonctionnalité particulière du logiciel. Nous avons à chaque fois choisi de séparer la partie Client et la partie Serveur pour que les diagrammes soient plus clairs. Les scénarios sont donc en quelque sorte scindés en deux pour bien décomposer les deux applications.

3.5.1 Connexion (Partie Cliente)

Le processus de connexion contient trois phases. Tout d'abord, le socket doit se connecter au serveur, puis le client envoie ses identifiants et enfin, il doit récupérer la liste de ses contacts. L'utilisateur et le serveur jouent bien leur rôle d'acteur.

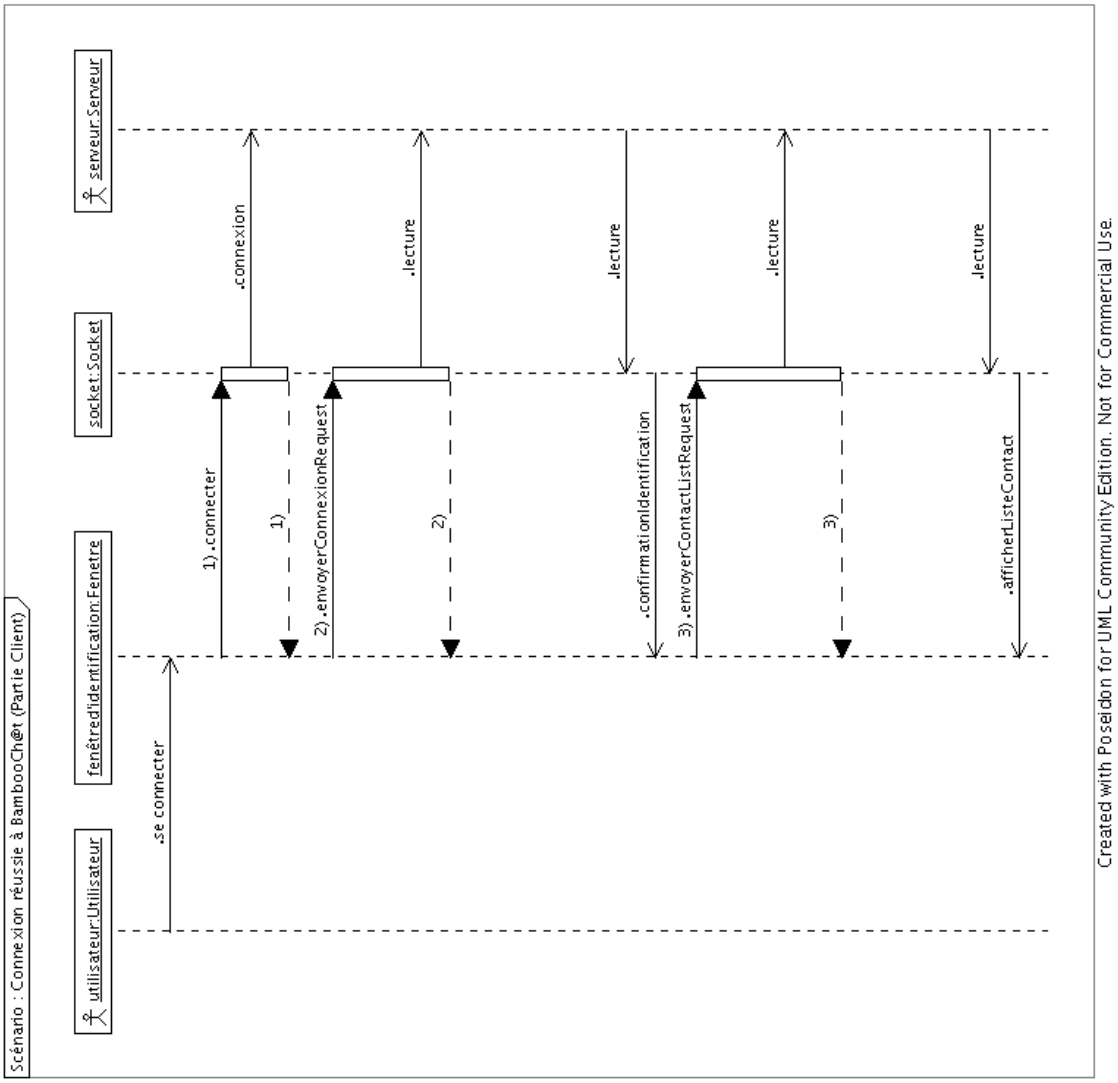


FIG. 3.7 – Connexion à BambooCh@t (Client)

3.5.2 Connexion (Partie Serveur)

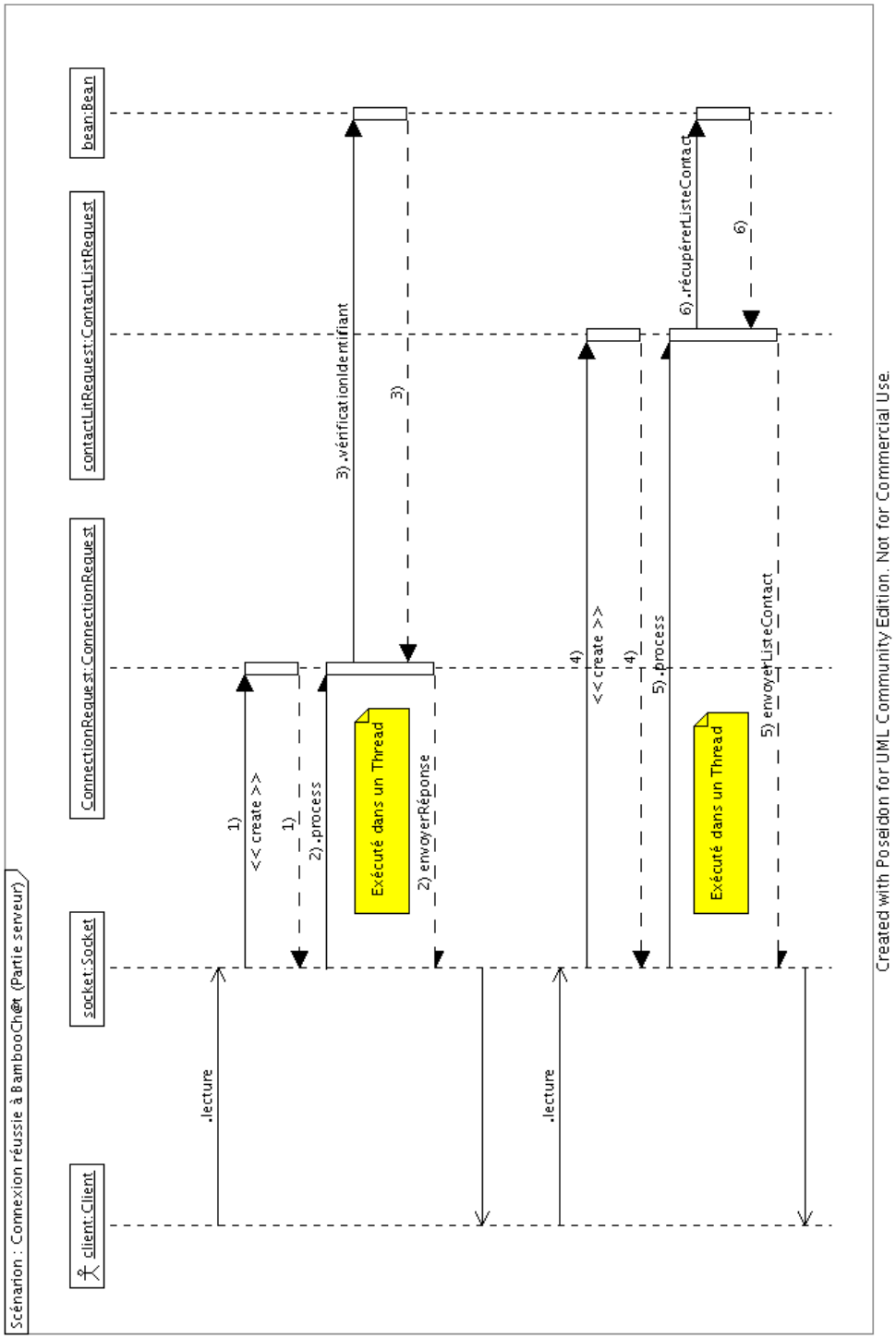


FIG. 3.8 – Connexion à BambooCh@t (Serveur)

3.5.3 Envoi d'un message (Partie Cliente)

L'envoi d'un message est la fonctionnalité de base de l'application. Elle est relativement simple. Ici nous voyons le chemin suivi par le message de l'expéditeur jusqu'au serveur. La réception sera traitée dans la section suivante.

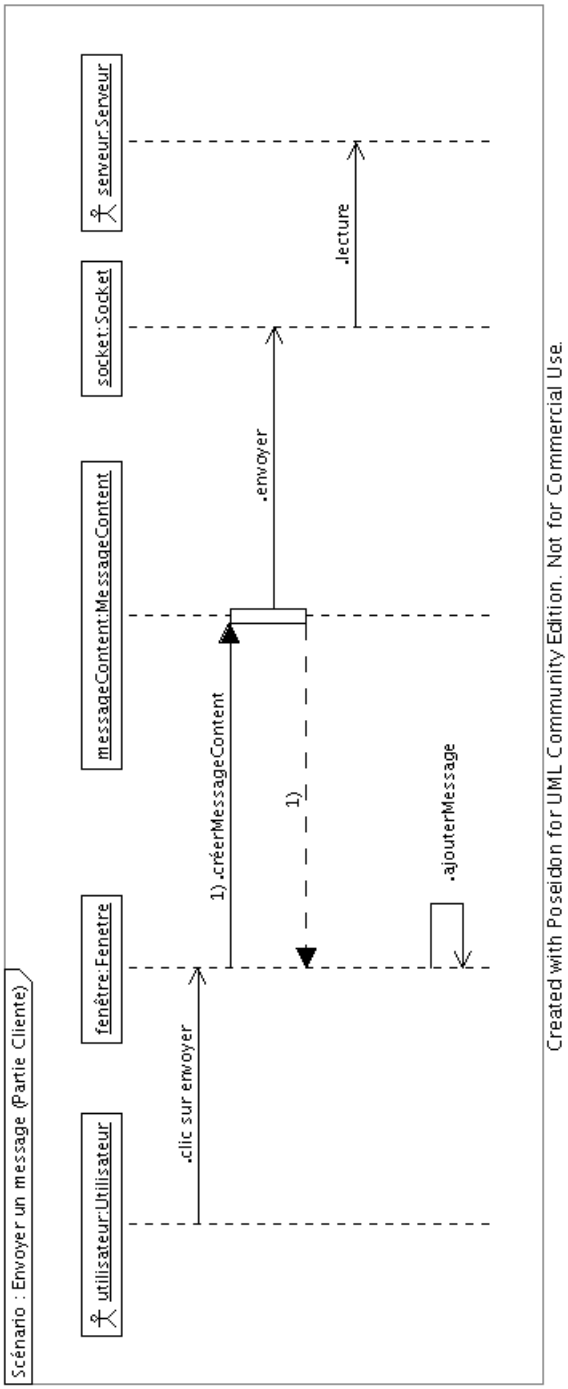


FIG. 3.9 – Envoyer un message (Client)

3.5.4 Envoi d'un message (Partie Serveur)

Lorsque le message arrive dans le serveur sous format XML, il faut le décoder pour le transformer en MessageContent. Ensuite, on cherche et on récupère le destinataire du message. Le message est alors renvoyé au serveur de nouveau sous format XML.

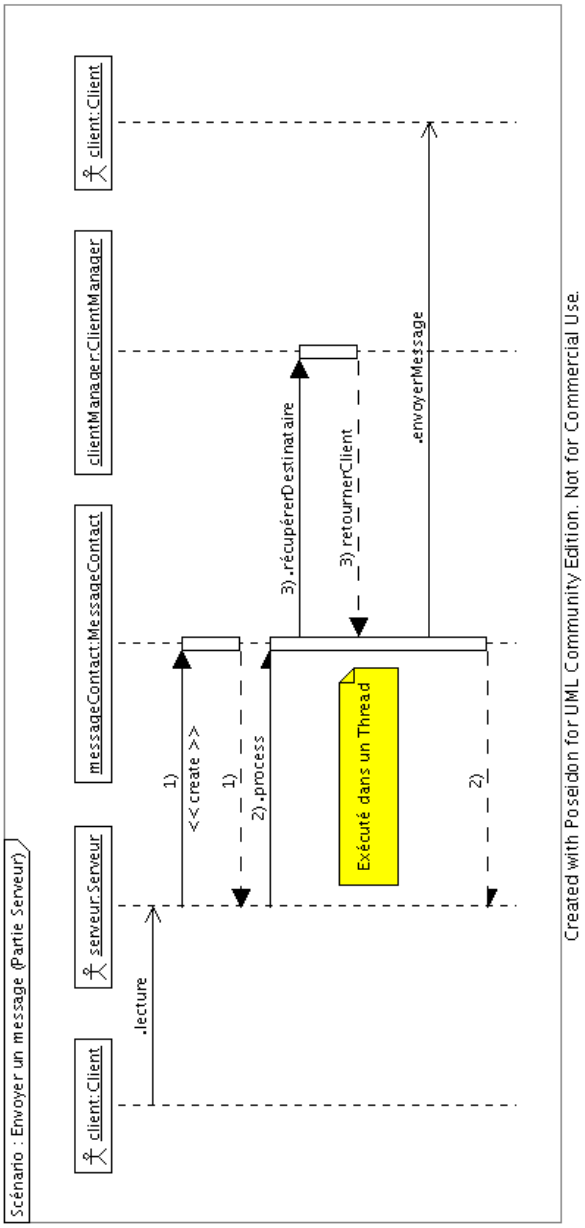


FIG. 3.10 – Envoyer un message (Serveur)

3.5.5 Réception d'un message

Lorsque l'on reçoit un message du serveur, on recherche la fenêtre correspondant à la connexion, ou on en crée une nouvelle si elle n'existe pas. Puis on affiche le message dans cette fenêtre.

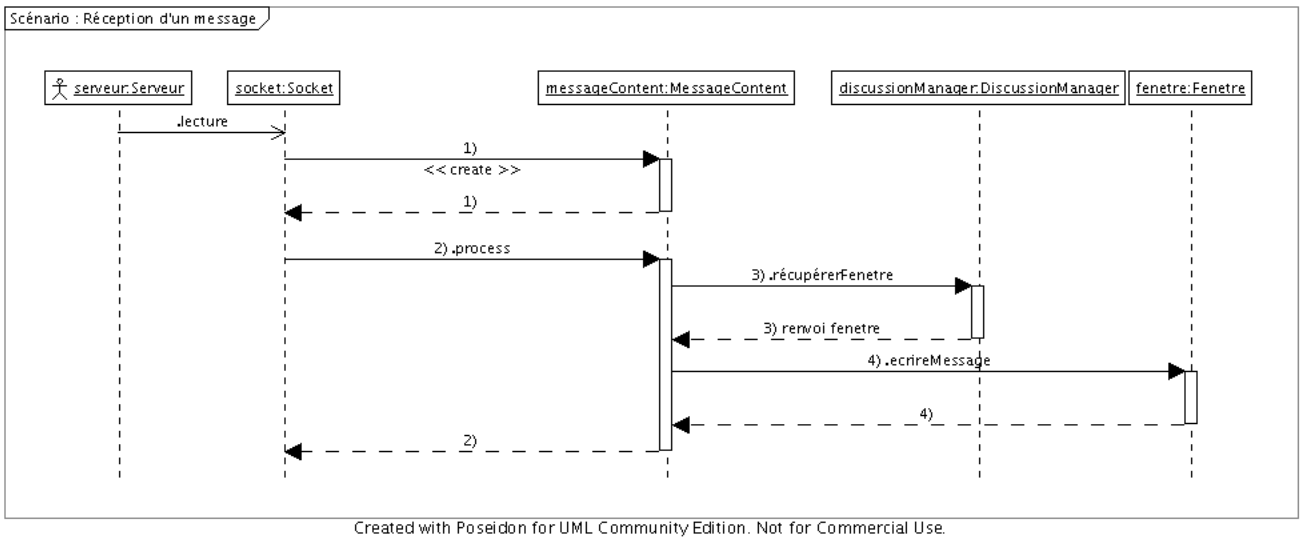


FIG. 3.11 – Réception d'un message

3.5.6 Ajouter un contact (Partie Cliente)

Ce scénario représente comment marche l'ajout d'un contact. L'ajout est effectué en deux étapes : la demande (ce que l'on voit ici) et la réponse (partie suivante). Dans cette partie, on se contente d'envoyer le nom du contact que l'on souhaite ajouter.

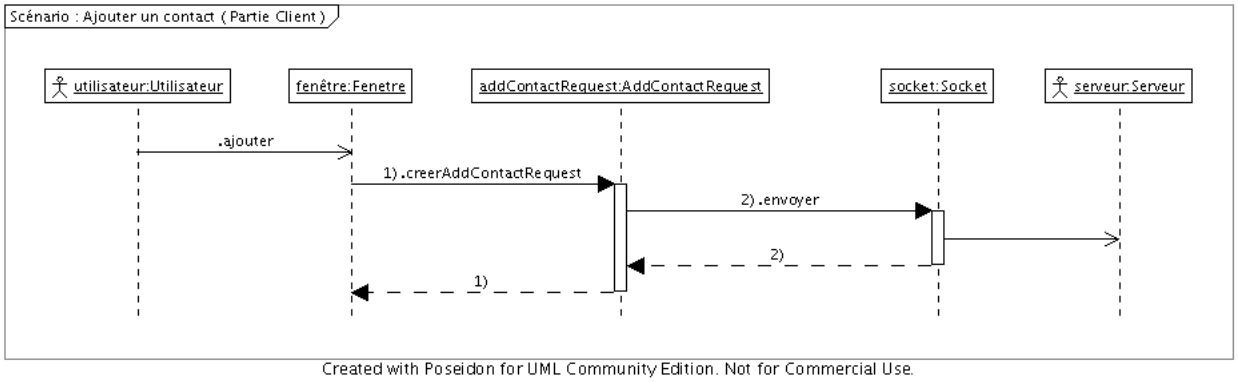


FIG. 3.12 – Ajout d'un contact (Client)

3.5.7 Ajouter un contact (Partie Serveur)

Ici nous voyons comment le serveur traite l'ajout d'un contact par un utilisateur. Une fois la demande reçue et analysée, on effectue les changements nécessaires dans la base de données par l'intermédiaire d'un EJB (Entreprise Java Bean), puis on renvoie une réponse au client.

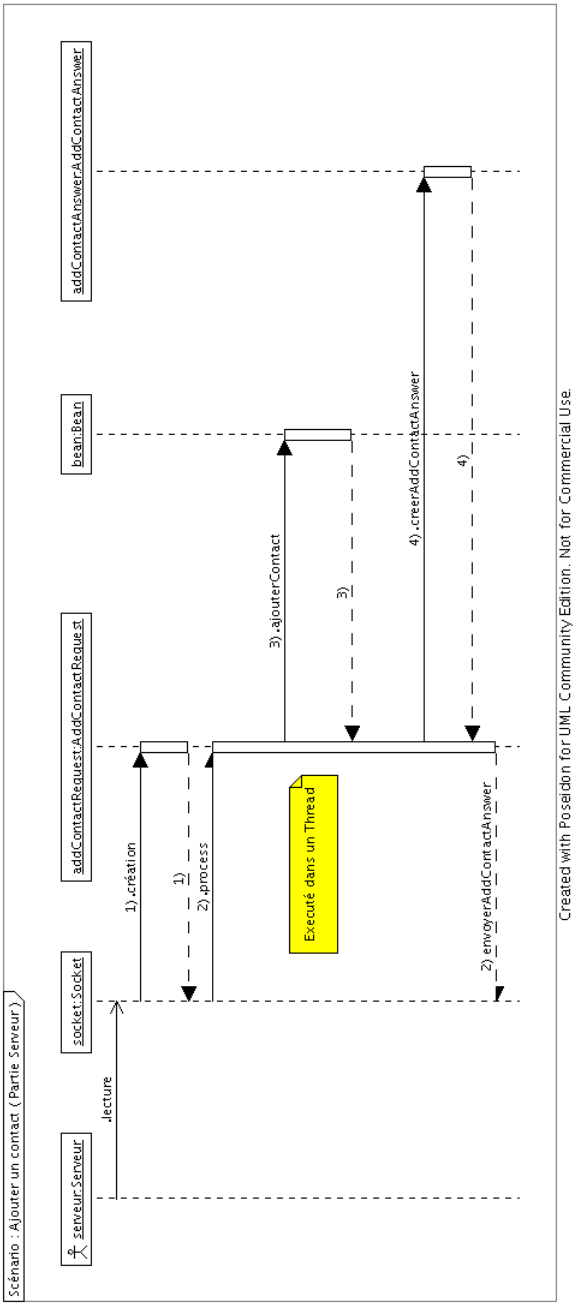


FIG. 3.13 – Ajout d'un contact (Serveur)

3.6 Diagrammes de classes

3.6.1 Les diagrammes de classes

Les diagrammes de classes sont les diagrammes les plus connus du langage UML. Ils permettent d'appréhender, d'un point de vue logique, la structure statique du système en indiquant :

- la structure des objets composant l'application
- les liens structurels entre les objets

Nous allons ici présenter les classes illustrant la hiérarchie entre les messages. Nous avons divisé les classes en trois diagrammes afin de faciliter la lecture.

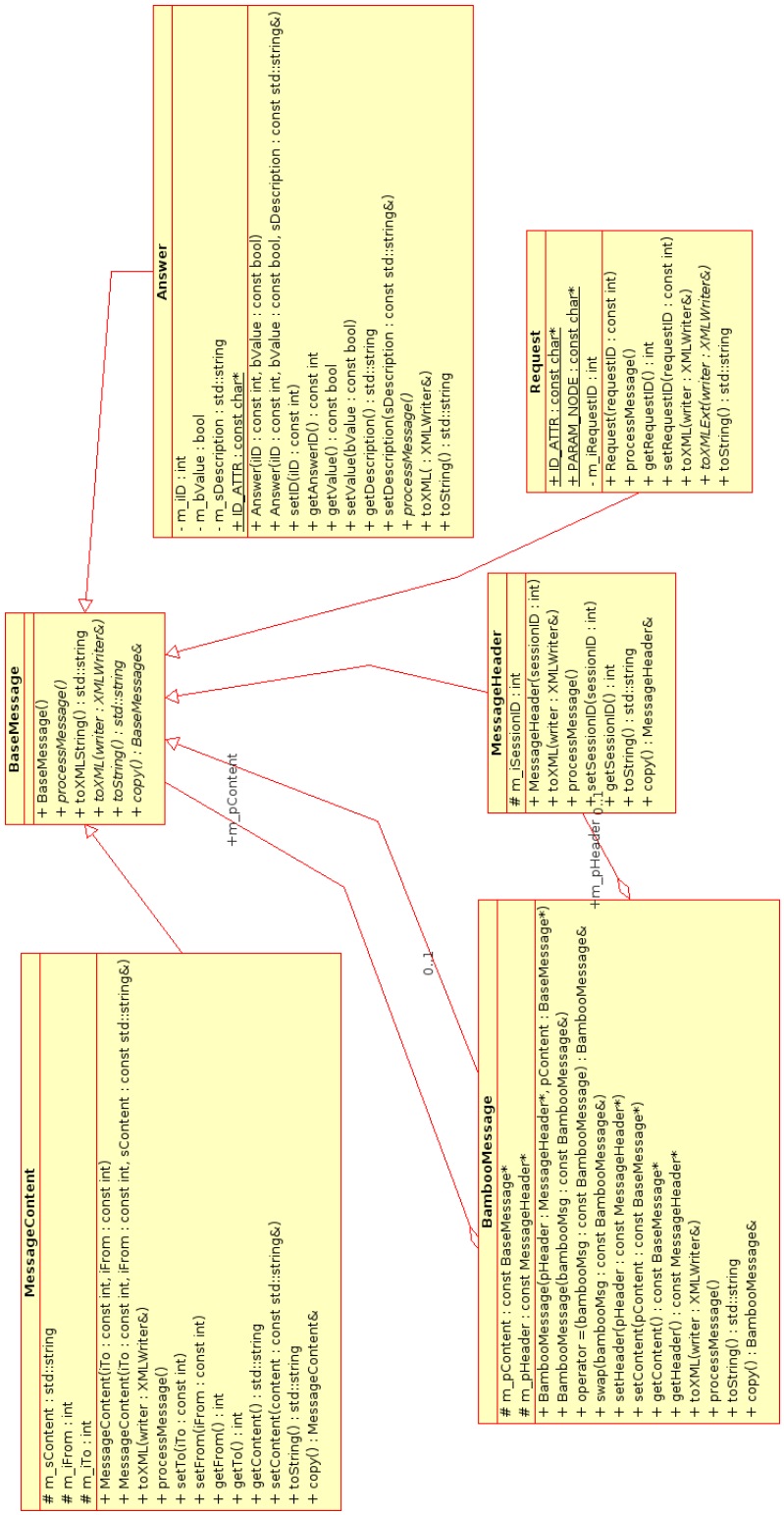


FIG. 3.14 – Diagramme des classes Message

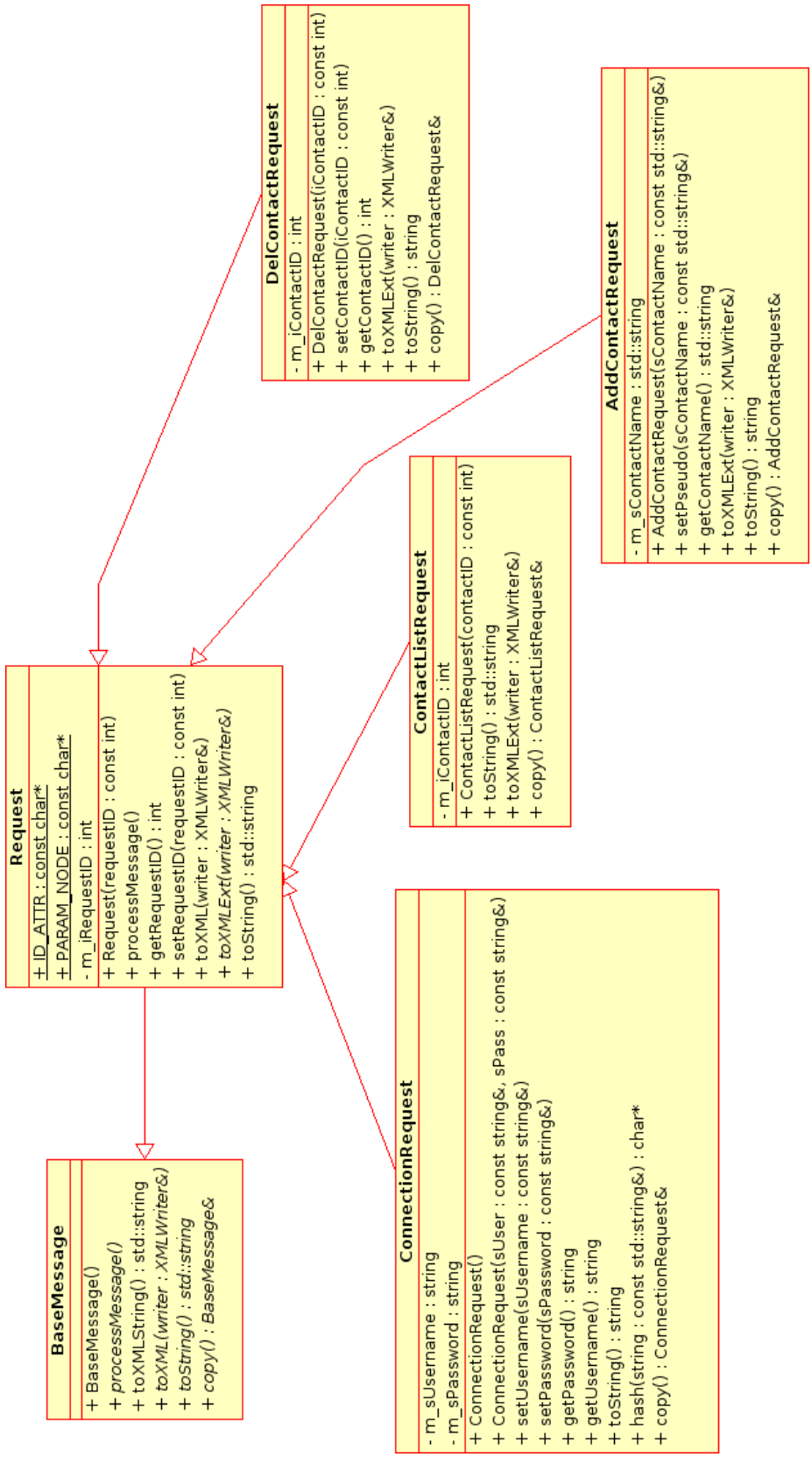


FIG. 3.15 – Diagramme des classes : Requêtes

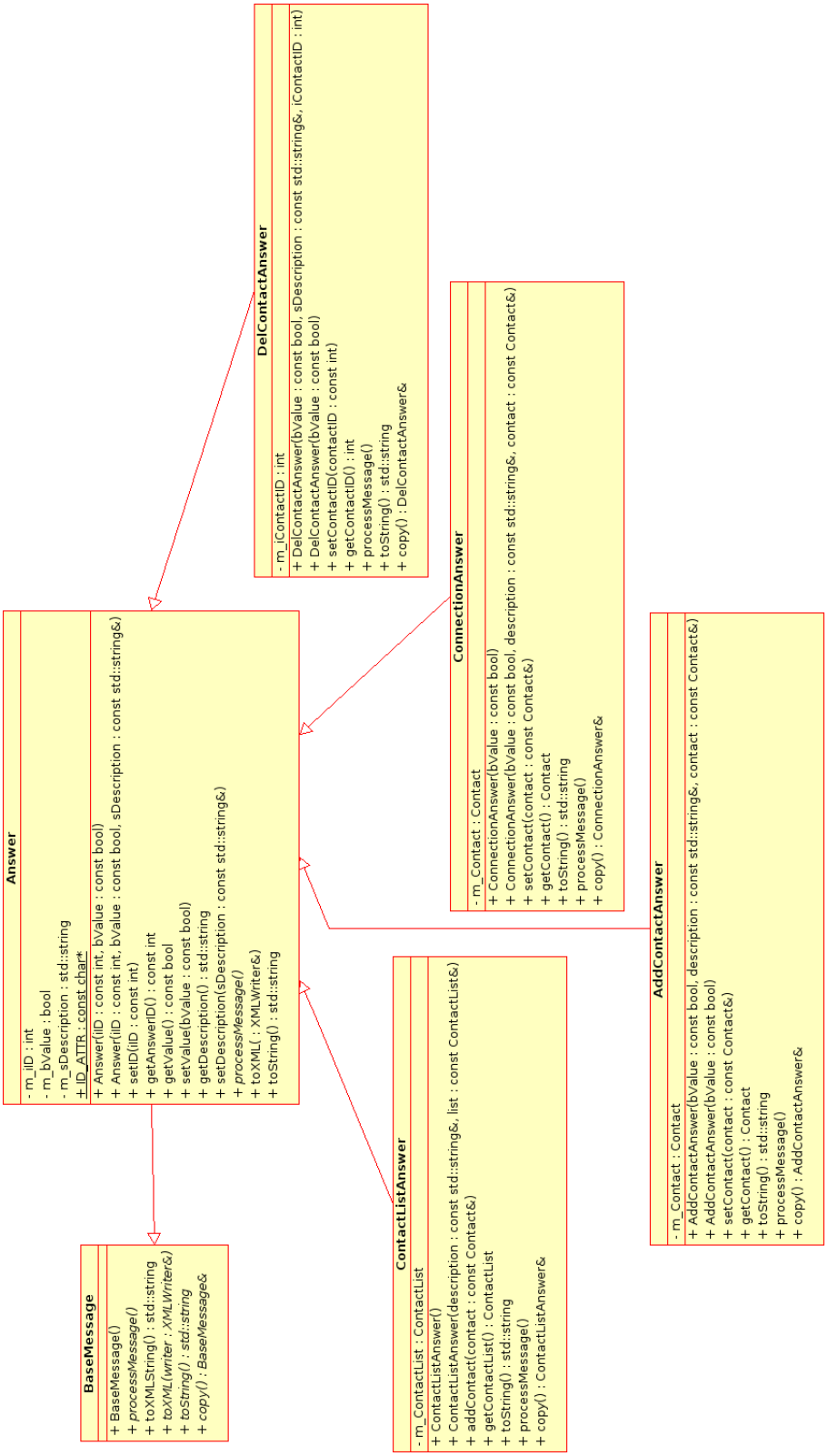


FIG. 3.16 – Diagramme des classes : Réponse

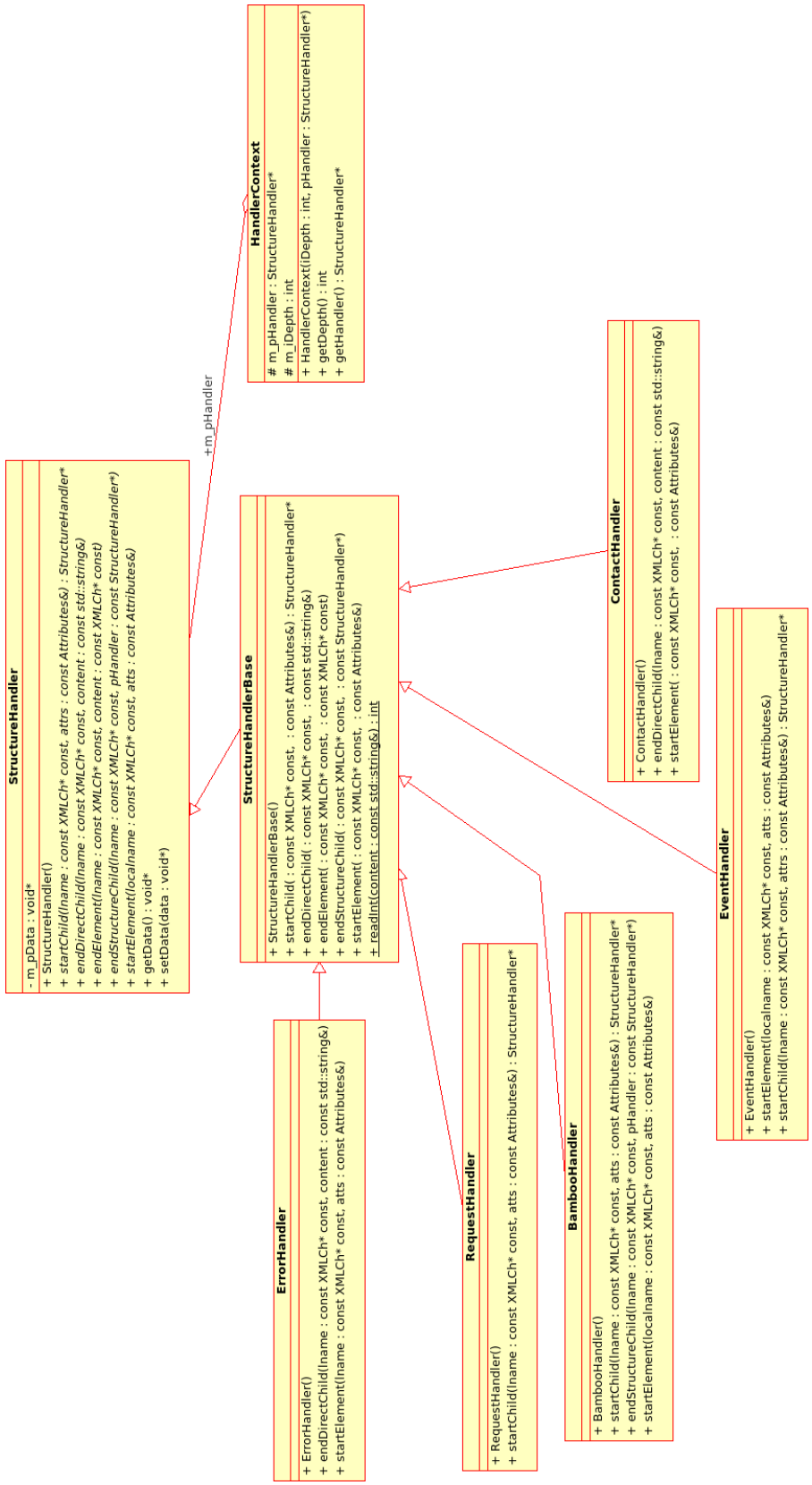


FIG. 3.17 – Diagramme des classes : Analyse XML

3.6.2 Les paquetages

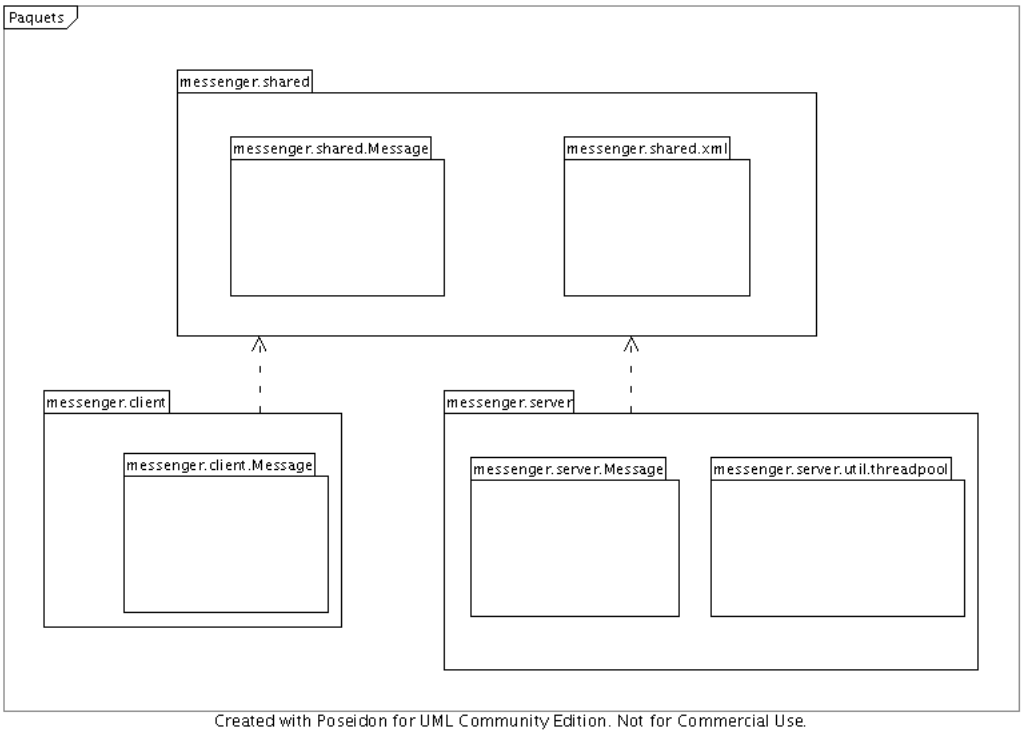


FIG. 3.18 – Paquets

3.7 Diagrammes de collaborations

Ils fournissent une représentation synthétique des interactions entre les objets.

3.7.1 Envoi d'un message

L'envoi du message est en quelque sorte la phase antérieure à la réception du message. La fenêtre dans laquelle on a passé le message envoie le message au coeur de l'application : la classe KernelMessenger. Cette classe gère l'interface entre le graphique et le programme en lui-même. Le message est alors codé en XML à l'aide de la méthode toXML(). Le message sous cette forme est envoyé au serveur pour être traité de la manière suivante.

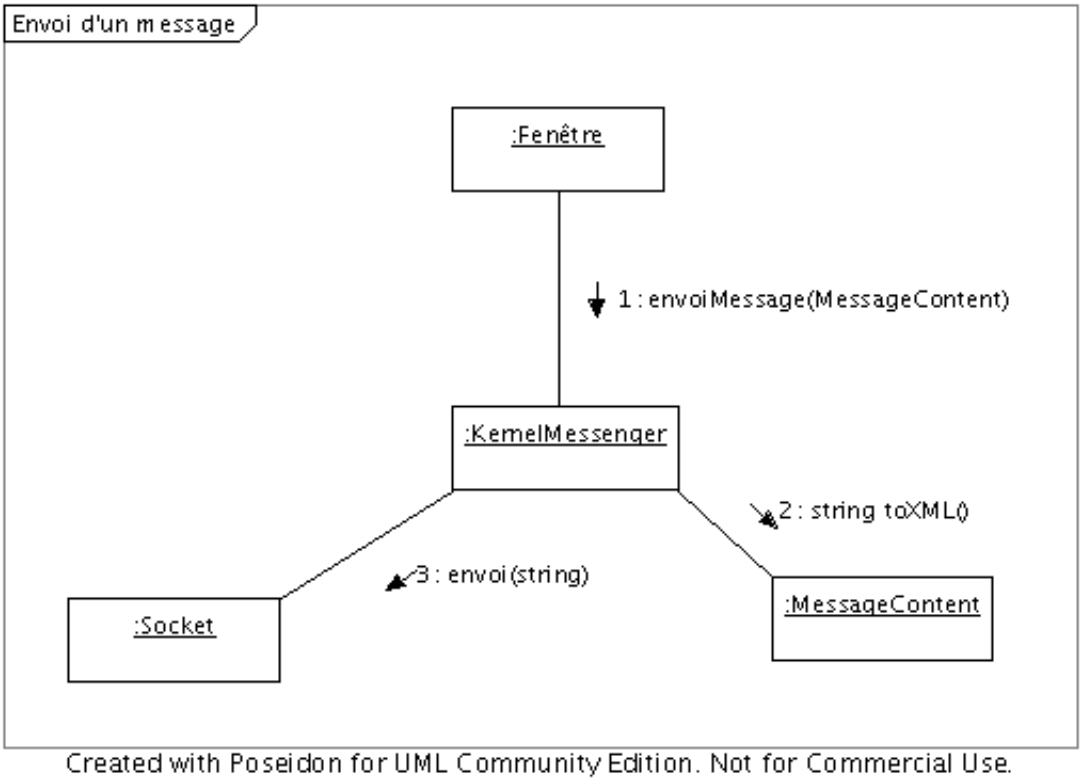


FIG. 3.19 – Envoyer un message

3.7.2 Réception d'un message (Partie Serveur)

Le déroulement de cette action est centré autour des PoolWorker. Pour bien comprendre ce qu'est un PoolWorker, il faut expliquer la notion de ThreadPool. Un ThreadPool est une sorte de bassin où interagissent un ensemble de PoolWorker. Dès qu'une action doit être exécutée, un des threads disponible (un PoolWorker) l'effectue. Si tous les PoolWorker sont déjà occupés, le premier qui deviendra libre s'occupera de la tâche. Pendant ce temps, la tâche est mise en attente.

Le Socket reçoit donc le message sous format XML. Le message est ensuite passé à un PoolWorker disponible. Celui-ci le décode pour le traduire en type MessageContent à l'aide de XMLParser. Puis, on récupère le destinataire. Enfin, on retransforme le message en XML avant de l'envoyer au client. Il est important de noter que lorsque le message est crypté, des modifications sont effectuées sur le contenu du message.

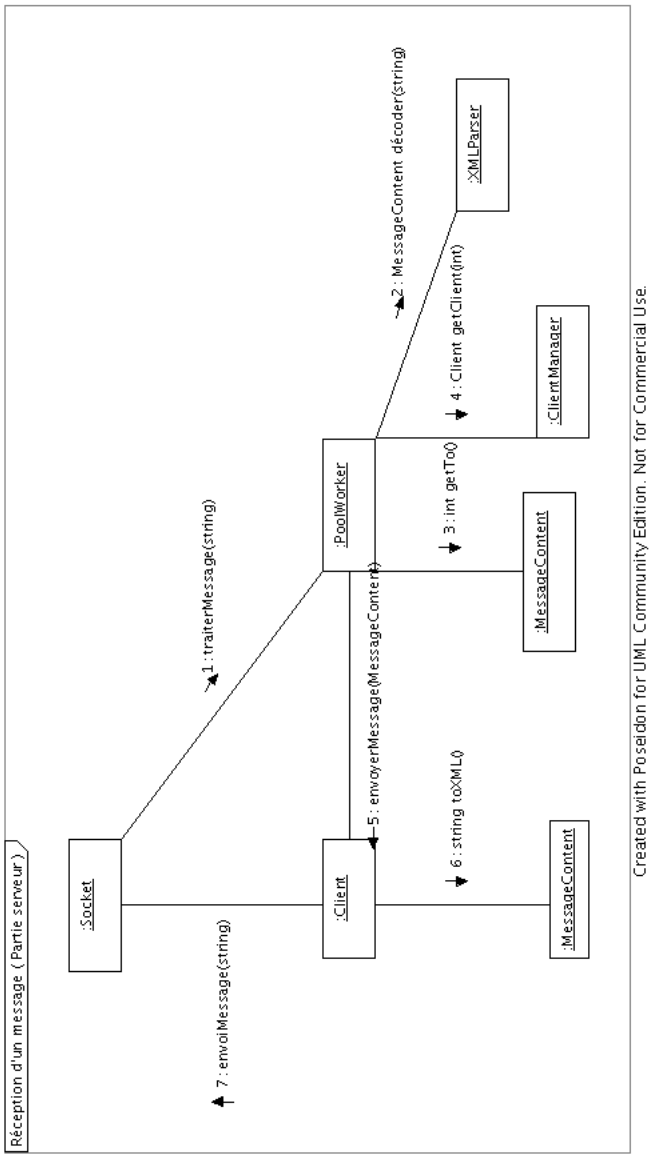


FIG. 3.20 – Réception d'un message (Partie serveur)

3.7.3 Réception d'un message (Partie Cliente)

La partie Client est plus facile. Elle est basée sur la classe KernelMessenger. Le socket envoie le message sous format XML au KernelMessenger. On effectue la même opération que précédemment en parsant le message. On récupère alors la fenêtre dans laquelle le message doit être affiché, puis on le fait apparaître dans celle-ci.

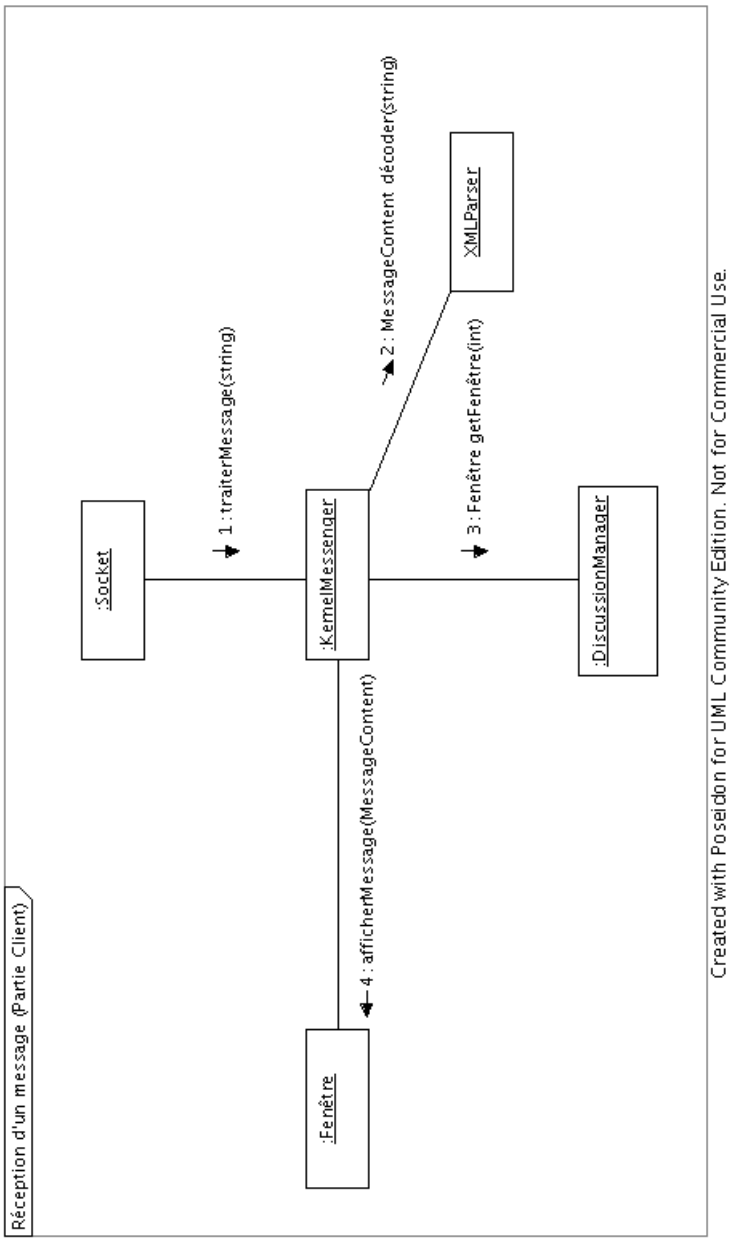


FIG. 3.21 – Réception d'un message (Partie client)

3.8 Diagrammes d'états-transitions

Les diagrammes d'états transitions ont pour but de représenter les états successifs d'un objet en fonction des sollicitations externes et/ou des interactions avec d'autres objets. Nous avons choisi de représenter les différents états de l'objet Message. Celui-ci est au coeur de notre application. Nous avons choisi de le représenter en trois parties.

* Émission : le message est initialisé puis codé en XML pour être reconnu par le serveur

* Partie Serveur/Client : le message est analysé puis renvoyé au serveur

* Réception : le message est décodé. Il est sous la forme Message. Il est alors affiché dans la fenêtre du destinataire

3.8.1 Émission du message

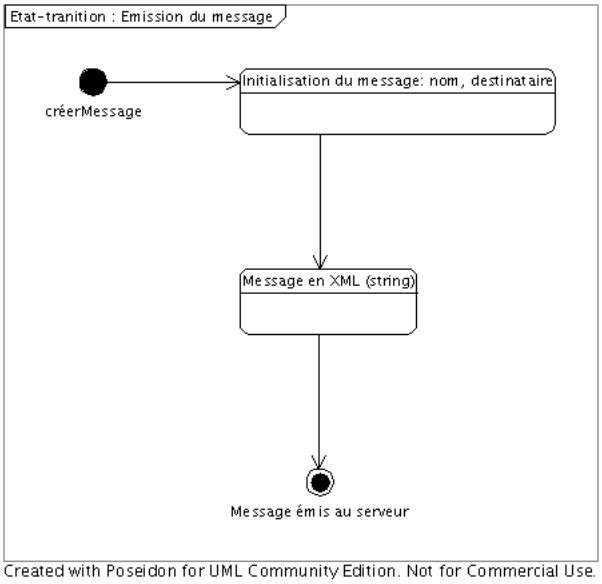


FIG. 3.22 – Émission d'un message

3.8.2 Serveur/Client

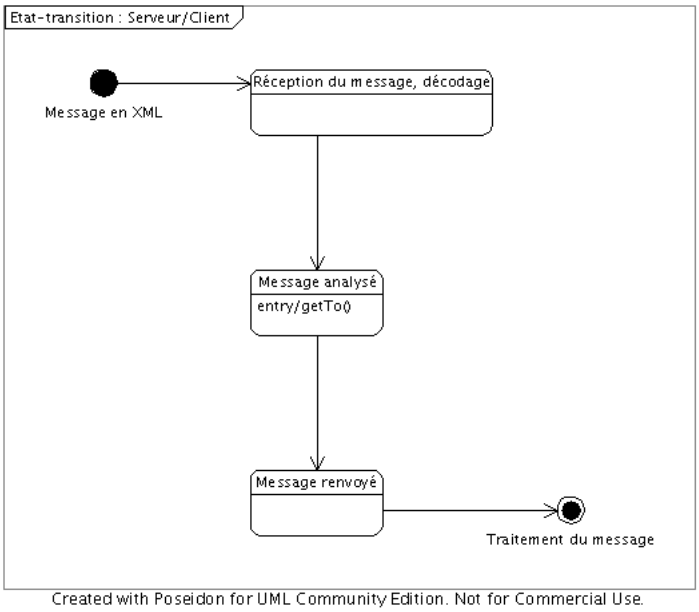


FIG. 3.23 – Serveur/Client

3.8.3 Réception du message

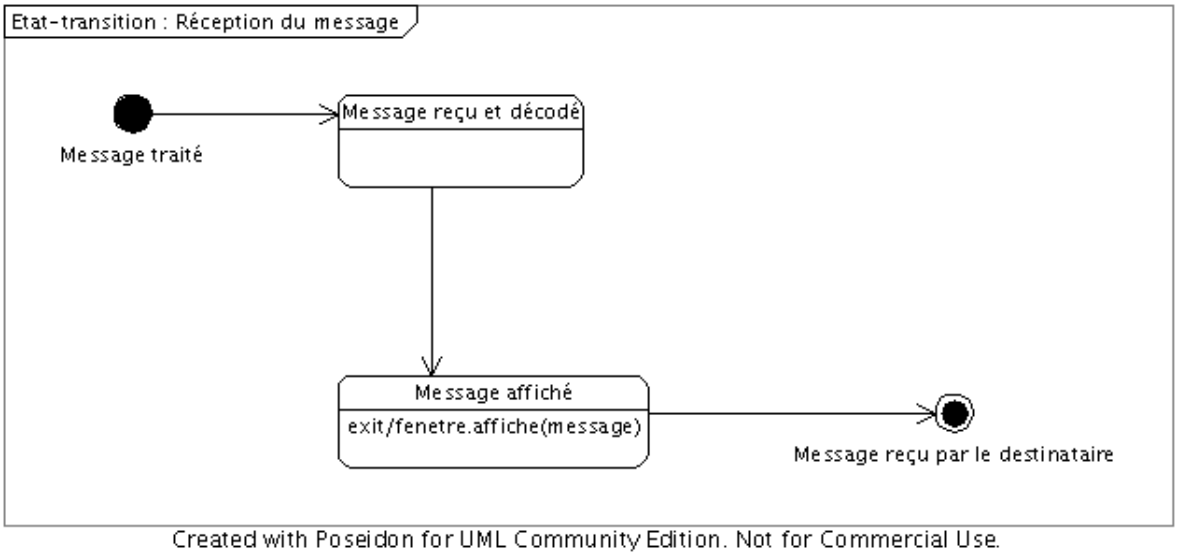


FIG. 3.24 – Réception du message

Chapitre 4

Conclusion

Ce projet fût très intéressant à réaliser. Il nous a permis de bien comprendre la démarche à suivre pour la réalisation de futurs grands projets. Il semble très efficace de se tenir à cette méthode car elle prend en compte la plupart des problématiques à résoudre. Ce projet a été conçu en parallèle avec le développement de notre application. Au début, les diagrammes classiques tels que ceux de séquences ou de classes avaient déjà été effectués pour bien établir le design de l'application. Nous avons vu que UML permet également une étude détaillée grâce par exemple aux diagrammes d'états-transitions et de collaborations qui sont très bénéfiques à la compréhension du logiciel. Le langage UML est de plus très visuel, ce qui est bien sûr un atout lors de la conception de l'application. Cependant, la version 1 du langage UML souffre de quelques lacunes au niveau du parallélisme et de la programmation réseau. UML2 semble combler ces petits manques et le futur aUML (a pour agent) permettra au langage d'être encore plus complet.

Table des figures

3.1 Business Model 5
3.2 Les différents cas d'utilisation 6
3.3 Connexion à BambooCh@t 7
3.4 Ajouter un contact 8
3.5 Supprimer un contact 9
3.6 Envoyer un message 10
3.7 Connexion à BambooCh@t (Client) 11
3.8 Connexion à BambooCh@t (Serveur) 12
3.9 Envoyer un message (Client) 13
3.10 Envoyer un message (Serveur) 14
3.11 Réception d'un message 15
3.12 Ajout d'un contact (Client) 15
3.13 Ajout d'un contact (Serveur) 16
3.14 Diagramme des classes Message 18
3.15 Diagramme des classes : Requêtes 19
3.16 Diagramme des classes : Réponse 20
3.17 Diagramme des classes : Analyse XML 21
3.18 Paquets 22
3.19 Envoyer un message 23
3.20 Réception d'un message (Partie serveur) 24
3.21 Réception d'un message (Partie client) 25
3.22 Émission d'un message 26
3.23 Serveur/Client 27
3.24 Réception du message 27