Erlang IRC Server

**Compte rendu d’analyse**

Auteurs : Vincent Berthoux, Vincent Gonçalves, Nicolas Zermati

Table des matières

[Choix d’orientation 3](#_Toc192332334)

[Les consignes données par Nicolas Fauvet 3](#_Toc192332335)

[Notre interprétation du sujet 3](#_Toc192332336)

[Les étapes d’analyses 3](#_Toc192332337)

[Le choix du langage Erlang 4](#_Toc192332338)

[Nos priorités 4](#_Toc192332339)

[Les difficultés rencontrées 5](#_Toc192332340)

[Outils UML 5](#_Toc192332341)

[Répartition des tâches 6](#_Toc192332342)

# Choix d’orientation

## Les consignes données par Nicolas Fauvet

Réaliser un logiciel de messagerie instantanée. De préférence basé sur IRC dans ce cas il n'est pas nécessaire de développer le serveur. Il nous a été conseillé d'ajouter des fonctionnalités graphiques telles que le tableau blanc, ou l’envoi de dessins réalisés à la souris. Mis à part ces lignes directrices aucune autre contrainte ne nous a été posée.

## Notre interprétation du sujet

A Partir de la consigne qui nous a été donnée nous nous sommes focalisé sur deux choses : le fait qu’un logiciel de messagerie instantanée comporte un client mais aussi un serveur et le fait que ce logiciel soit basé sur IRC. A partir de ce constat nous avons jugé plus intéressant la création d’un serveur compatible avec les clients IRC actuels plutôt qu’un nième client IRC en Java.

Dans l’optique de réaliser un serveur nous nous sommes documenté sur le protocole IRC et sur ses implantations existante. Nous avons choisit de nous baser sur la RFC 2812 et 2813. Après avoir observé le fonctionnement habituel d’un serveur normalisé, nous avons effectué les étapes d’analyse.

## Les étapes d’analyses

Nous avons constaté qu’il n’y avait que peu de cas d’utilisations différents du serveur. Ces cas sont décrits dans notre diagramme des cas d’utilisation que vous trouverez jointe à ce dossier. La majorité des fonctionnalités passent par une commande envoyée au serveur. On a tout de même catégorisé ces différentes commandes en différents usages.

Nous avons également fourni un cahier des charges à notre *client*. Pour cela nous avons du avancer dans l’analyse de notre projet en proposant concrètement un projet, en évaluant ses coûts, en s’engageant sur des fonctionnalités et sur une compatibilité.

A travers le cahier des charges nous avons du informer le client du langage de programmation utilisé, et des pré-requis à l’installation de notre logiciel sur une machine. Et donc par conséquent nous avons du choisir un langage de programmation. Nous avons donc étudié quel serai le langage le plus adapté et offrant le plus de facilité de développement. Notre choix s’est donc porté sur un langage appelé Erlang.

Nous avons par la suite analysé la RFC et nous avons formalisé au travers de diagrammes de séquences chacun des messages que le serveur sera susceptible de recevoir. Nous avons également élaboré quelques scénarios d’utilisation toujours grâce à des diagrammes de séquence. Ces digrammes son joints à ce dossier.

Notre analyse comporte également un diagramme d’objet. Grâce à ce diagramme et à la RFC du protocole, nous avons établi une partie des contraintes qui s’applique sur le modèle. Pour faciliter la compréhension du diagramme nous avons également établi un dictionnaire des données. Le diagramme, les contraintes et le dictionnaire des données sont également joints au dossier.

## Le choix du langage Erlang

Erlang est un langage de développement concurrent temps réel. De part sa conception, les applications écrites en erlang sont très résistantes aux pannes et surtout, leur code peut être mis à jour en temps réel. Il est donc principalement utilisé pour des applications nécessitant une haute disponibilité.

En 1998, Erlang devint [Open Source](http://en.wikipedia.org/wiki/Open_source) et commence à être de plus en plus utilisé par les entreprises de télécommunication, pour les routeurs téléphoniques. Mais erlang a également été utilisé pour le développement de serveur de messagerie instantanée.

Une des caractéristiques des plus intéressantes d’Erlang est le fait qu'il fonctionne par système de nœuds et de processus. Ce qui signifie que même si un nœud devient indisponible, l'application continue de fonctionner et grâce à sa possibilité de mettre à jour le code de l'application même lorsque celle-ci fonctionne, il devient alors facile de réparer le problème.

Mais le principal problème d'erlang est qu'il ne possède pas une documentation très riche étant donné que ce langage n'est pas très répandu. Si vous souhaitez plus d'informations, voici des sites internet en possédant : <http://www.erlang.org/> et <http://www.erlang.se/>

**Avantages**

* Erlang est un langage "open source".
* Capable de s'adapter à de gros projets tout en restant structuré.
* Erlang a été conçu pour programmer des applications hautement disponibles et robustes.
* Erlang est un langage fonctionnel, ce qui minimise le nombre de ligne de code (donc d'erreur).
* Erlang facilite grandement l'aspect distribué des applications ainsi que la programmation par threads.

Encore bien d'autres avantages développés sur :

[http://www.linux-france.org/prj/borsalino/projet/choix\_tech.php](http://www.linux-france.org/prj/borsalino/projet/choix_tech.php%20)

**Inconvénients**

* Erlang est nouveau pour nous.
* Ce n'est pas un langage orienté objet comme Java mais un langage orienté processus.
* Il sera difficile de mettre en œuvre exactement ce que nous avons vu en cours de GL (mais néanmoins possible).

## Nos priorités

* Stabilité.
* Compatibilité avec les clients IRC.

# Les difficultés rencontrées

Nous n’avons pas rencontré beaucoup de difficultés. La première et la seule que nous avons rencontré est celle du choix d’un outil de modélisation UML. Cela provient en parti du fait que nous utilisons un ensemble d’outils de travail collaboratifs évolués (Trac, et Subversion). Ceci nous prémuni donc contre une grande partie des problèmes dus a cet aspect d’un projet.

Une autre difficulté que nous avons eue est que comme nous avons commencé le projet avec la documentation RFC du protocole IRC, nous avons commencé à rédiger des scénarios pour chaque message (Le protocole IRC comporte une trentaine de messages). Mais cela fait partit, plus précisément, de la partie conception du projet. Donc nous avons refait 2 scénarios généraux qui regroupent le fonctionnement de tous les messages supportés par le serveur.

## Outils UML

Cela parait stupide, mais la recherche d'un outil UML qui pourrait créer un diagramme de séquence pour le client/serveur est une opération complexe, surtout sous les contraintes suivantes :

* Doit fonctionner sous Windows & Unix.
* Doit permettre de mettre des noms arbitraires sur les flèches.
* Doit ne pas avoir une interface horrible.

Les rejetés sont :

* Dia : interface très peu pratique (mais vraiment très peu).
* Violet UML : ne gère pas les messages asynchrones.
* Netbeans 6.0 : Beaucoup trop stricte dans son nommage, nous ne pouvons pas placer des noms de notre choix sur les interactions.
* Visio 2003 : fonctionne que sous Windows.

**Solutions trouvées**

Nous avons prit la décision d'alléger les contraintes, les personnes sous Windows utilisent Visio 2003 et les personnes sous Linux utilisent ce qu'elles trouvent ([InkScape](http://fr.wikipedia.org/wiki/Inkscape) a été utilisé).

# Répartition des tâches

Nous nous sommes répartis les taches de manière assez égale. Les systèmes de travail collaboratif ont été mis en place par Vincent Berthoux et nous avons donc pu en profiter pour tous collaborer autour de la rédaction d’un wiki dédié au projet.

Nous avons donc tous participé à l’élaboration des pages du wiki. Lorsque nous nous sommes répartis le travail qu’étais la transcription des messages IRC en diagrammes de séquence, nous avons séparé en trois les messages possibles. Nous avons par la suite effectué chacun notre travail.

Cette méthode de travail nous permet donc de nous répartir les tâches et d’avancer de façon uniforme dans les phases du projet. Nous avons créé des pages pour tout les documents que nous avons du rendre et chacun s’est occupé de pages.

Les pages qui ont étés créées sont les suivantes :

* L’interface d’administration
* Le cahier des charges (reprenant le cahier des charges qui à été rendu)
* Le compte rendu d’analyse
* Les canaux de discussion (principe de la discussion par canal)
* Le ChanServ (Bot qui enregistrera les canaux des utilisateurs)
* Les cas d’utilisation
* La page d’établissement de la connexion d’un utilisateur vers un serveur
* La page des conseils de rédactions
* La page des diagrammes d’objet et de classe
* Le dictionnaire des données
* La page des invariants
* 29 pages de messages IRC avec leurs diagrammes
* Le choix du langage Erlang
* Le NickServ (Bot de réservation de pseudonyme)
* Les Bilan des séances avec notre *client*
* Les problèmes rencontrés
* Et d’autres pages

Ces pages sont disponibles à l’adresse <http://217.70.188.170/trac/jim/>

Illustrons maintenant notre répartition des tâches :

