Erlang IRC Server

**Rapport d’analyse**

Auteurs : Vincent Berthoux, Vincent Gonçalves, Nicolas Zermati.

Table des matières

[Diagramme des cas d’utilisation : - 3 -](#_Toc192333997)

[Modèle des Objets : - 3 -](#_Toc192333998)

[Diagramme d’Objet : - 5 -](#_Toc192333999)

[Ivariants : - 6 -](#_Toc192334000)

[Client : - 6 -](#_Toc192334001)

[Serveur : - 6 -](#_Toc192334002)

[Canal : - 6 -](#_Toc192334003)

[Dictionnaire des Données : - 8 -](#_Toc192334004)

[IRC Message - 8 -](#_Toc192334005)

[Server Node - 8 -](#_Toc192334006)

[Client - 8 -](#_Toc192334007)

[Chan - 8 -](#_Toc192334008)

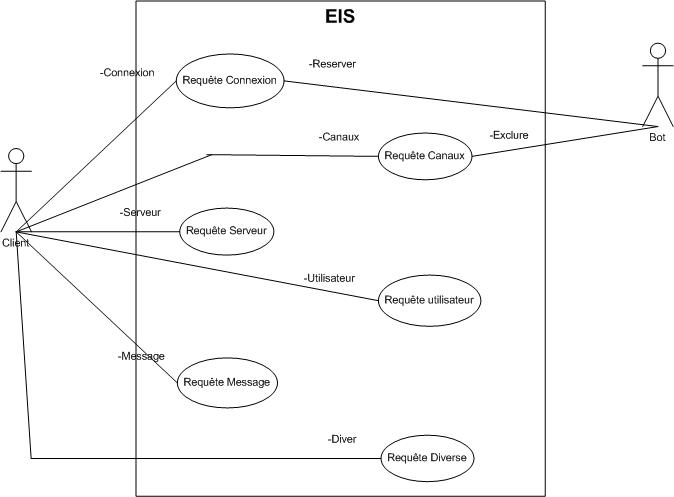
[Permissions - 9 -](#_Toc192334009)

[Perm Info - 9 -](#_Toc192334010)

[Scénario d’utilisation : - 10 -](#_Toc192334011)

[Manuel d’utilisation : - 11 -](#_Toc192334012)

# Diagramme des cas d’utilisation :



Nous avons 2 utilisateurs :

* Le Client
* Et un client particulier : le Bot

Par définition le client peut presque tout faire sur le système par l’intermédiaire de commande envoyé au serveur.

Plusieurs types de commandes peuvent être exécutés par le client :

* Les requêtes de type Connexion : qui permettent de se connecté à un serveur IRC, le quitter, mais aussi changer de pseudo …
* Les requêtes de type Canaux : qui permettent de rejoindre un canal spécifique, lister les canaux disponible, « kicker » un autre client d’un canal, …
* Les requêtes de type Serveur : qui permettent de recevoir des informations propre à un serveur.
* Les requêtes de type Message : qui permettent d’envoyer un message.
* Les requêtes de types utilisateurs : qui permettent de recevoir des informations d’un client
* Et enfin les requêtes de type Diverse : qui sont toutes les commandes qui ne rentrent pas dans les catégories définies ci-dessus.

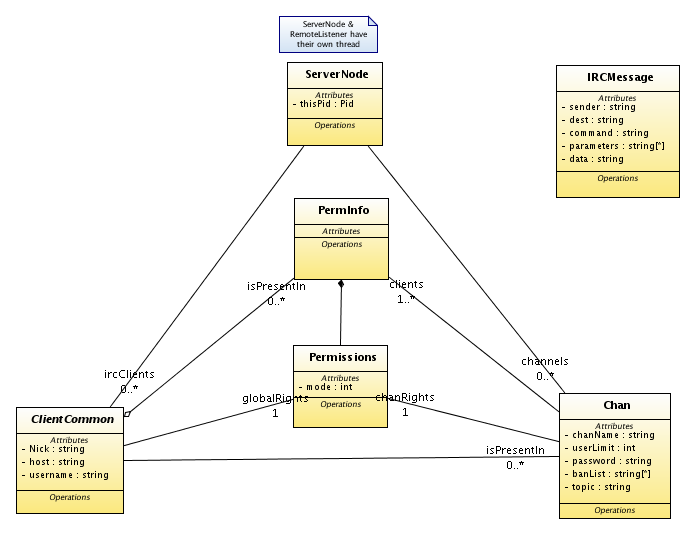
En revanche le Bot, qui a pour but de surveiller les canaux, pourra a toute instant exclure un client d’un canal. Deux sortes de motifs sont autorisées pour une exclusion :

* Le Spam
* Le Flood.

De plus un autre Bot sera mis en place pour permettre de réserver un pseudo sur le serveur.

# Modèle des Objets :

## Diagramme d’Objet :



En revanche le langage de programmation choisit, étant le langage ErLang qui est un langage fonctionnel orienté processus. Le diagramme d’objet présenté ci-dessus a été légèrement modifié pour pouvoir s’accorder à un diagramme traditionnel.

## Ivariants :

Dans cette partie nous présentons les différents invariants liés aux clients, au serveur et aux canaux :

### Client :

1. Le pseudo (Nick) de l’utilisateur est unique :

context Client

inv : forAllInstances()->isUnique( Nick )

1. Un utilisateur ne peut reçevoir les messages d’un canal que s’il est présent sur ce canal.

### Serveur :

1. Seuls les canaux visibles peuvent être listés par le client :

context ServerNode::List

post : result->forAll( c | NOT c.chanRights.isInvisible() )

### Canal :

1. Le nom du canal commence forcement par '#' ou '&' :

context Chan

inv : chanName->first() = '#' OR chanName->first() = '&'

1. Le nom de canal est unique :

context Chan

inv : forAllInstances->isUnique(chanName)

1. Le nombre d'utilisateur sur le canal ne peut excéder userLimit :

context Chan

inv : userLimit >= clients->size()

1. Un utilisateur banni ne peut rejoindre un chan :

context Chan::Join( user : CommonClient, sendedPassword : string )

inv : not isUserBaned( cli )

1. Si le canal est en mode *voiced* alors il faut avoir le mode +v pour que les messages soient transmis aux autres utilisateurs :

context Chan::Privmsg( sender : CommonClient, message : Text )

inv : (chanRights.isModed() AND clients->select( c | c = sender

AND c.rights.isVoiced() )) OR NOT chanRights.isModed()

1. Si le canal est protégé par mot de passe, seul un utilisateur fournissant le bon mot de passe pourra le rejoindre :

context Chan::Join( user : CommonClient, sendedPassword : string

inv : password = "" OR sendedPassword = password

1. La liste de clients *diffusés* ne peut pas être un client lointain :

context Chan

inv : broadcastedClients->forAll( c | no

oclIsTypeof(RemoteClient))

1. Tous les clients d'un canal doivent avoir celui ci dans leur liste :

context Chan

inv : broadcastedClients->forAll( c |

c.isPresentIn->exists( chan | chan = self() )

# Dictionnaire des Données :

## IRC Message

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code** | **Nom** | **Légende** | **Type** | **Règles de gestion** |
| MSG\_SENDER | sender | Expéditeur du message | String | Obligatoire, non null |
| MSG\_RECEIVER | dest | Destinataire du message | String | Obligatoire, non null |
| MSG\_COMMAND | command | Commande obligatoire | String | Obligatoire, non null |
| MSG\_PARAMETERS | parameters | Paramètres optionnels de la commande | String | Facultatif |
| MSG\_DATA | data | Données optionnelles du message | String | Facultatif |

## Server Node

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code** | **Nom** | **Légende** | **Type** | **Règles de gestion** |
| SRV\_PID | thisPid | Pid du processus serveur | Pid | Non null |

## Client

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code** | **Nom** | **Légende** | **Type** | **Règles de gestion** |
| CLI\_NICK | nick | Pseudonyme du client | String | Unique, non null |
| CLI\_HOST | host | URI | String | Non null |
| CLI\_USERNAME | username | Nom réel de l'utilisateur | String | Non null |

## Chan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code** | **Nom** | **Légende** | **Type** | **Règles de gestion** |
| CHN\_NAME | chanName | Le nom du canal | String | Non null, commence par un # ou un & |
| CHN\_LIMIT | userLimit | La limite d'utilisateurs présents | Entier | > 1 et < Limite fixée par le serveur |
| CHN\_PWD | password | Le mot de passe d'accès au canal | String | Optionnel, ne doit pas contenir de caractères spéciaux voir la commande mode |
| CHN\_BAN\_LIST | banList | La liste des utilisateurs banni | Liste de string |  |
| CHN\_TOPIC | topic | Le sujet du canal | String |  |

## Permissions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code** | **Nom** | **Légende** | **Type** | **Règles de gestion** |
| PRM\_MODE | mode | Permissions pour un canal ou pour un utilisateur | Entier |  |

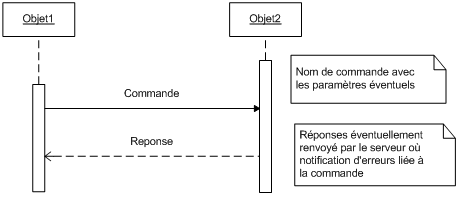
## Perm Info

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code** | **Nom** | **Légende** | **Type** | **Règles de gestion** |
| INF\_CLIENT | client | Pseudonyme du client dont on veut définir les permissions dans INF\_CHANNEL | String |  |
| INF\_PERM | permissions | Permissions dans INF\_CHANNEL d'un client | Permissions |  |
| INF\_CHANNEL | chan | Nom du canal dans lequel INF\_CLIENT a les permissions INF\_PERM | String |  |

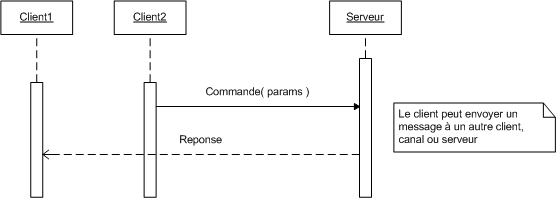
# Scénario d’utilisation :

Les scénarios présentés ci-dessous, montrent, de façon générale, le fonctionnement du serveur.

Le premier scénario montre l’exécution d’une commande simple entre le client et le serveur. Selon le type de la commande envoyé, le serveur renvoi ou non une réponse. Une réponse peut être le résultat de la commande envoyé ou alors une erreur propre à la commande.



Le deuxième scénario est utilisé lors d’échange de message entre les clients. Un premier client envoi son message au serveur. Enfin le serveur envoi le message au(x) destinataire(s) ou renvoi une erreur propre à la commande.



# Manuel d’utilisation :

Comme nous avons décidé avec notre client de ne faire que la partie serveur du projet, le manuel d’utilisateur ne peut être écrit en ce moment.

En revanche ce que nous pouvons être sûrs est que ce manuel contiendra les informations nécessaires à l’administrateur pour la maintenance du serveur.