Live-Jamming

Documentation Technique

15/05/2010

Nom du projet	**LIVE-JAMMING**
Chef de projet	NESPO Pierre
Membres	DUPUY Mathieu O'CONNEL Gregory PETIT Aylic SARDA Charles
Nom du fichier	2011_TD_FR_LIVEJAMMING
Date	15/05/2010
Auteur	NESPO Pierre

Sommaire

I.Modélisation générale

1.Client

Le client live-jamming a pour principaux composants :

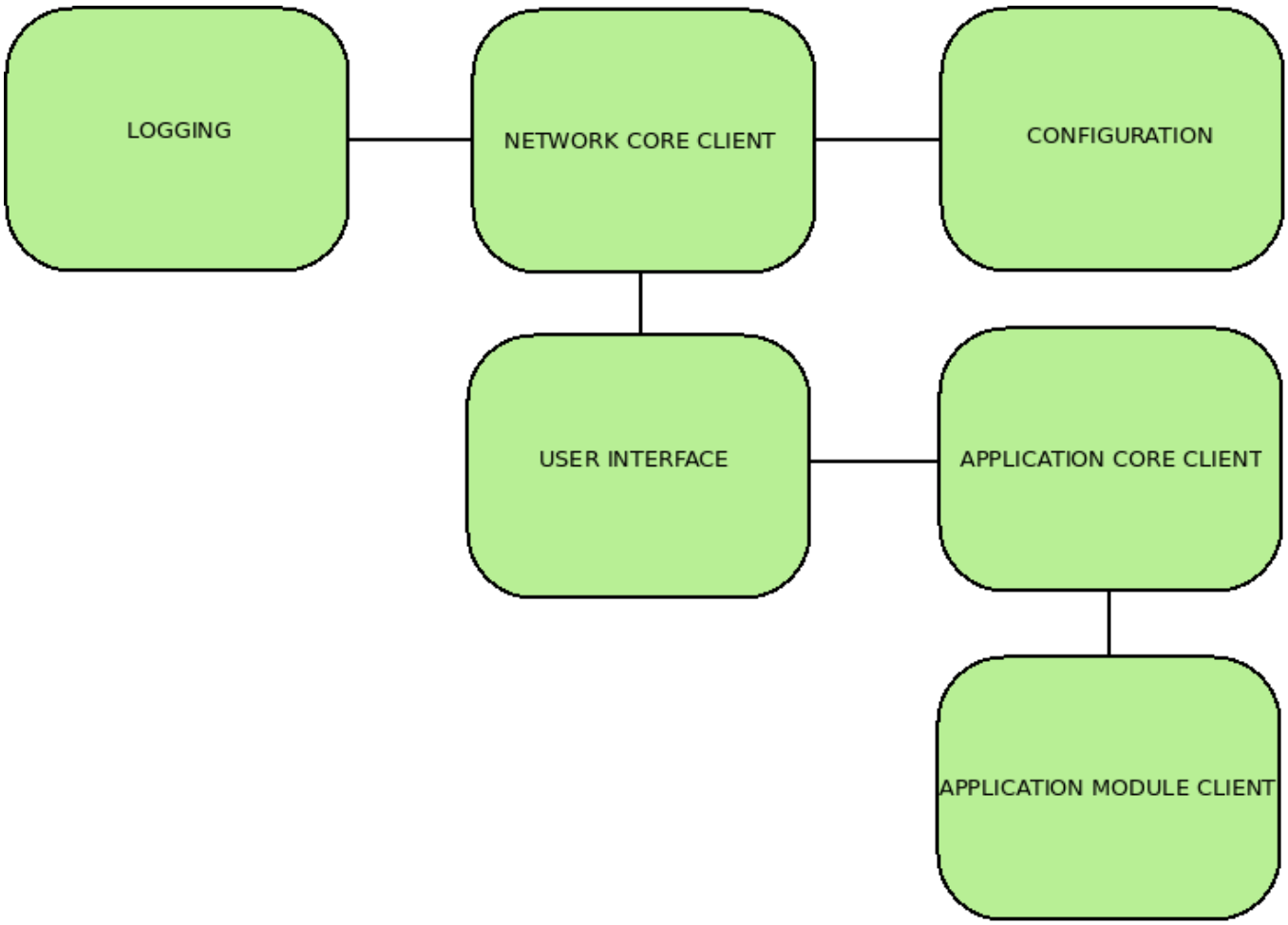
- Une interface utilisateur (User Interface).
- Un cœur réseau (Network Core).
- Un cœur applicatif (Application Core).
- Une interface IapplicationModule qui permettra d'ajouter souplement des modules clients (ajout de fonctionnalités sans toucher le cœur de l'application).

Il sera aussi composé de modules mineurs pas tels que :

- Logging (module permettant de garder une trace des actions effectuées).
- Configuration (module permettant de charger les configurations clients depuis des fichiers).

Pour une description détaillée de chaque module/Interface merci de vous rendre dans la section suivante.

Cf schéma ci-dessous.


LOGGING
NETWORK CORE CLIENT
CONFIGURATION
USER INTERFACE
APPLICATION CORE CLIENT
APPLICATION MODULE CLIENT

1.Serveur

Le serveur live-jamming a pour principaux composants :

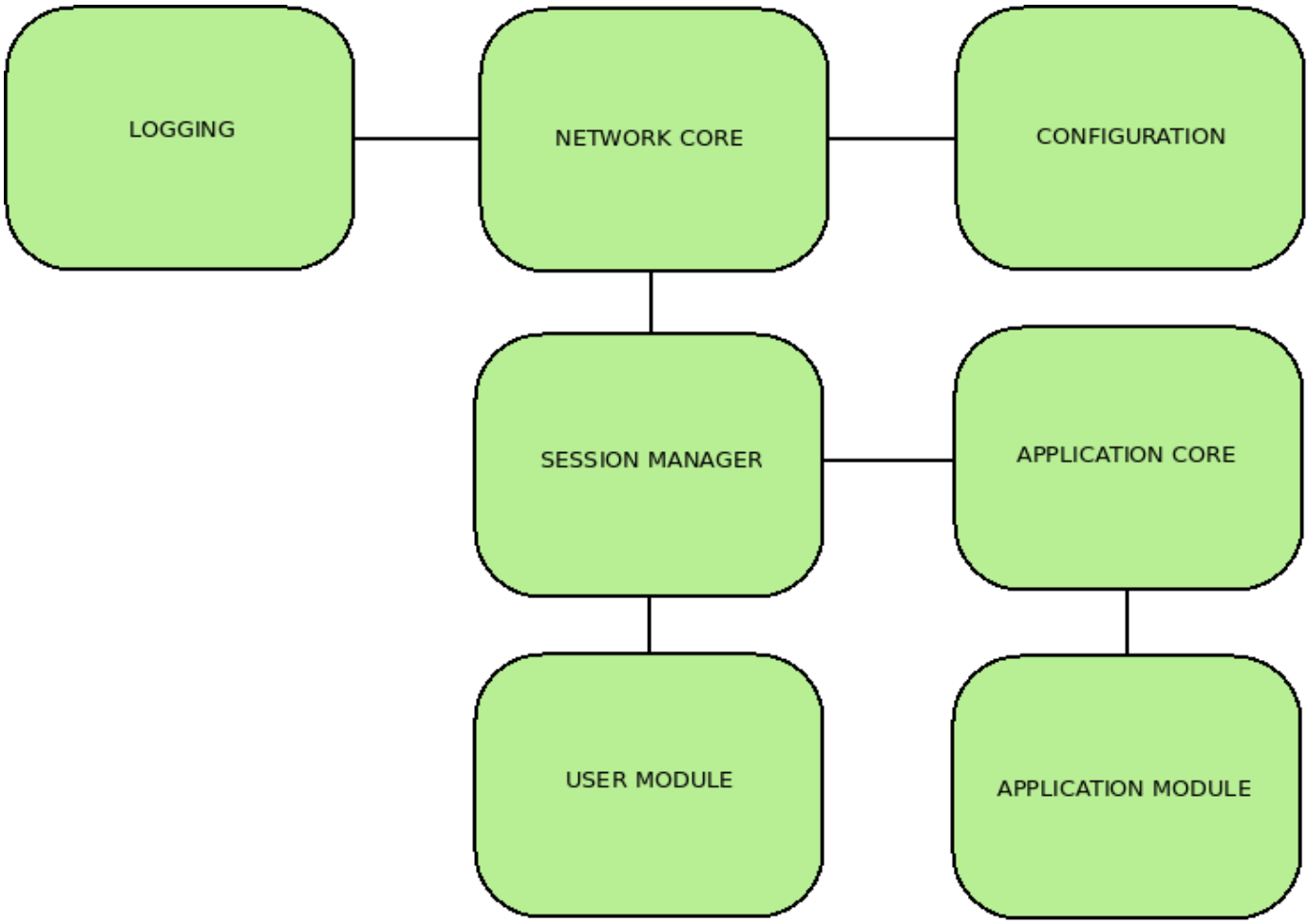
- Un cœur réseau (Network Core).
- Un module pour la gestion des sessions utilisateurs (Session management).
- Un cœur applicatif (Application Core).
- Une interface IUserModule qui permettra d'ajouter souplement des modules relatifs aux utilisateurs (ajout de fonctionnalités sans toucher le cœur de l'application).
- Une interface IapplicationModule qui permettra d'ajouter souplement des modules serveur (ajout de fonctionnalités sans toucher le cœur de l'application).

Il sera aussi composé de modules mineurs tels que :

- Logging (module permettant de garder une trace des actions effectuées).
- Configuration (module permettant de charger les configurations clients depuis des fichiers).

Pour une description détaillée de chaque module/Interface merci de vous rendre dans la section suivante.

Cf schéma ci-dessous.

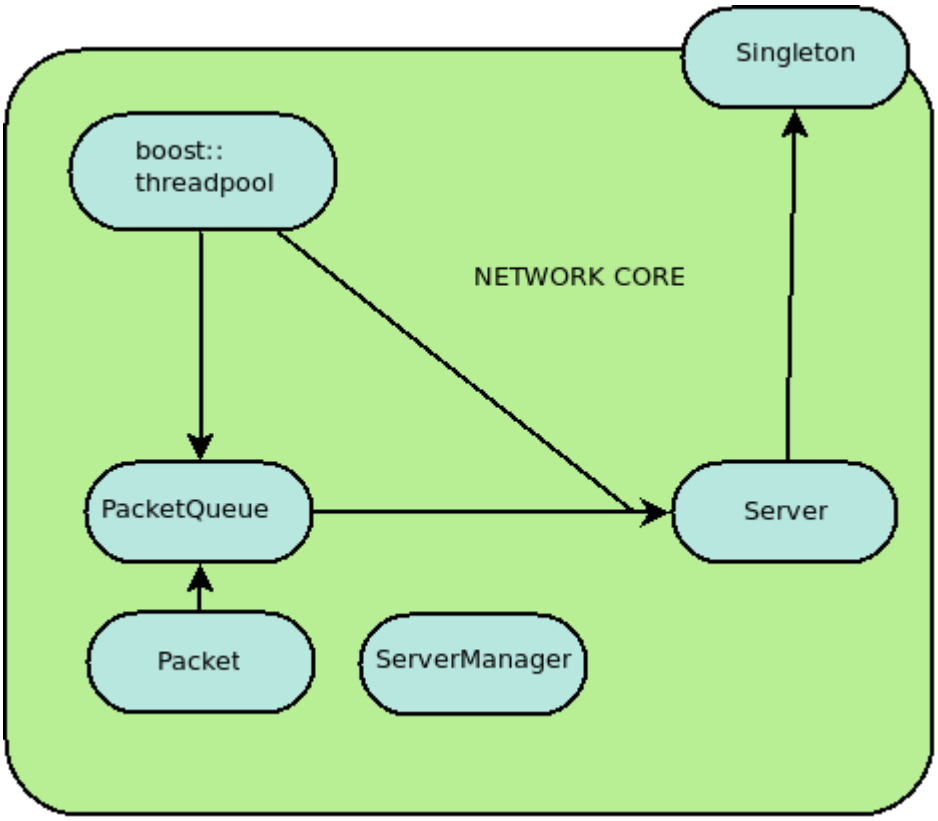

LOGGING
NETWORK CORE
CONFIGURATION
SESSION MANAGER
APPLICATION CORE
USER MODULE
APPLICATION MODULE

I.Modélisation détaillée

1.Les composants

a.Serveur

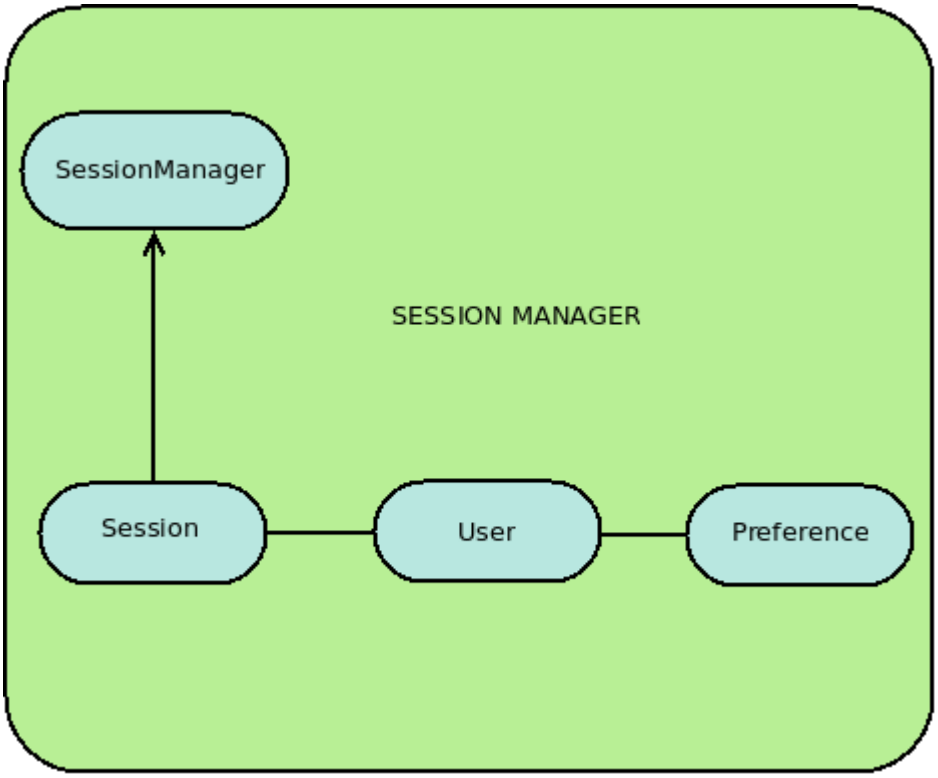
- Le cœur réseau :



Le pattern utilisé est celui du ThreadPool (un thread prioritaire de réception de toutes les données dans une file d'attente et *n*Thread traitant les paquets cette file d'attente)

Les différents composants de l'application ne nécessitant pas tous la même priorité, un mécanisme de priorisation des paquets est possible (audio > userinfo)

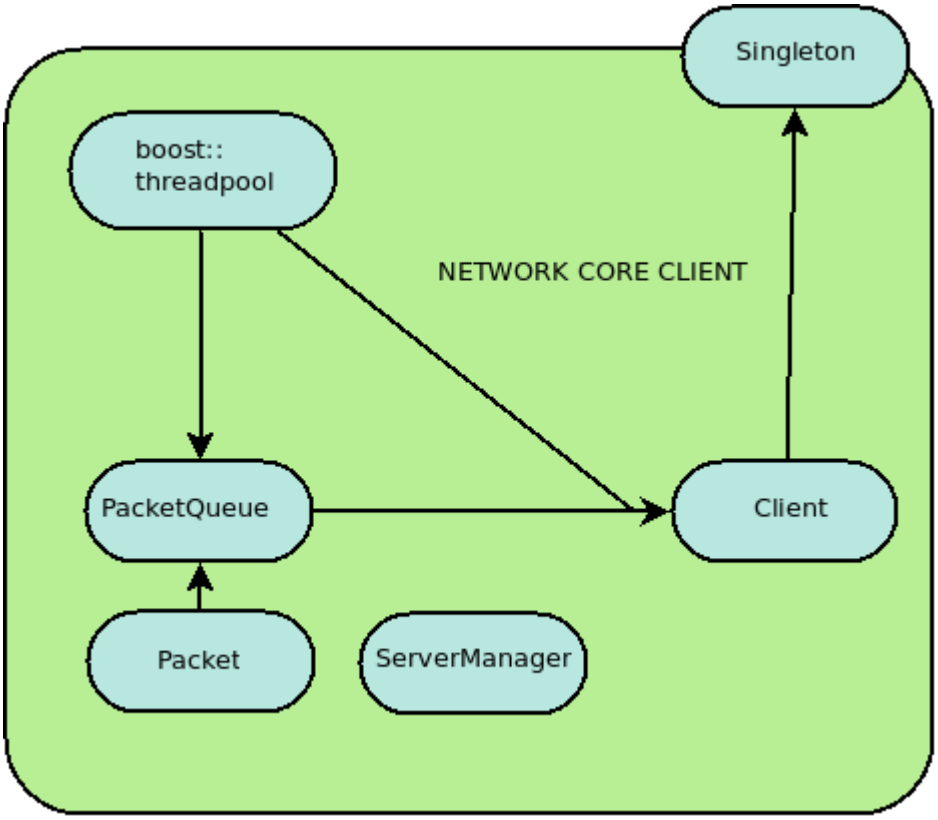
- Session Management :



Ce composant va permettre de gérer ce qui est relatif à l'authentification et la récupération des données utilisateurs. Il gère automatiquement la partie authentification du protocole et validera les paquets authentifiés.

a.Client

- Le cœur réseau :



Le pattern utilisé est celui du ThreadPool (un thread prioritaire de réception de toutes les données dans une file d'attente et nThread traitant les paquets cette file d'attente)

Les différents composants de l'application ne nécessitant pas tous la même priorité, un mécanisme de priorisation des paquets est possible (audio > userinfo)

a.Commun

- Le Logging :

C'est un composant permettant de garder traces des tous les évènements de l'application.

- Configuration :

Composant permettant de lire des fichiers de configuration au format YAML grâce a la librairie externe yaml-cpp.

1.Les modules

a.Serveur

- UserModule:

  C'est un backend de récupération des données utilisateurs permettant entre autre l'authentification

- ApplicationModule :

  Ici ce trouve les fonctionnalités additionnelles non contenu dans le cœur de l'application.

a.Client

Ici ce trouve les fonctionnalités additionnelles non contenu dans le cœur de l'application.

2.Les interfaces

Les interfaces telles que IapplicationModule ou IuserModule representent les contraintes que doivent respecter les modules pour pouvoir être utiliser par l'application.

3.Les librairies externes

a.Boost

Boost rend accessible un ensemble de librairies portable C++.
Licence : Boost Licence (~GPL).
Pour plus d'informations : http://www.boost.org/doc/

b.Boost ::threadpoool

Threadpool est une librairie C++ multiplateformes permettant l'utilisation d'un pool de thread.
Licence : Boost Licence (~GPL).
Pour plus d'informations: http://threadpool.sourceforge.net/reference/annotated.html

c.Fmod

Fmod est une librairie multiplateformes pour une utilisation en temps réél des input/output audio.
Pour plus d'informations : http://www.fmod.org

d.LibVorbis

LibVorbis est une librairie qui permet la manipulation, l'encodage, le decodage du format Ogg/Vorbis.
Pour plus d'informations : http://xiph.org/vorbis/doc/

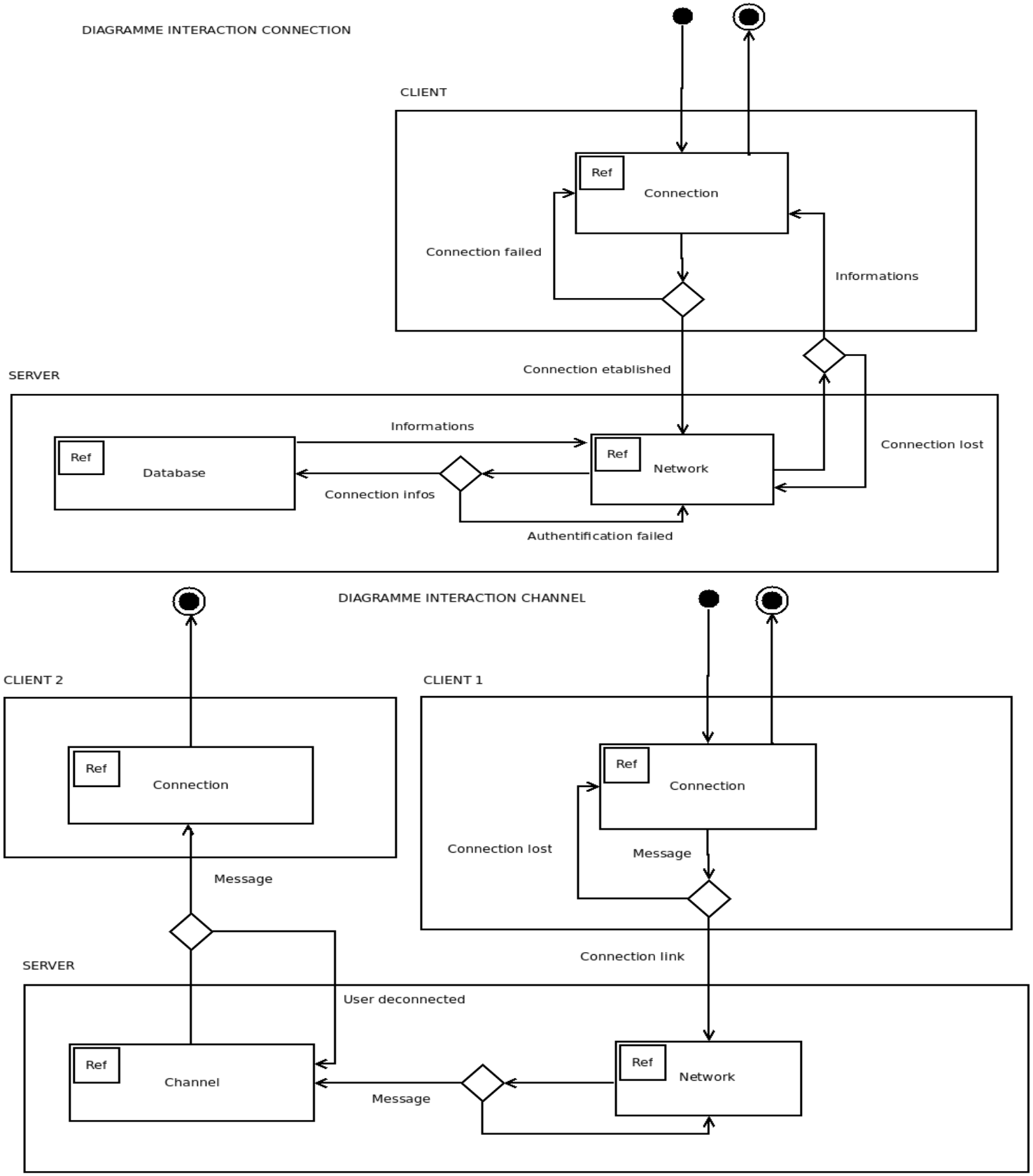
e.Yaml-cpp

Yaml-cpp est un parseur et un manipulateur de ressources au format YAML.
Pour plus d'informations : http://code.google.com/p/yaml-cpp/

f.Mysql++

MySql++ est une version C++ de la librairie MysSql C.
Pour plus d'informations : http://tangentsoft.net/mysql++/doc/

II.Les interactions



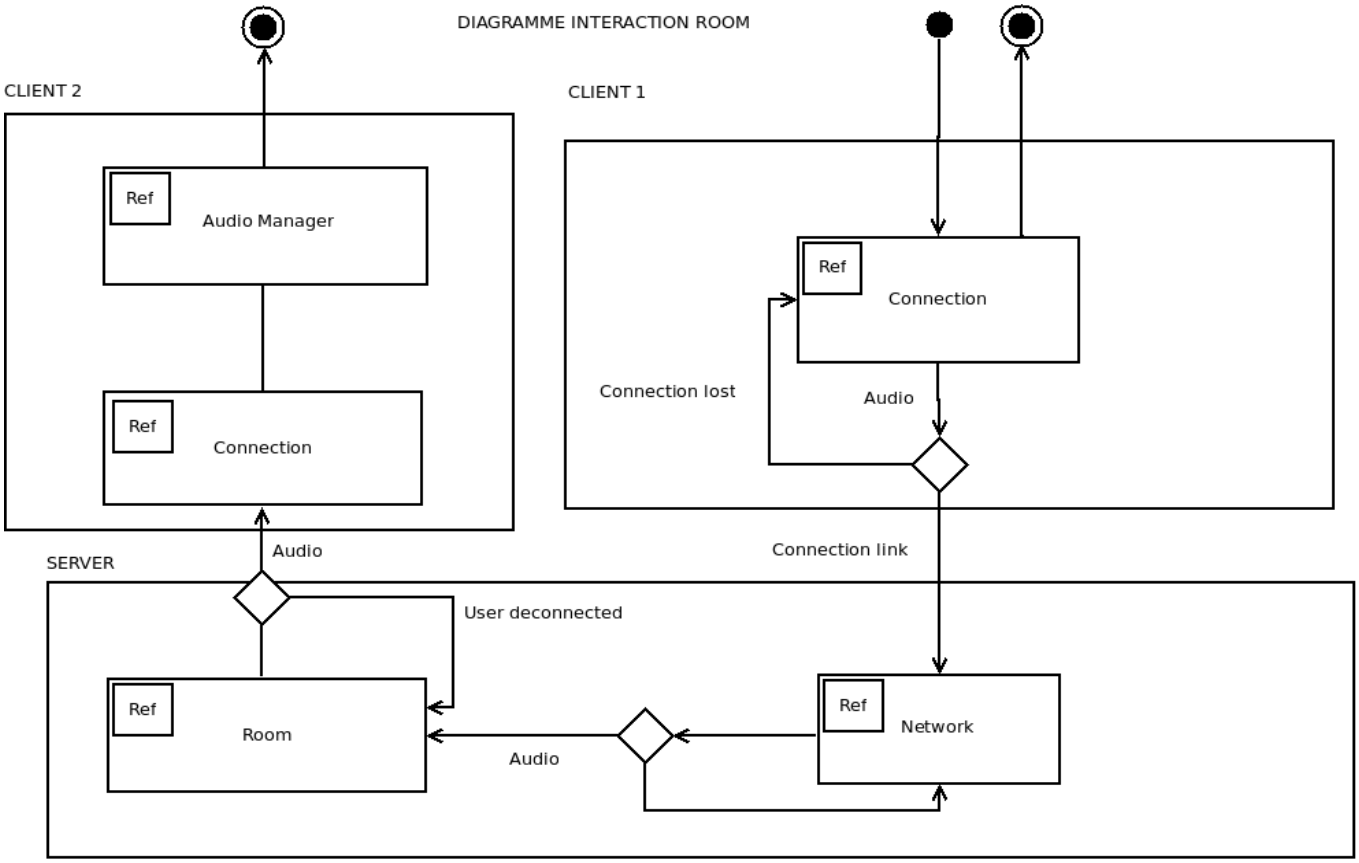


DIAGRAMME INTERACTION ROOM
CLIENT 2
Ref
Audio Manager
Ref
Connection
CLIENT 1
Ref
Connection
Connection lost
Audio
Audio
Connection link
SERVER
User deconnected
Ref
Room
Audio
Ref
Network

III.Le protocole Live-Jamming

UDP packet description :

```
-------------------------------------------- -------------------------------------------------------------------
| proto[4bit] | componentId[10bit] | requestId[6bit] |sessionId[32bit]  | datalen[12] |  |\\|  datas[...] |
-------------------------------------------- -------------------------------------------------------------------
```

Proto id : version 1 1

Components id :
- session 1
- channel 2
- room 3
- jam 4

Connection Steps :

Step 1: Client requests session with server:

```
{
 componentId :          SESSION_COMPONENTID          1

 requestId :            SESSION_AUTHREQUEST          1

 sessionId :            SESSION_ID                   0

 datas :        login
                password
}
```

packet format example:

```
-------------------------------------------------------------------------------------------------------------------- ---------------------------------------------------------------
PROTO | SESSION_COMPONENTID | SESSION_AUTHREQUEST | SESSION_ID | DATALEN |  DATAS =  null_terminated_login,null_terminated_pass |
-------------------------------------------------------------------------------------------------------------------- ---------------------------------------------------------------
```

Step 2: Server informs client that session has been created:

```
{
 componentId :          SESSION_COMPONENTID          1

 requestId :            SESSION_AUTHREQUEST_OK       2

 sessionId :            SESSION_ID                   X

 datas :        user informations
}
```

packet format example:

```
-------------------------------------------------------------------------------------------------------------------- ------------------------------------------------
PROTO | SESSION_COMPONENTID | SESSION_AUTHREQUEST_OK | SESSION_ID | DATALEN |  DATAS =  null_terminated_userinfos |
-------------------------------------------------------------------------------------------------------------------- ------------------------------------------------
```

Step 2 (alt): Server informs client that session has NOT beed created, bad authentification:

```
{
 componentId :          SESSION_COMPONENTID                    1

 requestId :            SESSION_AUTHREQUEST_NOK_BADAUTH        3
```

sessionId : SESSION_ID 0
}

packet format example:

PROTO	SESSION_COMPONENTID	SESSION_AUTHREQUEST_NOK_BADAUTH	SESSION_ID	DATALEN	DATAS = nul l

Step 2 (alt): Server informs client that session has NOT beed created, authentification duplicate:
{
componentId : SESSION_COMPONENTID 1

requestId : SESSION_AUTHREQUEST_NOK_DUPLICATE 3

sessionId : SESSION_ID 0
}

packet format example:

PROTO	SESSION_COMPONENTID	SESSION_AUTHREQUEST_NOK_DUPLICATE	SESSION_ID	DATALEN	DATAS = nul l

Step 3 : Client disconnects from server :
{
componentId : SESSION_COMPONENTID 1

requestId : SESSION_DISCONNECT 5

sessionId : SESSION_ID 0
}

packet format example:

PROTO	SESSION_COMPONENTID	SESSION_DISCONNECT	SESSION_ID	DATALEN	DATAS = nul l

Step 3 : Server disconnects from client:
{
componentId : SESSION_COMPONENTID 1

requestId : SESSION_DISCONNECTED 6

sessionId : SESSION_ID 0
}

packet format example:

PROTO	SESSION_COMPONENTID	SESSION_DISCONNECTED	SESSION_ID	DATALEN	DATAS = nul l

Channel / Message Steps :

CHANNEL_ID = 16bit
CLIENT_SESSION_ID = 32bit

User Join Channel :
{

componentId :	CHANNEL_COMPONENTID	2
requestId :	CHANNEL_JOIN	1
sessionId :	SESSION_ID	X
datas :	CHANNEL_ID	X

}

packet format example:

PROTO	CHANNEL_COMPONENTID	CHANNEL_JOIN	SESSION_ID	DATALEN	DATAS = CHANNEL_ID

User receive joined notification of a user in the Channel :
{

componentId :	CHANNEL_COMPONENTID	2
requestId :	CHANNEL_JOINED	4
sessionId :	SESSION_ID	X
datas :	CHANNEL_ID	X
	CLIENT_SESSION_ID	X

}

packet format example:

PROTO	CHANNEL_COMPONENTID	CHANNEL_LEAVED	SESSION_ID	DATALEN	DATAS = CHANNEL_ID

User leave Channel :
{

componentId :	CHANNEL_COMPONENTID	2
requestId :	CHANNEL_LEAVE	10
sessionId :	SESSION_ID	X
datas :	CHANNEL_ID	X

}

packet format example:

PROTO	CHANNEL_COMPONENTID	CHANNEL_JOIN	SESSION_ID	DATALEN	DATAS = CHANNEL_ID

User receive leaved notification of a user in the Channel :

{

componentId : CHANNEL_COMPONENTID 2

requestId : CHANNEL_LEAVED 13

sessionId : SESSION_ID X

datas : CHANNEL_ID X
CLIENT_SESSION_ID X

}

packet format example:

PROTO | CHANNEL_COMPONENTID | CHANNEL_LEAVED | SESSION_ID | DATALEN | DATAS = CHANNEL_ID |

Send channel message :

{

componentId : CHANNEL_COMPONENTID 2

requestId : CHANNEL_MESSAGE 6

sessionId : SESSION_ID X

datas : CHANNEL_ID X
MESSAGE X

}

packet format example:

PROTO | CHANNEL_COMPONENTID | CHANNEL_MESSAGE | SESSION_ID | DATALEN | DATAS = CHANNEL_ID / MESSAGE |

Receive channel message :

{

componentId : CHANNEL_COMPONENTID 2

requestId : CHANNEL_MESSAGE_RECV 8

sessionId : SESSION_ID X

datas : CHANNEL_ID X
CLIENT_SESSION_ID X
MESSAGE X

}

packet format example:

PROTO | CHANNEL_COMPONENTID | CHANNEL_MESSAGE_RECV | SESSION_ID | DATALEN | DATAS = CHANNEL_ID / CLIENT_SESSION_ID / MESSAGE |

<<<<<STEPS TO IMPLEMENT NOT DECIDED YET>>>>>

User informations Steps :

Change status :
{
sessionid : sessionid

type : informations : status_changed

datas : status

version : proto_version
}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

Get user profil :
{
sessionid : sessionid

type : informations : get_profil

datas : user

version : proto_version
}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

Evaluate latency :
{
sessionid : sessionid

type : informations : evaluate_latency

datas : latency

version : proto_version
}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

Room Steps :

Create room :
{
sessionid : sessionid

type : jam : create

datas : room_name
room_settings
room_participants

version : proto_version
}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

Leave room :
{
sessionid : sessionid

type : jam : end

datas : room_name
room_participants

version : proto_version
}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

Send invitation to room :
{
sessionid : sessionid

type : jam : invite

datas : room_name
room_invited
room_invitation_message

version : proto_version

}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

Receive invitation to a room :
{
sessionid : sessionid

type : jam : invite

datas : room_name
room_host
room_invitation_message

version : proto_version
}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

Send kick from room :
{
sessionid : sessionid

type : jam : kick

datas : room_name
room_kicked
room_kick_reason

version : proto_version
}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

Receive kick from a room :
{
sessionid : sessionid

type : jam : kick

datas : room_name
room_host
room_kick_reason

version : proto_version
}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

Room settings :
{
sessionid : sessionid

type : jam : settings

datas : room_name
room_settings

version : proto_version
}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

Jam Steps :

Start jam :
{
sessionid : sessionid

type : jam : start

datas : room_name

version : proto_version
}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

Record jam :
{
sessionid : sessionid

type : jam : record

datas : room_name

version : proto_version
}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

Stop jam :
{
sessionid : sessionid

type : jam : stop

datas : room_name

version : proto_version
}

packet format example:

---
sessionid | type = auth_request | datalen = ... | proto_version = ...| [datas] = null_terminated_login,null_terminated_pass |

IV.Le site internet/La base de données

1.Le site internet

Le site internet est réalisé à l'aide du Framework CakePhp (http://cakephp.org/) et est disponible à l'adresse http://www.live-jamming.com .

2.La base de données