Messages

University of Orléans

7 octobre 2013

Presentation

- Class of messages
- Format of messages
- Message class
- Messages track

Class of messages

Six class

There is six different class of messages that may be throw :

Example of utilisation

Class of messages

Six class

There is six different class of messages that may be throw :

- CS : client service message.

Example of utilisation

Class of messages

Six class

There is six different class of messages that may be throw :

- CS : client service message.
- CG : client game message.

Example of utilisation

Class of messages

Six class

There is six different class of messages that may be throw :

- CS : client service message.
- CG : client game message.
- SS : server service message.

Example of utilisation

Class of messages

Six class

There is six different class of messages that may be throw :

- CS : client service message.
- CG : client game message.
- SS : server service message.
- SG : server game message.

Example of utilisation

Class of messages

Six class

There is six different class of messages that may be throw :

- CS : client service message.
- CG : client game message.
- SS : server service message.
- SG : server game message.
- SP : server problem message.

Example of utilisation

Class of messages

Six class

There is six different class of messages that may be throw :

- CS : client service message.
- CG : client game message.
- SS : server service message.
- SG : server game message.
- SP : server problem message.
- SR : server recall message.

Example of utilisation

- MessageCS.connect() : message to request a connection.

Class of messages

Six class

There is six different class of messages that may be throw :

- CS : client service message.
- CG : client game message.
- SS : server service message.
- SG : server game message.
- SP : server problem message.
- SR : server recall message.

Example of utilisation

- MessageCS.connect() : message to request a connection.
- MessageSR.types : messages with all the types of the player.

Format of messages

Same for all

A message is represented by a string. All messages can be represented by four properties.

- ip : the ip of the client or server
- port : the port of the client or server
- type : the type of message
- parameters : the message datas.

A separator in top

To represent a message we use an auto generated separator. The final format is as follow :
SEPA+Ip+SEPA+port+SEPA+type+SEPA+Parameters+SEPA.

Format of messages

Parameters : different for all

The parameter format must be generic because there is message with no limit of parameters or even no parameters.

A separator inside another

To represent the parameters we will then uses another auto-generate and different separator. The final format is as follow :
SEPA+Param1+SEPA+Param2+SEPA+...+ParamN+SEPA.

Example of message

Main : SEP165.165.165.165SEP4444SEPsg_chatSEPParametersSEP
Parameters : SEP2playernameSEP2messageSEP2

Message class

The main : Message

This class contains the methods to build message. The important one are :

And the one that will be modified when a new message is created :

Note

Message class

The main : Message

This class contains the methods to build message. The important one are :

- new_separator : generate a new separator

And the one that will be modified when a new message is created :

Note

Message class

The main : Message

This class contains the methods to build message. The important one are :

- new_separator : generate a new separator
- getParam : return a parameter in a type with a limit of parameters.

And the one that will be modified when a new message is created :

Note

Message class

The main : Message

This class contains the methods to build message. The important one are :

- new_separator : generate a new separator
- getParam : return a parameter in a type with a limit of parameters.
- getListeParam : return a list in a type with no limit of parameters.

And the one that will be modified when a new message is created :

Note

Message class

The main : Message

This class contains the methods to build message. The important one are :

- new_separator : generate a new separator
- getParam : return a parameter in a type with a limit of parameters.
- getListeParam : return a list in a type with no limit of parameters.
- isMessage : return true if a message respect the format.

And the one that will be modified when a new message is created :

Note

Message class

The main : Message

This class contains the methods to build message. The important one are :

- new_separator : generate a new separator
- getParam : return a parameter in a type with a limit of parameters.
- getListeParam : return a list in a type with no limit of parameters.
- isMessage : return true if a message respect the format.

And the one that will be modified when a new message is created :

- isTypeListParam : return true for no limit params message.

Note

Message class

The main : Message

This class contains the methods to build message. The important one are :

- new_separator : generate a new separator
- getParam : return a parameter in a type with a limit of parameters.
- getListeParam : return a list in a type with no limit of parameters.
- isMessage : return true if a message respect the format.

And the one that will be modified when a new message is created :

- isTypeListParam : return true for no limit params message.
- isNotTypeListParam : the opposite.

Note

Message class

The main : Message

This class contains the methods to build message. The important one are :

- new_separator : generate a new separator
- getParam : return a parameter in a type with a limit of parameters.
- getListeParam : return a list in a type with no limit of parameters.
- isMessage : return true if a message respect the format.

And the one that will be modified when a new message is created :

- isTypeListParam : return true for no limit params message.
- isNotTypeListParam : the opposite.
- getTailleTypeMessage : return the number of params.

Note

Message class

The main : Message

This class contains the methods to build message. The important one are :

- new_separator : generate a new separator
- getParam : return a parameter in a type with a limit of parameters.
- getListeParam : return a list in a type with no limit of parameters.
- isMessage : return true if a message respect the format.

And the one that will be modified when a new message is created :

- isTypeListParam : return true for no limit params message.
- isNotTypeListParam : the opposite.
- getTailleTypeMessage : return the number of params.

Note

- We use static method and not object because we manage string for the messages.

Message class

MessageType for each type

For each of the six types(SG...etc) there is a class. Each type of message will be represent by :

Message class

MessageType for each type

For each of the six types(SG...etc) there is a class. Each type of message will be represent by :

- A static and public name starting by SG, SS...etc
- A static and public integer which specifies the number of parameters.
- A static and public integer for each parameter
- A static and public method

Message class

MessageType for each type

For each of the six types(SG...etc) there is a class. Each type of message will be represent by :

- A static and public name starting by SG, SS...etc
- Example : SG_CHAT_NAME="sg_chat"
- A static and public integer which specifies the number of parameters.
- A static and public integer for each parameter
- A static and public method

Message class

MessageType for each type

For each of the six types(SG...etc) there is a class. Each type of message will be represent by :

- A static and public name starting by SG, SS...etc
- Example : SG_CHAT_NAME="sg_chat"
- A static and public integer which specifies the number of parameters.
- Example : SG_CHAT_NUM_PARAMS=4;
- A static and public integer for each parameter
- A static and public method

Message class

MessageType for each type

For each of the six types(SG...etc) there is a class. Each type of message will be represent by :

- A static and public name starting by SG, SS...etc
- Example : SG_CHAT_NAME="sg_chat"
- A static and public integer which specifies the number of parameters.
- Example : SG_CHAT_NUM_PARAMS=4;
- A static and public integer for each parameter
- Example : SG_CHAT_MESSAGE = 4;
- A static and public method

Message class

MessageType for each type

For each of the six types(SG...etc) there is a class. Each type of message will be represent by :

- A static and public name starting by SG, SS...etc
- Example : SG_CHAT_NAME="sg_chat"
- A static and public integer which specifies the number of parameters.
- Example : SG_CHAT_NUM_PARAMS=4;
- A static and public integer for each parameter
- Example : SG_CHAT_MESSAGE = 4;
- A static and public method
- Example : When call :MessageSG.chat(...)

Message class

Two interfaces

The problem with this generic messages is that they will be many messages to handle. So to simplify and not forget the method two interface exists :

- onMessageReceived : contains one method for each message type with no parameter.
- setMessageSend : contains one method for each message with the parameters of the type of message.

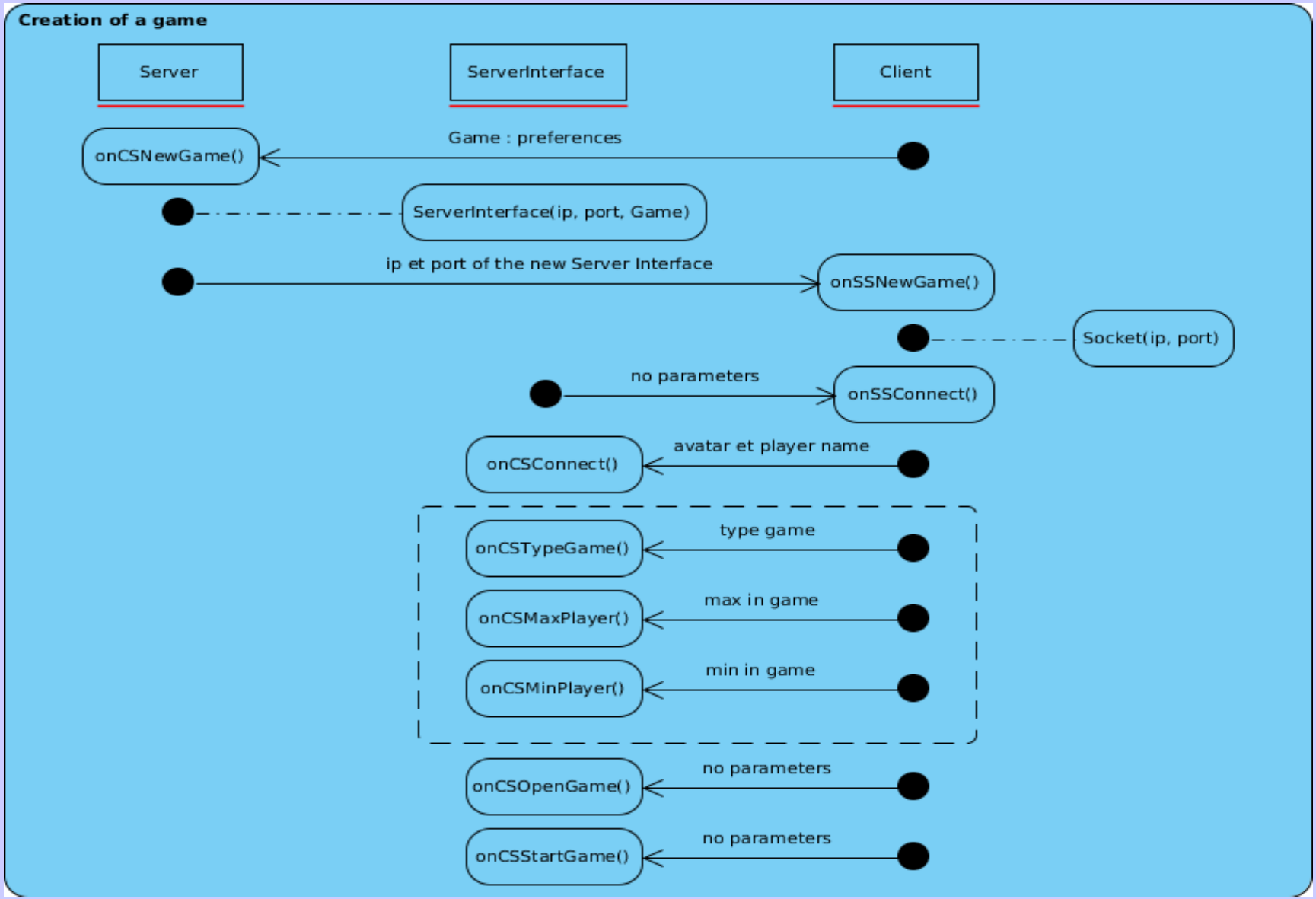
MessageManager : A usefull class

A MessageManager will implements the two interface.
To manage messages we build a class wich extends it and will :

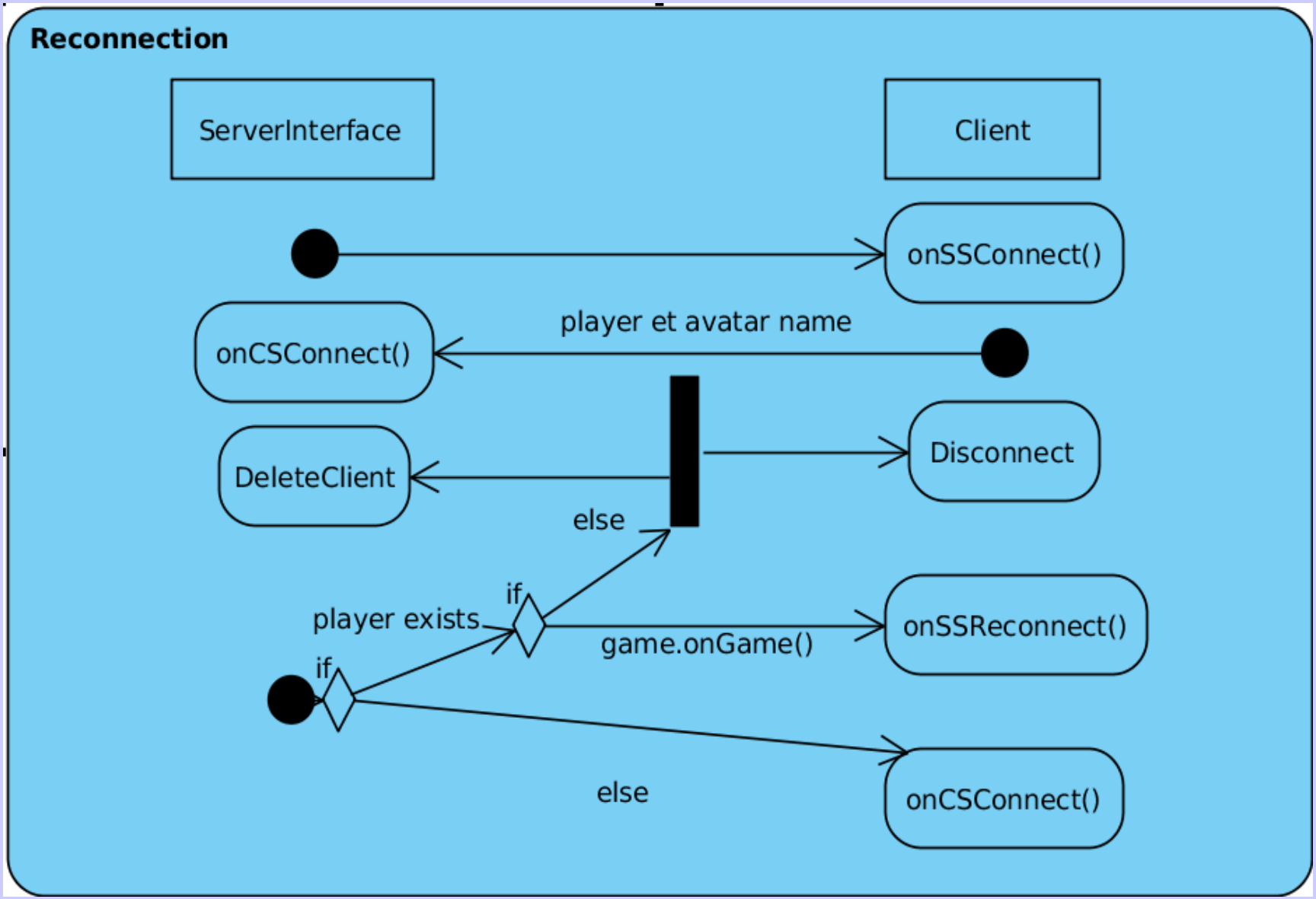
- implement a setMethod to send message.
- implement a onMethod to manage the receive messages.

Messages Track !

CS and SSNewGame Message



Reconnection first part



Reconnection second part