Distributed Component Programming Lab

Sylvain Vauttier
IMT Mines d'Alès

Topic

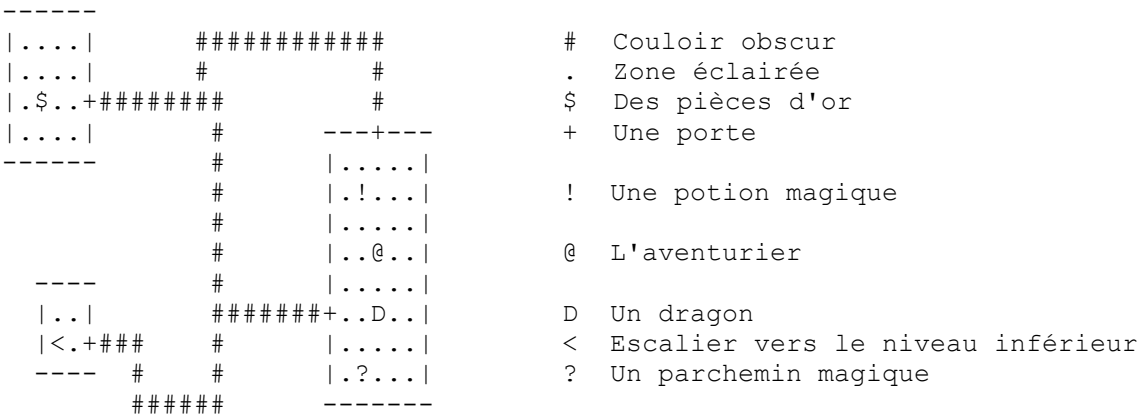
The aim of this project is to develop a "chat", i.e. a client/server software application that enables people to dialogue on internet, following several conceptual architectures.

This chat is composed of a server and a client application that respectively enable to:

- handle client connections, manage a list of user nicknames, handle to receive and forward messages from and to client applications,…
- get a connection on the server, post messages on the server, retrieve messages from the server,…

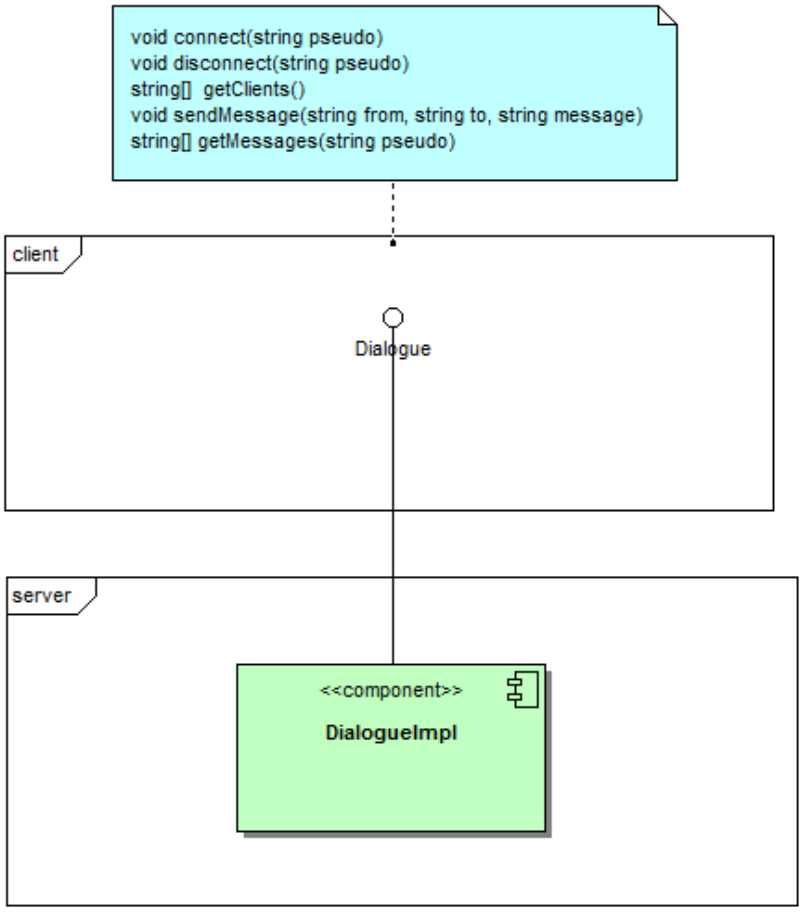
Extensions

- Transform the chat into a MUD (multi-user dungeon, see https://en.wikipedia.org/wiki/Roguelike and https://en.wikipedia.org/wiki/MUD). Hint : Transform first the chat into a MAS (multi-agent system, see https://en.wikipedia.org/wiki/Multi-agent_system)



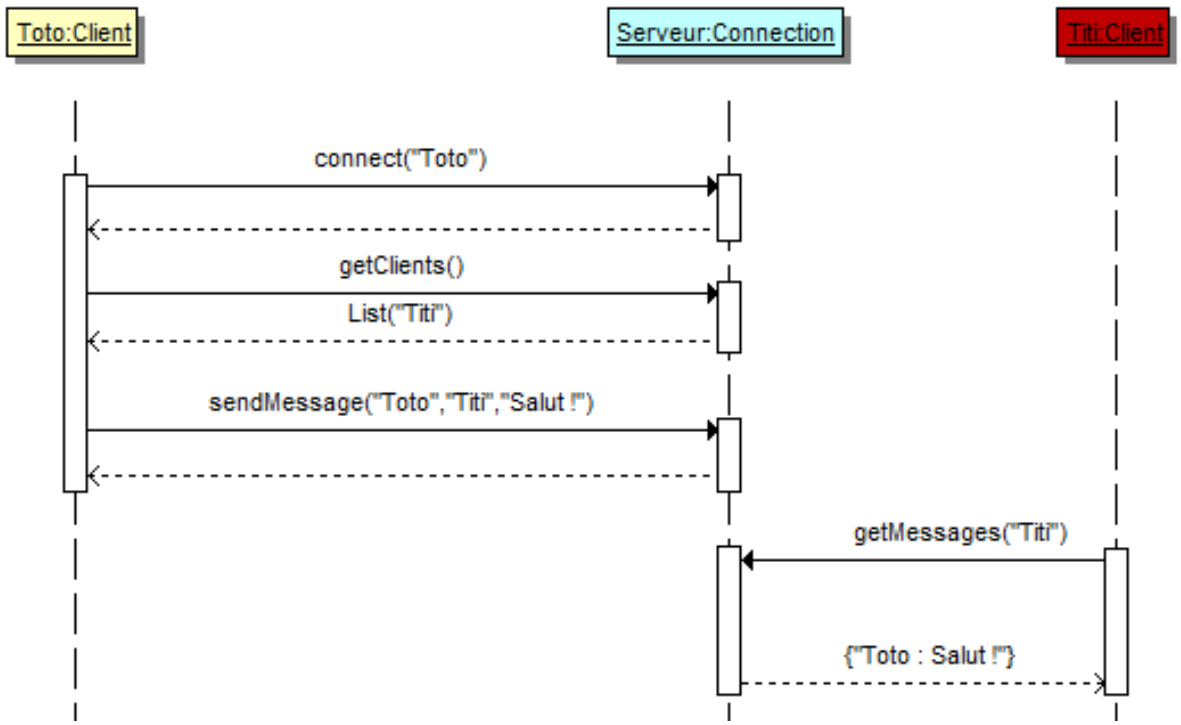
Architectures

Basic pull architecture

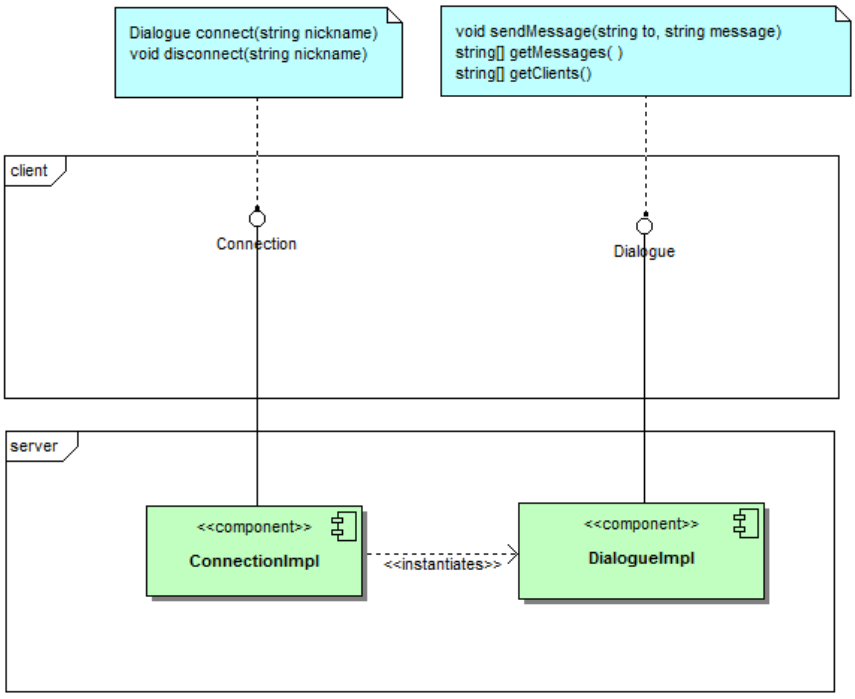


Client queries the server to retrieve its messages (possibly in a cyclic way using a thread).

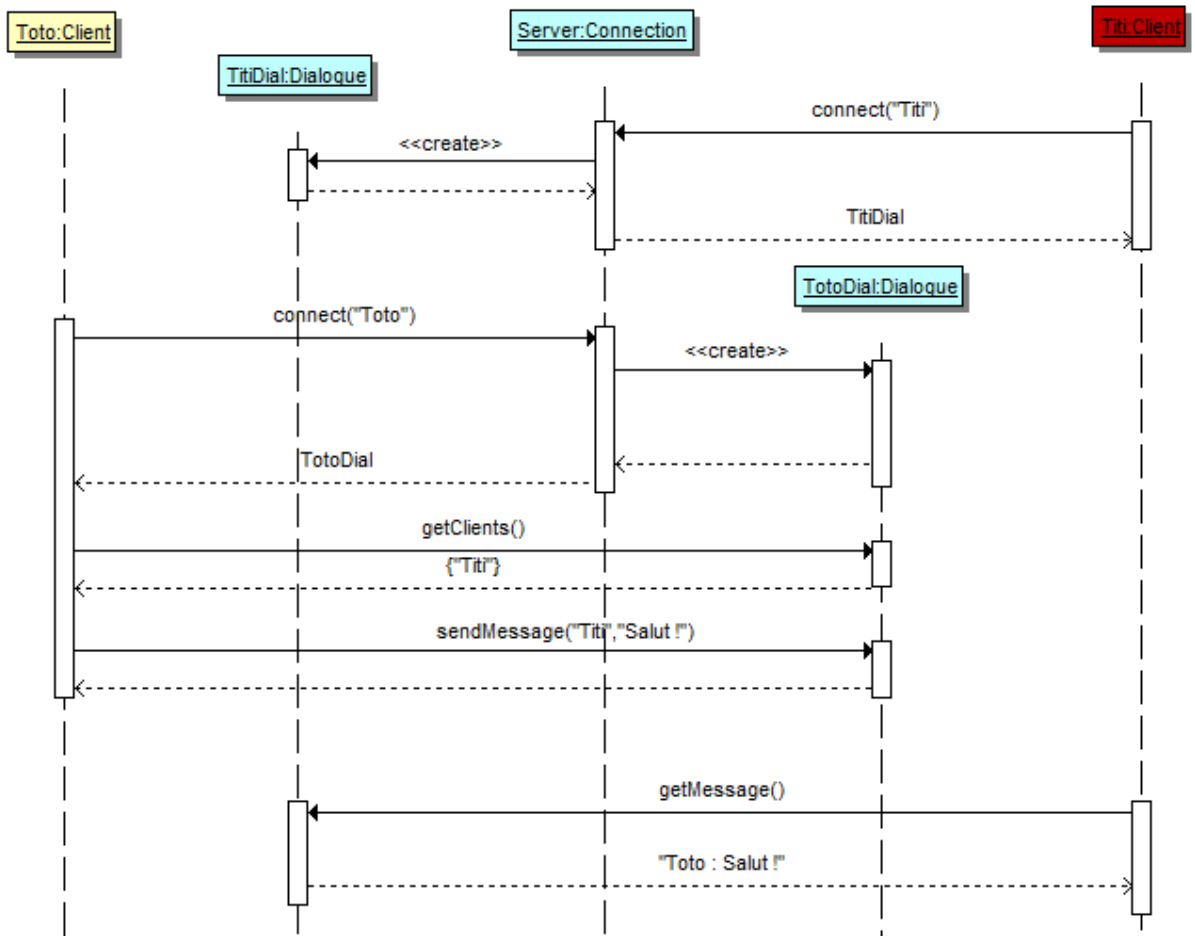
Extension: provided some signature changes, design a mechanism for a more secure identification.



Advanced pull architecture



Server app implements a connection component. This component acts as a factory for dialog components. A dialog component is created for each connected client that is then implicitly identified by this component.

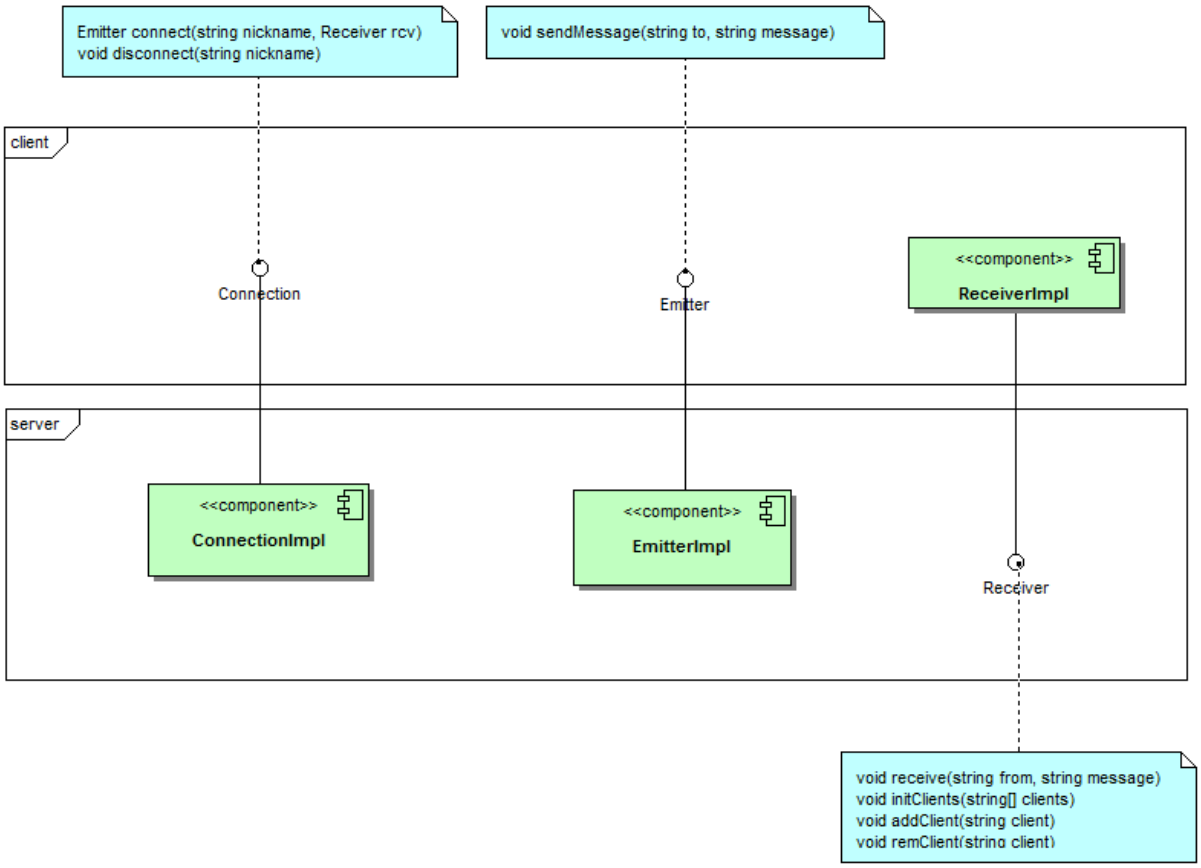


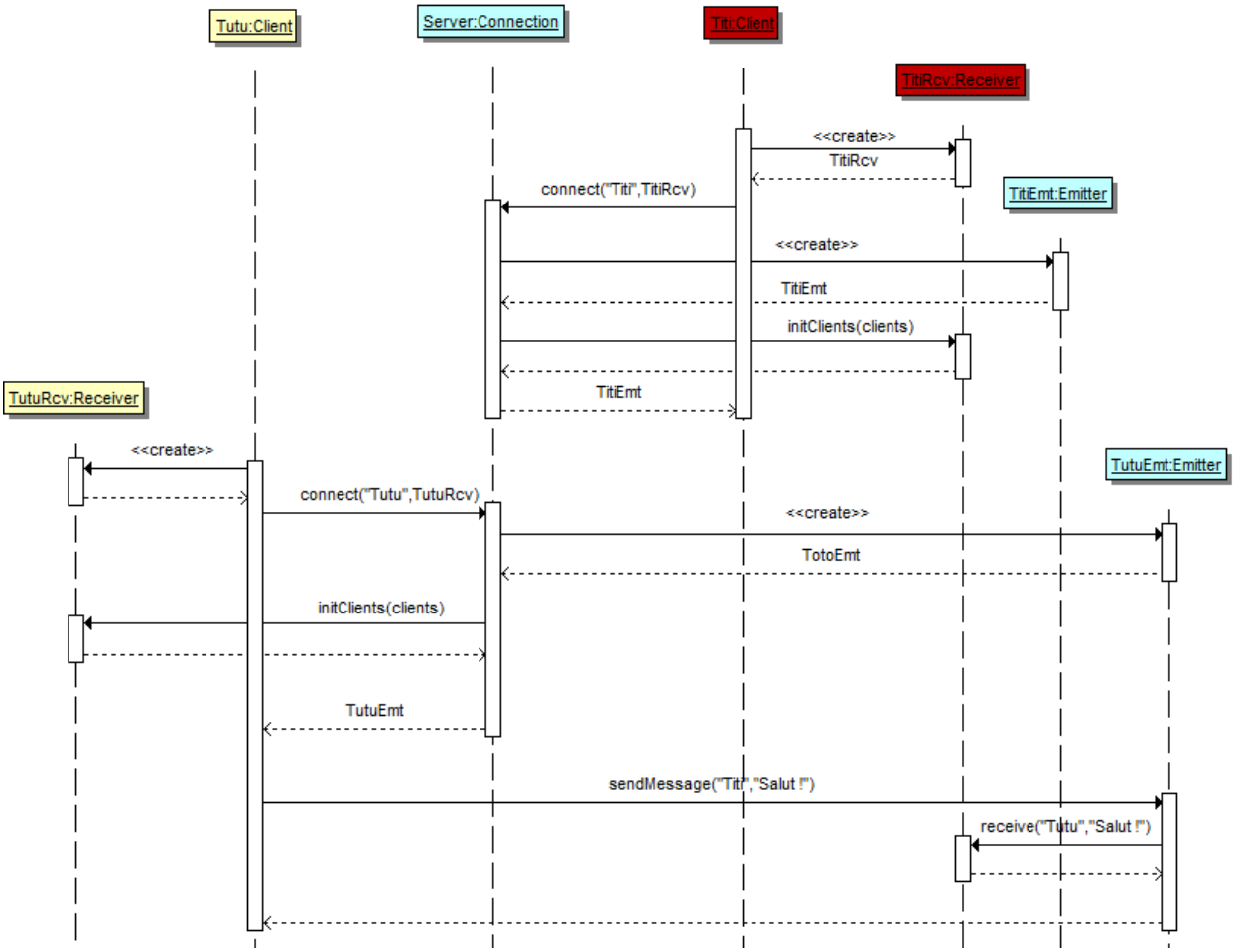
Push architecture

Client apps own a reception component (*ReceiverImpl*) that enable server app to send them immediately received messages (instant messaging).

Reception components also provide functions that enable server app to push to clients the list of connected clients (*initClients function*) then to update this list (*addClient* et *remClient functions*).

Reception component references are provided to server app thanks to an input parameter (*Receiver*) of the *connect* function of the server's connection component. This function returns a reference of the *Emitter* component that is instantiated to be used by the client to send messages to the server (and that incidentally represents the client app on the server).


Emitter connect(string nickname, Receiver rcv)
void disconnect(string nickname)
void sendMessage(string to, string message)
client
Connection
Emitter
<<component>>
ReceiverImpl
server
<<component>>
ConnectionImpl
<<component>>
EmitterImpl
Receiver
void receive(string from, string message)
void initClients(string[] clients)
void addClient(string client)
void remClient(string client)


Tutu:Client
Server:Connection
Titi:Client
TitiRcv:Receiver
<<create>>
TitiRcv
connect("Titi",TitiRcv)
TitiEmt:Emitter
<<create>>
TitiEmt
initClients(clients)
TitiEmt
TutuRcv:Receiver
<<create>>
TutuEmt:Emitter
connect("Tutu",TutuRcv)
<<create>>
TotoEmt
initClients(clients)
TutuEmt
sendMessage("Titi","Salut !")
receive("Tutu","Salut !")

Peer to peer architectures

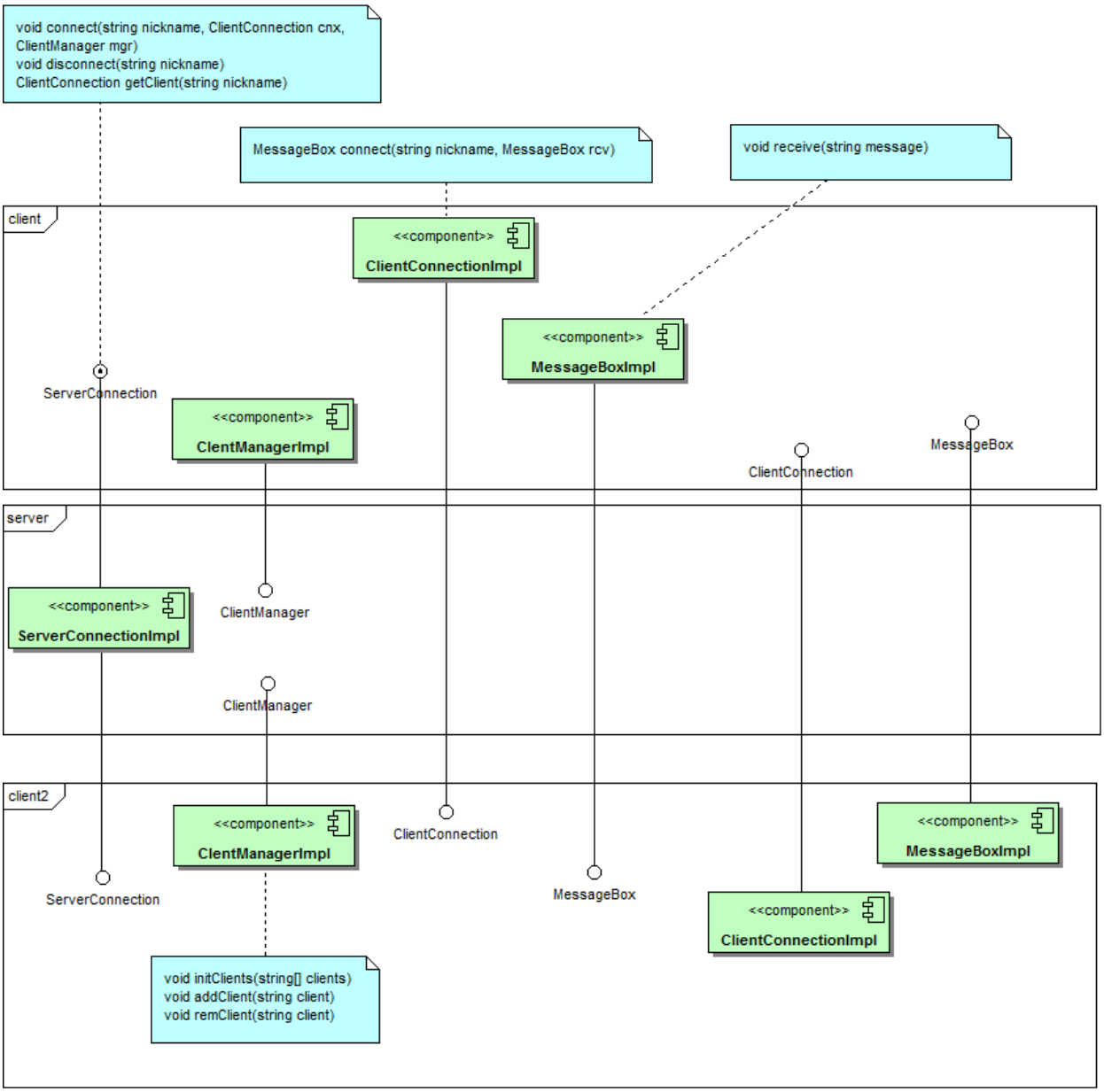
Client apps implement a direct connection component with other clients (*ClientConnection*). They provide the reference of this component to the server in order to build a directory of online clients (*connect* function of the *ServerConnection* component).

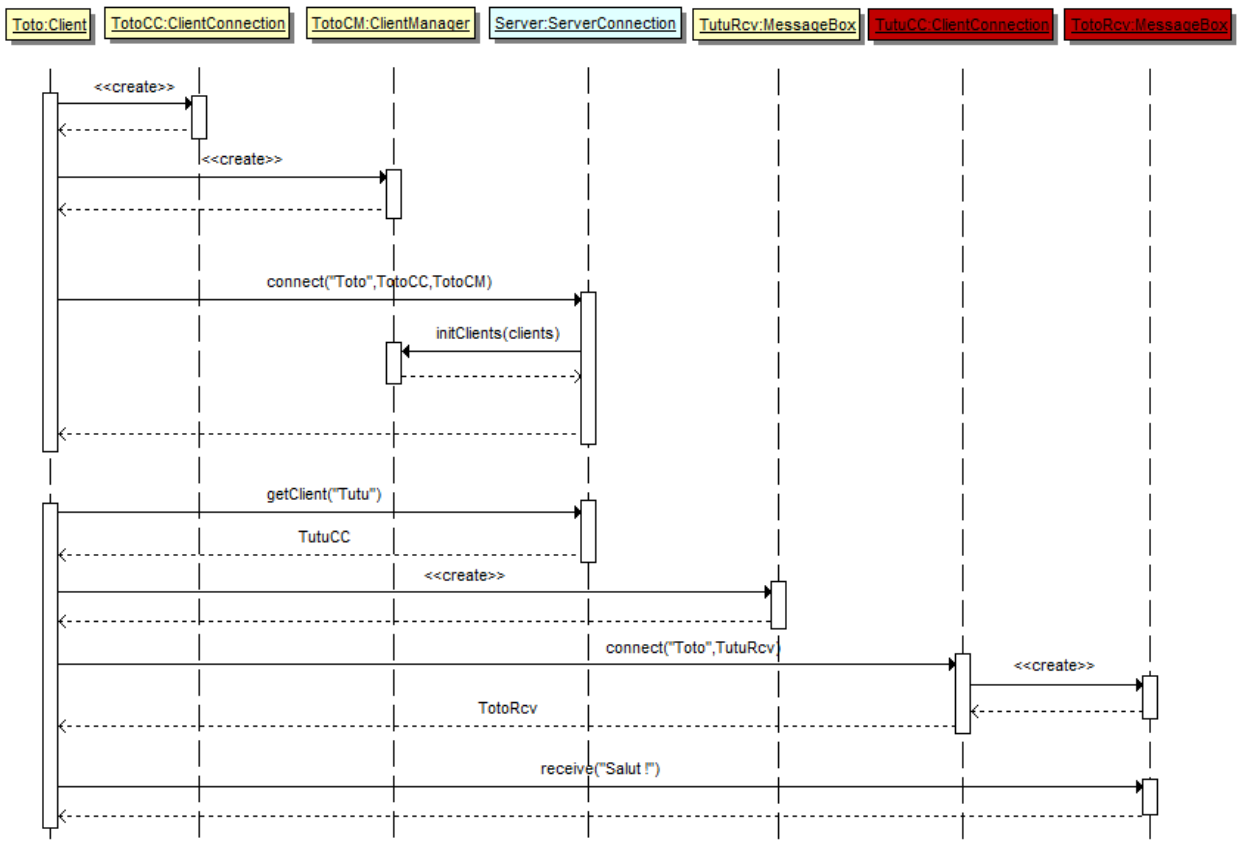
Client apps also implement a component that enable server to update a list of online clients (ClientManager). They provide the reference of this component to the server in order to receive this information (*connect* function of the *ServerConnection* component).

A client can then retrieve the reference of the connection component to another client from the server (*getClient* function of the *ServerConnection* component returning a *ClientConnection* reference).

ClientConnection components act as factories for MessageBox components. Every connection entails the instantiation of a MessageBox component that enables to receive messages from this specific connected client (*connect* function of the *ClientConnection* component).

The connecting client reciprocally instantiates a *MessageBox* component that is dedicated to the reception of messages sent to itself by the other client. A reference to this *MessageBox* component is provided to the other component as an input parameter of the *connect* function of the *ClientConnection* component.


void connect(string nickname, ClientConnection cnx,
ClientManager mgr)
void disconnect(string nickname)
ClientConnection getClient(string nickname)
MessageBox connect(string nickname, MessageBox rcv)
void receive(string message)
client
<<component>>
ClientConnectionImpl
<<component>>
MessageBoxImpl
ServerConnection
<<component>>
ClientManagerImpl
ClientConnection
MessageBox
server
<<component>>
ServerConnectionImpl
ClientManager
ClientManager
client2
<<component>>
ClientManagerImpl
ClientConnection
<<component>>
MessageBoxImpl
ServerConnection
MessageBox
<<component>>
ClientConnectionImpl
void initClients(string[] clients)
void addClient(string client)
void remClient(string client)


Toto:Client
TotoCC:ClientConnection
TotoCM:ClientManager
Server:ServerConnection
TutuRcv:MessageBox
TutuCC:ClientConnection
TotoRcv:MessageBox
<<create>>
<<create>>
connect("Toto",TotoCC,TotoCM)
initClients(clients)
getClient("Tutu")
TutuCC
<<create>>
connect("Toto",TutuRcv)
<<create>>
TotoRcv
receive("Salut !")