Babel Protocol

1. Introduction

1.1 Quick Summary:

The Babel protocol is a binary protocol.
The client-server communication uses TCP sockets.
The peer-to-peer (client to client) communications uses:
- a TCP connection to transmit text messages, and to establish a UDP connection
- a UDP connection to transmit media data (sound)

1.2 Full Summary:

Note: please refer to the sequence diagrams corresponding to the processes described in this summary when necessary.

The clients connect to the Babel server (using TCP sockets).
He can then sign up (create an account) or log in.

The server manages a database containing the users data (credentials and friends list).

Once the client has logged in, the server sends it the list of its friends and their status (online, offline, pending friend request).
During the session, the client is notified by the server when one of its friend connects/disconnects.
The client can add or remove friends (its friend list cannot exceed 100 friends).

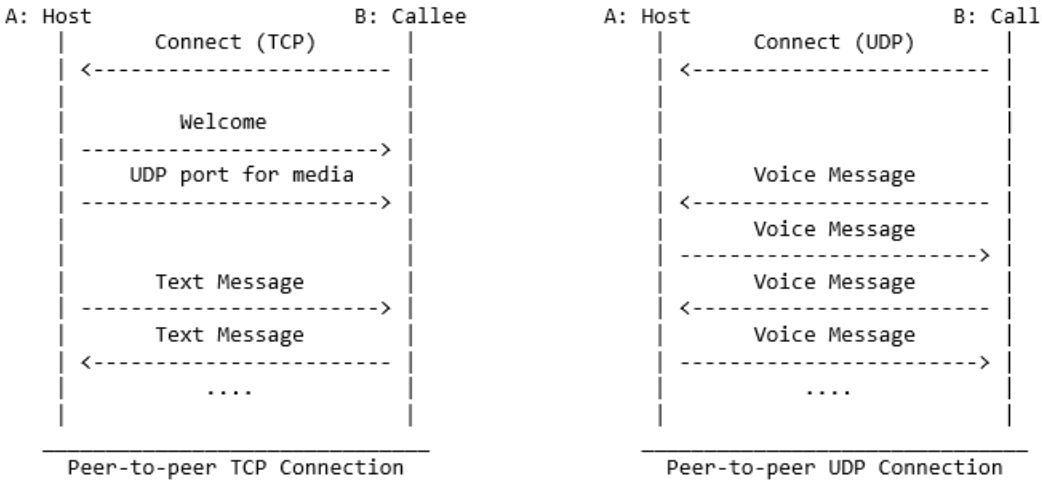
When a client wants to invite a friend to a call, the server is used as a proxy to initiate a peer-to-peer communication:
In the following description of the process, the client initiating the call will be called the CALLER, the other party will be called the CALLEE.
Arrows (->) with the same level of indentation are alternatives.

AddParticipantToCall process:

```
-> The client emits a call request to the server.
    -> The server relays the call request to the concerned party.
        -> The callee declines the call.
        |   -> The caller is notified
        |       [END of the communication]
        -> The callee accepts the call
            -> Then the server provides the ip of the callee to the caller, and asks it to provide a port
               for the callee to connect to.
                -> The server provides the callee the ip and port of the caller
                    [END of the communication]
```

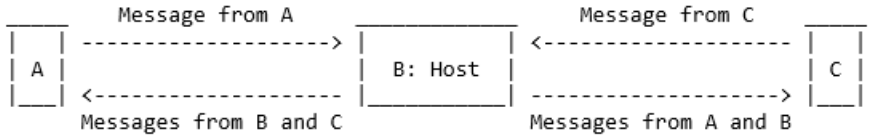
A TCP connection is then established directly between the caller and the callee.
This connection has three purposes:
- to provide the callee with the list of the participants
- to provide the callee with a port to establish a UDP connection for media data transfert (voice messages).
- to allow the callee and the caller to exchange text messages

Finally, the UDP connection is also established between the caller and the callee.
The caller and callee can exchange text message (on the TCP connection) and voice messages (on the UDP connection).



On a conference call, the caller is actually the host of the whole conference call.
Each callee transmit messages (text using TCP and voice using UDP) to (and only to) the host.
The host then broadcast everything to everyone.

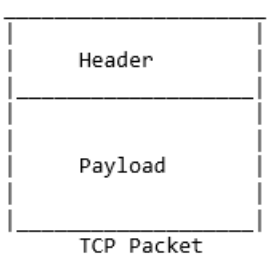
Conference call between A, B (the host) and C:



2. Protocol: Data

2.1 TCP Packet:

The TCP packets are composed of a header and a payload.



The header of a TCP Packet is defined as the following struct:

```
struct      TCPPacketHeader
{
    char    magicA;
    char    magicB;
    char    magicC;
    char    commandId;
    int     payloadSize;
}
```

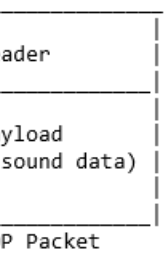
Description of the fields of the TCPPacketHeader struct:
- magicA: it is used as a magic number to identify a Babel packet (MAGIC_A = 1)
- magicB: it is used as a magic number to identify a Babel packet (MAGIC_B = 1)
- magicC: it is used as a magic number to identify a Babel packet (MAGIC_C = 7)
- commandId: it is used to identify the command described by the packet

- payloadSize: it is used to know the size of the payload

The total size of the packet is easily deduced:
packetSize = sizeof(TCPPacketHeader) + payloadSize
The payload contains the data necessary for the command specified by commandId.

2.2 UDP Packet:

The UDP packets are composed of a header and a payload.



UDP Packet

The header of a UDP Packet is defined as the following struct:

```
struct      UDPPacketHeader
{
    char    magicA;
    char    magicB;
    char    magicC;
    char    senderID;
    int     payloadSize;
}
```

Description of the fields of the UDPPacketHeader struct:
- magicA: it is used as a magic number to identify a Babel packet (MAGIC_A = 1)
- magicB: it is used as a magic number to identify a Babel packet (MAGIC_B = 1)
- magicC: it is used as a magic number to identify a Babel packet (MAGIC_C = 7)
- senderID: it is used to identify the user from whom the data originated
- payloadSize: it is used to specify the size of the payload

The total size of the packet is easily deduced:
packetSize = sizeof(UDPacketHeader) + payloadSize
The payload contains the raw sound data (compressed and encoded).

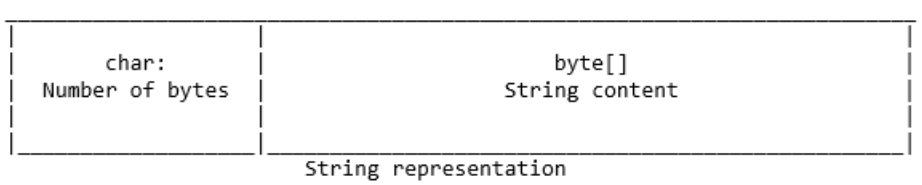
2.3 TPC: Data transmission:

The only data types transmitted over the network are:
- char
- int
- string
- List of any of the above types

If a command takes many arguments, they are concatenated one after another in the payload, without any separator.

2.4 TPC: String representation:

In the payload, string are represented by:
- the size of the string (in bytes) on a byte
- the string content (byte array)

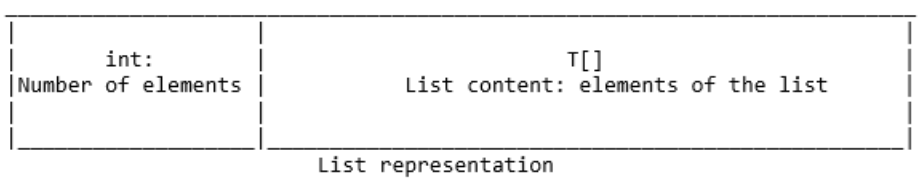


String representation

With this representation, string are limited to 256 characters.

2.5 TPC: List representation:

In the payload, a List<T> is represented by:
- the number of elements in the list (on an int)
- the elements of the string one after another without separator (T array)



List representation

The type T of the list is NOT specified.
Depending on the command, the recipient should know what kind of list it is receiving.

3. Protocol: Client/Server Commands
~~~

3.1 RequestAuth:

Command ID: 1
Name: RequestAuth
Arguments: NONE
Sender/Recipient: Server -> Client
Expected Answers: SignIn | SignUp
Possible ErrorCodes: NONE

Description:
RequestAuth is the first command sent by the server once a client connects.
It asks the client to either log in (SignIn) or create a new account (SignUp).
If a client is too long to do either, the server closes the connection.

3.2 SignUp:

Command ID: 2
Name: SignUp
Arguments: string: username, string: password
Sender/Recipient: Client -> Server
Expected Answers: OK | KO
Possible ErrorCodes: InvalidUsername | InvalidPassword | UnavailableUsername

Description:
SignUp is used by a client to create a new account.
~~~

```
    The server can confirm the successful creation of the account, or inform that a problem happened.
    If the account creation was successful, the user is signed in at the same time.



3.3 SignIn:
-----------


Command ID:             3
Name:                   SignIn
Arguments:              string: username, string: password
Sender/Recipient:       Client -> Server
Expected Answers:       SendFriendsList | KO
Possible ErrorCodes:    InvalidCredentials | UserAlreadyConnected

Description:
    SignIn is used by a client to log in.
    The server can either:
        - confirm the successful logging by sending to the user its friends list
        - inform the user that a problem happened



3.4 SendFriendsList:
--------------------


Command ID:             4
Name:                   SendFriendsList
Arguments:              List<string: username, char: status>: friends
Sender/Recipient:       Server -> Client
Expected Answers:       NONE
Possible ErrorCodes:    NONE

Description:
    Once a client has signed in, the server uses SendFriendsList to send to the now authentificated user the
    list of its friends and their status.
    The argument is a list of tuple: (string, char), that is to say that in the payload, each item of the
    list is a string immediately followed by a char.
    More precisely, an item of the list is used this way:
        - string: username of the friend
        - char: specifies the status of the friend, it is a value from the enum FriendStatus, defined below:
            enum FriendStatus
            {
                Offline = 0,
                Online = 1,
                FriendRequestPending = 2
            }



3.5 FriendStatusUpdate:
-----------------------


Command ID:             5
Name:                   FriendStatusUpdate
Arguments:              char: status, string: username
Sender/Recipient:       Server -> Client
Expected Answers:       NONE
Possible ErrorCodes:    NONE

Description:
    When a client is signed in, the server uses FriendStatusUpdate to notify it as soon as one of its friends
    connects or disconnects.
    There are two arguments, a char and a string:
        - char: specifies the status of the friend, it is the value Offline or Online from the enum
          FriendStatus (defined above), therefore:
            . 0  if the friend just went offline (disconnected)
            . 1  if the friend just went online (connected)
```

```
        - string: username of the friend whose status has to be updated



3.6 FriendRequest:
------------------


Command ID:             6
Name:                   FriendRequest
Arguments:              char: operation, string: username, string: username
Sender/Recipient:       (Client -> Server) | (Server -> Client)


    a. Client -> Server:
    -~-~-~-~-~-~-~-~-~-~

Expected Answers:       OK | KO
Possible ErrorCodes:    UnknownUser | AlreadyFriend | RequestPending | FriendsMaxedOut

Description:
    RequestFriend is used by a client to ask to add/remove a user as a friend.
    As the server receives this command, it checks its database and can either:
        - inform the client that a problem occured
        - inform the client that the request has been successfully processed
    There are three arguments, a char and two strings:
        - char: type of the operation: it is a value from the enum FriendRequestType, defined below:
            enum FriendRequestType
            {
                Removal = 0,
                Addition = 1,
            }
        - string: username of the asker (the user that asks to add/remove someone as a friend)
        - string: username of the user being asked (the user that will be removed as a friend / answer
          whether he wants to accept or decline the friend request)


    b. Server -> Client:
    -~-~-~-~-~-~-~-~-~-~

Expected Answers:       NONE
Possible ErrorCodes:    NONE

Description:
    RequestFriend is used by the server, on reception of a valid RequestFriend, if and only if the user being
    asked as a friend is online.
    It is used to notify the user being asked as a friend of the friend request.
    >>> Same use of the three arguments as the  (Client -> Server)  command



3.7 FriendValidation:
---------------------


Command ID:             7
Name:                   FriendValidation
Arguments:              char: boolean, string: username, string: username


    a. Client -> Server:
    -~-~-~-~-~-~-~-~-~-~

Expected Answers:       OK | KO
Possible ErrorCodes:    UnknownUser | AlreadyFriend | FriendsMaxedOut

Description:
    FriendValidation is used by a client to ask to accept/refuse a user as a friend.
    As the server receives this command, it checks its database and can either:
        - inform the asker that a problem occured
        - inform the asker that the validation has been successfully processed
    There are three arguments, a char and two strings:
```

```
        - char: used as a boolean value to specify if the request was accepted or refused:
            . 0  Refused
            . 1  Accepted
        - string: username of the asker (the user that asked to add/remove someone as a friend)
        - string: username of the user being asked (the user that answers whether he accepts or declines the
          friend request)


    b. Server -> Client:
    -~-~-~-~-~-~-~-~-~-~

Expected Answers:       NONE
Possible ErrorCodes:    NONE

Description:
    FriendValidation is used by the server, on reception of a valid FriendValidation, if and only if the
    user whose request has been validated is online.
    It is used to notify the user who made the friend request of the validation.
    >>> Same use of the three arguments as the  (Client -> Server)  command



3.8 AddToCall:
--------------


Command ID:             8
Name:                   AddToCall
Arguments:              string: username, List<string: username>: participants


    a. Client -> Server:
    -~-~-~-~-~-~-~-~-~-~

Expected Answers:       OK | KO
Possible ErrorCodes:    UnknownUser | UserOffline

Description:
    AddToCall is used by a client to call a friend (or add him/her to an ongoing call).
    There are two arguments, a string and a list of strings:
        - string: username of the callee (the user invited to join the call)
        - List<string>: list of the usernames of the users present in the call


    b. Server -> Client:
    -~-~-~-~-~-~-~-~-~-~

Expected Answers:       AnswerCall
Possible ErrorCodes:    NONE

Description:
    AddToCall is used by the server to notify a client of an incoming call.
    There are two arguments, a string and a list of strings:
        - string: username of the caller (the user inviting to join the call)
        - List<string>: list of the usernames of the users present in the call



3.9 AnswerCall:
---------------


Command ID:             9
Name:                   AnswerCall
Arguments:              char: answer, string: username
Sender/Recipient:       Client -> Server
Expected Answers:       ConnectToPeer
Possible ErrorCodes:    NONE

Description:
    AnswerCall is used by a client to notify the server that a call is accepted/refused.
    There are two arguments, a char and a string:
```

```
        - char: used as a boolean value to specify if the call was accepted or refused:
            . 0  Refused
            . 1  Accepted
        - string: username of the caller (the user inviting to the call)



3.10 CallDropped:
-----------------


Command ID:             10
Name:                   CallDropped
Arguments:              string: username
Sender/Recipient:       Server -> Client
Expected Answers:       NONE
Possible ErrorCodes:    NONE

Description:
    CallDropped is used by the server to notify a client that a user refused its call.
    There is only one argument, a string:
        - string: username of the callee who refused the call



3.11 Host:
----------


Command ID:             11
Name:                   Host
Arguments:              string: ip, string: username
Sender/Recipient:       Server -> Client
Expected Answers:       HostReady
Possible ErrorCodes:    NONE

Description:
    Host is used by the server to notify a client that a user accepted its call, and provide that user's IP
    so that the caller can identify it.
    The caller is asked to provide a port for the callee to connect to.
    There are two argument: two strings:
        - string: ip of the callee who accepted the call (so that the caller can identify it and accept its
          connection)
        - string: username of the callee who accepted the call



3.12 HostReady:
---------------


Command ID:             12
Name:                   HostReady
Arguments:              int: port, string: username
Sender/Recipient:       Client -> Server
Expected Answers:       NONE
Possible ErrorCodes:    NONE

Description:
    HostReady is used by the client to provide the server a port for the callee to connect to.
    There are two argument: an int and a string:
        - int: port on the client for the callee to connect to
        - string: username of the callee who accepted the call



3.13 ConnectToPeer:
-------------------
```

```
Command ID:             13
Name:                   ConnectToPeer
Arguments:              int: port, string: ip, string: username
Sender/Recipient:       Server -> Client
Expected Answers:       NONE
Possible ErrorCodes:    NONE

Description:
    ConnectToPeer is used by the server to provide the callee with the ip and the port of the caller to
    connect to.
    There are three argument: an int and two strings:
        - int: port on the caller for the callee to connect to
        - string: ip of the caller so that the callee can connect directly to it
        - string: username of the caller



3.14 OK:
--------


Command ID:             14
Name:                   OK
Arguments:              int: commandID
Sender/Recipient:       Server -> Client
Expected Answers:       NONE
Possible ErrorCodes:    NONE

Description:
    OK is used by the server to notify the client that a command was successful.
    There is only one argument, an int:
        - int: the ID of the command that was successful



3.15 KO:
--------


Command ID:             15
Name:                   KO
Arguments:              int: errorCode
Sender/Recipient:       Server -> Client
Expected Answers:       NONE
Possible ErrorCodes:    NONE

Description:
    KO is used by the server to notify the client that an error occured.
    There is only one argument, an int:
        - int: an error code that describes the error encountered




4. Protocol: Peer-to-Peer Commands
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


4.1 Welcome:
------------


Command ID:             16
Name:                   Welcome
Arguments:              List<string: username, char: status>: participants
Sender/Recipient:       Client -> Client
Expected Answers:       NONE
Possible ErrorCodes:    NONE
```

```
Description:
    Once a new client connects to the host, the host uses Welcome to send to this new guest who joined the
    call the list of the users in the call and their ID.
    The argument is a list of tuple: (string, char), that is to say that in the payload, each item of the
    list is a string immediately followed by a char.
    More precisely, an item of the list is used this way:
        - string: username of the user in the call
        - char: ID attributed to the user to identify him in the call
```

4.2 ParticipantStatusUpdate:

```
Command ID:             17
Name:                   ParticipantStatusUpdate
Arguments:              char: status, char: userID, string: username
Sender/Recipient:       Client -> Client
Expected Answers:       NONE
Possible ErrorCodes:    NONE

Description:
    When a participant joins/leaves the call, the host uses ParticipantStatusUpdate to notify everyone else
    that this user left/joined the call.
    There are three arguments, two char and a string:
        - char: specifies the status of the participant, it is the value Offline or Online from the enum
          FriendStatus (defined far above), therefore:
            . 0  if the participant just left the call
            . 1  if the participant just joined the call
        - char: ID attributed to the user to identify him in the call
        - string: username of the user whose status has changed
```

4.3 UDPReady:

```
Command ID:             18
Name:                   UDPReady
Arguments:              int: port
Sender/Recipient:       Client -> Client
Expected Answers:       NONE
Possible ErrorCodes:    NONE

Description:
    UDPReady is used by a host client (caller) to provide another client (the callee) with a UDP port to
    connect to.
    There is only one argument, an int:
        - int: UDP port for the callee to connect to
```

4.4 SendText:

```
Command ID:             19
Name:                   SendText
Arguments:              int: userID, string: message
Sender/Recipient:       Client -> Client
Expected Answers:       NONE
Possible ErrorCodes:    NONE

Description:
    SendText is used by a client to send a text message to another client.
    There are two arguments, an int and a string:
        - int: the ID of the user from whom the message originated
        - string: the message to send
```

5. Protocol: Error Codes

5.1 InvalidUsername:

ErrorCode: 1
Name: InvalidUsername
Commands: SignUp
Sender/Recipient: Server -> Client

Description:
To be valid, a username has to:
- be at LEAST 2 characters long
- be at MOST 20 characters long
- contain only alpha-numeric characters or underscores (A-Za-z0-9_)

5.2 InvalidPassword:

ErrorCode: 2
Name: InvalidPassword
Commands: SignUp
Sender/Recipient: Server -> Client

Description:
To be valid, a password has to:
- be at LEAST 2 characters long
- be at MOST 30 characters long
- contain only alpha-numeric characters, underscores, whitespace, commas, colons, semi-colons or arobases: (A-Za-z0-9_ ,:;@)

5.3 UnavailableUsername:

ErrorCode: 3
Name: UnavailableUsername
Commands: SignUp
Sender/Recipient: Server -> Client

Description:
The username provided to create this account is not available.

5.4 UnknownUser:

ErrorCode: 4
Name: UnknownUser
Commands: FriendRequest | FriendValidation | AddToCall
Sender/Recipient: Server -> Client

Description:
The user is unknown in this context:
- On a RequestFriend, it can mean that:

- no user with the given username could be found in the database (when trying to add a friend)
- no user with the given username could be found in the user's friends list (when trying to remove a friend)

- On a FriendValidation, it means that no user with the given username has made a friend request with the user who sent the command
- On a AddToCall, it means that no user with the given username has been found among the friends of the user who sent the command

5.5 UserAlreadyConnected:

ErrorCode: 5
Name: UserAlreadyConnected
Commands: SignIn
Sender/Recipient: Server -> Client

Description:
A client is already connected with this user account.

5.6 AlreadyFriend:

AlreadyFriend | FriendsMaxedOut

ErrorCode: 6
Name: AlreadyFriend
Commands: FriendRequest | FriendValidation
Sender/Recipient: Server -> Client

Description:
The user with the given username is already a friend of the user that is trying to add a friend.

5.7 FriendsMaxedOut:

ErrorCode: 7
Name: FriendsMaxedOut
Commands: FriendRequest | FriendValidation
Sender/Recipient: Server -> Client

Description:
The user who is trying to add a friend has reached its upper limit of friends (MAX_FRIENDS = 20).

5.8 UserOffline:

ErrorCode: 8
Name: UserOffline
Commands: AddToCall
Sender/Recipient: Server -> Client

Description:
The user with the given username is offline.

5.9 RequestPending:

ErrorCode: 9
Name: RequestPending

Commands: FriendRequest
Sender/Recipient: Server -> Client

Description:
The user with the given username has already sent the client a friend request, or the other way around.

5.10 AuthentificationTimeOut:

ErrorCode: 10
Name: AuthentificationTimeOut
Commands: NONE
Sender/Recipient: Server -> Client

Description:
The client has been connected to the server without being identified for too long.

5.11 UnknownCommand:

ErrorCode: 11
Name: UnknownCommand
Commands: Any
Sender/Recipient: Server -> Client

Description:
The command sent is unknown in this context (example: AddToCall without being signed in).
Also raised if a command is sent without the mandatory arguments.

5.12 InvalidCredentials:

ErrorCode: 12
Name: InvalidCredentials
Commands: SignIn
Sender/Recipient: Server -> Client

Description:
No user with this username and password could be find in the database.

6. Protocol: Security considerations

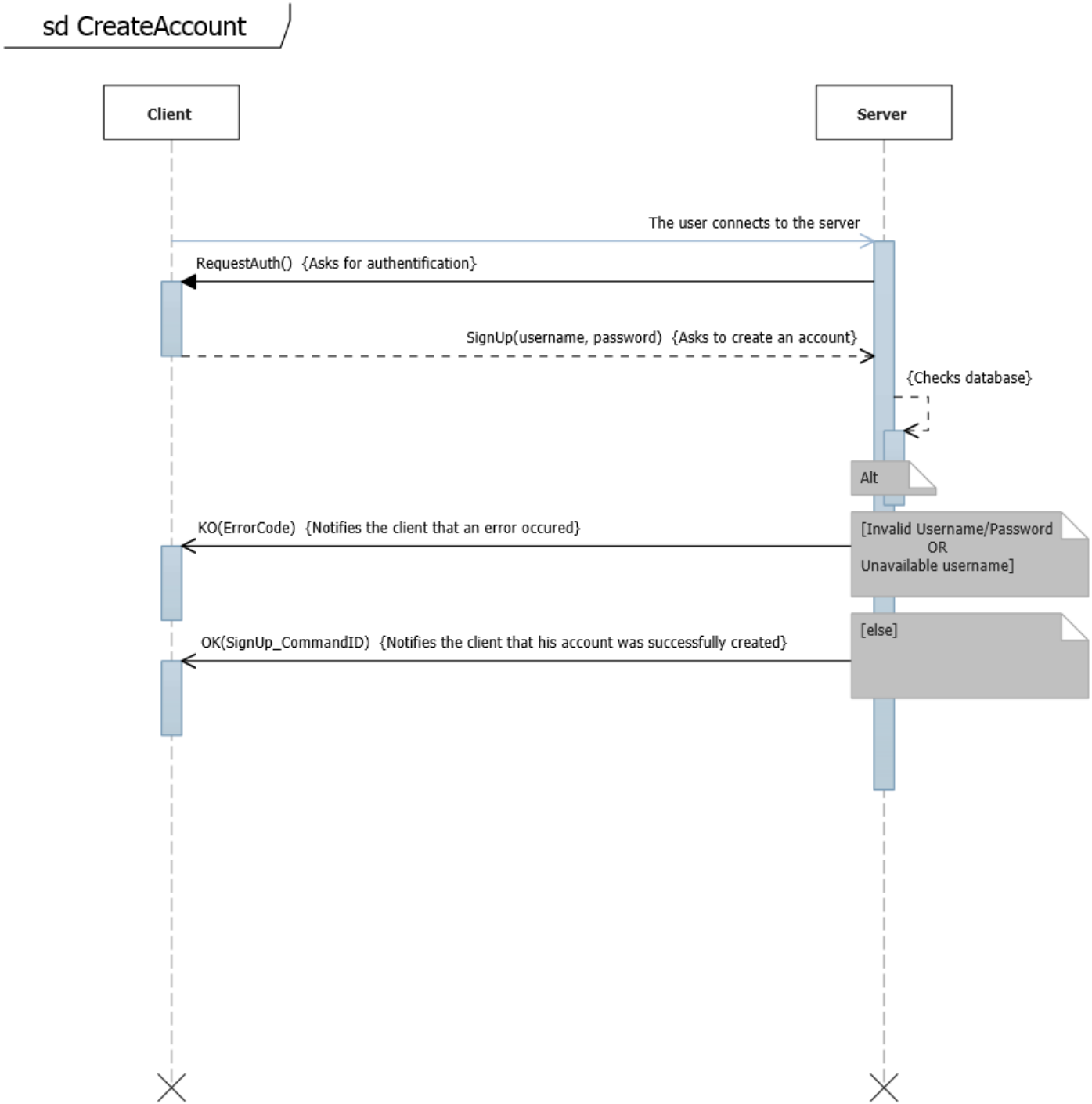
6.1 Call Requests:

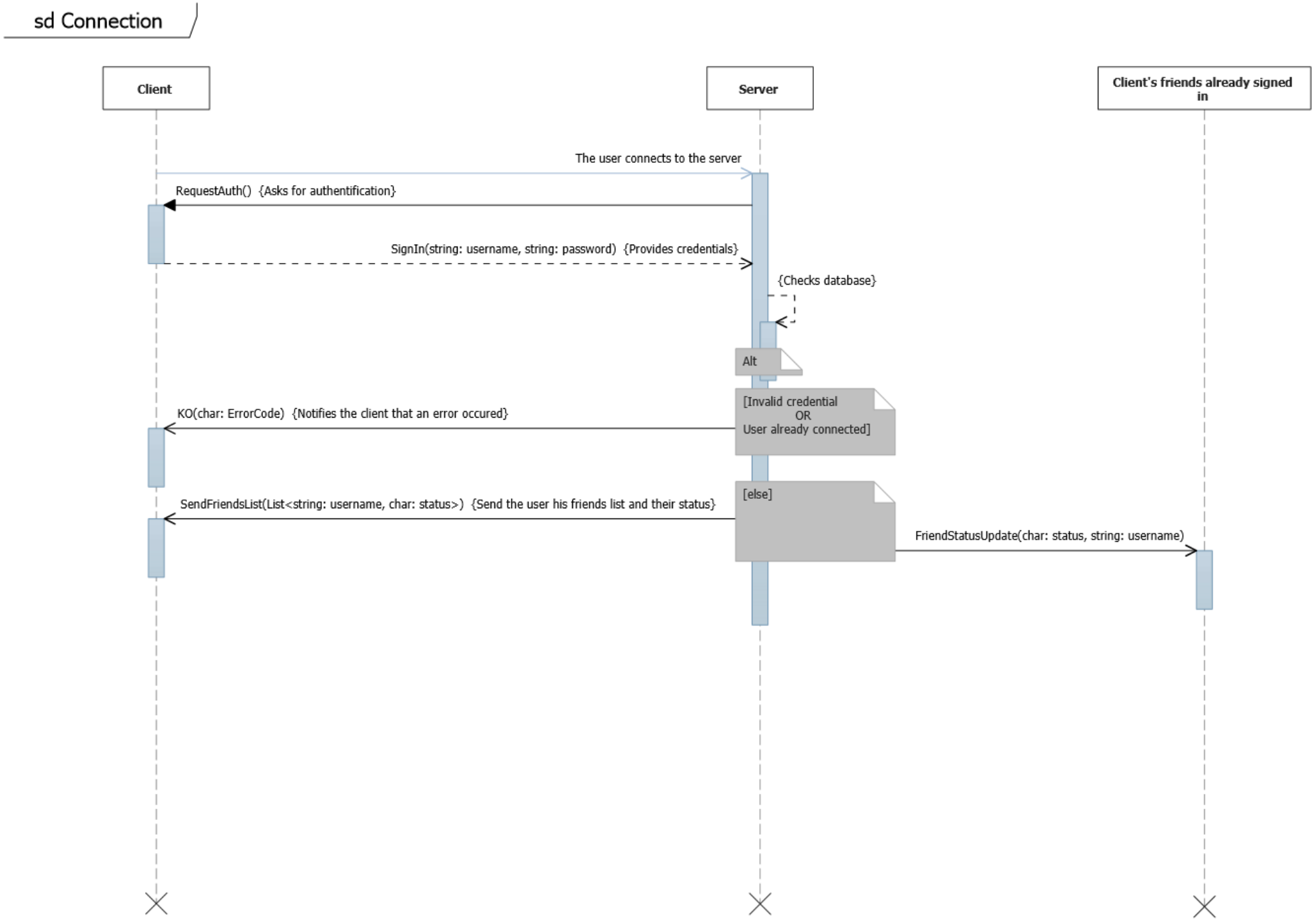
As soon as a call request is created, the server add it to a list of call requests it maintains.
The server keeps track of these call requests until either:
- the call request is refused
- the call request is accepted AND it has forwarded the ip and port of the caller to the callee.

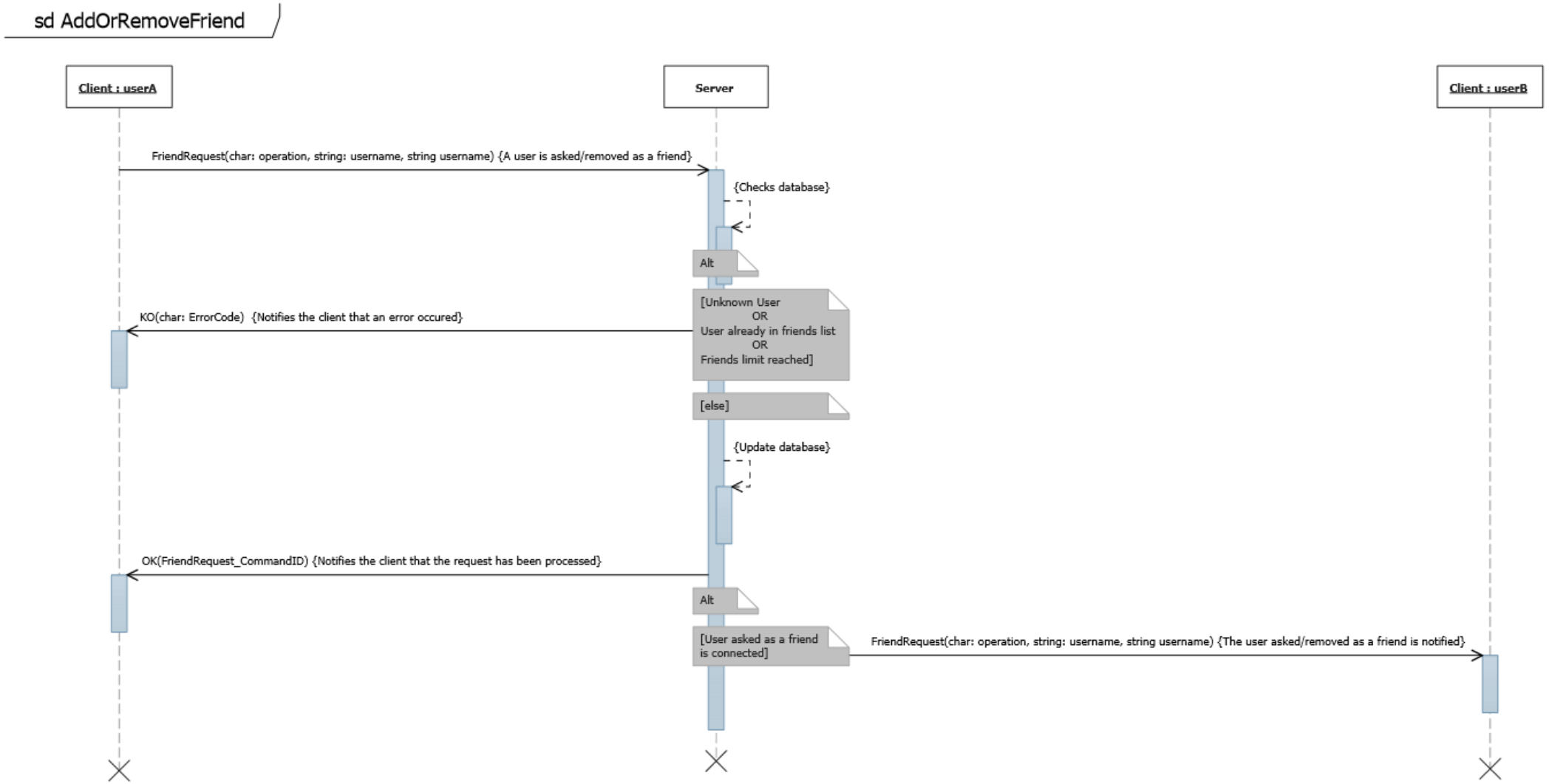
That way, a client cannot bypass the first step of the process and get the ip and a port of a client without its calling it.

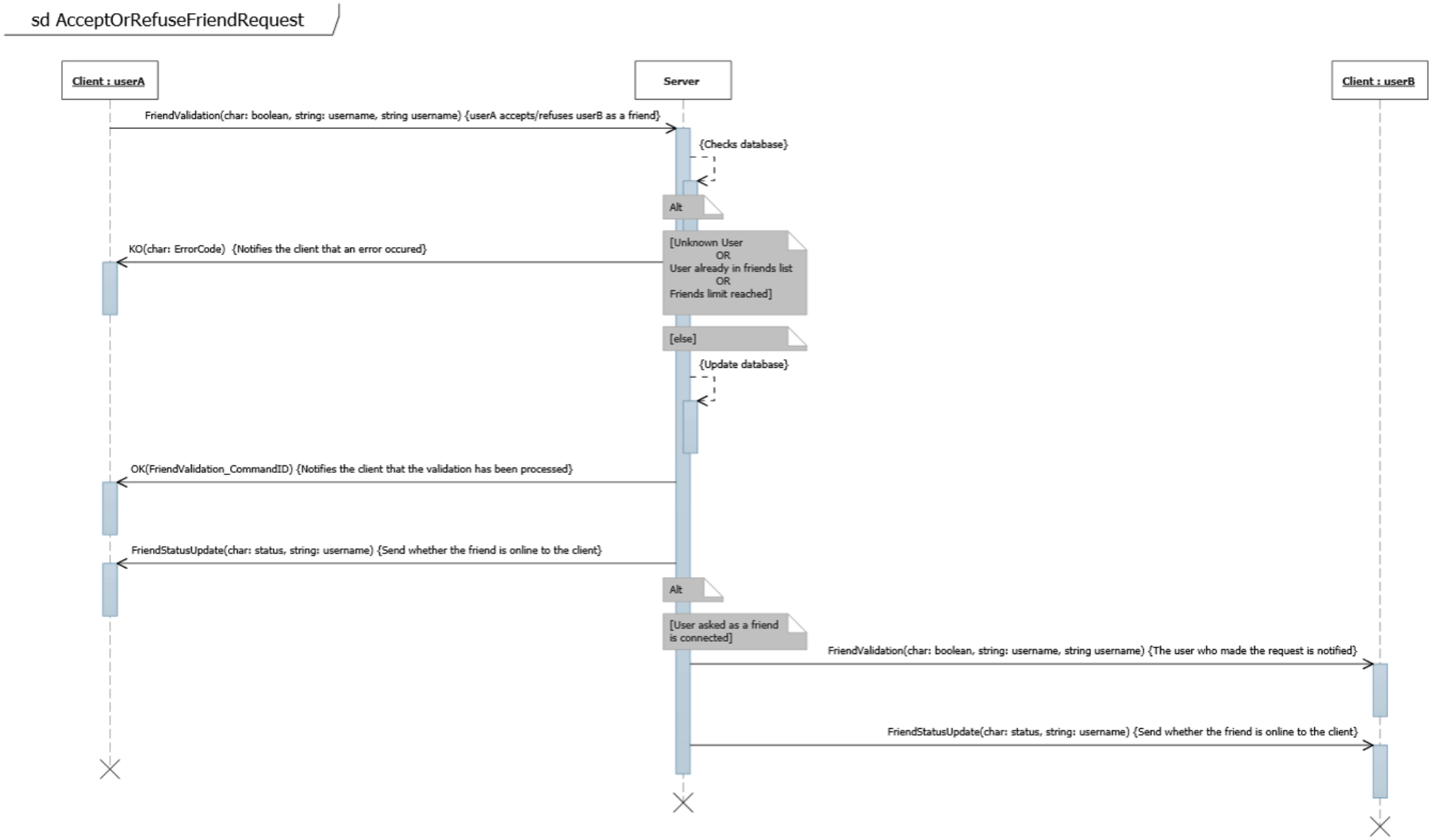
6.2 Peer-to-Peer connection:

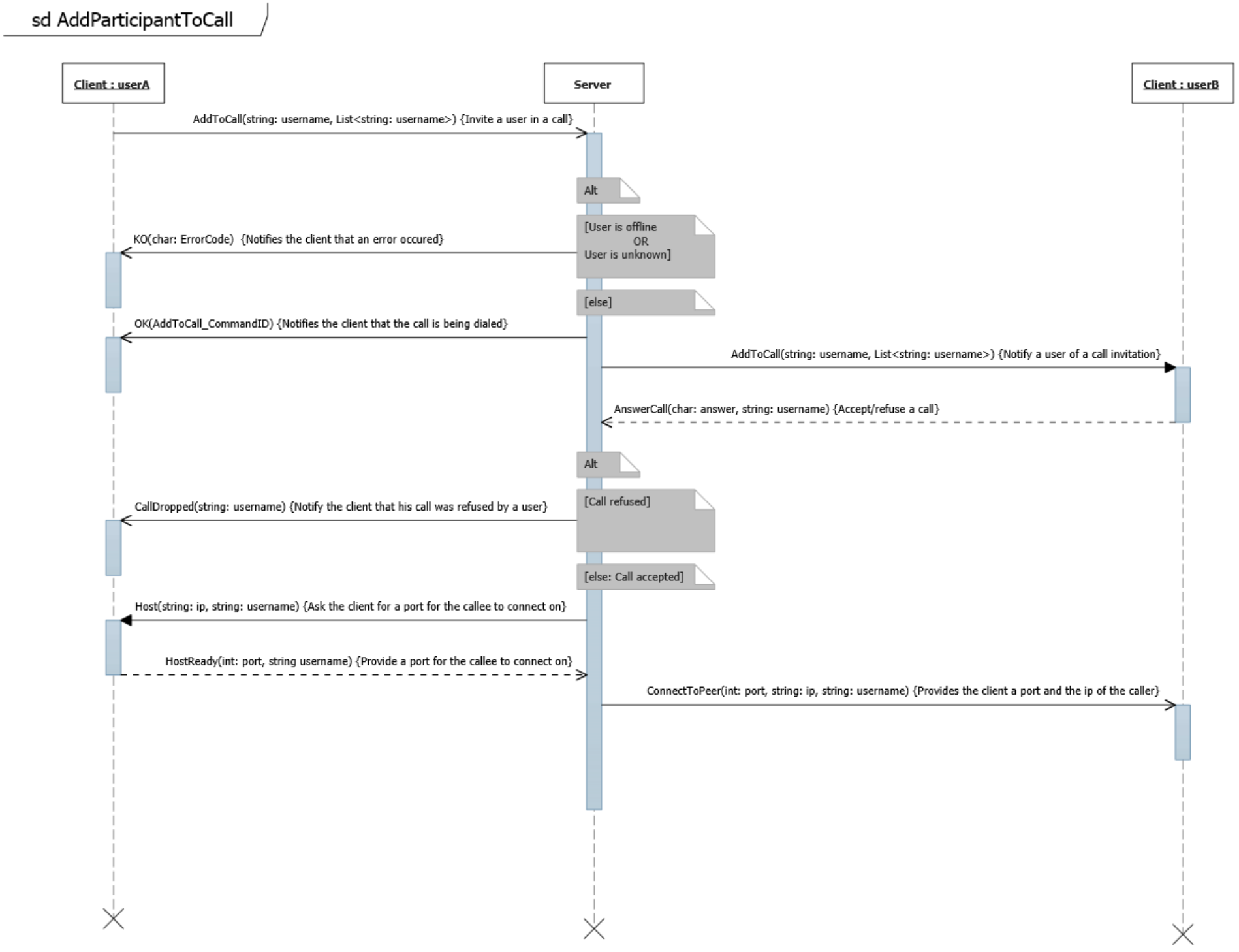
Before two clients can establish a peer-to-peer connection, the server acts as a proxy to provide the callee with the IP and a port of the caller.
But before that, the server provide the caller with the IP of the callee.
That way, when the caller receive a direct connection of a caller, it can check if this is the expected callee.

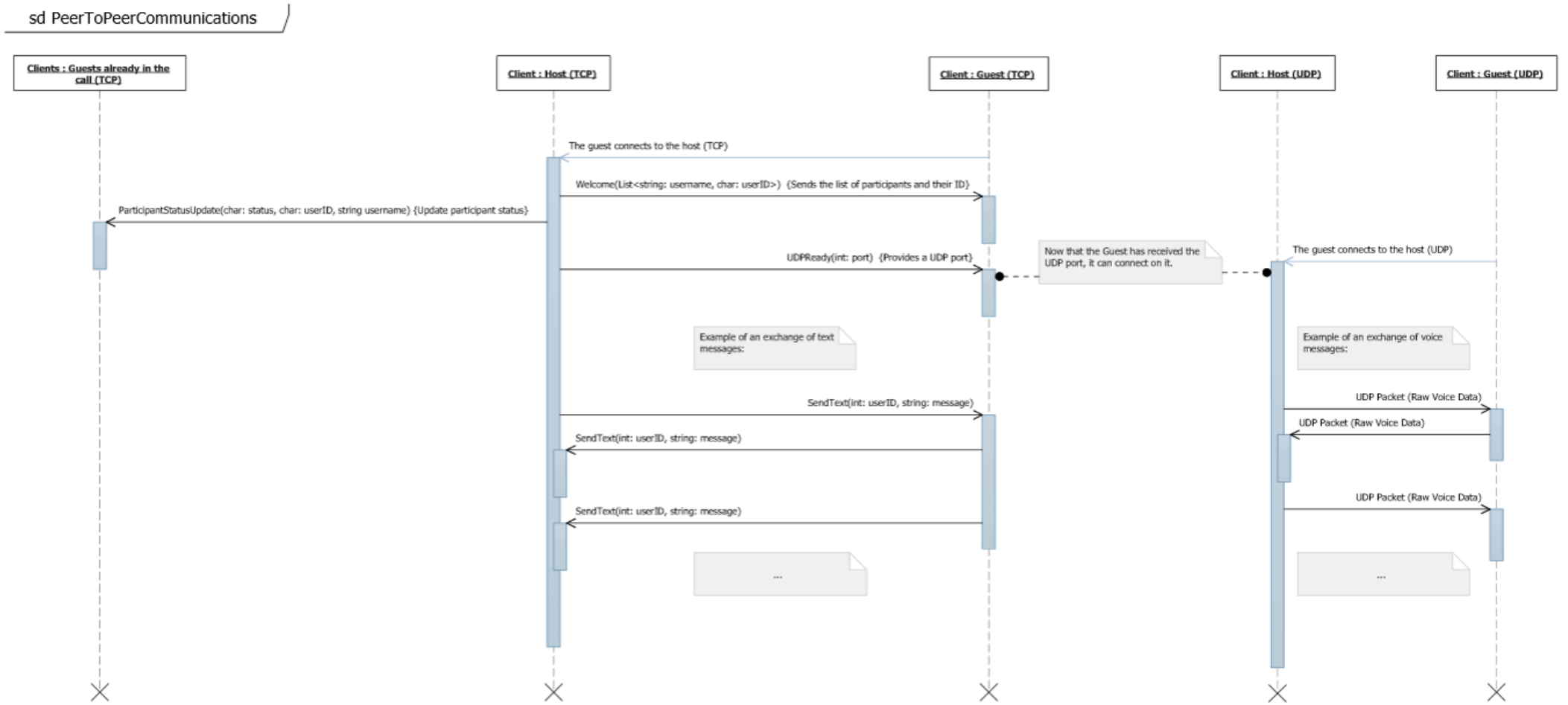

sd CreateAccount
Client
Server
The user connects to the server
RequestAuth() {Asks for authentification}
SignUp(username, password) {Asks to create an account}
{Checks database}
Alt
KO(ErrorCode) {Notifies the client that an error occured}
[Invalid Username/Password
OR
Unavailable username]
OK(SignUp_CommandID) {Notifies the client that his account was successfully created}
[else]


sd Connection
Client
Server
Client's friends already signed in
The user connects to the server
RequestAuth() {Asks for authentification}
SignIn(string: username, string: password) {Provides credentials}
{Checks database}
Alt
[Invalid credential
OR
User already connected]
KO(char: ErrorCode) {Notifies the client that an error occured}
[else]
SendFriendsList(List<string: username, char: status>) {Send the user his friends list and their status}
FriendStatusUpdate(char: status, string: username)


sd AddOrRemoveFriend
Client : userA
Server
Client : userB
FriendRequest(char: operation, string: username, string username) {A user is asked/removed as a friend}
{Checks database}
Alt
[Unknown User
OR
User already in friends list
OR
Friends limit reached]
KO(char: ErrorCode) {Notifies the client that an error occured}
[else]
{Update database}
OK(FriendRequest_CommandID) {Notifies the client that the request has been processed}
Alt
[User asked as a friend
is connected]
FriendRequest(char: operation, string: username, string username) {The user asked/removed as a friend is notified}


sd AcceptOrRefuseFriendRequest
Client : userA
Server
Client : userB
FriendValidation(char: boolean, string: username, string username) {userA accepts/refuses userB as a friend}
{Checks database}
Alt
KO(char: ErrorCode) {Notifies the client that an error occured}
[Unknown User
OR
User already in friends list
OR
Friends limit reached]
[else]
{Update database}
OK(FriendValidation_CommandID) {Notifies the client that the validation has been processed}
FriendStatusUpdate(char: status, string: username) {Send whether the friend is online to the client}
Alt
[User asked as a friend
is connected]
FriendValidation(char: boolean, string: username, string username) {The user who made the request is notified}
FriendStatusUpdate(char: status, string: username) {Send whether the friend is online to the client}


sd AddParticipantToCall
Client : userA
Server
Client : userB
AddToCall(string: username, List<string: username>) {Invite a user in a call}
Alt
[User is offline
OR
User is unknown]
KO(char: ErrorCode) {Notifies the client that an error occured}
[else]
OK(AddToCall_CommandID) {Notifies the client that the call is being dialed}
AddToCall(string: username, List<string: username>) {Notify a user of a call invitation}
AnswerCall(char: answer, string: username) {Accept/refuse a call}
Alt
[Call refused]
CallDropped(string: username) {Notify the client that his call was refused by a user}
[else: Call accepted]
Host(string: ip, string: username) {Ask the client for a port for the callee to connect on}
HostReady(int: port, string username) {Provide a port for the callee to connect on}
ConnectToPeer(int: port, string: ip, string: username) {Provides the client a port and the ip of the caller}


sd PeerToPeerCommunications
Clients : Guests already in the call (TCP)
Client : Host (TCP)
Client : Guest (TCP)
Client : Host (UDP)
Client : Guest (UDP)
The guest connects to the host (TCP)
Welcome(List<string: username, char: userID>) {Sends the list of participants and their ID}
ParticipantStatusUpdate(char: status, char: userID, string username) {Update participant status}
UDPReady(int: port) {Provides a UDP port}
Now that the Guest has received the UDP port, it can connect on it.
The guest connects to the host (UDP)
Example of an exchange of text messages:
Example of an exchange of voice messages:
SendText(int: userID, string: message)
SendText(int: userID, string: message)
SendText(int: userID, string: message)
UDP Packet (Raw Voice Data)
UDP Packet (Raw Voice Data)
UDP Packet (Raw Voice Data)
...
...