

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA  
CAMPUS DI CESENA

---

Scuola di Ingegneria e Architettura  
Laurea Magistrale in Ingegneria e Scienze Informatiche

TITOLO

*Elaborato in*  
MATERIA

*Presentata da*  
GABRIELE GUERRINI

---

Anno Accademico 2018 – 2019



# Table of contents

<b>1</b>	<b>Scoping</b>	<b>1</b>
1.1	Conditions of satisfaction . . . . .	1
1.2	Analisis of requirements . . . . .	1
1.2.1	Room . . . . .	2
1.2.2	Server . . . . .	4
1.2.3	Client . . . . .	5



# Chapter 1

## Scoping

### 1.1 Conditions of satisfaction

- Project must end up within 50 days from the starting date.
- Project must be developed using Scala language.
- Developing process must be an agile one, in particular Scrum.
- At least two working examples of application realized using the library must be provided.

### 1.2 Analysis of requirements

The deliverable should consist of a library that simplifies the development of distributed client-server applications based upon a concept of room. Videogames largely use rooms in every feature (match, lobby, trading room etc.) but there are lots of potential different domains where such idea can be applied (chatrooms, for instance).

The main idea of the product is to provide a low level support that handles all aspects of communication, so that developers that use the library can focus on program logic itself, without concerning about issues due to networking, serialization, synchronization between clients, integration of clients and rooms, and so on.

Hence, the library provides two main notions of:

- **Room:** place where clients can gather to do something.
- **Client:** entity that might do operation on rooms, such as joining, sending messages etc.

### 1.2.1 Room

A room is something that clients can join and interact with. Rooms have a type and are distinguishable by an univocal id. A room possesses a behavior and a state that embed the custom logic of the program. Obviously those vary from room to room, so the developer will be able to flexibly define its own type of room, namely a room that possesses its own state and behavior.

Rooms can be public or private. A public room can be joined by everyone; on the contrary, private rooms need a specific password to be joined. Public rooms can be created by clients and server both, while private rooms can be set up just by clients.

It must be possible to make a public room private, and so to set a password in such room. Similarly, a private room could be made public, and the existing password must be deleted so the room can be freely joined.

Rooms must expose a locking feature that permits to lock/unlock them. While locked, the room can not be joined by any new client. By default room are unlocked. Both clients in the room and server by itself should have the possibility to lock/unlock a given room.

Rooms closing can be automatic or not. A room is closed when it's going to be deleted, and so no client can join the room. Clients in the room must be notified that the room has been closed.

Auto closing can be specified when creating the room, and by default it is off. When the auto close is off, the room would close only in front of an explicit close calling. Instead, when auto close is on, the room would close if there is no client in the room for a certain period of time. Obviously, it would close in front of an explicit close calling too.

By default, the auto close is off.

#### *Client liveness*

The room must be aware of unreachable and/or inactive clients, and may kick them out after an established period of time. There are two possible ways to handle client liveness monitoring, that is:

- Client and server periodically exchange pings. Inactive clients are never kicked.
- The room kicks a client out if no message flew between such client and the room(both directions) within a certain period of time. The length of the period must be configurable. This is considered as a leave event.

*Room state*

The room state is made up of items that can be everything, from simple numbers to objects.

The state can be split in private and public one about visibility to clients. By default the state is private and never automatically communicated to clients in the room. Part (or the totality) of the state can be made public and periodically synchronized between all connected clients in the room.

*Room behavior*

The room behavior can be reactive and/or proactive both: the first specifies how the room should react as an event occurs, the latter instead defines how the room evolves as time passes.

Regarding reactive behavior, events that could occur, and that will be handled by the room, are:

- **Creation:** the room is created.
- **Closing:** the room is closed, intended as imminent deletion. No more client can join the room in this state.
- **Joinin:** a client joins the room. The room knows the client that is joining. When a client tries to join, as well as password if room is private and locking state, the room checks possible custom joining constraints defined in the room. The client successfully joins the room only if all joining constraint are satisfied.
- **Leaving:** a client exits the room. The room knows the client that is leaving.
- **Receiving a message:** the room receives a message from a client.

*Room property*

In the end, the room can have properties, i.e. metadata values that describe room features and that can be used for several purposes, such as room filtering, joining constraints etc.

Those properties can be defined on room creation and are public to clients, either the ones in the room or not.

A room property is made up of a name and a value; the value can be of four basic types: integer, double, string, boolean.

The room public/private state is considered a room property as well, and there it is by default.

Few examples of room property may be max number of clients in the room, min/max Elo required to join, fiendly fire.

*Communication*

A room must provide two possible mechanisms usefull for communication with joined clients, that is:

- **Tell:** the room sends a message to one specific client.
- **Broadcast:** the room sends a same message to all clients.

### 1.2.2 Server

A server-side developer should be able to create and launch a game server that is listening on a provided address and port. The server will transparently handle communication and routing between server itself and clients. The server should also be able to accept user defined routes. This may be usefull when handling features that leave aside the ones provided from the library, such as login, marketing and so on.

The game server can be stopped or terminated both. In the first case the server is temporarily suspended, and it's execution can be resumed; in the latter case, the server is permanently stopped.

It must be possible to define server behavior on starting and stopping.

Once the server has been created, it must be possible to execute three main operations, that is:

- Define room
- Create room
- Define matchmaking

*Room definition*

It must be possible to define a new type of room, i.e. a room with its own state and custom behavior. If the same room type is defined more than once, the first one will be considered as the correct one, and from the second on it will be ignored.

*Room creation*

The server can instantiate public rooms if required. Obviously, the just created room contains no clients at the beginning.

*Define matchmaking*



### 1.2.3 Client

A client should, in primis, display all existing rooms of a given type, both for public and private rooms. Currently locked rooms must not be displayed. Room visualization must include the possibility for clients to filter rooms using their properties. A client can specify a property name, a value and a strategy to be used to evaluate the property. Operations allowed on room properties while filtering are:

- **Equal:** the property value must be the same of the provided one.
- **Not equal:** the property value must not be the same of the provided one.
- **Greater:** the property value must be greater than the provided one.
- **Lower:** the property value must be lower than the provided one.

Moreover, a client can display just its joined rooms as well.

#### *Room creation*

A client can create a room of a certain type. If the room is private, the provision of a password is mandatory. The client must fail on creating the room if its type is not already defined server side. When creating a room, a client can optionally specify a set of starting properties; each property will be set in the room if present or, otherwise, it will be ignored.

Allowed actions that a client can do on a room are:

- Join
- Leave (precondition: Join)
- Reconnect
- Visualize properties
- Send message (precondition: Join)

#### *Join*

There are three ways a client can join a room, that is:

- **Radom join:** a client joins a random public room of a certain type; filter options may be used to provide some directives.

- **Join by Id:** a client joins a room using its id, and optionally a password(required if the room is private, any provided password will be ignored if the room is public).
- **Auto join:** when successfully creating a room, the client should automatically join it.

#### *Leave*

A client can leave a room. Obviously, this operation is not allowed if the client didn't previously join the room.

#### *Reconnect*

When a client leaves a room, the client can reconnect to such room. It is not considered as a join event. The reconnection fails if the client has not previously joined the room or if it took too long to reconnect. Indeed, there is a fixed period within a client can reconnect to the room.

By default the reconnection feature is disabled. The reconnection period is specified when enabling the reconnection feature.

#### *Visualize properties*

From a given room, a client should have the possibility to visualize all its properties. A client can also retrieve a single property value by specifying the right property name. An error should be notified if the sought property does not exist in the room.

#### *Send message*

A client can send messages to the room. No reply from the room is expected. This feature is enabled once the client joined the room.

#### *Reactive behavior*

In the end, a client can specify a reactive behavior too. Possible events are:

- **Receive message:** the client receives a message from the room.
- **State change:** the client is notified with the new public state of the world.
- **Close room:** the client is notified that the room has been closed.