Architecture and Design

Elie Canonici Merle, Colin González, Sylvain Ribstein,
Kevin Sanchis, Anas Aarab, Paul Laforgue

October 24, 2016
Last updated : November 9, 2016

Abstract

This file provides an the software architecture of the project of the course POCA 2016.

Contents

Introduction

KEYWORD: *scalability, separation of concerns, quality-driven, reusable solutions.*
We are focused on:

- **Liveness**: the operations eventually return something.
- **Safety**: the operations never return anything incorrect.

1 Concept

2 Architecture Description

2.1 Core

The core of the architecture holds within itself the most important features. In order to provide a correct program we try to keep the core as abstract as possible. One very important goal of the core is to give a framework capable to handle concurrency while respecting the properties of liveness and safety. However, having massive players implies a challenge in order to prevent lostupdates and keep some coherence. The strategy is to simulate Lamport's one writer, n readers regular multivalued register. Where the data is stored in shared memory, here a database. To keep clients updated, we shall use timestamps that validate or invalidate data. One advantage of using a register like model is that it has been verified and proved a robust and reliable model in concurrent applications. In the UML figure 1 we see the relationship between each object and its caracteristics.

2.2 BUS

The bus is a trivial abstraction of the communication bus, most likely a WAN TCP connection. However, since implementation hasn't been fixed, we prefer using an abstraction in the architecture giving the ability to other implementations further.

2.3 Abstract Server

We have chosen to admit that the engine, the database interface appilcation and the clients are treated as servers. This enables a horizontal model of extensability and no hierarchy between this objects respecting the philosophy of Scala. On the other hand, having non hierarchical objects makes it difficult to understand which program is playing a keyrole for an attacker giving more security. Servers are expected to implement the methods of the Serverizable interface. In fact, a client program is not always ran in server mode. Servers communicate with each other through BUS objects. In some cases, abstract servers might have to accept several connections. A server accepting connections are called $hosts$ and servers connecting to other servers $guests$.

2.4 Token

Pokemon like tokens, points of interests and possibly other users are represented as tokens to the client. This tokens are instancieted at run time with data stored in de database.

2.5 TokenCollection

Tokens are expected to be used as collections. The class TokenCollection provides this feature. This way, it is easy to manage several tokens at a time.

2.6 Interaction between abstract servers

In order to implement the model of the register, the write operations are done by the engine object and read operations by the engine and clients objects. The engine, allows to bufferise client's write requests , and write ätomicallyẗo the database. The database, represents the shared memory here of the register.

2.7 Protocol

2.7.1 Technical specification

Player

- In order to play, the player must sign in.
- He gives his username and password, the server will ask the database and send him a message which can be a connection to the game (the server will then load the last known state of the player : items, pokémons, position...) or a denial.
- If the player doesn't exist in the database, he has to create an account. He can do so by providing an username and a password to the server.
- While in-game, The player can do many actions :
  - Move to a location in the map with x,y coordinates or via GPS position.
  - List his items.
  - List his collection of pokémons.
  - Interact with tokens (point of interest, pokémons or other players).

Server

- Receives messages from player and send queries to the database.
- Sends information or acknowledgement to a player.

Actions in details

- The client/player wants to move to a location x,y. He asks the server. The server will update his position in the database, and will update the map. He will then send an acknowledgement to the client.
- The client/player signs in. The server will check if the username and the password exists in the database. If so, the player will be able to play, if not, he can retry or create an account.

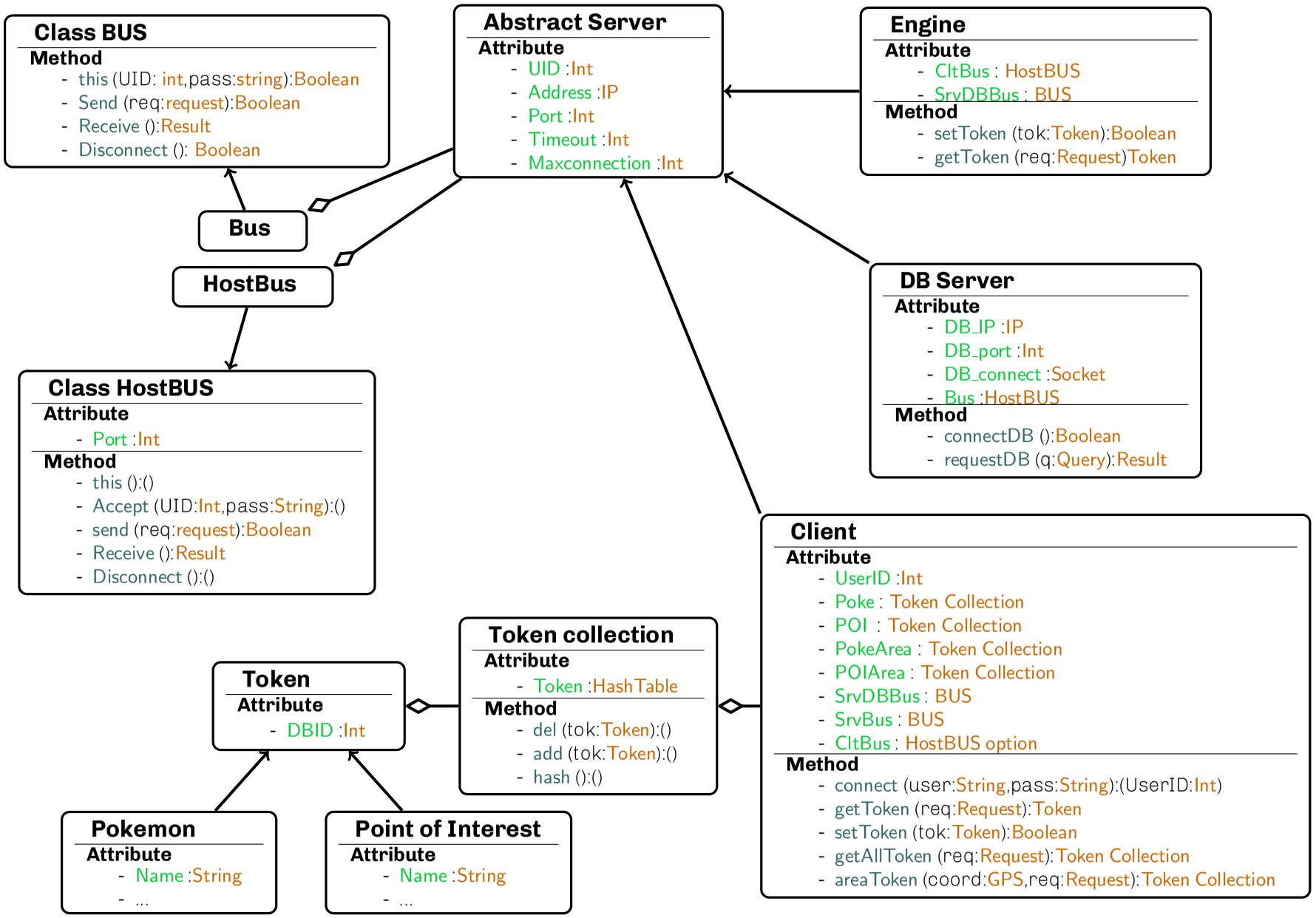

Figure 1: UML of the Core

- The client/player wants to see his items/pokémons. He asks the server. The server will get the list of items/pokémons in the database and send it to him. The client will send an acknowledgement to the server.
- If the client/player interacts with a token, this will be notified to the server and will have consequences :
  - If he interacts with a POI, he can get items from it. (and the server will update his items).
  - If he interacts with a pokémon, he can catch it (and the server will update his collection).
  - If he interacts with a player...

2.7.2 Functional specification

These messages can be sent to the server by the client or to a client by the server :

- **SIGNusername:password** : The client signs in with username and password.
- **SIOK** : The server informs the client that he's connected to the game.
- **SINO** : The server informs the client that his account doesn't exist.
- **CACCusername:password** : The client wants to create an account.
- **CAOK** : The server informs the client that his account is created.
- **CANO** : The server informs the client that his account creation failed (already exists in the database).

- **GETI** : The client wants to see his items.
- **ITEM[items]** : The server sends a list of item to a player/client.
- **ITOK** : The client sends an acknowledgement to the server when he receives the list of item.
- **GETC** : The client wants to see his collection of pokémons.
- **COLL[pokemons]** : The server sends a list of Pokémons to a player/client.
- **COOK** : The client sends an acknowledgement to the server when he receives the list of pokémons.
- **MOVEx:y** : The clients wants to move to a location (x,y).
- **MOOK** : The server updates the player's position in the database, and the map.

- **IWPOIpoi** : The client wants to interact with the POI 'poi'.

- **POID** : The server updates the state of the player (his list of items) and the state of the POI (has been visited by this player).
- **IWPOKpokemon** : The client wants to interact with the 'pokemon'.
- **POKD** : After the mini-game, the server updates the state of the player (his list of pokémons and items).

2.8 Database model

The diagram for *Connection and data acceses* is presented in figure **??**. We are using the modern database library Slick so we can enjoy "the static checking, compile-time safety and compositionality of Scala".

3 Extensions

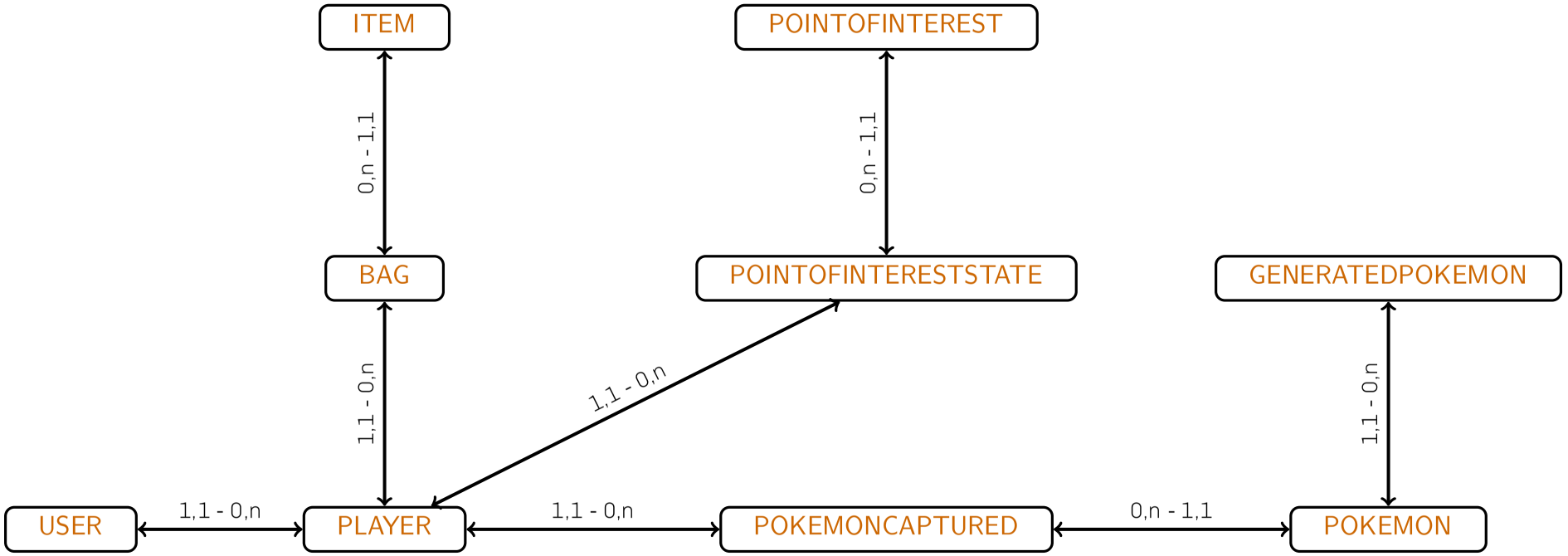

Figure 2: Relational Model