POWER ENJOY

SE2 PROJECT BY:

Stefano Brandoli, Silvia Calcaterra, Samuele Conti

RASD SUMMARY

GOALS and SUBGOALS

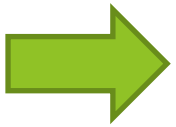
[G1] ALLOW USERS TO FIND AVAILABLE CARS AND RESERVE THEM

- [G1.1] Allow users to access the system
- [G1.2] Allow users to find available cars
- [G1.3] Allow users to reserve available cars

[G2] ALLOW USERS TO USE THE RESERVED CARS IN THE CITY AREA

- [G2.1] Manage the user reservation
- [G2.2] Handle unexpected car situations and user behavior

[G3] GUARANTEE A UNIFORM DISTRIBUTION OF THE CARS IN THE CITY



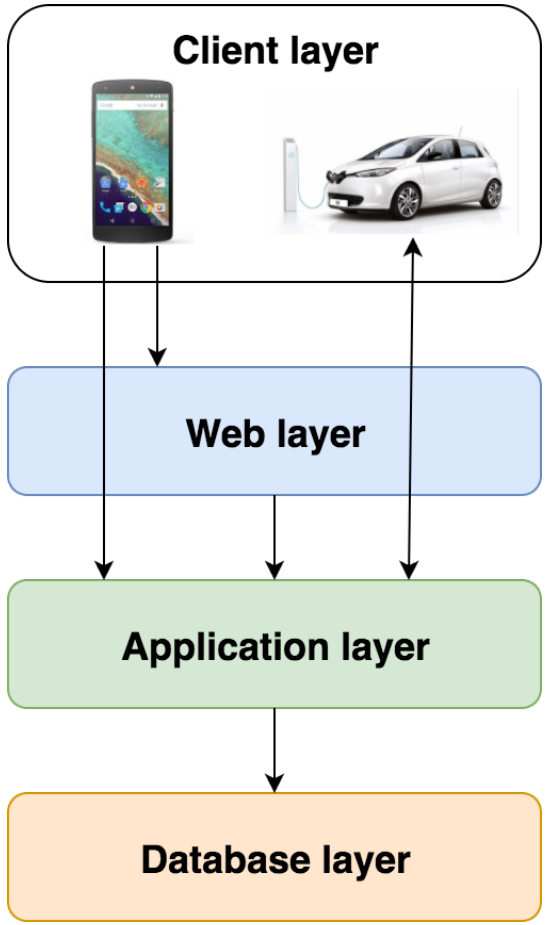
Money Saving Option

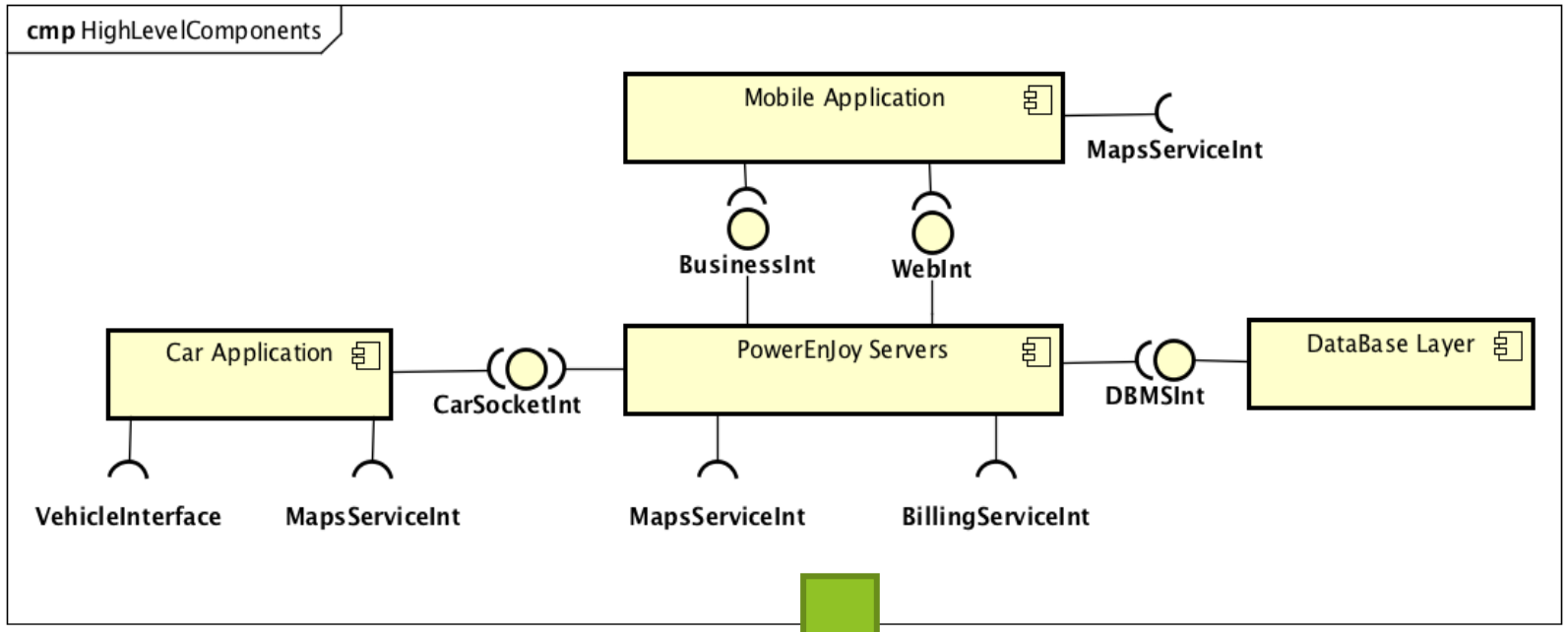
[G4] INCENTIVIZE USERS TO USE PROPERLY THE RESERVED CARS

- [G4.1] Reward the users that ease the experience of other users
- [G4.2] Penalize the users that ruin the experience of other users

DESIGN DOCUMENT

TIERS, MAIN COMPONENTS, BOUNDARIES



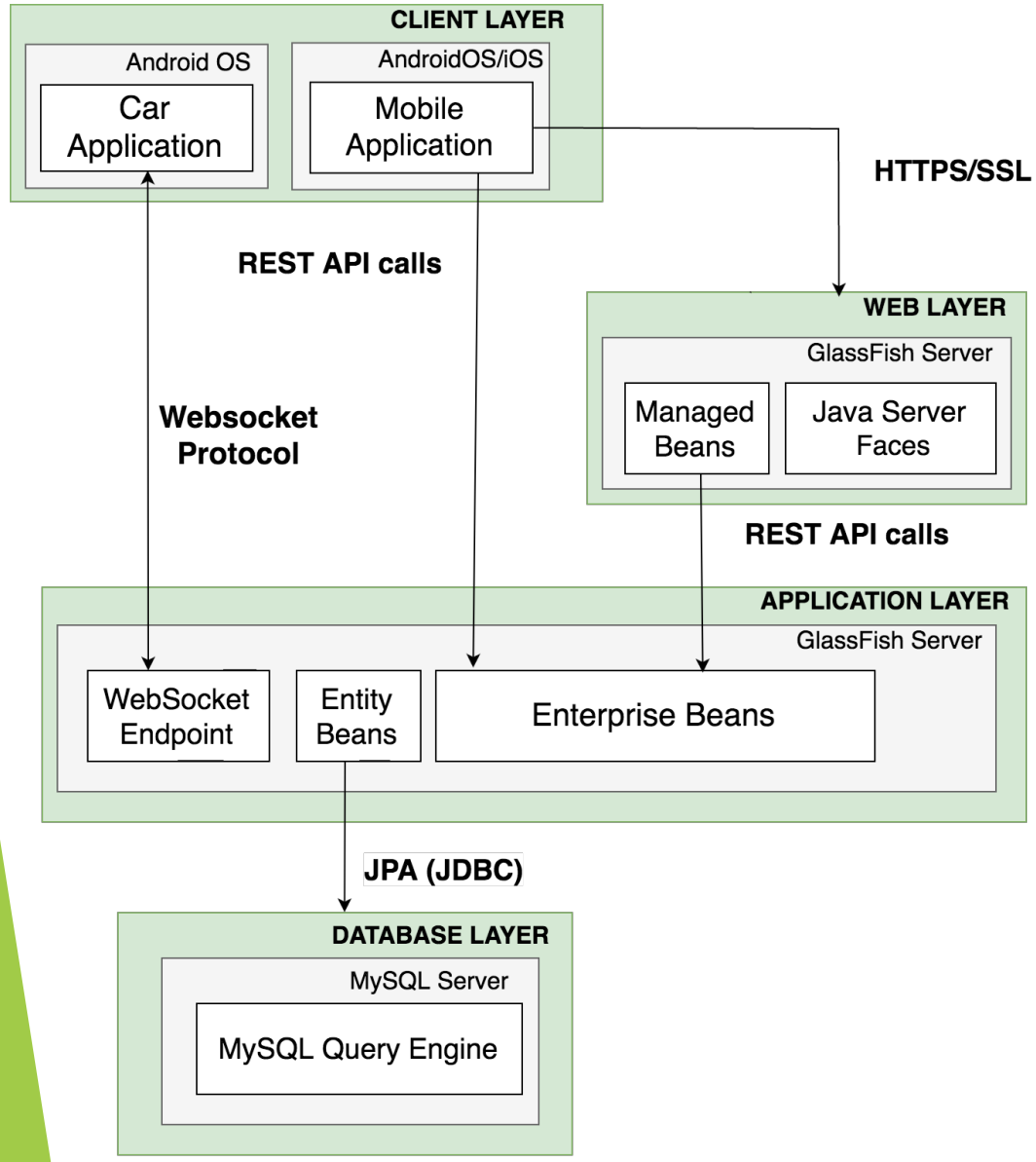


Not Fully Layered: but interaction with Cars is **CRITICAL** in our business

Refinement of the **RASD System Boundaries**

- **Vehicle Interface** — Car Interfacing
- **Maps Service Int** — Google Maps functionalities
- **Billing Service Int** — Stripe for billing management
- **DBMS Int** — JPA abstracting the DBMS connection

DESIGN and PATTERNS meet TECHNOLOGY



Architecture mainly based on **JEE 7**

THIN CLIENT/FAT SERVER

- Avoid platform desynchronization
- Device support flexibility
- Maintenance

→ **MVC**

4-TIER ARCHITECTURE

- Scalability
- Extendability: **future web browser**
- High decoupling between tiers

ACTIVE SERVER (SERVER PUSH)

- **Bidirectionality**
- HTTP Polling inefficiency
- **WebSocket technology**

→ **Publisher/ Subscriber**

API FIRST ARCHITECTURE

→ **Mobile devices first**

- **RESTful technology**: JAX-RS
- Easier API extension
- **JSON**: few assumptions about the receiver

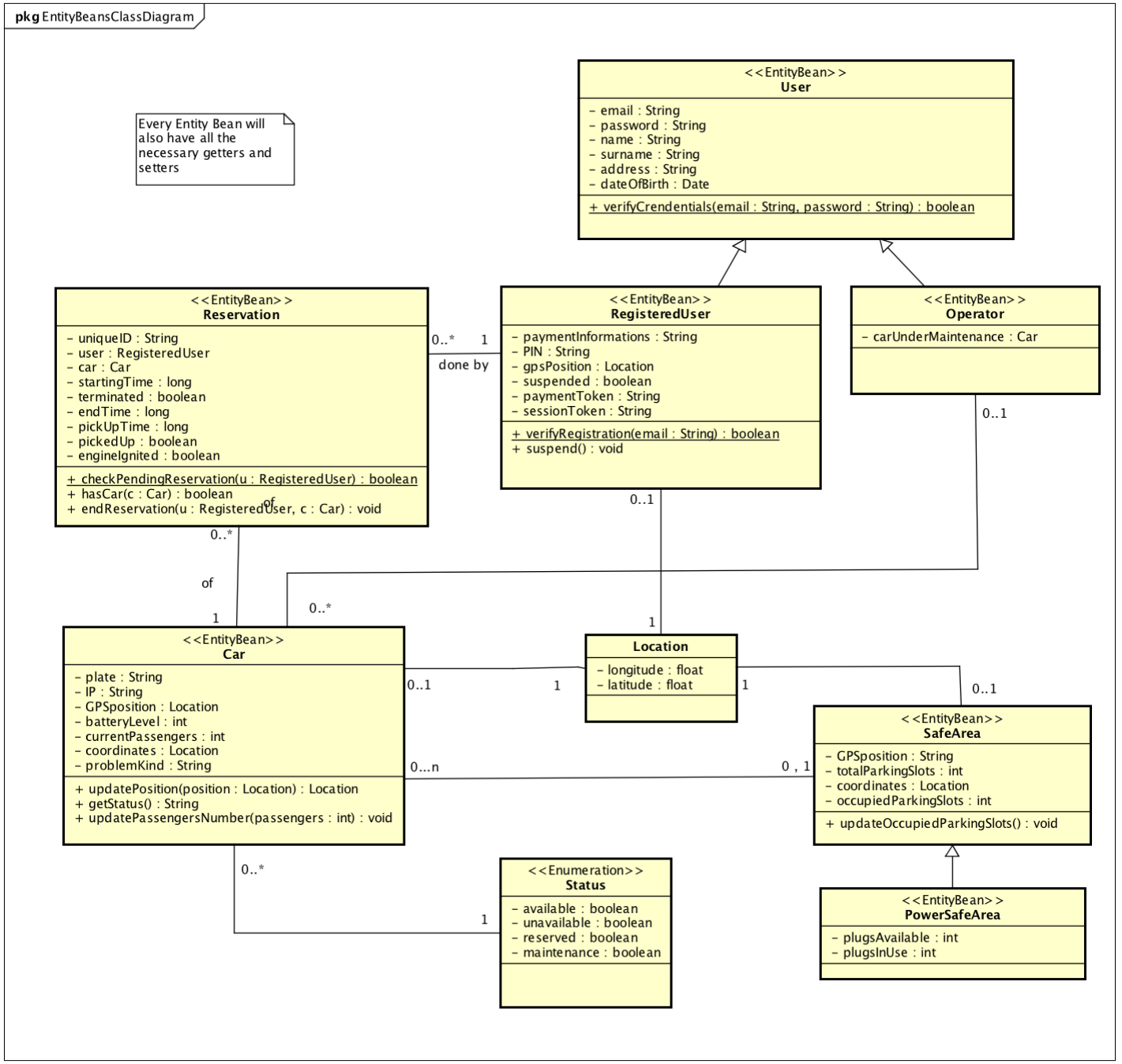
ENTITY BEANS

O/R MAPPING

- Handled by JPA
- JDBC is “under the hood”

SECURITY

- No payment information plainly stored in the DBMS
- **STRIPE Payment Token**



ENTERPRISE JAVA BEANS

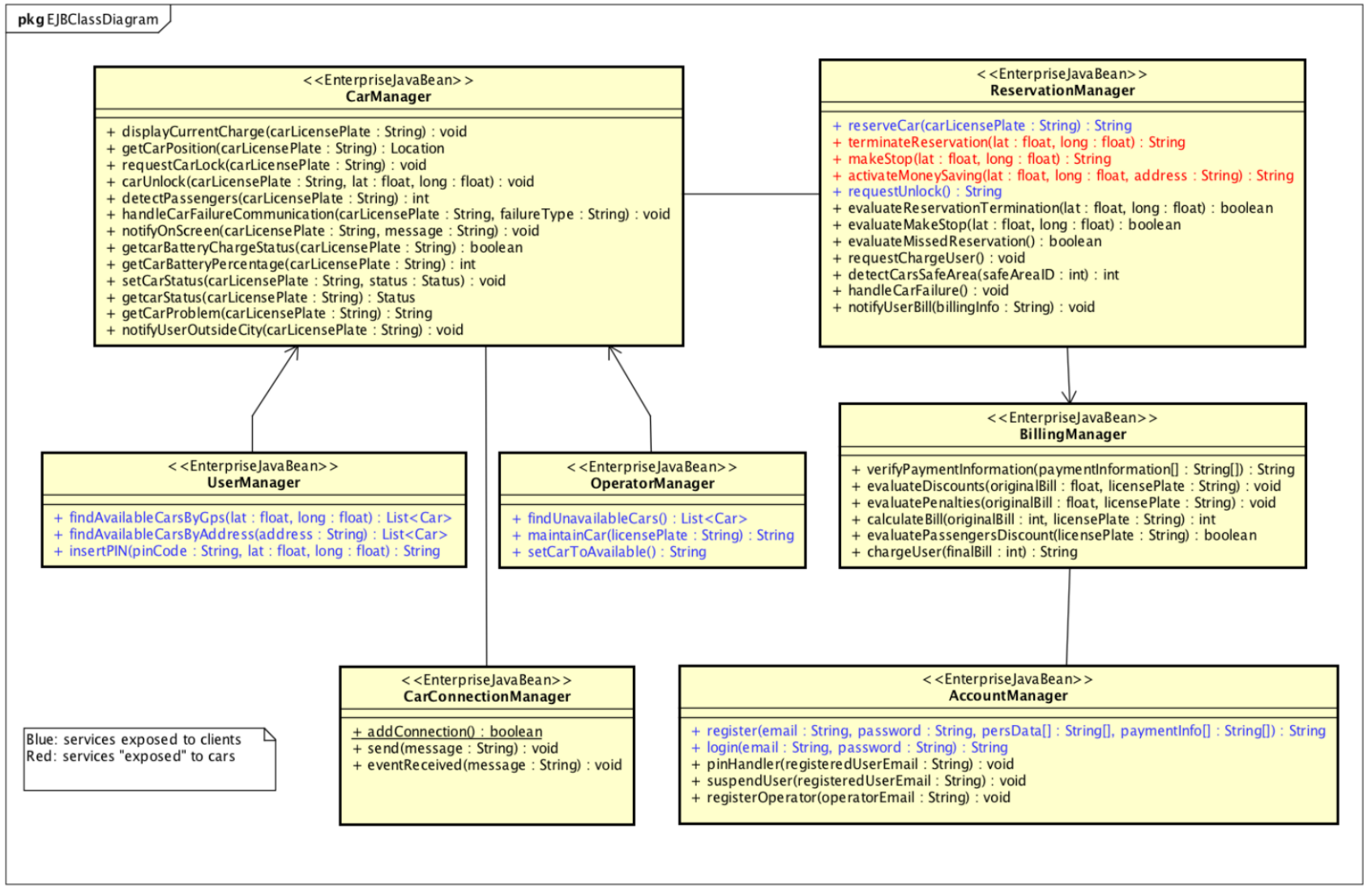
GENERAL PRINCIPLES

- High coesion, loose coupling
- Stateless EJB
  - RESTful principles

MAIN SUB-COMPONENTS

- Account Manager
- User Manager
- Operator Manager
- Reservation Manager
- Billing Manager
- Car Manager
- Car Connection Manager

Exposed API is detaily described in the DD



INTEGRATION TEST PLAN DOCUMENT

INTEGRATION STRATEGIES

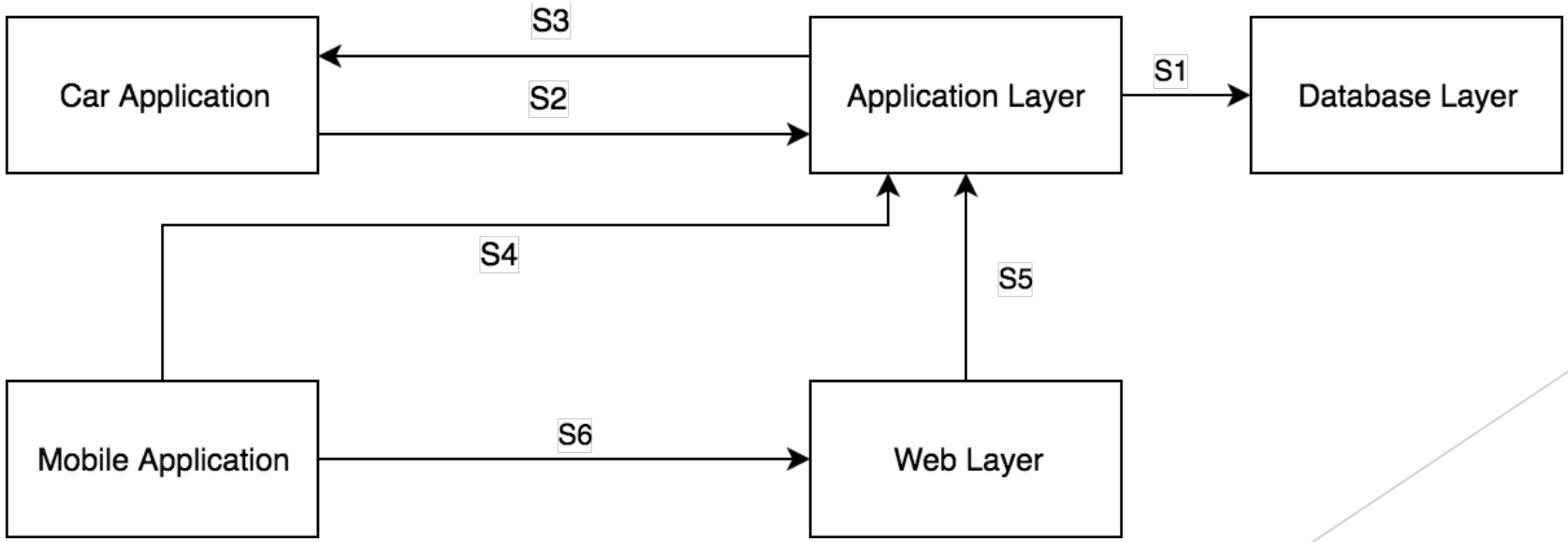
1) Integrate the units belonging to the same subsystem
2) Integrate the subsystems

BOTTOM-UP APPROACH

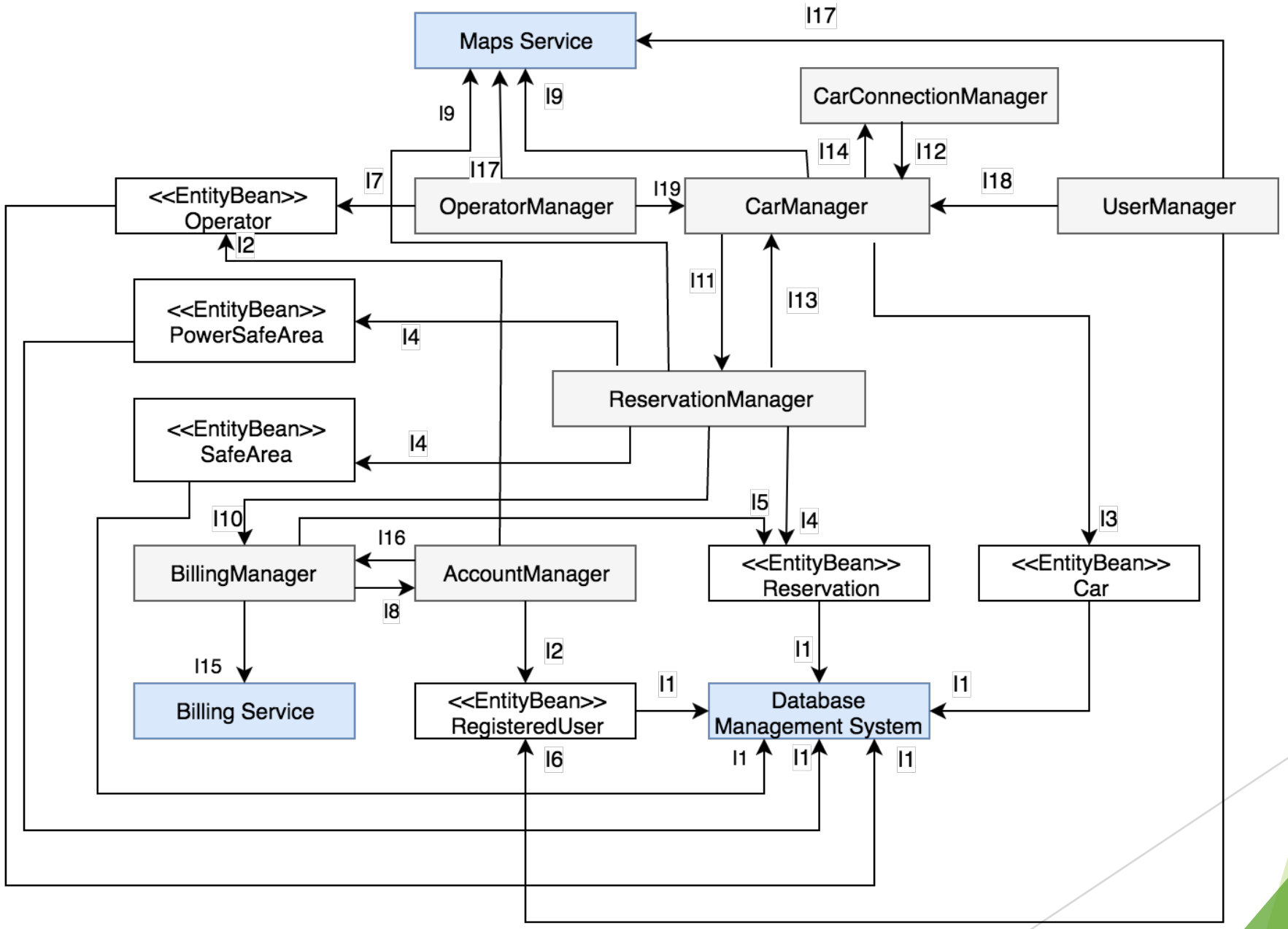
- Start the integration of the most independent units, in order to **limit the number of stubs/mocks**
- Follow more closely the **development process**

CRITICAL MODULES APPROACH

- **First** concentrate on the integration of **the riskiest units**
- Discover sooner eventual critical **"bad behaviours"** that can compromise the fulfillment of the goals



INTEGRATION: APPLICATION LAYER



TOOLS AND TEST EQUIPMENT

SOFTWARE TOOLS
They will be used to automate part of the testing:

- **jUnit** test the **integration of components**
- **Arquillan** check the integration between **components and containers**
- **Mockito** create **stubs and drivers** useful for testing
- **JMeter** test the **performance** and **non functional requirements**

DRIVERS
- **Car Application** driver
- **Application Layer** driver
- **Front End** driver

STUBS
- **Network** stub
- **Vehicle Interface** stub

We will also need:
- **Testing Database**
- Activate the **testing mode for the STRIPE account**
- **Android and iOS devices**

PROJECT PLAN

SIZE ESTIMATION

Following the **COCOMO II approach**, we've identified these **User Function Types**

EXTERNAL INPUTS
- User Registration
- User Login
- Unlock Reserved Car
- Request Car Reservation
- Terminate Car Reservation
- Make a Stop
- Enable Money Saving
- Request Car Maintenance
- Change Car Status

INTERNAL LOGICAL FILES
- User
- Registered User
- Operator
- Reservation
- Car Status
- Safe Area
- Power Safe Area

EXTERNAL OUTPUTS
- Send commands to a PowerEnjoy car
- Send payment notification

EXTERNAL INQUIRIES
- Find available cars by GPS position or address
- Find unavailable cars

EXTERNAL INTERFACE FILES
- Maps Service: Google Maps
- BillingService: Stripe
- Vehicle Interface

- Determine complexity levels
- Assign complexity weights
- Calculate UFPs
- Relate UFPs to SLOC

UFP = 196

***SLOC* = 46 x UFP = 9016**

COST AND EFFORT ESTIMATION: COCOMO II

- **POST-ARCHITECTURE** analysis
- We estimated cost and effort using the **Scale Drivers** and the **Cost Drivers**

Precedentness (PREC)	Low	4.96
Development flexibility (FLEX)	Nominal	3.04
…		
SF		**15.05**

RELY	Required Software Reliability	**High**	**1.10**
DATA	Database size	**Nominal**	**1**
…			
Total	**EAF**		**1.613**

$$\textit{Effort} = A \times EAF \times (KSLOC)^{E}$$

$$\textit{Duration} = C \times (PM)^{F}$$

49 person-months

13 months

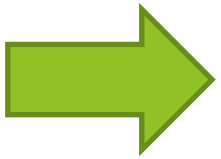
TASKS AND SCHEDULES

By generally following a standard
WATERFALL SCHEMA

1) Deliver the **RASD**
2) Deliver the **DD**
3) Deliver the **ITPD**
4) Deliver the **PP**
5) Implementation + **unit tests**
6) **Integration testing**
7) **Deployment**

Task	Starting Date	End Date
RASD	16/10/2016	13/11/2016
DD	14/11/2016	11/12/2016
ITPD	12/12/2016	15/01/2017
PP	16/01/2017	22/01/2017
Development	23/01/2017	16/10/2017
Deployment	17/10/2017	16/11/2017

As evaluated using the
COCOMO II model



- The project will last **13 months**
- Starting from **October 2016**, finishing in **November 2017**

RISK MANAGEMENT

PROJECT RISKS

- **Schedule Delays**
- Underestimated development time
- Requirements change
- Problems among team members
- Staff illness

TECHNICAL RISKS

- **Scalability issues**
- Loss of data
- Integration failure
- Issues with external services
- **Technical death**

BUSINESS RISKS

- Car provider bankrupt
- Budget
- Legislation change
- System acceptance
  - No acceptance by city administration
  - No acceptance by potential customers
- **Competitors**

CODE INSPECTION

APACHE OFBIZ

We had to perform the code inspection of some classes from the **Apache OFBiz** source code.

These were the classes assigned us:

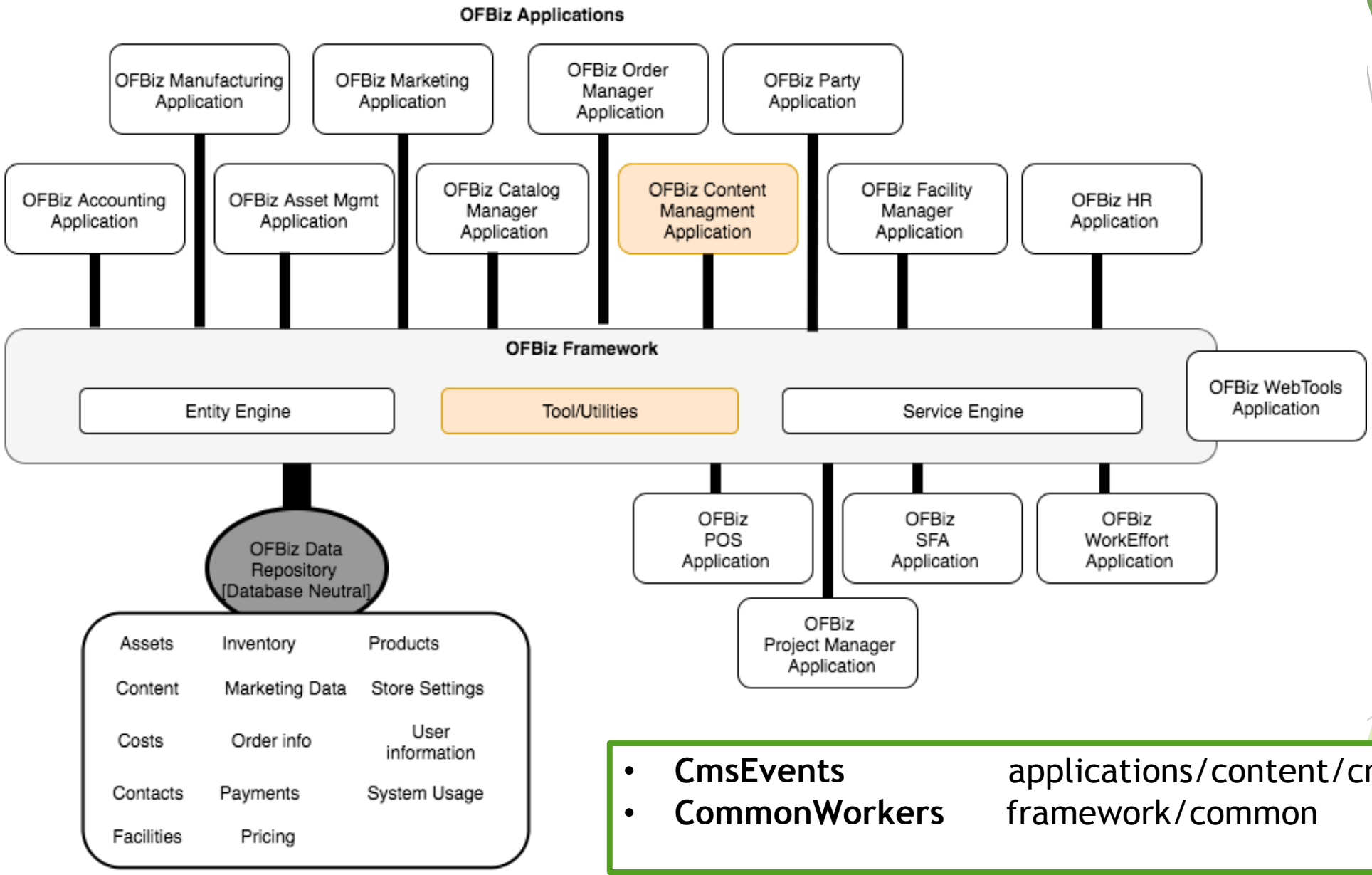
CmsEvents

A class used to setup a website whose content can be managed using the OFBiz built-in features of the Content Management Application (front-end user interface)

CommonWorkers

A class contained inside the framework component set of the system: it is a sort of utility class used by different classes in the entity package

APACHE OFBIZ ARCHITECTURE SCHEMA



- **CmsEvents** applications/content/cms
- **CommonWorkers** framework/common

CMS EVENTS

JavaDoc completely missing → **STARTING POINT:** an **XML snippet!!!**

SYNTACTIC ISSUES

- Some variables and methods names are not meaningful or don't follow naming conventions
- Some indentations missing
- Some curly braces missing in conditional statements
- Some file organization issues, in particular regarding line lengths and line breaks
- Minor issues with variables initialization and declaration

OO ISSUES

- The **main method** of the class is **very long**
- Some methods have **too many parameters**
- Strings copy pasted -> **no constants defined**

COMMON WORKERS

JavaDoc almost missing → **STARTING POINT:** directly see the code

MAIN ISSUES

- Some issues regarding line lengths and breaks
- Some issues regarding the lack of useful comments
- Some issues in the position of declarations inside the code
- Lack of significant actions taken for some catch blocks

QUESTIONS?