# CLIPS

Communication & Localization with Indoor Positioning Systems

---

## Università di Padova

### Template


leaf.gruppo@gmail.com

| | |
|---|---|
| **Versione** | |
| **Data Redazione** | |
| **Redazione** | |
| **Verifica** | |
| **Approvazione** | |
| **Uso** | |
| **Distribuzione** | |

# Diario delle modifiche

| Versione | Data | Autore | Ruolo | Descrizione |
|---|---|---|---|---|
| 0.02 | 2016-03-08 | Oscar Elia Conti | Progettista | Stesura sezione tecnologia Android |
| 0.01 | 2016-03-08 | Oscar Elia Conti | Progettista | Inizio stesura documento |

# Indice

# Elenco delle figure

# 1 Introduzione

## 1.1 Scopo del documento

## 1.2 Glossario

Allo scopo di rendere più semplice e chiara la comprensione dei documenti viene allegato il *Glossario v1.00* nel quale verranno raccolte le spiegazioni di terminologia tecnica o ambigua, abbreviazioni ed acronimi. Per evidenziare un termine presente in tale documento, esso verrà marcato con il pedice $_g$.

## 1.3 Riferimenti utili

### 1.3.1 Riferimenti normativi

- rif

### 1.3.2 Riferimenti informativi

- rif

# 2 Tecnologie utilizzate

In questa sezione vengono descritte le tecnologie sulle quali si basa lo sviluppo di BlueWhere.

## 2.1 Android

### 2.1.1 Descrizione

Android$_g$ è un sistema operativo mobile sviluppato da Google$_g$ e basato su kernel$_g$ Linux$_g$. È stato progettato per essere eseguito principalmente su smartphone$_g$ e tablet$_g$ con interfacce utente specializzate per orologi e televisori. Le versioni di riferimento sono la 4.4 e superiori. L'utilizzo di questa tecnologia è stato richiesto dal proponente.

### 2.1.2 Vantaggi

I principali vantaggi del sistema operativo Android sono:

- possiede una vasta fetta di mercato mobile;
- disponibile su un vasto numero di dispositivi;
- quasi totalmente gratuito ed Open Source$_g$.

### 2.1.3 Svantaggi

I principali svantaggi del sistema operativo Android sono:

- essendoci un vasto numero di produttori di smartphone e tablet che non aggiornano la versione di Android che rilasciano all'interno dei loro dispositivi, Android risulta essere estremamente frammentato;
- necessità di sviluppare applicazioni per dispositivi che possono differire per:
  - prestazioni;
  - risoluzione dello schermo;
  - durata della batteria;
  - sensori disponibili.

# 3 Descrizione dell'architettura

## 3.1 Metodo e formalismo di specifica

## 3.2 Architettura generale

# 4 Design pattern

## 4.1 Design pattern architetturali

## 4.2 Design pattern creazionali

## 4.3 Design pattern strutturali

## 4.4 Design pattern comportamentali

# 5 Stime di fattibilità e bisogno di risorse

# 6 Tracciamento

# 7 Descrizione design pattern

## 7.1 Design pattern architetturali

### 7.1.1 MVP

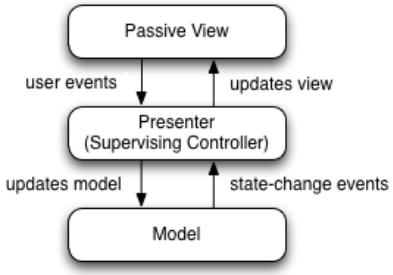


**Figura 1:** Struttura del pattern MVP

Model-View-Presenter (MVP) è un pattern architetturale derivato dal MVC (Model-View-Controller), utilizzato per dividere il codice in funzionalità distinte. Il suo principale ambito di utilizzo è nelle applicazioni un insieme di informazioni deve essere rappresentato mediante un'interfaccia grafica.

**7.1.1.1 Componenti** MVC è basato sul principio di disaccoppiamento di tre oggetti distinti, riducendo in questo modo le dipendenze reciproche; inoltre permette di fornire una maggiore modularità, manutenibilità e robustezza al software.

**7.1.1.1.1 Model** Il Model rappresenta il cuore dell'applicazione: esso definisce il modello dei dati definendo gli oggetti secondo la logica di utilizzo dell'applicazione, ossia la sua business logic. Inoltre, indica le possibili operazioni che si possono effettuare sui dati.

**7.1.1.1.2 View** Nel pattern MVP, il Model è un componente prevalentemente passivo, ma si occupa anche di notificare al Presenter eventuali modifiche del proprio stato. Nella struttura del pattern MVP, la View si occupa di prendere gli input dell'utente e passarli al Controller, affinché esegua operazioni sul Model.

**7.1.1.1.3 Presenter** Il Presenter è l'intermediario tra il Model e la View. Si occupa di implementare l'insieme di operazioni eseguibili sul mo-

dello dei dati attraverso una particolare vista, ossia l'application logic. Ad ogni View, deve corrispondere un diverso Controller.

#### 7.1.1.2 Applicabilità

### 7.1.2 Dependency injection

## 7.2 Design pattern creazionali

### 7.2.1 Singleton

### 7.2.2 Strategy

## 7.3 Design pattern strutturali

### 7.3.1 Facade

## 7.4 Design pattern comportamentali

### 7.4.1 Observer

# 8 Mockup dell'interfaccia grafica