Architecture Design

November 2022

1 Architecture Design

1.1 Architectural Approach

In this project, our group choose to design the UWC2.0 by using MVC pattern, as shown in Figure 1. There,

- View or User Interface takes responsibility for user to make contact with system. The contact from user becomes request and is sent to the controller. Otherwise, View takes another responsibility to render the content from Controller.
- Controller plays the most important role in the system. It will receive request passed from View, retrieve the right data, and return it back to the view. The major task here is that Controller needs classify requests and determine what to return. It is our responsibility to make Controller work properly.
- Model specifies how data is structured, and updates the view. More importantly, it will be the main part that requests and receives directly from Database. Model might encompass business logic, data abstraction, etc. and be used to design Database first.

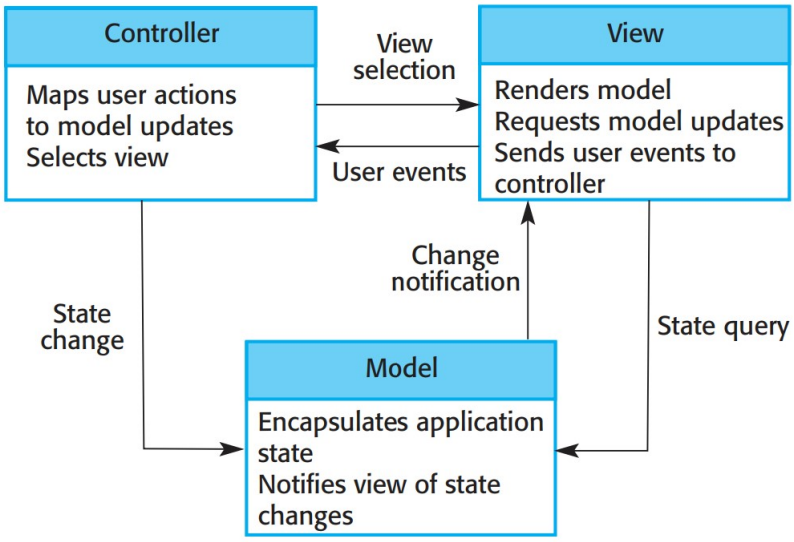

Figure 1: The MVC Architecture

The reason that our group setup this architecture is that:

- Easy to understand and implement. This architecture divides functionality clearly into parts, each has its own mission.

- Because of division in architecture, the system is easy to debug and maintain. Coding will have reusability, and avoiding hardcode is one of advantage.
- This architect the most common.

Modules

Name	Communicate
Description	The module transfers notifications between employees, back officers, and MCPs. It can also conduct chat sessions between system users.
Functions	`CreateSession` (UID, UID): bool `Notify` (UID, event_type): bool `SendMessage` (UID, message): bool

Name	Access Control
Description	The module performs user entry authentication by matching correct username and authenticate password
Functions	`Match` (username): employee `Authenticate` (password): bool `Login` (username, password): bool `SignOut` (UID): bool

Name	Vehicle Management
Description	The module manages vehicle
Functions	`CreateVehicle` (ID, capacity, load, location, type): vehicle `Update` (vehicle): bool `Edit` (vehicle): bool `Delete` (vehicle): bool `QueryVehicles` (): vehicle[ ]

Name	Employee Management
Description	The module manages employee information
Functions	`CreateEmployee` (ID, name, username, password, age, sex, salary) : employee `Update` (employee): bool `Edit` (employee): bool `Delete` (employee): bool `QueryEmployees` (): employee[ ]

Name	Map
Description	The module manages a grand map containing all MCPs, the depot, and the treatment plant of the company. It can perform changes to both MCPs and Routes on the map.
Functions	`DisplayMap` (MCPs): void `CreateMCP` (ID, location, capacity, load = 0): MCP `DeleteMCP` (ID): bool `CreateRoute` (MCPs): route `EditRoute` (route): bool `DeleteRoute` (route): bool `UpdateRoute` (route): bool `MatchRoute` (MCPs): route

Name	Task Management
Description	The module manages the schedule and area/route of employees
Functions	`LayoutSchedule` (shifts): void `AddShift` (shift): bool `EditShift` (shift): bool `DeleteShift` (shift): bool `MatchRoute` (MCPs): route `CreateRoute` (MCPs): route `AssignRoute` (route, UID): bool `LayoutRoute` (route): void

1.2 Implementation Diagram

Illustrated in Figure 2 is the implementation perspective of the task assignment module, enclosed in an MVC architecture. The `view` corresponds directly to a back officer UI. It includes options to assign tasks to an employee. Specifically, if one enters an employee's profile, they can choose to view and edit either that person's `schedule` or `work location`, which is a `route` or `area` depending on their role. The back officer can otherwise choose the `map` tab and fiddle with routes stored in the system.

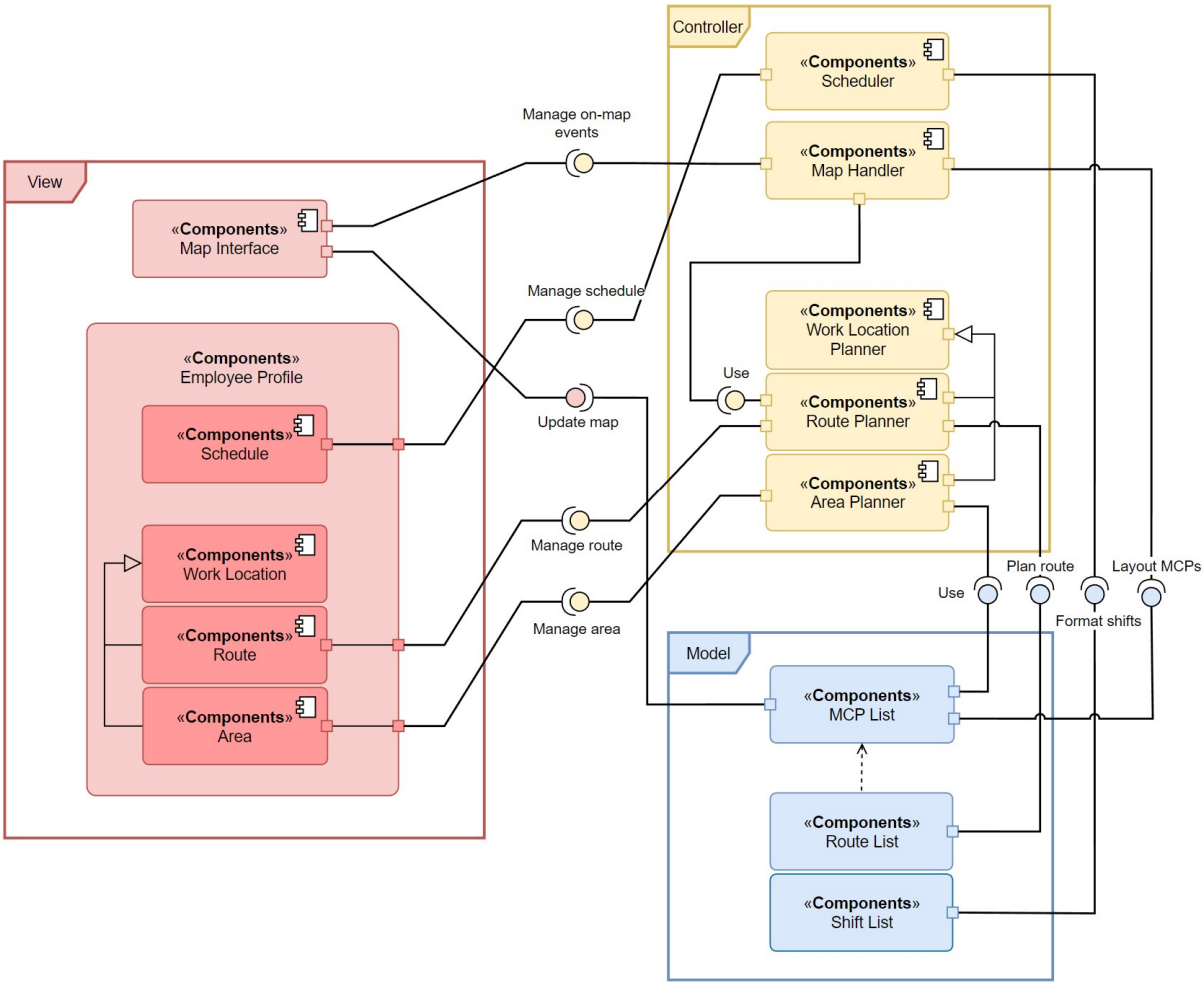

Figure 2: Task Assignment Implementation Diagram

In the diagram, components in `view` are handled by controller components, in which `Map Handler` is allowed to use `Route Planner` to aid route management without specifying any employee ID. It is designed so that the map interface can not mess with working areas of janitors. This is because each area is associated with a janitor location which is not displayed by the map. The controller performs its functionality using data provided by the database.