## **Design Goals & Principles**

**Modularity**

The architectural design must compose of replaceable and self-contained assembly of components to aid the process of implementation and future maintenance. The Model-View-Controller divides the large aggregation of components into units, ensuring loose coupling and high internal cohesion between components between layers. Modules within each layer are grouped according to its behaviour and purpose, for example, modules in the controller layer ensures communication is effected through the view and model layers, and demonstrates the concept of abstraction in the design. Each interface is deployed using different frameworks, each handling different interface design details. This allows easy change in the hidden interface design details without affecting its functionality as a system.

**Portability**

The architectural design must allow portability and reuse by employing abstract context interfaces. The application’s interfaces with the platform are separated from the computations and are achieved using a standard model of platform interface. For example, communication APIs such as Facebook, Twitter and SMS are independent of the platform, and can be reused.

**Malleability**

The architectural design must support adaptation to changing end-user requirements, and is highly modifiable and flexible. New components can be added in the respective layer without causing interference to the other layers. Also, obsolete components can be removed or updated using this environment.

**Conceptual Integrity**

## The architectural design must maintain conceptual integrity with a uniform application of limited number of design forms. The layered architecture is most suited to the system’s requirements and chosen for easier modifiability and assembly of different components. This approach ensures the modules are more independent and parallel and the system does not compromise under circumstance that requires upgrade, removal or insertion of a particular module, thereby offering reliability in the crucial crisis management system.

## **Candidate Architecture Analysis**

## The selected candidate architecture is the Model-View-Controller (MVC) architecture, reasons being:

## **MVC architectural style allows the separation of concerns during development phases.** It breaks the software components into three main categories, namely model, view and controller. The model is the part that handles the logic for the application data. It formats the data from database into an object by following strict rules of conversion. Incoming data from client side is parsed before it is saved into database, improving the data correctness and reliability. The view is the presentation of the data that model object contains, and whenever the data in the model object is changed, it will generate the view’s presentation of the model based on changes in the model. The controller is the part connecting the model and view and controls communication between model and view. It controls the data flow into model object and updates the view when there is any change of the data in model object. Thus, it promotes the reusability of business logic and template across the application.

1. **MVC architectural style allows parallel development process among team members.** MVC architectural style suggests the development of logic, view and model separately, ensuring code reusability. It allows all team members to build logic and design templates independently without affecting other progress and commitment. For example, one programmer can implement the view part while the other one creates the controller. Any delay caused by user interface design will not affect the progress of logic design as the processes are carried out independently. Development process is faster compared to other design patterns which don’t allow parallel development
2. **In MVC, modifications does not affect the entire model**. This is because Model and View parts are independent of each other. Any changes in the Model will remain isolated and therefore won’t affect the rest of the architecture. For example, any changes in logic has little effect to the user interface design as long as data is formatted according to models. This also means that any changes in one will not lead to multiple revisions to the other.

**Alternative Architectures**

|  |  |  |  |
| --- | --- | --- | --- |
| **Architecture** | **Description** | **Trade-offs** | |
| **Shared Data** | Data is shared between components through shared storage. The computational components are coordinated, with subroutines to a main program sequences through them. Data is then communicated between the components through shared storage. Communication between the computational components and shared data is an unconstrained read-write protocol. | **Advantages:**   1. Space-efficient 2. Time-efficient 3. Easy to understand since computational aspects are in different modules.   **Disadvantages:**   1. Changes in data storage format affects all modules, for example if the data structure is changed, then all the modules have to be modified to adjust to that change. Therefore it is not ideal for the CMS. 2. Changes in algorithm not well supported. 3. Poor stability if access to shared data is not handled properly. 4. Not supportive of reuse due to dependency of shared data. 5. Enhancements not easily incorporated. | |
| **Abstract Data** | Data type is defined by its behavior from user of data’s point of view, specifically in terms of values, operations and behavior of these operations. | **Advantages:**   1. The implementation of the ADT is encapsulated in the system which provides an friendly interface for users. This way, users don’t need technical knowledge of how it works to use it. 2. The implementation and the usage of ADT are two separate entities. Therefore, changes are localized in one place. For example, the implementation of ADT might change, the usage don’t need to change because the implementation comply to the agreed interface. 3. Different ADT but provides the same interface, which includes properties and abilities, are used interchangeably. This make the system more flexible.   **Disadvantages:**   1. Changes in implementation of ADT must comply to the interface between ADT and users, making it hard to modify implementation. It is not the select candidate architecture since any changes during the development process will be restricted. 2. Advanced technology in ADT users are being held back by the implementation of ADT which users have no knowledge about. 3. Not supportive for reusable code since implementation and usage are separate entities. | |
| **Pipe-and-filter** | The pump or producer is the data source.  The filter transforms or filters the data it receives via the pipes with which it is connected.  The pipe is the connector that passes data from one filter to the next. It is a directional stream of data. The sink or consumer is the data target. | **Advantages:**   1. Simple and easy to understand. 2. Easily maintainable and enhanceable. 3. Reusable 4. Supports concurrent execution.   **Disadvantages**:   1. Not good at handling interactive applications. Since the CMS is interactive, this architecture is not ideal. 2. When a filter needs to wait until it has received all data , its data buffer may overflow or cause a deadlock. 3. If the pipes only allow for a single data type, the filters will need to perform parsing. Creating different pipes for different datatypes means you can't link some of the pipes to some of the filters. | |
| **Layered** | Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application. Data moves from one defined level of processing to another. | **Advantages:**  1. Support designs based on the increasing levels of abstraction.  2. Support enhancement of functionality by introducing new components.  3. Support reuse of lower layer components while upper layer varies.  **Disadvantages:**   1. Not all the systems are easily structured in layered fashion. 2. It is not easy to find the right level of abstraction. It is difficult to define the different layers of the system, hence it is not the selected candidate architecture for the CMS. 3. It has lower performance. | |

## 

## **Scenario Evaluations**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scenario** | | | **Modification** | |
| **No.** | **Description** | **Type** | **Component** | **Change** |
| 1 | Insertion of new authorised agencies into the system. | Indirect | Create/Update Incident Info | To change the implementation according to the new agency. |
| Send dispatch command. |
| 2 | Addition of a new social platform. | Indirect | Update Social Media | To add in the respective social media’s SDK and its implementation in the system. |
| 3 | Change the type of outbreak the CMS can display. | Indirect | Create/Update Incident Info | To change the implementation according to the type of outbreak added. |
| Provide Information on Map |
| Input |
| 4 | Change of the current map due to updates (e.g. new roads) | Indirect | Update Map | To change the implementation according to the new map. |
| 5 | Change of the layout of the report generated. | Indirect | Generate Status Report | To change the implementation according to the new layout. |
| 6 | Add new status of the incident (ie, pending, confirmed, crisis, solved) | Indirect | Input | To change the implementation according to the new status. |
| Create/Update Incident Info |
| Update Map |
|
| Sort Incident |
| 7 | Include a time window for the operator to inform the general public via social media and SMS | Indirect | Create/Update Incident Info | To change the implementation according to  the new timer. |
|
|
|
| Update Social Media |
| 8 | Add new locations of CD shelters | Indirect | Provide Information on Map | To change the implementation according to the new location details. |
|

## **Evaluation of CMS Architectures**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Scenarios** | | **Type of Architecture Pattern** | | | |
| **No.** | **Weight** | **Shared Data** | **Abstract Data** | **Pipe and Filter** | **Layered** |
| **1** | 15 | 5/8 | 4/12 | 3/9 | 1/5 |
| **2** | 5 | 4/8 | 2/12 | 1/9 | 1/5 |
| **3** | 20 | 5/8 | 5/12 | 2/9 | 1/5 |
| **4** | 20 | 5/8 | 2/12 | 2/9 | 1/5 |
| **5** | 5 | 1/8 | 1/12 | 1/9 | 0 |
| **6** | 10 | 4/8 | 5/12 | 3/9 | 3/5 |
| **7** | 15 | 4/8 | 5/12 | 3/9 | 1/5 |
| **8** | 10 | 3/8 | 3/12 | 2/9 | 1/5 |
| **Total (Modification/cost)** | | 53.75 | 30.83 | 25.55 | 23 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute Architecture** | **Shared Data** | **Abstract Data** | **Pipe and Filter** | **Layered** |
| Change in data processing algorithm | **-** | **-** | **+** | **+** |
| Change in data representation | **-** | **+** | **+** | **+** |
| Change in function | **-** | **-** | **+** | **-** |
| Performance | **+** | **+** | **-** | **+** |
| Reuse | **-** | **+** | **+** | **+** |

Based on the quantitative approach of evaluating the CMS Architectures, we can see that the Shared Data architecture yield the highest in cost and modification to the program, making it the least desirable architecture pattern to follow due to poor modifiability in handling processing algorithms, data representations and reuse. The most promising architecture is the layered architecture, therefore we have chosen that architecture pattern amongst the rest. Albeit Pipe and Filter features promising results with its use of independent components, the layered architecture will be able to support better enhancement in functionality with the ease of introduction of new components without affecting other layers, making it score better in reliability and maintainability.