Project Assignment 3

Rui Zhu, Lingjing Huang, Entisar Seedi M Alshammry

1. What is software architecture? Why is software architecture needed? (10 points)

1) An architecture is the set of significant decisions about **the organization of a software system**, **the selection of the structural elements** and their **interfaces by which the system is composed**, together with their **behavior as specified in the collaborations** among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization these elements and their interfaces, their collaborations, and their composition.

2) Extensibility, Changeability, Simplicity, Efficiency

Software architecture represents a common abstraction of a system that most if not all of the system's stakeholders can use as a basis for mutual understanding, negotiation, consensus, and communication.

1. Summarize each of the common architecture styles along with its description as given in case study. Point out some of the shortcomings of each architectural approach. (40 points). At least half a page per architecture.
2. **Model-View Controller**

Description:

Model–view–controller is an architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts: Model, Viewand Controller. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.

Model is a synonym for the domain layer of objects. View is a synonym for UI objects, such as windows, Web pages, applets, and reports.

The Model-View Separation Principle has at least two parts:

1. Do not connect or couple non-UI objects directly to UI objects. For example, don't let a Sale software object (a non-UI "domain" object) have a reference to a Java Swing JFrame window object. Why? Because the windows are related to a particular application, while (ideally) the non-windowing objects may be reused in new applications or attached to a new interface.

2. Do not put application logic (such as a tax calculation) in the UI object methods. UI objects should only initialize UI elements, receive UI events (such as a mouse click on a button), and delegate requests for application logic on to non-UI objects (such as domain objects).

Shortcoming:

(1) Increase the system architecture and implementation complexity. For a simple interface, strictly abide by the MVC, to make the model, view and controller separation, will increase the complexity of the structure, and may produce too much of the update operation, reduce the operation efficiency.

(2) The view and the controller of the close connection between. The view and the controller is separated from each other, but really connected components, view without the existence of the controller, its application is very limited, and vice versa, thus blocking their independent reuse.

(3) In view of low efficiency model of data access. According to the model, the different operation interface, view may require multiple calls to get enough to display data. The frequent access unnecessary not change data, it will also hurt the performance.

(4) At present, general advanced interface tools or constructor does not support MVC mode. Transformation of these tools to meet the MVC needs and the establishment of separate parts of the cost is very high, thus causing trouble using the MVC.

**2) Data flows**

Description:

In data flow architecture, the whole software system is seen as a series of transformations on consecutive pieces or set of input data, where data and operations are independent of each other. In this approach, the data enters into the system and then flows through the modules one at a time until they are assigned to some final destination (output or a data store).

The connections between the components or modules may be implemented as I/O stream, I/O buffers, piped, or other types of connections. The data can be flown in the graph topology with cycles, in a linear structure without cycles, or in a tree type structure.

The main objective of this approach is to achieve the qualities of reuse and modifiability. It is suitable for applications that involve a well-defined series of independent data transformations or computations on orderly defined input and output such as compilers and business data processing applications. There are two types of execution sequences between modules: Batch sequential and Pipe and filter or non-sequential pipeline mode.

In the batch sequential style, processing steps, or components, are independent programs, and the assumption is that each step runs to completion before the next step starts. Each batch of data is transmitted as a whole between the steps. The typical application for this style is classical data processing.

The Pipe-and-Filter style emphasizes the incremental transformation of data by successive components. This is a typical style in the UNIX family of operating systems.

Shortcoming:

For Pipes and filters, It has some of the following disadvantages

1.Not suitable for dynamic interactions.

2.A low common denominator is needed for transmission of data in ASCII formats.

3.Overhead of data transformation between filters.

4.Does not provide a way for filters to cooperatively interact to solve a problem.

5.Difficult to configure this architecture dynamically.

Does not provide concurrency and interactive interface rather it provides high latency and low throughput. Further, external control is required for the implementation.

1. **Independent components**

Description:

Shortcoming:

1. **Virtual machines**

Description:

Virtual Machine architectures have the goal of achieving the quality of portability.

Virtual machines are software styles that simulate some functionality that is not native to the hardware and/or software on which it is implemented.

It can allow one to simulate (and test) platforms that have not yet been built (such as new hardware), and it can simulate "disaster'' modes (as is common in flight simulators and safety-critical systems) that would be too complex, costly, or dangerous to test with the real system.

Common examples of virtual machines are interpreters, rule-based systems, syntactic shells, and command language processors.

Interpretation of a particular module via a Virtual Machine may be seen as follows:

1.the interpretation engine selects an instruction from the module being interpreted;

2.based on the instruction, the engine updates the virtual machine internal state;

3.the process above is repeated;

Shortcoming:

there is a performance cost because of the additional computation involved in execution.

1. **Layered architectures**

Description:

Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database. In some cases, the business layer and persistence layer are combined into a single business layer, particularly when the persistence logic (e.g., SQL or HSQL) is embedded within the business layer components. Thus, smaller applications may have only three layers, whereas larger and more complex business applications may contain five or more layers.

Each layer of the layered architecture pattern has a specific role and responsibility within the application. For example, a presentation layer would be responsible for handling all user interface and browser communication logic, whereas a business layer would be responsible for executing specific business rules associated with the request. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request. For example, the presentation layer doesn’t need to know or worry about how to get customer data; it only needs to display that information on a screen in particular format. Similarly, the business layer doesn’t need to be concerned about how to format customer data for display on a screen or even where the customer data is coming from; it only needs to get the data from the persistence layer, perform business logic against the data (e.g., calculate values or aggregate data), and pass that information up to the presentation layer.

One of the powerful features of the layered architecture pattern is the separation of concerns among components. Components within a specific layer deal only with logic that pertains to that layer. For example, components in the presentation layer deal only with presentation logic, whereas components residing in the business layer deal only with business logic. This type of component classification makes it easy to build effective roles and responsibility models into your architecture, and also makes it easy to develop, test, govern, and maintain applications using this architecture pattern due to well-defined component interfaces and limited component scope.

Shortcoming:

1. There might be a negative impact on the performance as we have the extra overhead of passing through layers instead of calling a component directly.

2. Development of user-intensive applications can sometime take longer if the layering prevents the use of user interface components that directly interact with the database.

3. The use of layers helps to control and encapsulate the complexity of large applications, but adds complexity to simple applications.

4. Changes to lower level interfaces tend to percolate to higher levels, especially if the relaxed layered approach is used.

1. **Repository**

Description:

In Repository Architecture Style, the data store is passive and the clients (software components or agents) of the data store are active, which control the logic flow. The participating components check the data-store for changes.

A client sends a request to the system to perform actions (e.g. insert data). The computational processes are independent and triggered by incoming requests. If the types of transactions in an input stream of transactions trigger selection of processes to execute, then it is traditional database or repository architecture, or passive repository. This approach is widely used in DBMS, library information system, the interface repository in CORBA, compilers, and CASE (computer aided software engineering) environments.

System has 2 components:

1. Central data unit component.

2. Set of relatively dependent components.

• Central data unit called the repository shares

information with all the other units.

• There are differences in the information exchange

patterns.

• Thus there are 2 main control strategies to deal with

these information exchange patterns.

Shortcoming:

Because of being more vulnerable to failure and data replication or duplication, Repository Architecture Style has following disadvantages −

High dependency between data structure of data store and its agents.

Changes in data structure highly affect the clients.

Evolution of data is difficult and expensive.

Cost of moving data on network for distributed data.