2. Background

Contents

MeDIC and XAM system are the example of the systems, which based on the proposed architecture. These two systems are currently running, so the experiment of this research work will not involves with these systems. Instead, another system is introduced as, an existing system or a prototype. From this point, the existing system refers to the Customer and Contract Management System, a basic CRUD application, represents the relationship between Customer and Contract object types. The existing system and related technology are introduces in this chapter.

2.1. Existing System and Related Technology

The Customer and Contract Management System is a system represents the relationship between customer to contracts (One-to-Many) and contract to customer (One-to-One). For instance, the user can interacts with the customers and contracts with basic CRUD operations (Create, Retrieve, Update, Delete). User can navigates to a customer profile page and assigns multiple contracts to that customer or other way around, navigates a contract information page and selects a customer to bind to the contract [Figure 2.1].

The ModelRoot page is the main page of the system. It contains tables displaying list of customers and contracts. User can view the detail or delete any customer or contract object by navigates "delete" or "detail" link exists on each row. Also, under each table, there is a link for customer or contract creation.

Either navigate through the "detail" link or finishes the customer/contract creation successfully, user will be navigated to the customer/contract detail page. In the detail page, user can change the object's data including manage the re-

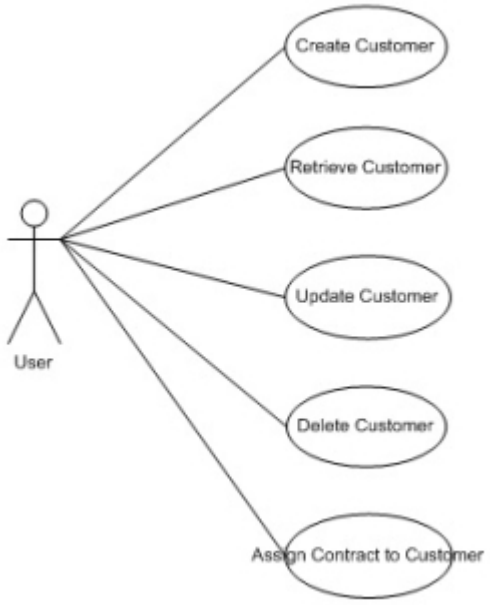

Figure 2.1.: Existing system's Use Case Diagram (part1)

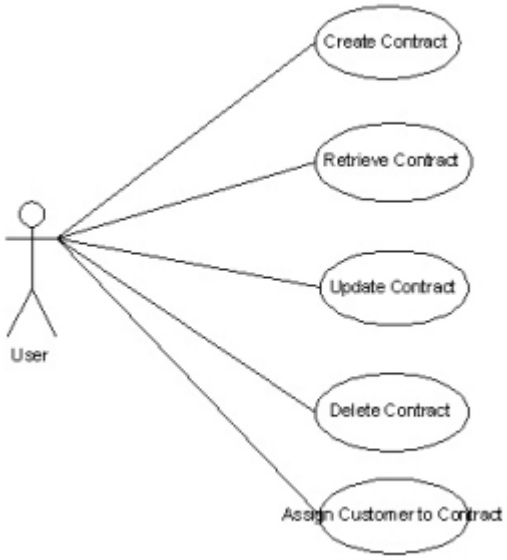

Figure 2.2.: Existing system's Use Case Diagram (part2)

lationship between customer and contract (assign). The user will be redirected back to the main page only when the "back" button clicked.

The architecture of the existing system shared similarities with the MeDIC system, XAM system, and other general web applications based on J2EE. Follow the J2EE approach, the system divides into 3 layers: Presentation layer, Business Logic layer, and Data layer.

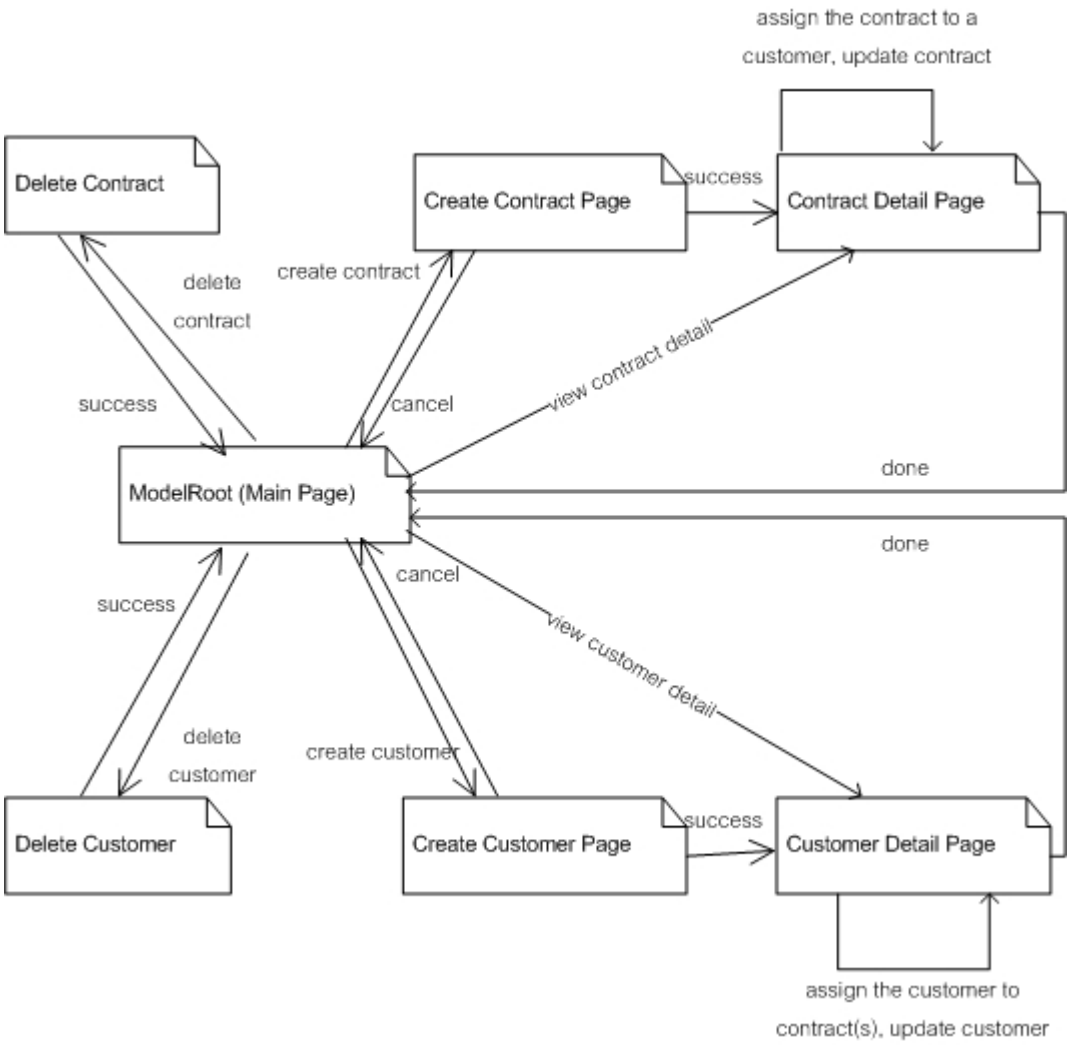

Figure 2.3.: Page Flow Diagram

The presentation layer composed with plain JSPs and Servlets. The request from the browser will be centralized to the Action Handler Servlet and delegates to the corresponding Action class. The Action class communicates with the Application Facade in the Business logic layer.

The Business Layer and Data layer are managed by EJB framework. The Application Facade delegates the request sends by Actions in Presentation Layer to the corresponding Controller. The controllers are the interfaces to simplify the business logic in Management. The Entity persisted, and the results from the database render in JSP page in the presentation layer. [Figure 2.3].

On the client layer, the existing system uses the Command pattern to encapsulate the request parameters. All requests pass through the Action Handler then the Action Handler delegates the request to the corresponding Action class.

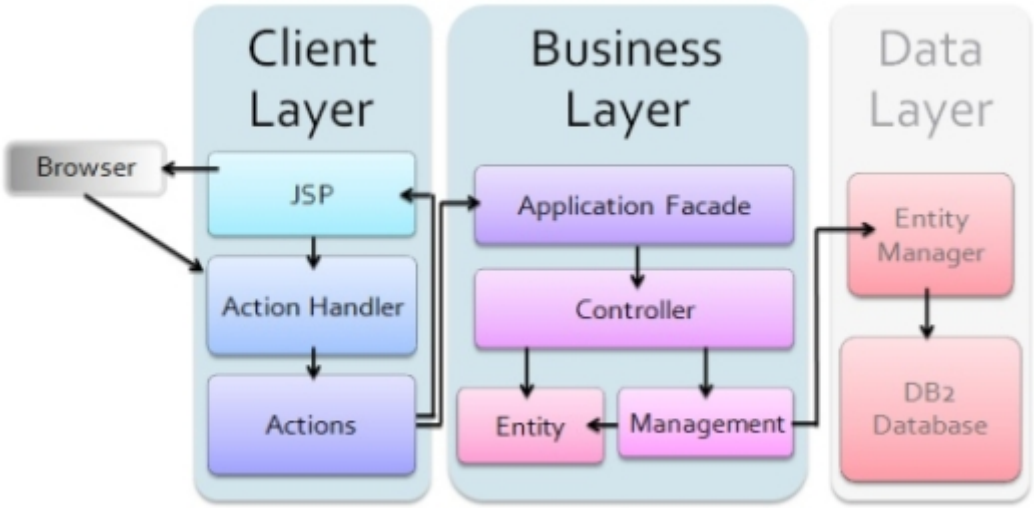

Figure 2.4.: Architecture diagram of the existing system

Both Action Handler and Actions are all ordinary Servlets built for request delegation and communication to the business layer. One example of the problem caused by this pattern is huge amount of Action classes since, for each user operation, one action class needs to be created. In consequence, at least ten action classes must be created per Entity (create, retrieve, update, delete, and assign actions), this will increase dramatically, if there are more object types in the system.

On the business layer, request and response are centralized through the only portal of the backend, the Application Facade. The Application Facade is a class, which provides a simplified interface to more complex business logic inside the Controller and Management modules.

2.1.1. Enterprise Java Beans (EJB)

In the proposed architecture, the whole backend is covered by Enterprise Java Beans architecture (EJB) specification, the server-side component architecture for Java platform of the enterprise application. EJB is one of J2EE specification, which enable rapid and simplified development of distributed, transactional, secure and portable applications.

Based on the existing system, the client layer communicates through Application Facade's Local or Remote interface. The corresponding Controller delegates the data to the responsible Management, which contains persistence logic and responsible in data layer communication. It does not matter how the system is distributed or clustered, EJB handles all the complexity, such as internal communication, transactional integrity, persistence, security, and deployment

[Figure 2.4].

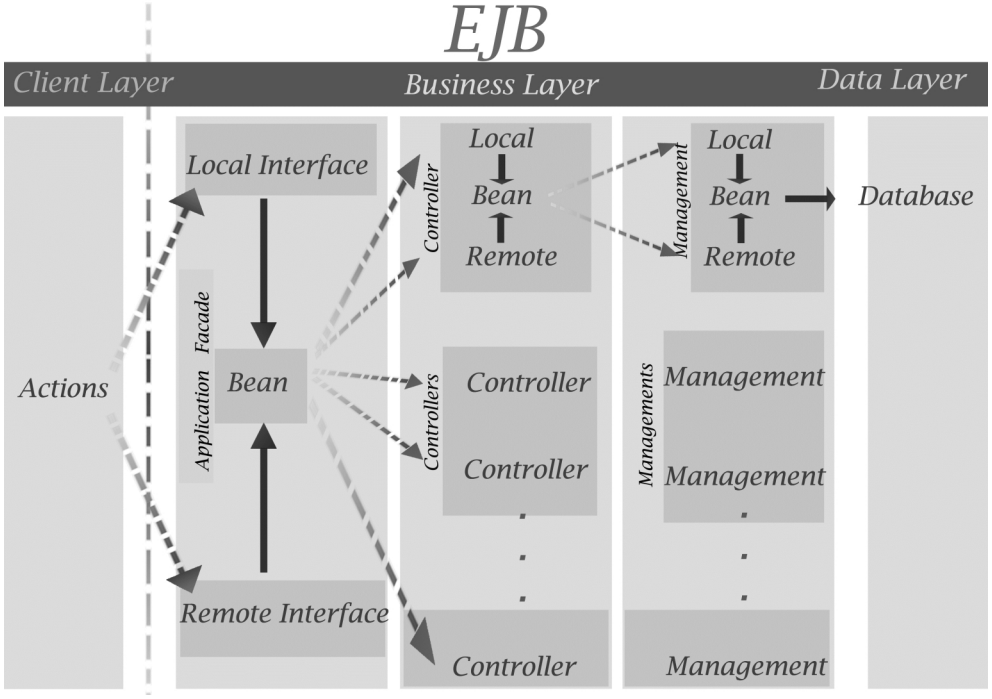

Figure 2.5.: Communication flow of EJB application

2.1.2. IBM Websphere Application Server 7.0 (WAS)

IBM Websphere Application Server (WAS) is an application server under IBM's Websphere brand [IBM11]. WAS is built using many open-standards such as Java EE, XML, and Web Services. WAS helps drive business agility by providing millions of developers and IT architects with an innovative, performance-based foundation to build, reuse, run, integrate and manage Service Oriented Architecture (SOA) application and services. WAS runs applications and services in reliable, highly available, scalable environment to ensure business opportunities are not lost due to application downtime.

WAS's history begins with version 6.1 and now into version 8, in this master thesis report, version 7.0.0.9 considered as the existing system environment and one of the main constraint for prototype development. In this version 7, WAS simplifying the adoption of new standards such as Service Component Architecture (SCA).

The development tool, which design specifically for working with WAS is the Rational Application Developer (RAD) [WCF+07], an IDE based on Eclipse IDE. RAD is also the main development tool of this master thesis.

2.1.3. Model-View- Controller pattern (MVC)

The MVC pattern or MVC architecture is one of the most commonly used web architecture [HSD10]. The reason that MVC pattern mentioned in this research paper is the existing system and most of the web frameworks are based on the MVC pattern.

The MVC architecture isolates domain logic into three tiers: model, view, and controller. Each tier responsible in different domain as following:

- The model represents enterprise data and business rules to access and update the data. Database system is also a part of the model.
- The view renders the content of the model. It is the view's responsibility to maintain the consistency in its presentation when the model changes. Normally, the view never communicates with the model directly, but through the controller. However, in some special cases, the view allows to communicate with the model directly, for example, in mobile application, where the resources is limited.
- The controller translates the interaction with the view into actions to be performed by the model. The actions performed by the model include activating business processes or changing the state of model. Based on the user interactions and outcome of the model actions, the controller responds by selecting the appropriate view.

Based on [Figure 2.5], these following steps are the data flow:

1. The user interacts with the user interface (ex. press the button).
2. The controller handles the request from user interface, then converts request into the appropriate action.
3. The controller notifies the model of the user action and sends the change in model state.
4. The model executes the transaction and return the result (date) back to the controller.
5. The controller sets the data to render at the view and response back to the user with the requested view.

Unlike Command and Facade pattern, MVC is a architecture pattern, while Command and Facade are design pattern. However, by taking the advantage of existing patterns, MVC can be implemented using Strategy, Composite, Observer, and Command design pattern of Gang of Four [FRBS04].

The most remarkable benefits of the MVC architecture is the clear separation

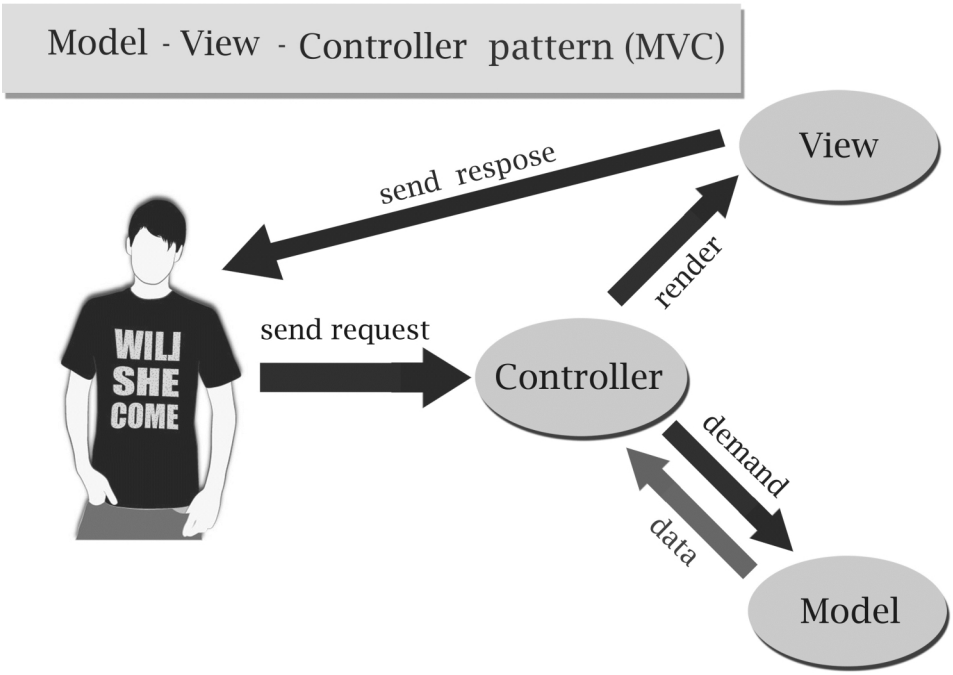

Figure 2.6.: Communication flow of MVC architecture

of each layer. In consequences, the developer can distributed development effort to some extent, so the implementation changes in one part of the system do not require changes to another. For instance, the system may have multiple views sharing the same model, which is easier to maintain , test, and upgrade the multiple system. To add new client, only adding view and controller is necessary. Since the model is completely decoupled from views, it allows lot of flexibilities to design and implement the model considering reusability and modularity. This model also can be extended for further distribution application, which makes the system extensible and scalable.

2.2. Framework Popularity and First Frameworks Selection

Nowadays, many presentation development frameworks, such as Java Server Faces (JSF), Wicket, and Tapestry, exist to cover the Presentation Layer. They provide variety of features and architecture enhancements. First step, is to narrow-down the scope of the focused framework based mainly, on framework popularity. The popularity and number of user can determine quality, effectiveness, and assured that the framework has enough qualification to pay attention and considered as one of the research experiment.

Based on the statistic from Google Trends [Rai11b] and Zero Turnaround's Java