**CHAPTER 3**

**DESIGN OF NEW SYSTEM**

In this chapter we will go through the design of new system which will solve the problems discussed in the second chapter.

Analysis is important in this chapter, because to provide a system which satisfy customer’s needs, requires a good analysis. Also we have to understand that every analysis does not fulfill customer’s needs. To provide systems which satisfy all necessary needs, we need to take enough time to think about the problems of existing system and how the new system will provide solutions.

As definition, analysis

As definition, analysis is concerned with the primary abstractions (classes and objects) and mechanisms that are presents in the problem domain. After analysis, we need to provide a design which helps us to illustrate in technical manner the result of analysis.

Design is defined as the result of the analysis expanded into a technical solution. To achieve all necessary requirements of analysis and design of new system, we will use UML (Unified Modeling Language).

What is the UML? In short, the Unified Modeling Language (UML) provides Industry standard mechanisms for visualizing, specifying, constructing, and Documenting software systems.

UML provides a language for describing system interactions, supported by a cohesive (united) set of definitions managed by an official group, the Object Management Group (OMG).

Those are the important goals of UML:

* To model systems (and not just software) using object-oriented concepts
* To establish an explicit coupling to conceptual as well as executable artifacts
* To address the issues of scale inherent in complex, mission-critical systems
* To create a modeling language usable by both human beings and machines

**Analysis and Design Methodology**

Before UML, many approaches, called methods, were developed that attempted to inject engineering principles into the craft of software engineering.

The most worthwhile engineering techniques are those that can be described both quantitatively and qualitatively, and used consistently and predictably.

The method must achieve better results than using no method, must be able to be applied to many different problems, and must be able to be taught to others relatively easily.

The multitude of different methods, each with its own unique notation and tools, left many developers confused and unable to collaborate. The lack of a well-established notation upon which creators of many methods and tools could agree made it more difficult to learn how to use a good method.

There are many methods like Data Structure-Oriented, Object-Oriented, Prototyping, etc. but in our case we will focus on Object-oriented methodology(OOM) for producing models that reflect a domain, such as a business domain or a machine domain, in a natural way, using the terminology of the domain.

With Object-oriented methodology classes and objects are usefully.

An object is an entity that has a well-defined role in the application domain and has state (collection of values for its **variables**), behavior (capability to execute **methods**), and identity (distinguished instance of a **class**). An Object can be a tangible or visible entity (e.g., a person, place, or thing).

A class is a set of objects that shares a common structure and a common behavior.

Object-oriented methodology has characteristics known as tree pillar:

**Data encapsulation:** with this characteristic information is hidden and we don't need to know how some components are implemented to use them. Also Implementation can change without affecting any calling code.

**Inheritance:** is a code re-use issue, we can extend code that is already written in a manageable manner. Inheritance is more; it supports polymorphism at the language level.

With inheritance we can take an existing object type (collection of fields and methods) and extend it:

* Create a special version of the code without re-writing any of the existing code (or even explicitly calling it!).
* End result is a more specific object type, called the sub-class / derived class / child class.
* The original code is called the super class / parent class / base class.

**Polymorphism:** Create code that deals with general object types, without the need to know what specific type each object is. In other words, multiple forms can be applied on one method.

Here are some examples of polymorphism:

Example 1: Generate a list of employee names:

* All objects derived from Employee have a name field.
* No need to treat managers differently from anyone else.

Example 2: Shape objects types used by a drawing program.

We want to be able to handle any kind of shape someone wants to code (in the future).

We want to be able to write code now that can deal with shape objects (without knowing what they are).

With methods draw () we can get different shapes such as rectangle, triangle, circle etc. If each could be a kind of shape (could be specializations of the shape class) and each knows how to draw itself.

With Object Oriented methodology some concepts are most used. For better understanding we are going to explain more.

* **A class**



* **Relationships**

A relationship is an association between the instances of one or more entity types that are of interest to the organization.

The three most important relationships are: association, generalization and dependency.

* **Association**

An association is a relationship between two classes represented by a solid line. Associations are bi-directional by default, so both classes know about each other and about the relationship between them



* **Generalization**

The inheritance relationship, also known as the *generalization* relationship, is used to indicate that one class is a specialization of another.

* **Dependency**

A dependency is a relationship between elements, one independent and one dependent. A change in the independent element will affect the dependent element

* **Aggregation**

Aggregation is a relationship where one class is part of another class.

In basic aggregation, the class that forms part of the whole class can exist independently, so the life of an instance of the part class is not determined by the whole class.

Basic aggregation is represented using an empty diamond symbol next to the whole class

As example, a computer in a warehouse contains a motherboard, but although the motherboard disports of the computer, it can exist as a separate item.

* **Composition**

Composition is a strong type of aggregation where the whole class contains the instance of the part class.

The life time of the part class depends on the existence of the whole class

Composition relationships are represented using a filled diamond symbol next to the whole class

As example, a building contains a number of rooms and a room cannot exist without a building. If a Building class instance is destroyed, its Room class instances destroyed too

* **Multiplicity(Cardinality)**

The multiplicity is a number or a range that indicates how many instances of connected types can be involved in the relation.

* **Unified Process (UP)**

In tandem(combination of people) with their initial textbooks on the UML, the three most recognized UML leaders, Grady Booch, Ivar Jacobson, and James Rumbaugh, introduced the Unified Software Development Process (Booch, Rumbaugh, and Jacobson,1999) based on work done within a process group at Rational Software Corporation.

More commonly called the Unified Process; it is becoming recognized as a premiere process for developing complex systems when using UML. Each of these three methodologists already had his own processes; this process pulls together the best features from each and adds more industry-recognized best practices.

The Unified Process is really a macro process for development, aimed at both managers and technicians. Micro process activities organized around requirements, analysis, design, implementation, and testing are still present, but are placed in a larger framework (Structure) for producing commercial software.

We have to note that although this process is often used when developing object-oriented systems modeled with UML, we must stress that the process does not come from the OMG, nor is it the only process available for UML modelers.

**Design of the New System**

**Requirements Analysis**

The requirements analysis captures the functional and non-functional requirements of the system to be produced. With requirements analysis we will focus on use case diagram.

Use case diagram contains elements for the system, the actors, and the use cases, and displays the different relationships between these elements. Use case diagram shows a number of external actors and their connection to the use cases that the system provides.

* **Actor**

Actor is someone or something external that interacts with the system or subject. An actor can be a role played by a human being or another system.



* **Use case**

A use case is a description of a functionality (a specific usage of the system) that the system provides or a set of scenarios that describing an interaction between a user and a system, including alternatives.



* **Relationships**

Relationships are made between actors and use cases. And it uses an association symbol.



* **System boundary**

A set of scenarios that describing an interaction between a user and a system, including alternatives**.**



* **Use case description**

Use case description contains the complete and general description; a number of actual scenarios are used to illustrate what happens when the use case is instantiated. The scenario (events) description illustrates a specific case, with both the actors and the use cases involved as actual instances. Customers can better understand a complex use case when a more practical scenario describes the behavior of the system.

* **Title:** a use case name
* **Goal:** what a system plans to do
* **Actor:** the actor participated in the use case
* **Precondition(s):** state of the system before the use case starts
* **Post condition(s):** state of the system when the use case terminates
* **Main normal flow:** the actual steps of the use case
* **Alternative flow:** steps which can exist in case a normal flow fails.

**Figure 4**: Use case Diagram for online forex bureau exchange management system

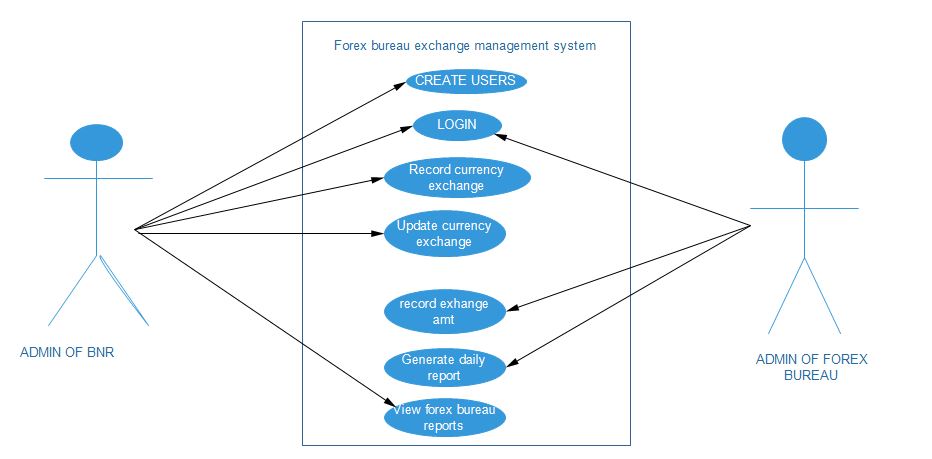


Table 1: Creating users use case descrition

|  |
| --- |
| **Name:** creating a users |
| **Actor:**  employee of the BRN/ owners of the forex bureau |
| **Description:** this will help to create a new user of the system. |
| **Pre-condition:** administration have the privilege to access the system |
| **Post-condition:** the system give the user access on data |
| **The Main Normal flow:**   1. Admin logs in the system with the correct credentials 2. The system provides the page to create a new user 3. Admin create the users of the system. 4. The user have username to login the system |
| **Alternative flow:**   1. **A.**  if the pastor or cashier entering the incorrect username 2. the system inform the user to enter the registered user name |

Table 2: Login users use case descrition

|  |
| --- |
| **Name:** login the system |
| **Actor:**  BNR/ forex bureau users |
| **Description:** this will help to login of the system. |
| **Pre-condition:** administration have the privilege to access the system |
| **Post-condition:** the system give the access on data |
| **The Main Normal flow:**   1. users of the system logs in the system with the correct credentials 2. The system provides the page to access the system. 3. The user have username to login the system |
| **Alternative flow:**   1. **A.**  if the BNR users or forex bureau users entering the incorrect username 2. the system inform the user to enter the registered user name |

Table 3: Updating currency exchange

|  |
| --- |
| **Name:** updating currency exchange |
| **Actor:**  Administrator of BNR |
| **Description:**  This will help to updating new currency exchange |
| **Pre-condition:** administration of the BNR have the privilege to access the system |
| **Post-condition:** the system give the user access on data |
| **The Main Normal flow:**   1. admin of BNR logs in the system with the correct credential 2. The system provides the page to updating the new currency. 3. Admin record today new currency 4. The user have username to login the system |
| **Alternative flow:**   1. **A.**  if admin of BNR entering the incorrect username or password 2. the system inform the user to enter the registered user name |

Table 4: record exchange amount

|  |
| --- |
| **Name:**  Record exchange amount |
| **Actor:**  Admin of the forex bureau |
| **Description:** this will help to record the exchange amount of every customers |
| **Pre-condition:** users of the forex bureau have the priviledge to access the system |
| **Post-condition:** the system give the user access on data |
| **The Main Normal flow:**   1. employee of the forex bureau logs in the system with the correct credentials 2. the system allow him/her to access the system 3. he fills / record all amount exchange from the customers |
| **Alternative flow:**   1. **A.**  if the employee of the forex bureau entering the incorrect username 2. the system inform the user to enter the registered user name |

Table 5: View report use case diagramm

|  |
| --- |
| **Name:** view report |
| **Actor :** Admin of BNR/ Forex bureau exchange |
| **Description :** this allow BNR / FOREX BUREAU to view the activities report |
| **Pre-condition:** the users must be login into the system |
| **Post-condition:** the report is read. |
| **Main normal flow:**   1. the system users logs into the system 2. users access the system and view all reports 3. the users generate the total currency exchange from difference services 4. the system generate report accordingly to the daily you request 5. the system users record a report into databases |
| **Alternative flow:**   1. after view the report the system users choose another page |

Table 6: Logout use case description

|  |
| --- |
| **Name:** logout |
| **Actor:** all users |
| **Description :** **:** allows users to exit the system to prevent other people to access system without permission |
| **Pre-condition:** users being logged in the system |
| **Post condition:** the system display the homepage |
| **Normal flow:** logout   1. Admin and all users press the button of the logout button |

**Domain analysis**

Domain analysis helps identify entities of the system by illustrating meaningful conceptual classes in problem domain. Also it can be defined as the process of analyzing related software system in a domain to find their common and variable parts. Domain analysis can be called product line analysis. Domain analysis is the first phase of domain engineering. It is a key method for realizing systematic software reuse. To achieve the domain analysis we use a class diagram.

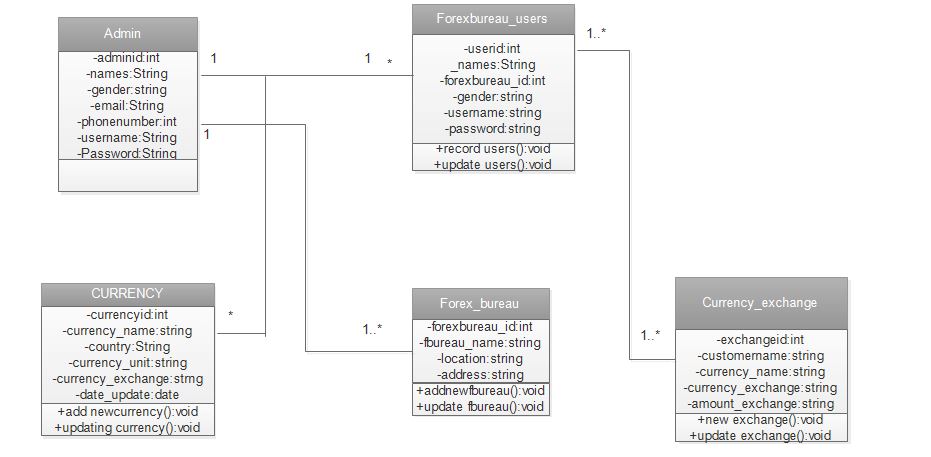
A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.

In the diagram, classes are represented with boxes which contain three parts:

* The top part contains the name of the class. It is printed in Bold, centered and the first letter capitalized.
* The middle part contains the attributes of the class. They are left aligned and the first letter is lower case.
* The bottom part gives the methods or operations the class can take or undertake. They are also left aligned and the first letter is lower case.

**Figure 5: Class Diagram for new forex bureau exchange management system**



**The system design has three main parts:**

**Client side components:**

This part combines **http clients** and **phone clients**. If a client send request with computer will use set of hardware and software such as computer, network tools and web browser as an interface of application server.

If client send request with a cell phone will use set of hardware and software such as cell phone and sms interface.

* **Server side components**

This part control requests from clients and sends or retrieve them into database. Because clients can send request on computers or cell phone, it combine set of hardware and software such as server computer, network tools, application server, and database

* **Database:** Is a store of all needing information on server side. in our case we choose oracle as the application database.

**Sequence diagram**

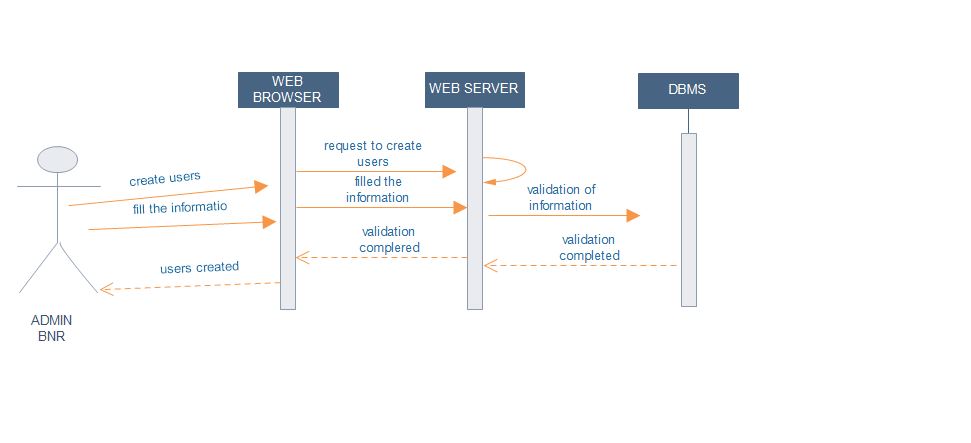
A sequence diagram shows a dynamic collaboration between a numbers of objects. The important aspect of this diagram is that it shows a sequence of messages sent between the objects. It also shows an interaction between objects, something that happens at one specific point in the execution of the system. The diagram consists of a number of objects shown with vertical life lines. Time passes downward in the diagram and the diagram shows the exchange of messages between the objects as time passes in the sequence or function. Messages are shown as arrows between the vertical life lines. Time specifications can be shown as constraints on the diagram. Comments can be added in a script in the margin of the diagram. A sequence diagram represents an interaction fragment. These fragments can take an operator in the upper corner that indicates any special handling for that section.

**The symbols are useful in sequence diagram:**

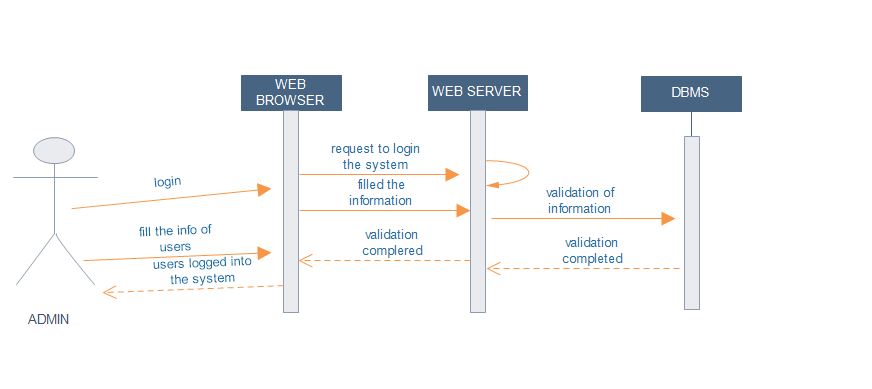
Table 7: View table of the element sequence of the diagram

|  |  |
| --- | --- |
| **Term and definition** | **Symbol** |
| **An actor:**   * It can be a person or system that derives benefit from and is external to the system. * It participates in a sequence by sending and/or receiving messages. * It is placed across the top of the diagram. |  |
| **An object:**   * It participates in a sequence by sending and/or receiving messages. * It is placed across the top of the diagram. |  |
| **A lifeline:**   * Denotes the life of an object during a sequence. |  |
| **An activation**:   * Is a long narrow rectangle placed atop a lifeline? * It denotes when an object is sending or receiving messages |  |
| **A message**:   * It conveys information from one object to another one. * A operation call is labeled with the message being sent and a solid arrow, whereas a return is labeled with the value being returned and shown as a dashed arrow. |  |

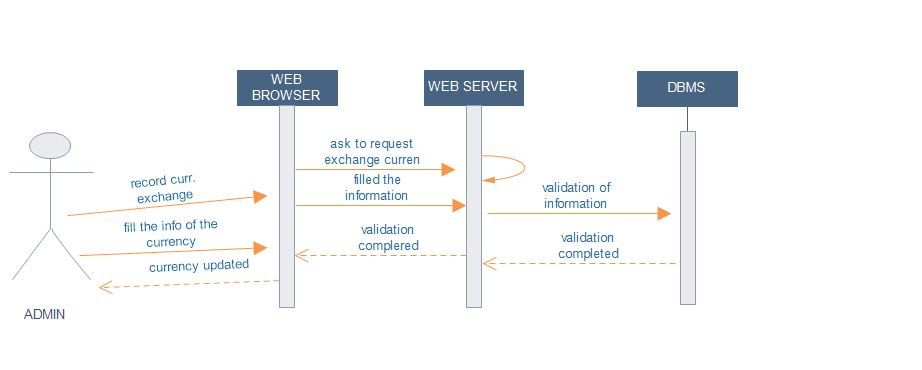
**Figure 3: Create user sequence diagram**

****

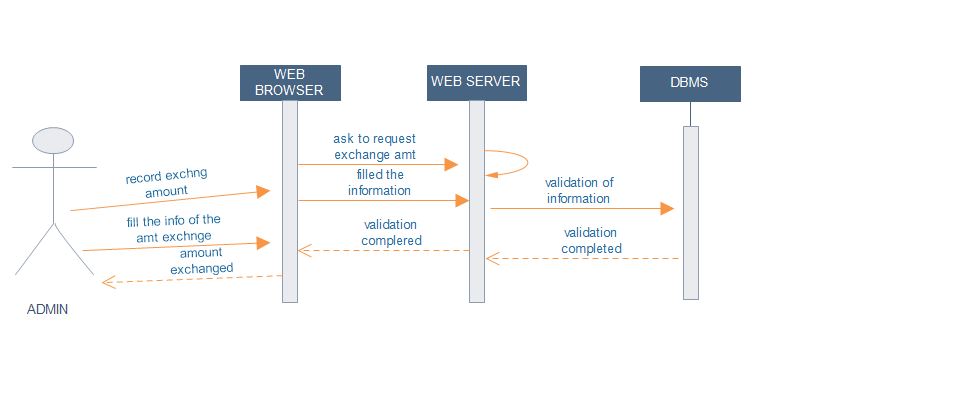
**Figure 4: login the system sequence diagram**

****

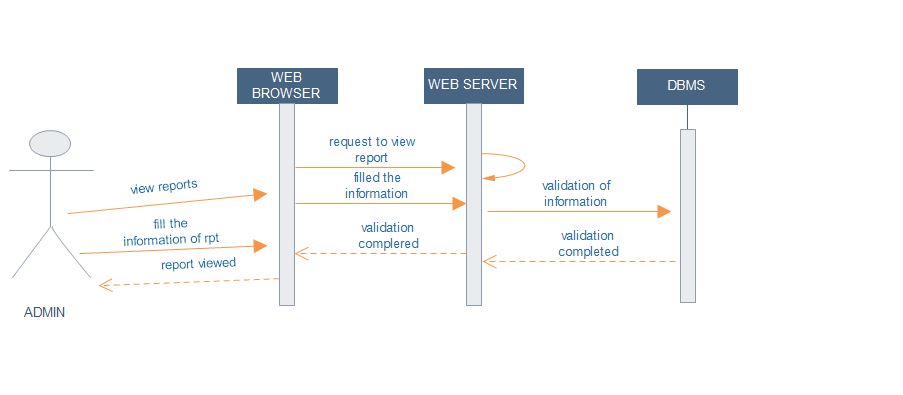
**Figure 5: updating currency exchange sequence diagram**

****

**Figure 6: exchanging amount sequence diagram**

****

**Figure 7: view report sequence diagram**

****

**Database schema**

The overall description of the database is called the database schema. The schema is specified during the database design process and is not expected to change frequently. However, the actual data in the database may change frequently; for example, it changes every time we insert details of a new member of staff or a new property. The data in the database at any particular point in time is called a database instance. Therefore, many database instances can correspond to the same database schema. The schema is sometimes called the intension of the database, while an instance is called an extension (or state) of the database.

**Figure 8: database schema of Forex bureau exchange management system**

