VirtualQ System Design with CASE Tools Document.

20. CASE Tool Setup

StarUML was installed and configured in order to model the system design.

Another name of a new project was developed as VirtualQ - System Design.

Use of all UML templates of diagrams was made possible.

21. Team Workspace Setup

An ordered workspace was produced within the CASE tool with the following folders:

Use Case Diagrams

Class Diagrams

Sequence Diagrams

Activity Diagrams

Component Diagrams

Deployment Diagrams

This makes all design artifacts to be organized properly.

22. Project Structure Creation

The project was packaged into the following:

Functional Design

Structural Design

Behavioral Design

Architectural Design

UML diagrams that are related to each package are included in the package.

23. Actors Identification

The players of the VirtualQ system are identified as:

Primary Actors:

Customer

Bank Staff

Admin

Supporting Actor:

System

24. Use Case Definition

The use cases identified are:

Login / Register

View Banks

Search Bank

Join Queue

View Queue Status

Receive Notification

Serve Next Customer

Monitor Dashboard

Manage Staff

View Reports

25. Use Case Relationships

The next relationships were noted:

Select Service <<|human|>Join Queue <|human|>Select Service

Get Customer <<|human|>Get Customer <<|human|> Update Queue.

Monitor Dashboard <<include|>View Reports.

These references are used to provide orderly interaction among system functions.

26. Use Case Documentation

Use Case: Join Queue

Precondition:

Customer should be registered into the system.

Main Flow:

Customer selects bank.

Customer selects service that is necessary.

System generates token.

Customer is enrolled in the queue.

Alternative Flow:

In case of full queue, a message is shown in the system.

Postcondition:

Customer is able to join the virtual queue.

27. Class Identification

The following classes were determined:

Customer

Queue

Bank

ServiceCounter

Staff

Admin

28. Class Attributes and Methods Definition

Customer Class:

Attributes: customerID, name

Functions: joinQueue, viewStatus.

Queue Class:

Attributes: queueID, queueLength

operations: insertCustomer, deleteCustomer.

Staff Class:

Methods: serveNext()

Admin Class:

Methods: monitorSystem()

29. Class Relationships

The relational patterns found are:

Queue is related with Customer.

Staff manages Queue.

Admin supervises Staff.

Bank has several ServiceCounters.

30. Sequence Diagrams Creation

Sequence diagrams were drawn regarding:

Join Queue Process

Serve Customer Process

View Queue Status

Admin Monitoring

These charts indicate communication between actors and system objects.

31. Interactions between Objects Documentation

Customer comes to System and gets in queue.

System communicates with Queue to make updates.

Staff communicates with Queue to attend to people.

Admin communicates with Dashboard to check system.

32. Alternative Flows Inclusion

Other possible scenarios were:

Login failure

Queue full

No staff available

Network delay

Such cases are recorded in sequence diagrams.

33. Activity Diagrams Creation

Activity diagrams were drawn up on:

Joining workflow of customers in line.

Staff Service Workflow

Such diagrams have decision nodes and parallel activities.

34. Workflow Documentation

The workflow includes:

Junction points (Service available?)

Serve at several counters simultaneously.

Notification on approaching token.

35. Component Diagram Creation

The system components that can be identified are:

User Interface Module

Queue Management Module

Admin Module

Map Integration Module

These elements come into action to give complete functionality to the system.

36. Creation of the Deployment Diagram

The deployment structure is:

Client Device (Browser)

↓

Web Server

↓

Application Logic

↓

Simulated Database

This demonstrates the deployment of the system in a web set up.

37. Design Document Creation

The system is based on a web-based architecture which is modular.

The simplicity and academic feasibility are guaranteed through the simulation approach.

The modules are designed separately in order to make them maintainable and scalable.