Chapter 3

Requirement Analysis and System Specification

This chapter will focus on system requirements and problem domain analysis. The object-oriented methodologies called Unified Modeling Language (UML) will be used as modeling techniques to explain structural aspects of the system. UML will help us in specifying, constructing, visualizing and documenting the software system and its components.

3.1 System Pre-Requisition

The primary objective of this project as mentioned in chapter 1 is to build a system called QS-Space that capable of carry out *Quadratic Sieve* factorization method in distributed and parallel way. The system should able to function as follow:

- The system should be able to delegate jobs consistently and collect data from the participated nodes without any partial failure (with the help of transaction management system).

- When designing distributed application, platform heterogeneity is always the key issue. It is always required that an application to be built need to be designed in different architecture in order to make compatible with the specific platforms. This is truly a redundant job and therefore platform independent language such as Java is to be considered in order to leverage the portability across heterogeneous platforms.

- The result must obtained in a finite time – the amount of time user willing to wait before getting the result. The system must scalable - more computation power contributes to the system resource pool, lesser time is needed to get the result. It is expected to take long period of time to finish a task, thus, the system should be able to handle a consecutive batch of tasks. System reliability and consistency need to be achieved when designing a distributed system that can perform iteratively – an intensive job running in a long period of time. The system should able to give the status of each finished task and log the system messages and errors for future references. Remote access control, monitoring and remote notification are essential important especially when user is not willing to spend time in front of the system waiting for result that might took hours.

- Any hosts that sit within the network boundary can participate with this project by downloading the necessary files (slave module) from a delegated web server. The installation should be simple, fast and hassle free from complex configuration. Slave module is allowed to be configured transparently while requesting a task.

- System consistency and resource availability is the main concern for this project. Total number of slave nodes required for a test run must be maintained. Any nodes that failed or deactivated during the test run would have to be detected by the system and a recovery mechanism will be executed immediately in order keep the system stay on its consistent state.

3.2 Non-Functional Requirements

The non-functional requirements can be summarized as follow:

- Distribution
- Persistent data management
- The system must able to self-describe
- The targeted system can be Windows, UNIX, Linux and others that allowed by the implementation language.

3.3 Use Cass Driven Analysis

Object-oriented analysis will be performed to determine the system requirements and identifying classes and their relationships to other classes in the problem domain. A use case is a typical interaction between user and a system that captures users' goals and needs.

3.3.3 Use Case Diagram

Below is the UML use case diagram showing the system relationships that can be captured with respect to the above system pre-requisition addressed.

The central of use case is **Start Factorization Process** and **Stop Factorization Process**– they contain all the coordination and data management functionalities for decomposing a problem into sub-tasks that can be distributed to all **Slave** and finally return the collected result to **Administrator**. **Start Factorization Process** use case is extended by **Check Previous Incomplete Process** which will be triggered by the system when a previous last incomplete process record has been detected in the

persistent storage. **Resume Previous Incomplete Process** use case capable of checking of all incomplete processes and let **Administrator** decide which to be continued. **Remote Access Control** use case allows **Remote Administrator** to monitor and control several functionalities of the system remotely. The **Start Factoring Process** use case and **Slave** is communicated indirectly by accessing the **Access Task Pool** and **Access Result Pool** use case.

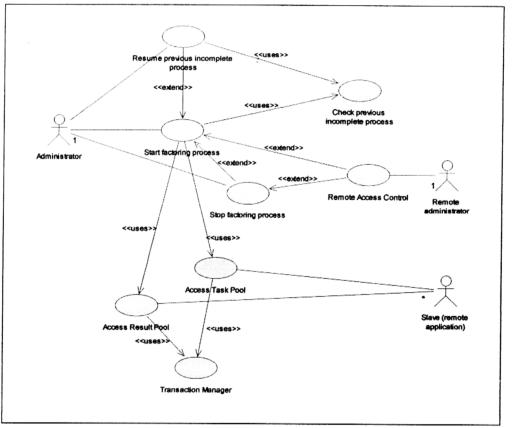

Figure 3.1 Use case diagram for QS-Space application

There are three actors: **Administrator**, **Remote Administrator** and **Slave**. **Slave** provides computational operation on tasks that assign to it, whereas **Administrator** and **Remote Administrator** are responsible for the management of whole system operations. Below are the use case dictionaries of each actor.

a. **Administrator**

The system must configure before a meaningful operation as desired can performed. **Administrator** also responsible for monitoring and bring up other necessary companion systems needed.

Number of actor: One

Experience: Expert user

Location: Inside a building.

b. **Remote Administrator**

Functional as **Administrator** but with limited access rights: allows monitoring the system processes, start and stopping the system operations.

Number of actor: One

Experience: Novice to advanced user

Location: Anywhere in the building or at home with network connection

c. **Slave**

It is subset of the system but disjointed from the system and communicate indirectly, responsible to perform task commanded by the system independently.

Number of actor: At least one and unlimited numbers.

Location: Anywhere in or out side of the building with network connection.

3.3.2 Use Case: Start Factoring Process

It is best to illustrate the interactions in the use case by using UML interaction diagram (Figure 3.2). There are two modes of starting a factoring process: a system can be executed as **Single Operation** or **Consecutive Operation** (Figure 3.3) that

capable of factoring multiple composite numbers. The system must capable of storing maximum of 20 composite numbers to be executed by **Consecutive Operation**.

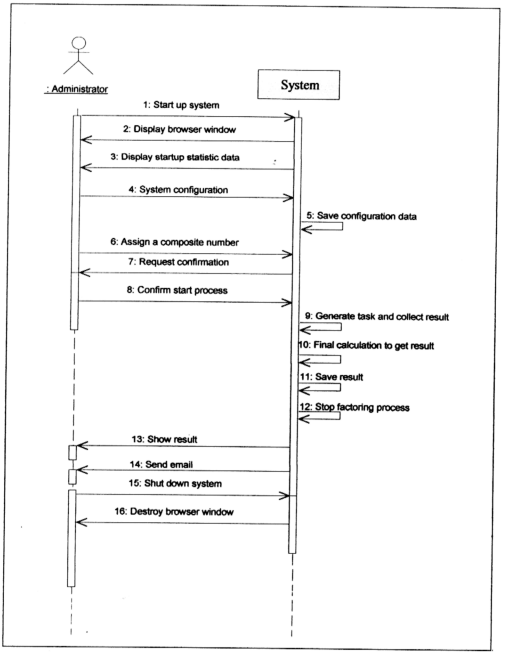

Figure 3.2 Sequence diagram **Start Factoring Process** use case

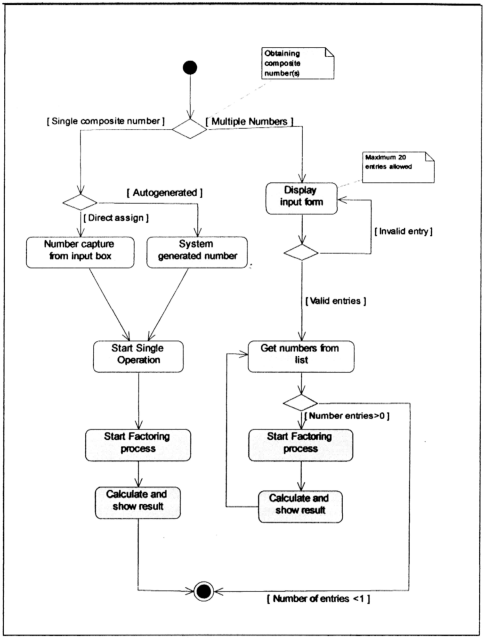

Figure 3.3 Activity diagram for **Start Factoring Process::Single Operation** and **Start Factoring Process::Consecutive Operation**

3.3.3 Use case: Resume Previous Incomplete Process

This use case is responsible for reinitialize all information and parameter data from the persistent storage in order to return back to its previous state of inconsistency. All

relevant data need to be loaded into the system environment before a previously terminated process can be resumed. It uses **Check Previous Incomplete Process** use case for query any incomplete process recorded in the persistent database.

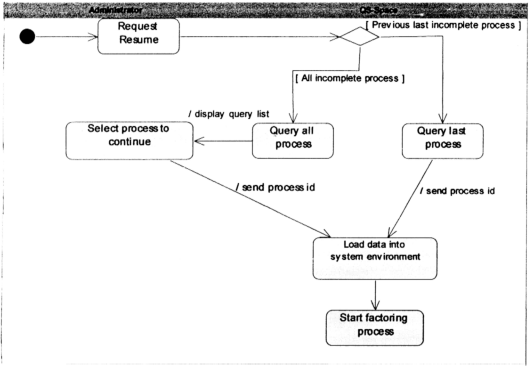

Figure 3.4 Activity diagram for **Resume Previous Incomplete Process** use case

3.3.4 Use Case: Access Task Pool and Access Result Pool

Since the project's communication protocol has been defined earlier based on tuple-space (section 2.4.3); hence, communication between **Slaves** with the system will be performed indirectly by accessing the space entries. Both of these use cases are responsible for providing mechanisms for messages repository and interchanging. The message is in the form of an object called *Entry*. Further explanation of this process based on master-slave framework will be discussed in next chapter. Sequence diagram

3.5 and 3.6 illustrate the basic interactions of tuple-space operations between system and slave.

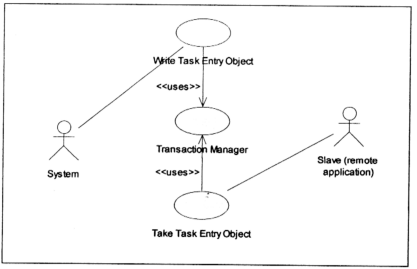

Figure 3.5 Access Task Pool use case

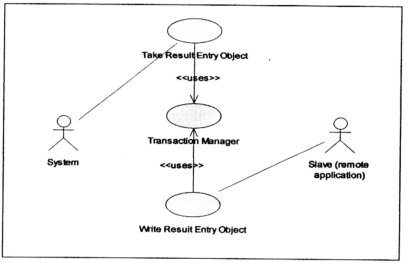

Figure 3.6 Access Result Pool use case

Transaction manager use case ensures that every transaction operation of *Take* and *Write* is atomic (ensuring an operation is fully completed or not happen at all if failure occurred).

3.3.5 Use Case: Remote Access Control

Remote Access Control use case provides limited access to the system as follow:

a. Monitor the progress of system operation

b. Start and stop process remotely

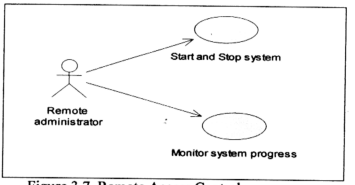

Figure 3.7 Remote Access Control use cases

3.4 Chapter Summary

Drafting system requirements needs the understanding of how a system may behave to achieve its goal. However, the software designer will need the appropriate tools to do the jobs well. Unified Modeling Language or in short UML provides the guidance of how visual representation of notation can be used to construct object-oriented application.

The use case model able shows the relation of users and the use of the system. Use cases' nontrivial dynamic behaviors are specified further with the help of UML's activity diagrams and sequence diagrams. Next chapter will focus on how the requirements discussed in this chapter can be used to design system framework and classes of QS-Space components.