DGS

INTRODUCTION

1.1. ABOUT PROJECT:

The main aim of this project is, to allow users to distribute the Job processing, situated globally across multiple computers.

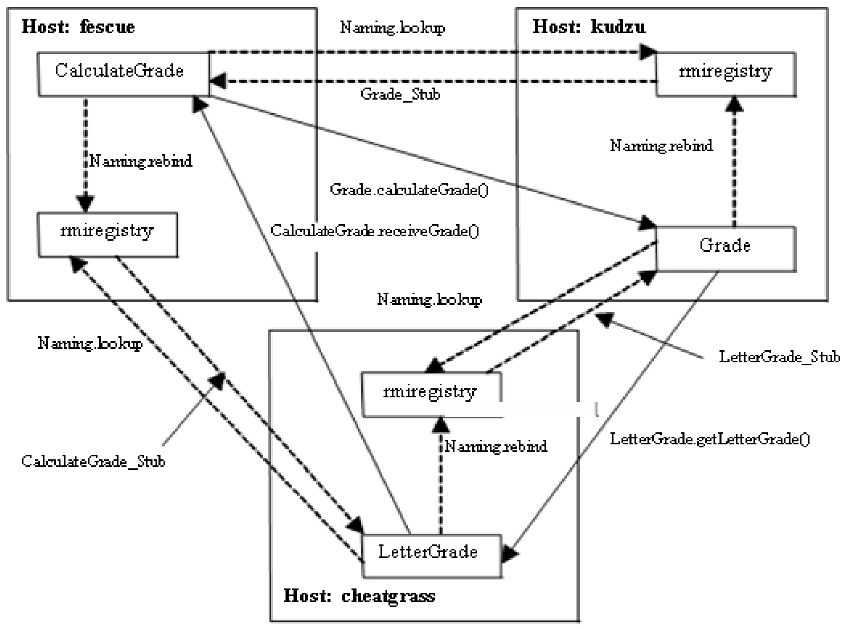
. The project has been developed in **JAVA** by using RMI Technology with Windows2000 server as operating system with necessary JDBC. RMI provides a simple infrastructure for executing methods remotely. The "Distributed Grading System" consists of three processes that communicate remotely over the network this can be briefly explained in the project description.

- This project has been developed based on RMI technology.
- Remote Method Invocation (RMI).
- RMI is used in Java only.
- RMI technology is used to communicate two or more systems simultaneously.
- It maintains the data in distributed databases.

1.2. PROJECT DESCRIPTION:

This section describes the architecture of the system. The diagram below shows the three hosts that were part of the demonstration, as well as the processes running on each host. The diagram also shows the communication between the various processes. The dotted lines indicate the communication a process has with the rmiregistry, while the solid lines show the actual remote method invocation.

There are also a number of UML diagrams attached that show the overall design of the system. The diagrams show the inheritance relationships between the classes. There is also an event diagram attached that shows the order of operations while the system is running.



The system consists of three processes running on different machines. The first process, called CalculateGrade, runs on fescue. The first thing this process does is it registers with the naming registry on the local host. It does this so one of its remote methods, receiveGrade, can be called remotely. The process then gets a reference to the Grade object by performing a lookup on the rmiregistry running on kudzu. It then invokes the calculateGrade method on the remote object, passing as a parameter an instance of the RemoteFile class. After calling this method, the CalculateGrade process waits to receive a final letter grade from the third process, which is called LetterGrade. Once it receives that grade, it prints the grade out to the screen.

The Remote File class is used to simulate a file. RMI requires that an object that is a parameter to a remote method must be serializeble.

This allows the object to be transferred from one virtual machine to another. Since input streams aren't serializeable, a new object had to be created to simulate a file when it's passed to a remote method. The remote file class does just that. It allows the programmer to easily send the contents of a file to a remote method, so the file can be processed on a remote machine. The constructor of the class takes a string as a parameter. The string indicates the name of the file that will be used. The constructor reads the contents of the file into a byte array. Any method that operates on the remote file class can then call the getInputString function. This function returns an input stream to the byte array that stores the contents of the original file. Once a process has the input stream, it can operate on it just as it would for a normal file. So the RemoteFile object makes it appear to the application that the store locally.

The second process, called Grade, registers with the naming registry on kudzu and then waits for a client to call one of its methods. The method that the CalculateGrade process calls takes a RemoteFile object as an input parameter. The remote file contains a number of integers, which represents grades that a student has a accumulated in a course. The process reads in the contents of the file, and calculates the average of these grades. When that is complete, the process gets a reference to the LetterGrade object by performing a lookup on the rmi registry running on cheatgrass. The reference is then used to call the getLetterGrade method of the LetterGrade object. This method takes one parameter, which is the average that was just calculated.

The third process, called LetterGrade, registers with the naming service running on cheatgrass and then waits for a client to call one of its methods. The method that the grade process calls takes one parameter, which represents the final average of a student. This process then assigns a LetterGrade for that average. So if the average is an 85, this process will assign grade of ‘B’. Once the LetterGrade is determined, the process calls the receive method of the CalculateGrade process, which is running on the first machine. This method takes one parameter which is the final lettergarde.

1.3. PROBLEMS IN THE EXISTING SYSTEM:

1. Maintaining the operations in stand alone system.

2. User cannot send two or more request at a time.

3. It is difficult to maintain the more number of calculations.

1.4. OBJECTIVES OF THE PROPOSED SYSTEM:

- Development of user friendly system, which can use in globally across multiple computers.
- The main advantage in this proposed system is communication of two or more systems simultaneously.
- Easy of updating the calculations.
- The maintenance can be done more efficiently and effectively.

1.5. PROBLEM DESCRIPTION:

In this project, the main aim is, each and every server must be communicating with all other servers. Here mainly one objective is available.

The objective is shown as follows.

1. Administrator

Administrator:-

In this the administrator is given to a server to perform some request operations onto different servers which are distributed globally. Here the constraint is these distributed servers are by default connected to LAN or WAN and some default security levels.

In this project there is only one module. This module is the main basic objective of the project. The brief description of this module is as follows.

1.6. OVERVIEW OF MODULES:

Module 1:

This module is given to a server to perform some request operations onto different servers which are distributed globally. Here the constraint is these distributed servers are by default connected to LAN or WAN and some default security levels.

2. SOFTWARE & HARDWARE REQUIREMENTS

SOFTWARE REQUIREMENTS:

The minimum software requirements for this project are as shown below.

Technologies	**: RMI, J2sdk1.4,**
	JDBC, DAO.
Database	**: Oracle**
Operating System:	**Windows 2000 prof/XP.**

HARDWARE REQUIREMENTS:

The minimum hardware requirements for this project are as shown below.

Processor	**: Intel P-IV based system**
Processor Speed	**: 250MHz to 833MHz**
RAM	**: 256MB to 2GB**
Hard Disk	**: 10GB**