GEORGE MASON UNIVERSITY
Computer Science Department

Distributed Software Systems

Spring 2006

Assignment 3

A Calendar Tool for Work Groups

DUE DATE: May 2

In this assignment, you will extend the calendar tool you developed for Assignment 2. Please read the handout for Assignment 2 for a description of the basic functionality you have to implement.

1 System Architecture

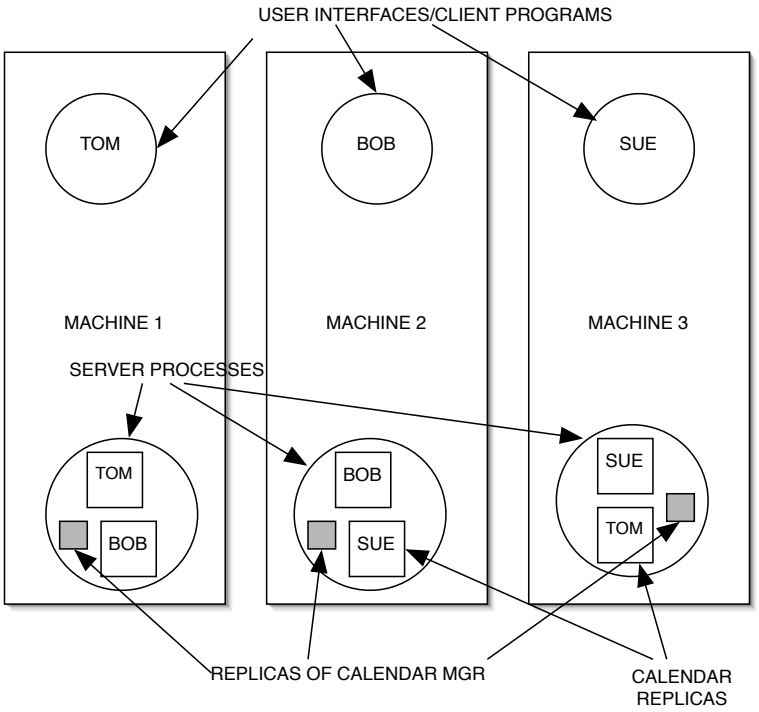

Figure 1: Distributed Software Architecture for the Calendar Tool

The software architecture of your application is illustrated in Figure 1. Note the following points:

- Multiple machines have a "server" process that houses the following objects

    1. A single Calendar Manager object. This is the same calendar manager that you implemented for Assignment 2, i.e., it acts as a class factory for calendar objects. However, instead of there being a single calendar manager for the entire system, you now have a calendar manager per machine. Each calendar manager maintains a replica of the user list – a data structure that contains information about the existing calendar objects in the system. The calendar manager's functionality is discussed in more detail in Section 2.

2. Calendar objects. Each server process may house several calendar objects corresponding to the calendar event databases of different users.

- Client programs (the user interface programs) use the Naming Service to obtain references to one of the available calendar managers. The calendar managers maintain a list of all existing users and the references for their corresponding calendar objects. Thus, a client program (corresponding to a particular user) can obtain a reference to the calendar object for that user from any calendar manager. The location of the calendar object is transparent to the client program.

2 Calendar Manager

The calendar managers taken together essentially implement a *fault-tolerant distributed* class factory. Each calendar manager maintains a replica of the "user list" which is a data structure containing an entry for each calendar object that exists. When clients contact the calendar manager in order to obtain a reference to their calendar objects, this list is checked, and if there is no entry corresponding to that user, a new calendar object is created, and a corresponding entry is inserted into the user list. Since the user list is replicated, you need to propagate this new entry to all the calendar managers. You therefore have to implement a replica control scheme that ensures that all the replicas are consistent. I suggest using a simple scheme such as "read one, write all available copies".

One of the main requirements for the Calendar Manager service is that it be fault-tolerant. This is achieved via replication. If a machine on which a replica of the Calendar Manager service is running goes down, this should not affect the operation of the Calendar Manager service, i.e. you should still be able to create Calendar objects as described above.

You should also be able to start new replicas of the Calendar Manager service dynamically. This means that you will need to devise a mechanism for keeping track of all available Calendar Manager replicas, and for initializing a new Calendar Manager replica based on the current state of the existing replicas.

Note that each calendar manager object is a persistent object, i.e., the replicated "user list" should persist beyond the lifetime of the server process in which the calendar manager object is instantiated.

3 The Calendar class

The functionality of the calendar class was described in the handout for Assignment 2. For this assignment, you have to extend the calendar class in two ways:

- make the calendar objects *persistent*. In other words, each user's calendar database should survive beyond the lifetime of the server process in which the corresponding calendar object is instantiated. You cannot assume the existence of a central file store; instead, you should use the local disk on a machine where the server process is running for storing a copy of the calendar object.

- replicate each calendar object so that if the primary replica is unavailable, a user will still be able to access his/her calendar. You will need to design and implement a mechanism for keeping all the replicas of a calendar object persistent.

Additional Requirements

Handling Partial Failure The biggest challenge in designing a distributed application is to handle partial failures that can occur during an operation. These partial failures could be due to host crashes or network partitions. While designing your application, you have to be cognizant of the possibility of partial failure and design your application to be able to handle these failures in a graceful fashion. In other words, partial failures should not leave the application in an inconsistent state. For example, in the calendar tool application, if a calendar manager that has locked several calendars fails, it will leave the system in an inconsistent state. You are *not required* to handle partial failures in your project implementation. However, you **have to submit a design document** describing how (i) the possible failures that can occur in your system (ii) how partial failures would affect

different parts of your application and how/why your current implementation is inadequate for handling partial failures, and (ii) explaining how you can extend your implementation to make it resilient to partial failures.

This document is worth 15% of your grade for this project.

4 Submission

You will need to demonstrate the effectiveness of your design and the correctness of your implementation. Your demonstration should show:

- your user interface in action as a user views her personal calendar and that of other users. Note that users can view their own calendars as well as the public and open events in the calendars of other users
- that users can schedule events in their personal calendars,
- that the system is able to schedule shared events such as group meetings
- that the user interface is able to alarm the user when events like appointments are about to occur.
- that the events scheduled by a user are persistent, i.e, they are not lost if the server containing the user's calendar object is shut down.
- that each calendar is replicated so that it is available as long as one copy is accessible.
- the fault-tolerant nature of the calendar manager service. I should be able to start a new replica of a calendar manager, and to shut down (or kill) an existing replica without the calendar manager service becoming unoperational.

Your interface should do a reasonable job of error checking, e.g., your program should not crash if I type in an event time that is not meaningful.

Schedule

April 10 - 21 Schedule a design review meeting with me. This should take around 15 minutes. Ideally before this meeting you should have written the remote interfaces that will be needed for your distributed application.

May 1 - May 8 Schedule a time to give me a demo of your application. This should take around 15-20 minutes. Submit a document that describes your system architecture and protocol design. You also need to include a discussion of how your system can handle partial failure, as described above. Please also submit a hard copy of the source code for your application.