“Distributed Calendar Implementing Wuu and Bernstein’s Algorithm”

Nick Marton and Ian O’Boyle

This Distributed Calendar which utilizes the Wuu and Bernstein’s Algorithm, uses three classes in its function. These classes are implemented in Node.py, Event.py, and Appointment.py.

The Node class utilizes Event and Appointment objects in its execution. The command to start running a Node locally is “python Node.py [PORT\_NUMBER]”. This command also starts the user client to allow scheduling of appointments from command-line input. After a user schedules an event for the calendar, their request is made into an Appointment object, A Node will then use that Appointment object as a record of the user-scheduled event. An Appointment object is then used in the creation of an Event object, which will be stored in a Node’s log.

An Appointment object contains a name, a date, a start time, an end time, and the names of all other users associated with the appointment. The implementation of this object also contains methods to compare two Appointment objects together and determine if they are conflicting.

The Event class is implemented in Event.py. An Event represents an action that takes place at a Node, Events are then stored in a Node’s log. An Event includes information like the Event’s operation type, the ID number of the Node that created the Event object, and a timestamp of the Event’s creation. This implementation also includes functions for determining if Event objects are equal.

The Node class is implemented in Node.py. A Node object has a unique ID, a local clock of the Node enforced as an integer and incremented whenever it is referenced, a local calendar of events maintained as a dictionary data structure, a local log of event records maintained by this Node, a 2D Time Table of Integers, the number of total Nodes in the distributed system enforced as an integer, Dictionary of form [Int: (IPADDRESS, PORT\_NUMBER)], containing the NodeIDs to IP address relationship of all nodes in the system. The implementation of the Wuu and Bernstein’s algorithm is also found in this file, with all necessary functions as methods of the Node object. When a Node receives a message, it will look for any conflicting appointments by saving its dictionary state before receiving updating its knowledge from the message. After receive is finished, the Node will extract new insertion of Appointments that were not present in the previous dictionary. For each of these new entries, the Node will determine if the event connected with this new entry conflicts with an entry already existing in its previous Dictionary.

If the Node finds a conflict, it will find the appointment conflicting with the new appointment, this will be called the Original Appointment. Now there are two cases to consider with the Original and new appointment: The event corresponding to the Original Appointment is present in the Node’s log, or it is not. For the case it is not in the Node’s log, the newer Appointment is deleted. For the case it is in the Node’s log, we first extract the Node ID of the Original Appointment and the Node ID of the Newer Appointment, we then compare what this Node knows that it knows about the Nodes that generated the Older Appointment and the Newer Appointment. More formally:

Ti[i][oR.node] < Ti[i][nR.node] where i is the id of this node and oR an nR are the event records denoting the original event and new event respectively. Choosing to keep the event that happened earlier.

After running the command a listener begins waiting for user input to schedule events. Scheduling events must take this form :

“acting\_user **schedules** name\_of\_appointment (tuple\_of\_users\_involved) (start\_time\*, end\_time\*) day\_of\_week”

Or alternatively for delete:

“acting\_user **deletes** name\_of\_appointment (tuple\_of\_users\_involved) (start\_time\*, end\_time\*) day\_of\_week”

\*Times must be in half-hour intervals.

Other commands at the listener include:

**print log**:Display the log of the local Node instance

**print Calendar:** Display the Calendar of the local Node instance