Distributed Systems. 5DV020, HT09
Ludvig Widman(dit06lwn), Emil Eriksson(c07een)

Tutors: Lars Larsson, Daniel Henriksson
Date: September 11, 2009

Assignment part 1

Analysis and design

Contents

1 Bonus Level

We intend to solve bonus level two with dynamic groups.

2 Tools and language for implementation

Since the clear recommendation is to use Java and Java RMI we have decided to follow the recommendation. While there are languages that would be more interesting and easier to design GUIs in (cocoa/objective c), Java is more easily ported and the available help from lecturers also helped tip the balance in Java's favor.

JUnit will be used for unit testing and logging will be done through the log4j-package.

3 Project plan

Date	Milestone	Content
11 Sept	Part 1 due	Written project plan
18 Sept	Milestone 1	Create all interfaces
		Build dataobjects
		Design GUI mockups for test app
25 Sept	Milestone 2	Implement basic methods like non-ordered ordering and non-reliable multicast.
2 Oct	Milestone 3	Implement advanced methods like total ordering and reliable multicast.
9 Oct	Milestone 4	Implement GUI-application and debug mode.
16 Oct	Milestone 5	Resolve distributed problems.
		Write on the report.
		Make test protocol.
23 Oct	Finalization	Everything should be finished. Minor adjustments.
27 Oct	Hand in report	
28-30 Oct	Demonstration	

4 Design

4.1 GCom

A framework (middleware) for communicating between distributed nodes within groups. This module exposes the functionality of the other modules that are part of GCom so that referencing them directly should not be needed in order to use them.

4.1.1 Interface

+ joinGroup(String groupName)
  Contact RMI registry and get group-leader
  Contact group-leader

+ createGroup(String groupName) Register at RMI registry

+ removeGroup(String groupName)
  Removes the group from the RMI registry

+ leave(String groupName)
  Send GroupView excluding sending Member

+ listGroups
  Ask registry

+ listMembers(String groupName)

+ sendMessage(String groupName, Message)
  Sends a message to the specified group via CommunicationModule

+ connectToRegistry(String host, int port)
  Connects to a RMI registry

4.2 CommunicationModule

Sends and receives messages to and from other processes. This module must support both reliable and basic multicast. Received data is sent to the MessageOrdering-Module. The GroupManagementModule is also notified with the senders process ID.

4.2.1 Interface

+ receive(Message) This method is called remotely by other nodes when sending messages to this node.

+ send(Group, Message)
  Look up communication type in group
  Look up group members

- multicastB Private method used internally to perform a basic multicast.

- multicastR Private method which uses multicastB to perform a reliable multicast.

4.3 MessageOrderingModule

Sorts incoming messages and makes sure that they're delivered according to the specified order (Non-ordered, FIFO, Causal, Total, Causal-Total). Non-ordered is the main class and the other ordering methods is implemented as subclasses.

4.3.1 Interface

+ queueMessage(Message)

4.4 GroupManagementModule

The GroupManagementModule contains collection of Group objects. It monitors if a process stops responding. Notifies changes in the states to all members of the group (via the CommunicationModule). The GroupManagementModule also resolves group names to a list of members.

4.4.1 Interface

+ addGroup(String groupName)
+ removeGroup(String groupName)
+ listGroups

4.5 Group

Stores a group's state and group members.

4.5.1 Interface

+ addMember
  Adds the member to the member list
+ removeMember
  Removes the member from the member list
+ listMembers
+ update(ProcessID)
  Updates the last seen vector time for the process

4.6 GUI

The GUI class contains the application that uses GCom and is mostly used for testing and demonstration. We intend to write some kind of simple chat application. The GUI registers listeners in the GCom class to get information.

4.7 Message

The Message is a basic data object which can be serialized. It contains a Vector clock, a destination group, message type and the actual message.

4.8 RMI Registry

A central RMI-registry. Single point of failure. Leader registers and unregisters group.

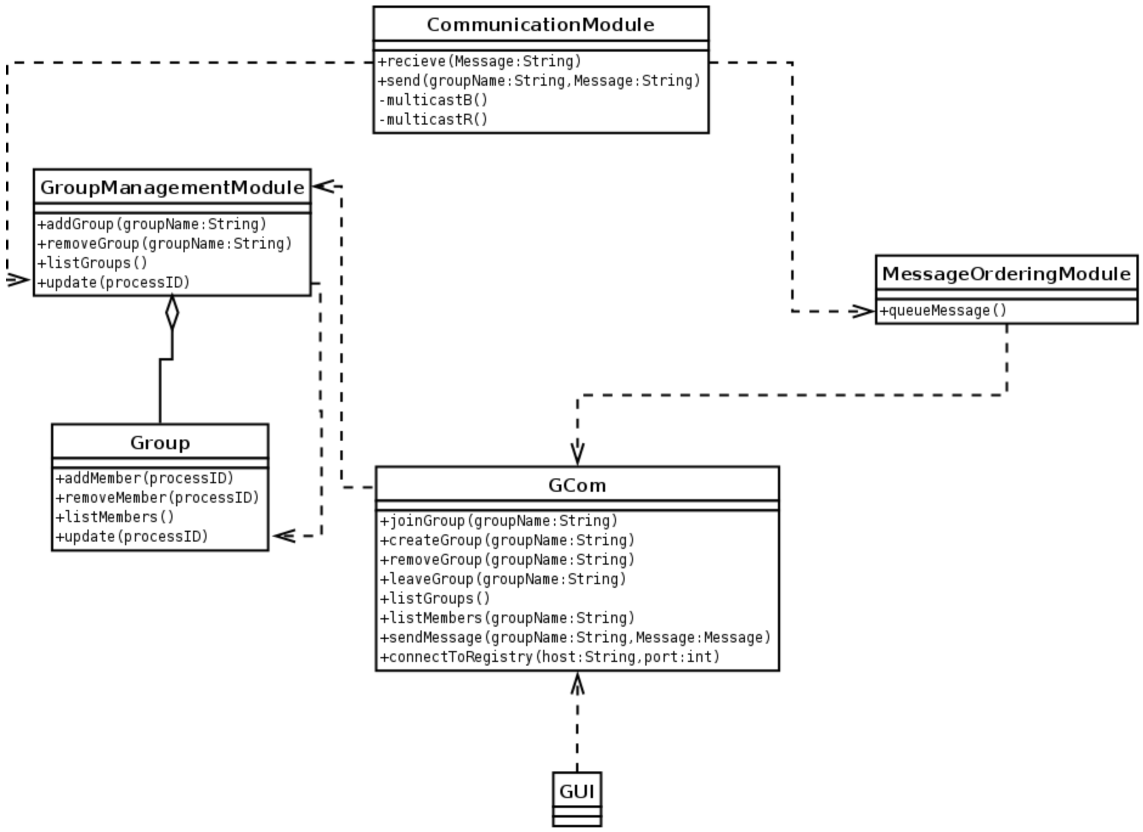
CommunicationModule
+recieve(Message:String)
+send(groupName:String,Message:String)
-multicastB()
-multicastR()
GroupManagementModule
+addGroup(groupName:String)
+removeGroup(groupName:String)
+listGroups()
+update(processID)
MessageOrderingModule
+queueMessage()
Group
+addMember(processID)
+removeMember(processID)
+listMembers()
+update(processID)
GCom
+joinGroup(groupName:String)
+createGroup(groupName:String)
+removeGroup(groupName:String)
+leaveGroup(groupName:String)
+listGroups()
+listMembers(groupName:String)
+sendMessage(groupName:String,Message:Message)
+connectToRegistry(host:String,port:int)
GUI

Figure 1: How the modules are related.

5 Use cases

A few of the scenarios that might arise during using and testing GCom.

5.1 A node creates a new group

When a node wants to create a new group, it connects to the RMI-Registry and leaves a reference to itself under the name of the channel.

If this succeeds it sets itself as the group leader and adds itself to it's view of the group in the GroupManagementModule.

If the registration fails an exception indicating if the name was already taken or the RMI-Registry was unresponsive is thrown.

5.2 A group leader removes a group

The leader removes all members from it's view of the group in GroupManagement-Module and then sends this updated view to all members of the group via the CommunicationModule. It then unregisters the group name in the registry.

5.3 A member joins a group

When joining a group the prospective member connects to the RMI-Registry and gets a reference to the group leader and sends it a join request. The leader adds the new member to it's view of the group and then sends this updated view to all members of the group, including the new member. When the new member receives the updated view it knows it is part of the group.

5.4 A member leaves group

When a member wants to leave a group it sends a group view excluding itself to all the members (including the group-leader) of the group.

5.5 Send message to a group

First a list of all members in the group is obtained from the GroupManagement-Module and then the message is sent to all of the members.

5.6 Group-leader leaves group

When the group-leader leaves the group the remaining members starts electing a new group leader by creating a semi-random (and unique) value and sending this to all other nodes in an election-message.

When a node receives election-messages it compares the value in the incoming message with the value it created and sends the larger of these to all other nodes. When a node has received messages with it's own value from all members of the group it sets itself as group-leader and registers with the RMI-Registry.

For now we do not take into account that a node might crash during the election.

If the group-leader leaves a single member behind this node realizes it is alone in the group and sets itself as group-leader and registers with the registry.

5.7 The last member in a group leaves

When the last member of a group leaves it is the group-leader and it unregisters itself from the registry.

6 Fault tolerance

6.1 Detecting crashed members

The GroupManagementModule gets notified when a message is received and can therefore keep track of the time since the last message from a process, measured in vector time. If a process does not send anything for a long time it will be marked as crashed and removed from the group list.

6.2 If the registry restarts or is unresponsive

The leader of each group occasionally checks that the group is registered in the RMI-registry. If the registry has restarted for some reason the leader will re-register the group.

6.3 If the group leader crashes

If the group leader crashes the group will detect that no messages arrive from the leader and then elect a new leader.

6.4 If the last member of a group crashes

If the last member of a group crashes no one will unregister the group. This will potentially lead to a lot of "dead" references in the registry if there is a bug causing nodes to crash when they are the last to leave a group. In order to counter this some extra testing will be done to make sure that nodes behave when leaving and unregistering with the registry. If the time allows we will try to implement a way to clear out old references in the registry (other than restarting it periodically).

6.5 Netsplit

If the group is split in two partitions, due to a cable failure or similar, the partition with the leader will continue to work as before. The other partition will elect a new leader when they discover that they no longer receive messages from the leader. The new leader will try to reregister the group in the registry. The leader for the other part will do the same and therefore compete over the reference in the registry. Because of this newcomers will join the group having the reference in the registry at the time the node connects.

7 Requirement Specification

7.1 Group management

REQUIRED External processes must be able to find and connect to group leaders via a RMI registry.

REQUIRED The group leader must regularly make sure that the group and leader is registered in the RMI registry.

REQUIRED Any process must be able to create a new group and register itself as leader in the registry.

REQUIRED Any process must be able to join a group by connecting to the group leader.

REQUIRED Every process in a group must notify the other processes of any known changes to the group composition.

REQUIRED Every process must keep track of the members of all the groups it's a member of.

REQUIRED Every process must monitor if another process in the group stops responding and report a corresponding change in group composition.

REQUIRED If the leader of the group leaves or crashes a new leader must be elected in the group.

7.2 Communication

REQUIRED It must be possible to send messages to the members of joined group.

REQUIRED Both non-reliable and reliable multicast must be supported for sending messages.

REQUIRED The communication method chosen by the group creator (reliable or non-reliable multicast) must be used when communicating in a group.

REQUIRED Every message must contain a updated vector clock.

7.3 Message Ordering

REQUIRED Any message must arrive in proper order (according to the selected sorting requirements).

REQUIRED A message must not be delivered until all previous messages are delivered if the message ordering demands so.

8 Limitations

- The RMI-registry is a single point of failure. If the registry crashes no processes can find or join groups.
- If a group is split in two parts due to network failure and both parts still have contact with the registry bad things will happen. This is assumed to never happen.
- All processes are assumed to be good. If a process starts to lie or send malicious messages bad things will happen.