UMEÅ UNIVERSITY
Department of Computer Science
Report

October 25, 2009

Deliverable 2

Distributed systems, HT-09

GCom

Anton Johansson, dit06ajn@cs.umu.se
Jonny Strömberg, dit06jsg@cs.umu.se

`~/dit06ajn/edu/dist/GCom`

Supervisors
Lars Larsson, larsson+ds@cs.umu.se
Daniel Henriksson, danielh+ds@cs.umu.se

Contents

1 Introduction

This report explains a solution for implementing a distributed group communications middleware.

A distributed system is composed of separated processes that coordinate activities by passing messages and a middleware is a software layers that enables rapid development of other software by supplying simple method-calls that hides the underlying implementation details off the middleware.

The middleware described in this report is called *GCom* and provides an API[1] for group communication with different message sending/delivery rules. Two communication methods are implemented: *Reliable multicast*, *Basic multicast*, described in greater detail in section 4.3.

Four message-ordering types are implemented: *Non-ordered*, *First in first out*, *Casual*, *Total* and *Casual-Total*, described in greater detail in section 4.4.

The system is implemented in the programming language *Java*[2] and uses *Java RMI*[3] for network communication.

The original specification of this practical assignment can be found at (*October 25, 2009*):
`http://www.cs.umu.se/kurser/5DV020/HT09/assignment.html`

2 Problem analysis

The group communication for *GCom* is specified to be a distributed system, which means there can be no central server that coordinates all activities for individual group members. Four guidance on how to implement such a system the book *Distributed Systems: Concepts and Design*[DKC05] list three important consequences of a distributed system:

- **Concurrency:** Program execution are concurrent. In the case of *GCom*, message receiving and handling are concurrent with other parts of the middleware such as message sending and ordering.
- **No global clock:** There is no global clock to coordinate activities by. That is clock timestamps can not be used to order messages received by *GCom*.
- **Independent failures:** All individual parts of the distributed system can fail at any time and place in execution. This must be considered when implementing algorithms for coordinated actions of *GCom*.

[1] Application programming interface
[2] `http://java.sun.com/`
[3] `http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp`

The environment in which *GCom* will execute will defined by a model for distributed system called *asynchronous*-system defined by three assumptions [DKC05]:

- There is no guarantee of **execution speed**, a process may respond to a request immediately or after several years.
- In a similar manner there can be **transmission delays** in the network were messages are passed. A message can take arbitrary long time to arrive at its destination.
- As stated before, there is **no global clock**. One process can make no assumptions about the clock in another process.

2.1 Group partitioning

When considering the previous characteristics of the environment for *GCom*, a group of processes can at any time be divided in two groups without any means for communication between them. It would be impossible for the groups to determine whether the group members of the other group still executes and behaves normally. Therefore a partition of a group is treated as a crash of all the members cut off. This means that merging such a group when communication can be achieved again is done by a new join for all the members in one off the groups.

2.2 Member failures

A member of a group is considered to have failed only when throws a *RemoteExceptioin*[4] as defined by *Java RMI*. This means that *GCom* makes no guarantee about the time it takes to send a message to a group. This guarantee could be achieved simply by changing the definition of a member failure to include a time-limit for message delivery.

2.3 Group discovery

When a process wants to communicate with other processes using *GCom* there must be a way to find groups and group members already existing. That starting point is defined by a global address known by all *GCom* members. This starting point will contain a service for group discovery, described in more detail in section 4.1.

[4] `http://java.sun.com/javase/6/docs/api/java/rmi/RemoteException.html`

3 Usage

All files needed to use this middleware are located at: `~/dit06ajn/edu/dist/GCom`

This catalog contains the following sub directories:

- The directory `src` contains the source code.
- The directory `src/main/resources/` contains configuration files for standard behaviour of the compiled system, see section 3.2
- The directory `src` contains the source code.
- The directory `bin` will, after a successful compilation, contain all the compiled sources as well as configuration files used by this middleware.
- The directory `lib` contains all requires third-party libraries needed by *GCom*, se section 3.3.
- The directory `doc` contains the Javadoc API for *GCom*.

3.1 Compilation

The following commands will require the software tool *Apache Ant*[5]. More details about what happens using *ant* in this project is found in the file *build.xml*[6].

To compile *GCom* issue the following command:
`salt:./GCom> ant`
This will create a directory `bin` if it does not already exists and compile/move source-code and configuration files to that directory.

The root-directory for class-files when using *GCom* is compiled to *bin/main/java*, while the root-directory for test-code is compile to *bin/test/java*.

To create *jar*-file of the compiled sources issue the following command:
`salt:./GCom> ant jar`
This will create *GCom.jar* which can be used when developing in third party software or directly as a *GNS*-server (see section 4.1) by running:
`salt:./GCom> java -jar GCom.jar`

3.2 Configuration

The compiled system uses two configuration-files to define its standard behaviour, these files are located in the directory *src/main/resources/*.

[5] http://ant.apache.org/
[6] http://ant.apache.org/manual/using.html

3.2.1 application.properties

The file *application.properties* defines the standard multicast and ordering types to use when communication with a group. Notice though that these settings are only used for the creator of a group that did not exist from before. When connection to an existing group, the settings from that group will suppress the settings in *application.properties*. CodeSnippet 1 shows the content of an example configuration that uses

CodeSnippet 1 applications.properties

```
# Used by GNS
gcom.gns.port=1078

# FIFO, TOTAL_ORDER, NO_ORDERING,
# CASUAL_ORDERING, CASUALTOTAL_ORDERING
gcom.ordering=FIFO

# BASIC_MULTICAST, RELIABLE_MULTICAST
gcom.multicast=RELIABLE_MULTICAST
```

3.2.2 logback.xml

The file *logback.xml* defines the behaviours of logging when using *GCom* and *Logback*, see section 3.3.1. The settings in this file is ignored by default if a system property *logback.root.level* is set to a specified logging level, e.g. *logback.root.level=OFF* turns all logging off.

Every class has a separate logger-name consisting of its fully qualified class-name. This means that logging can be configured per class or package. For example to only print debug information from the *se.umu.cs.jsgajn.gcom.management* package you could use the configuration shown in CodeSnippet 2. For more information configuring logback check their manual[7].

CodeSnippet 2 logback.xml

```
<configuration>
   <!-- ... -->
   <logger name="se.umu.cs.jsgajn.gcom.management">
      <level value="${logback.root.level:-DEBUG}" />
   </logger>

   <root>
      <level value="${logback.root.level:-OFF}"/>
      <appender-ref ref="STDOUT" />
   </root>
</configuration>
```

3.3 Required libraries

Basic functionality of *GCom* requires no extra libraries other than the standard edition *Java 6* platform. However for some extra functionality *GCom* internally uses some third party software located in the *lib* directory and described in the following sections.

[7] `http://logback.qos.ch/manual/configuration.html`

3.3.1 SLF4J and Logback

For logging capability *GCom* uses *Simple Logging Facade for Java (SLF4J)*[8] which provides a facade for different implementations of logging frameworks. The logging back-end used by default is *Logger*[9]. This combination provides logging capabilities to get information about *GCom*s status at run-time.

3.3.2 JUnit

For testing the individual parts of *GCom*, tests are written using the *JUnit testing framework*[10]. The tests cover some special parts parts of the system where unexpected results would otherwise be very hard to debug. All tests are located in the folder *src/test/java/* and are compiled to *bin/test/java/*.

To run all tests issue the following command:

```
salt:./GCom> ant test
```

Note hoverer that some tests require that some ports are note bound on *localhost* by other processes, and therefore the tests can fail because of bind exceptions.

4 System description

The *GCom* middleware is separated in three different modules to separate different behaviours, see figure 1. The first module which will have the closest connection to implementing software is the *management module*. This module handles group membership changes and actions. The second module *ordering module*, handles message ordering. The third module *communications module* is the one that actually sends and receives messages to and from the group. These modules will be discussed in more detail in the following sections.

4.1 Group Name System

To act as an entry point for group members in a *GCom* system there must be an instance of *se.umu.cs.jsgajn.gcom.GNS* running on a machine with a known address. The *GNS* acts as a Group Name System service which means that it resolves a group name consisting of string to an instance of the remote interface of the group leader.

The *GNS* is *GCom* only critical point of failure. If the *GNS* crashes no new group members can join groups without an instance of any group members remote interface.

If the *GNS* where to crash it can be started once again with a serialized object of the groups it contained before the crash. This is done with the following command:

```
salt:./GCom> java -jar GCom.jar GroupSettingMap.ser
```

The previous command provides the *GNS* with a file *GroupSettingMap.ser* which is saved by the last instance of a running *GNS*.

4.2 Group management module

The *group management module* is the top one in the application stack, see figure 1. Software using the *GCom* middleware will should handle all communication through this module by creating an instance of it and passing an instance of *se.umu.cs.jsgajn.gcom.Client* to the constructor of the *Group management module*.

A newly created *group management module* will initialize all other modules needed for a fully functional *GCom* application stack.

The default implementation of the *group management module* is implemented in the class *s.u.c.j.g.management.ManagementModuleImpl*.

[8] http://www.slf4j.org/
[9] http://logback.qos.ch/
[10] http://www.junit.org/

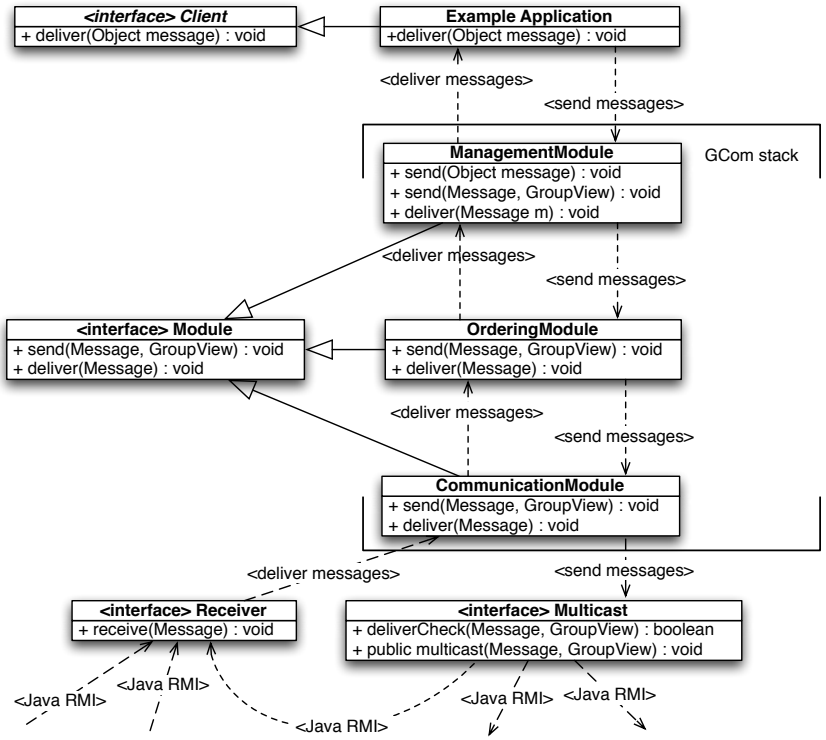

Figure 1: GCom stack

4.2.1 Group leaders

4.2.2 Error handling

4.2.3 Group changes

4.3 Communications module

4.3.1 Basic multicast

4.3.2 Reliable multicast

4.4 Message ordering module

4.4.1 Non-ordered

4.4.2 FIFO

4.4.3 Causal

4.4.4 Total

4.4.5 Causal-Total

4.5 Debugger

5 Limitations

6 Tests

References

[DKC05] Jean Dollimore, Tim Kindberg, and George Coulouris. *Distributed Systems: Concepts and*

Design (4th Edition) (International Computer Science Series). Addison Wesley, May 2005.

A Appendix