



College of Engineering, Design and Physical Science
Electronic and Computer Engineering

Assignment
**Java RMI Programming for Distributed
Systems**

Distributed Computing Systems Engineering Msc

Author: Christoph Gschrey

Date: 26. January 2018

Supervisor: Mr. Müller-Hofmann

ASSIGNMENT SUBMISSION FORM

Please note: that no course work will be accepted without this cover sheet.

Please ensure: that you keep a copy of work submitted and retain your receipt in case of query.

Student Number:	SPO ID Number (Office use only):	
Course:		Level:

MODULE	
Module Code:	Module Title:
Lab / Assignment:	Deadline:
Lab group (if applicable):	Date Stamp (Office use only):
Academic Responsible:	
Administrator:	

Please note: that detailed feedback will be provided on a feedback form.

✂.....

RECEIPT SECTION (Office Copy)	
Student Number:	SPO ID Number (Office use only):
Student First Name:	Student Last Name:
Module Code:	Module Title:
Lab / Assignment:	
Lab group (if applicable):	Deadline:
Academic Responsible:	Number of Days late:

DECLARATION	
I have read and I understand the guidelines on plagiarism and cheating in the Handbook and I certify that my contribution to this report fully complies with these guidelines. I confirm that I have kept a copy of my work and that I have not lent my work to any other students.	
Signed:	Date Stamp (Office use only):

✂.....

RECEIPT SECTION (Student Copy)	
Student Number:	Student Name:
Lab / Assignment:	
Lab group (if applicable):	Module Title:
Academic Responsible:	Deadline:
Module Code:	Date Stamp (Office use only):

The University penalty system will be applied to any work submitted late.

IMPORTANT: You **MUST** keep this receipt in a safe place as you may be asked to produce it at any time as proof of submission of the assignment. Please submit this form with the assignment attached to the Department of Design Education Office in the Michael Sterling Building, room MCST 055.

Contents

1	Introduction	1
2	Network Programming in Java	2
2.1	Sockets in Java	2
2.1.1	TCP Socket	2
2.2	Java Remote Method Invocation	2
3	Job Server/Client Architecture	4
3.1	Overall Architecture	4
3.2	UDP-Multicast	4
3.3	The Job Server	4
3.4	The Job Client	4
3.5	Workload-Balancing	4
4	Other possible solutions	5
4.1	Server Broadcast	5
4.2	Workload Balancer	5
5	Conclusion	6
	Bibliography	7
A	Appendix	i

1 Introduction

In today's world, where increasingly large amounts of data have to be processed and exchanged by distributed systems, the requirements for communication between distributed systems are becoming more and more complex. Not only performance is a problem, but also the requirement to make the services of the distributed systems available to several users at the same time. In other words, every system that is part of the architecture of a distributed system must be able to communicate with every other subsystem at the same time.

There are several technologies in the Java environment that can be used for communication within such a distributed system. The most basic variant is the use of Java Sockets. But this assignment favors the Java Remote Method Invocation. This report considers both technologies and presents a possible solution for a distributed system in which both of them play a role.

2 Network Programming in Java

This chapter explains the theoretical basis for this assignment. The first section focuses on Java sockets, while the second deals with Remote Method Invocation (RMI). The aim is to highlight the differences as well as the advantages and disadvantages of both technologies.

2.1 Sockets in Java

A socket is an endpoint of a bi-directional communication connection between two programs running in different processes on the same computer or on different computers in a network . One socket takes on the role of the server, while the second socket acts as a client. Socket classes are used in Java to represent the connection between a client program and a server program. Sockets provide a simple API and require a hostname and a free port number to be initialized. If two or more sockets are running on the same computer, the hostnames and portnumbers must be different. Before the two processes (no matter if they are running on different computers or not) can communicate with each other, the client and server sockets must perform two different tasks.

First of all, the client must know the hostname and port number of the server in order to connect to it. How exactly this is done depends on the programmer's decision. The server's address data can be stored permanently in the client's program code or can be retrieved by other techniques such as DNS. When establishing a connection between client and server, the client must also identify itself by sending its own host address to the server. The only thing the server has to do is to listen to possible clients attempting to open a connection.

After the connection has been established, data can be transferred in both directions between server and client. The next two chapters show two different socket types and their differences.

2.1.1 TCP Socket

In general there are two different socket types. The first is the **Stream Socket** whose communication is based on the Transmission Control Protocol (TCP).

2.2 Java Remote Method Invocation

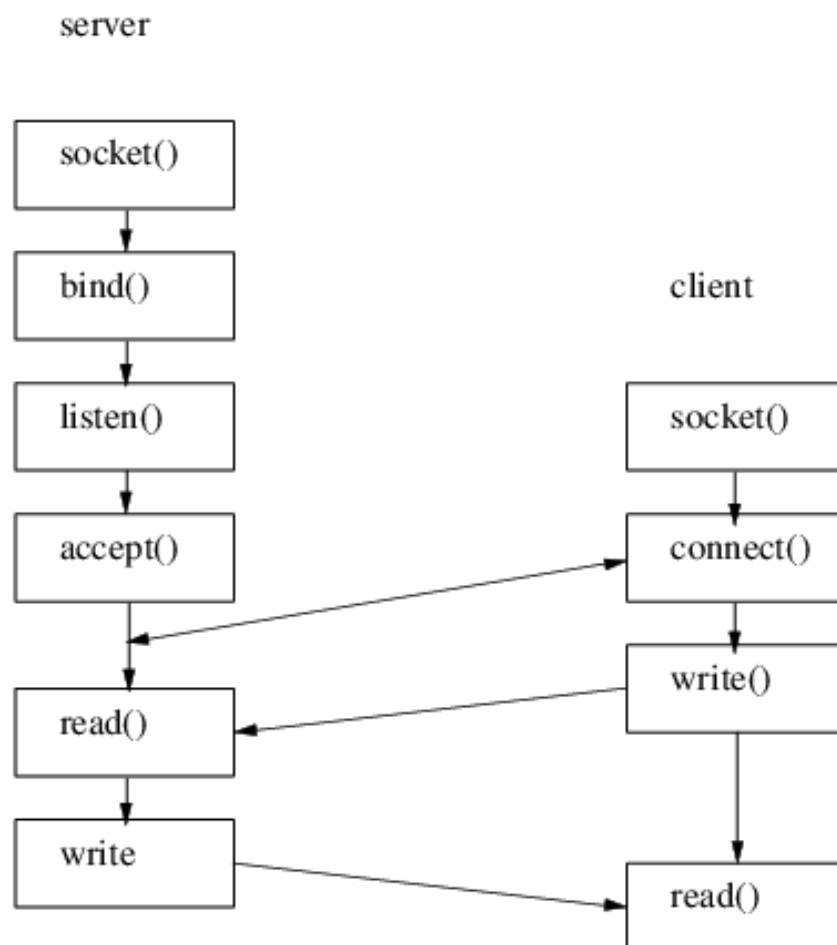


Figure 2.1: TCP communication between server and client

3 Job Server/Client Architecture

3.1 Overall Architecture

3.2 UDP-Multicast

3.3 The Job Server

3.4 The Job Client

3.5 Workload-Balancing

4 Other possible solutions

4.1 Server Broadcast

4.2 Workload Balancer

5 Conclusion

Bibliography

- [Bec04] Kent Beck. *Test-Driven development by Example*. Pearson, 2004. ISBN: 8131715957.
- [WMV03] Laurie Williams, E. Michael Maximilien, and Mladen Vouk. “Test-Driven Development as a Defect-Reduction Practice”. In: *14th IEEE International Symposium on Software Reliability Engineering ISSRE 2003*. Denver, Colorado: IEEE, 2003. URL: <http://ieeexplore.ieee.org/document/1251029/> (visited on 10/01/2016).

A Appendix