Distributed Systems

October 4, 2012

COMP512 Distributed Systems

School of Computer Science

McGill University

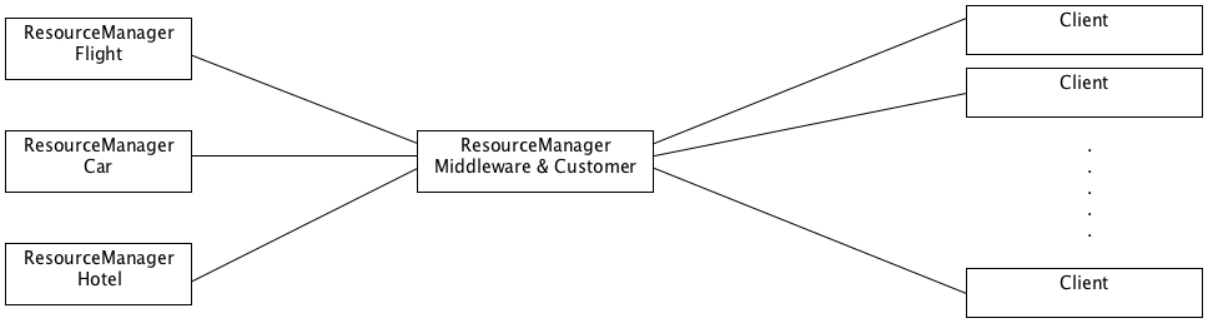
Table of Contents

Section 1: Introduction

Purpose of Document

This document's purpose is to give an overview of the design of a possible implementation of a distributed system.

General Diagram



The following modules are suggested for this implementation:

- ResourceManager is responsible for dealing with some type of data. It could be Flights, Cars, Hotels.
- Middleware is responsible for handling requests from clients. It will decide if it should request another ResourceManager for another piece of information. It will also be responsible for handling the list of customers.
- Clients will be able to connect to the middleware and make requests regarding any info available in any ResourceManager.

Observation: The diagram above shows how the ResourceManagers and clients will communicate. Since the RMI framework provides a stable and reliable communication we won't be discussing it in section 2. We will mainly focus on the TCP/IP implementation.

Section 2: The Design

2.1 Design Goals

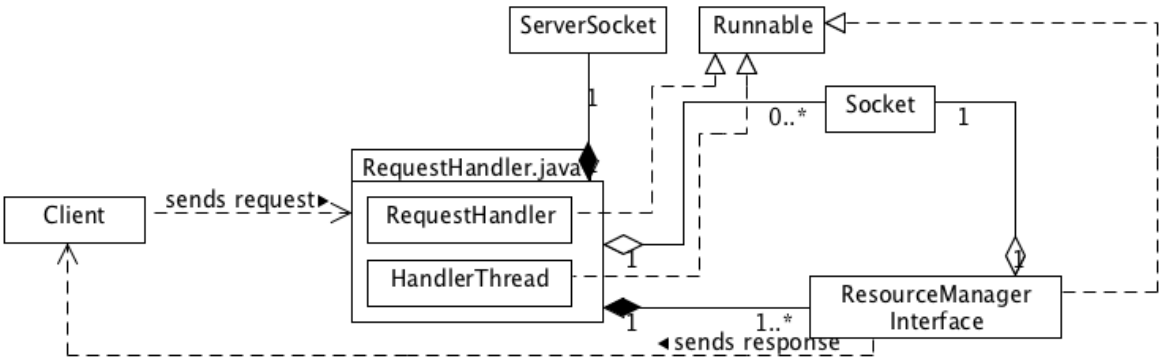
All design and engineering decisions will be taken with the following principle in mind:

Simple is better than complicated.

We will always try to eliminate choices that will make the usability and implementation of the application complicated.

2.2 Architecture

2.2.1 Class diagram



The above diagram is a simplified class diagram of the system. The diagram focuses on the Request Handler Module. The core classes of the request handler are the RequestHandler, the HandlerThread classes, and an implementation of the ResourceManager interface (ResourceManagerImpl or MiddlewareImpl).

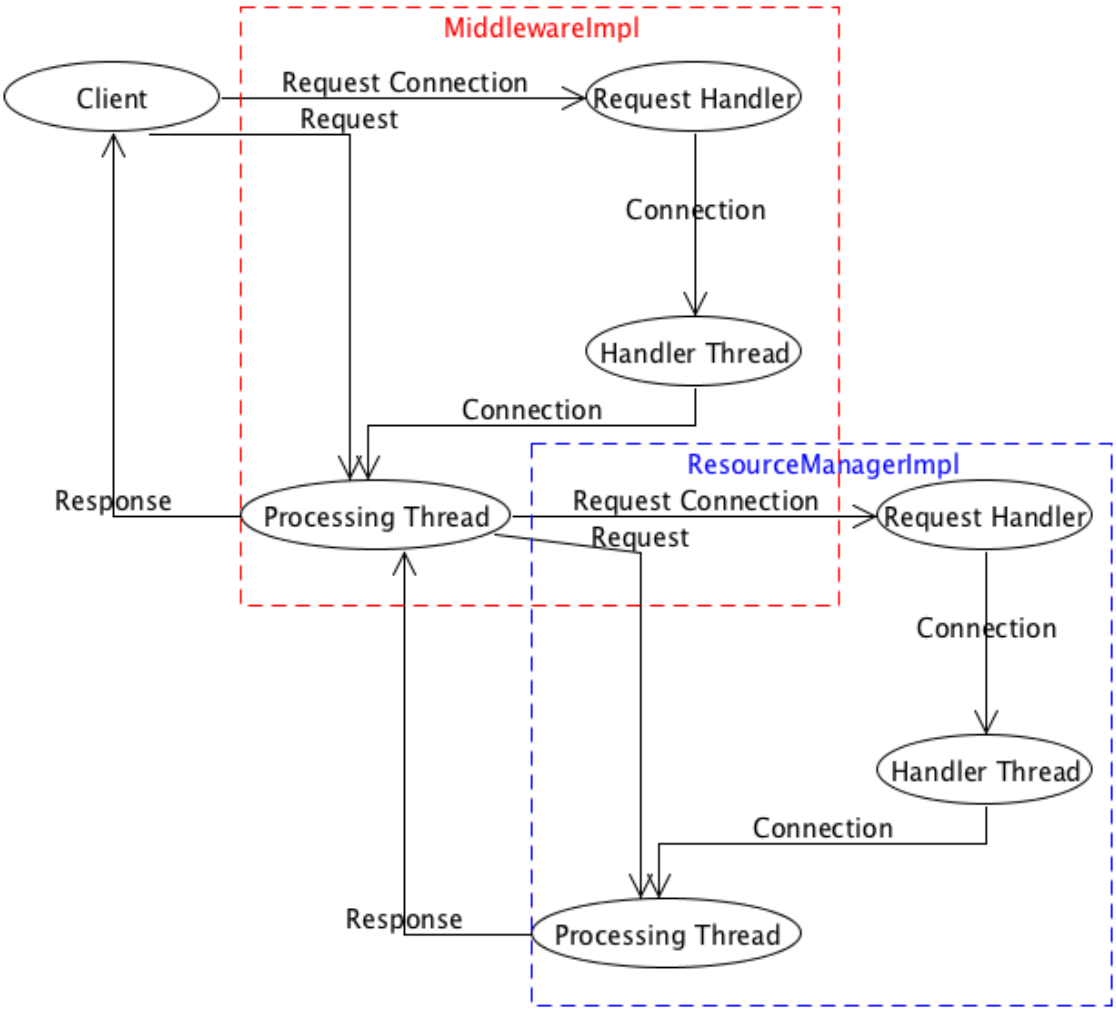
The Request Handler is responsible for handling new connections. The Client-Server pattern can clearly be seen with the interaction between the Client module and the Request Handler module. The client sends messages via a network connection to the Request Handler. Upon receiving a message from the client, the Request Handler does some execution and may return a message to the Client. The Client and the Request Handler are independent of each other and there can be several Clients connecting to the Request Handler at once.

There is only one instance of the RequestHandler and its HandlerThread running per server. This is necessary since there should only be one Socket listening for client connections for the application.

For simplicity we will call ProcessingThread an instance of the classes MiddlewareImpl or ResourceManagerImpl.

Internally in the ProcessingThread, a sequence of operations must be done on a received message. First a message is read by the ProcessingThread. This message is then passed to a parsing method. The parsing method with break the message up and identify the request. Once this is done, the request is passed along to the necessary methods for execution. During execution of the request, additional information may be retrieved/request from another ResourceManager. Once all information is gathered, execution is completed. The end result of execution is formatted and then returned to the client.

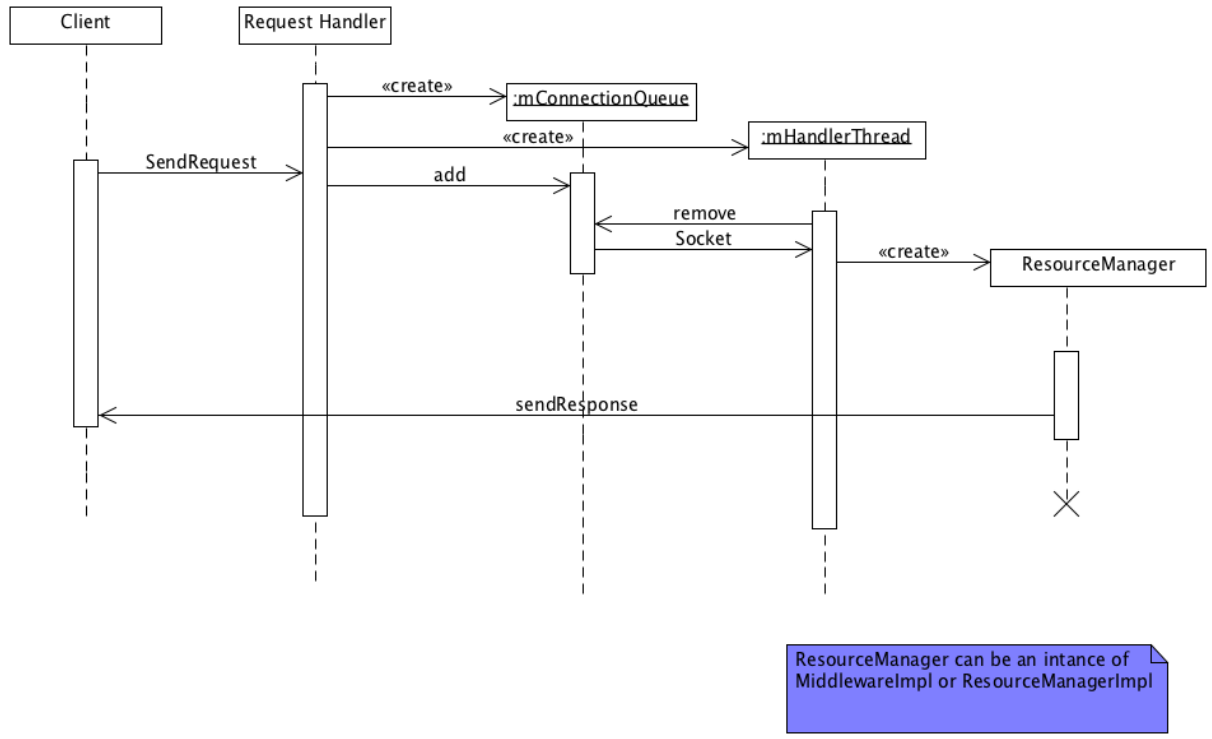
An example of this communication can be seen in the dataflow diagram below



Each component of the RequestHandler uses Sockets from the java.net library to communicate with the Client. Only the RequestHandler uses a ServerSocket to accept connections from the Client. The RequestHandler has two sub-components: the RequestHandler and the HandlerThread. The HandlerThread is an inner class of the RequestHandler. The RequestHandler, HandlerThread and ProcessingThread all run in independent threads, all implementing the Runnable interface, to allow multiple requests to be handled at the same time.

A Client will first make a connection with the RequestHandler. This connection will be added to a queue of connections waiting for processing. The HandlerThread goes through the queue of connections and starts a ProcessingThread for each connection. The ProcessingThread then finishes the interaction with the Client, making extra request to another resource if required.

2.3 Execution After Client Connection



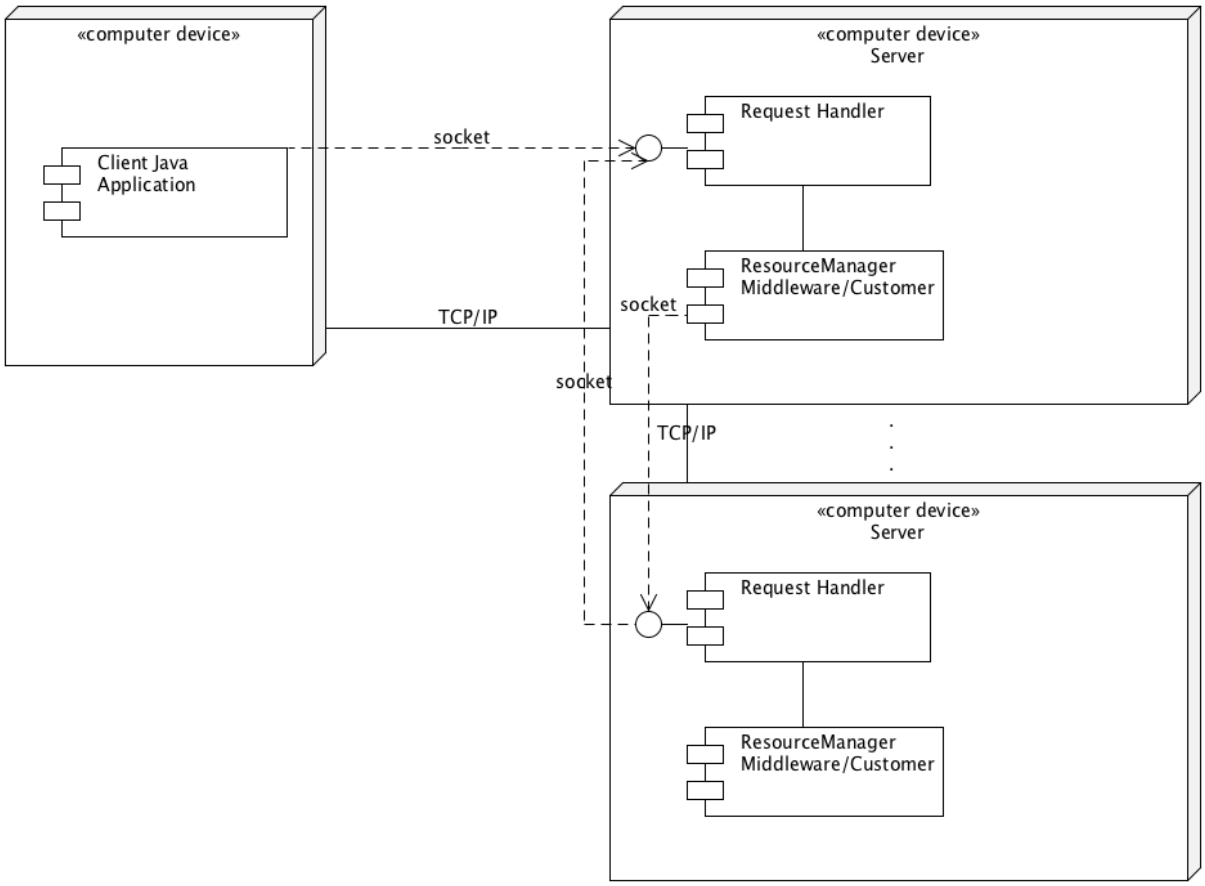
Upon startup, the RequestHandler creates its list of connections for the Client as well as starting the HandlerThread. Once the RequestHandler has been set up, it can accept connections from the Client module. The Client module makes a connection and sends a request to the RequestHandler. The RequestHandler adds this connection to the ConnectionQueue upon success. The HandlerThread then takes the connection and starts a ProcessingThread to interact with the Client connection. At this point, there are several different paths of execution which could lead to another request/connection to another resource manager, if that happens the

current server becomes a client. After execution has completed, the ProcessingThread returns the results to the Client and terminates.

2.4. Installation

2.4.1 Installation diagram

The diagram below exemplifies how the environment is set up.



3. Tests

- Method Testing: All methods were tested to ensure that they were working.
- Connectivity Testing: Connection testing was done to make sure that not only the modules could communicate locally but also in 3 different networked computers
- Stressing and concurrence Testing: Automated script was develop to make multiple requests so we could verify that the system can handle multiple clients at the same time