COMP 512 Project
Final Report

Milestone 3 - 2-Phase Commit

Group 30
Oliver Clark - 260674845
Matthew Etchells - 260680753

November 7 2018

Contents

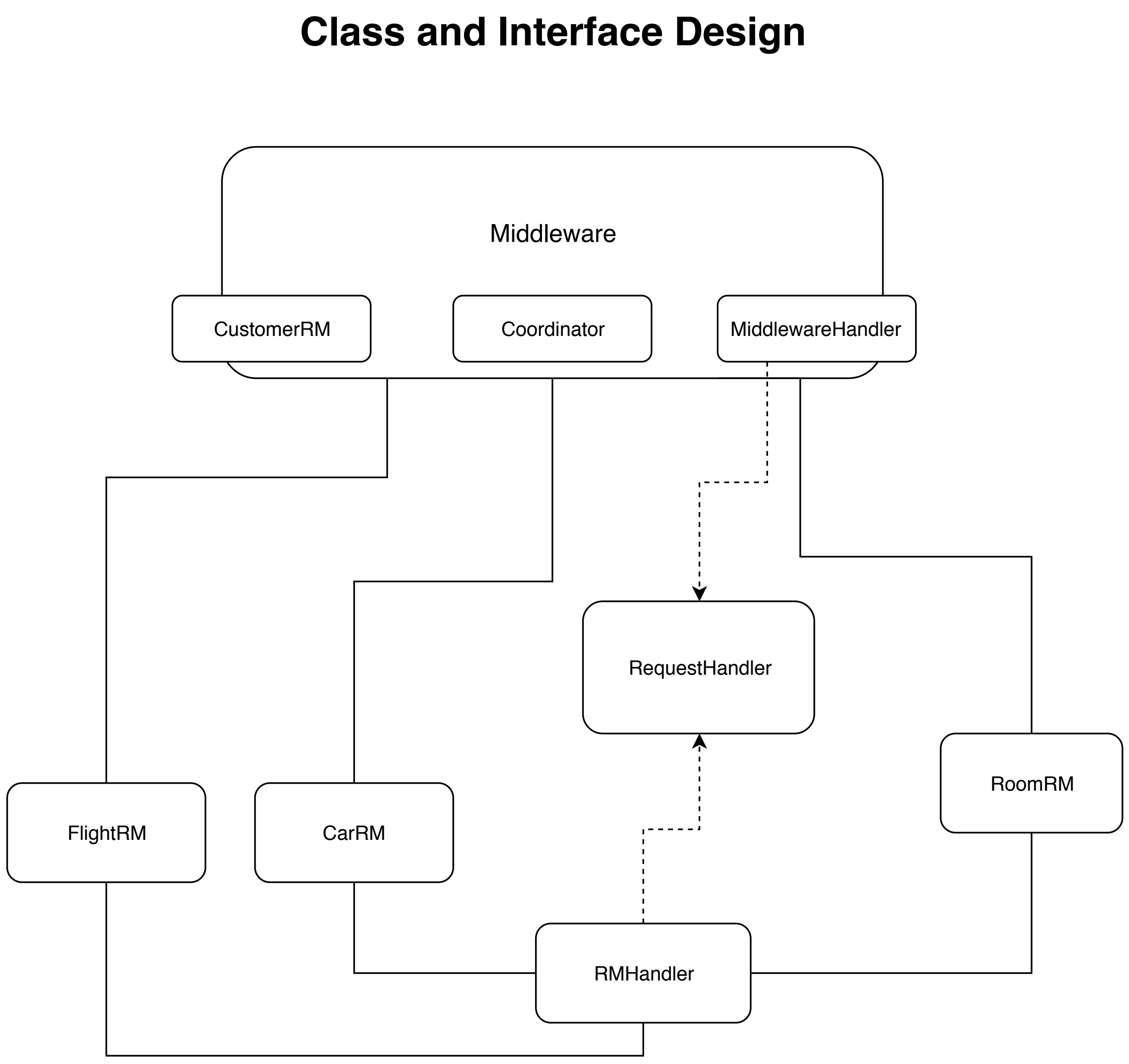
1 Introduction

The following paper describes the final version of the COMP-512 Fall 2018 course project. The project is a distributed reservation system to allow a user to reserve flights, cars, rooms, or some combination of the three.

2 TCP Socket Distribution

In order to implement the distrubted system, TCP was used due to its relevance and wide use across many Computer Science fields.

The first decision to be made in implementing the system using TCP Sockets was to define a high-level communication protocol between the components. In order to keep the system relatively simple, Objects were used to for all levels of communication (client - middleware, middleware - resource manager). This was implemented using ObjectInputStream and ObjectOutputStream. The following system diagram describes the high level TCP architecture between system components.



3 Lock Management

Strict 2 phase locking was used as a locking protocol. This was implemented using distributed lock managers located at each resource manager, as well as the customer resource manager (located at the Middleware). Customer data was replicated across RMs meaning that operations involving a customer required locks at all RMs. For non customer operations, a single lock at the corresponding RM is used.

3.1 Lock Conversion

Lock conversion was implemented in the following simple and straightforward manner. Lock conversion occurs when a transaction holding already holding a shared lock requests an exclusive lock. However, the lock can only be converted to an exclusive lock if and only if no other transaction currently holds a shared lock. In this case, the requesting transaction releases its shared lock on the resource before receiving an exclusive lock for said resource. In the other situation, (another transaction exists, holding a shared lock on the requested resource) the requesting transaction cannot have its lock converted, and must wait accordingly. Finally, if a transaction already holds an exclusive lock to a resource, and requests another exclusive lock to the same resource, the request is considered redundant and ignored

4 Transaction Management

For the final version of this project, we used a strict 2 phase locking protocol in order to control access to resources. Furthermore, we used a 2 phase commit protocol (reliant on user commands (start), (commit,xid), (abort,xid)). Upon receiving a request to commit a transaction, the coordinator (Middleware) asks all participants involved in the transaction if they can commit. If all participants can, a 'DoCommitRequest' is sent, telling the participants that they can go ahead a perform a commit. In order to preform an abort, a Resource Manager must simply throw away any changes, and revert to the most recent stored version (the one pointed to by the master record).

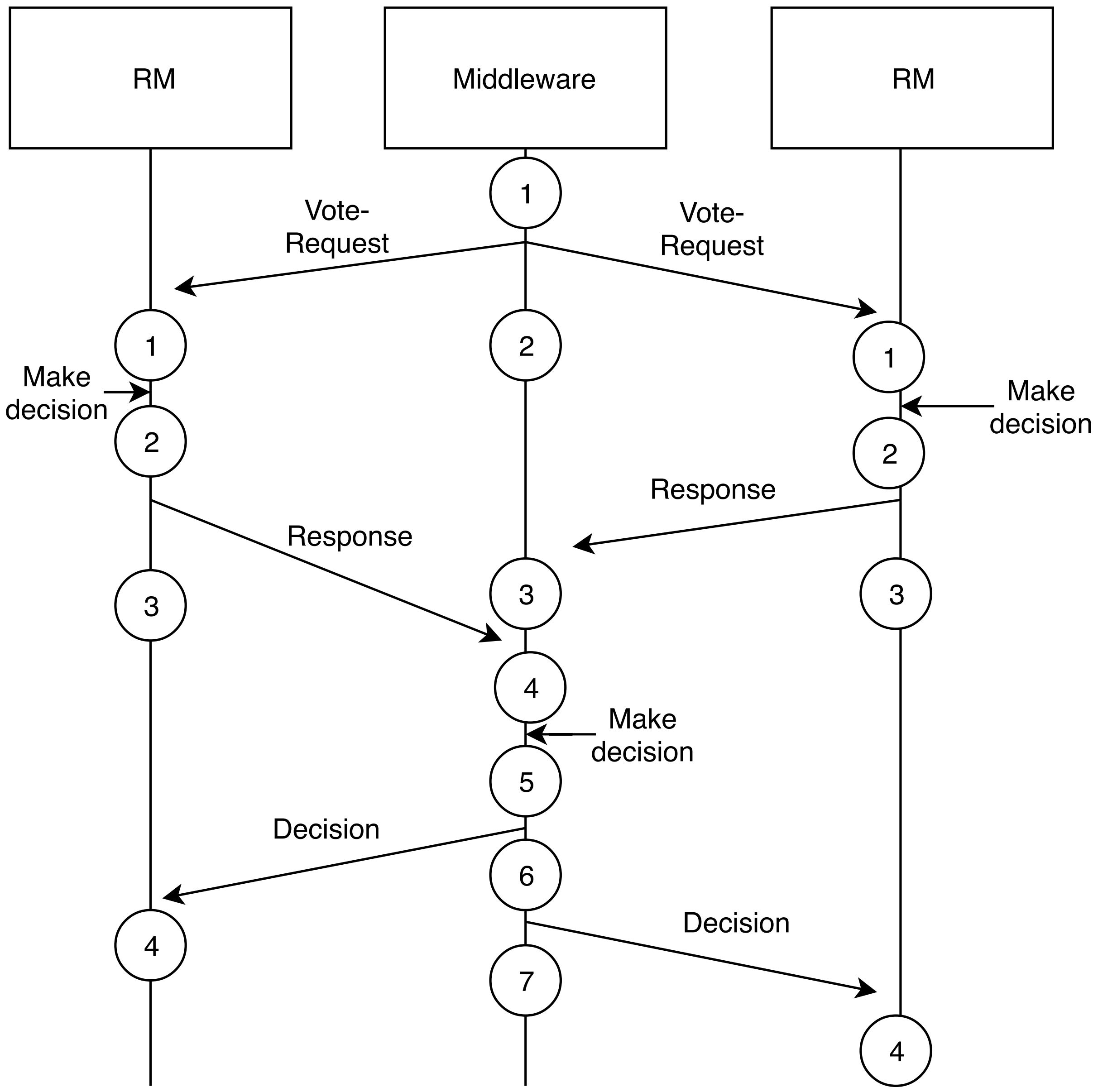
5 Failure Management

5.1 Shadowing

In order to achieve data persistence a shadowing protocol was used. To achieve this, two versions of the data were stored at each Resource Manager, versionA and versionB. Finally, a master record was stored also stored at each Resource Manager. The master record always contains one key-value pair, with a key denoting the location of recent version (versionA or versionB), and a value denoting the last transaction to commit the key version. When staging commits, the version NOT pointed to in the master record is updated with the most recent data. When finalizing commits, the master record is updated to point to the version updated during the previous staging phase.

5.2 Recovery

For this system, recovery was implemented at the Resource Managers. Recovery ensures that the correct data is stored at the Resource Manager upon restart after a crash (there is existing commands to force crashes at different locations in the Resource Manager, (CrashFlightRM,mode),(CrashRoomRM,mode),(CrashCarRM,mode). During recovery, reads from persistent storage the status of transactions. Using these statuses, the recovery chooses how to proceed, either continuing, or aborting the transaction (and informing the Middleware coordinator). The following diagram shows the various different crash modes available at the Resource Managers and the Middleware. Crashes are referred to by their integer mode identifier.


RM
Middleware
RM
1
Vote-Request
Vote-Request
1
2
1
Make decision
2
Make decision
2
Response
Response
3
3
3
4
Make decision
5
Decision
6
4
Decision
7
4