SOEN 423
Design document for distributed-HCMS

SOEN 423

November 29, 2019

Table 1: Group

Name	ID Number	Email
Morteza Ahmadi	40038235	morinob93@gmail.com
Yashar Dabiran	40042187	yashardabiran75@gmail.com
Jj Devies	40009083	jj@seivad.ca

Table 2: Revision history

Version	Date	Changes
1.0	29th November 2019	Completed documentation

1 Introduction

1.1 Purpose

The software design document describes the architecture and system design of the distributed health management system. It includes a class diagram, an architecture diagram. It's intended to to communicate the design of the system to software developers and to coordinate its implementation.

1.2 Scope

This project is planning to provide a worldwide platform to facilitate the all the processes of an appointment for clinics as well as hospitals.

2 Class Diagram

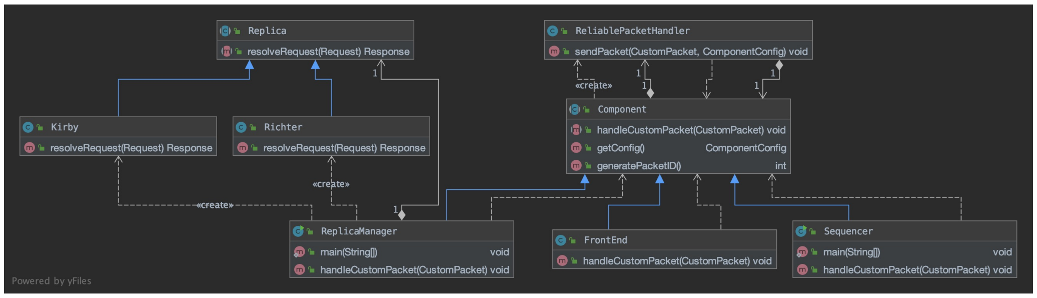
Replica
resolveRequest(Request) Response
ReliablePacketHandler
sendPacket(CustomPacket, ComponentConfig) void
Kirby
resolveRequest(Request) Response
Richter
resolveRequest(Request) Response
Component
handleCustomPacket(CustomPacket) void
getConfig() ComponentConfig
generatePacketID() int
«create»
«create»
«create»
1
1
1
1
1
1
ReplicaManager
main(String[]) void
handleCustomPacket(CustomPacket) void
FrontEnd
handleCustomPacket(CustomPacket) void
Sequencer
main(String[]) void
handleCustomPacket(CustomPacket) void
Powered by yFiles

Figure 1: Class Diagram

2.1 Description Of Class Diagram

Component

Component is an abstract class which is the parent class to all the components. *FrontEnd*, *Sequencer*, *Replica Manager*, and *Replica* extend *Component*. A *Component* has a *Reliable Packet Handler* object which enables the component to send packets reliably. In Addition, a component provides an interface for its children classes a way to handle a request.

Front End

Front End is front-end CORBA. Client sends an CORBA object to the *FrontEnd* containing which request to handle. Once all the reponses has been received, the *FrontEnd* then decides which response is correct by simply comparing all the received reponses. Lastly, it sends the correct reponse to the client.

Sequencer

Sequencer keeps track of all the requests and that was sent by the *FrontEnd*. When *Sequencer* receives a request, it adds a label to that request which is a number, then the request is added to the history of all requests. Finally it multicasts the request to all the replicas.

Replica Manager

Replica Manager is responsible for managing requests and call the replica to handle the request. *Replica Manager* keeps track of the requests in the form a priority queue whose head is the oldest received request. Once a request has been received it is compared to the last received request in the request queue, if the difference in their label is more than one it indicates that the request has been received out of their order and *Replica Manager* waits for the missing request. It will fail based on whether the missing request has been received.

Replica

Replica handles all request that has been passed down by *Replica Manager*. On top of every *Replica* there is a *Router* class which decides which server should handle the incoming request. Once the request is passed to the desired server, servers will communicate through UDP servers.

Reliable Packet Handler

Reliable Packet Handler is the crucial part of the system. It is responsible for sending and receiving packets reliably. It consists of two functions *Send Packet* and *Start Listening*. As long as an acknowledgment message is not received in *Start Listening* the *Send Packet* function will send the packet every 500 miliseconds for 5 seconds. Once a packet is received in *Start Listening* function, it will be handled by the component class.

3 Architectural Diagram

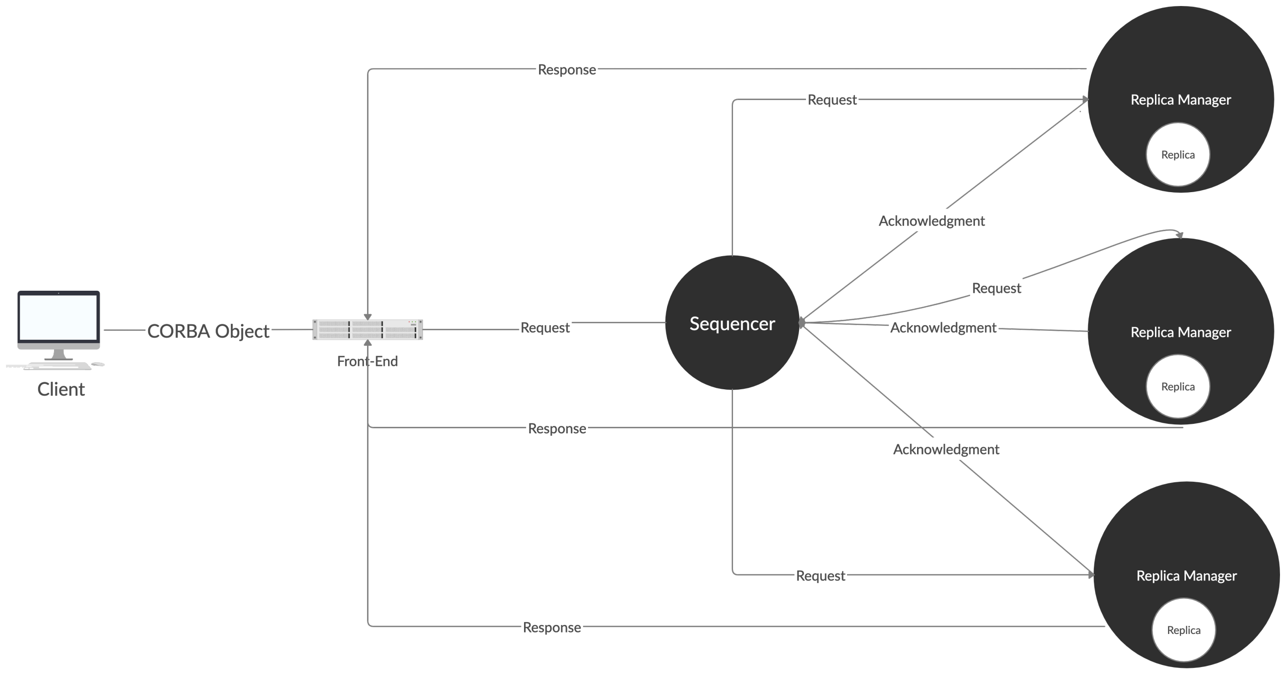

Figure 2: Architectural Diagram