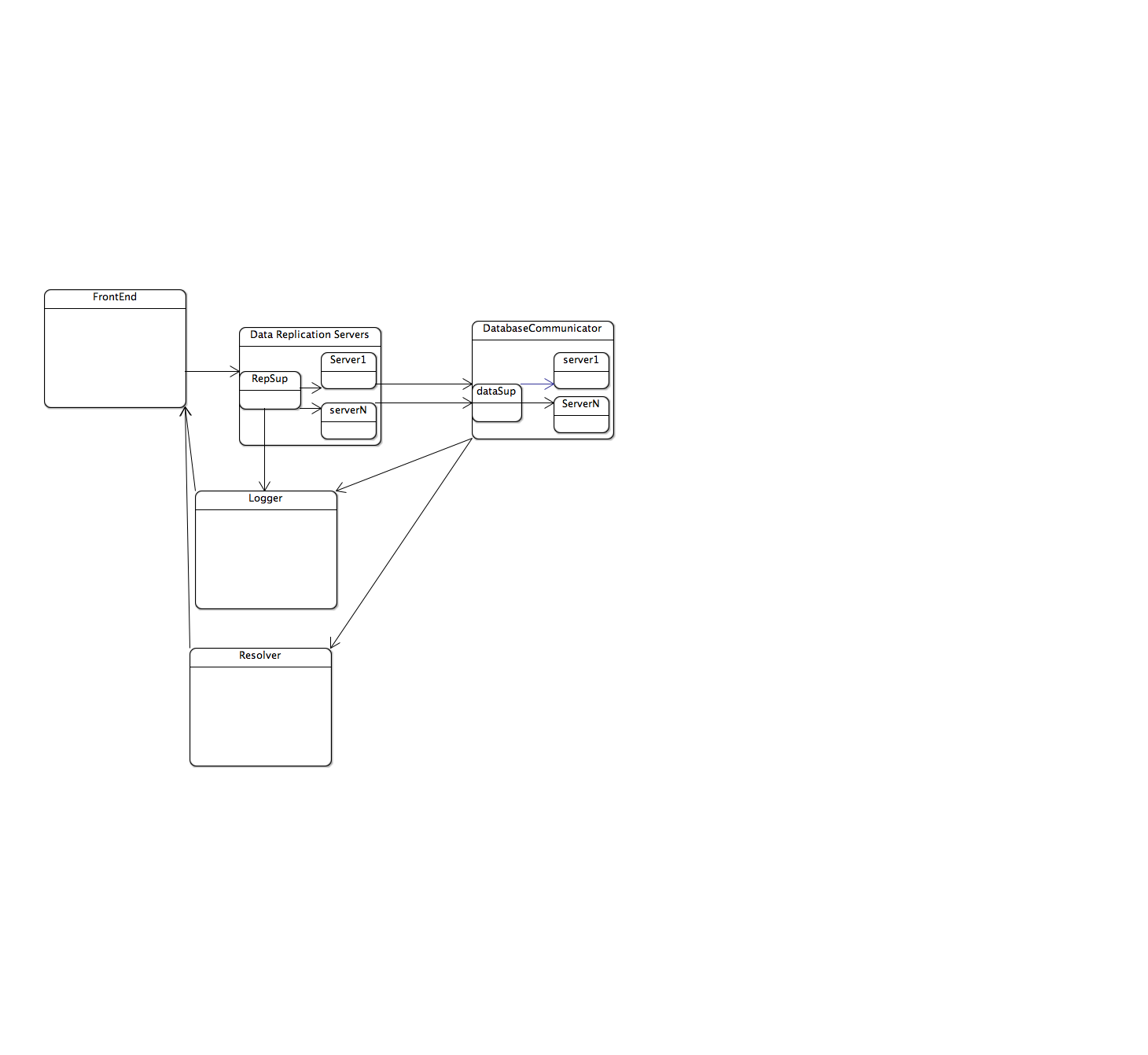
Jack Davey

**Eventual Consistency Design Document**

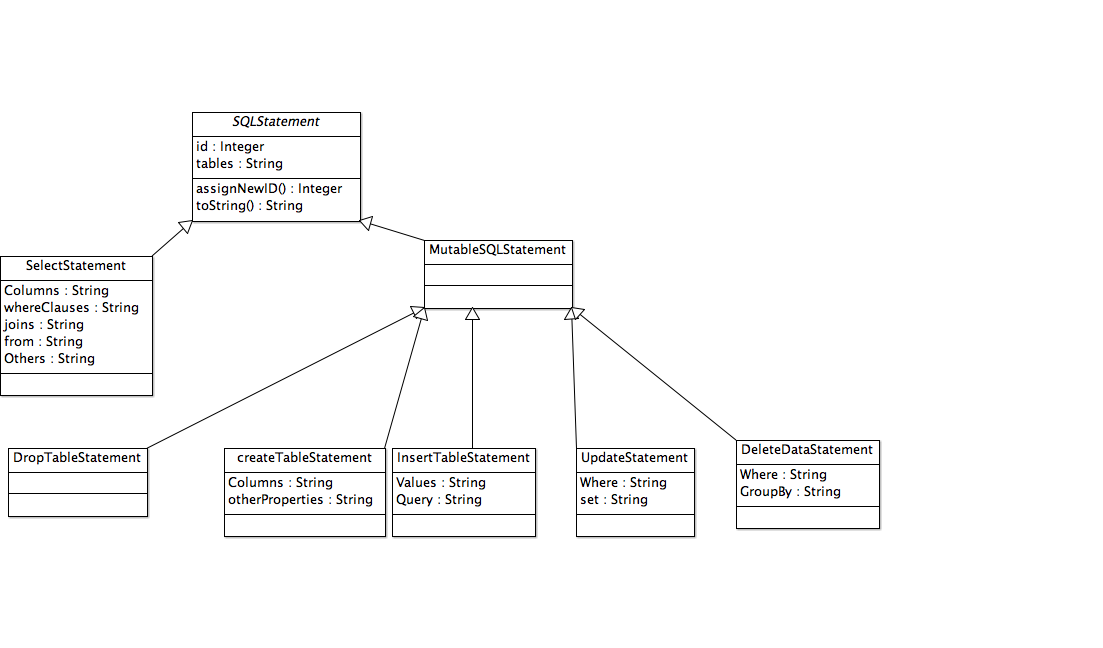
This document sets out the overall architecture for my system. As I will be using the Akka framwwork to implement this system, I decided the best way to represent my architecture would be using a process network diagram. This allows the to identify the main processes that the system is made up of and show how they communicate together.



I will now go through the diagram and give an outline of what each of these components will actually do.

**FRontEnd**

This is the only part of the system that may not be an Akka Actor, The reson for this si because it needs to communicate with the end user, and therefore the communication cannot be asynchronous. Its main responsibility is to take messages from the end user and send them onto the appropriate parts of the system. In doing this, it should also make sure that these pieces of data re encoded in the right form. This mainly consists of taking the web service requests that are coming into the system and translating those into an SQL object. These SQL objects have a version number associated with then, and it is these that we use to ensure eventual consistency. The class diagram for this is shown below



**State replication Servers**

These processes are responsible for keeping multiple copies of the same data, In a lot of the main query operations, these are queriesat random to find out what pieces of data they have available to them. All requests to these servers are initially sent to a supervisor actor. This supervisor passes jobs to the worker servers. At set intervals, these servers send data tot eh database server to make their data consistent.

**Database Communicator**

This process has the job of communicating with the database, it creates a new communicator actor for each new request it receives, It is also responsible for overseeing the process of ensuring eventual consistency. .

**Logger**

Because the actor model relies on asynchronous communication, a user may not always get all the information associated with a request. As an example, when an update is mad eto the database, that update may not be made fully consistent for a while after the user interaction has finished. In order to counter this, all processes in the system send messages to a logger process. Upon receiving a message from the frontend, the log can provide a textual representation of all the logging messages it has had.

**Resolver**

The resolver process is responsible for reporting all instances where the eventual consistency algorithm has had to choose one value over another. It receives notifications from the dataCommunicatoer process when this happens. Then upon receiving a message from the front end, it can send all these to the user. Another message to the front end will force this process to choose the discarded data state.

**Algorthms**

I will now look at the algorithms that I will use to achieve eventual consistency. Creating tables and dropping tables are done by sending the SQL object straight through to the database communicator. If we are dropping a table, then a message is sent to the replication table to ensure that all updates in the system relating

When it comes to removing, updating, or inserting a piece of data, then we start off by running the request on the backend database, but not committing it. If this works, then re send the data to the replication servers, if it fails, we send an error back to the user. This is done so that the user gets some feedback from the iitial request as to whther the update will or won’t be applied.

Once at the replication servers, a server is chosen at random,, the replication supervisor sneds the request through to the server. This kicks off the consistency algoritm that happens at all severs. If there have been no other updates within a period of time, stored as a sonstant, then the update counter is incremented, and we send the data onto all other servers so that they an do the same. If there were other other updates going then we get the vector clocks from all other replicas ans compare themk, if the new ones come befor eth eold, we can get rid of the new update and cary on with what we had going originally. If this fails, we just use the update that came in physically later, but send all values through to the consistency resolver so that the user can deal with it at a later stage.

For selects, the user runs the query on the database, we then choose on eof the replication servers at random, if the data from the query contains any data from the the sever, the stale data is replaced.