**Java Package**

**DysFys (Simple Distributed File system)**

**Done By :**

19pw06 - Balachander G

19pw28 - Sanjay T

**Problem description:**

**Aim:**

The objective of this project is to develop a distributed file sharing system

using the RMI framework in java.

**Description:**

It is required to implement a replicated file system. There will be one main server (master) and data will be partitioned and replicated on multiple replicaServers. This file system allows its concurrent users to perform transactions, while guaranteeing ACID properties.

**Design Assumptions:**

1. Master server never transfers data.
2. Creating a new file operation is atomic.
3. All files of active transactions can fit in memory of replica servers.
4. Files are small so that they can be returned as one bulk in response to read requests.
5. File chunks have a fixed size defined in Configurations class.
6. Writes must be performed through the primary replica of the file.

**UML Diagrams:**

1. **Class Diagram:**

System design is implemented as a class diagram as given below.The system design implements the following functions so in order to depict their functional responsibilities a class diagram is created based on the following entities:

* Master Server
* Replica Server
* client

These entities are represented as classes which in turn implements interfaces such as:

* ReplicaReplicaInterface
* ReplicaServerClientInterface
* ReplicaMasterInterface
* MasterServerClientInterface

Master Server

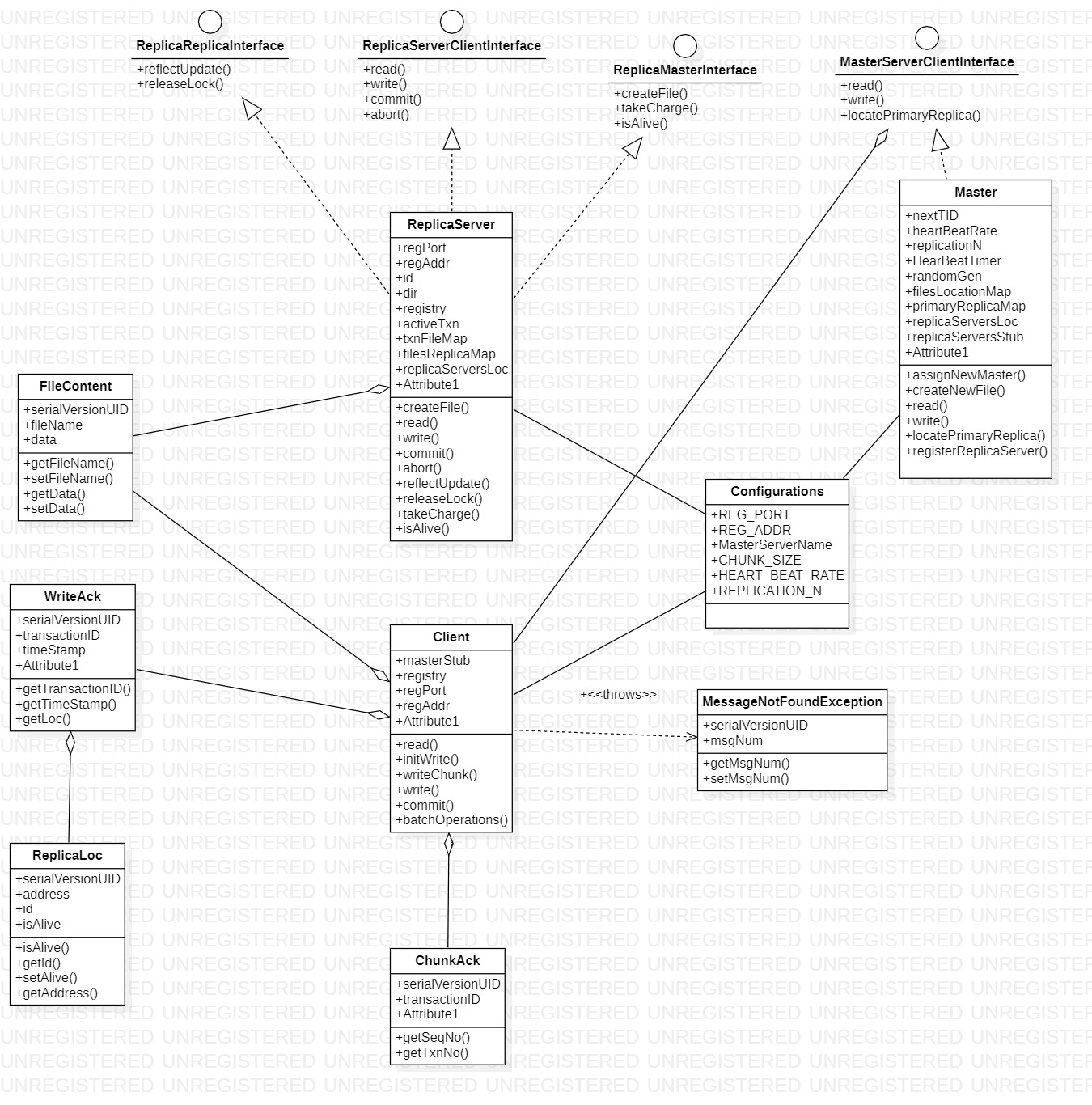
* Keeps track of replica servers addresses and state (up/down) .
* Saves meta-data about files (mapping between filenames and primary replica server responsible for that file, mapping between filenames and replica servers hosting a copy of that file).
* Regularly sends Heartbeat messages to the replica server to check their state.
* Grants r/w request from clients, on approval replies with:
* readAck < List of replicas having the files >
* writeAck < primary replica location, timeStamp, transaction ID >

Replica Server

* Keeps track of files that it is their primary replica, also keeps the location of other replica servers having those files.
* Make sure that the data of active transactions are not reflected in the system until the commit is called.

Client

* Contact the master for any approving r/w requests.
* If master approves communicates with the replica server(s) to perform the transaction.

****

**Fig: 1 Class Diagram**

1. **Sequence Diagram**

This sequence diagram or system sequence diagram shows server client interactions arranged in time sequence. There are three basic operational scenarios namely:

1. Read operation
2. Write operation
3. Commit operation

**I. Read Operation:**

* Client contacts the master server requesting read operation with file name.
* Master replies with a list of all replica servers having a copy of the file.
* Clients contact one or more replica servers asking for the file.
* If the file is currently being updated (write lock) the call at the Replica server blocks until the update operation is completed.
* Replica server sends the file to the client in bulk.

**II.Write Operation:**

* Client contacts the master server requesting write operation with file name.
* If the file is not found in the system, Master randomly selects 3 lucky replicas to save the new file and assigns one of them as the primary replica for the new file.
* Master initiates a CreateNewFile operation at each of the lucky replicas.
* Master informs the primary replica of the file to take charge of that file.
* Master replies to the client with a write acknowledgement containing primary replica location, time stamp, transaction ID.
* Client initiates a write operation with the primary replica using the transaction ID obtained from the writeAck.
* Clients send chunks of the file tagged with a sequence number for every message.
* For every chunk received at the primary replica, the server replies with a chunk back. contacting the same seq. number.
* The primary replica saves the received chunks in memory until a commit operation is executed.

**III.Commit operations:**

* Client request commit from the primary replica server of the file using the transaction ID obtained in the writeAck message.
* The primary replica initiates a reflect update operation at slave server (servers having copies of the same file).
* The client sends the file content to all slave servers.
* Slave servers lock the file and write the file content on disk.
* Primary replica server flushed the file to the disk after obtaining appropriate locks.
* Primary replica informs the slave replicas that the commit operation is done so that they can release locks obtained in step 4.

**Communication Between Entities:**

1. Master → Replica Server
2. Replica Sever → Replica Server
3. Client → Master
4. Client → Replica Server

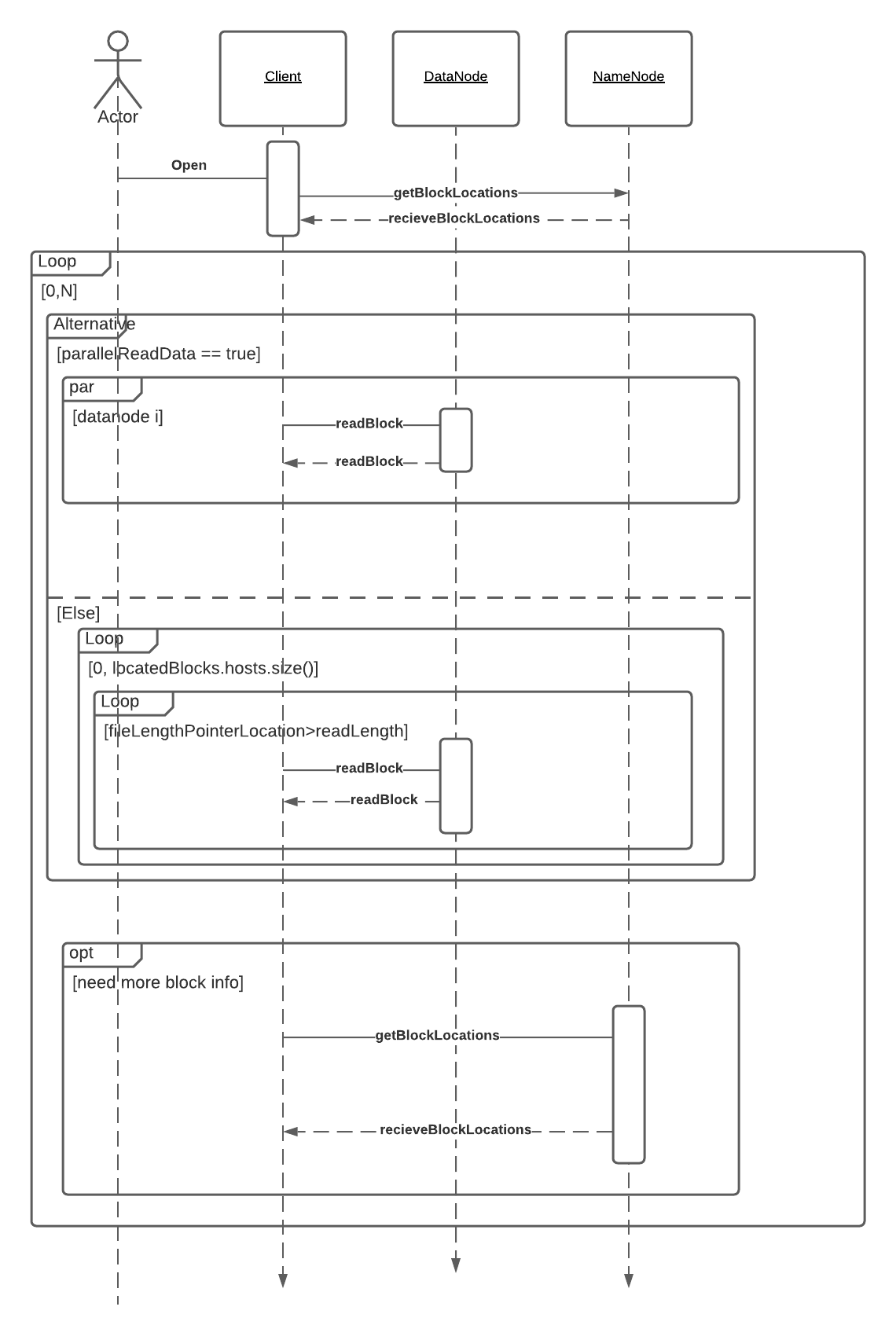
**Sequence diagrams to represented for operations are:**

2.1 Sequence Diagram during Read operation

2.2 Sequence Diagram during Write operation

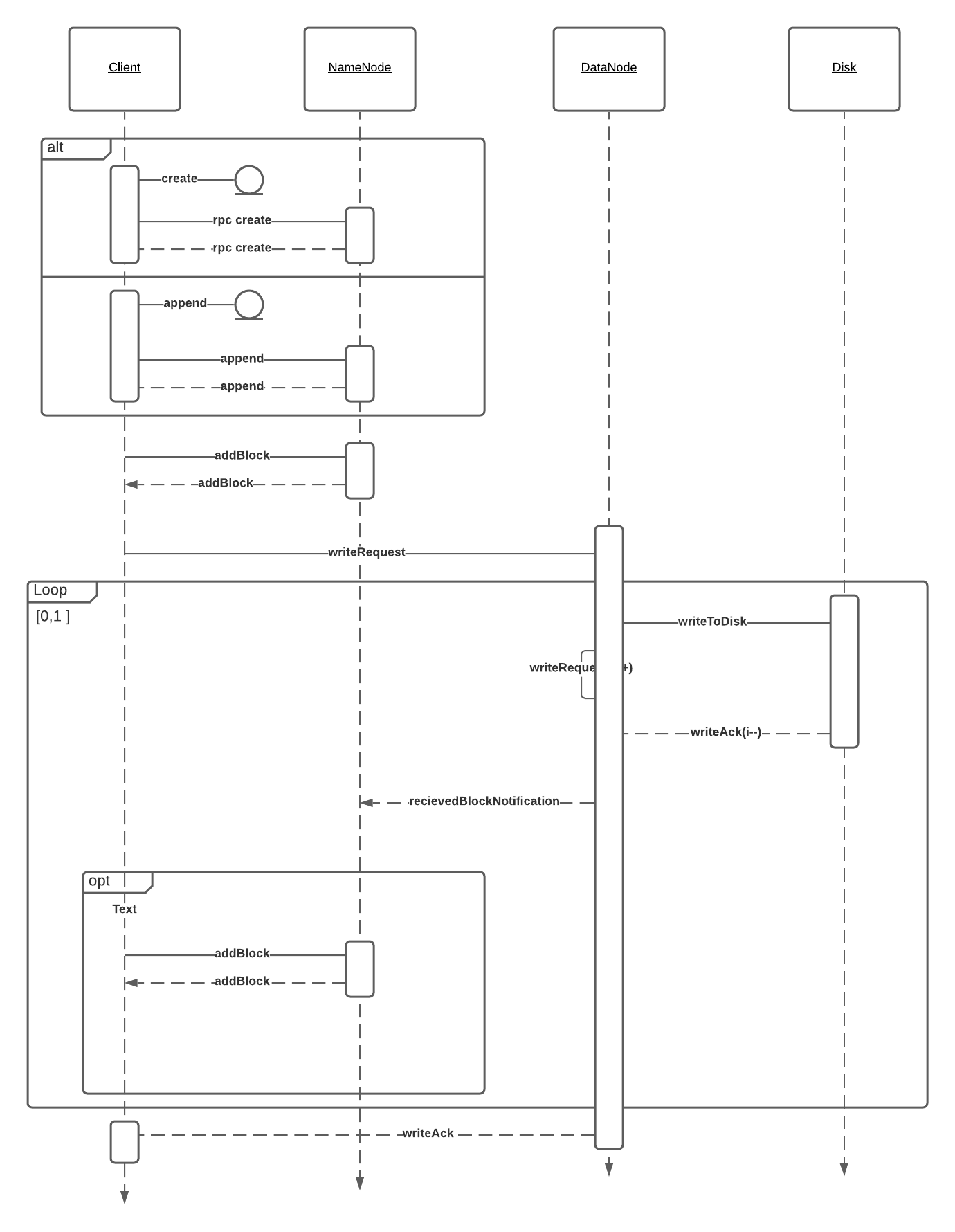
2.3 Sequence Diagram during Commit operation

**2.1 Sequence Diagram during Read operation:**

****

**Fig: 2.1 sequence diagram for read operation**

**2.2 Sequence Diagram during write operation:**

****

**Fig: 2.2 sequence diagram for write operation**

**2.3 Sequence diagram during commit operation**

****

**Fig: 2.3 sequence diagram for commit operation**