Telecooperation: Internet Lab.
2nd Assignment – Peer-to-Peer Networks and Distributed Hash Tables
Fernando Ruiz Vera - 1177767 - fruizvera@yahoo.com
You Yue - 1282746 - you@stud.tu-darmstadt.de

1 GENERAL DESCRIPTION OF THE SYSTEM.

A Class for the Graphic User Interface (GUI) was created with the name DHTStorage. This class is responsible of accepting the user commands and also display the information concerned to the actual node, which means the id of actual, predecessor and successor node in addition to the outgoing and incoming messages sent and received by the actual node respectively. The class DHTStorage contains a reference to the ChordNode class which runs in a separate thread.

Before enter into the details of the ChordNode class better is to introduce the ChordNet Interface which provide the signature for the four methods that must be supported by the Chord API as follows:

- *public void join(String HostAddress, int port, long hash)*
- *public void leave()*
- *public void store(String name, long hash, int value)*
- *public int retrieve(String name, long hash)*

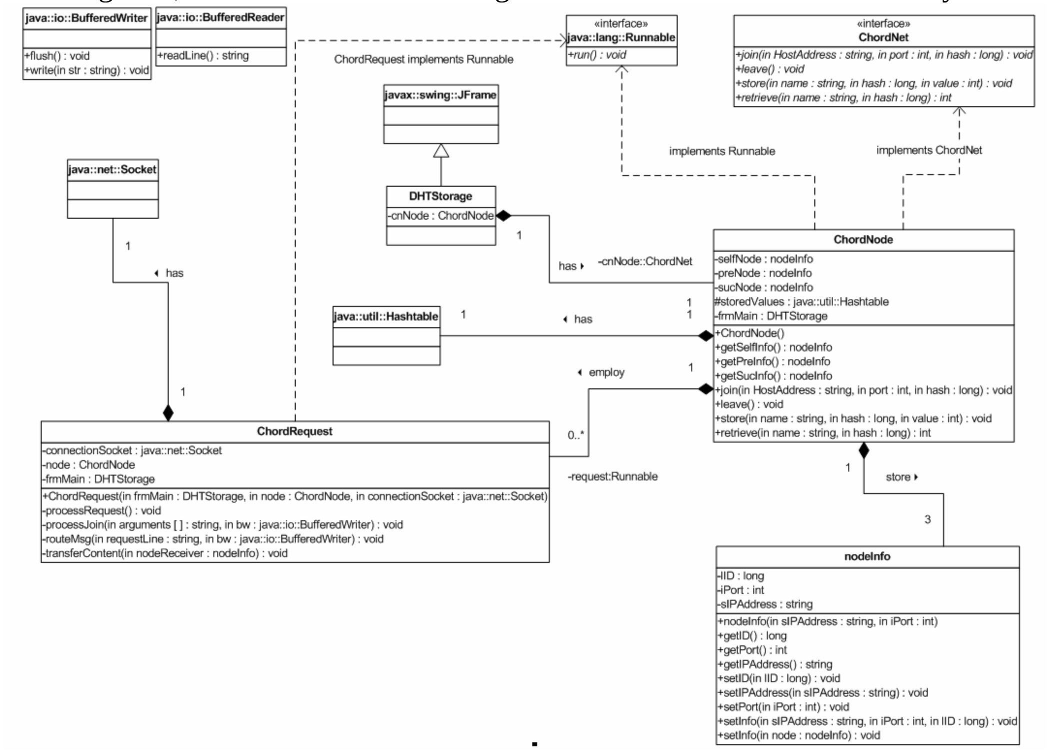
The class ChordNode implements two interfaces. First the java::lang::Runnable interface to allow the execution of this class in a separated thread into the GUI. The second interface implemented by the ChordNode class is the ChordNet interface already mentioned above. In consequence, the ChordNode class implements the method under the contract established by the Runnable and the ChordNet interface. To comply with the Runnable interface, the *run()* method is implemented where an instance of a ServerSocket is created on the port number corresponding to the actual node. This instance called *listenSocket* wait for incoming messages and is passed as a parameter to the ChordRequest Class.

The ChordNode class works as originating point for the four operation described by the ChordNet interface, which means that the ChordNode class create the messages JOIN, LEAVE, STORE and RETRIEVE, wait for the corresponding answers as JOIN_OK, DONE, OK, NOTFOUND and finally inform to the network about the new changes with the messages NEWNODE, TRANSFER, NODEGONE. In order to store the key-value pairs, each node contains a hash table.

In contrast to the ChordNode class, the ChordRequest class is responsible for the decision of process an incoming message or simply route it forward or backward. For certain group of messages (JOIN, STORE, RETRIEVE), the decision is made after parsing the message and check the hash value passed as parameter. For other messages as (NEWNODE, LEAVE, NODEGONE, TRANSFER), that have been sent over a direct connection to the destination node are processed without check any special condition.

In order to facilitate the management of the actual, predecessor and successor information in each node, another class called nodeInfo was created which has the hashcode, IP address and port number of each of them; and provide read and write access to them via get and set methods.

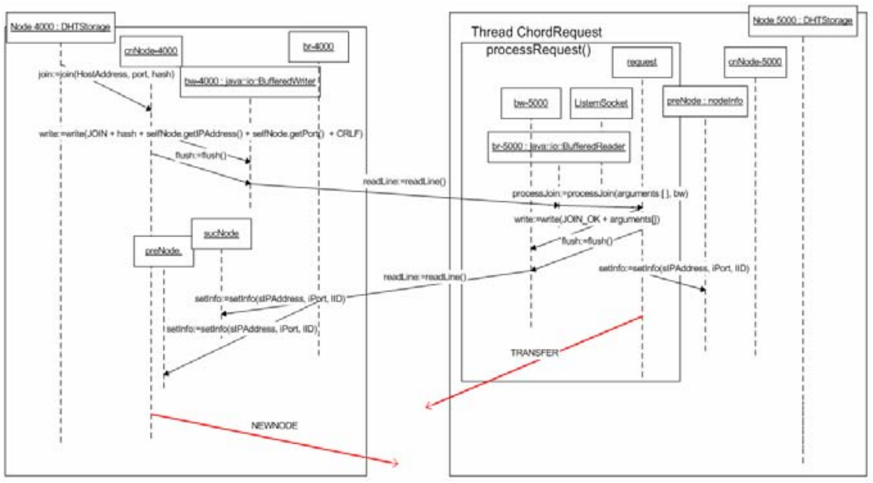
In the figure 1, it is showed the Class diagram with the static structure of the system.



java::io::BufferedWriter
+flush() : void
+write(in str : string) : void
java::io::BufferedReader
+readLine() : string
ChordRequest implements Runnable
«interface»
java::lang::Runnable
+run() : void
«interface»
ChordNet
+join(in HostAddress : string, in port : int, in hash : long) : void
+leave() : void
+store(in name : string, in hash : long, in value : int) : void
+retrieve(in name : string, in hash : long) : int
javax::swing::JFrame
implements Runnable
implements ChordNet
java::net::Socket
DHTStorage
-cnNode : ChordNode
1
has
-cnNode::ChordNet
ChordNode
-selfNode : nodeInfo
-preNode : nodeInfo
-sucNode : nodeInfo
#storedValues : java::util::Hashtable
-frmMain : DHTStorage
+ChordNode()
+getSelfInfo() : nodeInfo
+getPreInfo() : nodeInfo
+getSucInfo() : nodeInfo
+join(in HostAddress : string, in port : int, in hash : long) : void
+leave() : void
+store(in name : string, in hash : long, in value : int) : void
+retrieve(in name : string, in hash : long) : int
java::util::Hashtable
employ
0..*
-request:Runnable
store
3
ChordRequest
-connectionSocket : java::net::Socket
-node : ChordNode
-frmMain : DHTStorage
+ChordRequest(in frmMain : DHTStorage, in node : ChordNode, in connectionSocket : java::net::Socket)
-processRequest() : void
-processJoin(in arguments [ ] : string, in bw : java::io::BufferedWriter) : void
-routeMsg(in requestLine : string, in bw : java::io::BufferedWriter) : void
-transferContent(in nodeReceiver : nodeInfo) : void
nodeInfo
-lID : long
-iPort : int
-sIPAddress : string
+nodeInfo(in sIPAddress : string, in iPort : int)
+getID() : long
+getPort() : int
+getIPAddress() : string
+setID(in lID : long) : void
+setIPAddress(in sIPAddress : string) : void
+setPort(in iPort : int) : void
+setInfo(in sIPAddress : string, in iPort : int, in lID : long) : void
+setInfo(in node : nodeInfo) : void

Figura 1. Static Structure - Class diagram

Figure 2 shows the interaction between classes for the particular case of a join operation. Even this is a particular case the rest of operation work in a similar way. A detail not showed in this sequence diagram is the case where a message needs to be rerouted to the successor node and it is waited the answer message to be routed back.



Node 4000 : DHTStorage
cnNode-4000
br-4000
join:=join(HostAddress, port, hash)
bw-4000 : java::io::BufferedWriter
write:=write(JOIN + hash + selfNode.getIPAddress() + selfNode.getPort() + CRLF)
flush:=flush()
readLine:=readLine()
sucNode
preNode
setInfo:=setInfo(sIPAddress, iPort, lID)
setInfo:=setInfo(sIPAddress, iPort, lID)
NEWNODE
Thread ChordRequest
processRequest()
Node 5000 : DHTStorage
request
cnNode-5000
bw-5000
ListenSocket
preNode : nodeInfo
br-5000 : java::io::BufferedReader
processJoin:=processJoin(arguments [ ], bw)
write:=write(JOIN_OK + arguments[])
flush:=flush()
readLine:=readLine()
setInfo:=setInfo(sIPAddress, iPort, lID)
TRANSFER

Figura 2. Class interaction for a join operation.

2 SPECIAL DESIGN CHOICES

2.1 Hash calculation for Node's IDs and keys.

For both Node's identifiers and key of the stored strings, the calculation of the hash code must return an unsigned value. The following fragment of code shows how it is done for the IP Address; the same procedure is made for the key.

```
String sAddress = sIPAddress + ":" + Integer.toString(iPort);

lID = sAddress.hashCode() % Long.MAX_VALUE;
if (lID  < 0 )
        lID += Long.MAX_VALUE;
```

From certain length of the string the *.hashCode()* method return negative values in order to place them into the positive range the maximum of a Long type is added.

2.2 Initial State of nodes.

Right after the execution of the GUI main program allocated in the DHTStorage class, an instance of the ChordNode class is created and placed into a Thread. The ChordNode constructor method initializes the three references of type nodeInfo with the same information. One of these it is used to store the actual node information itself and the other two correspond to the predecessor and successor. Figure 2, left.

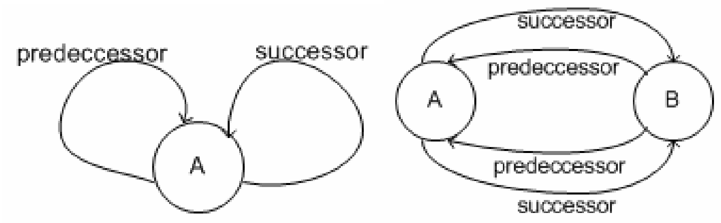


Figura 3. Initial state of a node (left). Interconnection state of the first two nodes (right).

2.3 Interconnection of the first two nodes.

Having two nodes initially connected with them selves, a join message is sent form one to the other. It is assumed that the nodes do not contain any key-value pairs in their corresponding hash tables at the initial state. Hence during the join transaction there is no TRANSFER message and without need to check any condition the predecessor and successor of each node point to the other. Figure 2, right.

3 WORKING WITH THE GRAPHIC USER INTERFACE.

In order to execute the main application a parameter with the port number where is desired to listen to the messages coming form other nodes must be provided. Also must be taken into account that a package with the name com.vbcolombia.p2p was created with the following simple structure as it is showed in the figure 3.

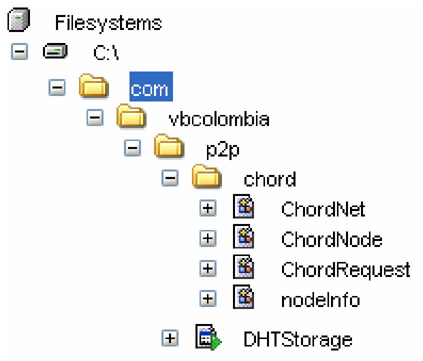


Figura 4. Package structure

From the command line it can be executed as:

```
> java -classpath "C:\" com.vbcolombia.p2p.DHTStorage <port number>
```

If the idea is run the program with the real values for nodes identifiers and key-value pairs the left side manage this kind of operation. At the right side three options exists for testing purposes whose operation is explained subsequently.

- Join with Id: this option is used when a node is desired to join the network with a specific identifier. For this case the identifier is replaced by the value given as Id and the Join message is sent to the specified node in the Connection panel.
- Retrieve this Id: this option create a RETRIEVE message with the value given as Id. If the value is found, it is showed in the Retrieve panel. Ignores the hash calculation of the string Name at the Storage panel
- Store at this Id: this option construct the STORE message with the value entered in the Storage panel and the provided Id. Ignores the hash calculation of the string Name at the Storage panel.

Every message that it is sent or received for a node is reflected in the textarea at the Local History panel. At any time, this textarea can be clean it up by clicking the Clear History button.

To finish, any change on the information about predecessor, successor and current node (only change when the option Join with Id is used), is reflected at the label of the Node Information panel.