NIO API

pedro.benedicteillescas@bsc.es

1 TransferManager calls

- **Initialize**

```
void init(String s, MessageHandler message)
```

Initializes connector s (for instance "NIO") and TransferManager with the associated MessageHandler.

- **Start server**

```
void startServer(Node n)
```

Opens a server in the node.

- **Start connection**

```
Connection startConnection(Node n)
```

Connects to a node.

- **Shutdown**

```
void shutdown()
```

Shutdowns TransferManager and NIO.

2 Connection calls

- **Send command**

```
void sendCommand(Object o)
```

Sends a command through this connection. The object sent must implement the Java interface Serializable.

- **Send data file**

```
void sendDataFile(String s)
```

Sends the file with the passed name through this connection.

- **Send data object**

```
void sendDataObject(Object o)
```

Sends a data object through this connection. The object sent must implement the Java interface Serializable.

- **Receive object**

  ```
  void receive()
  ```

  Receive an object through this connection.

- **Receive object/file**

  ```
  void receive(String s)
  ```

  Receive an object or file through this connection. If the data received is from a file, it will write it in disc with name s.

- **Receive object**

  ```
  void receiveDataObject()
  ```

  Receive a data object through this connection. This method ensures that the data received will be saved as an object, despite the format it had in its node of origin.

- **Finish connection**

  ```
  void finishConnection(Transfer t)
  ```

  Finishes the connection related with the transfer.

3 MessageHandler callbacks to implement

- **Initialize**

  ```
  void init();
  ```

  Initialization routine.

- **Error handler**

  ```
  void errorHandler(Connection c, Transfer t, NIOException e);
  ```

  The transfer could not be completed.

- **Data received**

  ```
  void dataReceived(Connection c, Transfer t);
  ```

  New data received. If the data destination is a file, it has already been written to disk. If the destination is an object, it can be reached using t.object.

- **Command received**

  ```
  void commandReceived(Connection c, Transfer t);
  ```

  New command received.

- **Write transfer finished**

  ```
  void writeFinished(Connection c, Transfer t);
  ```

  A write transfer initiated by this node has finished.

- **Conexion finished**

```
void connectionFinished(Connection c);
```

A connection finished by this node has successfully finished.

- **Shutdown**

  ```
  void shutdown();
  ```

  Shutdown method.

4 How to use the API

The NIO API is connection oriented. This means that in order to send or receive data a connection is needed. First, one node must start a server. Then another node can start a connection to that server. Once the connection has been started, the node that started the connection must send data. When the other node received the data, the appropriate callback will be called (commandReceived or dataReceived), and then the connection can be finished or reused to send data back.
It is important to note that a connection always needs to have something to do next. This means that after each sequence of sends, there must be either a receive or finishConnection.

5 Starting the NIO API

```
NIOTest nt = new NIOTest();
TransferManager.init("NIO", nt);
```

To start the NIO API create an instance of the MessageHandler implementation (in this case NIOTest). Then use the static method init of TransferManager.

6 Examples

- **Start a server**

  ```
  Node n = (Node) new NIONode(ip, port);
  TransferManager.startServer(n);
  ```

  Start a new Server that listens to incoming connections.

- **Start a connection and send a command**

  ```
  Node n = new Node(ip, port);
  Connection c = TransferManager.startConnection(n);
  c.sendCommand(object);
  ```

  Create a new Connection to the desired node. Then send the command (always serialized as an object) through the previously established connection.

- **Send a file through an existing connection and finish the connection afterwards**

  ```
  c.sendDataFile(fileName);
  c.finishConnection();
  ```

  Send a file with fileName name, and finish the connection when the file has been sent.

- **Receive a file or object through an existing connection**

  ```
  c.receive(fileName);
  ```

  Receive an object or file (with the specified name) through an existing connection.