

Contents

Introduction.....	1
Implementation Details.....	1
Three projects Summary:	2
• XMLReader Project.....	2
• PetrinetEngine Project.....	2
• NetExecutionResults Project	3
Project File Structure.....	3
Socket Communication in the three Projects.....	4
Using Semaphore in the Place class	7
Results	8
Execution Sequence	8
Conclusion	17

Figure 1: general execution Method	2
Figure 2: a figure showing the project files and structure	4
Figure 3:add method with semaphore control and thread.....	7
Figure 4: remove token via semaphore usage and thread.....	8
Figure 5: sample petrinet	9
Figure 6:XMLReader project execution completion.....	15
Figure 7: Project two final output	17

Table 1: XMLReader socket communication code	5
Table 2: PetrinetEngine socket communication code segment.....	6
Table 3: NetExecutionResults socket communication code	6
Table 4: sample XML file used in testing	9
Table 5: Project XMLReader initial output window test	12
Table 6:PetriNet Engine execution console output results test.....	14
Table 7:NetExecutionResults console output and results.....	16

Introduction

This project is about developing a simple simulation of a multiprogramming operation system that uses several features of OS developed including parallel programming, networking, scheduling.

We will be developing a simulation of Petri Nets that are read from XML file then built and processing using scheduling algorithms.

The final program will be split into three parts that communicate via TCP/IP protocol using sockets, where each part will do a separate processing simulating the distributed systems operation method.

Similar applications exist in every day application development in multi-tier programming models or in distributed system models in general.

Implementation Details

in the initial two deliverables we have developed a single application that reads XML files containing the Petri net description then build the net and execute it where transitions select tokens from input places based on the policy defined in this transitions which are the different scheduling algorithm (FCFS, SJF, EDF, SRTF).

The final output shows all information together inside one output window, including the XML components, the petri net building process and the execution results as shown in the appendix for a sample XML file and the resulting Output.

In the body report we will only concentrate on the final deliverable work.

For the final work, we have developed three projects using Java Netbeans 8, where each project is responsible for a single component and operation of the simulation as the following:

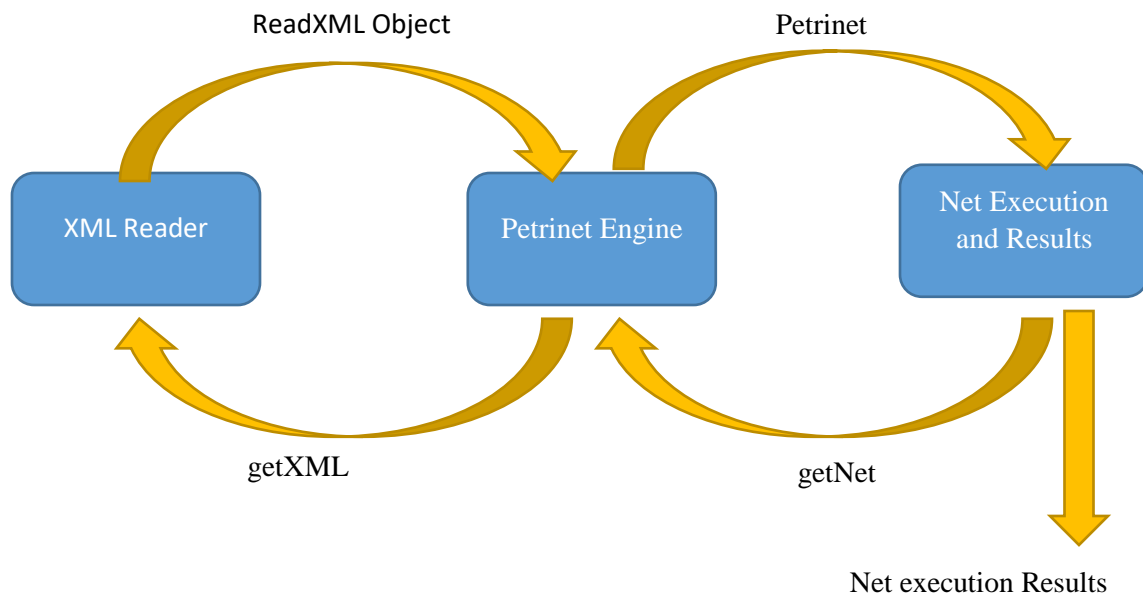


Figure 1: general execution Method

We must first understand that we created a package called “petri” which contains a set of classes that are used in all three projects (like a proxy classes), so that information transition is easy, and we transfer all data using objects instantiated from classes found inside this package.

All classes inside this package implement Serializable so that they can be transferred via TCP/IP sockets with ease and marshaling of classes from streams becomes as simple as casting.

Three projects Summary:

- **XMLReader Project**
 - It creates a Server Socket that listens on a special Port called XMLPort=13131, and accepts one connection that produces a Client socket which expects a special Message from the Protocol with value of "GetXML"
 - Once this message arrives, the XMLReader reads a local XML file containing the Petrinet data and constructs a special Object of ReadXML type that contains all necessary data (Tokens, Places, Transitions, Topology, Marking), however this data is flat and not connected and only reflects what is inside the XML file.
 - The output of this project is the ReadXML object sent to the Engine server and the list of XML elements that were ready printed to the project console.
- **PetrinetEngine Project**
 - It creates a client socket name xmlSocket that connects to XMLReader project server socket, sends “GetXML” message and gets ReadXML object.
 - It creates a server socket that listens on a special port called EnginerPort=13132, which is stored inside a class called Protocol as static value, Accepting connections from other classes and expecting a string message called “GetNet”.

- Once the “GetNet” message arrives it builds the Petrinet obtained from the XMLReader server and sends the ready to execute Petrinet objects to the Execution Client which connected the Engine server.
 - The output of this project is the Petrinet object sent to the execution server and the structure and linkage of the petrinet elements printed to the console of the application.
- **NetExecutionResults Project**
 - It creates a client socket and connects to the EnginerPort, then sends a text message called “GetNet” which will result in obtaining an object of type Petrinet that contains a fully connected Petri net that is ready to be executed.
 - It runs the Petrinet simulation by going through all transitions and firing them till no more firings possible.
 - The output is the firing and execution of the petri net with the scheduling algorithms in each transition
 - We didn’t know how to program the scheduling algorithms to produce the waiting time and response time for the entire net while following the petrinet execution sequence rules.

Project File Structure

In this section we talk a little about the project structure of files.

All three projects uses the same package called petri which contains the following list of Classes

Class Name	Class Description
Marking	Which marks the place with token
Petrinet	Which contains the final tree structure and contains a build method for building the Petri net based on the Topology. The Petrinet is built at the Engine
Place	The place which contains token, which has been developed in this Deliverable to contain Semaphore and two threads which will be explained more at another section
Protocol	Contains message names, port numbers, host name for all three projects and used uniformly in the three projects.
ReadXML	Which contain the XML file contents after being translated to list of Tokens/Marks/Transitions/Topolgoies/Places without being connected as objects.
Token	Token information
Topology	Links places to transitions
Transition	Contains scheduling algorithms, firing mechanism.

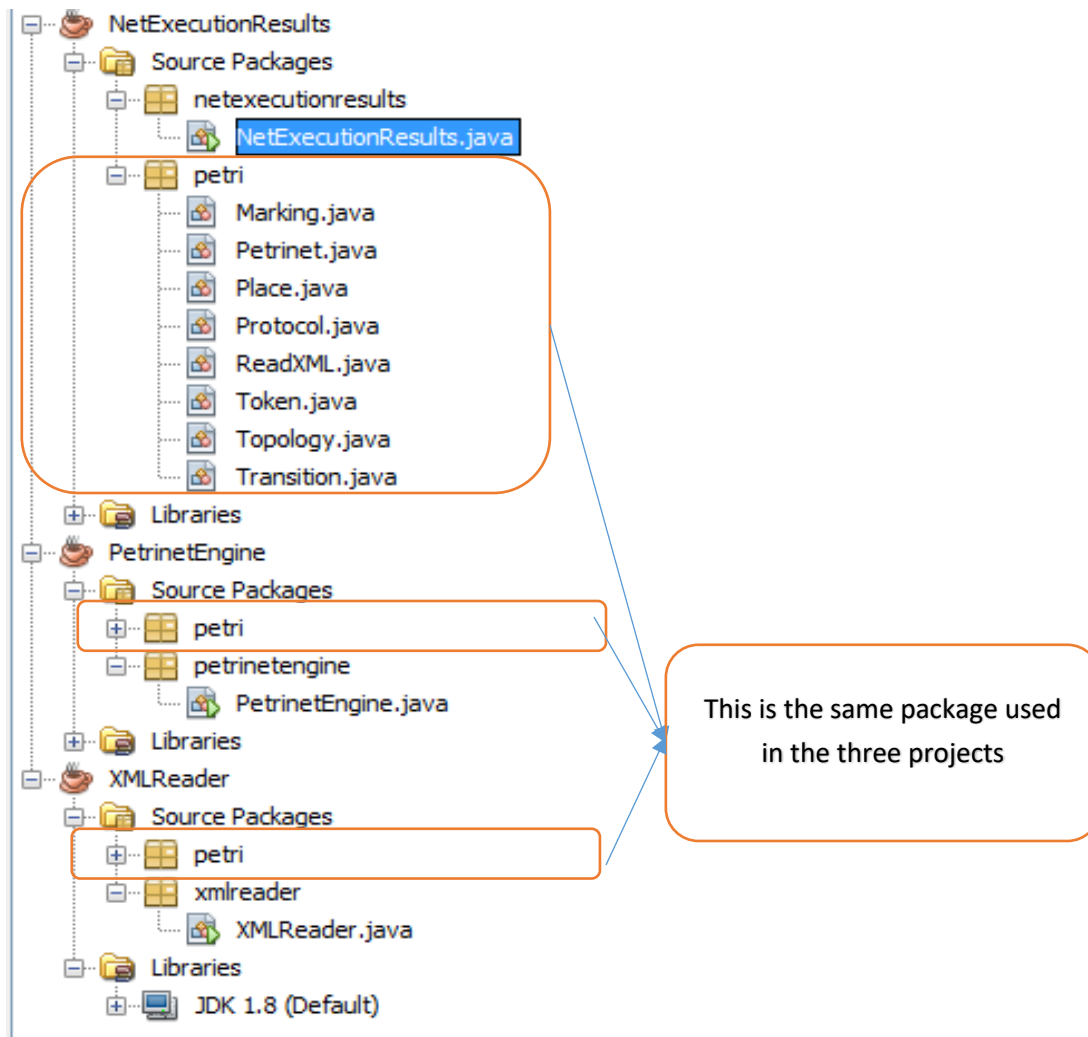


Figure 2: a figure showing the project files and structure

While for each project there exists one main file that contains the socket communication and execution of that project segment.

Socket Communication in the three Projects

All socket communication uses TCP/IP object streams for example:

As seen in the code bellow, we use `ObjectOutputStream` and `ObjectInputStream` in the data communication of the Socket which is the result of listen of the `ServerSocket` on the `XMLPort`.

Our execution also runs only once and exits the program after processing one `GetXML` message and reading one `XMLFile` which is passed via `Parameter` to the `Project` or the command line running of this application.

Table 1: XMLReader socket communication code

XMLReader Class of XMLReader Project has the following socket communication	
<pre> if (args.length==1) { reader.readXML(args[0]); //data.xml try { //run server and listen for incoming connections ServerSocket serverSocket = new ServerSocket(Protocol.XMLPort); System.out.println("Waiting For Engine XML Request..."); Socket clientSocket = serverSocket.accept(); System.out.println("Engine XML Request Received..."); ObjectOutputStream out = new ObjectOutputStream (clientSocket.getOutputStream()); ObjectInputStream in = new ObjectInputStream(clientSocket.getInputStream()); String message=in.readUTF(); if (message.equalsIgnoreCase(Protocol.getXML()) { //return ReadXML object out.writeObject(reader); //send the entire object // out.writeObject(reader.tokens); out.flush(); Thread.sleep(5000); System.out.println("XML Content Read and Sent via Sockets."); } else{ System.out.println("Expecting Message:"+Protocol.getXML()); } //if xml file is requested we send it. } catch (Exception ex) { Logger.getLogger(XMLReader.class.getName()).log(Level.SEVERE, null, ex); } } </pre>	

The petri net engine code model is a little bit more complicated the other two projects in terms that it must first connect to the XMLReader server socket to obtain the ReadXML data then listens for connection coming from the Execution client and reply to it with the Petrinet object, so this results in three sockets (client socket, server socket, new client socket)

Table 2: PetrinetEngine socket communication code segment

PetrinetEngine Class of PetrinetEngine Project has the following socket communication	
<pre>//run server and listen for incomming connections Socket xml_socket=new Socket(Protocol.hostname, Protocol.XMLPort); ObjectOutputStream out = new ObjectOutputStream (xml_socket.getOutputStream()); ObjectInputStream in = new ObjectInputStream(xml_socket.getInputStream()); out.writeUTF(Protocol.getXML); out.flush(); System.out.println("XML Request Sent..."); Object ob=in.readObject(); ReadXML xml=(ReadXML)ob; System.out.println("XML Request Received..."); //build Petrinet Petrinet net=new Petrinet(xml); net.build(); ServerSocket serverSocket = new ServerSocket(Protocol.EnginePort); System.out.println("Waiting for Executing Request..."); Socket netSocket = serverSocket.accept(); System.out.println("Execution Request Received..."); ObjectOutputStream net_out = new ObjectOutputStream (netSocket.getOutputStream()); ObjectInputStream net_in = new ObjectInputStream(netSocket.getInputStream()); String message=net_in.readUTF(); if(message.equalsIgnoreCase(Protocol.getNet)){ //return ReadXML object net_out.writeObject(net); //send the entire net net_out.flush(); }</pre>	

The final class is the simplest communication, as it only has a client socket connecting to the engine server socket and obtaining the ready to execute petri net, which is used to run the simulation which we assume will happen in the third project via simulate(net) method.

Table 3: NetExecutionResults socket communication code

NetExecutionResults class socket communication code segement	
<pre>// TODO code application logic here Socket net_socket=new Socket(Protocol.hostname, Protocol.EnginePort); ObjectOutputStream out = new ObjectOutputStream (net_socket.getOutputStream()); ObjectInputStream in = new ObjectInputStream(net_socket.getInputStream()); out.writeUTF(Protocol.getNet); out.flush(); System.out.println("Net Request Sent..."); Object ob=in.readObject(); Petrinet net=(Petrinet)ob; System.out.println("Net Request Received..."); simulate(net);</pre>	

Using Semaphore in the Place class

We have added a semaphore to the place class, which allows only for one access to one token in the place, either for adding one token or for removing one token.

By importing the Semaphore class from the `import java.util.concurrent.*;` package and creating it with 1 capacity `sem=new Semaphore(1);`

Now we control adding tokens to the place via add method that uses the same semaphore as the remove method and must first acquire the semaphore to add the token to the tokens list and release the semaphore after adding is complete.

The adding thread must use join in order to die after starting and adding of the token.

```
public void add(Token t){
    // tokens.add(t);
    addingThread=new Thread(){
        public void run(){
            try {
                sem.acquire();
                System.out.println("Adding Semaphore Lock Acquired");
                tokens.add(t);
                sem.release();
                System.out.println("Adding Semaphore Lock Released");
            } catch (InterruptedException ex) {
                Logger.getLogger(Place.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    };
    addingThread.start();
    try {
        addingThread.join();
    } catch (InterruptedException ex) {
        Logger.getLogger(Place.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Figure 3: add method with semaphore control and thread

The same method is used for the removal of tokens from the place.

We create a removal thread that must first Acquire the same semaphore and release it after removal happens and returns the token.

The token then is added to the next place using the Transition fire method, which is the location that the adding and removal is called from.

So we update the Transition method to call these two methods instead of dealing with the list elements directly.


```

public Token remove(int index){
    // Token t= tokens.remove(index);

    final Token t=tokens.get(index);
    try {
        removingThread=new Thread(){
            public void run(){
                try {
                    sem.acquire();
                    System.out.println("Removing Semaphore Lock Acquired");
                    tokens.remove(index);
                    sem.release();
                    System.out.println("Removing Semaphore Lock Released");
                } catch (InterruptedException ex) {
                    Logger.getLogger(Place.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        };
        removingThread.start();
        removingThread.join();
    } catch (Exception ex) {
        Logger.getLogger(Place.class.getName()).log(Level.SEVERE, null, ex);
    }
    return t;
}

```

Figure 4: remove token via semaphore usage and thread

Results

Execution Sequence

We must first run XMLReader Project, then Run PetriEngineer Project then NetExecutionProject.

Otherwise the projects will get stuck waiting indefinitely for the other project to run or the message to arrive, or get a concurrency error or sever socket error due to absence of server socket listening for incoming connections.

We didn't work must on error recovery, nor made the projects run indefinitely.

Each project only runs once and handles one file per its life cycle.

We will present the output of the three projects which are run over the following Petrinet (which is a small modified Petrinet over the existing one in the project report which had some repetitions and isolated places and don't represent a correct Petrinet)

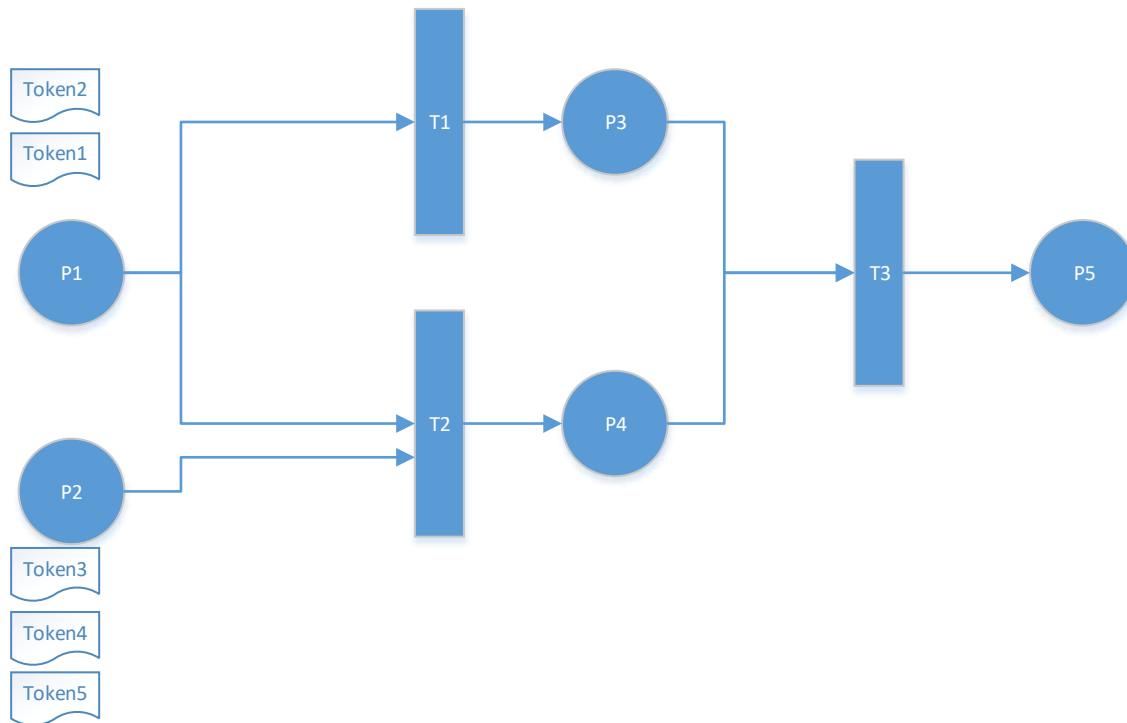


Figure 5: sample petrinet

The XML file of this petri net is the following

Table 4: sample XML file used in testing

Sample xml file
<pre> <?xml version="1.0" encoding="UTF-8"?> <Petrinet> <Token> <ID>1</ID> <Arrival> 5 </Arrival> <Burst> 100 </Burst> <Priority> 1 </Priority> <Deadline> 200 </Deadline> </Token> <Token> <ID>2</ID> <Arrival> 15 </Arrival> <Burst> 50 </Burst> <Priority> 2 </Priority> <Deadline> 150 </Deadline> </Token> <Token> <ID>3</ID> <Arrival> 12 </Arrival> <Burst> 40 </Burst> <Priority> 1 </Priority> </pre>

```
<Deadline> 70 </Deadline>
</Token>
<Token>
<ID>3</ID>
<Arrival> 12 </Arrival>
<Burst> 40 </Burst>
<Priority> 1 </Priority>
<Deadline> 70 </Deadline>
</Token>
<Token>
<ID>4</ID>
<Arrival> 13 </Arrival>
<Burst> 25 </Burst>
<Priority> 1 </Priority>
<Deadline> 70 </Deadline>
</Token>
<Token>
<ID>5</ID>
<Arrival> 20 </Arrival>
<Burst> 25 </Burst>
<Priority> 2 </Priority>
<Deadline> 70 </Deadline>
</Token>
<Place>
<ID> 1 </ID>
<Name> P1 </Name>
</Place>
<Place>
<ID> 2 </ID>
<Name> P2 </Name>
</Place>
<Place>
<ID> 3 </ID>
<Name> P3 </Name>
</Place>
<Place>
<ID> 4 </ID>
<Name> P4 </Name>
</Place>
<Place>
<ID> 5 </ID>
<Name> P5 </Name>
</Place>
<Marking>
<PID>1</PID>
<TokenID>1</TokenID>
</Marking>
<Marking>
```

```
<PID>1</PID>
<TokenID>2</TokenID>
</Marking>
<Marking>
<PID>2</PID>
<TokenID>3</TokenID>
</Marking>
<Marking>
<PID>2</PID>
<TokenID>4</TokenID>
</Marking>
<Marking>
<PID>2</PID>
<TokenID>5</TokenID>
</Marking>
<Transition>
<ID>1</ID>
<Name>T1</Name>
<Policy> FCFS </Policy>
</Transition>
<Transition>
<ID>2</ID>
<Name>T2</Name>
<Policy> SJF </Policy>
</Transition>
<Transition>
<ID>3</ID>
<Name>T3</Name>
<Policy> EDF </Policy>
</Transition>
<Topology>
<PID>1</PID>
<TID>1</TID>
<Direction>Input</Direction>
</Topology>
<Topology>
<PID>1</PID>
<TID>2</TID>
<Direction>Input</Direction>
</Topology>
<Topology>
<PID>2</PID>
<TID>2</TID>
<Direction>Input</Direction>
</Topology>
<Topology>
<PID>3</PID>
<TID>1</TID>
```

```

<Direction>output</Direction>
</Topology>
<Topology>
<PID>4</PID>
<TID>2</TID>
<Direction>output</Direction>
</Topology>
<Topology>
<PID>4</PID>
<TID>3</TID>
<Direction>input</Direction>
</Topology>
<Topology>
<PID>3</PID>
<TID>3</TID>
<Direction>input</Direction>
</Topology>
<Topology>
<PID>5</PID>
<TID>3</TID>
<Direction>output</Direction>
</Topology>
</Petrinet>

```

Now running first project will give us the following output console window:

Table 5: Project XMLReader initial output window test

Project XMLReader initial output window
<p>run:</p> <p>----- List of Tokens -----</p> <p>TokenID =1 Arrival Time =5 Burst =100 Deadline =1 Priority =200</p> <p>TokenID =2 Arrival Time =15 Burst =50 Deadline =2 Priority =150</p> <p>TokenID =3 Arrival Time =12 Burst =40 Deadline =1 Priority =70</p>

TokenID =3
Arrival Time =12
Burst =40
Deadline =1
Priority =70

TokenID =4
Arrival Time =13
Burst =25
Deadline =1
Priority =70

TokenID =5
Arrival Time =20
Burst =25
Deadline =2
Priority =70

----- List of Places -----

PlaceID=1 ,PlaceName=P1

PlaceID=2 ,PlaceName=P2

PlaceID=3 ,PlaceName=P3

PlaceID=4 ,PlaceName=P4

PlaceID=5 ,PlaceName=P5

----- List of Transition -----

TID=1 ,TName=T1 ,Policy=FCFS

TID=2 ,TName=T2 ,Policy=SJF

TID=3 ,TName=T3 ,Policy=EDF

----- List of Markings -----

placeID=1 ,tokenID=1

placeID=1 ,tokenID=2

placeID=2 ,tokenID=3

placeID=2 ,tokenID=4

placeID=2 ,tokenID=5

```

----- List of Topolgoies -----
placeID=1 ,transitionID=1 ,direction=Input

placeID=1 ,transitionID=2 ,direction=Input

placeID=2 ,transitionID=2 ,direction=Input

placeID=3 ,transitionID=1 ,direction=output

placeID=4 ,transitionID=2 ,direction=output

placeID=4 ,transitionID=3 ,direction=input

placeID=3 ,transitionID=3 ,direction=input

placeID=5 ,transitionID=3 ,direction=output

Waiting For Engine XML Request...

```

We can see that the output shows the content of the XML file as it was parsed and converted to relevant objects and before building the tree.

It also shows that it is waiting for Engine Request.

Now if we run the Engine Project we get the following Output at the Engine Console window.

Table 6:PetriNet Engine execution console output results test

PetriNet Engine execution console output, initial run
<p>run:</p> <p>XML Request Sent...</p> <p>XML Request Received...</p> <p>Linking Transition and Places...</p> <p>TID=1 ,TName=T1 ,Policy=FCFS</p> <p>Input Places For Transition ID:1</p> <p>PlaceID=1 ,PlaceName=P1</p> <p>Output Places Transition ID:1</p> <p>PlaceID=3 ,PlaceName=P3</p> <p>TID=2 ,TName=T2 ,Policy=SJF</p> <p>Input Places For Transition ID:2</p> <p>PlaceID=1 ,PlaceName=P1</p> <p>PlaceID=2 ,PlaceName=P2</p> <p>Output Places Transition ID:2</p> <p>PlaceID=4 ,PlaceName=P4</p>

TID=3 ,TName=T3 ,Policy=EDF
Input Places For Transition ID:3
PlaceID=4 ,PlaceName=P4
PlaceID=3 ,PlaceName=P3
Output Places Transition ID:3
PlaceID=5 ,PlaceName=P5
Transition Input and Outputs were linked

Linking Places to Tokens using Markings

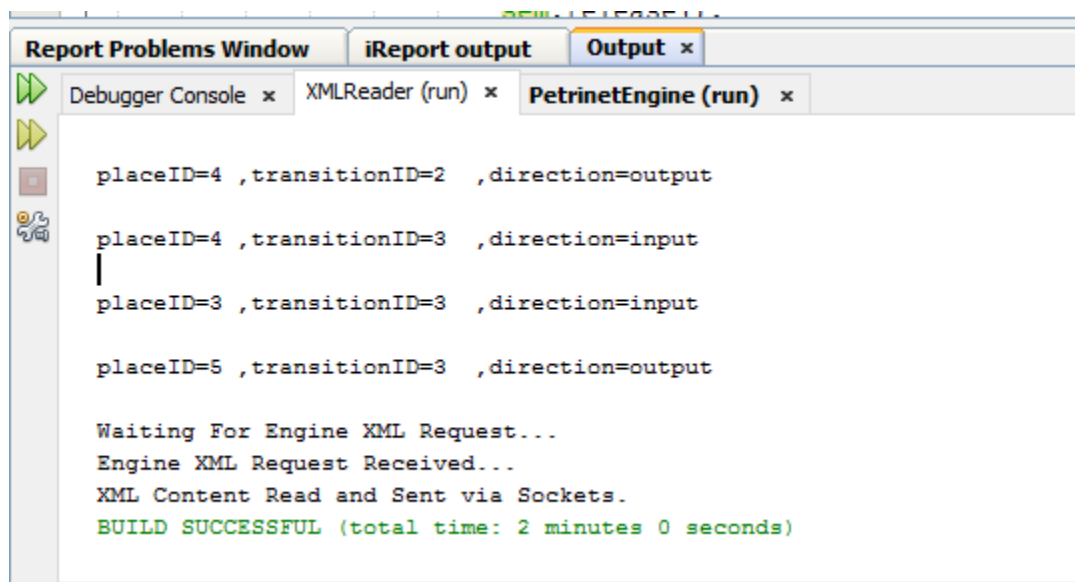
Place with ID:1 was linked with Token ID:1 using Marking
Place with ID:1 was linked with Token ID:2 using Marking
Place with ID:2 was linked with Token ID:3 using Marking
Place with ID:2 was linked with Token ID:3 using Marking
Place with ID:2 was linked with Token ID:4 using Marking
Place with ID:2 was linked with Token ID:5 using Marking
Place-Token Linkage Complete.

Waiting for Executing Request...

We can see that the engine client has connected to the XMLReader server and obtained the ReadXML objects and built the Petrinet and connected places to transitions and built the right tree and produced the linkage results.

It also shows that it is now waiting for Execution Sever connection (in the last line)

If we go back to the first project console window we can see that it shows the XML request received and file sent notification and exit.



The screenshot shows the 'iReport output' window with the 'Output' tab selected. The output text is as follows:

```
placeID=4 ,transitionID=2 ,direction=output  
placeID=4 ,transitionID=3 ,direction=input  
placeID=3 ,transitionID=3 ,direction=input  
placeID=5 ,transitionID=3 ,direction=output  
  
Waiting For Engine XML Request...  
Engine XML Request Received...  
XML Content Read and Sent via Sockets.  
BUILD SUCCESSFUL (total time: 2 minutes 0 seconds)
```

Figure 6:XMLReader project execution completion

Now if we run the third project we get the following output

Table 7: NetExecutionResults console output and results

NetExecutionResults console output
<p>run:</p> <p>Net Request Sent...</p> <p>Net Request Received...</p> <p>Transition with ID:1 using scheduling[FCFS]is enabled and was fired:</p> <p>Removing Semaphore Lock Acquired</p> <p>Removing Semaphore Lock Released</p> <p>Token wiht ID:1 was removed from Place with ID:1</p> <p>Token was added to Transition output place with ID:3</p> <p>Adding Semaphore Lock Acquired</p> <p>Adding Semaphore Lock Released</p> <p>Transition with ID:2 using scheduling[SJF]is enabled and was fired:</p> <p>Removing Semaphore Lock Acquired</p> <p>Removing Semaphore Lock Released</p> <p>Token wiht ID:2 was removed from Place with ID:1</p> <p>Token was added to Transition output place with ID:4</p> <p>Adding Semaphore Lock Acquired</p> <p>Adding Semaphore Lock Released</p> <p>Removing Semaphore Lock Acquired</p> <p>Removing Semaphore Lock Released</p> <p>Token wiht ID:5 was removed from Place with ID:2</p> <p>Token was added to Transition output place with ID:4</p> <p>Adding Semaphore Lock Acquired</p> <p>Adding Semaphore Lock Released</p> <p>Transition with ID:3 using scheduling[EDF]is enabled and was fired:</p> <p>Removing Semaphore Lock Acquired</p> <p>Removing Semaphore Lock Released</p> <p>Token wiht ID:5 was removed from Place with ID:4</p> <p>Token was added to Transition output place with ID:5</p> <p>Adding Semaphore Lock Acquired</p> <p>Adding Semaphore Lock Released</p> <p>Removing Semaphore Lock Acquired</p> <p>Removing Semaphore Lock Released</p> <p>Token wiht ID:1 was removed from Place with ID:3</p> <p>Token was added to Transition output place with ID:5</p> <p>Adding Semaphore Lock Acquired</p> <p>Adding Semaphore Lock Released</p> <p>BUILD SUCCESSFUL (total time: 0 seconds)</p>

The final output shows the firing of transitions and movement of tokens, as well as semaphore acquirement and releases and exit of the application.

If we go back to the second project execution window we see that it received the third server message and it sent file after building the tree and existed.

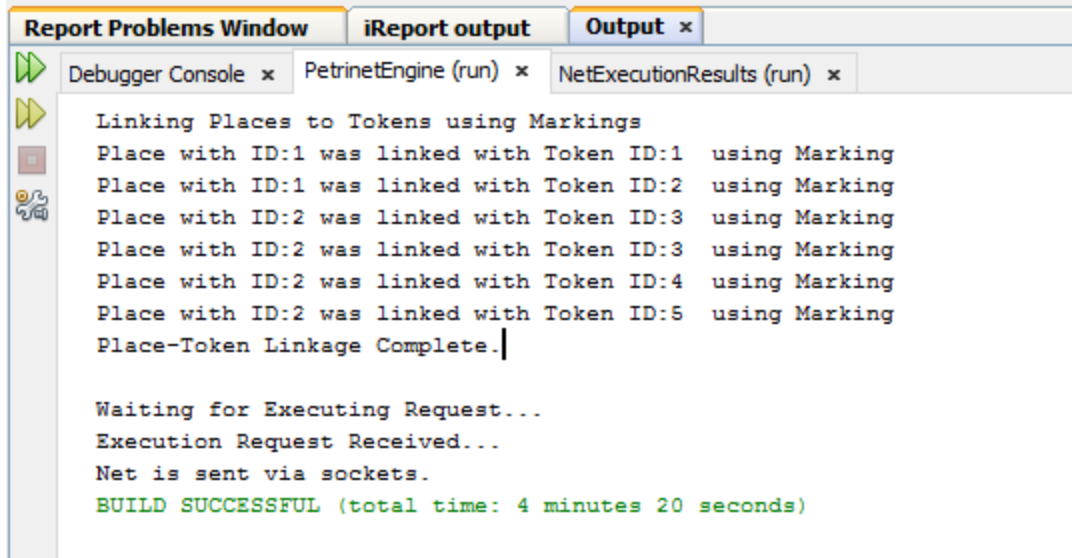


Figure 7: Project two final output

Conclusion

We have built a three projects, to simulate the petrinet execution where first program reads the XML files and builds initial objects, the second program reads these objects and builds the Petri net tree, the third programs reads the petrinet tree objects from the second program and executes it.

The first and second programs have server socket while the third program only contains a client socket. Each program does a different job on its own.

We used a semaphore in the place which controls adding and removal of tokens and allows only one token to be accessed at any given time by the transitions which in turn uses the scheduling algorithm defined in the policy of the transition.

Output of each program was shown in this report which was tested over a sample XML file.