



Universitetet
i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study program/specialization:

Spring semester, 2010

Master in Computer Science

Open

Author: Jing Kou

.....
(Author's signature)

Instructor: Prof. Dr. Chunming Rong; Eng^oTomasz Wlodarczyk, UiS

Supervisor: Prof. Dr. Chunming Rong, UiS

Title of Master's Thesis: Reasoning Techniques Used For Data Processing

Norwegian title:

ECTS: 30

Subject headings:

Reasoning Techniques; Data Processing;
Model; Jess; Petri Net; R; Rserve; Jade; Pig

Pages: 57

+ attachments/other: 15

Stavanger, June 15th 2010

Reasoning Techniques Used For Data Processing

Based On JESS, Petri Net and Rserve

Jing Kou

University of Stavanger

June 15th 2010

Acknowledgements

The author would like to express her special gratitude to:

Prof. Dr. Chunming Rong from University of Stavanger;

Eng °Tomasz Włodarczyk from University of Stavanger.

Abstract

In the oil industry, it is very important to know the current status of drilling processes which can be obtained by analyzing the data from sensors on the drilling engines. The data which oil companies get is complicated, so, in order to analyse the data, it has to be processed first.

There are several methods of intelligent data analysis such as JESS, Petri Nets, R functions, Bayesian Networks and so on. Which of the reasoning techniques can be used to process the data and how to use it in a system are left for users to research and develop. To resolve the problem, upon the study of many other data-processing methods, this paper proposes several novel models for data processing step by step. To validate the effectiveness and the feasibility of the models, the author designs and implements several related systems to interface the reasoning techniques into the systems. The functions of every module in the system and the interrelations between them are achieved in the form of class and the core data structure is described in detail as well.

In chapter 2, the author first analyses the characteristics of reasoning technologies for identifying use. In chapter 3, the author chooses JESS as the reasoning technique to process and monitor the data. Based on the monitoring results from chapter 3 and Petri Net technique, chapter 4 develops another data processing system called 'SUP system' and also analyses the performance of the system. The first 2 models are only used for single server, but when there is a lot of data need to be processed, multi servers are required. In order to solve this problem, the author also does some research on Rserve in a distributed environment in chapter 5.

The results prove that some models and systems are well developed and the reasoning techniques are well used in the systems, but some other reasoning techniques have limitations in the related models due to the reason of researching time and the author's knowledge.

Table of Contents

1	INTRODUCTION.....	1
1.1	Background	1
1.2	Thesis Overview.....	1
1.2.1	What this Application can do	1
1.2.2	Why this Application is useful	2
1.3	Chapter Settings	2
2	Theory of the Reasoning Techniques	3
2.1	JESS	3
2.1.1	Introduction	3
2.1.2	Example.....	3
2.2	Petri Net.....	4
2.2.1	Introduction	4
2.2.2	Example.....	5
2.3	R and Rserve	6
2.3.1	R	6
2.3.2	Rserve.....	6
2.4	Bayesian Networks.....	8
2.4.1	Theory	8
2.4.2	Bayesian Network	9
2.5	Conclusion of the Chapter.....	9
3	Data Processing Model Based On JESS	10
3.1	Model	10
3.2	Developing Environment	11
3.2.1	Why use Multi-agent System, JADE and JESS	11
3.2.2	Multi-agent System	12
3.2.3	Developing Multi-agent System with JADE.....	12
3.3	Implementation of the Model.....	13
3.3.1	Step 1: Embed JADE into Eclipse.....	13
3.3.2	Step 2: Create Agents and Define Tasks	13
3.3.3	Step 3: Agents Communication.....	15
3.3.4	Step 4: Embed JESS into Eclipse.....	18
3.3.5	Step 5: Write JESS Files	19
3.3.6	Step 6: Integrate JADE Agents with JESS.....	19
3.3.7	Step 7: Test and Result.....	23
3.4	Conclusion of the Chapter.....	23
4	Data Processing Model Based On Petri Net.....	24
4.1	Model	24
4.2	Developing Environment	25
4.2.1	GPenSIM.....	25
4.2.2	PDF, TDF and MSF	26
4.2.3	Global Info	28
4.2.4	Colored GPenSIM	29
4.3	Implementation of the Model.....	30
4.3.1	Petri Net Gragh of the System.....	30
4.3.2	Module of the User Side.....	30
4.3.3	Module of Login Request.....	31
4.3.4	Module of SUP Request.....	31
4.3.5	Module of the Answer Side.....	32

4.3.6 Performance of the System	33
4.4 Conclusion of the Chapter	35
5 Data Processing Model Based On Rserve	36
5.1 Model	36
5.2 Developing Environment	37
5.2.1 Pig.....	37
5.2.2 Pig Latin	37
5.2.3 Pig Code Written In Java	38
5.2.4 Pig Installation.....	38
5.2.5 Pig UDF.....	39
5.3 Implementation of the Model	40
5.3.1 Step 1: install Rserve	40
5.3.2 Step 2: install Pig.....	42
5.3.3 Step 3: embed Rserve into Pig	43
5.3.4 Step 3: Pig UDF with Rserve	45
5.4 Conclusion of the Chapter	55
6 Conclusion and Future Work	56
6.1 Conclusion.....	56
6.2 Future Work	56
7 References	57
8 Appendix	59
8.1 Appendix A- Example of Farmer's Dilemma Problem	59
8.2 Appendix B- Example of Norwegian Traffic Lights.....	64

1 INTRODUCTION

This chapter presents a short overview of this work. It starts with the description of the background. Then a general overview of the whole project is given. This chapter finishes with the outlines of the report.

1.1 Background

In the oil industry, it is very important to know the current status of drilling processes which can be obtained by analysing the data from sensors on the drilling engines. Drilling is usually operated by the service companies such as Schlumberger, BHI and Halliburton. They collect data from platforms and make them available on their servers. For operating companies such as StatoilHydro, Shell and ConocoPhillips, they need to know whether everything is going ok or not by analysing the data. The data that the oil companies get is complicated, so, in order to analyse the data, it has to be processed first. The intelligence part of how to process the data with reasoning techniques is left for user to develop.

There are several methods of intelligent data analysis such as JESS, Petri Net, R functions, Bayesian Networks and so on. However, which of those methods can be used to process data in oil industry and how to interface the available reasoning techniques in related systems are left for users to research and develop.

1.2 Thesis Overview

1.2.1 What this Application can do

The main task of the thesis is to research on the reasoning techniques that can be used to process data. Based on the research, the authour develpes 3 models to introduce how to use these reasoning techniques in related systems. The paper is divided into the following steps further more.

- The data processing system which integrates JADE and JESS seamlessly is developed and implemented.
- The data processing system which is based on Petri Net technique and the JESS monitoring results is developed. The performance of the system is also analysed.

- In order to process data with multi servers in a distributed environment, a research on the interface between Rserve and Pig is made in the paper.

1.2.2 Why this Application is useful

For oil companies, it is very important to process and analyze the data, because on one hand operators want to know whether everything is going ok or not; on the other hand it may direct operators to find new resources. This project gives out several models to show how to process data with reasoning techniques.

1.3 Chapter Settings

As showing above, chapter 1 introduces the background of this work, and gives an overview of it. The rest of the thesis is organized as follows. Chapter 2 describes the main reasoning techniques that we use in the thesis. Chapter 3 gives out how to interface JESS with JADE to process and monitor data. Chapter 4 proposes a system based on Petri Net technique and the monitoring results of chapter 3; it also analyzes the performance of the system. Chapter 5 uses R as the reasoning technique in a distributed environment. Finally, chapter 6 presents our conclusion and future work.

2 Theory of the Reasoning Techniques

In this chapter, it presents several basic methods of intelligent data analysis such as JESS, Petri Net, R, and Bayesian networks.

2.1 JESS

2.1.1 Introduction

JESS^[1] is short for Java Expert System Shell. It was originally inspired by the CLIPS^[2] expert system shell, but now, it has grown into a distinct Java-influenced and rule-based environment. It is a rule-based programming environment, a rule engine and scripting environment. It provides a tool to develop systems with intelligent reasoning abilities. It has a fast and efficient algorithm which is called Rete algorithm. Rete algorithm can build a network of pattern-matching nodes to solve problems with rules. First, it will do the pattern matches, then use a set of memories to store the information about the results of the matches, and then give out the available matches.

During all the available rule engines, JESS is very small, light, and is also one of the fastest engines. JESS is written in Java language which is easily to intergrate with other Java based techniques such as JADE etc. Using JESS, the user can design rules according to using knowledge, then build Java software to reason the rules.

2.1.2 Example

Farmer's Dilemma Problem^[1]

A simple example is given as below. The point is to get the farmer, the fox, the cabbage and the goat across a stream. But the boat only holds 2 items. If left alone with the goat, the fox will eat it. If left alone with the cabbage, the goat will eat it.

The JESS codes are presented in 'Appendix A- Example of Farmer's Dilemma Problem'. The solutions are shown in Figure 1.

Solution found:

```
Farmer moves with goat to shore-2.  
Farmer moves alone to shore-1.  
Farmer moves with fox to shore-2.  
Farmer moves with goat to shore-1.  
Farmer moves with cabbage to shore-2.  
Farmer moves alone to shore-1.  
Farmer moves with goat to shore-2.
```

Solution found:

```
Farmer moves with goat to shore-2.  
Farmer moves alone to shore-1.  
Farmer moves with cabbage to shore-2.  
Farmer moves with goat to shore-1.  
Farmer moves with fox to shore-2.  
Farmer moves alone to shore-1.  
Farmer moves with goat to shore-2.
```

Figure 1: Result of Farmer's Dilemma Problem

2.2 Petri Net

2.2.1 Introduction

Petri net^[3] is one of the several mathematical modeling languages which is used for the description of discrete distributed systems.

A Petri net is actually a directed graph, it includes transitions, places, and directed arcs. Transitions which are signified by bars are discrete events that may occur; Places which are signified by circles are conditions; Directed arcs which are signified by arrows describe the relationship between the transitions - which places are pre- and/or post conditions for which transitions.

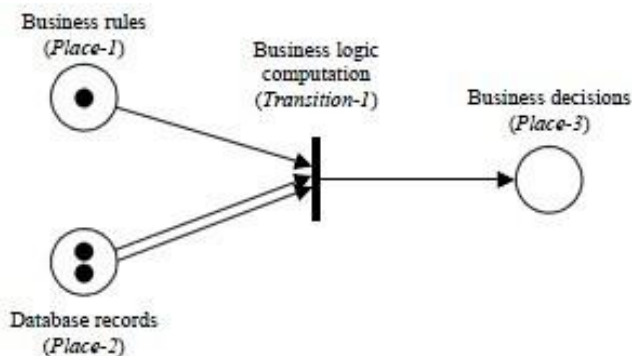


Figure 2: Petri Net Model for Business Logic Computation

As can be seen from Figure 2^[4], this Petri net shows a model for a simple business logic computation. This computation model has 2 inputs and 1 output. The business rules and database records are inputs; The business decisions are output. These 3 which are drawn as circles are called places. The black spots inside a place are called tokens. A place usually holds a number of parts. The number of parts inside a place is indicated by the tokens. The computations which are drawn as vertical short bars are called transitions. The arc which connect place and transition is a path for a discrete part to flow. A place can have several arcs. In this Petri net graph, there are 3 places, 3 tokens, 1 transition and 4 arcs altogether.

2.2.2 Example

Norwegian Traffic Lights^[5]

An example of a Petri net model for Norwegian traffic lights is show as Figure 3. It shows how the Petri networks. The codes can be seen in ‘Appendix B- Example of Norwegian Traffic Lights Petri Net Model’.

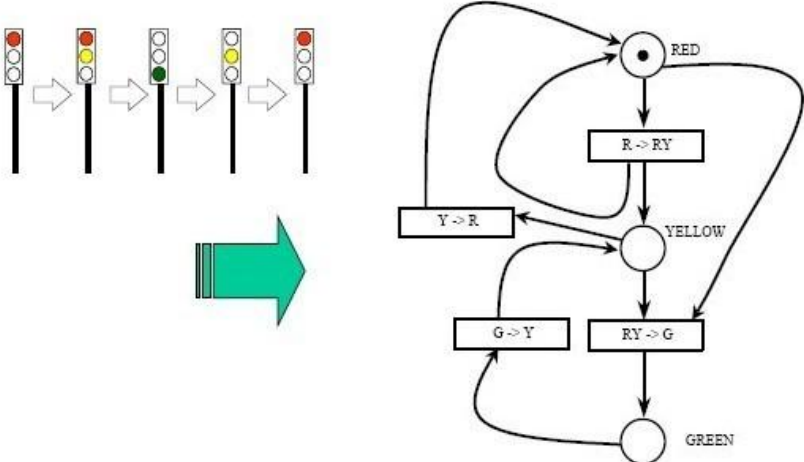


Figure 3: Petri Net Model for Norwegian Traffic Light

2.3 R and Rserve

2.3.1 R

R^[6] is a language and environment which is used for statistical computing and graphics. It provides modeling, statistical tests, time-series analysis, classification, clustering and so on; R also provides graphical techniques and is highly extensible.

2.3.2 Rserve

Rserve^[7] is a TCP/IP server. When the user wants to use the functions of R, he doesn't need to initialize R or link against R library first. Once a Rserve connection is created, it will have a separate working directory and name space. Client-side implementations are available for most of the popular programming languages such as C, C++ and Java. Most of the data types of R can be transformed into C or Java data types. Rserve supports encrypted user/password authentication. Rserve also supports remote connection, authentication and file transfer. Typical use is to integrate R backend for computation of statistical models, plots etc.

Rserve has the following features:

- Fast – if the user wants to use Rserve, he doesn't need to initialize R first.
- Client independence - the client is independent because the client is not linked to R.
- Persistent - Once a connection is created, it will have its own working directory and namespace. If the user creates an object, it will be persistent until the connection is finished.
- Security - Rserve supports encrypted user/password authentication, so it provides basic security. The user can also configure Rserve to let it accept only local connections to provide some security.
- Configurable - There is a file for configuration. In this file, the user can control settings, the user can also enable/disable features, for example: authorization, remote access or file transfer and so on.
- Binary transport - R objects are sent as binary data (not text) in the transport protocol.
- Automatic type conversion - Most of the data types of R can be transformed into C/Java data types. R also has some new data types such as RBool, RList and so on, Java client also provides classes for these new data types.

- File transfer - The user can transfer files between the client side and the server side. So, in this way, the user can use Rserve as a remote server.

Rserve is actually a server. The clients send requests to the server, and Rserve listens to the incoming connections, once the connection is installed, it will respond to the requests. For some reasons, some users still use the old editions of Rserve. In order to use Rserve, the user needs to install R-1.5.0 or a higher edition.

Rserve itself is a server. The server can only be useful when there are clients, therefore three client frameworks are developed:

- REngine Java client - JRclient which is located in src/client/java-old is an older Java API that was used in Rserve 0.4 and earlier. REngine Java client which is located in src/client/java-new is a full client that allows any Java application (JDK 1.4 or higher) to access an Rserve. Compared with previous Java clients, this new client API is more flexible, with better design, has better exception handling and is aimed to support both JRI and Rserve transparently. It is a full client suite entirely written in java. It allows any Java application to access an Rserve. Most of the data types of R can be transformed into Java data types, such as int, double, arrays, String or Vector. R also has some new data types such as RBool, RList etc, and Java client also provides classes for these new data types.
- C++ client - C++ client is located in src/client/cxx directory in the Rserve source package. It also provides basic interface to Rserve from any C++ program.
- R client - R client is a small client directly in the Rserve package.
- Rcli.c - Rcli.c is a lightweight client that demonstrates how to connect to Rserve from C language. It is available in early Rserve versions, now replaced by the C++ client.

Rserve is a TCP/IP server. It is basically possible for the user to write any language clients as long as these languages support TCP/IP sockets. In Rserve, the client side and server side are separated. When the user directly links against R library, it may result in multi-threading problems, the separation of client/server side can prevent this problem from happening.

Rserve is actually provided as a regular R package and can be installed like this. The user can start Rserve executable Windows or typing R CMD Rserve on the command line to start the Rserve, however, the user cannot start it by the library command. Once the Rserve is started,

it runs in local mode by default and it doesn't have enforced authentication. The applications can use their services from the server after it starts. The applications can be written in Java. When using other Rserve clients, the principles are identical, so, using Java as the starting point poses no limitation. A simple example is shown as below:

```
RConnection c = new RConnection();
REXP x = c.eval("R.version.string");
System.out.println(x.asString());
```

JRI can be used to access R from Java in one application without the need for the client/server concept. JRI uses JNI to link R directly into Java. The following Java code illustrates the easy integration of Rserve:

```
RConnection c = new RConnection();
... ..
... ..
```

Once the Rconnection is installed, the user doesn't need to create it more than once.

2.4 Bayesian Networks

2.4.1 Theory

The Bayesian theory^[8] presents two important concepts: the Bayesian probabilities and the theorem which is also known as rule. A probability can be thought as a quantitative measure of the strength of one's knowledge or of one's beliefs. This way, we can assess them using experts' knowledge and without having historical data. With this concept it is possible to deal with subjective beliefs and use them into a mathematical model. Other idea subjacent to Bayesian is the conditionality. Instead of a classical approach, Bayesian uses the notion of a probability of an event as a consequence of other events' probabilities.

The Bayesian theorem is:

$$P(A/B) = \frac{P(AB)}{P(B)} = \frac{P(B/A)P(A)}{P(B)}$$

2.4.2 Bayesian Network

A Bayesian network is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Formally, Bayesian networks are directed acyclic graphs whose nodes represent random variables in the Bayesian sense: they may be observable quantities, latent variables, unknown parameters or hypotheses. Edges represent conditional dependencies; nodes which are not connected represent variables which are conditionally independent of each other. Each node is associated with a probability function that takes as input a particular set of values for the node's parent variables and gives the probability of the variable represented by the node. For example, if the parents are m Boolean variables then the probability function could be represented by a table of 2^m entries, one entry for each of the 2^m possible combinations of its parents being true or false.

Efficient algorithms exist that perform inference and learning in Bayesian networks. Bayesian networks that model sequences of variables are called dynamic Bayesian networks. Generalizations of Bayesian networks that can represent and solve decision problems under uncertainty are called influence diagrams.

2.5 Conclusion of the Chapter

This chapter simply presents several basic methods of intelligent data analysis such as JESS, Petri Net, R, and Bayesian Networks. However, how can we use these reasoning techniques to process data? The following chapters will give out some models and show how to use the reasoning techniques in the models.

3 Data Processing Model Based On JESS

In the oil industry, it is very important to know the current status of drilling processes which can be obtained by analyzing the data from sensors on the drilling engines. People need to know whether everything is going ok or not by analysing the data. However, the data need to be processed first in order to be analyzed.

This chapter chooses JESS as the reasoning technique for data processing. The author first gives out a model to show how to use JESS to process the data together with JADE^[9]. Based on the model, the author designs a system and implements it. The experimental results prove that JESS can be used very well to process the data together with JADE.

3.1 Model

For operating companies such as StatoilHydro, Shell and ConocoPhillips, they need to know whether everything is going ok or not by analysing the data. If the data is abnormal, an alarm should be given out.

To resolve the problem of real-time processing requirements, this chapter proposes a novel model for data processing based on JESS technology, named RTMD (Real-Time Monitoring of Data) model. This model is a development of a former project called ‘Data Querying and Transformation Application’^[10] developed by another master student Baodong Jia. He creates a ‘Query Client’ to fetch data from the servers of the service providers and then chooses semantic technologies such as XML^[11], RDF^[12], and XSLT^[13] to process the data. Based on the ‘Query Client’, the next step is to monitor the data and give out alarms when the data is abnormal. The monitoring part is based on JADE and JESS technologies. As can be seen in Figure 4^[14], JADE is used as the agent development environment here. Based on this environment, three agents are created. They are Agent 1 (Feeder Agent), Agent 2 (Reasoner Agent) and Agent 3 (Alarm Agent). Agent 1 (Feeder Agent) fetches data from the ‘Query Client’. Then Agent 1 (Feeder Agent) sends the data to Agent 2 (Reasoner Agent) with data rate e.g. 1 data point per second. Agent 2 (Reasoner Agent) interfaces with JESS to see if rule/rules are matched. If so, Agent 2 (Reasoner Agent) sends message to Agent 3 (Alarm Agent) to give out the corresponding alarms.

To validate the effectiveness and the feasibility of this novel model, a system is designed and implemented, which is called RTMD system. Experimental results show that RTMD system can effectively deal with data from sensors and give out alarms when the data is abnormal.

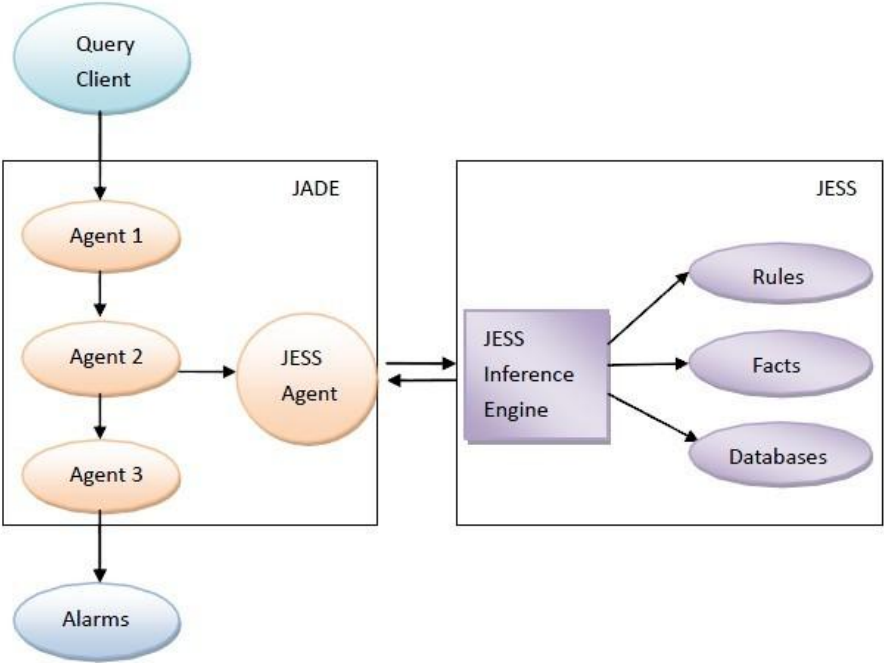


Figure 4: The Model of Alarm System

3.2 Developing Environment

3.2.1 Why use Multi-agent System, JADE and JESS

We use multi-agent system^[15] in this project because it is easy to use agents to fetch data from ‘Query Client’.

We use JADE on agent technology because:

- It has all the agent features that we need.
- Communication between agents is easy to complement.
- It is efficient and tolerant of faulty programming.

Although JADE provides all the mandatory components for the development of autonomous agents, currently JADE alone does not endow agents with specific capabilities beyond those needed for communication and interaction. On the negative side, JADE may be disappointing to AI people because it lacks mechanisms for ‘intelligence’, planning or reasoning.

However, if the system wants to give out alarms, it has to have the reasoning ability to decide when to give out alarms and what kind of alarm should be given out. So, a reasoning tool is needed. We choose JESS as the reasoning technique here.

3.2.2 Multi-agent System

An agent is a proactive, dynamic, autonomous and goal-oriented software entity. The user needs to create an agent in order to achieve a goal. Since the agent has artificial intelligence techniques, it knows how to choose the best actions to achieve the goal. But a single agent can not solve complicated problems individually, so, a multi-agent system is needed.

A multi-agent system is a group of agents. These agents communicate with each other to coordinate their activities to solve complicated problems that cannot be solved by single agent.

3.2.3 Developing Multi-agent System with JADE

JADE Overview

JADE is short for Java Agent Development Environment. It is a middleware that facilitates the development of multi-agent systems^[16]. It is a robust and efficient environment for distributed agent systems.

JADE includes the following:

- A runtime environment where JADE agents can ‘live’ and that must be active on a given host before one or more agents can be executed on that host.
- A library of classes that programmers can use to develop their agents.
- A suite of graphical tools that allows administrating and monitoring the activity of running agents.

Creating Agents

In order to create an agent, the user need to define a class that extends the JADE.core.Agent class and overriding the default implementation of some methods. The methods which include SetUp and TakeDown() are automatically invoked by the platform during the agent lifecycle.

Each agent instance is identified by an ‘agent identifier’ in order to be consistent with the FIPA specifications. The agent identifier is represented as an instance of the JADE.core.AID class. The getAID() method of the Agent class allows retrieval of the local agent identifier.

Defining Agent Tasks

What an agent has to do is typically written in ‘behaviours’. A behavior includes an actual job that an agent will carry out. A behavior is implemented as an object of a class that extends JADE.core.behaviours.Behaviour.

Each such behavior class has to implement two abstract methods. First, it is the action() method which defines the operations to be performed when the behaviours is in execution; Second, it is the done() method which returns a Boolean value to indicate whether or not a behavior has completed and removed from the pool of behaviours. To make an agent execute the tasks represented by a behavior object, the behavior must be added to the agent.

Agents Communication

The JADE communication paradigm is based on asynchronous message passing. A message in JADE is implemented as an object of the JADE.lang.acl.ACLMessage object and then calling the send() method of the Agent class.

3.3 Implementation of the Model

3.3.1 Step 1: Embed JADE into Eclipse

We use the Eclipse platform as our programming environment, which is an open source Integrated Development Environment that provides support for language Java. The Eclipse environment comes with a plugin^[17] to integrate JADE within Eclipse.

3.3.2 Step 2: Create Agents and Define Tasks

Agents can be created according to the thread path^[18] as shown in Figure 5.

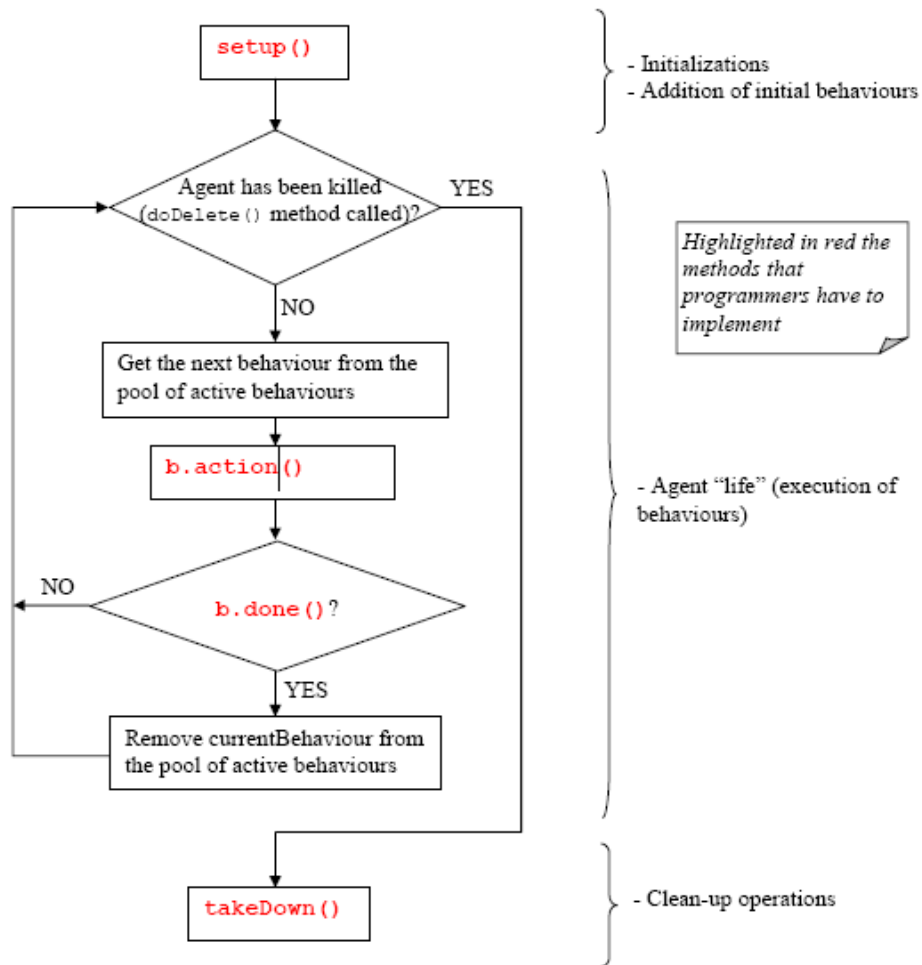


Figure 5: Agent Thread Path of Execution

The core code of creating an agent is as following:

```

public class FeederAgent extends Agent{
    public void setup(){
        SendTestdataBehavior t = new SendTestdataBehavior();
        addBehaviour(t);
        ...
        ...
    }
    public class SendTestdataBehavior extends Behaviour{
        boolean finished = false;
        public void action()
        {
            ...
            ...
            finished=true;
        }
    }
}
  
```

```

    }
    public boolean done(){
        return finished;
    }
}
...
...
}

```

Three kinds of agents which execute different tasks are created in this project, as can be seen in Figure 6. They are Agent 1 (Feeder Agent), Agent 2 (Reasoner Agent) and Agent 3 (Alarm Agent).

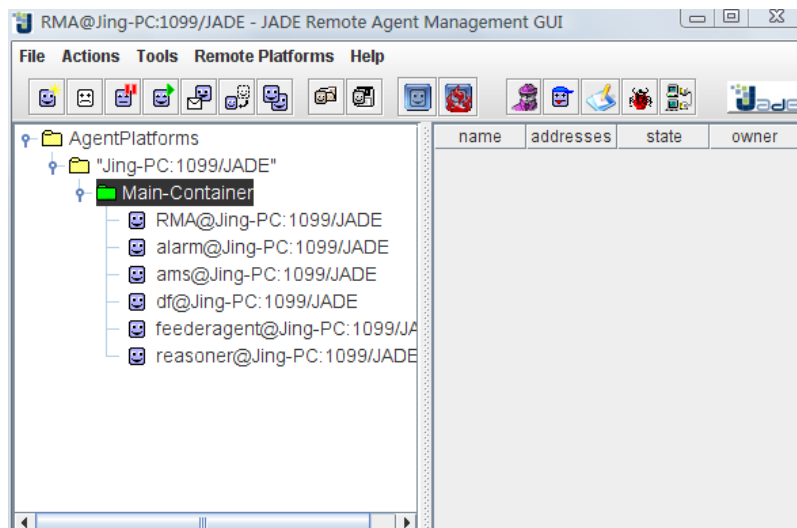


Figure 6: JADE Agents

3.3.3 Step 3: Agents Communication

Agent 1 (Feeder Agent) fetches data from the 'Query Client'. Then, Agent 1 (Feeder Agent) sends data to Agent 2 (Reasoner Agent). Agent 2 (Reasoner Agent) process the data and decide whether an alarm should be given out or not. If so, Agent 2 (Reasoner Agent) sends message to Agent 3 (Alarm Agent) to give out corresponding alarms.

The core code to send a message to an agent is as following:

```

public class SendTestdataBehavior extends Behaviour{
    boolean finished = false;
    public void action()

```

```

{
    AID receiverID = new AID("reasoner", AID.ISLOCALNAME);
    ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
    msg.setSender(getAID());
    msg.addReceiver(receiverID);
    String datastream;
    for (int i=1;i<=500;i++)
    {
        datastream = ""+i;
        msg.setContent(datastream);
        send(msg);
        System.out.println("sendAgent sent the message:"+i);
    }
    finished=true;
}
public boolean done(){
    return finished;
}
}

```

The core code that an agent receives a message is as following:

```

public class RangeBehavior extends Behaviour
{
    boolean finished = false;
    public void action()
    {
        ACLMessage msgReceive = receive();
        if(msgReceive != null)
        {
            int j=Integer.parseInt(msgReceive.getContent());
            System.out.println("reasonerAgent received message: "+ j);
            if(j > 20)
            {
                System.out.println("reasonerAgent received the message: "+ j);
                //send message to the alarm agent if the data > 20
                AID receiverID = new AID("alarm", AID.ISLOCALNAME);
                ACLMessage msgSend = new
ACLMessage(ACLMessage.INFORM);
                msgSend.addReceiver(receiverID);

```

```

        msgSend.setContent("illegal data: "+ j);
        send(msgSend);
    }
    else
    {
        System.out.println("reasonerAgent received the message:"+ j);
    }
}
else
{
    block();
}
}
public boolean done()
{
    return finished;
}
}

```

The result of agents communication is shown as Figure 7. sendAgent sends data to reasonerAgent, then reasonerAgent processes the data, if the data is more than 20, reasonerAgent sends a message to alarmAgent, then alarmAgent gives out alarms.

```

sendAgent sent the message: 19
reasonerAgent received the message: 19
sendAgent sent the message: 20
reasonerAgent received the message: 20
sendAgent sent the message: 21
reasonerAgent received the message: 21
alarmAgent sent the message: illegal data 21
sendAgent sent the message: 22
reasonerAgent received the message: 22
alarmAgent sent the message: illegal data 22

```

Figure 7: Agents Communication

In order to show how agents can communicate with each other clearly, we only use a simple rule (when the data is more than 20, an alarm will be given out) here. However, there are actually many complicated rules in oil industry, and it is impossible to use reasonerAgent to process the data when the rules are very complicated, so a reasoning tool is needed here. The

following steps will show how to use JESS as the reasoning tool and how to integrate it with JADE.

3.3.4 Step 4: Embed JESS into Eclipse

JADE has already been embedded into Eclipse as shown in step 1. In order to use JESS as the reasoning tool, we have to embed JESS into Eclipse as well. The steps of embedding JESS into Eclipse are as following:

Installation:

1. Exit Eclipse.
2. Open the 'Eclipse' file which includes five zip files: gov.sandia.jess.debug_7.1.0.zip, gov.sandia.jess.editor_7.1.0.zip, gov.sandia.jess.feature_7.1.0.zip, gov.sandia.jess_7.1.0.zip, gov.sandia.jess.reteview_7.1.0.zip.
3. Extract all these zip files to the current folder, and then you can get two files: plugins which includes gov.sandia.jess.debug_7.1.0, gov.sandia.jess.editor_7.1.0, gov.sandia.jess_7.1.0, gov.sandia.jess.reteview_7.1.0 and features which includes gov.sandia.jess.feature_7.1.0.
4. There are also two files called plugins and features in the Eclipse installation folder. Copy the content of them from JESS files to Eclipse files.

Identification of the Installation

1. Open Eclipse, then click 'help', then choose 'about Eclipse SDK', a JESS button is supposed to be found here.
2. Click 'Plug-in Details', 3/4 JESS plugins are supposed to be found here.
3. JESS is installed in Eclipse successfully, as can be shown in Figure 8.

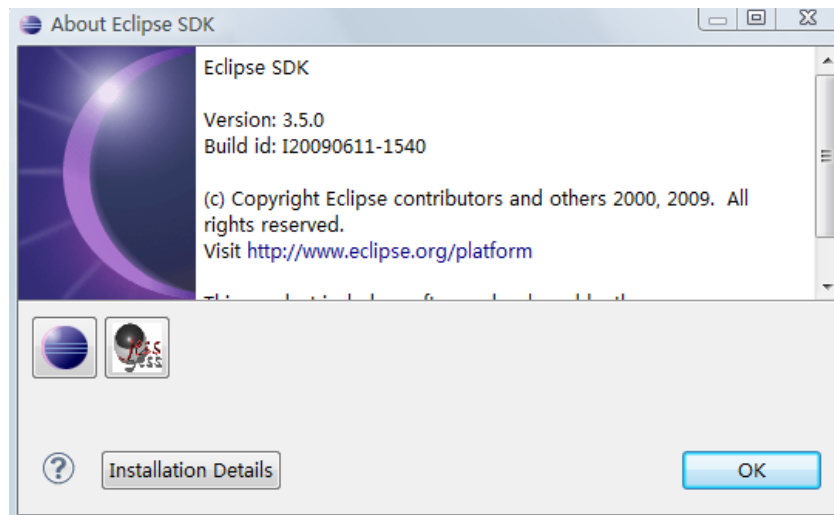


Figure 8: Embed JESS into Eclipse

3.3.5 Step 5: Write JESS Files

Write the related JESS files. It includes rules, facts and databases. A simple example is shown as Figure 9. It shows that if the temperature is more than 20, then an alarm is given out. In the oil industry, there are actually many complicated rules to show when to give out alarms and what kind of alarms should be given out. For example, if the temperature changes, different temperature alarms should be given out. Those kinds of rules can be developed to JESS files in the way that is shown in the Example of Farmer's Dilemma Problem .

```
(bind ?data (fetch DATA))
;(bind ?data 50)

(while (> ?data 20) do
  (reset)
  (run)
  (printout t crlf " illegal data " crlf)
  (break)
)
```

Figure 9: A Simple Jess File

3.3.6 Step 6: Integrate JADE Agents with JESS

Now, the agents can communicate with each other in the environment JADE, and different rules can be written into JESS files, so the next step is to integrate JADE agents with JESS

seamlessly. Before integrating both tools, it is essential to keep in mind some of the main issues concerning the functioning of both JADE agents and Jess engines.

Introduction

JADE offers the environment and facilitates message sending/receiving; JESS enables a declarative implementation of the decision module. Their relationship is shown as Figure 10^[19]:

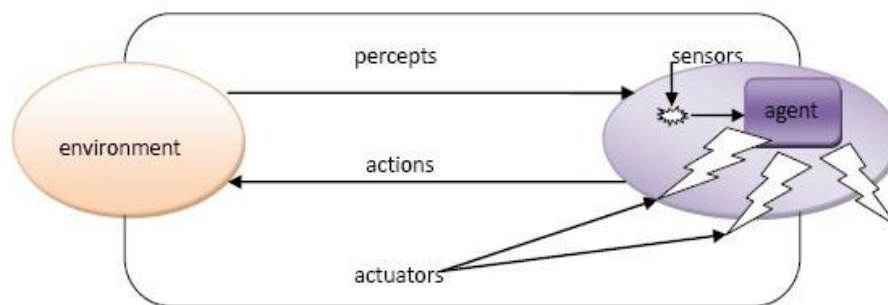


Figure 10: JADE and JESS

Implementation

When an agent reasons and makes decisions, it will interact with other agents, so, it is very important to remember that a JADE agent is single-threaded. When the user programs with agents, he should always take this into account.

JESS is a rule-based programming environment, a rule engine and scripting environment. It has a fast and efficient algorithm called Rete algorithm. In order to embed JESS into a JADE agent, first, the user needs to create a `jess.Rete` object and manipulate it appropriately; Second, the user needs to run the inference engine `Rete.run()` which is included in `Rete` class, after this, the engine will fire all the applicable rules and return when there are no more rules to fire; Third, the engine will stop when there are no more rules to fire and the block the calling thread. It is important to know that we will block the entire single-threaded agent if we block the calling thread. It also depends on how much time the reasoning will take. However, in `Rete` class, we have another run method which will allow us to specify the maximum number of cycles the engine should run. The integration of JADE and JESS is shown as Figure 11^[20].

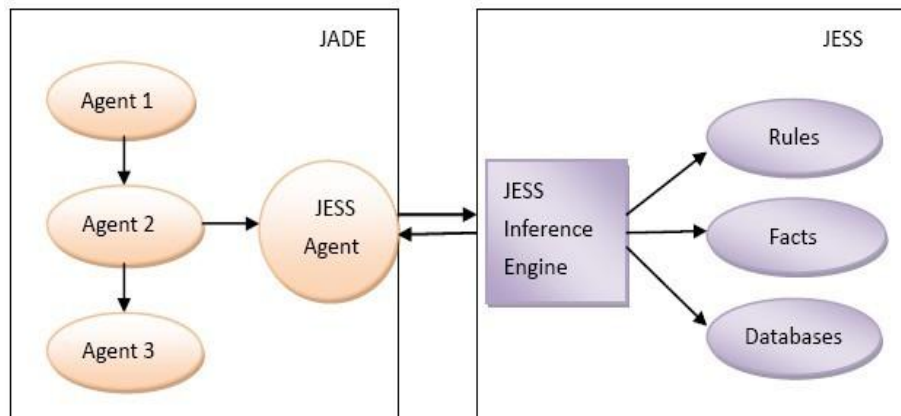


Figure 11: the Integration of JADE and JESS

Integrating JADE Agents with JESS ---Approach A^[19]

In order to integrate JESS with JADE agents, the user needs to embed an instance of the Jess engine inside a behaviour. When the agent is reasoning, we want it to be able to continuously reason until there are no more rules to fire, so, we need a CyclicBehaviour whose action will consist of running the Jess engine continuously.

When one agent is reasoning, please remember that don't block other agent's behaviours for a long time. The following code snippet shows the implementation:

```
class JessBehaviour extends CyclicBehaviour {
    // the JESS engine
    private jess.Rete jess;
    JessBehaviour(Agent agent, String jessFile) {
        super(agent);
        // create a JESS engine
        jess = new jess.Rete();
        // load the JESS file
        try {
            // open the JESS file
            FileReader fr = new FileReader(jessFile);
            // create a parser for the file
            jess.Jesp j = new jess.Jesp(fr, jess);
            // parse the input file into the engine
            try {
                j.parse(false);
            } catch (jess.JessException je) {
                je.printStackTrace();
            }
        }
    }
}
```

```

    }
    fr.close();
    } catch (IOException ioe) {
        System.err.println('Error loading Jess file - engine is empty');
    }
}
...
...
}

```

Integrating JADE Agents with JESS---Approach B

The following code shows how to integrate JADE and JESS in another way. Assume that `r` is an object of class `RETE`. In JESS, `Rete.store(String, Object)` `Rete.store(String, Value)`, `Rete.fetch(String)` can be used to store and fetch data. `r.store('DATA',j)` is used to pass `j` to `DATA` from JADE to JESS, where `j` belongs to JADE and `DATA` belongs to JESS respectively. `jessFile` can get the data with the function of `(fetch DATA)`. JESS can use `r. getGlobalContext ()` to pass data back to JADE.

```

class JessBehaviour extends CyclicBehaviour {
    private jess.Rete jess;
    JessBehaviour(Agent agent, String jessFile) {
        Rete engine = new jess.Rete();
        try {
            engine.store('DATA',j);
            engine.batch('jessFile');
            engine.executeCommand('(run)');
            Context result = engine.getGlobalContext();
            System.out.println(result);
        } catch (jess.JessException je)
        {
            je.printStackTrace();
        }
    }
}
...
...
}

```

3.3.7 Step 7: Test and Result

The result of a simple test is shown in Figure 12. When the temperature is more than 20 degree, it gives out an alarm: 'illegal data'. More complicated tests can be done with corresponding JESS rules, then, different kinds of alarm results will be given out.

```
sendAgent sent the message:24
reasonerAgent received message: 24
sendAgent sent the message:25

illegal data
[Context, 0 variables: ]
sendAgent sent the message:26
reasonerAgent received message: 25

illegal data
[Context, 0 variables: ]
sendAgent sent the message:27
```

Figure 12: Tests and Results

3.4 Conclusion of the Chapter

As discussed above, JESS and JADE can be used for real time monitoring of data, but besides the real time monitoring usage, what if the users want to save the monitoring results for future use and analysis. The following chapter will discuss a model like this.

4 Data Processing Model Based On Petri Net

As mentioned in the former chapter, JESS and JADE can be used for real time monitoring of data. Besides the real time usage, the monitoring results can also be stored into a database in case the user may want to check it or use it afterwards. This chapter developed a system based on Petri Net^[21] technique and the monitoring results database. The user can log into the system and check the data that he is interested. If the data exists, the user may want to check it or update it to the latest edition; if the data doesnt exist, the user may want to publish new data. The performance of the system is analysed as well.

4.1 Model

Based on the database mentioned above, a SUP (searching, updating or publishing) model is developed and shown as Figure 13.

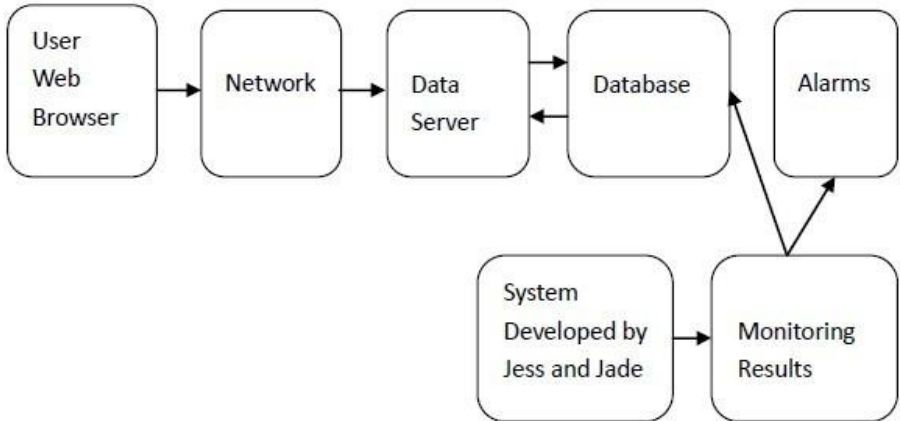


Figure 13: the Structure of SUP System

The user web browser collects the message input by the user, such as register message, user name, password etc. The data server responses user’s requests. There is a pool for collecting all the requests. If it is a login request, the server will first encrypt the password and then return the result to show whether the password is valid or not. If it is a searching, updating or publishing request, the server will construct the path of the database file which has data inside, and return the result after finishing the corresponding operations. The database manages all the data information, as mentioned above.

According to the model, a SUP (searching, updating or publishing) system is developed as well. The user can log into the system and check the data that he is interested. If the data exists, the user may want to check it or update it to the latest edition; if the data doesn't exist, the user may want to publish new data. Figure 14 shows the relationship between the modules.

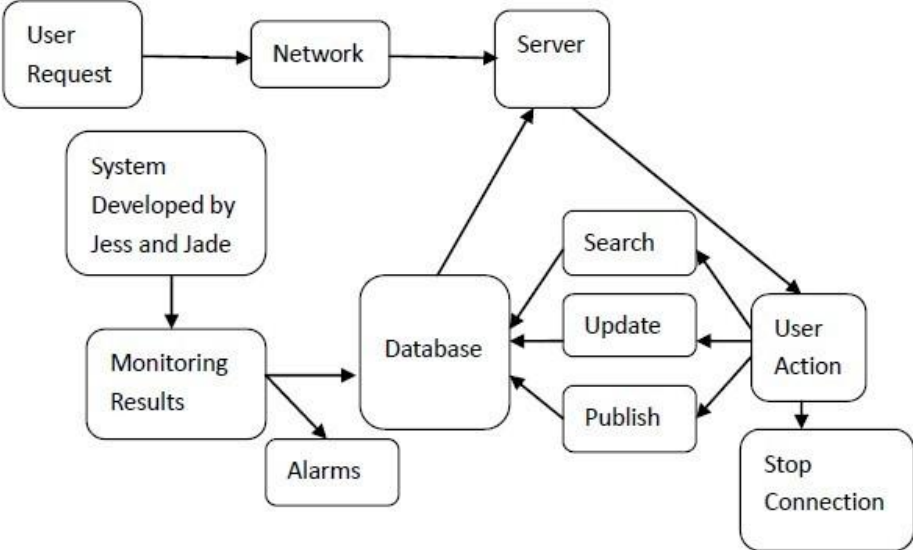


Figure 14: Connection Between Modules

The purpose is to test if the system can be implemented or not. If so, we want to test the performance of the system, such as the average response time for each login request, ajax request, searching request, updating request or publishing request.

4.2 Developing Environment

4.2.1 GPenSIM

Because of Petri Net's^[22] good modeling and simulation of discrete event-driven systems, it is becoming more and more widely used by the research communities. For free academic use, there are a number of Petri Net tools available, however, these tools sometimes are very slow and different when programming via graphical user interfaces. GPenSIM^[23] is one of the Petri Net tools for modeling and simulation of discrete-event systems. Compared to other tools, it has three distinct advantages:

- 1) First, it is very simple and easy for the user to use because there are not many rules to remember.

- 2) Second, it allows the user to program with a very simple language because it is a non-graphic program.
- 3) Third, it allows the user to make use of matlab toolboxes like fuzzy logic, control systems, etc because it is well integrated with MATLAB^{[24][25]} platform.

In order to install GPenSIM, we need to unzip the GPenSIM toolbox functions under a directory first, and then start matlab and go to the file menu to select “set path” command, then select “add with subfolders” to add GPenSIM directory.

4.2.2 PDF, TDF and MSF

In order to define a Petri Net, it involves two steps:

- 1) First, the user needs to define the Petri Net graph which is also called Petri Net structure. This is the static part. In order to define the Petri Net graph, the user has to identify the basic elements (places and transitions) and connect these elements with arcs.
- 2) Second, the user needs to define the Petri Net markings. This is the dynamic part.

Petri net Definition File (PDF)

In GPenSIM, PDF^[23] is short for Petri Net Definition File. PDF offers the definition of a Petri net graph. The Petri Net model can be divided into many modules, and each module can be defined in a separate PDF, so there may be a number of PDFs.

The codes of defining the Petri Net graph are shown as below:

```
Function [PN name, set of places, set of trans, set of arcs] ...
= SUP_pn_def(global info)
PN_name = '.....';
set of places = {.....};
set of trans = {.....};
set of arcs = {.....};
```

Transition Definition File (TDF)

In GPenSIM, TDF^[23] is short for Transition Definition File. If there are enough tokens in the input places, the transition can fire. The transition which can fire is called enabled transition. The conditions for firing a transition are kept in a TDF.

A simple example can be seen as below:

```
function [fire, new_color, override, selected_tokens, global_info] = ...
    tUL_def (PNname, new_color, override, selected_tokens, global_info)
new_color = '...';
if global_info.GI >= 1
    global_info.GI = global_info.GI - 1;
    fire = 1;
else
    fire = 0;
end;
```

The Names of the TDF files must follow a strict naming policy. For example, the TDF for the transition 'trans1' must be named 'trans1_def.m'.

Main Simulation File(MSF)

In GPenSIM, MSF^[23] is short for Main Simulation File. After the Petri net graph is defined, the MSF can be written as below:

```
clear, clc;
pn = petrinetgraph('...');
global_info.MAX_LOG_SIZE = .....;
global_info.GI = .....;
.....
dynamic.initial_markings={.....};
dynamic.firing_times={...}
[Results, global_info, colormap] = gpensim(PNname, dynamic_info, global_info);
printsys(PN_name, Results);
print_colormap(PN_name, colormap, '...');
plotp(PN_name, Results, {.....});
```

The roles of PDF, TDF and MSF

The processing steps of a Petri Net graph simulation are as follows:

- 1) Right before the simulation starts, a PDF will be loaded into memory by MSF. The only use of a PDF is to represent a static Petri net graph.
- 2) MSF loads PDF (or PDFs) into memory and then starts the simulation.
- 3) After the simulation starts, the TDF will be called if the transition for this TDF is enabled. Meanwhile, MSF will be blocked during the simulation runs; therefore, MSF does not

have any control of what is going on during the simulation.

- 4) When the simulation completes, the control will be passed back to MSF again, and also the simulation results.

The relationship among PDF, TDF and MSF can be seen in Figure 15^[23].

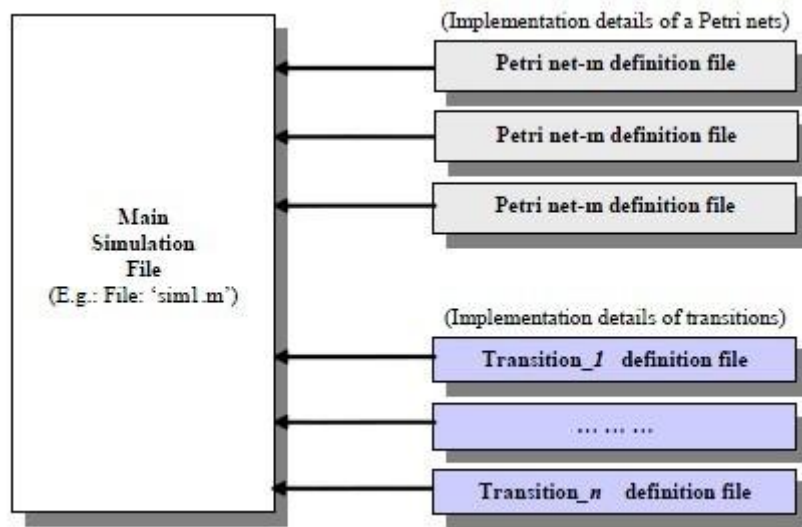


Figure 15: the Relationship among PDF, MSF and TDF

4.2.3 Global Info

Different files such as main simulation file MSF, Petri net definition files PDFs, and transition definition files TDFs are defined by the user. These different files need to exchange parameters and values with each other.



Figure 16: Global_info

A packet called ‘global_info’ is used here to access and exchange global parameters and values, as can be seen from Figure 16^[23]. The parameters and values will be packed together as a global_info packet if they need to be passed to different files. Global_info packet is visible in all the files, so the values in the packet can be read and changed.

4.2.4 Colored GPenSIM

There are 2 different kinds of tokens. The tokens which will be consumed when the transition fires are called input tokens; The tokens which will be deposited when the transition fires are called output tokens.

If the tokens inside a place are distinguishable, it matters which token arrives into the place first or last, it also matters whether a token is deposited into a place by one transition or another. In this case, every token is unique, it is identifiable with a unique token ID.

A token like this has a structure consisting of 3 elements:

- tokID (integer value): a unique token ID.
- creation_time (integer value): the time the token was created by a transition.
- t_color (set of strings): a set of colors.

In GPenSIM^[23], only transitions can manipulate the colors. The colors are inherited by default, which means when a transition fires, it collects all the colors from the input tokens and then passes these colors to the output tokens. However, colors inheritance can be prevented by overriding. An enabled transition can select specific input tokens based on preferred colors; it can also select specific input tokens based on the time when tokens are created.

In GPenSIM, colored tokens can only be utilized by transitions; the transition definition files can be coded with controlling colored tokens^[23] since the transitions are active:

- When a transition fires, the colors are inherited by default. This means when a transition fires, it inherits colors of all input tokens and deposits new tokens into output places. The new tokens have all the colors inherited from the input tokens. The inheritance of the colors can be prohibited by overriding.
- When a transition fires, it can consume input tokens both with or without specific colors.
- When new tokens are deposited into the output place, new colors can be added by the

transition in addition to the inherited colors. However, if the inheritance is overridden, the output tokens will only have the new color added by the transition.

4.3 Implementation of the Model

4.3.1 Petri Net Graph of the System

The Petri Net graph of the whole system^[26] can be seen in Figure 17.

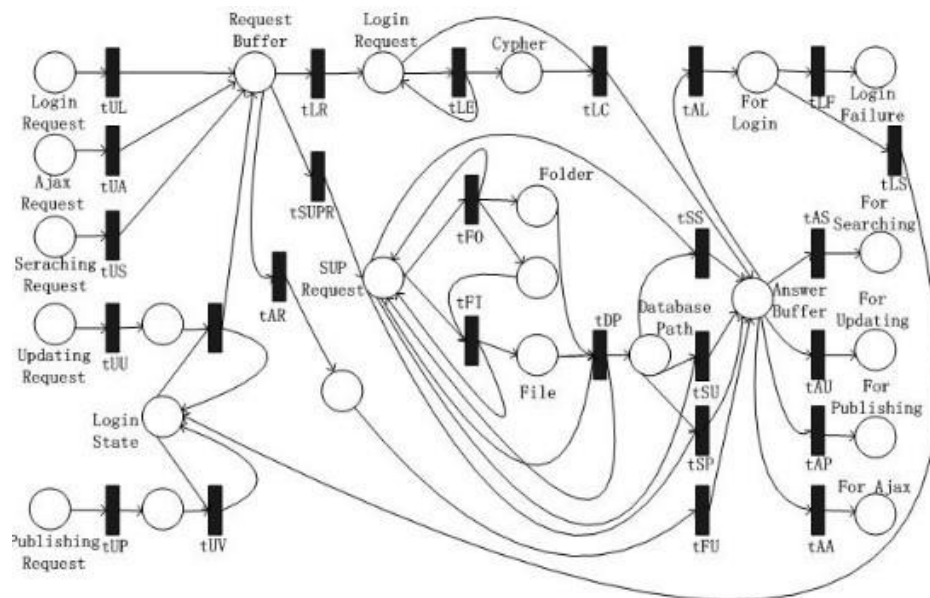


Figure 17: Petri Net Graph of the Whole System

4.3.2 Module of the User Side

There are five kinds of requests are available in the user side, as can be seen from Figure 18, they are 'Login Request', 'Ajax Request^[27]', 'Searching Request', 'Updating Request', 'Publishing Request'.

Sometimes, it may be hard for a user to memorize the full name, so, if the user just input the first several letters, an Ajax request will be sent to the server to get all available names starts with the inputted words.

In order to produce these requests, five transitions which give out requests are used here. A request buffer is used to collect all the requests.

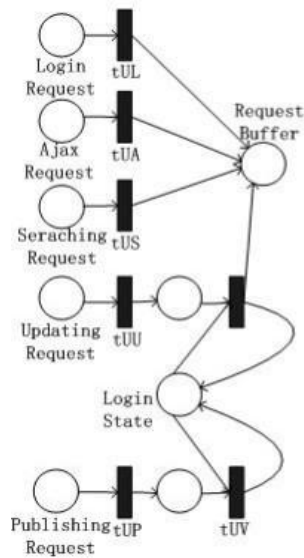


Figure 18: Petri Net of User Side

4.3.3 Module of Login Request

Transition 'tLR' is used to select login requests from the request buffer. The login information consists of username and password. The password is encrypted because it is not safe to store password using clear text. Username obtained from the original request is used to find the corresponding encrypted password stored in password file. The result of verifying the password which can be either successful or failed is passed to the answer buffer. The Petri Net of this module can be seen in Figure 19.

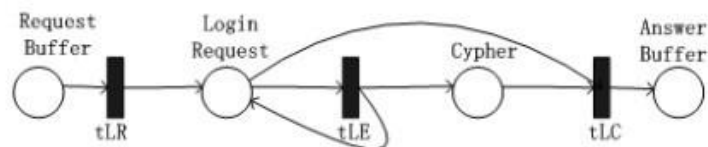


Figure 19: Module of the Login Request

4.3.4 Module of SUP Request

Transition 'tSUPR' is used to select searching requests, updating requests and publishing requests. When doing the searching, updating and publishing operations, the path of the database file need to be known first. In order to find database files faster, a special method is used here to construct the path of database file directly. Transitions 'tFO' and 'tFI' are used to find folder name and file name of the database file, and then transition 'tDP' gives full path of

database file. As 'tFO' and 'tFI' cannot fire at the same time, so an additional place is used here to let them fire alternatively, which will make sure the number of tokens in searching, updating and publishing requests never increase or decrease abnormally. The Petri Net of this module can be seen in Figure 20.

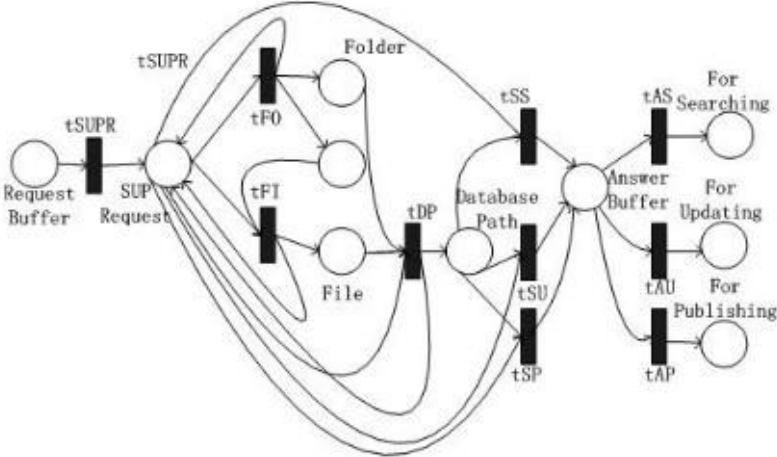


Figure 20: Module of the SUP Request

4.3.5 Module of the Answer Side

All the answers to the different requests are put in the answer buffer. Five transitions(tAL, tAS, tAU, tAP and tAA) are used to select different answers from the buffer. The answer for login request is a little special as it has two possible states which indicate successful and failed. The Petri Net of this module can be seen in Figure 21.

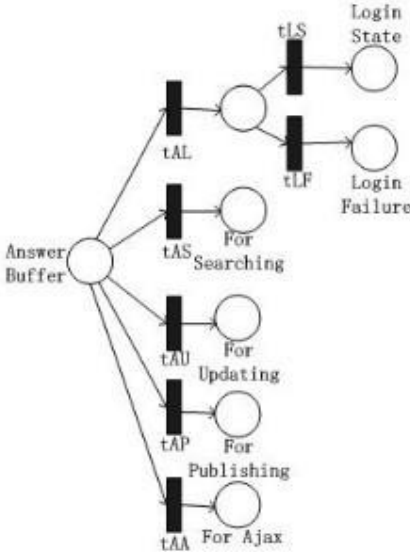


Figure 21: Module of the Response Side

4.3.6 Performance of the System

In this chapter, we will simulate the system and test the performance of it, such as the average time for each login request, ajax request, searching request, updating request or publishing request. The firing time for each transition varies slightly. The firing times for the transitions are defined in the MSF file.

Step 1: Response Time of a Login Request.

In order to get the time, global info.Gl is set to 1, and the other three variables are set to 0. After running the simulation, we get the time showed in Figure 22. As can be seen, it is very fast as well with a time about 0.2 second.

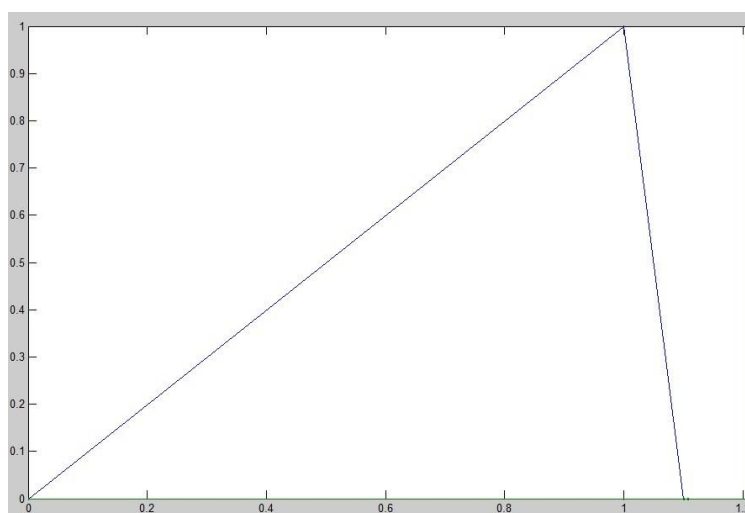


Figure 22: Response Time of a Login Request

Step 2: Response Time of an Ajax Request.

In order to get the time, global info.Ga is set to 1, while other three variables are set to 0. After running the simulation, we get the time showed in Figure 23. As can be seen, it is very fast with a time about 0.2 second.

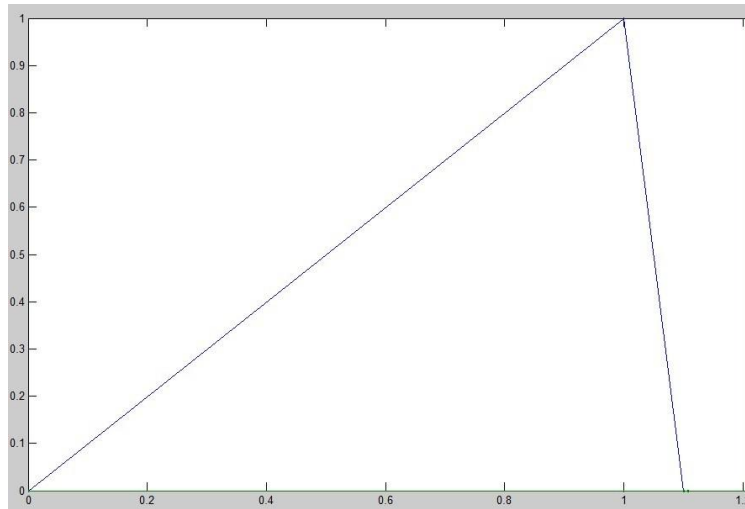


Figure 23: Response Time of an Ajax Request

Step 3: Response Time of a Searching Request.

In order to get the time, global info.Gs is set to 1, and the other three variables are set to 0. After running the simulation, we get the time showed in Figure 24. As can be seen, it is also very fast with a time about 0.2 second.

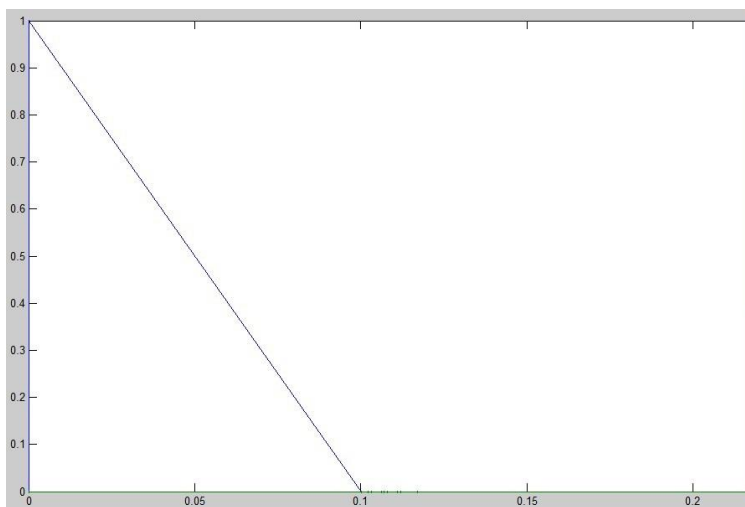


Figure 24: Response Time of a Searching Request

Step 4: Response Time of a Updating Request.

In order to get the time, global info.Gl and global info.Gu are set to 1, and the other two variables are set to 0. After running the simulation, we get the time showed in Figure 25. As can be seen, it's still very fast with a time about 0.4 second.

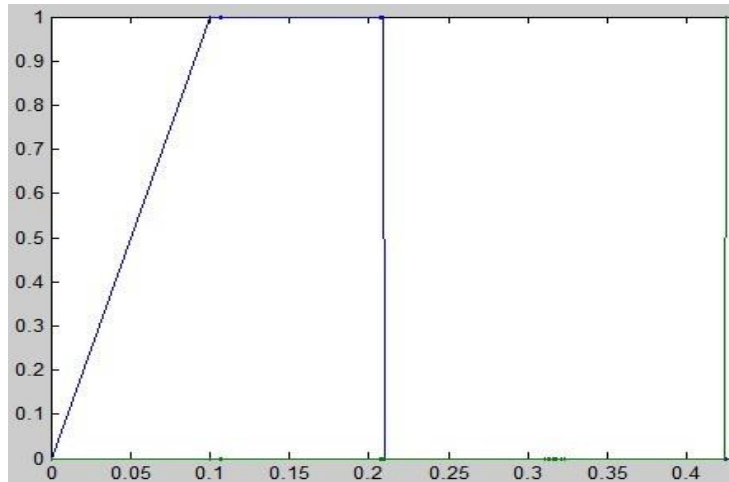


Figure 25: Response Time of a Updating Request

Step 5: Response Time of a Publishing Request.

In order to get the time, global info.Gl and global info.Gp are set to 1, and the other two variables are set to 0. After running the simulation, we get the time showed in Figure 26. As can be seen, it's still very fast with a time about 0.4 second.

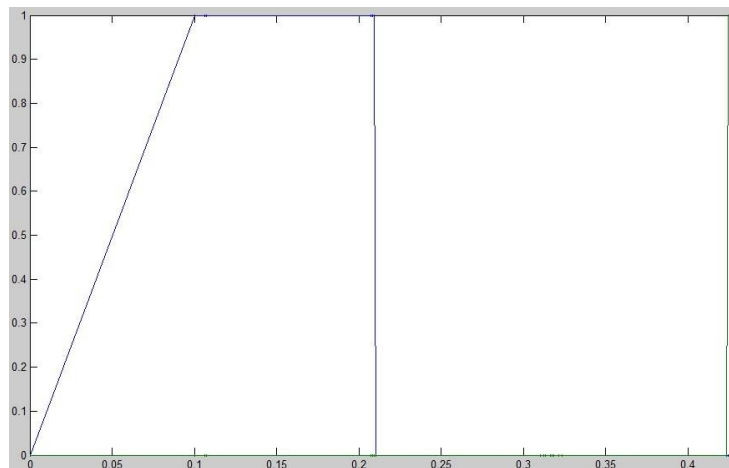


Figure 26: Response Time of a Publishing Request

4.4 Conclusion of the Chapter

As can be seen from the simulation results, we can see that the SUP system is robust because it only takes about 0.2 second to finish a request. It is still very fast to get a response even if there may be hundreds of requests per second. But what if there are thousands of requests per second, or even more? In this situation, one server is not enough, multi servers are required. A new model which will use multi servers in a distributed environment will be discussed in next chapter.

5 Data Processing Model Based On Rserve

As mentioned in the former chapter, only a single server is used in that system, but when there is much data need to be processed, multi servers are required. In order to solve this problem, the author does some research on R [28][29][30][31] and Rserve with multi servers in a distributed environment. This chapter chooses R as the reasoning technique for data processing. The author first gives out a model to show how she wants to process data with R. Based on this model, the author also does a research on the interface between Rserve and Pig [32].

5.1 Model

This chapter is based on a model which is used to process data with multi servers in a distributed environment. The model can be seen in Figure 27.

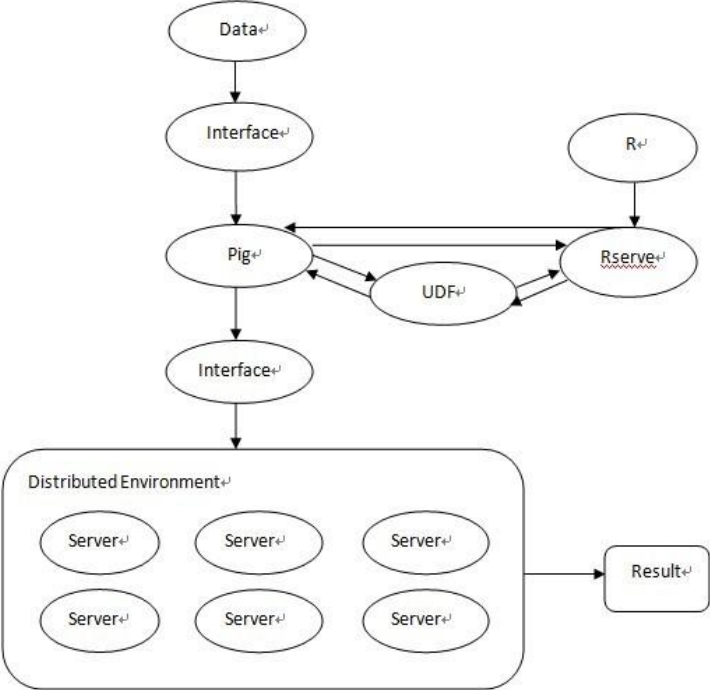


Figure 27: The Model

As can be seen from the model, R is used as a reasoning technique here. R provides a wide variety of statistical and graphical techniques, and is highly extensible. Rserve is a TCP/IP server which allows other programs to use the functions of R from various languages without the need to initialize R or link against R library.

5.2 Developing Environment

5.2.1 Pig

When there are large data sets, the user needs a high-level language to express the data analysis and the complicated programming infrastructures. Pig^[32] is a platform for this. The reason why pig programs can handle very large data sets is because they have an important property: their structure is amenable to substantial parallelization.

5.2.2 Pig Latin

At the present time, Pig has several layers, such as infrastructure layer and language layer. The infrastructure layer consists of a compiler and it produces sequences of Map-Reduce programs. The language layer consists of a textual language called Pig Latin^[33]. Pig Latin has some very important properties such as ease of programming, optimization opportunities and extensibility etc.

There are different kinds of Pig Latin statements. Normally, a Pig Latin statement is an operator that takes a relation as input and produces another relation as output. However, LOAD and STORE statements are different, they are used to read data from and write data to the file system. The user can write Pig Latin statements for multiple lines but must end them with a semi-colon ';'. Pig Latin statements are generally organized in the following manner:

- 1) A LOAD statement - used to read data from the file system.
- 2) A series of "transformation" statements - used to process data.
- 3) A DUMP statement - used to display outputs to the screen, or
- 4) A STORE statement - used to write outputs to the file system.

A simple example can be shown as below. The example is used to load the file 'jing/passwd' first, then generate id and display it.

```
A = load 'jing/passwd' using PigStorage(':');  
B = foreach A generate $0 as id;  
dump B;
```

5.2.3 Pig Code Written In Java

Pig codes can be written in Java, a simple example is shown as below. This example implements the same functionality with the example in the former chapter.

```
public static void runIdQuery(PigServer pigServer, String inputFile) throws IOException
{
    pigServer.registerQuery("A = load 'jing/passwd' using PigStorage(':');");
    pigServer.registerQuery("A = load " + inputFile + " using PigStorage(':');");
    pigServer.registerQuery("B = foreach A generate $0 as id;");
    pigServer.store("B", "jing/idoutb");
}
```

5.2.4 Pig Installation

The following steps ^[34] show how to install pig:

- 1) Install Java. Java 1.6 or higher is installed; JAVA_HOME environment variable is set to the root of the Java installation.
- 2) Install Pig. To install Pig, do the following:
 - Download the Pig tutorial file to your local directory.
 - Unzip the Pig tutorial file and store them in a newly created directory named pigtmp.
 - Move to the pigtmp directory.
 - Review the contents of the Pig tutorial file.
 - Copy the pig.jar file to the appropriate directory on your system. For example: /home/me/pig.
 - Create an environment variable, PIGDIR, and point it to your directory. For example, export PIGDIR=/home/me/pig (bash, sh) or setenv PIGDIR /home/me/pig (tcsh, csh).
- 3) Run the Pig scripts - in Local or Hadoop mode.

The user can run pig cripts in both local mode and mapreduce mode. If the user wants to run pig cripts in local mode, he doesn't need to install Hadoop or HDFS. However, if the user wants to run pig cripts in mapreduce mode, he has to install Hadoop or HDFS first.

To run the Pig scripts in local mode^[34], do the following:

- Set the maximum memory for Java.

```
java -Xmx256m -cp pig.jar org.apache.pig.Main -x local script-local.pig
```
- Move to the pigtmp directory.

- Review Pig Script.

- Execute the following command.

```
$ java -cp $PIGDIR/pig.jar org.apache.pig.Main -x local script-local.pig
```

- Review the result files, located in the part-r-00000 directory.

The output may contain a few Hadoop warnings which can be ignored:

```
2010-04-08 12:55:33,642 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics
```

```
Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
```

To run the Pig Scripts in Mapreduce Mode^[34], do the following:

- Move to the pigtmp directory.

- Review Pig Script.

- Copy the excite.log.bz2 file from the pigtmp directory to the HDFS directory.

```
$ hadoop fs -copyFromLocal excite.log.bz2
```

- Set the HADOOP_CONF_DIR environment variable to the location of your core-site.xml, hdfs-site.xml and mapred-site.xml files.

- Execute the following command:

```
$ java -cp $PIGDIR/pig.jar:$HADOOP_CONF_DIR org.apache.pig.Main script-hadoop.pig
```

- Review the result files, located in the script-hadoop-results:

```
$ hadoop fs -ls script-hadoop-results
```

```
$ hadoop fs -cat 'script-hadoop-results/*' | less
```

5.2.5 Pig UDF

When using pig, the user may need to specify custom processing. Pig provides extensive support for user-defined functions (UDFs)^[35]. We can use existing functions as UDFs, and we can also write our own functions. These functions can be used as a part of almost every operator in Pig.

The following steps show how to create and use a UDF in pig:

- 1) Write our own functions or use the existing functions.
- 2) Build pig.jar which is used to compile the UDF.
- 3) Compile the UDF and then create a jar file that contains it.
- 4) Register the jar file which contains the UDF in order to provide the location of the jar file.
Note that there are no quotes around the jar file because having quotes would result in a syntax error.
- 5) Run the script that uses the UDF.

5.3 Implementation of the Model

5.3.1 Step 1: install Rserve

We need to have R-1.5.0 or higher installed on our system in order to be able to use Rserve. The easiest way to install Rserve is to install it from CRAN, simply use

```
install.packages("Rserve")
```

in R. Unix users, please note that R must have been configured with `--enable-R-shlib` in order to use Rserve. Rserve comes now as an R package, so one way to start Rserve is from within R, just type

```
library(Rserve)  
Rserve()
```

If the Rserve is successfully installed, we can see the following result shown in Figure 28:

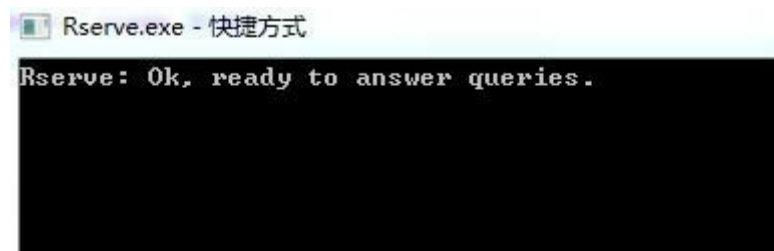


Figure 28: Rserve Installation

The server is nothing without clients, we will use java REngine Java client in the following chapters. This client API is more flexible, with better design, has better exception handling and is aimed to support both JRI and Rserve transparently. Java client is located in `src/client/java-new`. It is a full client suite that allows any Java application (JDK 1.4 or higher) to access an Rserve. The suite is written entirely in Java. It provides automatic type translation for most objects such as `int`, `double`, `arrays`, `string` or `vector` and classes for special R objects such as `RBool`, `RList` etc.

One can use the following code to test if the Rserve is successfully installed or not:

```
import org.rosuda.REngine.*;
import org.rosuda.REngine.Rserve.*;
import org.rosuda.REngine.Rserve.protocol.*;
public class Rtest {
    public static void main(String [] args)
    {
        Rtest myt = new Rtest();
        System.out.println("hello world");
    }

    public Rtest()
    {
        try{
            RConnection c = new RConnection("127.0.0.1");
            REXP x = c.eval("R.version.string");
            System.out.println(x.asString());
        }catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

If the java client is successfully installed, we can see the following result shown in Figure 29:

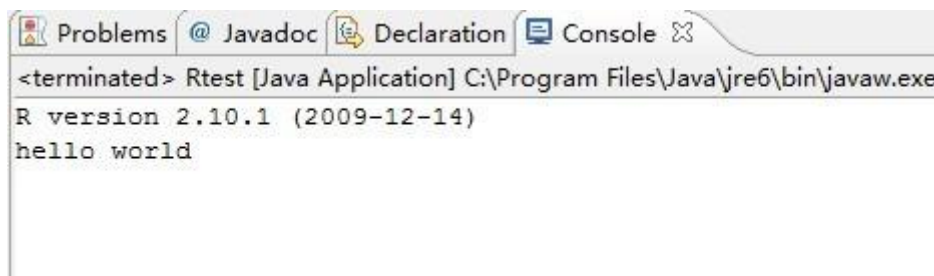


Figure 29: Rserve Test Result

5.3.2 Step 2: install Pig

Pig can be installed according to the steps described in chapter 4.2.3. The following code can be used to check if the pig is installed successfully or not.

```
import java.io.IOException;
import org.apache.pig.PigServer;
public class pigTest{
    public static void main(String[] args) {
        try {
            PigServer pigServer = new PigServer("mapreduce");
            runIdQuery(pigServer, "jing/passwd");
        }
        catch(Exception e) {
        }
    }
    public static void runIdQuery(PigServer pigServer, String inputFile) throws IOException {
        pigServer.registerQuery("A = load '" + inputFile + "' using PigStorage(':');");
        pigServer.registerQuery("B = foreach A generate $0 as id;");
        pigServer.store("B", "jing/idout");
    }
}
```

If the pig is installed successfully, we can see the following result shown in Figure 30:

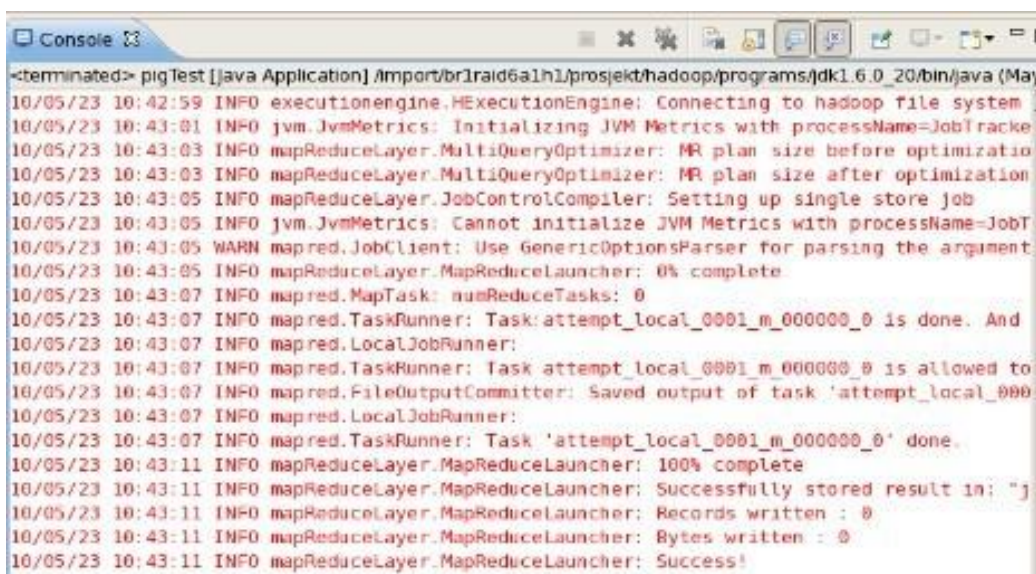


Figure 30: Pig Installation

5.3.3 Step 3: embed Rserve into Pig

The following codes show how to embed R functions into Pig codes.

```
import java.io.IOException;
import org.apache.pig.PigServer;
import org.rosuda.REngine.*;
import org.rosuda.REngine.Rserve.*;
import org.rosuda.REngine.Rserve.protocol.*;

public class pigRserver{
    .....
    public static void main(String[] args) {
    try {
        RTest myt = new RTest();
        PigServer pigServer = new PigServer("mapreduce");
        runIdQuery(pigServer, "...");
        .....
    }
    catch(Exception e) {
        .....
    }
}

public static void runIdQuery(PigServer pigServer, String inputFile) throws IOException {
    pigServer.registerQuery("...");
    pigServer.registerQuery("...");
    pigServer.store("...", "...");
    .....
}

public void RTest()
{
    try{
        RConnection c = new RConnection("152.94.1.68");
        .....
    }catch(Exception e)
    {
        e.printStackTrace();
    }
}
```

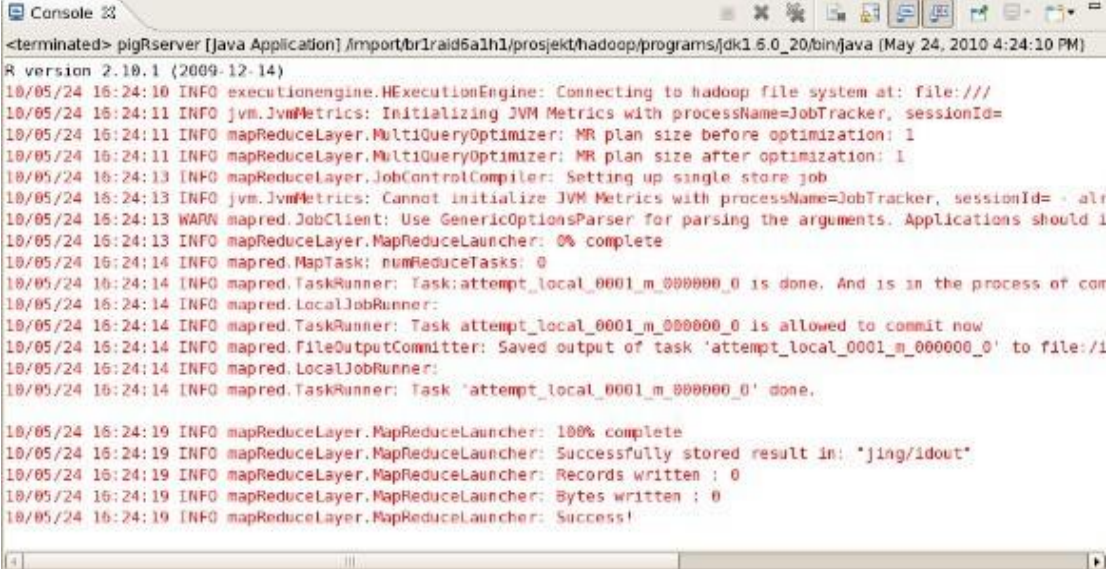
```
}  
}
```

The Pig server and Rserve can share the same parameters. First, Rserve processes the data with R functions, then, pig codes can invoke the processing results in pig server. A simple example can be shown as below.

```
import java.io.IOException;  
import org.apache.pig.PigServer;  
import org.rosuda.REngine.*;  
import org.rosuda.REngine.Rserve.*;  
import org.rosuda.REngine.Rserve.protocol.*;  
public class pigRserver{  
    static String a="";  
    public static void main(String[] args) {  
        try {  
            RTest myt = new RTest();  
            PigServer pigServer = new PigServer("mapreduce");  
            runIdQuery(pigServer, "jing/passwd");  
        }  
        catch(Exception e) {  
        }  
    }  
    public static void runIdQuery(PigServer pigServer, String inputFile) throws IOException {  
        pigServer.registerQuery("A = load " + inputFile + " using PigStorage(':');");  
        pigServer.registerQuery("B = foreach A generate $0 as id;");  
        pigServer.store("B", "jing/idout");  
        System.out.println(a);  
    }  
    public void RTest()  
    {  
        try{  
            RConnection c = new RConnection("152.94.1.68");  
            REXP x = c.eval("R.version.string");  
            a=x.asString();  
        }catch(Exception e)  
        {  
            e.printStackTrace();  
        }  
    }  
}
```

```
}  
}  
}
```

The result can be seen in Figure 31.



```
<terminated> pigRserver [Java Application] /import/br1raid6alh1/prosjekt/hadoop/programs/jdk1.6.0_20/bin/java (May 24, 2010 4:24:10 PM)  
R version 2.10.1 (2009-12-14)  
10/05/24 16:24:10 INFO executionengine.HExecutionEngine: Connecting to hadoop file system at: file:///   
10/05/24 16:24:11 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=   
10/05/24 16:24:11 INFO mapReduceLayer.MultiQueryOptimizer: MR plan size before optimization: 1   
10/05/24 16:24:11 INFO mapReduceLayer.MultiQueryOptimizer: MR plan size after optimization: 1   
10/05/24 16:24:13 INFO mapReduceLayer.JobControlCompiler: Setting up single store job   
10/05/24 16:24:13 INFO jvm.JvmMetrics: Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - all   
10/05/24 16:24:13 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should i   
10/05/24 16:24:13 INFO mapReduceLayer.MapReduceLauncher: 0% complete   
10/05/24 16:24:14 INFO mapred.MapTask: numReduceTasks: 0   
10/05/24 16:24:14 INFO mapred.TaskRunner: Task:attempt_local_0001_m_000000_0 is done. And is in the process of com   
10/05/24 16:24:14 INFO mapred.LocalJobRunner:   
10/05/24 16:24:14 INFO mapred.TaskRunner: Task attempt_local_0001_m_000000_0 is allowed to commit now   
10/05/24 16:24:14 INFO mapred.FileOutputCommitter: Saved output of task 'attempt_local_0001_m_000000_0' to file:/i   
10/05/24 16:24:14 INFO mapred.LocalJobRunner:   
10/05/24 16:24:14 INFO mapred.TaskRunner: Task 'attempt_local_0001_m_000000_0' done.   
  
10/05/24 16:24:19 INFO mapReduceLayer.MapReduceLauncher: 100% complete   
10/05/24 16:24:19 INFO mapReduceLayer.MapReduceLauncher: Successfully stored result in: "jing/idout"   
10/05/24 16:24:19 INFO mapReduceLayer.MapReduceLauncher: Records written : 0   
10/05/24 16:24:19 INFO mapReduceLayer.MapReduceLauncher: Bytes written : 0   
10/05/24 16:24:19 INFO mapReduceLayer.MapReduceLauncher: Success!
```

Figure 31: Embed Rserve into Pig

So, R functions can be embedded in Pig codes in this way, but it's very limiting. The users would be forced to use some custom java code every time they need R functions in Pig.

5.3.4 Step 3: Pig UDF with Rserve

Step 1: Why develop UDF with Rserve?

R functions can be embedded into pig but it's very limiting. The users would be forced to use some custom java code every time they need R in Pig. R function has to be visible like any other function in Pig for the users. Pig provides extensive support for UDFs as a way to specify custom processing. Functions can be a part of almost every operator in Pig.

Step 2: Write our own functions or use the existing functions.

An example^[35] can be shown as below. The example is from 'Pig UDF Manual', but there are some mistakes in this manual and the codes from it don't work, so, some changes are made to the codes and shown as below.

```

package myudfs;
import java.io.IOException;
import java.util.List; ;
import java.util.ArrayList;
import org.apache.pig.EvalFunc;
import org.apache.pig.PigWarning;
import org.apache.pig.data.Tuple;
import org.apache.pig.data.DataType;
import org.apache.pig.impl.logicalLayer.schema.Schema;
import org.apache.pig.impl.logicalLayer.FrontendException;
import org.apache.pig.FuncSpec;
public class UPPER extends EvalFunc <String> {
public String exec(Tuple input) throws IOException {
if (input == null || input.size() == 0)
return null;
String str = null;
try {
str = (String)input.get(0);
return str.toUpperCase();
}
catch (ClassCastException e) {
warn("unable to cast input "+input.get(0)+" of class "+
input.get(0).getClass()+" to String", PigWarning.UDF_WARNING_1);
return null;
}
catch(Exception e){
warn("Error processing input "+input.get(0), PigWarning.UDF_WARNING_1);
return null;
}
}
@Override
public Schema outputSchema(Schema input) {
return new Schema(new
Schema.FieldSchema(getSchemaName(this.getClass().getName().toLowerCase(), input),
DataType.CHARARRAY));
}
@Override
public List<FuncSpec> getArgToFuncMapping() throws FrontendException {

```

```

List<FuncSpec> funcList = new ArrayList<FuncSpec>();
funcList.add(new FuncSpec(this.getClass().getName(), new Schema(new Schema.FieldSchema(null,
DataType.CHARARRAY))));
return funcList;
}
}

```

The first line indicates that the function is part of the myudfs package. The UDF class extends the EvalFunc class which is the base class for all eval functions. It is parameterized with the return type of the UDF which is a Java String in this case. The next step is to implement the exec function. This function is invoked on every input tuple. The input into the function is a tuple with input parameters in the order they are passed to the function in the Pig script. In our example, it will contain a single string field corresponding to the user ID.

The first thing to decide is to check if the input data is null or empty; If so, it returns null. If not, the first letter of the user ID will be turned into capital letter.

Step 3: Build pig.jar which is used to compile the UDF.

Now the functions are implemented, it needs to be compiled and included in a jar. We need to build pig.jar to compile the UDF. The following sets of commands are used to check out the code from SVN repository and create pig.jar^[35]:

```

svn co http://svn.apache.org/repos/asf/hadoop/pig/trunk
cd trunk
ant

```

Step 4: Use pig.jar to compile the UDF and then create a jar file that contains it.

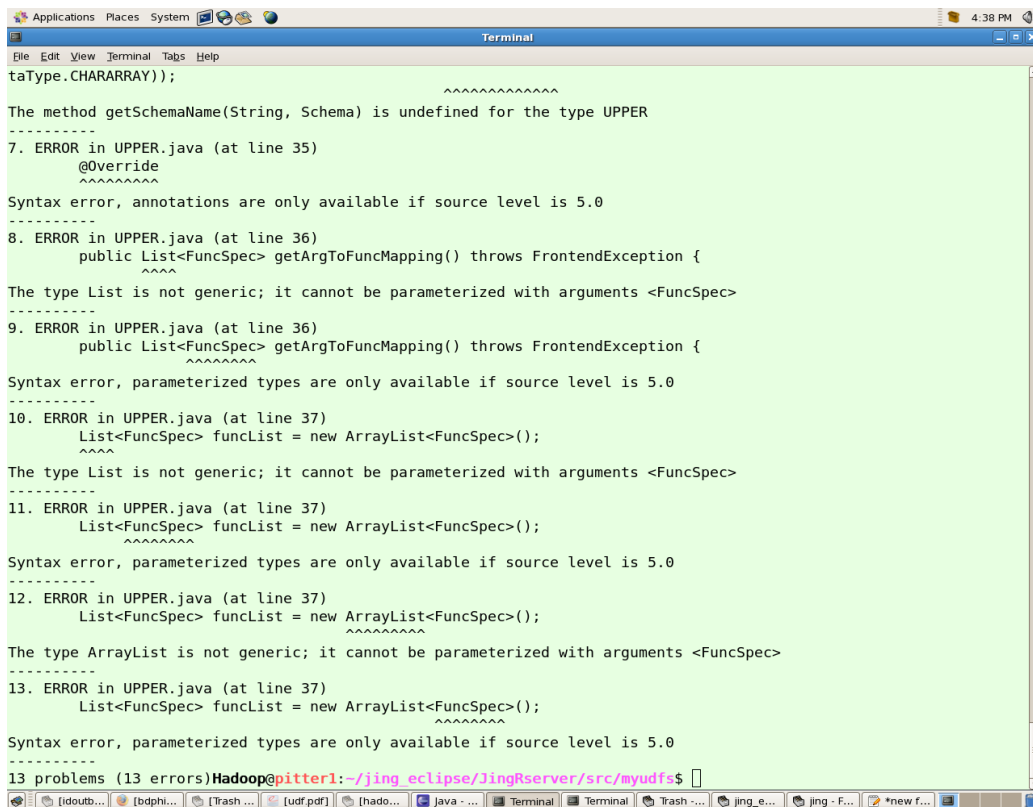
Now, pig.jar should be seen in the current working directory. The set of commands below first compiles the function^[35] and then creates a jar file that contains it.

```

cd packagename
javac -cp pig.jar javafilename.java
cd ..
jar -cf jarname.jar packagename

```

When we try to compile UDF with pig.jar, several errors are shown in Figure 32.



```
taType.CHARARRAY));
~~~~~
The method getSchemaName(String, Schema) is undefined for the type UPPER
-----
7. ERROR in UPPER.java (at line 35)
  @Override
  ~~~~~
Syntax error, annotations are only available if source level is 5.0
-----
8. ERROR in UPPER.java (at line 36)
  public List<FuncSpec> getArgToFuncMapping() throws FrontendException {
  ~~~~
The type List is not generic; it cannot be parameterized with arguments <FuncSpec>
-----
9. ERROR in UPPER.java (at line 36)
  public List<FuncSpec> getArgToFuncMapping() throws FrontendException {
  ~~~~~
Syntax error, parameterized types are only available if source level is 5.0
-----
10. ERROR in UPPER.java (at line 37)
  List<FuncSpec> funcList = new ArrayList<FuncSpec>();
  ~~~~
The type List is not generic; it cannot be parameterized with arguments <FuncSpec>
-----
11. ERROR in UPPER.java (at line 37)
  List<FuncSpec> funcList = new ArrayList<FuncSpec>();
  ~~~~~
Syntax error, parameterized types are only available if source level is 5.0
-----
12. ERROR in UPPER.java (at line 37)
  List<FuncSpec> funcList = new ArrayList<FuncSpec>();
  ~~~~~
The type ArrayList is not generic; it cannot be parameterized with arguments <FuncSpec>
-----
13. ERROR in UPPER.java (at line 37)
  List<FuncSpec> funcList = new ArrayList<FuncSpec>();
  ~~~~~
Syntax error, parameterized types are only available if source level is 5.0
-----
13 problems (13 errors)Hadoop@pitter1:~/jing eclipse/JingRserver/src/myudfs$
```

Figure 32: Syntax Error

As can be seen from Figure 32, almost all the errors are resulted by ‘Syntax error, parameterized types are only available if source level is 5.0’. If the user is using an IDE, in Eclipse for example, there are 2 ways to fix it: First, install Java 5.0 (Window > Preferences > Java > Installed JREs); Second, The compiler compliance level has to be elected to 5.0 (Window > Preferences > Java > Compiler). After it is fixed, we can see that the Java edition we are using is a newer one. So, we can compile the UDF with pig.jar again, but the same errors happen again. Now we already use the higher edition of Java, why the errors happen again? This is because even if JVM or the user’s IDE’s environment might run on a different version after Java 5.0 is installed, we are programming in a hadoop cluster and we are using cmd window to compile the UDF, it has its own java. So, even if the edition of the Java is a higher edition in Eclipse, but the Java which cmd window is using is still a older edition.

The Java edition which cmd window is using can be checked with the code ‘java -edition’, now we can see that the cmd window is still using the older edition. The way to fix it is go to

the file which has a higher edition, and use the java there. A simple example can be seen as below:

```
bin/javac -cp /home/prosjekt/hadoop/jing_eclipse/JingRserver/src/myudfs/pig.jar
/home/prosjekt/hadoop/jing_eclipse/JingRserver/src/myudfs/RTest2.java
```

Now myudfs.jar should be seen in the current working directory. This jar can be used in the script now.

Step 5: Register the jar file which contains the UDF in order to provide the location of the jar file.

Users often run their scripts in different environments, so jar locations or versions may change. Note that there are no quotes around the jar file because having quotes would result in a syntax error.

```
register myudfs.jar;
```

Step 6: Run Pig codes(written in java) that uses the UDF.

At the beginning, we try to develop both the UDF and the pig codes which invokes the UDF in Java. An example is shown as below:

```
import java.io.IOException;
import org.apache.pig.PigServer;
public class pigTest2{
    public static void main(String[] args) {
        System.out.println("...1...");
        try {
            PigServer pigServer = new PigServer("mapreduce");
            runIdQuery(pigServer, "jing/passwd");
        }
        catch(Exception e) {
        }
    }
    public static void runIdQuery(PigServer pigServer, String inputFile) throws IOException {
        System.out.println("...2...");
        pigServer.registerQuery("A = load 'jing/passwd' using PigStorage(':');");
    }
}
```

```

pigServer.registerQuery("A = load " + inputFile + " using PigStorage(':');");
pigServer.registerQuery("B = foreach A generate $0 as id;");
pigServer.store("B", "jing/idoutb");
System.out.println("...3...");
pigServer.registerQuery("REGISTER myudfs.jar;");
System.out.println("...4...");
pigServer.registerQuery("C = foreach B generate myudfs.UPPER(id);");
pigServer.store("C", "jing/idoutc");
}
}

```

The result can be seen in Figure 33.

```

-terminated- pigTest2 [Java Application] /import/tr1raid6a1h1/projekt/hadoop/programs/jdk1.6.0_20/bin/java (May 23, 2010 11:36:07 AM)
...1...
10/05/23 11:36:07 INFO executionengine.HExecutionEngine: Connecting to hadoop file system at: file:///
10/05/23 11:36:07 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
...2...
10/05/23 11:36:08 INFO mapReduceLayer.MultiQueryOptimizer: MR plan size before optimization: 1
10/05/23 11:36:08 INFO mapReduceLayer.MultiQueryOptimizer: MR plan size after optimization: 1
10/05/23 11:36:10 INFO mapReduceLayer.JobControlCompiler: Setting up single store job
10/05/23 11:36:10 INFO jvm.JvmMetrics: Cannot initialize JVM Metrics with processName=JobTracker, sessionId=
10/05/23 11:36:10 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should i
10/05/23 11:36:11 INFO mapReduceLayer.MapReduceLauncher: 0% complete
10/05/23 11:36:11 INFO mapred.MapTask: numReduceTasks: 0
10/05/23 11:36:12 INFO mapred.TaskRunner: Task:attempt_local_0001_m_000000_0 is done. And is in the process of com
10/05/23 11:36:12 INFO mapred.LocalJobRunner:
10/05/23 11:36:12 INFO mapred.TaskRunner: Task attempt_local_0001_m_000000_0 is allowed to commit now
10/05/23 11:36:12 INFO mapred.FileOutputCommitter: Saved output of task 'attempt_local_0001_m_000000_0' to file:/i
10/05/23 11:36:12 INFO mapred.LocalJobRunner:
10/05/23 11:36:12 INFO mapred.TaskRunner: Task 'attempt_local_0001_m_000000_0' done.
10/05/23 11:36:16 INFO mapReduceLayer.MapReduceLauncher: 100% complete
10/05/23 11:36:16 INFO mapReduceLayer.MapReduceLauncher: Successfully stored result in: 'jing/idoutb'
10/05/23 11:36:16 INFO mapReduceLayer.MapReduceLauncher: Records written : 0
10/05/23 11:36:16 INFO mapReduceLayer.MapReduceLauncher: Bytes written : 0
10/05/23 11:36:16 INFO mapReduceLayer.MapReduceLauncher: Success!
...3...

```

Figure 33: Result of 'registerQuery'

As can be seen from the result, the command 'pigServer.registerQuery("REGISTER myudfs.jar;");' was just skipped when the codes were running. That is because jars can only be registered on the command line. It means that currently 'register' can only be done inside a Pig Latin script^[38]. So, the pig UDF can be written in Java, but if the user wants to invoke the PDF, he can only write Pig Latin script to invoke it.

Step 7: Run the Pig script which invokes the UDF.

An example is shown as below:

```

-- myscript.pig
hadoop fs -rm myudfs.jar
hadoop fs -copyFromLocal myudfs.jar jing

```



```

register myudfs.jar;

A = load 'jing/passwd' using PigStorage(':');
B = foreach A generate $0 as id;
C = foreach B generate myudfs.UPPER(id);

dump C;

```

The command below can be used to run the script.

```
java -cp pig.jar org.apache.pig.Main -x local myscript.pig
```

The original document we used can be seen in Figure 34.

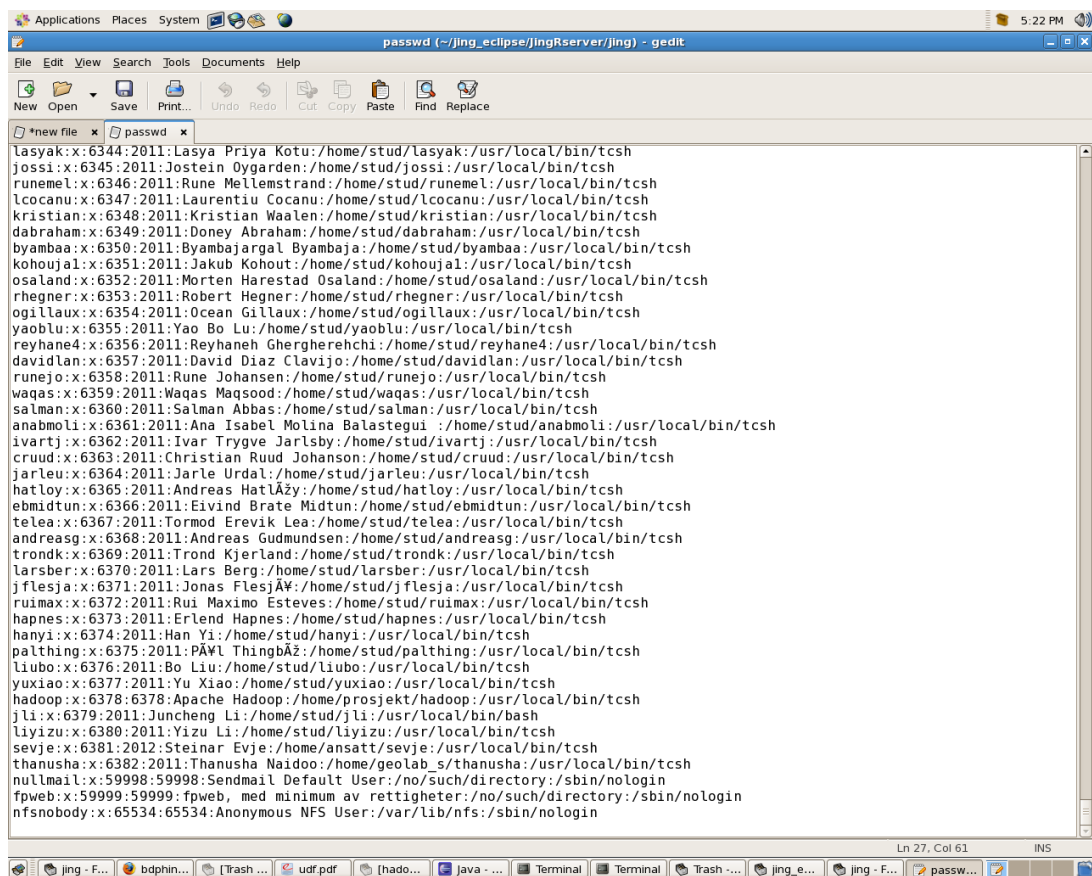


Figure 34: Original Document

The id results are shown in Figure 35.

```
cruud
jarleu
hatloy
ebmidtun
telea
andreasg
trondk
larsber
jflesja
ruimax
hapnes
hanyi
palthing
liubo
yuxiao
hadoop
jli
liyizu
sevje
thanusha
nullmail
fpweb
nfsnobody
```

Figure 35: ID Result

The results after invoking the UDF are shown in Figure 36.

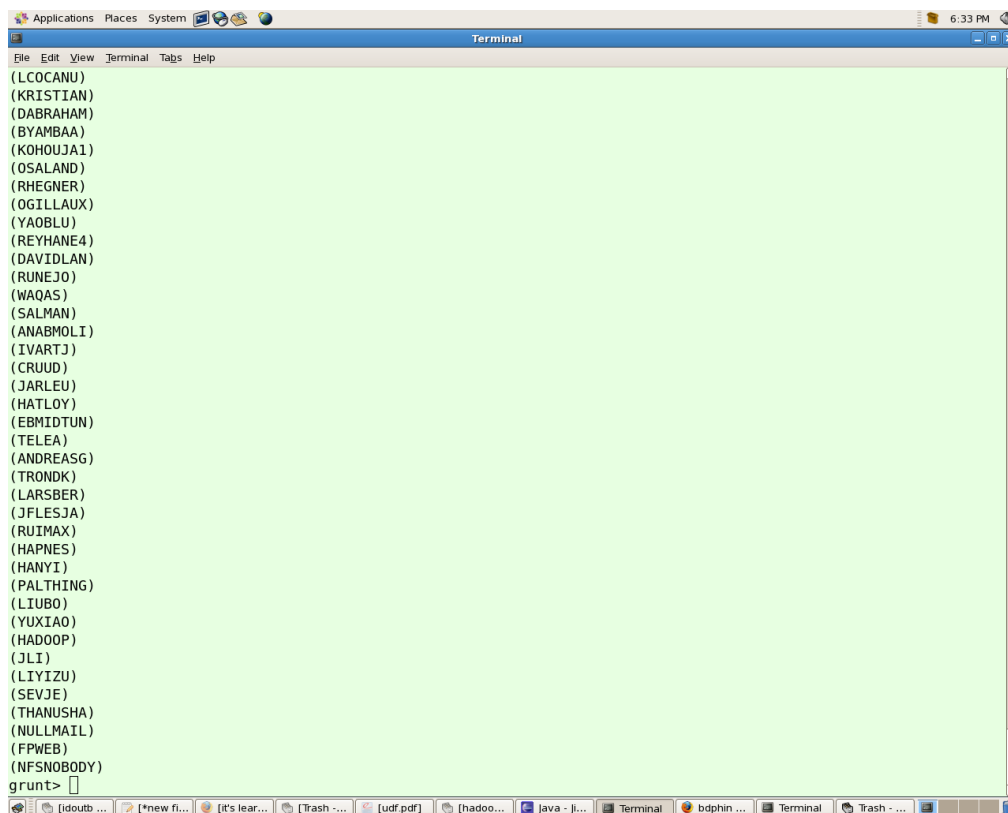


Figure 36: Final Result

Step 8: Pig UDF with Rserve

The relationship among Pig codes, UDF, Rserve and R is shown in Figure 37.

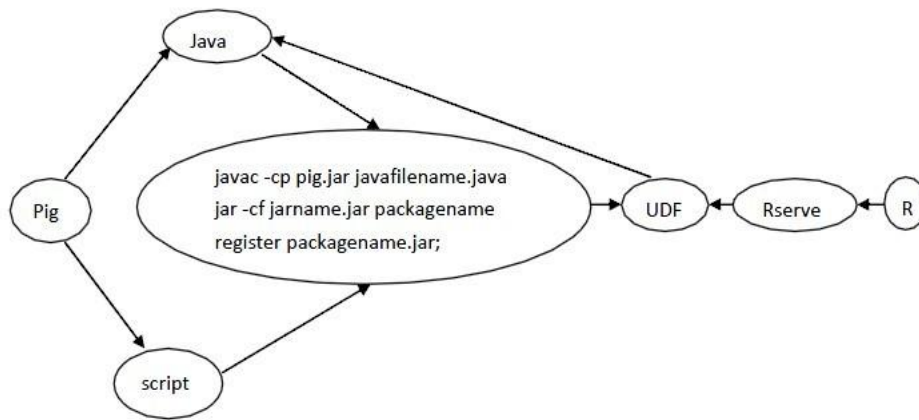


Figure 37: Relationship among Pig, UDF, Rserve and R

As can be seen from Figure 37, Pig codes can be written in Java and script, UDF can be written in JAVA, R functions can be put into UDF. If Pig codes want to invoke UDF, 3 important steps must be implemented (according to Pig UDF Manual):

- 1) `javac -cp pig.jar javafilename.java`; This command is used to compile UDF with `pig.jar`.
- 2) `jar -cf jarname.jar packagename`; This command is used to create a jar file that contains the UDF.
- 3) `register packagename.jar`; This command is used to register the jar file which include the UDF, so the pig codes who invoke this UDF will know where the UDF is.

If Pig codes want to invoke UDF, there are 2 ways, use Java or script.

Step 1, consider using java.

As shown from the former chapters, 'register packagename.jar;' can only be done inside a Pig Latin script. So, it means that the pig UDF can be written in Java, but if the user wants to invoke this PDF, he can only write Pig Latin script to invoke it. The result of step 1 can be seen in Figure 38.

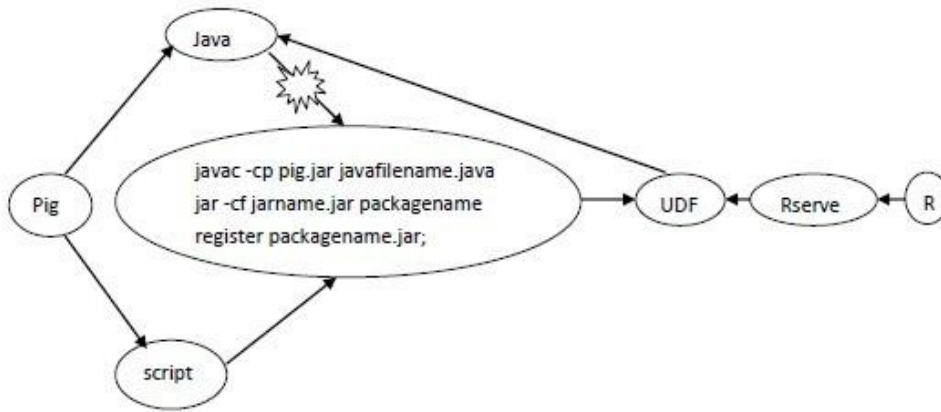


Figure 38: Result of Step 1

Step 2, consider using script.

R functions can be put into UDF (written in java), then the UDF is a function including R functions. If pig wants to use the UDF, the 3 requirements must be satisfied, now the problem comes up, when pig.jar try to compile the UDF, pig.jar cannot find Rserve engine, as can be seen from Figure 39.

```

    RConnection c = new RConnection("152.94.1.68");
    ^
/home/prosjekt/hadoop/jing_eclipse/JingRserver/src/myudfs/RTest2.java:29: cannot
find symbol
symbol  : class REXP
location: class myudfs.RTest2
    REXP x = c.eval("R.version.string");
    ^
8 errors
hadoop-pitter1:~/programs/jdk1.6.0_20$ bin/javac -cp /home/prosjekt/hadoop/jing_eclipse/JingRserver/src/myudfs/pig.jar /home/prosjekt/hadoop/jing_eclipse/JingRserver/src/myudfs/RTest2.java
/home/prosjekt/hadoop/jing_eclipse/JingRserver/src/myudfs/RTest2.java:14: package org.rosuda.REngine does not exist
import org.rosuda.REngine.REXP;
    ^
/home/prosjekt/hadoop/jing_eclipse/JingRserver/src/myudfs/RTest2.java:15: package org.rosuda.REngine.Rserve does not exist
import org.rosuda.REngine.Rserve.RConnection;
    ^
/home/prosjekt/hadoop/jing_eclipse/JingRserver/src/myudfs/RTest2.java:28: cannot find symbol
symbol  : class RConnection
location: class myudfs.RTest2
    RConnection c = new RConnection("152.94.1.68");
    ^
/home/prosjekt/hadoop/jing_eclipse/JingRserver/src/myudfs/RTest2.java:28: cannot find symbol
symbol  : class RConnection
location: class myudfs.RTest2
    RConnection c = new RConnection("152.94.1.68");
    ^
/home/prosjekt/hadoop/jing_eclipse/JingRserver/src/myudfs/RTest2.java:29: cannot find symbol
symbol  : class REXP
location: class myudfs.RTest2
    REXP x = c.eval("R.version.string");
    ^
5 errors
hadoop-pitter1:~/programs/jdk1.6.0_20$
  
```

Figure 39: Compile Error

The user has to use pig.jar to compile UDF, but if Rserve is included in the UDF, pig.jar doesn't know what Rserve is and cannot compile it. So, the next step is to figure out a way to compile the UDF.

In order to compile the UDF, 3 jars are needed: pig.jar, REngine.jar and RserveEngine.jar. First, we can change the classpath to show JVM how to find the jar files, but unfortunately it doesn't work. Second, instead of changing the classpath, we can move REngine.jar and RserveEngine.jar to the same folder which includes pig.jar, and use these 3 jars together to compile the java file, but the similar error happens, it still cannot find the jars. So, the result of step 2 can be seen as Figure 40.

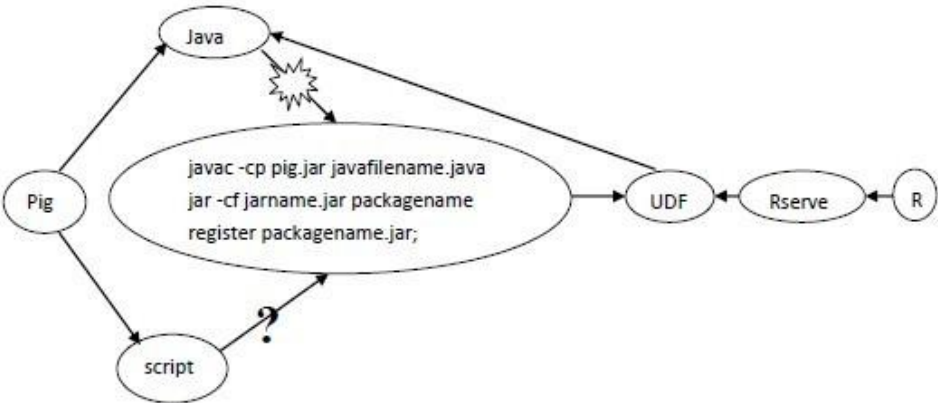


Figure 40: Result of Step 2

5.4 Conclusion of the Chapter

Due to the reason of researching time and the author's knowledge, this reasoning technique has limitations in the model, R functions can be embedded into pig but it's very limiting, so future work needs to be done on it.

6 Conclusion and Future Work

6.1 Conclusion

For oil companies, on one hand, operators want to know whether everything is going ok or not; on the other hand, it may direct operators to find new resources. So, it is very important to process the data before analysis.

This paper gives out several models to show how to process data with reasoning techniques. The alarm system which integrates JADE and JESS can process the data and give out alarms when the data is abnormal; The SUP data processing system which based on Petri Net technique and the JESS monitoring results is developed as well and the performance of the system is also analysed; The first 2 models are only used for a single server, however, in order to process data with multi servers in a distributed environment, the author also does a research on the interface between Rserve and Pig.

6.2 Future Work

There are some more reasoning techniques available to process data, Bayesian Network, for example. More research on other reasoning techniques can be done in the future.

Due to the reason of the researching time and the author's knowledge, the Pig UDF with Rserve is not completely finished, more work can be done in the future.

7 References

- [1] Ernest Friedman-Hill. Jess in Action---Rule-Based Systems in Java [M]. 2009
- [2] <http://clipsrules.sourceforge.net/>
- [3] Carl Adam Petri and Wolfgang Reisig (2008) Petri net
- [4] GPenSIM, version 3 - General Purpose Petri Net Simulator, Reggie Davidrajuh, 2008
- [5] Discrete Simulation and Performance Analysis, Reggie Davidrajuh, 2009
- [6] <http://www.r-project.org/> R
- [7] <http://www.rforge.net/Rserve/index.html>
- [8] Bayesian Networks, Ben-Gal, Irad (2007)
- [9] <http://JADE.tilab.com>
- [10] Data Querying and Transformation Application Based on Witsml Connector, Baodong Jia, June 22th, 2009
- [11] Grigoris Antoniou, Frank van Harmelen. A Semantic Web Primer[M]. 2008
- [12] G. Klyne, J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax[M], 2004
- [13] A. Berglund. Extensible Stylesheet Language (XSL) [M]. 2006
- [14] Alarm System, Jing Kou, 2009
- [15] <http://www.cse.sys.t.u-tokyo.ac.jp/furuta/teaching/csd/CSD04.pdf>
- [16] <http://www.dia.fi.upm.es/~phernan/AgentesInteligentes/referencias/bellifemine01.pdf>
- [17] <http://ajava.org/course/open/13966.html>
- [18] <http://www.cs.mu.oz.au/682/Week6b.ppt>
- [19] http://jade.tilab.com/doc/tutorials/jade-jess/jade_jess.html
- [20] Developing Intelligent Agent Applications with JADE and JESS, Bala M. Balachandran, University of Canberra
- [21] <http://www.petrinets.info/>
- [22] <http://petri.net/>
- [23] GPenSIM - General Purpose Petri Net Simulator, Reggie Davidrajuh, 2009
- [24] <http://www.mathworks.com/products/matlab/>
- [25] <http://www.math.utah.edu/lab/ms/matlab/matlab.html>
- [26] Notes Publishing System and Its Extension, Baodong Jia and Jing Kou, 2009
- [27] <http://tech.163.com/special/00091SVT/ajax.html>
- [28] <http://cran.r-project.org/manuals.html>
- [29] <http://cran.r-project.org/doc/Rnews/>

[30] Using R for Data Analysis and Graphics - Introduction, Examples and Commentary

[31] <http://rbbs.biosino.org/Rbbs/forums/list.page>

[32] <http://hadoop.apache.org/pig/>

[33] http://hadoop.apache.org/pig/docs/r0.5.0/piglatin_users.html

[34] <http://hadoop.apache.org/pig/docs/r0.5.0/setup.html>

[35] Pig UDF Manual

[36] <http://rosuda.org/Rserve/>

[37]

<http://www.unibielefeld.de/biologie/Oekosystembiologie/bio7app/documents/javadoc/com/eco/bio7/rbridge/RServe.html>

[38] <https://issues.apache.org/jira/browse/PIG-1226>

8 Appendix

8.1 Appendix A- Example of Farmer's Dilemma Problem

```
;;;=====
;;; Farmer's Dilemma Problem
;;;=====
```

```
...*****
;;;
;;;* TEMPLATES *
...*****
;;;
```

```
;;; The status facts hold the state
;;; information of the search tree.
```

```
(deftemplate MAIN::status
  (slot search-depth)
  (slot parent)
  (slot farmer-location)
  (slot fox-location)
  (slot goat-location)
  (slot cabbage-location)
  (slot last-move))
```

```
...*****
;;;
;;;* INITIAL STATE *
...*****
;;;
```

```
(defacts MAIN::initial-positions
  (status (search-depth 1)
    (parent no-parent)
    (farmer-location shore-1)
    (fox-location shore-1)
    (goat-location shore-1))
```

```
(cabbage-location shore-1)
(last-move no-move)))
```

```
(deffacts MAIN::opposites
  (opposite-of shore-1 shore-2)
  (opposite-of shore-2 shore-1))
```

```
...*****
;;;
;;;* GENERATE PATH RULES *
...*****
```

```
(defrule MAIN::move-alone
  ?node <- (status (search-depth ?num)
                  (farmer-location ?fs))
  (opposite-of ?fs ?ns)
  =>
  (duplicate ?node (search-depth (+ 1 ?num))
             (parent ?node)
             (farmer-location ?ns)
             (last-move alone)))
```

```
(defrule MAIN::move-with-fox
  ?node <- (status (search-depth ?num)
                  (farmer-location ?fs)
                  (fox-location ?fs))
  (opposite-of ?fs ?ns)
  =>
  (duplicate ?node (search-depth (+ 1 ?num))
             (parent ?node)
             (farmer-location ?ns)
             (fox-location ?ns)
             (last-move fox)))
```

```
(defrule MAIN::move-with-goat
```

```

?node <- (status (search-depth ?num)
             (farmer-location ?fs)
             (goat-location ?fs))
(opposite-of ?fs ?ns)
=>
(duplicate ?node (search-depth (+ 1 ?num))
            (parent ?node)
            (farmer-location ?ns)
            (goat-location ?ns)
            (last-move goat)))

```

```

(defrule MAIN::move-with-cabbage
  ?node <- (status (search-depth ?num)
                 (farmer-location ?fs)
                 (cabbage-location ?fs))
  (opposite-of ?fs ?ns)
=>
(duplicate ?node (search-depth (+ 1 ?num))
            (parent ?node)
            (farmer-location ?ns)
            (cabbage-location ?ns)
            (last-move cabbage)))

```

```

...*****
;;;
;;;* CONSTRAINT VIOLATION RULES *
...*****
;;;

```

```

(defmodule CONSTRAINTS)

(defrule CONSTRAINTS::fox-eats-goat
  (declare (auto-focus TRUE))
  ?node <- (status (farmer-location ?s1)
                 (fox-location ?s2&~?s1)
                 (goat-location ?s2))

```

=>

(retract ?node))

(defrule CONSTRAINTS::goat-eats-cabbage

(declare (auto-focus TRUE))

?node <- (status (farmer-location ?s1)

(goat-location ?s2&~?s1)

(cabbage-location ?s2))

=>

(retract ?node))

(defrule CONSTRAINTS::circular-path

(declare (auto-focus TRUE))

(status (search-depth ?sd1)

(farmer-location ?fs)

(fox-location ?xs)

(goat-location ?gs)

(cabbage-location ?cs))

?node <- (status (search-depth ?sd2&:(< ?sd1 ?sd2))

(farmer-location ?fs)

(fox-location ?xs)

(goat-location ?gs)

(cabbage-location ?cs))

=>

(retract ?node))

...*****

;;;* FIND AND PRINT SOLUTION RULES *

...*****

(defmodule SOLUTION)

(deftemplate SOLUTION::moves

(slot id)

```
(multislot moves-list))
```

```
(defrule SOLUTION::recognize-solution
```

```
(declare (auto-focus TRUE))
```

```
?node <- (status (parent ?parent)
```

```
(farmer-location shore-2)
```

```
(fox-location shore-2)
```

```
(goat-location shore-2)
```

```
(cabbage-location shore-2)
```

```
(last-move ?move))
```

```
=>
```

```
(retract ?node)
```

```
(assert (moves (id ?parent) (moves-list ?move))))))
```

```
(defrule SOLUTION::further-solution
```

```
?node <- (status (parent ?parent)
```

```
(last-move ?move))
```

```
?mv <- (moves (id ?node) (moves-list $?rest))
```

```
=>
```

```
(modify ?mv (id ?parent) (moves-list ?move ?rest)))
```

```
(defrule SOLUTION::print-solution
```

```
?mv <- (moves (id no-parent) (moves-list no-move $?m))
```

```
=>
```

```
(retract ?mv)
```

```
(printout t crlf 'Solution found: ' crlf crlf)
```

```
(bind ?length (length$ ?m))
```

```
(bind ?i 1)
```

```
(bind ?shore shore-2)
```

```
(while (<= ?i ?length)
```

```
(bind ?thing (nth$ ?i ?m))
```

```
(if (eq ?thing alone)
```

```
then (printout t 'Farmer moves alone to ' ?shore ' ' crlf)
```

```
else (printout t 'Farmer moves with ' ?thing ' to ' ?shore ' ' crlf))
```

```

(if (eq ?shore shore-1)
  then (bind ?shore shore-2)
  else (bind ?shore shore-1))
(bind ?i (+ 1 ?i)))
(reset)
(run)

```

8.2 Appendix B- Example of Norwegian Traffic Lights

```

function [PN_name, set_of_places, set_of_trans, set_of_arcs]...
  = NO_light_def(global_info)
% file: pn_def.w:
% definition of petri net graph for Norwegian traffic lights

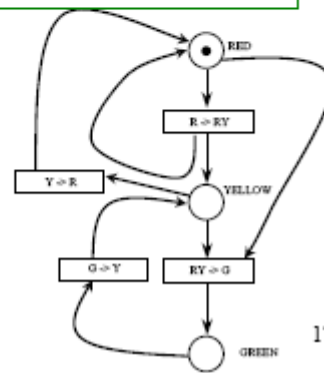
PN_name='Pet Net graph for traffic light (NOR)';
set_of_places={'RED', 'YELLOW', 'GREEN'};

set_of_trans={'CR_RY','CR_Y','CR_G','CR_R'};

set_of_arcs={'RED','CR_RY',1, 'CR_RY','RED',1, 'CR_RY','YELLOW',1,...
  'RED','CR_Y',1, 'YELLOW','CR_Y',1, 'CR_Y','GREEN',1,...
  'GREEN','CR_G',1, 'CR_G','YELLOW',1,...
  'YELLOW','CR_R',1, 'CR_R','RED',1};

```

Petri Net Definition File



17

Main Simulation File

```
% the main file to run simulation
% for the system given in figure-5
clear, clc;
pn = petrinetgraph('NO_light_def');
dynamic_info.initial_markings = {'RED', 1};

Results = gpsim(pn, dynamic_info);
printsigs(pn, Results);
```

Transition Definition File-1 (tR_RY.m)

Transition tR->RY will fire only if there is a token in place **RED** and there is no token in place **YELLOW**.

```
function [fire, global_info] = tR_RY_def(PN, global_info)
% function fire = tR_RY_def(PN)

pR = get_place(PN, 'RED');
pY = get_place(PN, 'YELLOW');

fire = (pR.tokens) & not(pY.tokens);
```