Siamak Varghaei

A Multi-Agent Approach for Distribution System Restoration

Bachelor’s Thesis August 2009
Siamak Varghaei

A Multi-Agent Approach for Distribution System Restoration

Bachelor’s Thesis, August 2009
A Multi-Agent Approach for Distribution System Restoration

Author(s):
Siamak Varghaei

Supervisor(s):
Professor Morten Lind
Ph.d. Arshad Saleem

Department of Electrical Engineering
Centre for Electric Technology (CET)
Technical University of Denmark
Elektrovej 325
DK-2800 Kgs. Lyngby
Denmark

www.elektro.dtu.dk/cet
Tel: (+45) 45 25 35 00
Fax: (+45) 45 88 61 11
E-mail: cet@elektro.dtu.dk

Release date: 13-08-2009
Class: 1 (offentlig)
Edition: 1st edition
Comments: This report is a part of the requirements to achieve the Bachelor’s Degree at the Technical University of Denmark.
The report represents 20 ECTS points.

Rights: © Siamak Varghaei, 2009
This project was completed during spring semester 2009. The project is about the application of a multi agent system (MAS) to restore a distributed power system. The part of restoration of power in the three bus model has been considered.

The multi agent system developed in JADE, a software platform which has the ability to make agents and act on the agent’s behaviour. The JADE platform has been complied in Eclipse.

A three bus network model has been used to make some simulations to work with the MAS. The model has been simulated in Simulink. The connection between the physical model and JADE was considered. Communication between Matlab and JADE has been created with Matlab Embedded Function, a programmable function.

A restoration strategy based on agent technology has been used to propose a method. In this project tested one scenario as fault scenario, and the MAS has been tested successfully in one fault scenario.
# TABLE OF CONTENT

Abstract ........................................................................................................................................ 3  
List of figures.................................................................................................................................. 9  
List of tables .................................................................................................................................. 11  

1 Introduction........................................................................................................... 15  
  1.1 Background........................................................................................................... 15  
  1.2 Problem formulation.......................................................................................... 15  
  1.3 Personal motivations......................................................................................... 16  
  1.4 Method, limitation and background................................................................ 16  
  1.5 Related work..................................................................................................... 16  
  1.6 How to read the report .................................................................................... 17  

2 The jade platform ................................................................................................. 18  
  2.1 Overview.......................................................................................................... 18  
  2.2 Jade and Agents ............................................................................................... 18  
  2.3 Compiling and Launching JADE.................................................................... 20  
  2.4 Package ............................................................................................................ 21  
  2.5 Message Transport Service ............................................................................ 22  

3 Multi Agent systems ............................................................................................. 25  
  3.1 Overview.......................................................................................................... 25  
  3.2 Basic concept of Agents.................................................................................. 25  
  3.3 The Foundation for Intelligent Physical Agents (FIPA)............................... 26  
  3.4 Creating Agents ............................................................................................... 29  
  3.5 Agent communication....................................................................................... 30  

4 why MAS? ............................................................................................................. 33  
  4.1 Overview.......................................................................................................... 33  
  4.2 MAS Applications in electric power systems............................................. 33  
  4.3 Ontologies......................................................................................................... 36  
  4.4 Restoration Structure ....................................................................................... 36  

5 Reconfiguration using mas .................................................................................. 41  
  5.1 Overview.......................................................................................................... 41  
  5.2 Types of agents in the MAS ........................................................................... 41
Table of content

5.3 Interaction between Agents to reconfigure an Electric Power Distributed System

6 Simulation model
   6.1 Overview
   6.2 Electric Power Distribution System simulation model
   6.3 Communication Middleware

7 Simulation results
   7.1 Fault scenario 1

8 Conclusion
   8.1 Results
   8.2 Perspectives
   8.3 Further work

References

A Matlab simulink models

B Matlab script

C Feederagent1 code

8
LIST OF FIGURES

Figure 2-1: JADE platform ........................................................................................................ 19
Figure 2-2: Relationship between the main architectural elements ........................................ 20
Figure 2-3: Standard output at JADE start-up ........................................................................ 21
Figure 3-1: Description of the agent management ontology .................................................. 27
Figure 4-2: Network diagram with location of agents shown ............................................... 35
Figure 4-3: Diagram of the Finite State Machine ................................................................... 37
Figure 4-4: Flowchart of all states .......................................................................................... 40
Figure 5-1: Architecture of the multiagent system ............................................................... 41
Figure 5-2: The FIPA Cancel Meta-Protocol ........................................................................ 43
Figure 5-3: The idle three bus model ...................................................................................... 44
Figure 5-4: A blackout load occurs as a fault ........................................................................ 44
Figure 5-5: FAG1 ask the FAG2 through SAB which is connected ..................................... 45
Figure 5-6: LAG2 will be restored, switch S1 will be closed ................................................. 45
Figure 5-7: LAG4 will be restored; switches S1 and SAC are closed ..................................... 46
Figure 6-1: An overview of simulink model and JADE software .......................................... 48
Figure 6-2: Three bus modeled in PowerFactory ................................................................. 49
Figure 6-3: Power on Feeder1 .............................................................................................. 50
Figure 7-1: Simulation result for power on Feeder1 ............................................................. 54
LIST OF TABLES

Table 3-1: ACL messages parameters ................................................................. 28
Table 3-2: Foundation for Intelligent, Physical Agents (FIPA) ......................... 29
1 INTRODUCTION

1.1 Background

When an electric power supply is caused by a fault, it is necessary to restore the power system to an optimal target configuration after the fault. The problem of approaching a target system is referred to as a power system restoration.

A multiagent system technology is developed for some kind of application including power system restoration. The uptake of this technology has increased over the few years in terms of so many research projects.

MAS technology offers a high degree of scalability, independence of network topology and flexibility because they work autonomously and make local decisions. MAS’s are composed of multiple interacting computing elements, called agents.

In this project focuses on the application of MultiAgent System (MAS) to develop a reconfigurable Electric Power Distributed System. In this project, the focus on a multiagent system to restore a power distributed system will be treated. The purpose is to test and investigate a multiagent system to find and restore a fault that will be occurred in a power system.

1.2 Problem formulation

- What benefits can a multiagent system offer, and what are options for applying a multiagent system in a power distributed system?
- Which method or strategy should be chosen to restore the power and how should they be designed?
- How the power should distributed system and a multiagent system work together?
1.3 **Personal motivations**

It was interesting for me to investigate the application of an agent technology for a distributed power system. My personal motivation to do this project has been an interest to find a method in a multiagent system to restore power. Working with simulink and how it connects with another software as JADE, how the agents in multiagent system exchanging information was a way to know much more about the reel power distributed network.

1.4 **Method, limitation and background**

The purpose of this project is to develop a multiagent system for a power distributed system to restore a fault that will be happened in a physical model. The multiagent system running on JADE, and all the codes that helps the agents to interact with each other will be complied on a software called Eclipse.

The physical power distributed model has been simulated in Matlab under Simulink option with help of some toolbox. There is also has been complied a matlab function that helps to make a TCP/IP connection that it is useful to transfer message with JADE.

Connection between JADE and Simulink has been operated successfully and it was successful to send/receive data from JADE to Simulink or conversely.

1.5 **Related work**

The method that has been used in this project is done by Nagata. The work on this project presents the idea that comes from presented [4].

In 2003, the IntelliTEAM II Automation Restoration System was introduced by S&C electric company [5]. IntelliTEAM uses distributed intelligence and peer-peer communication to find and isolate a fault on line section and then restore the power. IntelliTEAM can automate a part of circuit in a distribution system which consists of some components and based on agent technology. There are some teams that each team has responsibility for negotiation with another team in order to power restoration. The communication between team is kind of peer-peer communication and it is done through radio and fiber optics [5]. This system is installed on some networks and works successfully but it doesn’t support FIPA specifications.
1.6 **How to read the report**

This rapport consists of eight chapters. In chapter 2 an overview of the JADE platform is given. In chapter 3 a representation of multiagent system has been described. This chapter gives the reader knowledge about the agents and interaction between them. Chapter 4 presents the reconfiguration of power. In chapter 5 represents the method that has been used in this project, and the fault scenario will be shown by some figures. Chapter 6 represents the simulink model and how the model can make a connection to the JADE. In chapter 7 one fault scenario will be described and the results will be shown as figures.
THE JADE PLATFORM

2.1 Overview

This chapter represents an overview of the JADE platform. Definition of agent multi-agent system and negotiation between agents will be described. Building and compiling agents will be described.

2.2 Jade and Agents

JADE (Java Agent Development Framework) is a software framework implemented in Java language [3]. JADE framework implements multi agent-based systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools. The first JADE platform was developed by Telecom Italia in 1998 by the need to early FIPA specifications. FIPA specifications will be discussed in chapter 3. With the help of European Commission’s team, JADE opened source in 2000 by Telecom Italia by LGPL license [1]. This license gives users rights to copy the software and distribute the copy, have access to the source code and gives rights to users to change also the code.

Users can find software, example code and documentations in JADE website http://jade.tilab.com. It is interesting that, when JADE first was made public by Telecom Italia, it was used only with FIPA community but as the software grew up, users became a globally distributed developer community. One of the most essential duties of JADE is, implementing agents over an object-oriented language, Java, making Application Programming Interface (API).

Graphical User Interface will be used to monitoring and controlling an agent on JADE platform. The following figure shows a view of JADE platform.
A JADE platform holds containers inside itself. Main container is the first container that will be launched, and all the others containers must register with the main container. Others containers will be called ‘Container-1’, ‘Container-1’, etc. The main container has some duties as:

- Managing the container table (CT).
- Managing global agent descriptor table (GADT).
- Hosting the AMS and the DF.

Figure 2-1 shows the GUI of the Remote Monitoring Agent (RMA). This GUI will be appeared with typing -gui. There are also other agents on GUI, as, Agent Management System (ams) and Directory Facilitator (df).

Every agent has a name and address that will be separated with @ sign. The agent RMA@IBM:1099/JADE is an agent which its name is RMA and its address is IBM:1099/JADE. Each address contains a host and a port number. In this case the host is IBM and port number is 1099.

Figure 2-2 shows the relationship between the main architectural of the JADE.
The jade platform

Figure 2-2: Relationship between the main architectural elements

The main container has some special responsibility as follows:
Managing the container table (CT) is the registry of all container and all addresses at the platform.
Managing the global agent description table (GADT) doing register all the agents status and location.
Hosting AMS and the DF makes the agent management and white page service and the default yellow page service of the platform.

2.3 Compiling and Launching JADE

JADE software is divided into two parts, the main distribution and the add-ons. The add-ons include self-contained modules which are developed by the open source community. The main distribution consists of five files:

- jadeBin.zip
- jadeDoc.zip
- jadeExamples.zip
- jadeSrc.zip
- jadeAll.zip

Each of these files contains some files that will be useful for programmers. In this case, launching JADE, the jade/lib folder is useful. The first step that a user should to do to launch the platform, setting local Java CLASSPATH, the set of directories and Java archive
that JVM will look for byte code. To do this if JADE folders are under drive c:\, will the set scripts looks like the following command:

```
prompt> set JADE_HOME=c:\jade
prompt> set CLASSPATH=%JADE_HOME%\lib\jade.jar; %JADE_HOME%\lib\jadeTools.jar; %JADE_HOME%\lib\http.jar; %JADE_HOME%\lib\iiop.jar; %JADE_HOME%\lib\commons-codec\commons-codec-1.3.jar;%JADE_HOME%\classes
```

To launch the platform the following command will be used:

```
prompt> java jade.Boot -gui
```

After writing this command on Windows cmd, the JADE platform Figure 2-1 is going to show up, and some information as output about the agents, containers, host and port number will be showed the Windows cmd screen as Figure 2-3 shows.

![Figure 2-3: Standard output at JADE start-up](image)

### 2.4 Package

Each package contains some classes and interfaces that they implement functions in JADE. Some of the main packages will be show here:

- jade.core
- jade.content
- jade.domain
- jade.gui
- jade.imtp
- jade.lang.acl
- jade.mtp
- jade.proto
2.5  Message Transport Service

One the most essential services of JADE platform are Transport Message Service (TMS) that is essential for an agent platform. TMS makes management within and between platforms. In the following part will be discussed messages exchanging within and between platforms.

2.5.1 Message Transport Protocols

To start interoperability between JADE platforms, JADE will implement all the Message Transport Protocols (MTPs) which is defined by FIPA. Each MTP contains the definition of a transport protocol and a standard encoding of the message envelope. JADE will always start a HTTP with the initialization of a main container. This makes a server socket on the main container host and listens for incoming messages at the URL (http://hjem:7778/acc). A standard output when launching a main container will be looks like the following text:

INFO: MTP addresses:
http://hjem:7778/acc

2.5.2 IMTP

The JADE Internal Message Transport Protocol (IMTP) will be used for exchanging messages between agents in different containers of the same platform. Difference between IMPT and MPT can be explained as:

- IMTP can be used only for internal communications and it doesn’t need to be compiled with any FIPA standards.
- When messages need to manage the distributed platform, IMTP will be useful.

There are two main implementations available on IMTP. The first one is RMI and it is a default option based on Java. The second one using TCP sockets based on a private protocol. Both implementations allow fine-tuning of the IMTP for a network and devices.
3
MULTI AGENT SYSTEMS

3.1 Overview

This chapter represents basic definition of Multi Agent System (MAS). A MAS system is a collection of corporation’s agent. Agent-Oriented Programming (AOP) is a new software paradigm that takes concepts of artificial intelligence to the mainstream of distributed systems.

3.2 Basic concept of Agents

There is no single definition of an agent but all the definitions are agreed that an agent is a special software component that has autonomy, which provides an interoperable interface to an arbitrary system and/or behaves like a human agent, working for some clients in pursuit of its own agenda. The following definition is from (Wooldridge 2002) which in turn is adapted from (Wooldridge and Jennings 1995):

“An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives” [1].

Expression agent has found its own way into a number of technologies and has been widely used, for example, in artificial, database, operating systems and computer networks literature [3].

A multi-agent system can make complex systems. Agents in this system can interact with each other indirectly, by acting on the environment, or directly, by communication and negotiation.

An agent has some general characteristics as: autonomous, social, reactive, proactive.

- **Autonomy.** That agents are autonomous means they are independent and they make theirs own decision. No humans and no others has control over theirs action and behaviour.
Multi Agent systems

- **Reactivity.** An agent is reactive means that the agent is able to act to changes in its environment in timely fashion and make some decision based on those changes.

- **Proactiveness.** The capacity to exhibit a goal-directed behaviour by taking the initiative for advancing towards an objective.

- **Social ability.** An agent is social means that an agent can make contact with others agents as they can send message to each other. Agent interaction can be formed in terms of performative such as:
  - **Request.** The receiver will perform the message from the sender as the sender wants.
  - **Inform.** The receiver will be aware of a fact that the sender wants it.
  - **Propose or CFP.** The sender wants to make a negotiation with the receiver.

### 3.3 The Foundation for Intelligent Physical Agents (FIPA)

The Foundation for Intelligent Physical Agents (FIPA) is an IEEE computer Society standards organization. FIPA was accepted by IEEE as its eleventh standards committee on Jun 2005 [1].

FIPA specifications represent a collection of standards which are intended to promote the interoperation of heterogeneous agents and the services that they can represented.

JADE framework integrates tightly with the FIPA agent specifications. FIPA specification describes a safe environment for agents and the language of communication between them.

The complete set of specifications can be viewed in terms of different categories:

- Agent communication
- Agent transport
- Agent management
- Abstract architecture and application

Agent communication is the core category at the heart of the FIPA multi-agent system model.
3.3.1 Agent Management

Agent management is a normative framework within which FIPA agents can exist, operate and be managed. The agent management contains some components as depicted in Figure 3-4 and will be described in the following.

![Figure 3-1: Description of the agent management ontology](image)

- **Agent Platform (AP)** provides the physical infrastructure in which agents can be deployed. AP consists of machines, operating systems, FIPA agent management components and any additional support software.
- **Agent** is a basic part on an agent system, and it must have an owner and must support one notion of identity which using the FIPA agent identifier (AID).
- **Directory Facilitator (DF)** provides yellow pages services to other agents, and it is a mandatory component of the AP. Any agents may register their services on DF or ask DF to show what services the agents are offered by other agents.
- **Agent Management System (AMS)** has the responsibility for creation or deletion of agents and overseeing the moving an agent to AP or from AP. All agents must be registered with AMS to get an AID.
- **Message Transport Service (MTS)** is a transport service provided by an AP to transport the FIPA ACL message between agents on an AP and between agents on different APs.

3.3.2 FIPA Agent Communication Language

Communication is one of the most important tasks in MAS between agents because it enables them to exchanging messages and sharing information for obtains their goals. There are two most popular agent languages, Knowledge Query Manipulation Language
Multi Agent systems

(KQML) and FIPA Agent Communication Language (FIPA ACL). The most users are using agent communication language ACL, which incorporates many aspects of KQML.

The FIPA ACL is based in speech act theory which says that: messages represent actions or communicative acts as they do some action by the ability of being sent. The FIPA ACL message parameters are shown in Table 3-1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>performative</td>
<td>Type of communicative act of the messages</td>
</tr>
<tr>
<td>sender</td>
<td>Identity of the sender</td>
</tr>
<tr>
<td>receiver</td>
<td>Identity of the intended recipients</td>
</tr>
<tr>
<td>reply-to</td>
<td>Participant in communication</td>
</tr>
<tr>
<td>content</td>
<td>Content of the message</td>
</tr>
<tr>
<td>language</td>
<td>Expression of the content</td>
</tr>
<tr>
<td>encoding</td>
<td>Expression of the content</td>
</tr>
<tr>
<td>ontology</td>
<td>Expression of the content</td>
</tr>
<tr>
<td>protocol</td>
<td>Structure a conversation</td>
</tr>
<tr>
<td>conversation-id</td>
<td>Control of conversation</td>
</tr>
<tr>
<td>reply-with</td>
<td>Control of conversation</td>
</tr>
<tr>
<td>in-reply-to</td>
<td>Control of conversation</td>
</tr>
<tr>
<td>reply-by</td>
<td>Control of conversation</td>
</tr>
</tbody>
</table>

Table 3-1: ACL messages parameters

Here is one example of inform performative. In this action Agent-A informs Agent-B the price offered during the “round-01” of an auction. The first part of this message shows the communicative act “inform”. The communicative act “inform”, is information passing according to table 3-2.

(inform
 :sender Agent-A
 :receiver Agent-B
 :reply-with round01
 :content (price (bid good02) 100)
 :language fipa-sl
 :ontology auction
 )
Table 3-2 shows all performative of the FIPA ACL.

**Table 3-2:** Foundation for Intelligent, Physical Agents (FIPA)

<table>
<thead>
<tr>
<th>Communicative Act</th>
<th>Information passing</th>
<th>Requesting Information</th>
<th>Negotiation</th>
<th>Action performing</th>
<th>Error handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept-proposal</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>agree</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>cancel</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>cf2</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>confirm</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>discard</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>failure</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>inform</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>inform-if</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>inform-ref</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>not-understood</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>propagate</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>propose</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>query-if</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>query-ref</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>refuse</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>reject-proposal</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>request</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>request-when</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>request-whenever</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>subscribe</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### 3.3.3 FIPA Interaction Prototype

An agent that wants to make a conversation with other agents follows two rules, initiator and responder. An initiator makes a request to an agent or more than one agent. When the request is received, the agent will inform or refuse the request.

As you can see in the previous example, a conversation of messages can be seen as a sequence of performative acts such as REQUEST, INFORM or REFUSE.

### 3.4 Creating Agents

For at make an agent, programmer should define a class which it extends the `jade.core.Agent` class and `setup()` will be implemented. Typical operations that an agent shows in `setup()` method is:

- Show a GUI
- Open a connection to a database
- Register the services in the yellow page
- Start the basics behaviour

Here will be showing a simple agent of HelloWorldAgent class:

```java
import jade.core.Agent;
```
This code makes an agent which its name is HelloWorldAgent. After running the agent, a message will be shown on the output that will say: **Hello World. I’m an agent**

### 3.5 Agent communication

Agent communication is the one of the most fundamental features of JADE, and it implemented in FIPA specifications. The communication paradigm is based in asynchronous message passing, which means every agent has a mailbox where the JADE runtime posts messages sent by other agents.

Each message has the following structures:

- **Sender**
- **List of the receivers**
- **Performatives**, what a sender wants if it sends a message.
- **Content**, the information to be exchanged by the message.
- **Content language**, both sender and receiver have to be able to encode the messages.
- **Ontology**, both sender and receiver have to be able to understand the meaning of the messages symbol in communication.
- **Some extra options** that helps to control the communications and specify timeouts for receiving a reply.

#### 3.5.1 Sending a message

To send a message to another agent, the ACLMessage field should be fill out with the receiver, the language and the content of the message, and at the end of the code put the `send()` method which is sending message to the receiver. The following simple code, send a message to Agent2 with content “Hello from Agent1”:

```java
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID("Agent2", AID.ISLOCALNAME));
msg.setContent("Hello from Agent1");
send(msg);
```

#### 3.5.2 Receiving message

The JADE run-time posts a message to the receiver into the receiver’s queue as soon as it arrives. The receiver will get the message from its message queue with `receive()`
method. The following code is an example of $\text{receive()}$ method which it returns the first message in the message queue (if there is a message), and $\text{null}$ if there isn’t any message in message queue.

```java
ACLMessage msg = receive();
if (msg != null) {
}
```
4 WHY MAS?

4.1 Overview
In a power distribution system when an electric power supply interruption is caused by a fault, the system should restore the power quickly to an optimal target configuration after the fault. The problem of obtaining a target configuration is called as a power system restoration. Various approaches have so far been proposed to obtain the target configuration. These approaches can be classified into four categories [10]:

1. Heuristics
2. Experts Systems (ESs)
3. Mathematical programming (MP)
4. Soft Computing

Each of these have theirs ability to solve a problem, and of course with some weakness. Heuristics and ESs have been used in industries but they have their own lacks to the optimality to the solution. MP is good to obtain the optimal solution after the formulation, but it needs some engineering judgment in formulating restoration problems. Soft computing methods are easy to implement, but they need long computation time until solution. In this project the MAS and all the agents have been designed on the work published by Nagata [10] used in heuristically method.

4.2 MAS Applications in electric power systems
A multiagent system is a collection of collaborating intelligent and autonomous computational entities called agent [10]. The flexibility and adaptability of multiagent system make them attractive for several real world applications. Many important computer applications such as planning, process control, and communication network configurations and concurrent systems will be benefited from a multiagent system approach. Multiagent systems have been applied for solving problems such as reconfigurations, restorations, fault identifications, diagnosis and power system protection.
why MAS?

Most of the multiagent systems for power system reconfiguration and restoration use an agent to represent an electric component such as loads, switches. A major problem with such solution is the limited applications especially for large scale power systems with hundreds of component. Therefore in a distribution system with a large number of pieces of equipment a system that operates more efficiently and autonomously is required.

In order to make processing efficient the system is implemented as a “two-layer multi-agent system” consisting of several “feeder agents” in the upper layer and a group of “load agents” in the lower layer. This reduces the communication load and the load on the “feeder agents”. In concrete terms, using a two-layer system allows for a balance between local optimization for the lower-level agents and a large-scale optimization for the upper-level agents. On the other hand the blackout load in a distributed system can be efficiently minimized while each agent operates autonomously based on local information.

Figure 4-5 shows the standard configuration of a distribution system used in this project. In this system, the power is provided to the three feeders FA1, FA2 and FC3 from three power sources. Electric power is then provided to the load on the network via each feeder from each bus bar. The switches can be opened or closed when system makes a decision to restore the system.

The implemented system is designed and simulated in Matlab. In the simulated system will be measured current and voltage from each bus on each feeder. Theses measured current and voltage show a power that provided after each load. If power to a load is lost, the feeder agent will notify this problem and then make a decision. So there is talk about negotiating between each feeder agent and load agent. The information that is necessary to describe a load is load number, voltage level and power level.
Figure 4-2: Network diagram with location of agents shown
why MAS?

The “feeder agents” and the “load agents” each have simple restoration rules. Using the proposed method allows for efficient minimization of blackout load in a distributed system while allowing each agent to operate autonomously based on local information.

4.3 Ontologies

The ontology describes the concepts of a domain and the relationship between those concepts in a structure manner. Ontologies for use with the Java Agent Development Framework contain a class hierarchy of concepts, predicates, and agent actions. An ontology has been created in this project. The ontology’s purpose is to structure information transferred between LoadAgent and FeederAgent. The structure of this ontology is called the LoadAgentOntology.

Concept models the physical concepts such as: load, line section and switches. Predicates specify concept relationship, and can always be evaluated as true or false.

As example for predicate is $\text{OnLineSection}(\text{LineSection}, \text{Fault})$ which is used to discuss whether a fault occurred on the feeder. Location and type of a fault is important for the FeederAgent as regards to make any conclusion about how many of the LoadAgents on the feeder is affected by the fault.

Agents use the ontology for passing of information, formulating questions and requesting the execution of actions related to their specific domain. The LoadAgentOntology consists of two agent actions $\text{SwitchOn}$ and $\text{SwitchOff}$ that FeederAgent sending request to LoadAgent to connect or disconnect a switch.

4.4 Restoration Structure

Each FeederAgent have a behavior called **Finite State Machine (FSM) Behavior** that controlling the state of the FeederAgent. There are 6 different states in each Feeder-Agent which all of them will be discussed in the following.
Figure 4-3: Diagram of the Finite State Machine

**State 1**: the FeederAgent begins in idle state, waiting for any message from its Load-Agent LAG. If the message indicates power loss in one the load, the FA will store the pre-fault power value and the FA will add these individual pre-fault consumption values and go to next state, state 2. If a FA receives notifications from its LAG that power has returned it will stay in the same state, but register its services on the DF yellow pages.
why MAS?

If the FA has done any subscriptions with other FAs, it will cancel these and send request to the appropriate LAGs to disconnect the interconnecting switches to these feeders.

In this states the FA waits for any subscription cancellations from other FAs.

**State 2:** in this state a power negotiation initialized and goes to next state, state 3.

**State 3:** in this state the power that coming from state 2 will be evaluated, if no power has been obtained, the FA will return to the state 1, if the power has been obtained fully, it will go to state 5, and if the power has been obtained partly, it will go to state 4.

**State 4:** the FA will decide how many loads will be restored. In this state send a request to each LAGs to connect or disconnect its load.

**State 5:** the FA will decide which switch should be connected to get the power. The FA will request the one of its LAGs closest to the switch to connect it. If the action has been performed successfully, the FA returns to the state 1.

**State 6:** in this state the FA will send a switching action command to its LAG to perform the control action, and then goes to state 1, else stay in state 6.
why MAS?

**Figure 4-4:** Flowchart of all states
5
RECONFIGURATION USING MAS

5.1 Overview

The purpose of this chapter is to propose a framework for an “autonomous distributed restoration system” that reliably maintains a distributed system through the use of multiagents. The following distributed system, that will be described later, is implemented as two layers multi-agent. This multi-agent system consists of three “feederagents” in the upper layer and fifteen “loadagent” in the lower layer. This reduces the communication load and the load on the “feeder agents”. Each “load agents” and “feeder agents” have simple restoration rules.

5.2 Types of agents in the MAS

Figure 4-5 shows the architecture for the proposed distributed multiagent system. The system consists of two layers, upper layer and lower layer. The upper layer consists of “FeederAgent” called FA and the lower layer consists of “LoadAgent” called LAG.

![Architecture of the multiagent system](image)

**Figure 5-1:** Architecture of the multiagent system
5.2.1 **LoadAgents**

Each of these agents has some information about their adjacent LAG. As in simulation model will be shown, the LAG’s from 1 to 5 belongs to FA1, and the LAG’s from 6-10 belongs to FA2 and LAG’s from 11-15 belongs to FA3.

Each LAG has also states information for switches, as they are “open” or “closed”. On each feeder, there are five switches. The value of each loads and the current value are known in each LAG. Implementation of these agents must follows some rules that will be described as follow:

- If a blackout occurs due to a fault, all LAG that are in the blackout open switches in order to prepare restoration.
- If rule-1 is completed, a request restoration message will be sent to the FA that the LAG belongs to.
- The LAG get a message form FA that power can be provided, if the power is less than power for LAG, the switch is open.
- If there is extra power to be provided, then extra power will be sent to neighbor LAG to expanding the restoration.
- If there is no extra power to be provided, this will be reported to the FA that the LAG belongs to use another strategy.
5.2.2 FeederAgents

In order to implement these agents, each FeederAgent (FA) should have some information. These information’s are described here as follow:

Each FAs has information about the neighbor FA, and it will do with a switch. The entire FA can make connection through a switch. FA1 exchanges message with FA2 through SAB, and FA1 with FA3 through SAC and FA2 with FA3 through SBC. In initially situation this switches is open, but when a fault happens these switches will be closed to make a connection with each other and make a request or to inform each others.

There are also some rules for feederagent to restore the fault.

- If a request message received from the LAG and power for restoration can be provided, then a message is sent to the applicable LAG through the SAB.
- If a request message received from the LAG and power for restoration can be provided, then a message is sent to the applicable LAG through the SAC.
- If a request message received from the LAG and power for restoration can be provided, then a message is sent to the applicable LAG through the SBC.

One of the behaviors that will be used in FeederAgents is Finite State Machine (FSM) Behavior.

![Figure 5-2: The FIPA Cancel Meta-Protocol](image)

The task of this behavior in FeederAgent is to controlling the 6 states in FeederAgent. Duties of each state discussed in the previous chapter.
5.3 Interaction between Agents to reconfigure an Electric Power Distributed System

As it was explained in previous part about the FA and LAG tasks, in this part will be described the restoration step by step by helps of some figures.

In Figure 5-3 as you can see the normal state of the three bus model is shown. In this model all switches on each feeder are closed, but the switches that connected the each feeder together are opened. In an idle state, the feeders do not have any connection with each others.

Here will be described one case of restoration system. In this case a single fault will be occurred on a load, called blackout load.

First the LAG1 experience a blackout, all the LAG’s on Feeder1 open the switches to prepare for restoration as Figure 5-4 shows.

---

**Figure 5-3:** the idle three bus model

**Figure 5-4:** A blackout load occurs as a fault
Then the blackout load request a restoration to FA1 which is the blackout load belong to, when the FA1 received this request, it asks the neighbour feeder, Feeder 2, which is connected to SAB for a response.

The Feeder 2 sends the power that can be provided to Feeder 1. Then Feeder 1 reports the LAG2 about the power that can be provided, LAG2 restores its own load after closing SAB and inform the LAG3 about the power that can be provided. The LAG2 closed the switch itself and LAG3.

At the last part of restoration, the FA1 ask the other feeder, FA3, through the switch SAC, about the power that can be provided. FA3 responds to FA1 about the power and LAG4 will restore itself and inform the adjacent load, LAG5, and switch between them will be closed.
Figure 5-7: LAG4 will be restored; switches S1 and SAC are closed.

Figure 5-7 shows the system after restoration operations. As a result this case is one in which there is a high probability that all blackout loads can be restored. In this case all blackout loads can be restored through cooperation among agents.
6
SIMULATION MODEL

6.1 Overview

In this chapter will be described the simulation model developed in this work. The implemented simulation model is shown in Figure 6-1. It consists of three main components.

- The simulink implementation of the electric power distribution system. Three bus distributed system with six voltage sources, fifteen loads and switches and twelve transmission lines.
- The Java Development Framework, JADE
- Three agents (Connection, ServerProxyAgent, Server) that implemented in JADE to make a connection between Matlab and JADE.

In this project used Simulink to simulate the model, because Simulink is software that is useful for modeling, simulating and analyzing dynamics systems which support linear and nonlinear systems, model in continuous time and sample time.

In simulink model to create a power distributed system the SimPowerToolbox was useful.

Figure 6-1 shows a view of two software’s which they have been connected to each other by a ServerAgent.
Figure 6-1: An overview of simulink model and JADE software
Figure 6-3 shows a power distributed system which is modeled in PowerFactory. The model wasn’t tested in PowerFactory.

**Figure 6-2:** Three bus modeled in PowerFactory
6.2 Electric Power Distribution System simulation model

The simulation model of the electric power distributed system in our case was modeled in Simulink. With simulink, the SimPowerSystems toolbox was useful to model the system.

When the simulink model works in normal state, without any fault, the power that can be measures from each load is essential for the MAS.

In the normal state, feeders have no connection with each other, means the switch (SAB, SAC and SBC) between any two feeders are opened, and of course switches on each feeder are closed.

All components on each feeder have same values and same status in normal state. But when a fault occurs each feeder will decide about theirs own switches status.

Voltage provided by a Programmable Voltage Source in Simulink. The purpose to use the voltage source was to get the power back from JADE and can act on the simulated model. The voltage source is 220KV with 60Hz. The power that can be provided by this voltage source is about 20KW on each feeder.

Figure 6-3 shows the sample simulation of the power Feeder1.

![Figure 6-3: Power on Feeder1](image)
6.3 Communication Middleware

In this part the implementation of the communication middleware used in Simulation model will be described.
The simulation model contains one “Embedded Matlab Function” to send/receive information to/from the multiagent system.

6.3.1 Client/Server Socket Communication

A socket is a two-way communication link between two programmes running on the network with a port number. A socket has some duties as: connect to a remote machine, send information, receive information, select a connection, bind to a port, and get the incoming information. During a simulation a connection between JADE and simulink is implemented by SocketProxyAgent. The SocketProxyAgent has behaviours to listen and respond messages.

To use sockets, it needs to have a Server and Client. In this project the simulink is a client and the MAS is a server.

In JADE platform, there are three classes that have duties to make a connection, listen to port and write on the port, these three agents called ServerAgent, Connection and SocketProxyAgent.

When the JADE platform launching, the SocketProxyAgent is trying to search for a name and a port number to listen for connection. The default port number that is used in this project is 6789. On the other side, the Matlab function is also should be connected through this port number, and it will be done as this command in Matlab:

```
> t = tcpip('localhost', 6789)
```

In simulation model because of large number of loads and feeders, each components read and listen from theirs own port number.
If the load number one on Feeder1 should inform the LAG1 about its states, it may write a script as this:

```matlab
echotcpip('on', 110);
t = tcpip('localhost', 6789);
send_P_B1 = num2str(P_B1);
open(t);
fwrite(t, value_P_B1);
back_P_B1 = fscanf(t, '%5d');
echotcpip('off');
```
The first line, open the port number 110. The second line makes a TCP/IP connection on port number 6789, which ServerSocketProxyAgent will listen on this port number. Line number three open this connection and then line after ‘fwrite’, it will send the load value which is appeared on Matlab workspace. Function ‘fscanf’ will listen/receive data from the LAG. At the end will connection be closed to prepare to start a connection on different ports.

Each of the Agents in JADE should also send/receive from a specific port, so the JADE platform should be run with this command as follow:

```
-gui jade.Boot -port 110 LAG1:workPack.LAG1
-gui jade.Boot -port 111 LAG2:workPack.LAG2
```

The load number one in simulation model can send/receive data on port 110 as described in JADE will also open this port for LAG1. Each of agents in the MAS has a specific port number which is the same as the simulation model.
7 SIMULATION RESULTS

7.1 Fault scenario 1

In this project was tested one scenario fault with one fault. In this case the fault will be on load one, which is in JADE called LAG1.

To test the simulated model and JADE to find and restore a fault, the model will be simulated in idle state where the switches on each feeder are closed. The simulation result of this test was showed in Figure 5-3. In this case the fault is will be occurred on Feeder1, so the focus is for power on Feeder1. A fault will be a voltage offset, when source goes down and there is no power applied on Feeder1.
As Figure 7-1 shows the model is running in idle state, which there is no fault occurred. To getting a fault to simulated model, it has been put a Fault block that make fault on three phases. The fault begins on time 0.1 second and ends on time 0.5 second. Figure 7-2 shows a result of fault without restoration.
Now both simulink model and JADE running to restore the fault.
Now we compile the Matlab embedded function. It sends the values as they called (send_P_B1, 2, 3…) to the JADE and after the message exchanging in agents, the values sends back from JADE to the Matlab as they called (back_P_B1, 2, 3,…).
These values are going to take place on Matlab workspace. Then from the model by help of a simulink block “FromWorkspace”, the values will be sent to the model.
When the values from JADE reads from simulated model, the power will be restored and it increases to the start value.
The power that provided by idle model (without fault) is about 20KW and the power that has been defined to JADE is also the same. When a fault occurs in the simulated model, the value for power will change and it depends on the fault. As it has been shown in previous figure fault occurs during time interval[0.1s 0.5s]. Figure 7-3 has been divided to three plots, the first plot shows the power with fault, the second plot shows the voltage on feeder1, and the last plot shows the power with fault drawing with yellow color plus the power that coming back from JADE drawing with red color. As
Simulation results

the figure is showing the power back from JADE has the correct values and this value will put in the simulated model to restore the power lost.

![Simulation Result](image)

**Figure 7-3:** Result of simulated model with fault.

It wasn’t possible to have an online effect in Simulink, means; the running model should stop and then put the backing values from JADE in the model and run it again. Figure 7-4 shows the result of the restoration power in simulated model. As the figure is showing the power is coming back to the idle states where it has power on 20KW and certainly the power has the same as backing value from JADE.
Figure 7-4: Result of the power restoration on the simulated model.

Figure 7-5 shows the result of power on the three feeders. These three power power have the same values as expected.
Figure 7-5: Result of the power restoration for all feeders.
CONCLUSION

8.1 Results

In this project a method to restore a distributed network has been worked. The multiagent system build by help of JADE framework presented. A certain strategy for prioritizing certain loads in case of shortage of alternative sources has been proposed. One fault scenario has been considered. Using the method was successful to minimize the blackout load in the distributed system while allowing each agent to operate autonomously based on local information. The results of the simulation in physical model and implementing the proposed method show that the restoration is possible. This project has corporate FIPA protocols in the multiagent system. FIPA protocol has been used to exchanging message or to transfer the power between feeders. It has been shown that the ontology can be used to transfer power between feeders and loadagents during the restoration process. A physical model has been created for the multiagent system in Matlab Simulink, and it has been simulated successful. The connection between the simulated model and the multiagent system has been setup in order to send/listen powers. This connection allows measurement data to be transferred from Simulink and JADE. TCP/IP is a method that makes this connection possible in Matlab Simulink.

8.2 Perspectives

Multiagent systems have been applied to solve the problems as restoration, fault identification, reconfiguration and power system protection. Most of the applications depending on multiagent systems use an agent to represent a component on a circuit like switches, loads, sources, buses. A major problem with such solutions is a large scale of a power system with hundreds of components. To solve this problem it has been chose a method “two-layer” which is a perspective of Nagata application.

The multiagent system makes the agents able to identify and manage during a fault, after the fault has been restored, some behaviour is necessary to be done. The network
system must be maintained in stable condition even if the agent communities are not synchronized.

### 8.3 Further work

The physical part of this project, simulation of the power network, is modelled in simu-link, but Matlab has some more limitation to build a network system with compare to PowerFactory. The next work could be to simulate a model in PowerFactory. The multiagent system has ability to control a dynamic system, thus the multiagent system could be a good technology for some system such as low voltage, antenna, micro electronic systems etc.
REFERENCES


[3] Padgham/Winikoff: *Developing Intelligent Agent System*


[5] Doughlas M.Staszesky, Dean Carig, and Craig Befus; *Advanced Feeder Automation Is Here*


[7] Aristotle University of Thessaloniki (AUTH); *A Multi-agent System for Power Distribution Network Protection and Restoration: Designing Concept and an Application Prototype*


A
MATLAB SIMULINK MODELS
Matlab simulink models

Discrete, s = 5e-005
powergui

Voltage Measurement

Controlled Voltage Source

Current Measurement

Parallel RLC Branch1

Parallel RLC Branch2

Parallel RLC Branch3

Parallel RLC Branch4

Parallel RLC Branch5

From Workspace 1

Current Measurement1

Current Measurement2

Current Measurement3

Current Measurement4

Current Measurement5

Subsystem1

Subsystem2

Subsystem3

Subsystem4

Subsystem5

Controlled Voltage Source1

From Workspace 2

From Workspace 3

From Workspace
Matlab simulink models
Matlab simulink models
MATLAB SCRIPT

%Send/Receive data from MAS on the port

echotcpip('on',110);
t=tcpip('localhost',6789);
value_P_B1=num2str(P_B1);
fopen(t);
fwrite(t,value_P_B1);
back_P_B1=fscanf(t,'%20d');
echotcpip('off');

echotcpip('on',111);
t=tcpip('localhost',6789);
value_P_B2=num2str(P_B2);
fopen(t);
fwrite(t,value_P_B2);
back_P_B2=fscanf(t,'%5d');
echotcpip('off');

echotcpip('on',112);
t=tcpip('localhost',6789);
value_P_B3=num2str(P_B3);
fopen(t);
fwrite(t,value_P_B3);
back_P_B3=fscanf(t,'%5d');
echotcpip('off');

echotcpip('on',113);
t=tcpip('localhost',6789);
value_P_B4=num2str(P_B4);
fopen(t);
fwrite(t,value_P_B4);
back_P_B4=fscanf(t,'%5d');
echotcpip('off');

echotcpip('on',114);
t=tcpip('localhost',6789);
value_P_B5=num2str(P_B5);
fopen(t);
fwrite(t,value_P_B5);
back_P_B5=fscanf(t,'%5d');
Matlab script

echotcpip('off');

echotcpip('on',115);
t=tcpip('localhost',6789);
value_P_B6=num2str(P_B6);
fopen(t);
fwrite(t,value_P_B6);
back_P_B6=fscanf(t,'%10d');
echotcpip('off');

echotcpip('on',116);
t=tcpip('localhost',6789);
value_P_B7=num2str(P_B7);
fopen(t);
fwrite(t,value_P_B7);
back_P_B7=fscanf(t,'%5d');
echotcpip('off');

echotcpip('on',117);
t=tcpip('localhost',6789);
value_P_B8=num2str(P_B8);
fopen(t);
fwrite(t,value_P_B8);
back_P_B8=fscanf(t,'%5d');
echotcpip('off');

echotcpip('on',118);
t=tcpip('localhost',6789);
value_P_B9=num2str(P_B9);
fopen(t);
fwrite(t,value_P_B9);
back_P_B9=fscanf(t,'%5d');
echotcpip('off');

echotcpip('on',119);
t=tcpip('localhost',6789);
value_P_B10=num2str(P_B10);
fopen(t);
fwrite(t,value_P_B10);
back_P_B10=fscanf(t,'%5d');
echotcpip('off');

echotcpip('on',120);
t=tcpip('localhost',6789);
value_P_B11=num2str(P_B11);
fopen(t);
fwrite(t,value_P_B11);
back_P_B11=fscanf(t,'%10d');
echotcpip('off');
echotcpip('on',121);
t=tcpip('localhost',6789);
value_P_B12=num2str(P_B12);
fopen(t);
fwrite(t,value_P_B12);
back_P_B2=fscanf(t,'%5d');
echotcpip('off');

echoTCPip('on',122);
t=tcpip('localhost',6789);
value_P_B13=num2str(P_B13);
fopen(t);
fwrite(t,value_P_B13);
back_P_B13=fscanf(t,'%5d');
echotcpip('off');

echoTCPip('on',123);
t=tcpip('localhost',6789);
value_P_B14=num2str(P_B14);
fopen(t);
fwrite(t,value_P_B14);
back_P_B14=fscanf(t,'%5d');
echotcpip('off');

echoTCPip('on',124);
t=tcpip('localhost',6789);
value_P_B15=num2str(P_B15);
fopen(t);
fwrite(t,value_P_B15);
back_P_B15=fscanf(t,'%5d');
echotcpip('off');
package workPack;

import java.util.Iterator;
import java.util.Vector;
import java.util.ListIterator;
import java.util.Enumeration;

import jade.content.ContentManager;
import jade.content.Predicate;
import jade.content.lang.Codec;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.Ontology;
import jade.content.onto.OntologyException;

import jade.core.Agent;
import jade.core.AID;

import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.FSMBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.proto.ContractNetResponder;
import jade.proto.SubscriptionResponder;
import jade.proto.SubscriptionResponder.Subscription;
import jade.proto.SubscriptionResponder.SubscriptionManager;

import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPANames;
public class FeederAgent1 extends Agent {

    float prefaults[] = {0, 0, 0, 0, 0}; // Initially no pre-fault values
    private float current_powerlevel = 20000;
    private Ontology ontology = PowerNegotiationOntology.getInstance();
    private int step = 0;
    private float power_transfer = 0;
    private float current_request = 0;
    private int loadagentmsg_count = 0;
    private long deadline = 0;
    private Codec codec = new SLCodec();
    private String reqID = "FA1";
    private String subID = "FA1";
    private int informCnt = 0;
    private float localload_priority = 2;
    private Vector MySubscriptions = null; // Has to be initialized

    // Priorities of loads at this feeder:
    private AID LA1 = new AID("LAG1", AID.ISLOCALNAME);
    private AID LA2 = new AID("LAG2", AID.ISLOCALNAME);
    private AID LA3 = new AID("LAG3", AID.ISLOCALNAME);
    private AID LA4 = new AID("LAG4", AID.ISLOCALNAME);
    private AID LA5 = new AID("LAG5", AID.ISLOCALNAME);
    private AID FA1 = new AID("FeederAgent1", AID.ISLOCALNAME);
    private AID FA2 = new AID("FeederAgent2", AID.ISLOCALNAME);
    private AID FA3 = new AID("FeederAgent3", AID.ISLOCALNAME);
private AID[] LAIDs = {new AID("LAG1", AID.ISLOCALNAME),
   new AID("LAG2", AID.ISLOCALNAME),
   new AID("LAG3", AID.ISLOCALNAME),
   new AID("LAG4", AID.ISLOCALNAME),
   new AID("LAG5", AID.ISLOCALNAME)};

// State names
private static final String STATE_1 = "1";
private static final String STATE_2 = "2";
private static final String STATE_3 = "3";
private static final String STATE_4 = "4";
private static final String STATE_5 = "5";
private static final String STATE_6 = "6";

protected void setup() {
    getContentManager().registerLanguage(codec);
    getContentManager().registerOntology(ontology);

    // Adding FSM behaviour:
    FSMBehaviour fsm =
        new FSMBehaviour(this) {
            public int onEnd() {
                System.out.println("FSM behaviour completed.");
                myAgent.doDelete();
                return super.onEnd();
            }
        };

    // Register states:
    fsm.registerFirstState(new State1(), STATE_1);
    fsm.registerState(new State2(), STATE_2);
    fsm.registerState(new State3(), STATE_3);
    fsm.registerState(new State4(), STATE_4);
    fsm.registerState(new State5(), STATE_5);
    fsm.registerState(new State6(), STATE_6);

    // Register the transitions
    fsm.registerTransition(STATE_1, STATE_1, 1);
```java
fsm.registerTransition(STATE_1, STATE_2, 2);

fsm.registerTransition(STATE_2, STATE_3, 3);

fsm.registerTransition(STATE_3, STATE_3, 3);
fsm.registerTransition(STATE_3, STATE_4, 4);
fsm.registerTransition(STATE_3, STATE_1, 1);
fsm.registerTransition(STATE_3, STATE_5, 5);

fsm.registerTransition(STATE_4, STATE_4, 4);
fsm.registerTransition(STATE_4, STATE_6, 6);
fsm.registerTransition(STATE_4, STATE_1, 1);

fsm.registerTransition(STATE_5, STATE_6, 6);

fsm.registerTransition(STATE_6, STATE_6, 6);
fsm.registerTransition(STATE_6, STATE_1, 1);

addBehaviour(fsm);

DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("PowerGeneration");
sd.setName("FeederAgent1");
dfd.addServices(sd);

try {
    DFService.register(this, dfd);
} catch (FIPAException fe) {
    fe.printStackTrace();
}

System.out.println("Agent "+getLocalName()+" is waiting for data from matlab...");
```
Feederagent1 code

```java
protected void takeDown() {

    try {
        DFSerice.deregister(this);
    }

    catch (FIPAException fe) {
        fe.printStackTrace();
    }

    // Printout a dismissal message
    System.out.println("-agent "+getAID().getName()+" terminating.");

}

private class State1 extends OneShotBehaviour {
    private int exitValue;

    public void action() {

        MessageTemplate template = MessageTemplate.and(MessageTemplate.MatchConversationId("load_notification"),
            MessageTemplate.MatchPerformative(ACLMessage.INFORM));

        MessageTemplate subcancel = MessageTemplate.and(MessageTemplate.MatchProtocol(FIPANames.InteractionProtocol.FIPA_SUBSCRIBE),
            MessageTemplate.MatchPerformative(ACLMessage.CANCEL));

        ACLMessage loadagentmsg = myAgent.receive(template);
    }
```
FeederAgent1 code

```java
ACLMessage cancelmsg = myAgent.receive(subcancel);

if (loadagentmsg != null) {

    // Registering or deregistering this connection
    if (loadagentmsg.getEncoding().equals("voltag e_down")) {
        System.out.println(myAgent.getLocalName() + "got LA1 encode");

        AID LoadID = loadagentmsg.getSender();
        if (LoadID.equals(LA1)) {
           efaults[0] = Float.parseFloat(loadagentmsg.getContent());
        } else if (LoadID.equals(LA2)) {
           efaults[1] = Float.parseFloat(loadagentmsg.getContent());
        } else if (LoadID.equals(LA3)) {
           efaults[2] = Float.parseFloat(loadagentmsg.getContent());
        } else if (LoadID.equals(LA4)) {
           efaults[3] = Float.parseFloat(loadagentmsg.getContent());
        } else if (LoadID.equals(LA5)) {
           efaults[4] = Float.parseFloat(loadagentmsg.getContent());
        }
```
if (LA1.equals(loadagentmsg.getSender())) {

    try {
        DFServe.deRegister(myAgent);
        System.out.println(myAgent.getLocalName() + " is deregistered");
        doWait(1000);
        doWake();
    }
    catch (FIPAException fe) {
        fe.printStackTrace();
    }
}

loadagentmsg_count++;
System.out.println("loadagentmsgcount: "+loadagentmsg_count);
    float load_power = Float.parseFloat(loadagentmsg.getContentView());
    System.out.println(load_power);
    current_powerlevel = current_powerlevel - load_power;
    System.out.println(current_powerlevel);
    current_request = current_request + load_power;
    System.out.println(current_request);

    }

else
if (loadagentmsg.getEncoding().equals("voltage_up")) {

    System.out.println(myAgent.getLocalName() + "startup current_powerlevel: " + current_powerlevel);

    if (LA1.equals(loadagentmsg.getSender())) {

        AID[] PowerGeneratingAgents;

        }
boolean registered = false;
DFAgentDescription tp = new DFAgentDescription();

try {
  DFAgentDescription[] result = DFService.search(myAgent, tp);

  PowerGeneratingAgents = new AID[result.length];

  for (int i = 0; i < result.length; ++i) {
    if(result[i].getName().equals(myAgent.getName())){
      System.out.println(myAgent.getName() + "already registered...");
      registered = true;
    }
  }
}

catch (FIPAException fe) {
  fe.printStackTrace();
}

if(registered == false){
  //Registration
  DFAgentDescription dfd = new DFAgentDescription();
  dfd.setName(getAID());
  ServiceDescription sd = new ServiceDescription();
  sd.setType("PowerGeneration");
  sd.setName("FeederAgent1");
  dfd.addServices(sd);

  try {
    DFService.register(myAgent, dfd);
    System.out.println(myAgent.getLocalName() + " is registered");
  }
  catch (FIPAException fe) {
    fe.printStackTrace();
  }
}
Feederagent1 code

    }

    //Cancel any
    if(MySubscriptions!=null){
        ListIterator list = MySubscriptions.listIterator();
        while (list.hasNext()) {
            ACLMessage submsg = (ACLMessage) list.next();
            submsg.setPerformative(ACLMessage.CANCEL);
            submsg.setConversationId("powerup_cancel");
            myAgent.send(submsg);

            ACLMessage command = new ACLMessage(ACLMessage.REQUEST);
            command.setContent("disconnect_ticket");
            Iterator con = submsg.getAllReceiver();
            while(con.hasNext()){ AID contractor = (AID)
                con.next();

                if(contractor.equals(FA2)){
                    command.addReceiver(LA2);
                    //switch SAB
                }
                else
                    if(contractor.equals(FA3)){
                        command.addReceiver(LA4);
                        //switch SAC
                    }
                    else {
                        System.out.println(myAgent.getLocalName()+" No match..");
                        }
                        //Send out command to the appropriate BusAgent to perform the control action
                        command.setConversationId("comma
//Registration end

} //Registration end

else if(cancelmsg !=null){

//handle cancellation from other FAs

    float power_contract = extractContent(cancelmsg,1);
    current_powerlevel = current_powerlevel - power_contract;
    System.out.println(myAgent.getLocalName()+ " CANCEL power_contract "+power_contract);
    current_powerlevel = current_powerlevel - power_contract;
    System.out.println(myAgent.getLocalName()+"current_powerlevel after CANCEL "+current_powerlevel);

    String cancelConID = cancelmsg.getConversationId();

    if(MySubscriptions!=null){

    MySubscriptions = null; //clearing vector

}
ListIterator list = MySubscriptions.listIterator();
while (list.hasNext()) {
    ACLMessage submsg = (ACLMessage) list.next();
    String ConID = submsg.getConversationId();
    if (ConID.equals(cancelConID)) {
        MySubscriptions.removeElement(submsg);
        System.out.println(myAgent.getLocalName()+" removed sub: "+submsg.toString());
    }
}

//Send control action to disconnect
switch
    ACLMessage command = new ACLMessage(ACLMessage.REQUEST);
    command.setContent("disconnect_tie");
    AID contractor = cancelmsg.getSender();
    if(contractor.equals(FA2)) {
        command.addReceiver(LA2); //switch SAB
    } else if(contractor.equals(FA3)) {
        command.addReceiver(LA4); //switch SAC
    } else {
        System.out.println(myAgent.getLocalName()+" No match..");
    }
myAgent.send(command);
//Send reply:
ACLMessage inform = cancelmsg.createReply();
    inform.setPerformative(ACLMessage.INFORM);
    myAgent.send(inform);
    current_request = power_contract;

    exitValue = 2;

}

else{
    exitValue = 1;
}

}

public int onEnd() {
    return exitValue;
}

private class State2 extends OneShotBehaviour {
    private int exitValue;

    public void action() {
        //In this state a power negotiation is initialized
        System.out.println("Executing behaviour " + getBehaviourName());
        myAgent.addBehaviour(new RequestPower(current_request));

        exitValue = 3; //going to next state
    }
}
public int onEnd() {
    return exitValue;
}

private class State3 extends OneShotBehaviour {
    private int exitValue;

    public void action() {
        // Waiting for RequestBehavior to finish
        // System.out.println("Executing behaviour "+getBehaviourName());
        // if negotiation_done == true || no_proposals

        if (step == 4) {
            if (power_transfer > 0) { // something has been agreed
                if (power_transfer < current_request) {
                    // has not got full amount
                    // go to prioritazation

                    exitValue = 4;
                } else {
                    exitValue = 5; // full amount achieved,
                }
            }
        } else {
            exitValue = 1; // failed, return to state 1
        }
    }
}
public int onEnd() {
    return exitValue;
}

private class State4 extends OneShotBehaviour {
    private int exitValue;

    public void action() {
        // Prioritize loads

        /* Priorities:
         * 2213
         */

        // Get power obtained
        // Send shed command to lowest prioritized

        float sum = 0;
        int i = 0;

        while((sum + prefaults[i]) <= power_transfer) {
            // increment
            sum = sum + prefaults[i];

            System.out.println("sum: "+sum);
            ACLMessage connectload = new ACLMessage(ACLMessage.REQUEST);
            connectload.setContent("connect_load");
            connectload.setConversationId("load_command");
            myAgent.send(connectload);

            i++;
        }

        System.out.println("i before while shed: "+i);
Feederagent1 code

```java
while (i<4){

    ACLMessage shedload = new ACLMessage(ACLMessage.REQUEST);
    //shedload.addReceiver(BAIDs[i++]);
    shedload.setContent("disconnect_load");
    shedload.setConversationId("load_command");
    myAgent.send(shedload);
}

exitValue = 6;
}

public int onEnd() {
    return exitValue;
}

private class State5 extends OneShotBehaviour {
    private int exitValue;

    public void action() {
        System.out.println(myAgent.getLocalName() + "Executing behaviour " + getBehaviourName());
        //full amount achieved
        //Send out switching command to LoadAgents to switch in loads:
        int i;
        for (i=0; i<4; i++) {
            ACLMessage connectload = new ACLMessage(ACLMessage.REQUEST);
            connectload.addReceiver(LAIDs[i]);
            connectload.setContent("connect_load");
            connectload.setConversationId("load_command");
            myAgent.send(connectload);
        }
        exitValue = 1;
    }
```
private class State6 extends OneShotBehaviour {
    private int exitValue;

    public void action() {
        if (MySubscriptions != null) {
            int size = MySubscriptions.size();

            //System.out.println(myAgent.getLocalName()+"subCnt: "+subsCnt);
            System.out.println(myAgent.getLocalName()+"size "+size);
            System.out.println(myAgent.getLocalName()+"informCnt "+informCnt);

            if (size == informCnt) { // not all subs have been added yet
                informCnt = 0; // reset variable
            }

            ListIterator list = MySubscriptions.listIterator();
            ACLMessage command = new ACLMessage(ACLMessage.REQUEST);
            command.setContent("connect_tie");

            while (list.hasNext()) {
                ACLMessage submsg = (ACLMessage) list.next();
                System.out.println(myAgent.getLocalName()+submsg.toString());
                Iterator con = submsg.getAllReceiver();
                while (con.hasNext()) {
                    AID contractor = (AID) con.next();
                    if (contractor.equals(FA1)) {

                        // More code...
                    }
                }
            }
        }
    }
}
Feederagent1 code

    command.addReceiver(LA2);
    //switch SAB
    } else if(contractor.equals(FA1)) {
        command.addReceiver(LA4);
    //switch SAC
    } else if(contractor.equals(FA1)) {
        command.addReceiver(LA3);
    //switch S3
    } else if(contractor.equals(FA1)) {
        command.addReceiver(LA1);
    //switch S1
    } else if(contractor.equals(FA1)) {
        command.addReceiver(LA5);
    //switch S5
    } else {
        System.out.println(myAgent.getLocalName() + " No match..");
    }

    } }

    } //Send out command to the appropriate BusAgent to perform the control action
    command.setConversationId("command");
    System.out.println(myAgent.getLocalName() + command.toString());
    myAgent.send(command);

    exitValue = 1; //return
} else {
    exitValue = 6;
}
} else {

private class RequestPower extends Behaviour {

    private String powerrequired; // The best offered price (modified: amount of power offered)
    private float proposal_byBest;
    private float power_required;
    private float total_proposed = 0;
    private int count = 0;
    private int repliesCnt = 0; // Count the replies from other agents
    private MessageTemplate mt; // The template to receive replies
    private float power_proposal; // used in two states
    private AID[] PowerGeneratingAgents;
    private int no_of_proposals=0;
    private ACLMessage rejectproposal = new ACLMessage(ACLMessage.REJECT_PROPOSAL);

    Vector Proposals = null;

    public RequestPower(float power){
        this.power_required=power;
        step = 0;
    }

    public void action(){

        switch (step) {

            case 0:
/ Send the cfp to other agents
ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
cfp.setProtocol(FIPANames.InteractionProtocol.FIPA_CONTRACT_NET);

// Update the list of power generating agents
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("PowerGeneration");
template.addServices(sd);
try {
DFAgentDescription[] result = DFService.search(myAgent, template); //CHANGE BACK TO myAgent!
System.out.println("Found the following power generating agents:");
PowerGeneratingAgents = new AID[result.length];
for (int i = 0; i < result.length; ++i) {
if(result[i].getName().equals(myAgent.getName())){ //CHANGE BACK TO myAgent!
;
} else{
PowerGeneratingAgents[i] = result[i].getName();
System.out.println(PowerGeneratingAgents[i].getName());
}
}
} catch (FIPAException fe) {
fe.printStackTrace();
}

for (int i = 0; i < PowerGeneratingAgents.length; ++i) {

gAgents.length; ++i) {
    cfp.addReceiver(PowerGeneratingAgents[i]);
}
deadline = System.currentTimeMillis()+2000; //Taking the current time and specifying deadline

cfp = fillContent(cfp,String.valueOf(power_required),localload_priority);
cfp.setConversationId(reqID);
myAgent.send(cfp);

// Prepare the template to get proposals (PROPOSE as well as REFUSE)
mt = MessageTemplate.and(
    MessageTemplate.MatchConversationId(reqID));

step = 1;

case 1:
    // Receive all proposals/refusals from other agents
    //System.out.println(myAgent.getLocalName()+"In case 1");

    ACLMessage cfpreply = myAgent.receive(mt);

    if (cfpreply != null) {
        // Reply received
        if (cfpreply.getPerformative() == ACLMessage.PROPOSE) {
            // This is an offer
            no_of_proposals++;

            if(Proposals == null){

            }
Feederagent1 code

```java
Proposals = new Vector();
}
Proposals.addElement(cfpreply);
//Adding the proposal msg

//System.out.println("distance: "+Distance);
//System.out.println(myAgent.getLocalName() +" Got proposal");

rejectproposal.addReceiver(cfpreply.getSender());
//System.out.println("in cfpreply" + rejectproposal.toString());
}

repliesCnt++;  //Both PROPOSE and RE- FUSE will be counted
}

long currentTime = System.currentTimeMillis();
if ((repliesCnt >= PowerGeneratingAgents.length) || ((currentTime >= deadline) && (Proposals != null))) {
    if (Proposals != null) {
        float power_temp = 0;
        boolean powerfound = false;
        while (powerfound == false && count < no_of_proposals) {
            ListIterator iter = Proposals.listIterator();

            //float total_proposed = 0;
            ACLMessage bestmsg = new ACLMessage(ACLMessage.INFORM); //just initialized
            int bestdistance = 10;  //The shortest distance from power provider to initiator
            int distance = -1;  //The dis-
```
tance between this feeder and the others

while (iter.hasNext()) {

    ACLMessage promsg = (ACLMessage) iter.next();

    //get sender AID:
    String proposer = promsg.getSender().getLocalName();

    if(proposer.equals("FeederAgent1")){
        distance = 1;
    } else if(proposer.equals("FeederAgent2")){
        distance = 2;
    } else if(proposer.equals("FeederAgent3")){
        distance = 3;
    } else{
        System.out.println("error identifying proposer");
        distance = 100;   //dummy
    }

    if(distance < bestdistance){
        //System.out.println("int
distance if...");
        bestdistance = distance;
        bestmsg = promsg;
        //System.out.println("bestmsg "+bestmsg.toString());
        proposal_byBest = extractContent(promsg,1);
    }
}
float power_left = power_required - power_temp;
//System.out.println("power_temp:
"+power_temp);

//System.out.println(myAgent.getLocalName()+bestmsg.toString());
String BestProvider =
bestmsg.getSender().getLocalName();

rejectproposal.removeReceiver(new
AID(BestProvider, AID.ISLOCALNAME));
ACLMessage order = new ACLMes-
sage(ACLMessage.ACCEPT_PROPOSAL);
if(power_left > 0){  //Still
need more power
    if(proposal_byBest >= pow-
er_left){  //proposal is more than power left we need
        powerfound = true;
        order = fillCon-
tent(order,String.valueOf(power_left),localload_priority);
        //priority is one
    }
    else{  //accept full amount
        power_temp = power_temp +
proposal_byBest;
        order = fillCon-
tent(order,String.valueOf(proposal_byBest),localload_priori-
ty);  //priority is one
    }
    order.setProtocol(FIPANames.I
teractionProtocol.FIPA_CONTRACT_NET);
    order.setConversationId(reqID
);
    order.addReceiver(new
AID(BestProvider, AID.ISLOCALNAME));
    myAgent.send(order);
}
Proposals.removeElement(bestmsg);

count++;

} //System.out.println("out of loop...");

rejectproposal.setConversationId(reqID);

//System.out.println(rejectproposal.toString());
myAgent.send(rejectproposal);
step = 3;
} else{
    step = 4;
    System.out.println(myAgent.getLocalName()+" only got refusals, returning to main...");
}
else if((currentTime >= deadline)){
    step =4; //returning System.out.println(myAgent.getLocalName()+" got no messages, returning to main...");
} //System.out.println("currentTime:
"+currentTime);
    //System.out.println("deadline:
"+deadline);
    break;
    case 2:
    System.out.println(myAgent.getLocalName()+"In case 2");
    mt = MessageTemplate.and(
MessageTem-
plate.MatchProtocol(FIPANames.InteractionProtocol.FIPA_CON-
TRACT_NET),}
MessageTemplate.MatchConversationId(reqID);

    step = 3;
    //break;
  case 3:

  //System.out.println(myAgent.getLocalName()+"In case 3");
    // Receive the power order reply
  ACLMessage reply = myAgent.receive(mt);
  if (reply != null) {

    // power order reply received
    if (reply.getPerformative() == ACLMessage.INFORM){
      informCnt++;

      //System.out.println(myAgent.getLocalName()+" has received
      INFORM message from "+reply.getSender());

      //Update power_transfer variable:
      float power_agreement = extractContent(reply,1);
      power_transfer = power_transfer +
      power_agreement;
    }
  else {
      System.out.println(myAgent.getLocal
      Name()+" PowerRequest failed: Requested power not available
      from."+reply.getSender());
    }

    if(informCnt==count){   //All inform
      msg received
      step = 4;
    }
  }
}
else {
    block();
}
break;
}

public boolean done() {
    if (step == 2 && Proposals == null) {
        System.out.println("Attempt failed: "+ power_required + " refused");
    }
    return ((step == 2 && Proposals == null) || step == 4);
}

private ACLMessage fillContent(ACLMessage msg, String power_value, float prio) {
    Float priority = new Float(prio);
    try{
        ContentManager cm = this.getContentManager();
        msg.setLanguage(codec.getName());

        PowerRequest powerreq = new PowerRequest();
        powerreq.setPowerValue(power_value);
        //filling value
        powerreq.setPriority(priority);  //filling priority
        Power power = new Power();
        Request req = new Request();
        req.setPowerRequest(powerreq);
        power.setPowerRequest(powerreq);
    }
cm.fillContent(msg,power); //giving list to content manager
    return msg;
}
catch (OntologyException oe) {
    oe.printStackTrace();
    return null;
}
catch (CodecException ce) {
    ce.printStackTrace();
    return null;
}

private float extractContent(ACLMessage msg, int choose_return ) {
    try{
        ContentManager cm = this.getContentManager();
        Predicate pre = (Predicate) cm.extractContent(msg);

        Power power = (Power) pre;
        PowerRequest powerrequ = power.getPowerRequest();

        String powerrequest = powerrequ.getPowerValue();
        float power_request = Float.parseFloat(powerrequest);

        Float flo = powerrequ.getPriority();
        float prio = flo.floatValue();

        if(choose_return == 1){
            return power_request;
        } else if(choose_return == 2){
            return prio;
        } else{
            return -1;
        }
    }
} //main Agent end
Feederagent1 code

www.elektro.dtu.dk/cet

Department of Electrical Engineering
Centre for Electric Technology (CET)
Technical University of Denmark
Elektrovej 325
DK-2800 Kgs. Lyngby
Denmark
Tel: (+45) 45 25 35 00
Fax: (+45) 45 88 61 11
E-mail: cet@elektro.dtu.dk