Smart Grid Energy Trading Using Multi Agent Systems

A Project Report

Submitted by

ROHIT KABRA

(EE08B082)

in partial fulfillment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

AND

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

THESIS CERTIFICATE

This is to certify that the project report titled **Smart Grid Energy Trading using Multi Agent Systems** submitted by **Rohit Kabra**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology in Electrical Engineering** and **Master of Technology in Power Systems and Power Electronics**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. K.S.Swarup
Research Guide
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai
Date:.

ACKNOWLEDGEMENTS

I am grateful to my guide, *Prof. K.S.Swarup*, for his encouragement and guidance throughout my project. I thank him for giving me the freedom of working in my own way, which was crucial in getting a lot more interested in this area of than I was when I began working on the project. His comments and useful inputs were instrumental in making significant progress. This work has been carried out at the Department of Electrical Engineering, Indian Institute of Technology Madras

The completion of this project is in no small measure due to my Power Systems lab mates with whom I have had many a discussion. They have always maintained a lively atmosphere in the lab that is conducive to research. Their suggestions, encouragement and humor are contributors to this thesis.

I have been helped by a lot of colleagues throughout my journey at IITM, both academically and personally. My classmates in Electrical Engineering have helped me academically and have constantly raised the bar for success, improving me in the process.

My family has constantly supported me through the ups and downs in life. I am indebted to them for life and a line or two is hardly a fitting payback. Nevertheless, I would like to thank them for their love and affection they have showered on me throughout my life.

ABSTRACT

The electrical energy distribution model that evolved over the period of past couple of decades, there was little concern over the environmental issues and energy sources. However, the biggest change is coming from the consumers themselves, who are now installing their own solar panels and wind turbines since the last decade

Consumers can then become producers themselves and form a new actor on the field of the energy market, called "**Prosumers**". This will give the regular consumers the possibility to get their energy directly from these Prosumers without any intervention from a Genco. In this case the Prosumers can supply their own energy needs, as well as the energy needs of a few other consumers. This will create a new free energy market with Prosumers and Gencos that offer the same service, which is much more decentralized and where the consumers can decide where they get their energy and at which price. Prosumers can use weather forecasts to manage their solar panels and/or wind turbines production and adjust prices according to the market trends and raw material prices.

A lot of different systems have already been implemented in the field of energy market simulations. One of these systems is the Multi Agent Systems (MAS) by N. Capodieci, which was used a basis for this project report. This is a basic MAS that simulates the energy market by using Consumer, Prosumer and Genco agents that buy and sell energy in a contracting auction. It also has a time and weather simulation and a GUI. The scope of this research is to use the existing MAS by N. Capodieci that supports this simulation and to expand it by adding scalability and reliability improvements to make the system more usable. This was needed because these features have not been taken into account when that MAS was created. The system was limited to one host only and has no specific measures to prevent failures. This limited the usage of the system, as only a limited amount of agents could be run and the system could crash on a fault.

KEY WORDS: Prosumers, Smart Grids, Multi Agent Systems, Scalability and Reliability.

Table of Contents

List of Tables

List of Figures

Abbreviations

MAS	Multi Agent Systems
DT	Distribution Transformer
SCADA	Supervisory Control and data Acquisition
DMS	Distribution Management System
UC	Unit Commitment
MIP	Mixed Integer Programming
LMP	Locational Marginal Pricing
DF	Directory Facilitator
JADE	Java Agent Development
FIPA	Foundation for Intelligent Physical Agent
ISO	Independent System Operator
AMS	Agent Management System
EH	Exception Handling

Notations

Genco	Large scale generator Units
Prosumer	Both Producers (Mostly Renewable Energy) and Consumers of Energy

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

Electrical energy production by man-made machines has been around since 1831, when the first electric generator was invented by Michael Faraday. Since then there has been a huge transformation, the generators now have a much larger capacity, there are many more around and they are often placed in energy production facilities, which are scattered over a country. This resulted in the development of Electricity Grid and which is still being expanded, in order to provide energy across the country.

In this view, every consumer is restricted to a limited amount of energy production companies called Gencos. Because of limited Gencos, the competition between Gencos is very low, resulting in no innovation towards a more copious or decentralized system and often high prices for the consumers. This has changed in the last decade, as the market is still heavily regulated by the government, but market competition is possible and more common. This has resulted in a more open market where consumers can choose a Genco that suits their needs.

Another change from the last two decades, is the decoupling of the production sector (electricity production facilities), controlled by the Gencos, from the distribution sector (the electricity grid), controlled by the new transmission service operators, or TSOs.

Next-generation transmission and distribution infrastructure will be better able to handle possible **bi-direction energy flows**, allowing for **distributed generation** such as from photovoltaic panels on building roofs, but also the use of fuel cells, charging to/from the batteries of electric cars, wind turbines, pumped hydroelectric power, and other renewable sources.

To move to a more open and democratic energy market, certain changes have to be made to the current market. There is very little knowledge of how to properly design a retail electricity market and how to effectively incorporate other services. The energy distribution service requires quality improvements for the new market to function correctly, because of the higher granularity of the energy contracts. Because of this increased granularity they would need to handle a huge amount of operations in the system. Also, searching for the best sellers in the new market with millions of suppliers should be done autonomous.

The concept of intelligent control for regulating the power network variables is to be realized. The **intelligent multi agent based control** can be a solution in today's power network to maintain the dynamics such as adequate power balance along with quality voltage under changing system conditions such as load and power injection. The technology with multi-agent intelligent control may be main module of Smart Grid architecture. The idea behind any multi-agent system is to break down a complex problem handled by a single entity – a centralized system – into smaller simpler problems handled by several entities – a distributed system.

1.2 INTRODUCTION TO PROBLEM STATEMENT

The evaluation of the new Multi Agent System (MAS) is addressed on the two main factors. The **scalability** and **reliability** of the MAS. Predefined test cases are designed to test the factors. The test cases are based on the evaluations questions stated below. These questions correspond to the methods that have been implemented in the system and are used to test these methods effectiveness. The evaluation questions are measured by using certain metrics. The metrics are divided into two main categories:

- Systems parameters related.
- Coordination mechanisms related.

System metrics include system related performance measurements, such as wall-clock times and CPU and memory usage. Coordination metrics are more related to message performance on the entire MAS, such as time to reach convergence, or the average response time of agents. The different metrics are usually combined, by summing them up or adding weights, in order to obtain a single value which may be compared.

To evaluate the scalability of the old MAS and the new MAS and to compare their performance, some test cases are used in the thesis. Most of the tests are conducted on the old MAS as well as the new MAS, in order to compare performance on both main factors. These test cases are based on earlier research on scalability measurements of Multi-agent systems in general. The scalability of the system is viewed as the ratio between performance and resources.

Evaluation questions - Objectives:

- How many agents can the MAS handle before it becomes unstable?
- What is the scalability of the new MAS compared to the old MAS, in terms of performance per load?
- What is the scalability of the new MAS with respect to performance increase per resource?
- What is the performance overhead of critical agent* replication?
- What is the performance overhead of restarting an agent* in the system through the EH agent?

*Agents here directly correlates with the number of Loads, Gencos and Renewable sources of energy (Prosumers).

Coordination metrics:

- Total number of messages transferred between agents
- Number of agents in the system
- Number of replicas of top level intermediaries
- Number of hosts

System metrics:

- Time to reach convergence /Number of ticks(process/communication cycles) that have passed.(Ticks = milliseconds * 10.000)
- CPU and memory usage

CHAPTER 2
SMART GRID RESEARCH

2.1 INTRODUCTION

A **Smart Grid** is an electrical grid that uses information and communications technology to gather and act on information, such as information about the behaviors of suppliers and consumers, in an automated fashion to improve the efficiency, reliability, economics, and sustainability of the production and distribution of electricity.

There is no uniform definition of smart grid. According to the European Technology Platform, a Smart Grid is an electricity network that can intelligently integrate the actions of all users connected to it – generators, consumers and those that do both Prosumers – in order to efficiently deliver sustainable, economic and secure electricity supplies [1].

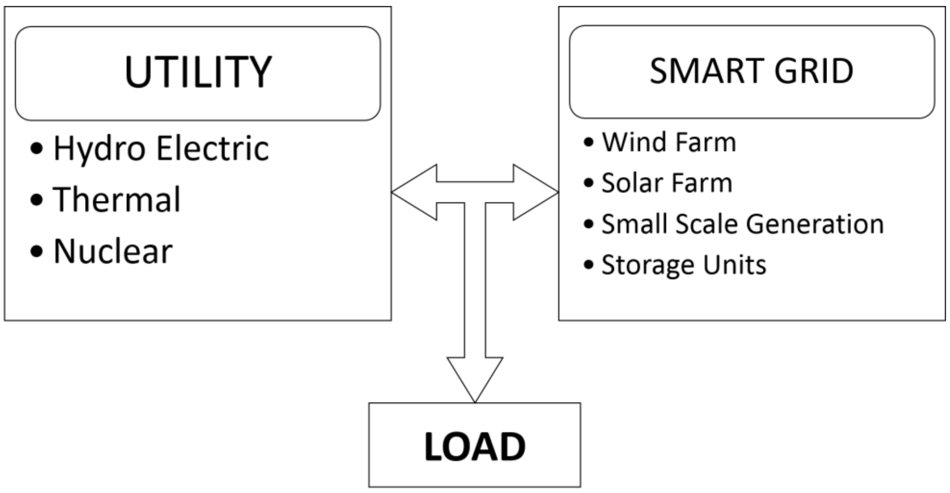


Figure 2.1 Block Diagram showing Smart Grid and Utility

According to the US Department of Energy, the smart grid is self-healing grid, enables active participation of consumers, operates resiliently against attack and natural disasters, accommodates all generation and storage options, enables introduction of new products, services and markets, optimizes asset utilization and operates efficiently, and provides reliable and high quality power for the digital economy [2]. According to the Australian Government smart grid combines advanced telecommunications and information technology applications with ‘smart’ appliances to enhance energy efficiency on the electricity power grid [3].

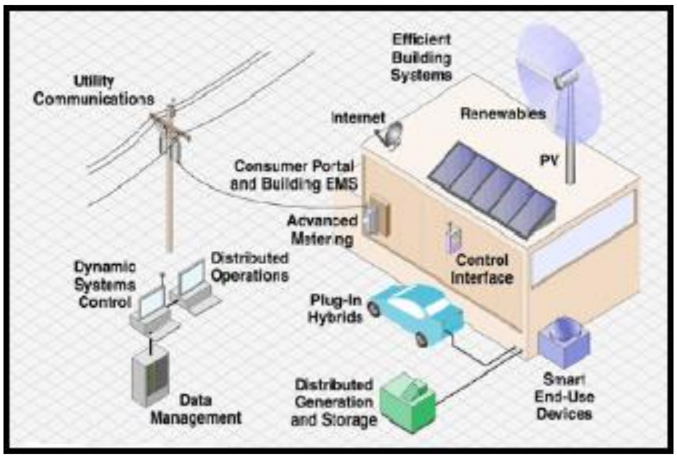


Figure 2.2 Flow diagram showing various components of Smart Grid [3]

2.2 EXPECTED BENEFITS FROM SMART GRID:

Smart grid has benefits to both utilities and customers. Some of these benefits are briefly described below:

- By applying advanced information technology (IT) and combining IT with ‘smart appliances’, smart grids can enhance energy efficiency on the electricity power grid, in homes and in businesses
- By using advanced meters, sensors, and digital controllers, smart grids will be able to automate, monitor and control the two-way flow of electricity across networks.
- By using smart grid, transmission and distribution companies will be able to improve control over the network and can gather complex, real-time information about grid performance.
- Smart grid can enhance the reliability of electricity supply by automatically preventing outages and improving the detection of power lines overloads and faults.
- Smart grid can manage voltage within the grid and help reduce the losses that occur as electricity travels along transmission and distribution lines.

2.3 SMART GRID IN INDIAN CONTEXT

India has formed smart grid forum and task force to study [2] and finalize the smart grid road map, keeping in view of the following points:

2.3.1 Smart Grid for India

The focus of Smart Grid to provide choices to each and every customer for deciding the timing and amount of power consumption based upon the price of the power at a particular moment of time India has recently experienced an impressive rate of growth as its government implements reforms to encourage foreign investment and improve conditions for its citizens. However, with its electrical grid, India loses money for every unit of electricity sold because India is home to one of the weakest electric grids in the world; the opportunities for building the Smart Grid are great.

2.3.2 Need for Smart Grid in India

With such enormous deficiencies in basic infrastructure, why would India want to consider investing in smart grid technologies? Ultimately for India to continue along its path of aggressive economic growth, it needs to build a modern, intelligent grid. It is only with a reliable, financially secure Smart Grid that India can provide a stable environment for investments in electric infrastructure, a prerequisite to fixing the fundamental problems with the grid. Without this, India will not be able to keep pace with the growing electricity needs of its cornerstone industries, and will fail to create an environment for growth of its high tech and telecommunications sectors.

2.3.3 Recent developments in Indian Grid

The Indian National Government, in cooperation with the State Energy Board, put forward a road to improvement when it announced the new Electricity Act of 2003, aimed at reforming electricity laws and bringing back foreign investment.

The act had several important measures:

- Unbundling the State Electricity Board's assets into separate entities for generation, transmission, and distribution, with the intention of eventual privatization
- Implementation of RAPDRP (Restructured Accelerated Power Development & Reform Program) program for power distribution utilities across the country for preparation of baseline data for each project covering Consumer Indexing, GIS Mapping, Metering of all DT (Distribution Transformer) and substation Feeders, and also automated data logging for all DTs, Feeders and SCADA(Supervisory Control and data Acquisition) /DMS (Distribution Management System) for energy auditing /accounting and IT based consumer service center.
- Adding capacity in support of a projected energy use growth rate of 12%, coinciding with a GDP growth rate of roughly 9%
- Improving metering efficiency
- Auditing to create transparency and accountability at the state level
- Improved billing and collection
- Mandating minimum amounts of electricity from renewable
- Requiring preferential tariff rates for renewable
- End use efficiency to reduce the cost of electricity

There has been a recent push in India to begin labeling appliances with energy use to help consumers determine operating costs. There has also been significant effort to improve energy efficiency, for example to increase the average energy efficiency of power plants up from 30% to 40%, and pushing major industries to reduce energy consumption after execution of Energy Conservation Act'2001.

2.3.4 Need for Design of India Grid in line with US.

As is the case in most of the world, the Indian national grid was not designed for high-capacity, long-distance power transfer. As is the case in the United States, India needs to interconnect regional grids. Although coal and hydro-electric potential has peaked in many parts of India, there are still several regions with excess capacity.

Large wind potential and increasing wind capacity in the south and west also create a need for transmission infrastructure. Unfortunately, like the United States, regions are generally sectionalized, with some asynchronous or HVDC links allowing for minimal power transfer. The biggest difference is that India's transmission grid only reaches 80% of its population, while the transmission grid in the United States reaches over 99% of its population.

2.3.5 Financial Health of the Indian Grid

India's transmission and distribution losses are among the highest in the world, averaging 30% of total electricity production, with some states as high as 50%. When non-technical losses such as energy theft are included in the total, average losses are as high as 40%. The financial loss has been estimated at 1.5% of the national GDP, and is growing steadily. India's power sector is still largely dominated by state utilities.

Despite several attempted partnerships with foreign investors, few projects have actually been implemented. This lack of foreign investment limits utilities' ability to raise needed capital for basic infrastructure. This financial frailty, coupled with public ownership of utilities and the related bureaucratic slowness, has made it very difficult for investors to take interest in India's grid. The Smart Metering Conference is now happening in India. In fact smart grid conference is going to happen in India in the 1st Quarter of 2011.

Practically speaking, the organization has to assess how its end-to-end delivery and operational value chains will be affected and determine how smart grid enhancements can add value to the customer and other stakeholders. Given sufficient time and effort, all of these challenges are manageable. The test for corporate leaders is to create a shared vision and engage internal and external stakeholders in a common focus to collaborate and ensure that smart grid benefits are delivered cost effectively. India has problems not unlike other developing countries India's grid is in need of major improvements.

This neglect has accumulated in a variety of system failures like:

- Poorly planned distribution networks
- Overloading of system components
- Lack of reactive power support and regulation services
- Low metering efficiency and bill collection
- Power theft

While the Indian government's ambitious "Power for All" plan calls for the addition of over 1 TW of additional capacity by 2012, it faces the challenge of overcoming a history of poor power quality, capacity shortfalls and frequent blackouts. One of the first things governments have to do when privatizing the state distribution utility electricity is to make the enterprise attractive to investors. This is not always easy because often a key reason for privatizing is that the government-owned electricity company has run up substantial losses and accumulated large debts under government ownership. Private firms are not interested in loss-making, debt ridden concerns. One way round this is for the government to assume the debts of the distributors so that the private firms take on investments that are debt free. Alternatively, the sector can be organized to ensure that the monopolistic structure is maintained so that investors will be more likely to make a profit. Another option is to increase prices or to guarantee a return to investors.

CHAPTER 3
ENERGY TRADING FOR SMART GRID

The change from the centralized and obsolete model of the energy network to the new open energy network, requires legislation changes to legally and economically work. Wholesale transactions (bids and offers) in electricity are typically cleared and settled by the market operator or a special-purpose independent entity charged exclusively with that function. Market operators do not clear trades but often require knowledge of the trade in order to maintain generation and load balance. The commodities within an electric market generally consist of two types: power and energy. **Power** is the metered net electrical transfer rate at any given moment and is measured in megawatts (MW) [3]. **Energy** is electricity that flows through a metered point for a given period and is measured in megawatt hours (MWh).

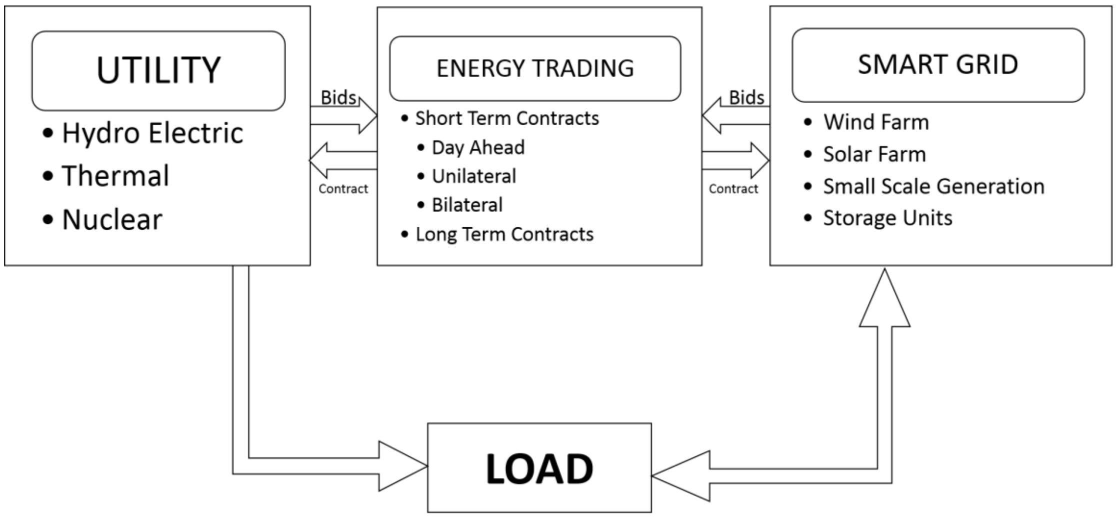


Figure 3.1 Block diagram showing Energy Trading in Smart Grid

3.1 ENERGY MARKET SIMULATIONS

Markets for energy related commodities are net generation output for a number of intervals usually in increments of 5, 15 and 60 minutes. Markets for power related commodities required by, managed by (and paid for by) market operators to ensure reliability, are considered ancillary services and include such names as spinning reserve, non-spinning reserve, operating reserves, responsive reserve, regulation up, regulation down, and installed capacity.

Apart from the major operators, there are markets for transmission congestion and electricity derivatives, such as electricity futures and options, which are actively traded. These markets developed as a result of the restructuring of electric power systems around the world. This process has often gone on in parallel with the restructuring of natural gas markets.

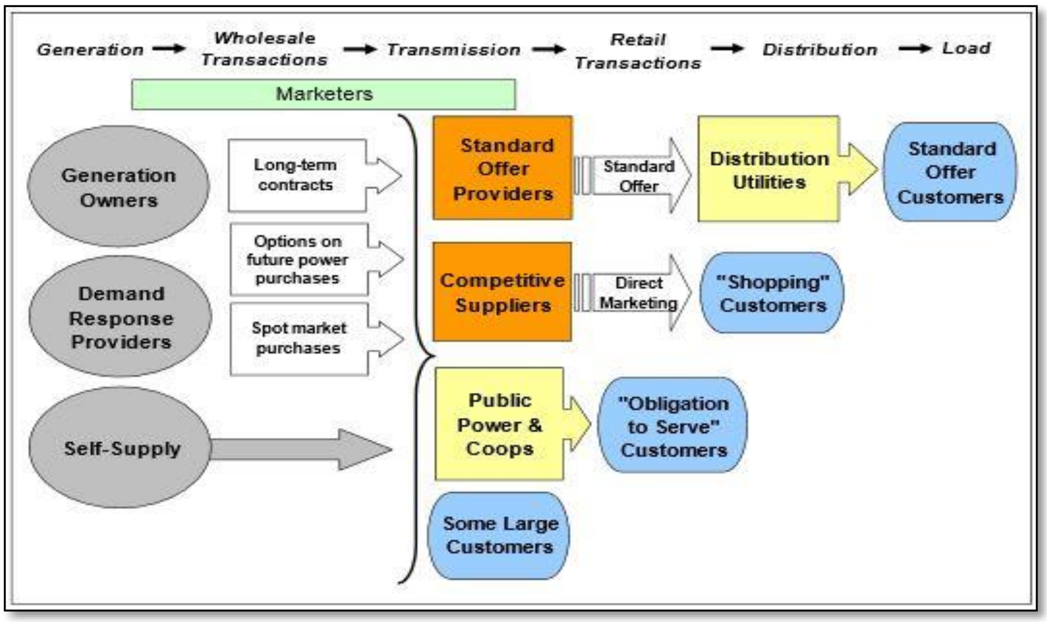


Figure 3.2: Overview of Energy Trading

Systems that focused on distribution, learning strategies and demand and supply balancing have already been studied extensively. This was often focused on using the laws of a certain country as a basis and simulating how these had to be changed in order to achieve a more deregulated market.

Market simulations provide a valuable mechanism to forecast market prices for both zonal and nodal energy markets. Long term and short-term simulations can be performed to forecast congestion locations and corresponding Locational Marginal Prices (LMPs).

Market simulation studies are normally performed using a Security Constrained Unit Commitment application that emulates the ISO/RTO Day Ahead (DA) market clearing process and calculates zonal or nodal prices.

For short-term studies, a more detailed DA clearing model is used with short-term forecasts of load, unit availability and unit bids. Also more detailed models of unit startup and shut down behavior are used. In addition, for nodal markets a full AC power flow is used to iterate with the Unit Commitment (UC) Mixed Integer Programming (MIP) dispatch to enforce the linearized transmission constraints.

For long-term LMP simulations, a Monte-Carlo process is used to model uncertain factors such as random unit outages and in some cases load levels, fuel prices, hydro conditions, Scheduling Points (SP) prices, etc. Monte-Carlo runs provide statistical values of LMPs and unit generation outputs.

Simpler models of the unit commitment constraints are generally used to make the intensive simulations computationally viable.

3.2 MARKET SIMULATION APPLICATIONS

- Market price forecasting
- Congestion forecasting
- CRR (Congestion Revenue Rights) strategic evaluation and analysis
- Transmission flow forecasting
- Loss factor forecasting
- Generation plant revenue and profit forecasting, investment evaluation
- Generator bid strategy evaluation
- Integrated Plant Expansion plan, where multiple plant expansion options are considered, automatically selecting the best set of options via dynamic programming.
- Market design studies where different market designs are simulated to determine benefits of alternative designs

Market Simulators may also be used to perform sensitivity studies of different plant expansion options, fuel price scenarios, load forecasts and transmission expansion options. These sensitivity studies may be used in investment risk analysis and evaluation.

The technologies that other researchers have used to create a market simulation are very different from each other. This also includes non-agent-based systems such as the system proposed by [4], where a web based JSP/Servlet solution is used. This system features a Java powered framework that uses the JSP/Servlet pages as resources. A lot more of the systems that are proposed today are agent systems. Since the year 2000, the first agent-based systems for energy market simulations have been created, with nowadays outdated technologies, such as CORBA and ZEUS. A Java toolkit that originated from agent-based modeling in social sciences, called RepastJ, was used in [5] to create an electricity market framework (AMES).

This project uses Java Agent Development Framework (JADE), which is the most commonly-adopted agent-oriented middle-ware that conforms to Foundation for Intelligent Physical Agents (FIPA), as the agent platform for development. There are also other researchers that have used JADE for an agent-based system for electricity market simulation. JADE was used to develop the wholesale electricity market that is modeled as a Multi-agent system.

There are also agent system that do not use a peer based model like JADE, which are SEPIA and MASCEM. MASCEM (a Multi-agent system that simulates competitive electricity markets) was created in [6] by using Open Agent Architecture (OAA), a framework for integrating heterogeneous software agents in a distributed environment.

CHAPTER 4
MULTI AGENT SYSTEMS

4.1 INTRODUCTION TO MAS

A Multi-agent system is a system that consists of several agents that interact with each other. These interactions are often handled by messages that are sent between the agents. These agents simulate intelligence by using methodical, functional, procedural or algorithmic search, find and processing approaches. Each of the agents can have different goals and behaviors, which together combines to a dynamic system. The agents have some critical features according to [7]: they are at least partially autonomous, no agent has a global view of the system or it cannot use this knowledge practically, there is no controlling agent. Multi-agent systems are very useful in solving problems that are difficult or impossible for an individual agent or a monolithic system to solve. This could be problems like modeling social structures or simulating a trading market. A lot of work has already been done in the field of Multi-agent systems, as it is used for a wide variety of applications.

The ease of use has improved, as a standard for communication between the agents that was defined for industrial and commercial Multi-agent systems was released for public use. This formal IEEE standard called FIPA (Foundation for Intelligent Physical Agent), is commonly used today and focuses on facilitating the interoperability of agents and Multi-agent systems across different software platforms.

Another improvement in the field of Multi-agent systems, is the development of Multi-agent platforms and programming languages. This makes the implementation of Multi-agent systems much easier and makes it possible to create Multi-agent systems that are used in actual operations.

The agent platforms that are available today for Multi-agent system development include: DESIRE, Jadex, TuCSoN and JADE among others [8]. According to [9], the most used platform is JADE.

These platforms are often combined with agent-oriented programming languages, which are used for the implementation of the agents' behavior within the Multi-agent system. These languages include: FULX, JACK Agent Language, 3APL, Jason.

4.2 JADE

To create a Multi-agent system, the easiest way is to use a specialized agent programming platform. By using a platform, the implementation of the system becomes much smaller, because the communication and several other aspects have already been taken care of. This makes it possible to focus solely on the agent implementation itself. There is a wide range of Multi-agent platforms available on the Internet. Each of these platforms differs in their features and their flaws and a lot of them are no longer being updated or supported. From this wide range of platforms, only one can be selected and used. This platform must match some criteria in order to be able to use the agents' possibilities to their full extent and prove to be the 'right' platform for this job. It must provide an easily updatable environment and a standardized multi-platform programming language, supporting libraries or extensions for fault tolerance, security and distribution.

A few other researchers [9] and [10] have already compared some of the more known, updated and used platforms on a list of criteria. These results can be used as a selection basis for this thesis.

From these sources it can be concluded that JADE, or Java Agent Development Environment, is the best choice for general purpose uses. The JADE platform supports Multi-agent system development with Java.

The choice of this platform for the implementation of the Multi-agent system was based on some advantages and criteria [9][10][11]:

- The MAS in this thesis is based on the MAS by N. Capodieci, which also uses JADE for implementation. The advantage of using the same implementation platform is that no recoding is needed, as the same implementation work can be used and adapted for the new system.

- JADE is updated regularly and has a large development crew and community.
- JADE uses Java and each agent is run in a separate thread, which is faster than conventional Java threads.
- JADE works on any platform that supports a Java Virtual Machine, or JVM.
- The methods and architecture proposed in this paper do not conflict with the possibilities of the JADE platform and Java.
- The JADE platform itself already implements and uses some of the methods proposed in this paper.
- The JADE platform is free and open source, which makes this a cheaper choice than a paid alternative and allows for customization of the source code.
- There is standard Java API documentation for JADE, as well as numerous other Internet sources containing tutorials, manuals and Q and A. Most of these are largely up to date.
- It has an excellent GUI with a lot of useful features and tools.
- It has already been used in a lot of development and research projects and has a high acceptance rate in the community.
- It supports the FIPA specification standard for Multi-agent system messaging.
- There are very good security features, such as SSL support for inter platform communications, permission grants and added security possibilities.
- The platform is easy to distribute on multiple hosts.
- There is a wide range of different extensions and libraries for additional features, such as added security, web service integration and embedded JADE for small devices.
- It supports multiple communication and transport protocols, such as socket, RMI and IIOP communication.

It is also possible to use a special agent language together with JADE for the implementation of the agents, but this is not used in this implementation as the Java programming language provides enough possibilities in this case for implementing the agent behavior.

4.3 JADE Framework:

JADE (**J**ava **A**gent **DE**velopment) was developed by Telecom Italia (CSELT) in 1998. JADE became open source software in 2000 and is developed by Telecom Italia (Library Gnu Public License).

Jade creates multiple containers for agents, each of which can be on the same computing system or different systems. Together, a set of containers forms a *platform*. Each platform must have a **Main Container** which holds two specialized agents called the AMS agent and the DF agent.

- The **AMS** (Agent Management System) agent is the authority in the platform. It is the only agent that can create and kill other agents, kill containers, and shut down the platform.
- The **DF** (Directory Facilitator) agent implements a yellow pages service which advertises the services of agents in the platform so other agents requiring those services can find them.

For example, in the figure below, the framework consists of 3 Hosts. They exist on a common network protocol stock. One of the host behaves as the front end while the two other hosts act as JADE containers. Each Container has its own set of Application Agents.

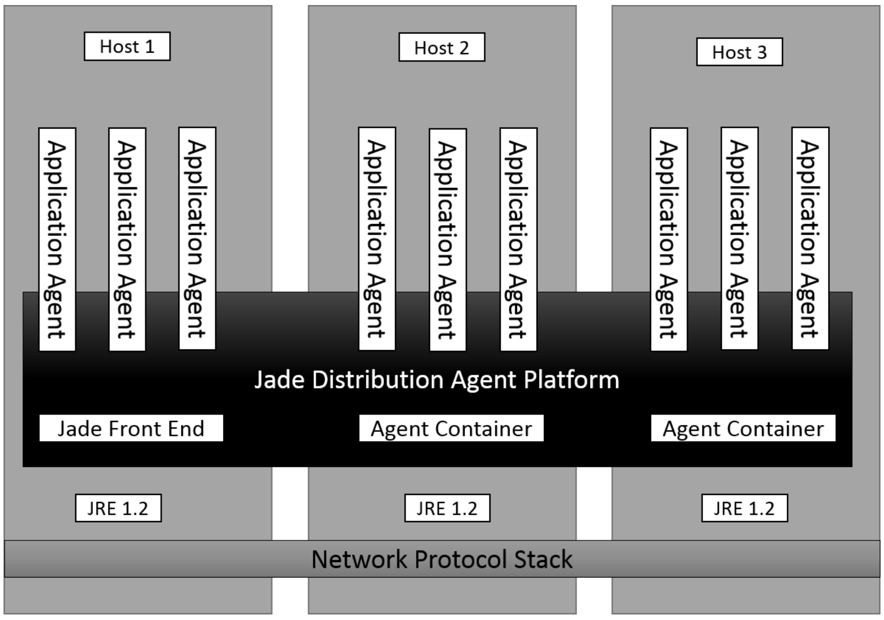
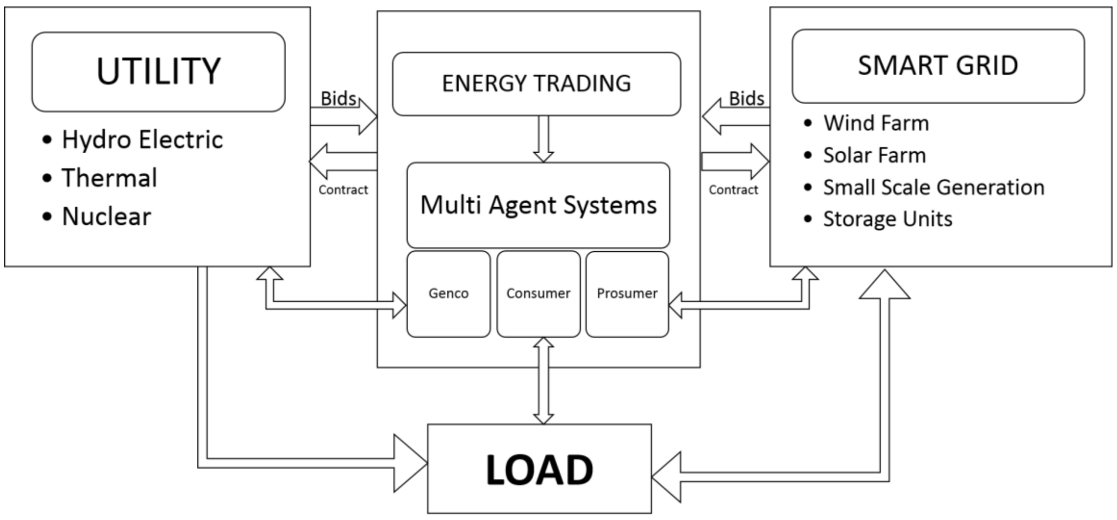


Figure 4.1: JADE Framework Structure

The proposed market has been designed and implemented; the following sections provide an insight to the architecture and configurations.

Given that there is a marketplace for trading energy, different order configurations should be made available to the participants. Using order configurations, one can express specific energy requirements, or usage patterns. The order configurations are composed of two behaviors. The first dimension, specifies whether units of an order can be partially matched, or if must be fully matched. "Fully match" indicated if a participant wants everything or nothing. The second dimension specifies if an order has to be matched immediately. If immediate match is required, possible matching is executed while the unmatched part of the order is automatically cancelled. Matching limitations of this dimension are the trading price and availability of the trading commodity. With these four order configurations, participants should be able to express their internal processes, or trading strategies. For instance, a fully matching order could be used for a process which requires the full amount of energy to be available for the entire duration.



\

Figure 4.4: Block Diagram showing Smart Grid Energy Trading using MAS

4.4 SCALABILITY IMPROVEMENT METHODS

In this section the methods that can be used to improve the scalability of the MAS are explained in detail. The advantages and disadvantages of each method are discussed.

4.4.1 Agent-level improvements:

These methods focus on the agent implementation and organization, in order to improve scalability. There are two methods described here, changing the agent organizational form and locating the agents based on caching lists.

Change agent organizational form:

Jennings and Turner have defined several organizational forms of MAS in [12], suitable for a trading scenario, comparable to our MAS. The forms are distinguished by the constraints within which the agents interact with each other. They have defined three different forms.

In the first organizational form, each customer can communicate with each supplier and the other way around. But customers are unaware of other customers and suppliers are unaware of other suppliers. Which means that agents of the same type cannot share information, form groups or undertake co-operative behavior

The second organizational form is the same as the first form, with the exception that it is also possible for costumers to communicate with other customers and for suppliers to communicate with other suppliers. In this case agents are social and represent a standard fully connected peer MAS.

The third organizational form expands on the second form, by adding an intermediary agent that undertakes collective tasks. This agent performs intermediary functions, such as matchmaking, recruitment, facilitation, etc., thus relieving the other agents of this work. Changing the organizational form, can increase the scalability of the system, because agents can share tasks and intermediaries can reduce the workload on other agents.

Advantages:

Choosing the right form for the current MAS can reduce the communication overhead and increase efficiency, by introducing more agent teamwork.

Disadvantages:

A disadvantage of this method is that some research has to be done, in order to pick the right organizational form for the current MAS. Also the chosen form might later turn out to be non-efficient for the current MAS. It is also possible that none of the forms matches what is required in this case.

Locate agents based on agents caching list:

Each agent in a MAS needs to know where other agents are, in terms of addresses, in order to communicate with them. This process can be time consuming, if the agent does not know what the addresses are. To increase the performance and scalability, agents can use caching lists to store the location of other agents [13]. In this approach, each agent has a list of other agents it knows. This list stores all the relevant information about other agents, such as addresses, names and expertise. This list may not be up to date or correct, and changes dynamically. It can be assumed that with a high message reliability and a slow frequency of change, the agents' lists are largely up to date and accurate.

When an agents needs an address of another agent, it checks its caching list. If the address is not there, the agent will contact some, or all agents in its caching list, for the address information. These other agents, will perform the same procedure recursively. To prevent duplicate request handling, a unique request identity is used. To guarantee cooperative behavior, payment schemes can be used. The communication overhead of this method is very low, with an average complexity of O (1), if the contact list is limited to a certain size.

Advantages

An advantage of this method is that it removes the need for middle agents to serve as brokers. The communication overhead is therefore reduced. It is also very suitable for heterogeneous MAS, because there is no dependence on middle agents. The method also has a very low overhead.

Disadvantages

This method may not be very suitable for unstructured MAS, because of the required inter-agent communication, which may result in slow response times.

4.4.2 System-level improvements:

These methods focus on the system structure and components, in order to improve scalability. There are five methods described here: hiding communication latencies, component distribution, component replication, agent scheduling and transparent access.

Hiding communication latencies:

This method, proposed in [14], focuses on geographical scalability. If a MAS spans a large area network, there may be severe communication latencies. Agents may be waiting very long for responses from other agents. These latencies cannot simply be solved, but it is possible to change the agents. The agents can be adapted to do other useful work, while they wait for responses. In this way the communication latencies can be hidden. It requires that agents can be interrupted when a response is delivered.

Advantages

When possible, a major advantage is that agents can perform other tasks when waiting for a response. The communication latencies can be largely hidden by doing other useful work.

Disadvantages

The agents have to be interrupt-able, to be able to handle the asynchronous requests. The agents must be able to do other useful work, instead of waiting for a response.

Component distribution:

Component distribution can be used to partition the MAS over multiple separate servers [15]. Agents are distributed over different physical machines, in order to spread the load. The distribution of components can be manually done by a human programmer. The different components can be hosted in different processes. This requires inter-process communication, which can be realized with Java RMI, simple socket communication or JADE. If component distribution is used in large-scale networks, this method should be combined with hiding communication latencies, whenever possible, in order to ensure performance increase.

Advantages

An advantage of this method is the easy implementation and the instant increase in scalability of the system. A large amount of machines can be added to support a large scale system.

Disadvantages

This method does have some drawbacks, such as the manual distribution. The designer must decide which components are distributed and how they are distributed. The best distribution strategy is therefore difficult to achieve, because of the varying load situations, which complicates adjustments to the distribution. Another disadvantage is that the distributed components are bound to one machine and cannot scale beyond the limits of this machine, unless the implementation supports dynamic moving of components to other machines.

Component replication:

Component replication can be used to replicate certain components of the MAS across a network. This can improve scalability by reducing communication latencies, by placing components close to where they are needed. Expected performance bottlenecks can be resolved by replicating.

In this way it can also spread the load on certain components, by decentralized load balancing. The replicated components can be hosted in different processes and on different machines. This requires inter-process communication, which can be realized with Java RMI, simple socket communication or JADE.

Advantages

One advantage is the reduction in communication latency, by bringing the components close to the agents that use them. Another advantage is the possibility to prevent bottlenecks in the system, by replicating heavily used components.

Disadvantages

A disadvantage of replication is inconsistency problems. These can be overcome, but introduce some amount of overhead. The replicas must be consistent with each other, which can be achieved by global synchronization or by adopting a weaker consistency model. This consistency model depends heavily on the application.

Another disadvantage is that replication increases resource consumption and complexity. By adding more agents, resources are wasted since all the agent specific services are also replicated. In addition, the system becomes more complex, since more components have to be managed. Load balancing is required for this to work.

Agent scheduling:

To increase performance of individual hosts and therefore scalability, agent/thread scheduling can be used [15]. This method enables the execution of large numbers of (reactive) agents. With agent scheduling, the agents that are not performing any tasks, are deactivated and only require memory resources. The agents that are active and performing tasks, can use all the resources. This scheduling of agents preserves the resources for the active agents, preventing resource wastes. To make the scheduling most efficient, there should be a large group deactivated agents and a small group of active agents. To determine which agents should be deactivated and which agents shouldn't, a scheduling policy must be used. This policy must also be able to control each agents' access to system resources. There are multiple scheduling policies, which use ranking of importance, statistics and heuristics.

A common form of scheduling involves messages received by agents. The agents that received messages are moved from the deactivated group to the active group. After the message has been processed, the agent is moved back. A variant of this approach uses events, such as a user logging on or off. The agents that are associated with this event, are moved to the active group. After the event has been processed, the agents are moved back. In both approaches, most of the agents should be reactive to make the methods efficient enough.

Advantages

The major advantage of this method is the large increase in performance that can be achieved, by efficient scheduling of the active and inactive agents.

Disadvantages

Unfortunately this method is not useful for large numbers of pro-active agents, because there is only a small number of inactive agents in this case and the performance can actually decrease in this case, because the scheduling itself is also computationally intensive.

Transparent access:

Transparent access provides a possibility to enable a MAS to scale beyond the limitations of underlying physical machines [15]. Scalability can be improved by providing transparent access to the distributed resources available. Transparent access prevents additional complexity of the MAS, by hiding resource locations. This results in simple access and flexible adding or removing of resources. Transparent access can be realized by using a transparent resource management layer to use/create threads and objects within other processes. The transparent access layer allows a host to farm out the execution of agents. Only by distributing the load it becomes possible to ensure that a large number of agents reside in a single agent host.

Agents themselves cannot access the system resources or services directly, but only through an environment object. This environment object is a proxy that keeps the implementation of its public methods hidden. This helps achieve two goals, fine-grained control and location independence.

Fine-grained control of the agents, provides a way to distribute resources among the agents according to the importance or vitality of their services and to disconnect troublesome agents. Physical location independence of the agents is achieved by interaction via a proxy and by hiding the location details. Agents can thus be moved freely by the system between processes.

Advantages

The major advantage of this method is that it increases the location independence of the MAS. This makes it possible to use different kinds of physical machines and/or software and add or remove resources. This method also increases the effectiveness of other system-level methods, such as replication and distribution.

Disadvantages

This method can increase the overhead on the system, because an extra layer is added to the system.

4.4.3 SUMMARY

Table 4.1: Summary of Scalability Improvement Methods

Method	Advantages	Disadvantages
Change agent organizational Form	Can reduce communication overhead and increase efficiency	Organizational form must fit the problem that the agents are modeling
Locate agents based on agents caching list	No need for middle agents. Is suitable for heterogeneous MAS.	May not be suitable for unstructured MAS.
Hiding communication Latencies	Agents can perform other tasks when waiting for a response.	Agents must be interrupt-able and immediate Communication must not be required.
Component distribution	The components are distributed, thus spreading out the workload.	Must be combined with hiding of communication latencies, to ensure performance increase. The components must be manually distributed.
Component Replication	Data is close to the agents. Bottlenecks can be prevented, by replicating heavily used components.	Possible data inconsistencies. Load balancing required.
Agent scheduling	Can increase performance, by efficient scheduling of active and inactive agents.	Useless for large numbers of pro-active agents. Computationally expensive.
Transparent access	Increases location independence. Increases effectiveness of other agent-level methods	Can increase overhead.

4.5 RELIABILITY IMPROVEMENT METHODS

In this section the methods that can be used to improve the reliability of the MAS are explained in detail. The advantages and disadvantages of each method are discussed after which a summary is given.

4.5.1 Agent-level improvements

These methods focus on the agent implementation and organization, in order to improve reliability. There are four methods described here: using sentinels to check the system, using agent teamwork to handle agent failures, refuse requests ability and increase agent mobility.

Using sentinels to check the system:

Sentinels can be used to increase the reliability of the system [16][17]. These sentinels are agents, which can guard specific functions or guard against specific states in a MAS. It is up to the designer to decide which functions are most vital for the systems integrity, because not all of the functionality can be guarded. Sentinels can take several actions to guard the system. They can choose alternative problem solving methods for agents, exclude faulty agents, alter parameters for agents, and report to human operators. They do not take part in the problem solving of other agents, but they can intervene in this process. By using semantic addressing, the sentinels can interact with other agents and monitor their communication and interaction, in order to build models of these agents. Some parts of these models are exact copies of the agent models and are called checkpoints. These points assist in detecting faulty agents and inconsistencies, by providing information of the internal state of an agent and its behavior Timers can be used to detect crashed agents or faulty communication links.

Advantages

The advantage of sentinels is, that they are separable from the system. The sentinels can be added after the whole system has been developed and tested. They can also be modified, without affecting the system. The communication mechanisms used by the sentinels and the relevant checkpoints can also be created and altered when needed.

Disadvantages

A disadvantage is that the freedom of the agents is limited by the sentinels. Also the functions that need to be guarded have to be decided by the designer and the system must have support for fault handling and reporting, in order for the sentinels to work. This method is also not very suitable for high volume MAS with highly frequent messages, because a lot of sentinels have to be added in this case, and they have to process a lot of messages.

Using agent teamwork to handle critical agent failures

Using agent teamwork to handle critical agent failures can be used as a method to increase the reliability of the system [18]. This method involves the usage of teamwork between the agents in the system as a technique to automatically recover a Multi-agent system from a sudden agent failure. These failures could be caused by a machine crash, network breakdown, or death of the agent process.

Each agent in the system finds other agents in the system and stores their name or address to be able to communicate with them. When a critical agent disconnects from the system, each agent that fails to contact this agent, attempts to inform the other agents in the system of this failure. Only the agents that regularly communicate with the now disconnected critical agent are informed. After successfully contacting an agent in this manner, this agent updates his information and gives up his attempts to contact the disconnected critical agent.

The Multi-agent system has recovered from failure of the disconnected critical agent when all the agents that interact with that agent have been contacted in this manner. The requests that were in progress at the time of the failure, and hence could not be completed, may be sent again by the requesting agent. This can be considered fault tolerant behavior and hence improves the reliability of the system.

Advantages

Results in minimal overhead, as the teamwork is only used in case of an agent failure. Critical agent failures can be solved and cascading effect can be prevented. Also this method is easy to implement, as it only requires a few special messages and some code to read them and to act on them.

Disadvantages

The use of teamwork may interfere with the required autonomy of agents in the MAS.

Refuse requests ability

The ability to refuse requests can increase the reliability of the agents [18]. Agents can refuse requests to stop flooding of messages. This is making the agents more autonomous and less susceptible to the influences of other agents. It can be implemented by using a message queue and refusing messages if the queue exceeds the maximum queue length.

Advantages

This method can prevent agent thrashing and make the system more reliable. Agent thrashing can occur when there are more messages being received by an agent than it can handle. These messages may stack up and consequently slow the entire system down. If messages are refused, this can no longer occur.

Disadvantages

A disadvantage of this method is the discarding of the messages itself. Some MAS models may require that no messages are discarded, or rely on certain messages being received.

Increase Agent Mobility

Agent mobility is measured in the ability of agents to be moved from one host to another. Agent mobility can be improved by increasing protocol independence and host independence of agents.

Increasing agent mobility can provide a more fault-tolerant system. For example, if a host is experiencing computational problems due to too many agents, the computational intensive agents can be moved to another host.

Advantages

The agents are no longer tied to certain protocols and/or hosts. In case of failures on a certain host, the agents could be moved to another host. This method can also increase the scalability by providing a way to support load balancing in a distributed environment, by moving agents.

Disadvantages

The moving of the agents itself could be computationally intensive, depending on the number of agents being moved and the size of the agents' data.

4.5.2 System-level improvements

These methods focus on the system structure and components, in order to improve reliability. There are four methods described here: distinct domain independent exception handling service, active replication, passive replication and critical agent/adaptive replication.

Distinct domain independent exception handling service

An exception handling service can be used to provide a way of reducing the exception handling within the agents [19]. This domain-independent service handles all the exceptions that occur within agents and thus reducing the load of the agents. The exception handling can be separated from the agents doing the logic and provide a way of control. The agents become simpler and do not need to know about the exception handling. This is also called the "citizen" approach. It requires at most that agents support three very simple directives *(‘‘are you alive?’’*, *‘‘resend RFB’’*, and *‘‘canceltask’’*). The service can prevent cascading effects of an exception, by informing other agents of the failure. The method enhances the reliability by offloading exception handling from problem solving agents to distinct, domain-independent services.

Advantages

The load on the agents in the MAS is reduced, by moving the exception handling from each agent to a central location. The agent implementation becomes simpler, as the agents do not have to handle the failures themselves. Fault cascading effects can be prevented.

Disadvantages

This method results in a more centralized system, which may conflict with the required autonomy of the MAS. Another disadvantage is the dependency of the service on communication with the agents. If this fails, the service is no longer able to detect faults.

Active replication

Replication can be used for data and/or computation, to make a distributed system more fault tolerant [20]. Active replication is a replication protocol where each component is replicated and all replicas concurrently process all input messages. This increases reliability, because a replica can immediately replace another, in case of a system failure.

Advantages

The advantage of active replication is, that it provides a fast recovery delay and is ideal for real-time constrained systems.

Disadvantages

Active replication leads to a high overhead, the overhead equals the amount of replicas. Which makes this method more resource intensive than passive replication. This method is not very suitable for large-scale, adaptive replication.

Passive replication

Replication can be used for data and/or computation, to make a distributed system more fault tolerant [20]. Passive replication is a replication protocol where each component is also replicated, but only one of the replicas processes all input messages and periodically transmits its current state to the other replicas in order to maintain consistency. If the active replica is faulty, a new active replica is chosen from the passive replicas and the execution is restarted. This increases reliability, because a mostly up to date backup can be restored, in case of a system failure.

Advantages

This method requires less CPU resources than the active approach, by activating redundant replicas only in case of failures and still provides a reliable backup mechanism. It also has a low overhead under failure free execution, because of the periodic updates.

Disadvantages

This method needs a checkpoint management which is still expensive in processing time and space and does not provide short recovery delays. As well as active replication, this method is not very suitable for large-scale, adaptive replication.

Critical Agent/Adaptive Replication

A different replication protocol is based on the criticality of certain agents. Only those agents that are defined as critical are replicated and the others are not. Furthermore, one must determine the most critical agents and the needed number of replicas of these agents. There are two cases here:

The agent's criticality is static, in which case, the organization structure of the agents doesn't change, the behavior is static and the number of agents is small. In this case the critical agents can be identified before run time and replicated where needed. The agent's criticality is dynamic, in which case, the organization structure of the agents is dynamic, the behavior is dynamic and the number of agents is large. In this case the critical agents cannot be identified before run time and must be based on the dynamic organizational structure.

This can be achieved by observing the domain agents and dynamically evaluating their criticality, based on semantic-level information and system-level information. This approach increases reliability, because the critical agents are replicated and can replace crashed critical agents. Non critical agents are not replaced in this case.

Advantages

Not a very big impact on performance, because not all agents are replicated, only the critical ones. The system is much more reliable, because it can keep functioning despite failures.

Disadvantages

A system where all of the agents are critical is not suitable for this method, because of the performance impact. Replicas may require synchronization for the system to function correctly.

4.5.3 SUMMARY

Table 4.2: Summary of Reliability Improvement Methods

Methods	Advantages	Disadvantages
Using sentinels to check the system	Sentinels can be added later on and can be modified on the fly.	Not very suitable for high volume MAS with highly frequent messages. Agent communication and world model needs to be public.
Using agent teamwork to handle critical agent failures	Can recover from critical agent failures. Prevents cascading effects. Simple implementation.	May interfere with the required autonomy of the MAS.
Refuse requests Ability	Prevents agent thrashing.	Might not suit all the MAS applications, because important messages might be discarded.
Increase agent Mobility	Agents are not tied to certain protocols and/or hosts.	Moving the agents around can be computationally intensive.
Distinct domain independent exception handling service	Reduced load on the agents in the MAS. Simpler implementation. Fault cascading effects can be prevented.	Centralized approach. Relies on communication with the agents.
Critical agent/adaptive replication	Not a big impact on performance. System still functions despite agent failures.	Not suitable for systems with large numbers of critical agents. Replicas may require synchronization.
Passive replication	Minimizes processor utilization by using checkpoints to restore faulty agents.	Requires checkpoint management which is expensive in processing time and space.
Active replication	Provides fast recovery	Lead to a high overhead.

4.6 SELECTED METHODS

From all the methods described in the previous sections, some have been selected as usable for this MAS. The next two sections discuss the selected methods for scalability and reliability and explains why these have been selected.

4.6.1 Scalability

Not all of the methods described in the previous section are used. Some of the methods are not suitable for this MAS, or do not provide an increase of scalability in this case. The methods that are used are:

- Locate agents based on caching lists
- Distribution
- Replication
- Agent scheduling
- Transparent access

Locate agents based on caching lists is also used, because it reduces the load on the middle agents/brokers to handle all communication as agents can store agent locations themselves. Especially when the current MAS is distributed, the load on the middle agents/brokers could become very large. This method also increases support for possible future changes, as it is suitable for heterogeneous MAS. The current MAS is not unstructured, so the disadvantage is not a problem. Distribution is used, because it is an essential method for increasing scalability. Without distribution the whole system is bound to one machine. With this method the agents are still bound to their respective machines, but not to only one. The agents do have to be distributed manually, but by examining the structure of the MAS this should not pose a big problem.

Replication is used, because it can further increase the performance gain of distribution. This is done by replicating the heavily used agents/components. This should make the system more scalable than by having only one of these components. Communication distances/latencies are also decreased by this method. The possible problems with this method can be solved by implementing a data consistency update. Load balancing is partially solved by the use of distribution and by limiting the amount of replicated agents.

Agent scheduling is used, because the current MAS does not have a large number of pro-active agents and thus does not limit this method. At certain times there are a lot of inactive agents in the system, waiting for responses, or when the auctions have ceased. Agent scheduling can make the system more efficient.

Transparent access is used, because it increases the effectiveness of distribution and replication and also makes these methods easier to implement. It also increases the location independence of the agents, thus making it easier to distribute these. The increase in overhead is limited and does not compare to the performance increase gained by using this method in combination with distribution and replication.

The methods that are not used are:

- Change agent organizational form
- Hiding communication latencies

Change agent organizational form is not used, because the alternative organizational forms do not apply to the current MAS. The current MAS uses a scheme where the suppliers communicate with the Consumers and where top-level intermediaries interact with the suppliers and Consumers. One alternative organizational form requires removal of the top-level intermediaries. Removing the top level intermediaries is not a viable solution, because their functionality has to be separated from the Consumers and suppliers and cannot be incorporated within these agents. The other form requires intercommunication between the Consumers and between the suppliers. This is not useful, because the Consumers have no messages or information which they need to discuss with themselves. This also goes for the suppliers. Therefore changing the form would result in loss of functionality or useless overhead, which is why it is not used.

Hiding communication latencies is not used, because the current MAS is not meant to be run on an Internet-scale network, meaning that communication latencies will be limited. Also the agents of the system do not have many other tasks to perform when waiting for a reply, making this method not efficient enough to implement.

4.6.2 Reliability

Not all of the methods described in the previous section are used. Some of the methods are not suitable for this MAS, or do not provide an increase of reliability in this case. When choosing the methods that are used, the degree of fault tolerance is of great importance. According to [21] there are four main sources of faults:

- Inadequate specification of software
- Software design error
- Processor failure
- Communication error

Where the first two are unanticipated in consequences and the last two can be considered in the design of the system. Even if all possible measures are taken to prevent faults, the first two sources above imply the difficulty in building fault-free systems. This emphasizes the need for fault tolerance. If the system cannot handle a fault and shows unexpected behavior, there is a system failure. If, on the other hand, the system can handle the fault situation, it is called fault tolerant. For the MAS to be more reliable, it needs to be fault tolerant as well. The following degrees of fault tolerance are proposed by [21]:

- Full fault tolerance, where the system continues to operate without significant loss of functionality or performance even in the presence of faults.
- Graceful degradation, where the system maintains operation with some loss of functionality or performance.
- Fail-safe, where vital functions are preserved while others may fail.

For this MAS, the aim is to achieve Graceful degradation, because Full fault tolerance is very difficult to achieve in a Multi-agent System and usually results in a performance decrease caused by the required methods, which will conflict with the requirement of a more scalable system. The methods that would be required for a Full fault tolerant system are Active or Passive replication and Using sentinels to check the system. These are not necessary for Graceful degradation, as Critical agent/adaptive replication and the distinct domain-independent exception handling service provide enough means to maintain system operation and most functionality.

This degree of fault tolerance results in the following methods that are used:

- Using agent teamwork to handle critical agent failures
- Refuse requests ability
- Distinct domain independent exception handling service
- Critical agent/adaptive replication

Using agent teamwork to handle critical agent failures is used, because it provides a way to handle failures or crashes in the critical agents that are used. These agents are critical to the system and the other agents in the system should be informed if these agents fail. By informing other agents, this method also prevents cascading effects of a failure. The overhead of this method is limited, because only in case of a failure additional messages are sent and additional functions are called. Finally the implementation is also simple. The autonomy of the MAS is no problem in this case, as the agents do need to be movable and do not require complete autonomy.

Refuse requests ability is used, because the buyers and sellers can be flooded by the offers sent to each other as the system is scaled up. This method prevents thrashing of the agents by message floods. It is also easy to implement, by using a message queue with limited length in combination with a garbage message collector. The disadvantage is not a problem in this MAS, because the offers can be re-sent and are not critical for the system.

Distinct domain independent exception handling service is used, because it makes exception handling easier than in the normal case. The implementation of the agents in the MAS becomes simpler, because of the central handling. The agents can perform more useful tasks instead of exception handling. Another big advantage is the preventing of fault cascading effects, which are common in Multi-agent systems. The disadvantages of this method are not problematic, as the required autonomy of the MAS is not violated. The agents can still act independently and negotiate on the prices. The failure of communication is a problem, which also applies to the entire MAS and therefore cannot be considered a major disadvantage of this method.

Critical agent/adaptive replication is used, because it increases the reliability of the system significantly. Failures of agents in the system no longer result in failure of the entire system. The impact on performance is very limited, as only the critical agents in the system are replicated. There is only a small number of critical agents in this MAS. The main portion of agents are the suppliers and Consumers. The suppliers and Consumers are not critical agents, as the system will continue to function despite failure of these agents. This makes one of the disadvantages obsolete. The synchronization of replicas is also limited, because there are not many critical agents and because almost no functionality needs synchronization.

The methods that are not used are:

- Using sentinels to check the system
- Increase agent mobility
- Passive replication
- Active replication

Using sentinels to check the system is not used, because it is not useful for high volume MAS with highly frequent messages, which resembles our MAS. The method also results in some communication overhead and limits the freedom of the agents in the system. Another point is that the checkpoints have to be decided manually and that the sentinels can be difficult to implement. The advantage of sentinels is not big enough to compensate these problems. Increase agent mobility is not used, because it is computationally expensive to move agents around.

Also the advantage of being able to move agents around does not weigh against the cost of moving in the current MAS. The agents are distributed manually and it is expected that the computational intensity of the agents does not vary much during the experiment, thus removing the need for load balancing by moving agents between workstations. Also the chances of a workstation crashing completely and requiring moving of agents, are not very high and acceptable for the current MAS, because of the limited running time.

Passive replication is not used, because it replicates all the agents and results in a less efficient system. All the information must be updated to the replicas of each agent and requires an inefficient checkpoint management system. The recovery delay is also high and becomes higher as the size of the system increases. This conflicts with the scalability demands of the system, also the system should be real-time. Also a replication of all the agents is not required for the current MAS to continue functioning.

Active replication is not used, because it also replicates all the agents and it results in a high overhead because all the replicas are updated in real-time. The overhead becomes larger as the system increases, which also conflicts with the scalability demands. The recovery delay is not an issue, as it is designed to be real-time. The replication of all the agents is not required for the current MAS to continue functioning.

4.7 AGENT BEHAVIOR

The behavior of each agent is explained in short in this section.

4.7.1 Agent Creator

The agent creator is separated from the MAS, because it is only used to start the MAS and its agents. The agent creator is always started first and creates and starts all the other agents in the MAS.

The agents are started in the following order:

- Time agent
- EH agent
- Weather agent
- Gencos
- Prosumers
- Consumers
- Balancer agent:

After this agent creator is done and terminates itself, it does not participate in MAS.

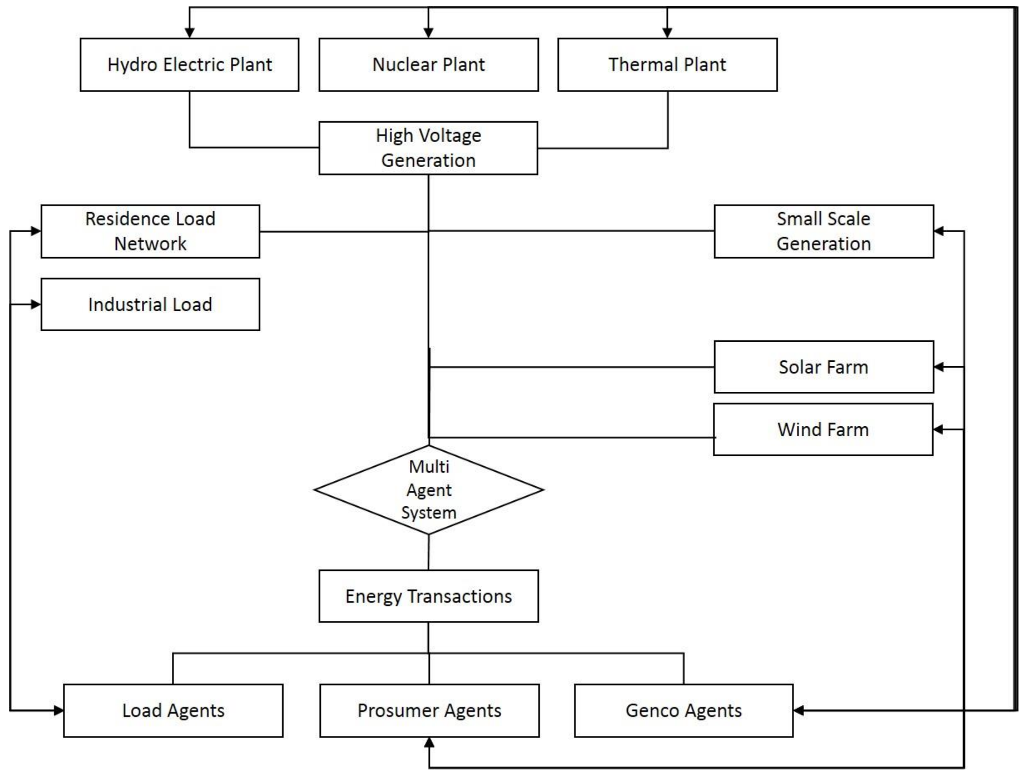


Figure 4.3: Flow diagram showing overview of Smart Grid Energy Trading using MAS

4.7.2 Time Agent

The time agent only interacts with the other top-level intermediaries, consisting of the Balancer agent and the weather agent. It waits for incoming requests from other agents and replies the current time of day. It does not contact other agents by itself. This time of day is based on the current system time of the host. The day is divided into 6 time slices, ranging 0 to 5.

4.7.3 Weather Agent

The weather agent interacts with most of the other agents, except for the Consumers and the Gencos. The first task of the weather agent is to ask for the current time of day from the time agents. The second task is to wait for incoming requests from other agents, asking for a weather forecast, and reply with the current forecast. The weather forecast calculation is based on the current time of day. The forecast consists of three factors, temperature, solar power and wind power. The weather agent also listens for incoming messages from the EH agent containing the failed agents name and can send messages to the EH agent with the name of a failed agent.

4.7.4 Balancer Agent

The Balancer agent interacts with all the other agents in the system. The GUI is initialized first before any other communication actions are taken. The GUI is only started with the first Balancer agent, the other Balancer agents do not display a GUI, in order to increase performance. After this the first step is to search for all the Suppliers, Genco or Prosumer, and inform them that they can send their name and position and energy production to the Balancer. The next step is to search for all the Consumers and inform them that they can send their name, area and energy demand to the Balancer.

The third step is to ask each weather agent for the forecast in their area and store it. Each weather forecast is bound to a different area and is displayed in the GUI. Also in this step, the total energy demand is calculated and used to balance the demand and supply, by sending a production threshold to each Genco. This step is concluded by informing all the agents that the negotiating can start.

The fourth step is to wait for messages from Consumers that have stipulated a contract with a supplier and update the GUI by showing a direct link between the Consumer and the supplier. Once every Consumer has established a contract, the next step is initiated. The fifth step is to update the GUI with a graph containing selling prices and expectations for each Consumer. Restart messages are sent to all the agents if the current time of day is below 6. If this is not the case, the next step is started.

The last step is used to send a kill signal to all the other agents, so only the Balancer is still active. The Balancer agent also listens for incoming messages from the EH agent containing a failed agents' name. It can also send a message with an agent name to the EH agent in case of an agent failure.

The failed agents are stored and displayed in the GUI. Depending on the agent type, certain additional actions are taken. If the failed agent was a Prosumer, the Balancer sends a special restart message to the newly started Prosumer containing the remaining production capacity. If the agent was a Genco, the Balancer first re-sends the request for information from the Genco and then resends the remaining production threshold to the Genco. If the agent was a Consumer, the Balancer re-sends the request for information message and then resend the start message.

4.7.5 Genco Agent

The Gencos interact with most of the other agents in the MAS, except for the time agent, the weather agent and the Prosumers. The first step is to wait for a message from each Balancer, requesting the name, area and energy production and send a reply with this information. The next step is to wait for a message from a Balancer containing the production threshold. The last step is to wait for incoming contracting requests from Consumers, a restart message from a Balancer or a kill message from a Balancer. The contracting requests from Consumers are replied with a proposal price based on the distance between the Genco and the Consumer or a message indicating that the Genco has sold all available energy if the Genco has reached its production threshold. The Genco can only send messages to the EH agent with the name of a failed agent. It is not able to receive messages from the EH agent, because it does not contact agents by itself and does not need to update the agents.

4.7.6 Prosumer

The Prosumers interact with most of the other agents in the system, except for the time agent and the Gencos. The first step is to ask for a weather forecast from every weather agent, but only the weather forecast of the weather agent that is within the same area as the Prosumer is used. In the second step the Prosumer checks if it has received a special restart message from a Balancer agent, containing the remaining production capacity. If this is the case, this production capacity is used instead of the normal capacity, which is calculated with the weather forecast. After this the Prosumer waits for a message from each Balancer, requesting the name, area and energy production and sends a reply with this information.

The next step is to wait for incoming messages from Consumers containing their area and energy demand. If the Prosumer has energy left, it replies with a proposed price, which is based on a starting price and the distance to the Consumer. When a Consumer cancels the negotiations, the Consumers' offer is removed. When an offer from a Consumer is received, the Prosumer proceeds to the next step.

The fourth step is to elaborate all the offers that it has received and find the best offer. The other offers are refused and the best offer is only accepted if it is higher than the expected earnings for the Prosumer. If the Prosumer still has energy left, it returns to step three, otherwise it moves on to the final step. The final step is to refuse all offers that are still present and all incoming offers. After this it waits for a restart message from a Balancer or a kill message from a Balancer. The Prosumer also listens for incoming messages from the EH agent containing the failed agents name and can send messages to the EH agent with the name of a failed agent.

4.7.7 Consumer Agent

The Consumers interact with the Gencos, the Prosumers and the Balancer agents. The first step is to wait for a message from each Balancer, requesting the name, area and energy demand and send a reply with this information. The second step is to wait for a start message from one Balancer, after this the negotiations can start. The third step is to search for all the suppliers and contact the Prosumers first. A message is sent to each Prosumer containing the area and the energy demand of the Consumer. It then waits for each Prosumer to reply with a proposal containing a proposed price. When all the Prosumers have replied, the Consumer moves to the next step.

The fourth step is to send a message to all the Gencos, containing the area and energy demand of the Consumer. It then waits for each Genco to reply with its area. The nearest Genco is then selected and saved. The next step is to remove the Prosumers that do not produce enough energy for the Consumers' demand. If no Prosumers remain, the Consumer goes to step eight.

The sixth step is to send new offers to the Prosumer until it accepts the offer or the amount of offers exceeds a set limit. The offers are raised with a certain amount each time. If the Prosumer accepts the offer, the Consumer moves to step seven. If the Prosumer cancels the negotiations, the Consumer removes the Prosumer and returns to step five to find a new cheapest Prosumer. If the Consumer receives a message from the Balancer agent, indicating that the auction round time is up, it sends a contracting message to the nearest Genco containing the area and the energy demand and then moves to step seven.

The seventh step is to wait for an incoming offer from the nearest Genco containing its price and reply that the offer is accepted with the Consumers area and demand. But only if no contract has been established yet. The contract details are then sent to every Balancer agent containing the Consumers name, the suppliers name and the total price divided by the demand. The Consumer then waits for a restart or a kill message from a Balancer agent. The eighth step is to send a message to every Genco containing the Consumers area and demand and wait for replies from the Gencos containing their price. The cheapest Genco is then selected and an accept message is sent to that Genco.

The contract details are sent to every Balancer agent containing the Consumers name, the suppliers name, the expected costs and the total price divided by the demand and the Consumers goes back to step seven. If the Gencos do not answer or if the auction round time is up, the Consumer contacts the nearest Genco with area and demand and goes back to step seven. The Consumer also listens for incoming messages from the EH agent containing the failed agents name and can send messages to the EH agent with the name of a failed agent.

4.7.8 EH Agent

The EH agent is not very complex, but does communicate with almost all the other agents in the system, except for the time agent. This is because the time agent does not contact any agents by itself, it only waits for requests from other agents. Therefore it cannot detect failed agents and does not need updates on the failed agents. The EH agent has a cyclic behavior, which is repeated until the agent terminates. It first searches for all the Balancer agents, Consumers, Prosumers and weather agents. Then it waits for an incoming message and checks if it is a kill message, in which case it terminates, or if it is a message containing a failed agent. The agent attempts to kill the failed agent if not already dead and restarts the agent with a different name, by adding “-r”. Depending on the failed agents' type, other agents are informed of the failed agent.

The following agents are informed in case of these failures:

- Time agent fails: Balancer agents, Consumers, Prosumers and weather agents are informed
- Weather agent fails: Balancer agents and Prosumers are informed
- Balancer agent fails: Consumers and Prosumers are informed
- Prosumer fails: Balancer agents and Consumers are informed
- Genco fails: Balancer agents and Consumers are informed
- Consumer fails: Balancer agents are informed

After this, the process is repeated.

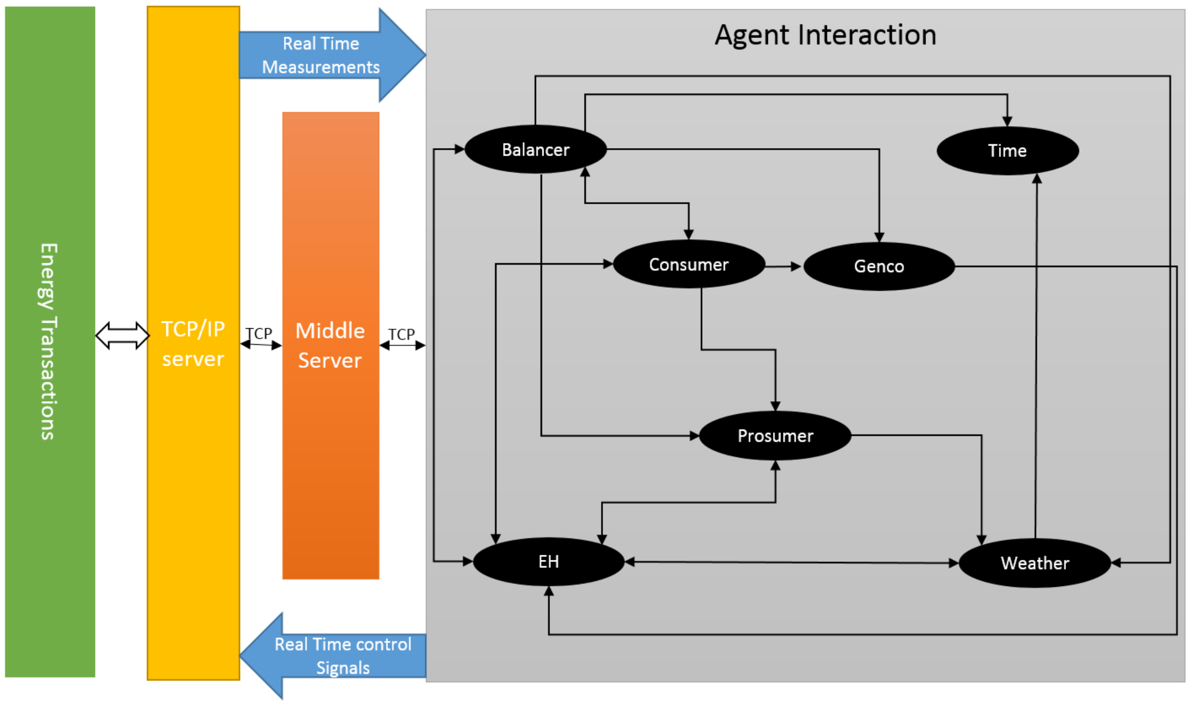


Figure 4.4: High Level Agent Interaction

CHAPTER 5

SMART GRID ENERGY TRADING USING MAS

5.1 IMPLEMENTATION

5.1.1 Agent Creator

This agent uses a single one shot behavior, because it only runs once. In this behavior the agent first starts a Time agent, by calling: *createNewAgent("ta"+containerNr, "simulation.TimeAgent", null).* After this a Weather agent and in case the Agent Creator is located in container one, an EH agent. After this an iteration is started, executing 30 times and creating 30 Consumers, 7 Prosumers and 3 Gencos. The name of the Consumers starts with a 'c', the Prosumers with a 'p' and the Gencos with a 'g' each followed by the current iteration number. After this the Balancer agent is started and in case the Agent Creator is located in container one, a “GUI” argument is also sent to the Balancer to enable the GUI on this Balancer. Finally the Agent Creator calls *doDelete()* to kill itself.

5.1.2 Time Agent

This agent uses the GC behavior and a Cyclic Behaviour, because it is constantly checking for incoming messages by calling *myAgent.receive().* If there is an INFORMATIVE type message, the agent retrieves the system date and time and uses the current seconds and divides this by 10. This results in a number between 0 and 5 indicating the time of day. This number is replied to the sender of the message. If the message was a “DIE” message, the agent calls *doDelete()* to kill itself.

5.1.3 Weather agent

This agent uses the GC behavior, the EH behavior and a standard behavior. In this class there is a switch over 6 cases, based on the current time of day from the Time agent. If the time is 0, the Weather agent first searches the DF for all the Time agents by calling *DFService.search(myAgent, template),* where template has the type Time Agent. After this the Weather agent sends a message to every Time agent requesting the current time of day. The agent then waits for a reply containing the time of day, by calling *myAgent.blockingReceive().* For each time of day different random weather is generated, for instance, the temperature and solar power is less in the morning than in the afternoon. The Weather agent checks for incoming INFORMATIVE type messages via *myAgent.blockingReceive(),* containing the "WF" string, requesting the weather and sends back the temperature, solar power and wind power. In case of a “DIE” message, the agent calls *doDelete()* to kill itself.

5.1.4 EH Agent

This agent uses the GC behavior and has a cyclic behavior. The agent first searches the DF for all the Consumers, Prosumers, Gencos, Balancer agents and Weather agents. It then checks for incoming FAILURE type messages containing the name of the failed agent, with *myAgent.receive().* The failed agent is killed via *ac.kill(),* if not already dead and restarted with an added "-r" to its name. The new agent's name is only sent to the agents that need this information, depending on the type of the agent. In case of a "DIE" message, the agent calls *doDelete()* to kill itself.

5.1.5 Genco Agent

This agent uses the GC behavior, the EH behavior and a standard behavior. In this class there is a switch over 4 cases.

Step 0: The Genco first searches the DF for the EH agent, by calling *DFService.search(myAgent, template),* where template has the type EH Agent. After this the agent moves to step 1.

Step 1: The Genco waits for an "IN" signal sent by a Balancer agent. If this has been received, the Genco searches the DF for all the Balancer agents and sends each one a message with content: *msg1.setContent("G" + position + " " + threshold),* where position is the area the Genco resides in and threshold is the maximum production threshold of the Genco. After this the agent moves to step 2. If a "DIE" signal was received, the agent calls *doDelete()* to kill itself.

Step 2: The Genco waits for a message containing the suggested production limit. If this has been received the Genco adds a random number between 0 and 5 to the production limit and moves to step 3. If a "DIE" signal was received, the agent calls *doDelete()* to kill itself.

Step 3: The Genco waits for different incoming messages. First INFORM type messages from Consumers, containing their position and demand can be received. This position is then used in combination with the demand to generate a proposed price. A reply is then sent to the Consumer containing: *reply.setContent(name + " " + proposed_price + " " + n_threshold),* where name is the name of the Genco, *proposed_pric*e is the proposed price and *n_threshold* is the threshold for the Genco. Also ACCEPT_PROPOSAL type messages from Consumers, containing the demand can be received. The demand is subtracted from the production threshold of the Genco. Another message type that can be received, is the SUBSCRIBE from Consumers. The Genco replies to this message with its name and position. If a "DIE" signal was received, the agent calls *doDelete()* to kill itself.

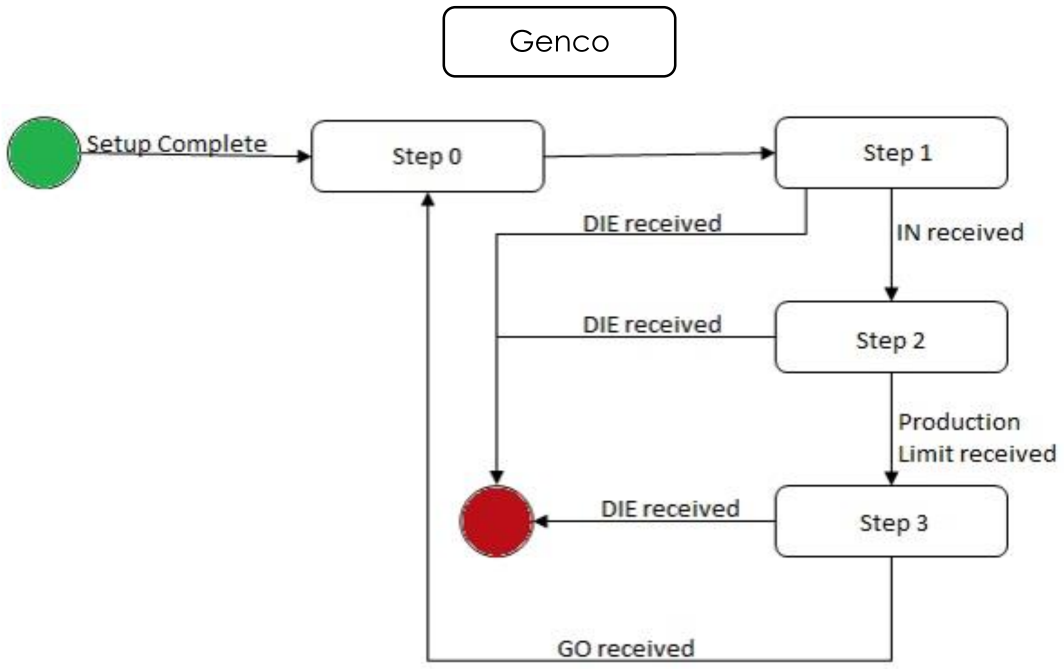


Figure 5.1: Flowchart showing Genco Interaction & Behavior

5.1.6 Prosumer agent

This agent uses the GC behavior, the EH behavior and a standard behavior. In this class there is a switch over 6 cases, based on the current step.

Step 0: The Prosumer first searches the DF for the Weather agents and the EH agent, by calling *DFService.search(myAgent, template),* where template has the type EH Agent or Weather Agent. Of the found Weather agents, the agent that has the same number as the container number where the Prosumer resides in, is stored. This represents the area of the agents. After this the agent moves to step 1.

Step 1: The Prosumer sends a message, containing "WF", to the Weather agent requesting a weather forecast. The Prosumer then waits for a few different message types. The first is a reply from the Weather agent containing a temperature, wind power and solar power. After this the Prosumer moves to step 2. The second message type is a message from a Consumer requesting negotiations. The Prosumer replies with a CANCEL message containing "NO". The third type is an "IN" or "RESTART" message from a Balancer, indicating that the negotiations have started. A "RESTART" message also contains a remaining supply. The supply is stored, as well as a Boolean indicating the received message. The last type is a "DIE" message, where the Prosumer will call *doDelete()* to kill itself.

Step 2: The Prosumer will calculate the new supply from the weather forecast with*: supply = (int) (supply * constant) + 1,* where supply is the solar or wind power and constant is a random number between 0 and 1. The agent now waits for an "IN" or "RESTART" message if not already received in step 1. If this message is received, the Prosumer searches the DF for all the Balancer agents. If it was an "IN" message, it sends a message to each Balancer, containing: *msg1.setContent("P" + position + " " + supply),* where position is the area that the Prosumer resides in and supply is the total production limit of the Prosumer. If it was a "RESTART" message, the Prosumer takes the remaining supply from the message and stores it as supply. After this the Prosumer moves to step 3. If a "DIE" message was received, the Prosumer calls *doDelete()* to kill itself. If any other message was received, the Prosumer sends a "MOV" message to every Balancer, to indicate that it is waiting.

Step 3: The Prosumer searches the DF for every Consumer, but only if this is the first run of step 3. Next the Prosumer checks for a few different message types. If an INFORM message is received from a Consumer, containing its area, the Prosumer calculates a proposed price with: *float proposed_price = (float) (((Math.abs(b_pos - position) + 1) * c_TSO) + starting_price),* where b_pos is the area of the Consumer, position is the area of the Prosumer, c_TSO is a random float constant and starting_price is the starting price of the Prosumer. If the Prosumer still has some supply left, it sends a PROPOSE reply with the proposed price and its remaining supply, in the other case a CANCEL message is sent. If the amount of sent replies is larger than or equal to the amount of Consumers in the system, the Prosumer moves to step 4. If an ACCEPT_PROPOSAL message is received from a Consumer, containing its name and energy demand, the Prosumer subtracts this demand from its supply. If a PROPOSE message is received, containing an offer from a Consumer, the Prosumer stores the offer and moves to step 4. If a "DIE" message is received, the Prosumer calls *doDelete()* to kill itself.

Step 4: The Prosumer first selects the best offer from the list of offers. Then the Prosumer sends every agent from the offers list an REFUSE message, containing "NO", except for the best offer. If this offer is less than expected and the number of refusals is smaller than the maximum, the best offer is refused as well. Otherwise, this agents gets an ACCEPT_PROPOSAL message containing "YS". If after this the Prosumer has no supply left, it sends a CANCEL message, to every Consumer from the offers list and moves to step 5. If it does have supplies left, it moves back to step 3.

Step 5: The Prosumer sends a CANCEL message once to every Consumer in the system. It then waits for incoming messages. If a "DIE" message is received, the Prosumer calls *doDelete()* to kill itself. If an offer from a Consumer is received, the Prosumer replies with an CANCEL message containing "NO".

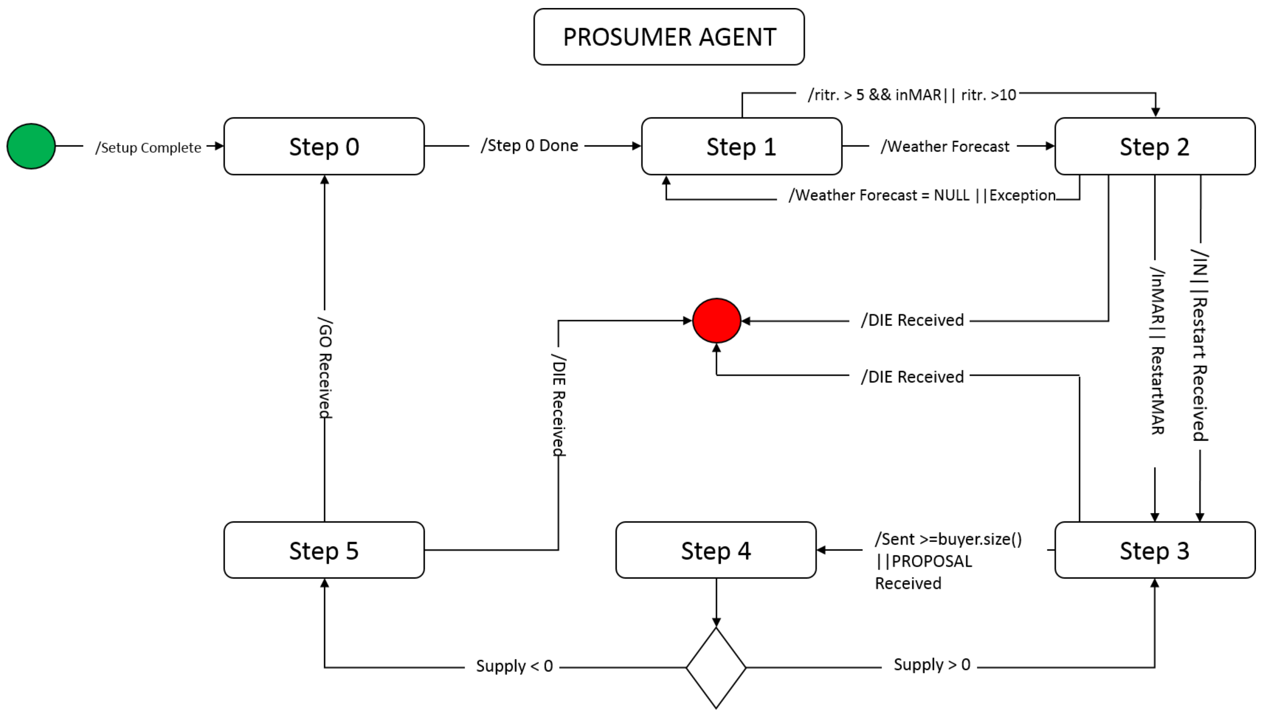


Figure 5.2: Flowchart showing Prosumer Interaction & Behavior

5.1.7 Consumer Agent

This agent uses the GC behavior, the EH behavior and a standard Behavior. In this class there is a switch over 8 cases, based on the current step.

Step 0: The Consumer first searches the DF for the EH Agent once. It then waits for incoming messages. If an “IN” message is received from a Balancer agent, the Consumer searches the DF for every Balancer agent and sends a message to each Balancer agent, containing *msg1.setContent("B" + position + " " + p_demand)*, where position is the area that the Consumer resides in and p_demand is the energy demand of the Consumer. It then moves to step 1. If a "DIE" message is received, the Consumer kills itself. If another message is received, the Consumer sends a "MOV" message to every Balancer agent.

Step 1: The Consumer waits for a "GO" message from a Balancer, after which it moves to step 2.

Step 2: The Consumer searches the DF for every "Seller" type agent once, this includes both the Prosumers and Gencos. These agents are stored in seller agents and a message containing the area and energy demand is sent to every Prosumer once. The Consumer then waits for incoming messages. If a PROPOSE message is received, containing an offer from a Prosumer, the offer is stored. If an offer is received from every Prosumer, the agent moves to step 3. If a CANCEL message is received from a Prosumer, this agent is removed from the list of Prosumers. If step 2 had been run more than twice amount of Prosumers and there is atleast one offer, the Consumer also moves to step 3.

Step 3: The Consumer first sends a message to every Genco once, containing its area and energy demand. It then listens for incoming messages. If a message is received from a Genco containing its name and area, the Consumer calculates the distance between the Genco and itself. Only the Genco that has the smallest distance is selected as nearest Genco. If a message has been received from every Genco, the Consumer moves to step 4. If a message is received from a Prosumer, containing "NO". The Prosumer is removed from the list of Prosumers. If no message is received for 5 times and there is already a nearest Genco selected, the Consumer also moves to step 4, otherwise it re-sends a message to every Genco, containing its area and energy demand.

Step 4: If the Consumer is not forced to contact a Genco, the Consumer checks the supply of each Prosumer and removes those that do not have enough supply for the demand of the Consumer. Of the other Prosumers, the one with the lowest price will be selected as best Seller. After this it calculates the expected price to pay for the power demand: ((float) bestPrice + (bestPrice * ((g.nextInt(20) + 1) / 100))) / demand. After this, or if the Consumer is forced to contact a Genco, the Consumer moves to step 5, or if no best Seller is found, the Consumer moves to step 7.

Step 5: The Consumer sends a PROPOSE message to the best Seller once, containing a bid of height stake and its energy demand. If after 10 runs no reply has been received from the best Seller, the Consumer deletes the Prosumer from the list and returns to step 4. After this the Consumer checks for incoming messages. If an ACCEPT_PROPOSAL message is received from a Prosumer, the Consumer replies with an ACCEPT_PROPOSAL message containing its demand and moves to step 6. If an REFUSE message is received from a Prosumer, the Consumer replies with a new PROPOSE message and a higher bid, increased with a random constant stake. If too many refusals have been received, the Consumer removes the best Seller from the list and replies with a CANCEL message, containing "NO", and moves back to step 4.

If a CANCEL message is received from a Prosumer, the Consumer removes the best Seller from the list and returns to step 4. If a SUBSCRIBE message is received from a Balancer agent, the Consumer sends an INFORM message to the nearest Genco, containing its area and demand. After this it moves to step 6. If an "IN" message is received from a Balancer agent, the Consumer replies its area and demand and moves back to step 3.

Step 6: If a contract was established with a Prosumer or Genco, the Consumer sends a message to all the Balancer agents containing the contract details: name + " " + bestSeller + " " + expected + "X" + ((bestPrice + new_stake) / demand) + "Y" + demand). If no contract was established with a Genco and the Consumer is forced, it sends an INFORM message to the nearest Genco, with its area and energy demand. After this the agent waits for incoming messages. If a "DIE" message is received, the agent kills itself. If a PROPOSE message is received from the nearest Genco and the demand is still above 0, the Consumer stores the nearest Genco in bestSellerG.

An ACCEPT_PROPOSAL message is sent to this Genco with the Consumers area and demand and an INFORM_REF message is sent to every Balancer with the contract details. Demand is set to 0 and forced is now false. If an REQUEST_WHEN message is received from a Balancer agent, the Consumer waits for 1.5 seconds and then re-sends a reply with the contract details. If an ACCEPT_PROPOSAL message is received from a Seller, the Consumer sends back a CANCEL message.

Step 7: The Consumer clears the message queue once and if no contract is forced and established, sends an INFORM message to every Genco once, containing area and demand. If after 5 runs, no Genco has replied, the Consumer sends a message to the nearest Genco, containing area and demand and sets forced to true and moves to step 6. After this the Consumer waits for incoming messages. If a PROPOSE message is received from a Genco, the Consumer checks if this Genco is the cheapest, in which case it is stored. The Balancer agents are informed of the contract details via an INFORM_REF message and demand is set to 0. After this the agent moves back to step 6. If a SUBSCRIBE message is received from a Balancer agent, the Consumer sends an INFORM message to the nearest Genco containing area and demand, and moves back to step 6.If an ACCEPT_PROPOSAL message is received from a Prosumer, the Consumer replies with a CANCEL message. If any other message is received, the Consumer re-sends the contract details to all the Balancer agents.

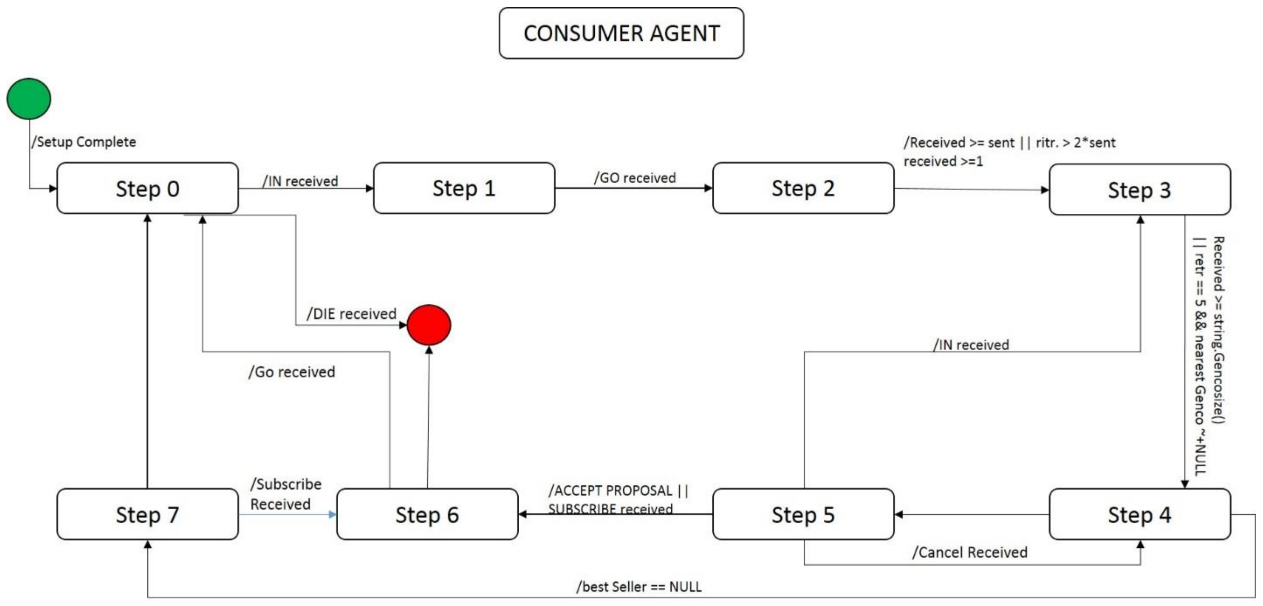


Figure 5.3: Flowchart showing Consumer Interaction & Behavior

5.1.8 Balancer agent

This agent uses the GC behavior, the EH behavior and a standard Behaviour. In the *setup()* method of the Balancer agent, the agent waits for a while before starting, this is done make sure that all the Balancer agents start at the same time. The agent waits for the next exact half minute before starting: *(Math.ceil((double)oldtime / 30000)*30000)*, where old time is the current time. After this the Balancer moves to the behavior, where there is a switch over 7 cases.

Step 0: The Balancer searches the DF for all the Weather agents, Time agents, the EH agent and other Balancer agents. It then moves to step 1.

Step 1: The Balancer searches the DF for every Seller once, which includes Prosumers and Gencos. Each Seller is sent an INFORM message once, containing "IN". The Balancer then waits for incoming messages. If an INFORM message from a Genco or a Prosumer is received containing their area and name, the Balancer stores this agent in Gencos or Prosumers. Also the supply and name of each Prosumer is stored in agentSupplies. After this the agent is drawn in the GUI, by calling: *myGUI.drawAgent(t_position, sa_name, agent_count),* with area, agent name and agent count. If a reply has been received from each Genco and Prosumer, the Balancer moves to step 2. If any other message is received, the Balancer sends a reply containing "OK". If no messages are received for 9 runs, the Balancer re-sends the messages to the Sellers.

Step 2: The Balancer searches the DF for every Consumer once. Each Consumer is sent an INFORM message once, containing "IN". The Balancer then waits for incoming messages. If an INFORM from a Consumer containing its area and demand, the Balancer stores this agent in buyers. After this the agent is drawn in the GUI, by calling: *myGUI.drawAgent(t_position, sa_name, agent_count),* with area, agent name and agent count. If a reply has been received from each Consumer, the Balancer moves to step 3. If any other message is received, the Balancer sends a reply containing "IN".

Step 3: The Balancer calculates the total energy demand of all the Consumers by adding their individual demands. The average needed energy production per Genco and per Prosumer is calculated, by: *Sg = (int) ((demand[0] * kg) / (gencos.size())) + 1* and *Sp = (int) ((demand[0] * kp) / (prosumers.size())) + 1.* For each Prosumer the stored energy production and needed production are compared and these differences are summed up over all the Prosumers. This so called deficit is divided by the number of Gencos and added up to the needed energy production of the Gencos: *deficit = deficit / gencos.size(); Sg += deficit*. Each Genco is then sent an INFORM message, containing this needed energy production, or Sg. It is also added to the agentSupplies list together with the agent name.

After this the Balancer sends a message containing "WF", to every Weather agent. Next every Consumer is sent a message containing "GO". The Balancer now enters a loop, in which it waits for all the incoming weather forecasts. If a message containing a weather forecast is received, the Balancer uses the weather forecast and the area of the Weather agent and displays it in the GUI: *myGUI.add(myGUI.new DrawTemp(temperature, solar, wind, area)).* If an INFORM_REF message is already received from a Consumer containing contract details, the Balancer replies with an REQUEST_WHEN message containing "WT". If other or no messages are received for 5 runs, the Balancer re-sends the messages to the Weather agents. After the loop the Balancer moves to step 4.

Step 4: The Balancer waits for incoming messages. If an INFORM_REF message is received from a Consumer, containing contract details, the Balancer stores the contract details from the message and replies with an INFORM_REF message "OK". The Balancer then checks whether the received contract was already received earlier on, in which case the contract details are discarded. The next check is to make sure that a Consumer only has one contract, if this is not the case, the contract details are also discarded. If the contract is not discarded, the details are stored in an object: *StatContractB.* These objects are added to the GUI and the GUI is refreshed: *myGUI.statsB.addElement(scb); myGUI.refreshGUI(t1, t2).* The list agent Supplies is updated by subtracting the Consumers' demand from the total energy production of the Genco or Prosumer. If all of the Consumers have sent their contract details, the Balancer moves to step 5.

Step 5: The Balancer adds a MouseListener and the drawSkeletonGraph to the GUI. After this the Balancer waits 2 seconds and then moves to step 6.

Step 6: The Balancer sends a "DIE" message to every agent in the system. It then waits indefinitely, until it is manually killed.

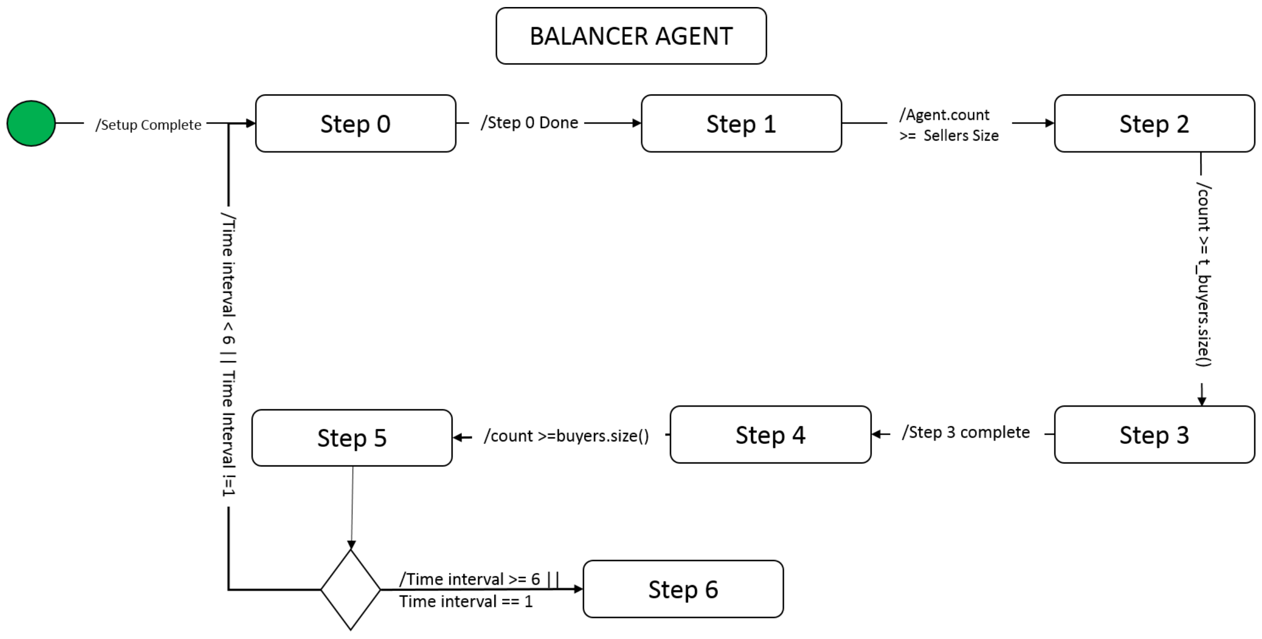

BALANCER AGENT
/Setup Complete
Step 0
/Step 0 Done
Step 1
/Agent.count >= Sellers Size
Step 2
/count >= t_buyers.size()
Step 3
/Step 3 complete
Step 4
/count >=buyers.size()
Step 5
/Time interval < 6 || Time Interval !=1
/Time interval >= 6 || Time interval == 1
Step 6

Figure 5.4: Flowchart showing Balancer Interaction & Behavior

5.1.9 GUI class

The GUI class contains all the GUI related code. It is created by the Balancer agent, but only on the first host of the MAS. The GUI class contains a number of methods, some of which are called by the Balancer agent. Some of the important methods are listed here together with their purpose:

- *public GUI():* Called by the Balancer agent. In this constructor the Jframe of 1150 by 800 is created and displayed, a MyCanvas is created and all the Vectors are initialized.
- *public void clear():* Called by the Balancer agent. This method is used to fully clear and repaint the GUI, by calling *(gui.getContentPane()).removeAll() and gui.repaint().*

- *public void add(JComponent comp):* Called by the Balancer agent and the GUI itself. This method can add a new JComponent to the GUI, by calling *gui.getContentPane().add(comp)* and *gui.repaint()*.
- *public void refreshGUI(String t1, String t2):* Called by the Balancer agent. This method is used to add a contract line between agent 't1' and agent 't2'. The locations of both agents are searched using: *((GUI.Contract) contractNet.get(i)).getPosx(t1)*. The locations are used to create a new QuadPoints line: new *QuadPoints(px, py, pxx, pyy)*, which is added to a lineContract Vector.
- *public void drawAgent(int pos, String name, int ac):* Called by the Balancer agent. This method is used to add agent 'name' in area 'pos' on the GUI. 'ac' is the current agent count and is used to find the correct data in the Vectors. A Consumer is magenta, a Supplier is blue and a Genco is red. A switch is used depending on the area of the agent, to generate a random position within the area: *generator.nextInt(Wstep) + CX + 5*. A new Contract object is made using this data: new *Contract(x.get(agent_count), y.get(agent_count), gui_string.get(agent_count))*, and is added to the Vector contractNet. After this the GUI is repainted.

There are also some other classes located within the GUI class. Most of these classes are JComponent classes and are added to the GUI, using the add(JComponent comp) method.

These JComponent classes each have a paint(Graphics g) method, which is called upon repaint and handles the drawing. These classes are listed below:

- *public class DrawTemp*: This class is created by the Balancer agent and is used to draw the information of each Weather agent in the GUI. The Balancer agent calls the constructor, public DrawTemp(int t, int s, int w, int c), where 't' is the temperature, 's' is the solar power, 'w' is the wind power and 'c' is the count of the current area.
- *public class DrawDeadAgents*: This class is created by the Balancer agent and is used to draw the names of the dead agents in the GUI. The Balancer agent calls the constructor, private *DrawDeadAgents()* and also the method private

void *updateDeadAgents(Vector<String> failedAgents)*. This method is used to update the Vector deadAgents.

- *public class drawSkeletonGraph:* This class is created by the GUI and is used to draw the empty graph and its axis onto the GUI.
- *public class drawLineStat:* This class is created by the GUI and is used to draw the graph of a selected agent, with expected price in red and real price in black.This final set of classes is used to store information on the agents position and the contract lines and to handle the mouse clicks:
- *public class Contract:* This class is created by the GUI and is used to store the agents in the system. It has a 'posx', 'posy' and 'agentName' as data fields, to store agent position and name.
- *public class QuadPoints*: This class is created by the GUI and is used to store the contract lines in the system. It has a 'posx1', 'posy1', 'posx2', 'posy2' as data fields, to store beginning and end positions.
- *class MyMouseListener extends MouseAdapter*: This class is created by the Balancer agent after the simulation has finished and is used to listen for mouse clicks. If public *void mouseClicked(MouseEvent evt)* is called, the 'x' and 'y' position are used in *((Contract) contractNet.get(i)).retrieveName(x, y)* to retrieve the agent that was selected. After this, public void *drawGraph(String n)* is called with the agent name 'n'. *(StatContractB) statsB.get(i)* is used to get the contract details and *gui.getContentPane().add(new drawLineStat())* is called to add the graph.

5.2 EVALUATION

5.2.1 Testing System

The test cases were evaluated on a used 10 MBPS Ethernet network of the following machine:

- Dell Vostro 1014
- Intel Core 2 Duo CPU @ 2.33GHz
- 2Gbyte RAM
- 160Gbyte disk
- MSI 8600GT PCI-EX 256MB DRR low profile

Installed software:

- Debian GNU/Linux 10.04, kernel 2.6.32-5-686
- Eclipse version 3.5.2 Java EE IDE for Web Developers
- JADE version 4.1
- JRE version 1.6.0.24 with just-in-time compiler enabled

5.2.2 Scalability Test Cases

The first scalability test case focuses on how many agents the MAS can handle before it becomes unstable. The MAS by N. Capodieci (old MAS) and the new MAS are compared in performance.

Test Case 1:

Table 5.1: Scalability Test Case-1

How many agents can the MAS handle before it becomes unstable?		
How to	Increase the amount of agents on the old and the new MAS until it becomes unstable.	
	MAS1	**MAS2**
Type	**Normal MAS**	**MAS – N.Capodieci**
Test	**6 Hosts**	**1 Host**
Metrics: • Number of agents in the system • Number of hosts • Total number of messages transferred between agents • Number of ticks that have passed to reach convergence • CPU and memory usage		

The old MAS is run on one host machine and the new MAS on six hosts. Measurements have been performed on the number of messages transferred, the time to reach convergence and the average CPU and memory usage of the system and JADE.

Results:

Table 5.2: Results of Scalability Test-1

How many agents can the MAS handle before it becomes unstable?		
Metrics	**Test1, new MAS**	**Test 1, old MAS**
Type	**Normal MAS**	**MAS – N.Capodieci**
Number of Hosts	6	**1**
Number of Agents	818(600c, 140p, 60g, 6b,6t, 6w)*	243(180c, 42p, 18g, 1b,1t, 1w)*
Average number of messages transferred between agents	2865639	65439
Average time to reach convergence/Number of ticks that have passed	519800 ms	1879993 ms
Average CPU and memory usage	46%, 566MB	39%, 667MB
Average CPU and memory usage by JADE	13%, 230MB	40%, 100MB

**Notation: c – consumer; p – prosumer; g – genco; b – balancer; t – time; w – weather;*

In the Test case 1, the limit of the new MAS is around 818 agents, consisting of 600 Consumers, 140 Prosumers, 60 Gencos, 6 Balancer agents, 6 Weather agents and 6 Time agents. The simulation only takes about 8 minutes, but it creates a high number of 2865539 messages. This is simply caused by the amount of agents that participate in the system. Because of this amount of messages, the system sometimes loses an important message, which can cause the system to halt. This problem increases as more agent are added, but around 818 agents the system still mostly works correctly.

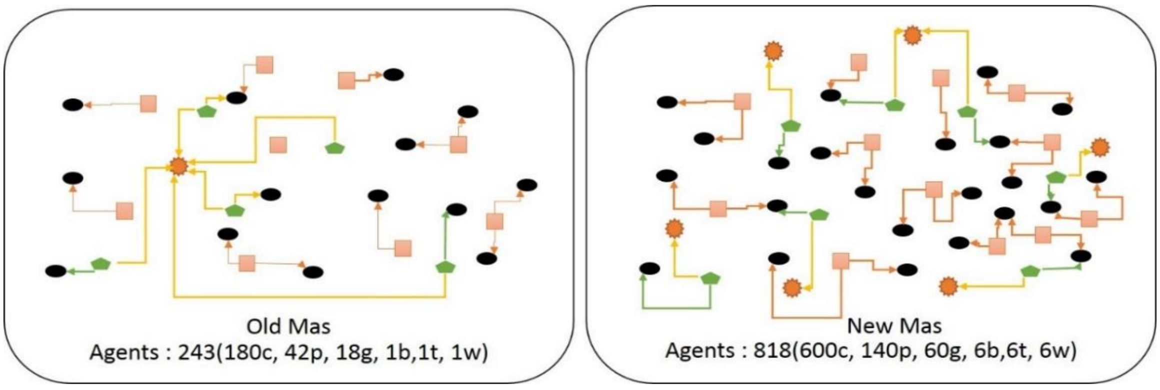

Old Mas
Agents : 243(180c, 42p, 18g, 1b,1t, 1w)
New Mas
Agents : 818(600c, 140p, 60g, 6b,6t, 6w)

Figure 5.5: Single line representation of Number of Agents which led to a stable MAS

As the agents are spread across 6 hosts, The CPU usage of the MAS is not very high. The JADE CPU usage is remarkably low, which is probably caused by the fact that the MAS has to do much more work than JADE. The memory usage of the MAS and JADE is pretty high, because of the amount of agents.

The limit of the old MAS is around 243 agents, consisting of 180 Consumers, 42 Prosumers, 18 Gencos, 1 Balancer, 1 Weather agent and 1 Time agent. With this amount of agents the system crashes after a while, due to the amount of RAM used, which is 667MB for the system alone. The amount of memory exceeds the Java heap space and the system crashes. Also the run-time is very long, around 31 minutes, which is much longer than the new MAS. The amount of messages is lower than for the new MAS, but this is caused by the0 much smaller amount of agents. The CPU usage of JADE is much higher than the new MAS, because the MAS is doing less work. The memory usage of JADE is much lower, which is also caused by the lower amount of agents.

It is clear that the new MAS is capable of handling about four times as many agents as the old MAS. In this case the amount of hosts was six, but in principle the system can use much more hosts and therefore be able to handle more agents. At the same time, the time to reach convergence is about four times lower on the new MAS than on the old MAS. The limit to the scalability of the system is therefore better on the new MAS than on the old MAS.

Test Case 2:

Table 5.3: Scalability Test Case- 2

What is the scalability of the new MAS compared to the MAS by N. Capodieci, in terms of performance per load?		
How to	Increase amount of Consumer and supplier agents on the old and the new MAS	
	MAS1	**MAS2**
Type	**Normal MAS**	**MAS – N.Capodieci**
Test1	6 hosts, 58 agents (30c, 7p, 3g, 6b, 6t,6w)	1 host, 43 agents (30c, 7p, 3g, 1b, 1t, 1w)
Test2	6 hosts, 98 agents (60c, 14p, 6g, 6b, 6t, 6w)	1 host, 83 agents (60c, 14p, 6g, 1b, 1t, 1w)
Test3	6 hosts, 178 agents (120c, 28p, 12g, 6b, 6t, 6w)	1 host, 163 agents (120c, 28p, 12g, 1b, 1t,1w)
Metrics • Number of agents in the system • Number of hosts • Total number of messages transferred between agents • Number of ticks that have passed to reach convergence • CPU and memory usage		

**Notation: c – consumer; p – prosumer; g – genco; b – balancer; t – time; w – weather;*

Results:

Table 5.4: Results of Scalability Test- 2

Metrics	**Test 1, new MAS**	**Test 1, old MAS**	**Test 2, new MAS**	**Test 2, old MAS**	**Test 3, new MAS**	**Test 3, old MAS**
Number of hosts	6	1	6	1	6	1
Number of agents	58(30c, 7p, 3g, 6b,6t, 6w)	43(30c, 7p, 3g, 1b, 1t, 1w)	98(60c, 14p, 6g, 6b, 6t, 6w)	83(60c, 14p, 6g, 1b, 1t, 1w)	178(120c, 28p, 12g, 6b, 6t, 6w)	163(120c, 28p, 12g, 1b, 1t, 1w)
Average number of messages transferred between agents	4578	7124	7408	54578	28189	531730
Average time to reach convergence/Number of ticks that have passed	10871 ms	10486 ms	12148 ms	38076 ms	22583 ms	696450 ms
Average CPU and memory usage	42%, 190 MB	51%, 234 MB	46%, 143 MB	55%, 306 MB	42%, 244 MB	72%, 606 MB
Average CPU and memory usage by JADE	36%, 100 MB	25%, 71 MB	32%, 113 MB	27%, 77 MB	25%, 132 MB	18%, 95 MB

**Notation: c – consumer; p – prosumer; g – genco; b – balancer; t – time; w – weather; e- exception handling*

The CPU usage of both MAS is largely constant and no trend is visible in the data. The CPU usage of the new MAS is consequently 9% lower than the old MAS, indicating that the scalability of the new MAS is considerably better on this point than the old MAS.

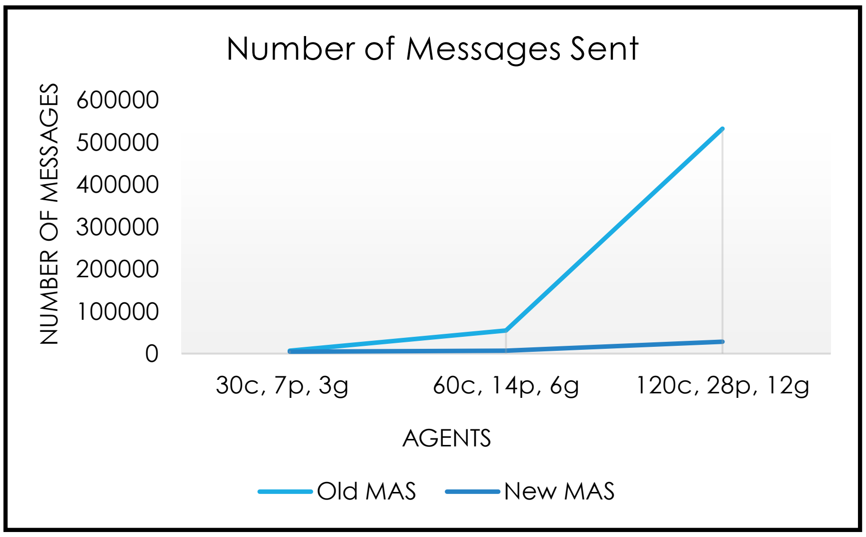


Figure 5.6: Number of Messages sent with the increase in the Number of Agents

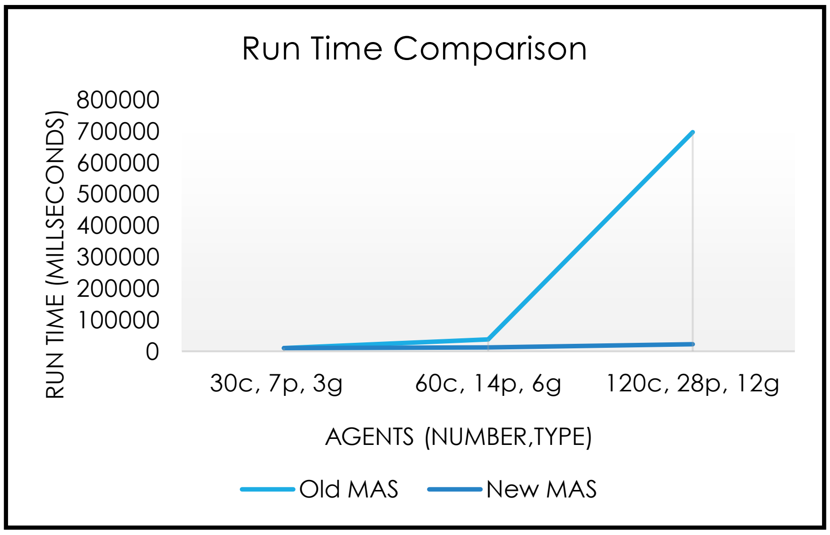


Figure 5.7: Run Time Comparison of MAS with the increase in Number of Agents

From these results, it can be concluded that the scalability of the new MAS is significantly better than the old MAS. The run-time of the old MAS is much longer than the new MAS, when the amount of agents starts to increase. Also, the amount of messages being sent in the old MAS rises much quicker than the new MAS.

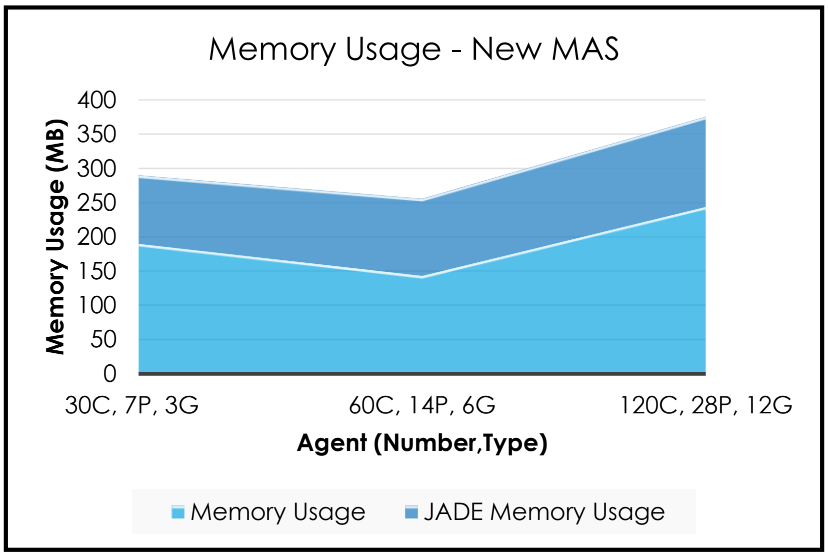


Figure 5.8: Memory Usage of New MAS with the increase in Number of Agents

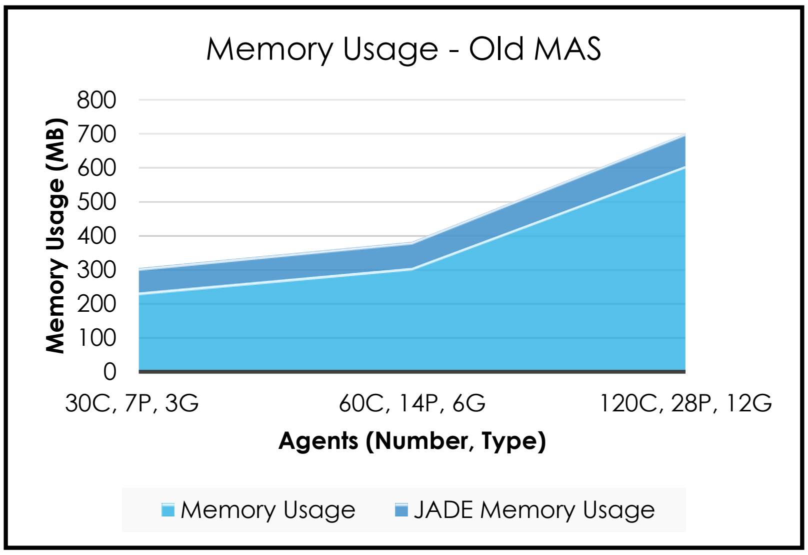


Figure 5.9: Memory Usage of Old MAS with the Increase in Number of Agents

The average time to reach convergence is shown in milliseconds, of both the new MAS and the old MAS. Three results for each MAS were obtained, for the different agent amounts. Both MAS performed similar in the first test, with the lowest amount of agents. But the second and especially the third test results are very far apart in run-time. The old MAS matches a Quadratic growth function**: y = 62.8327 x^2 + -6850.18x + 183961**. The new MAS also matches a Quadratic growth function: **y = 0.820937 x^2 + -66.5875x + 12221**, but a much lower one, as the coefficients are more than 50 times lower. The new MAS only requires 24 seconds to finish the simulation in the third test, the old MAS requires 11 minutes to do the same.

The average number of messages being sent during a simulation is shown, of both the new MAS and the old MAS. Three results for each MAS were obtained, for the different agent amounts. Both MAS performed similar in the first test, with the lowest amount of agents. But the second and especially the third test results are very far apart. The old MAS matches a Quadratic growth function: **y = 39.8171 x^2 + -3591.7x + 87084.7**. The new MAS also matches a Quadratic growth function: **y = 1.5751 x^2 + -118.263x + 6788.33**, but a much lower one, as the coefficients are more than 30 times lower.

From these results, it can be concluded that the scalability of the new MAS is significantly better than the old MAS. The run-time of the old MAS is much longer than the new MAS, when the amount of agents starts to increase. Also, the amount of messages being sent in the old MAS rises much quicker than the new MAS.

Test Case: 3

Table 5.5: Scalability Test Case- 3

What is the scalability of the new MAS with respect to performance increase per resource?	
How to	Increase the amount of hosts, while keeping the amount of agents the same
Type	**Normal MAS**
Test1	1 host, 99 agents (60c, 14p, 6g, 6b, 6t, 6w, 1e)*
Test2	3 hosts, same agent amount
Test3	6 hosts, same agent amount
Metrics • Number of agents in the system • Number of hosts • Total number of messages transferred between agents • Number of ticks that have passed to reach convergence • CPU and memory usage	

**Notation: c – consumer; p – prosumer; g – genco; b – balancer; t – time; w – weather; e- exception handling*

Results:

Table 5.6: Results of Scalability Test Case- 3

What is the scalability of the new MAS with respect to performance increase per resource?			
Metrics	**Test1**	**Test 2**	**Test 3**
Number of Hosts	1	**3**	**6**
Number of Agents	99(60c, 14p, 6g, 6b, 6t, 6w, 1e)*		
Average number of messages transferred between agents	7851	5058	5470
Average time to reach convergence/Number of ticks that have passed	17562 ms	10569 ms	17113 ms
Average CPU and memory usage	55%, 323MB	50%, 263MB	40%, 148MB
Average CPU and memory usage by JADE	22%, 111MB	32%, 118MB	39%, 115MB

**Notation: c – consumer; p – prosumer; g – genco; b – balancer; t – time; w – weather; e – exception handling*

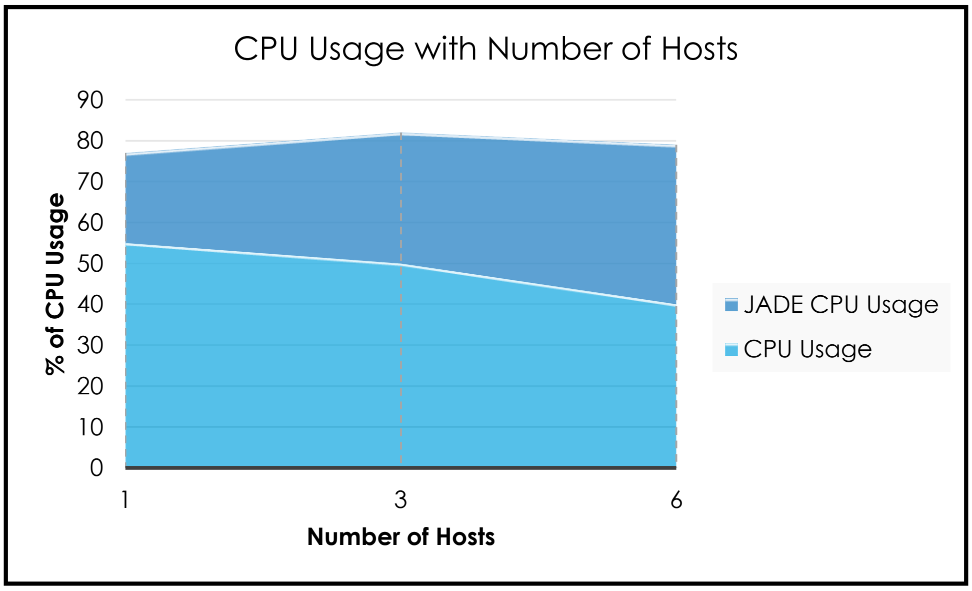


Figure 5.10: CPU Usage with the increase in Number of Hosts

From the results it is clear that, there is no clear trend in the data on the amount of messages transferred or the time to reach convergence, so no graphs have been made of this data. A reason for this could be that the agents themselves do not change and thus do not require more communication or more time to reach convergence. Two graphs have been made that show the CPU and memory usage of the system and JADE.

The CPU usage of JADE and the new MAS are shown in percentage and are stacked. Three tests were performed, using one, three and six hosts. The CPU usage of the new MAS decreases as the amount of hosts increases, which is shown by the light blue area in the graph. The percentage went from 55% to 50% and then to 40% and matches a Linear function**:** ***y = - 3.02632 x+ 58.4211.*** This decrease can be explained by the distribution of the agents over the different hosts. The CPU usage of JADE increases slightly as the amount of hosts increases, visualized by the dark blue area. This percentage went from 22% to 32% and to 39% and matches a Linear function: ***y = 3.31579x + 19.9474***. This increase is caused by the additional hosts in the system, as JADE handles more communication between the hosts. JADE only runs on one host machine and therefore the average CPU usage on the other hosts is defined by the light blue area and the corresponding linear function. The CPU usage on the JADE host will still only slightly rise until about 400 hosts are present, as defined by the two linear functions.

The memory usage of JADE and the new MAS are shown in MB of RAM and are stacked in the graph. Three tests can be seen here as well, using one, three and six hosts. The memory usage of the new MAS decreases as the amount of hosts increases, shown by the light blue area in the graph. The usage went from 323 MB to 263 MB and then to 148 MB and matches a Linear function: ***y = -35.2632 x + 362.211***. This decrease can be explained by the distribution of the agents over the different hosts. The memory usage of JADE does not change much as the amount of hosts increases, visualized by the dark blue area. This usage went from 111 MB to 118 MB and to 115 MB, which makes clear that there is no clear increase or decrease in memory usage.

These results indicate that for most of the hosts the resource usage will decrease significantly as more hosts are added, except for the host that is running JADE. The CPU usage on this host will increase slightly, but as the maximum amount of hosts is around 400, this does not pose a great threat on scalability. This means that the scalability with respect to performance increase per resource is significant enough to make adding hosts useful.

Test Case: 4

The fourth scalability test case focused on the performance overhead of restarting agents through the EH agent. In total seven tests have been performed, each on one host and using 84 agents, 60 Consumers, 14 Prosumers, 6 Gencos, 1 Balancer, 1 Time agent, 1 Weather agent and 1 EH agent. Measurements have been performed on the number of messages transferred, the time to reach convergence and the average CPU and memory usage of the system and JADE.

Table 5.7: Scalability Test Case- 4

What is the performance overhead of restarting an agent in the system through the EH agent?	
How to	Kill each agent type once, to initiate a restart of that agent
Type	**Normal MAS**
Test1	1 host, 84 agents (60c, 14p, 6g, 1b, 1t, 1w, 1e)*
Test2	1 host, same agent amount, kill a Balancer agent
Test3	1 host, same agent amount, kill a Weather agent
Test4	1 host, same agent amount, kill a Time agent
Test5	1 host, same agent amount, kill a Consumer
Test6	1 host, same agent amount, kill a Prosumer
Test7	1 host, same agent amount, kill a Genco
Metrics • Number of agents in the system • Number of hosts • Total number of messages transferred between agents • Number of ticks that have passed to reach convergence • CPU and memory usage	

**Notation: c – consumer; p – prosumer; g – genco; b – balancer; t – time; w – weather; e- exception handling*

Results:

Table 5.8: Results of Scalability Test Case: 4

What is the performance overhead of restarting an agent in the system through the EH agent?							
Metrics	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Test 6**	**Test 7**
Number of Hosts	1						
Number of Agents	99(60c, 14p, 6g, 6b, 6t, 6w, 1e)*						
Average number of messages transferred between agents	21047	12432	17730	17650	11189	5837	6606
Average time to reach convergence/Number of ticks that have passed	9756 ms	10923 ms	9989 ms	10269 ms	15157 ms	9005 ms	10129 ms
Average CPU and memory usage	58%, 275MB	54%, 230MB	55%, 226MB	55%, 246MB	53%, 197MB	54%, 202MB	52%, 178MB
Average CPU and memory usage by JADE	21%, 90MB	24%, 93MB	23%, 92MB	21%, 86MB	23%, 96MB	20%, 93MB	23%, 88MB

**Notation: c – consumer; p – prosumer; g – genco; b – balancer; t – time; w – weather; e- exception handling*

The data on the average CPU and memory usage of the new MAS and JADE is constant. A reason for this could be that the restart of an agent is not big enough to influence the system on the hardware level and only on the software level.

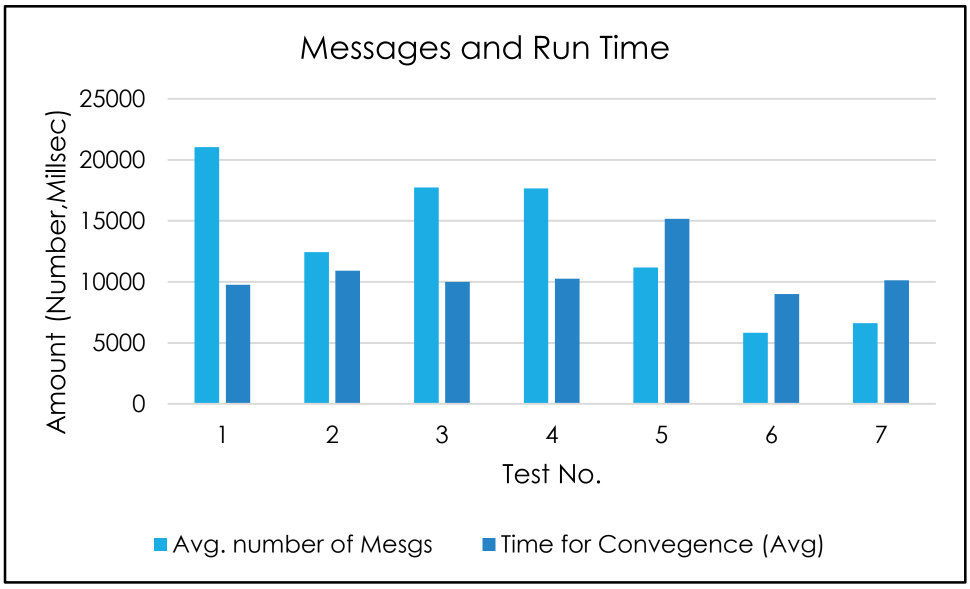


Figure 5.11: No. of Messages and Run Time (in Millisec) with different Agents

The first test is a normal run of the system without an agent being killed. The number of messages being sent in this test is higher than in all the other tests. This is caused by the fact that a killed agent cannot receive messages and it takes some time for the agent to be restarted. When a Weather or Time agent is killed, the number of messages is a little lower than in the normal case, as these agents are not contacted very often. A killed Prosumer or Genco results in a very large decrease of messages being sent, as these agents are contacted a lot during the simulation and also send a lot of messages. The run-time of the new MAS in milliseconds is largely the same for most tests, but is more than half as long in the case of a killed Consumer. This is caused by the fact that the simulation is only finished if every Consumer has a contract. As the killed Consumer has to be restarted and restarts its negotiations from the beginning, this takes some time and causes this delay.

Test Case: 5

Table 5.9: Scalability Test 5

What is the performance overhead of critical agent replication?	
How to	Compare the performance of the MAS with different amounts of replicas on six hosts.
Type	**Normal MAS**
Test1	6 hosts, 84 agents (60c, 14p, 6g, 1b, 1t, 1w, 1e)*, 0 replicas
Test2	6 hosts, 90 agents (60c, 14p, 6g, 3b, 3t, 3w, 1e)*, 2 replicas
Test3	6 hosts, 99 agents (60c, 14p, 6g, 6b, 6t, 6w, 1e)*, 5 replicas
Metrics • Number of agents in the system • Number of hosts • Total number of messages transferred between agents • Number of ticks that have passed to reach convergence • CPU and memory usage	

**Notation: c – consumer; p – prosumer; g – genco; b – balancer; t – time; w – weather; e- exception handling*

Results:

Table 5.10: Results of Scalability Test 5

What is the performance overhead of critical agent replication?			
Metrics	**Test1**	**Test 2**	**Test 3**
Number of Hosts	**6**		
Number of Agents	84(60c, 14p, 6g, 1b, 1t, 1w,1e)*	90(60c, 14p, 6g, 3b, 3t, 3w, 1e)*	99(60c, 14p, 6g, 6b, 6t, 6w, 1e)*
Average number of messages transferred between agents	3028	3581	5470
Average time to reach convergence/Number of ticks that have passed	6720 ms	7971 ms	17113 ms
Average CPU and memory usage	43%, 171MB	43%, 169MB	40%, 148MB
Average CPU and memory usage by JADE	36%, 112MB	31%, 125MB	39%, 115MB

**Notation: c – consumer; p – prosumer; g – genco; b – balancer; t – time; w – weather; e- exception handling*

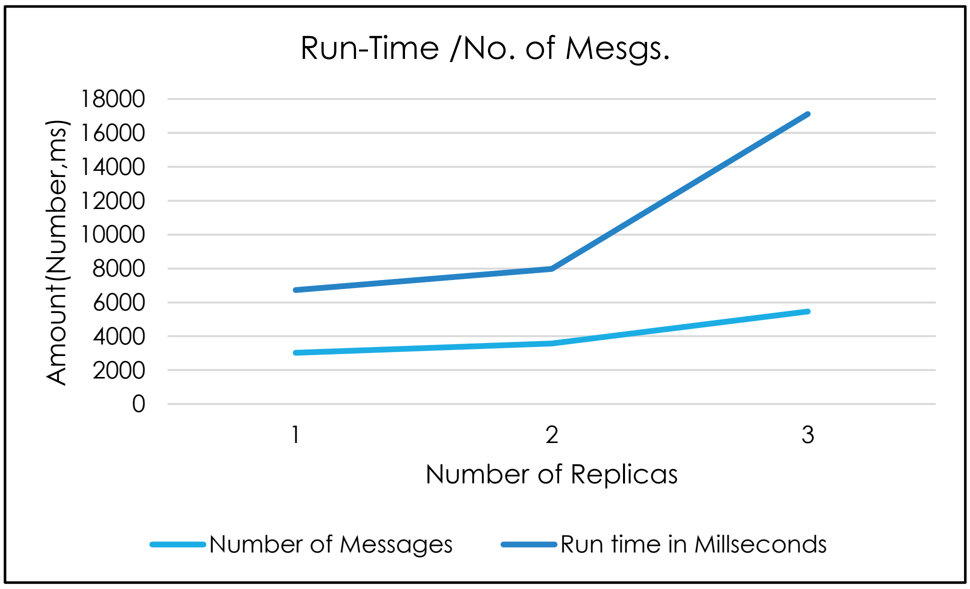


Figure 5.12: Run Time/No. of Messages with the increase in number of Replicas

5.2.3 RELIABILITY TEST CASES:

The first reliability test case focuses on the effectiveness of the EH agent to prevent complete system failure if a certain agent fails. To test this, the new MAS is run with the EH agent and without the EH agent for each test and the results are compared. Two tests have been performed, one on only one host machine, with 83 agents, 60 Consumers, 14 Prosumers, 6 Gencos, 1 Balancer, 1 Time agent and 1 Weather agent. And a test on six hosts, with 98 agents, 60 Consumers, 14 Prosumers, 6 Gencos, 6 Balancer agents, 6 Time agents and 6 Weather agents. In each test, every agent type was killed twice, resulting in twelve simulations per test. Measurements have been performed on the number of successful simulations, which is combined with the number of simulations in order to calculate a Success rate.

Table 5.11: Reliability Test ***Case: 1***

How effective is the EH agent in preventing complete system failure, when a certain agent fails?				
Metrics	**Test1, Without EH agent**	**Test 1, With EH agent**	**Test 2, Without EH agent**	**Test 2, With EH agent**
Number of Hosts	1		**6**	
Number of Agents	83(60c, 14p, 6g, 1b, 1t, 1w)*	84(60c, 14p, 6g, 1b, 1t, 1w, 1e)*	98(60c, 14p, 6g, 6b, 6t, 6w)*	99(60c, 14p, 6g, 6b, 6t, 6w, 1e)*
Success Rate = Number Of Successful Simulations/ Number Of Simulations	7 / 12 = 0.583	9 / 12 = 0.75	10 / 12 = 0.833	11 / 12 = 0.917

**Notation: c – consumer; p – prosumer; g – genco; b – balancer; t – time; w – weather; e- exception handling*

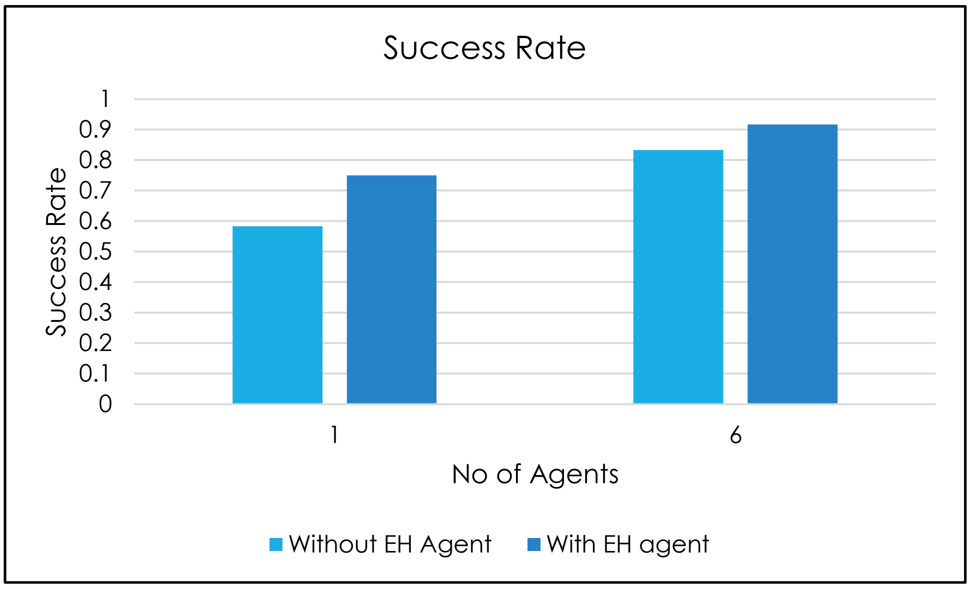


Figure 5.13: Success Rate With/Without Exception Handling Agent

Case 2:

The Success rates of the new MAS on one and six hosts are displayed in a graph. The Success rate of the new MAS with the EH agent is shown in dark blue and without the EH agent is shown in light blue. From this graph it is clearly visible that the EH agent results in higher Success rates on both tests. This means that, if an agent fails, the system continues to function more often with the EH agent, than without the EH agent. This indicates that an Exception Handling agent is effective in increasing the reliability of the system. Together with the scalability test results, this makes a very useful and effective method. The Success rate is higher in the test with six hosts, with and without the EH agent.

Table 5.12: Reliability Test Case-2

How effective is critical agent replication in preventing complete system failure, when a top-level intermediary fails?			
Metrics	**Test 1, Without EH agent**	**Test 2, Without EH agent**	**Test 3, Without EH agent**
Number of Hosts	**1**	**3**	**6**
Number of Agents	83(60c, 14p, 6g, 1b, 1t, 1w)*	89(60c, 14p, 6g, 3b, 3t, 3w)*	98(60c, 14p, 6g, 6b,6t, 6w)*
Replication Rate = Number of Extra replicas/Number of Agents	0 / 83 = 0	6 / 83 = 0.07	15 / 83 = 0.18
Success Rate = Number Of Successful Simulations/ Number Of Simulations	3 / 6 = 0.5	5 / 6 = 0.833	6 / 6 = 1

**Notation: c – consumer; p – prosumer; g – genco; b – balancer; t – time; w – weather; e- exception handling*

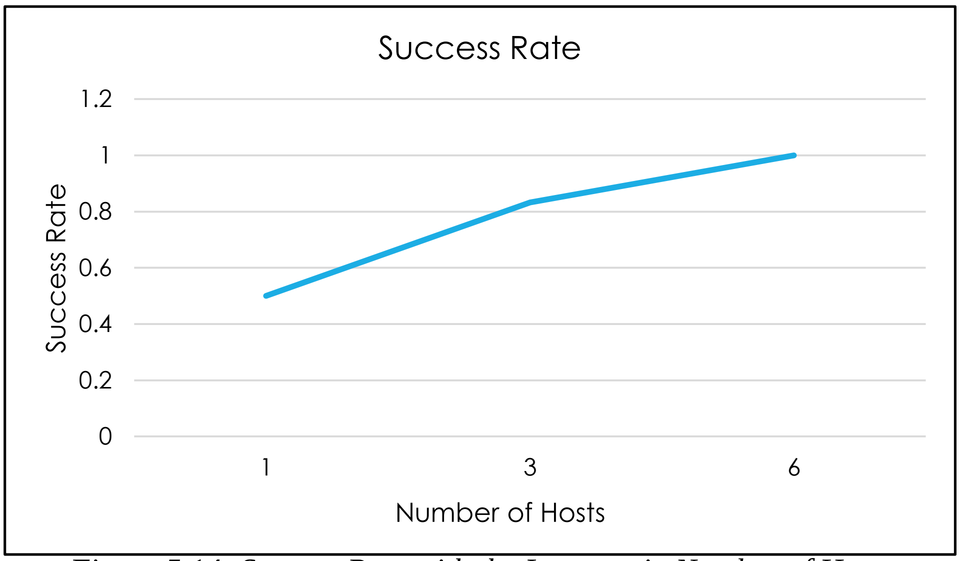


Figure 5.14: Success Rate with the Increase in Number of Hosts

The Success rates of the new MAS on one and six hosts are displayed in a graph. The Success rate of the new MAS with the EH agent is shown in dark blue and without the EH agent is shown in light blue. From this graph it is clearly visible that the EH agent results in higher Success rates on both tests. This means that, if an agent fails, the system continues to function more often with the EH agent, than without the EH agent. This indicates that an Exception Handling agent is effective in increasing the reliability of the system. Together with the scalability test results, this makes a very useful and effective method. The Success rate is higher in the test with six hosts, with and without the EH agent.

Case: 3

Table 5.13: Reliability Test ***Case- 3***

What is the impact of a queue limit and garbage collector on agent thrashing?				
Metrics	**Test 1, New MAS**	**Test 2, Old MAS**	**Test 3, New MAS**	**Test 4, Old MAS**
Number of Hosts	**1**			
Number of Agents	83(60c, 14p, 6g, 1b, 1t, 1w)*			
Average time to reach convergence/Number of ticks that have passed	9756 ms	38076 ms	11588 ms	78597 ms
Average CPU and Memory Usage	58%, 275MB	55%, 306MB	51%, 377MB	57%, 402MB
Average CPU and Jade Memory Usage	21%, 90MB	27%, 77MB	17%, 76MB	25%, 84MB

**Notation: c – consumer; p – prosumer; g – genco; b – balancer; t – time; w – weather; e- exception handling*

From these results, there is one significant difference noticeable in the second test between both MAS. This is the average time to reach convergence. The new MAS finishes in 12 seconds, while the old MAS takes 79 seconds. Even more interesting was the fact that the new MAS had a Spam agent that sent a message to each agent in the system every 50 milliseconds, where the old MAS could not finish the simulation in the same case and could only work if messages were sent every 500 milliseconds.

Compared to the first test, the new MAS performs slightly less with the Spam agent active in the second test, which was to be expected. The time to reach convergence is about 19% higher. The time to reach convergence is increased by 106%. Here the memory usage is also about 100 MB higher.

As seen from these results, the impact of a queue limit and garbage collector is huge. It can prevent agent thrashing due to a faulty agent which is spamming messages. This is clearly what happened to the old MAS in this test and what was prevented in the new MAS. The reliability of the new system is therefore much better, as the system can cope with faulty agent.

CHAPTER 6
CONCLUSIONS

6.1 SUMMARY

In this thesis, the main focus was to create a reliable Multi-agent system for a large scale distributed energy trading network. The need for such a system was described in the introduction and is linked to the need for changes in the current energy market. The current model is too monopolistic and favors the big energy production companies, or Gencos. The consumers are therefore bound to high prices and limited innovation, because of low market competition. This model is now slowly undergoing some changes, such as new Gencos being added, which tend to make it a more open market where there is more focus on innovation as well as environmental issues. However, there is still a lot that has to be improved, as today the so called Prosumers can only sell their energy back to their Genco and no other consumers. The ideal future vision is that of an open energy market where Prosumers and Gencos compete for the consumers. The necessary changes to the current market and energy grid therefore have to be identified and an energy market simulation can provide a platform where these changes can be tested before they are actually used. It can also be used after the actual changes have been made, as a testing and monitoring environment. It has been shown that the ideal candidate for such a simulation is a Multi-agent system, because of the autonomy and ability to model behavior.

In the state of the art different related projects have been discussed as well as Multi-agent systems in general. A lot of different platforms and agent languages exist that can be used to implement a Multi-agent system. Also a lot of different systems have already been implemented in the field of energy market simulations. One of these systems is the MAS by N. Capodieci, which was used a basis for this thesis. This is a basic MAS that simulates the energy market by using Consumer, Prosumer and Genco agents that buy and sell energy in a contracting auction. It also has a time and weather simulation and a GUI.

The scope of this research was to use the existing MAS by N. Capodieci that supports this simulation and to expand it by adding scalability and reliability improvements to make the system more usable. This was needed because these features have not been taken into account when that MAS was created. The system was limited to one host only and has no specific measures to prevent failures. This limited the usage of the system, as only a limited amount of agents could be run and the system could crash on a fault.

JADE was used as the platform for the Multi-agent system implementation, as explained in the background information. The JADE platform provided the best advantages and features for the system that needed to be created. Also, the MAS by N. Capodieci uses JADE, which was another major advantage. A lot of different methods of improving scalability and reliability for Multi-agent systems exist, but not all of these were suitable in this case. The advantages and disadvantages of each method have been compared and a selection has been made on this basis. Some of the more prominent methods were: the use of distribution and replication, which proved invaluable for the scalability and reliability of the system.

The architecture of JADE and the MAS has been clearly explained. JADE uses a container that wraps the agents. This allows the JADE system to distribute on several hosts, each running one or more containers, which results in a system that supports transparent access. JADE also implements an agent behavior scheduler and uses agent address caching, which are in total three of the methods chosen that can increase the scalability of a MAS. The architecture of the MAS itself was focused on the structure of the agent communication and interaction and the structure of the system, which corresponded largely to the JADE structure. Different methods have been realized and are visible in the architecture, such as distribution, replication and the EH agent.

The implementation was focused on the class diagram and class implementation of the agents. Three features that improve the reliability of the MAS have been implemented on this level, the queue limit, the garbage collector and the handling of messages from the EH agent.

A wide range of test cases have been created that focused on testing the selected methods in the field of scalability and reliability and are based on the research questions stated in the introduction.

In total eight test cases have been created, five for scalability tests and three for reliability tests. The tests cases focused on different aspects of the scalability and reliability of the system, such as the performance increase per added resource and the agent limit, as well as the impact of the EH agent and replication on the success rates.

The tests have been performed on a reasonably fast system and measures have been taken to ensure realistic and correct test results, such as running no other programs on the system and using similar systems for testing with multiple hosts. Also each of the tests within each test case has been performed five times, in order to average the values and remove spikes in the data.

6.2 CONCLUSIONS

- Number of Agents can the MAS handle before it becomes unstable?
  - The MAS can handle 818 agents on a 6 host system, which is four times as many as the old MAS. The MAS requires only 8 minutes to finish this simulation, where the old MAS takes four times as long

- Scalability of the new MAS compared to the old MAS, in terms of performance per load?
  - The new MAS was tested on 6 hosts, the old MAS on 1. The CPU usage of the new MAS did not increase as the amount of agents increased, this did happen on the old MAS. The memory usage of the new MAS is significantly 6lower than the old MAS and increases slower, but both match a linear function. The time to reach convergence and the amount of messages is limited in the new MAS and also rises much slower than on the old MAS, but both match a Quadratic function.

- Scalability of the new MAS with respect to performance increase per resource?
  - As new hosts are added to the MAS, the time to reach convergence and the amount of messages transferred does not change noticeably. The memory and CPU usage decreases significantly for each added host by a linear function, except for the host that is running JADE, which has a slight increase in CPU usage.

- Performance overhead of critical agent replication?
  - The memory and CPU usage of the MAS shows no specific changes if more replicas are added. The time to reach convergence and the amount of messages does increase significantly by a Quadratic function.

- What is the performance overhead of restarting an agent in the system through the EH agent?
  - The CPU and memory usage for restarting an agent does not differ from the normal case. The time to reach convergence and the amount of messages sent decreases in most of the cases, compared to the normal case. Only the restart of a Consumer has a negative effect on the time to reach convergence, which increases by 50%.

Reliability Evaluation questions:

- How effective is the EH agent in preventing complete system failure, when a certain agent fails?
  - The MAS with the EH agent results in higher success rates than the MAS without the EH agent. The success rates increase by 10-28%, depending on the amount of hosts.

- How effective is critical agent replication in preventing complete system failure, when a top level intermediary fails?
  - The MAS without critical agent replication has a success rate of 50%, where the MAS with a replication rate of 0.18 has a success rate of 100%.

- What is the impact of a queue limit and garbage collector on agent thrashing?
  - The new MAS was tested with a Spam agent sending a message every 50 milliseconds, the old MAS was tested with a Spam agent sending a message every 500 milliseconds. The new MAS finishes in 12 seconds, as the old MAS without queue limit and garbage collection takes 79 seconds. There is no clear distinction in memory usage or CPU usage of both MAS.

6.3 FUTURE WORK

The possibilities for this expanded functionality are endless, as the system can be expanded by all kind of agent types, learning strategies, more hosts, improved GUI, realistic weather, real world topology, etc. Some possible expansions that I would personally find useful or would have done if I had more time:

- Extended bug fixing and performance increases, by removing unnecessary messages and code.
- Adding realistic weather.
- Adding some small real world topology.

REFERENCES

[1] Sinha A, Neogi S., Lahiri R.N., Chowdhury S., Chowdhury, S.P., Chakraborty N, Smart grid initiative for power distribution utility in India, Power and Energy Society General Meeting, 2011 IEEE, 24-29 July 2011, Page(s): 1 – 8

[2] Rihan, M. ,Electr. Eng. Dept., Aligarh Muslim Univ., Aligarh, India Ahmad, Mukhtar; Salim Beg, M. Developing smart grid in India: Background and progress; Innovative Smart Grid Technologies - Middle East (ISGT Middle East), 2011 IEEE PES Conference on 17-20 Dec. 2011; Page(s): 1 - 6

[3] http://en.wikipedia.org/wiki/Electricity_market

[4] Marta Marmiroli and Hiroshi Suzuki. Web-based Framework for Electricity Market. *Power System and Transmission Eng. Center*, Mitsubishi Electric Corporation.

[5] S. Widergren, J. Sun, and L. Tesfatsion. Market Design Test Environments. *Proc. of 2006 IEEE PES General Meeting*. June 2006.

[6] I. Praca, C. Ramos, and Z. Vale. MASCEM: A Multi agent System that Simulates Competitive Electricity Markets. *IEEE Trans. Intelligent Systems*. Vol. 18, Pages. 54-60. Dec. 2003.

[7] Michael Wooldridge. *An Introduction to Multi Agent Systems*. John Wiley & Sons Ltd, 2002, paperback, 366 pages.

[8] Zahia Guessoum, Jean-Pierre Briot, Olivier Marin, Athmane Hamel, and Pierre Sens. Dynamic and Adaptive Replication for Large-Scale Reliable Multi-Agent Systems. In *Software Engineering for Large-Scale Multi-Agent Systems*. Lecture Notes in Computer Science, Vol. 2603/2003, Pages 70 211-235, 2003.

[9] Nanpeng Yu and Chen-Ching Liu. Multi-Agent Systems and Electricity Markets: State-of-the- Art and the Future. In *Power and Energy Society General*

Meeting - Conversion and Delivery of Electrical Energy in the 21st Century (IEEE), Pages 1-2, July 2008.

[10] G. Nguyen, T. Dang, L. Hluchy, Z. Balogh, M. Laclavik, and I. Budinska. *Agent platform evaluation and comparison.* Technical report for Pellucid 5FP IST-2001-34519, June 2002.

[11] N. Capodieci. P2P energy exchange agent platform featuring a game theory related learning negotiation algorithm. *Master's thesis, University of Modena and Reggio Emilia*, 2011.

[12] L.C. Lee, H.S. Nwana, D.T. Ndumu and P. De Wilde. The stability, scalability and performance of multi-agent systems. In *BT Technol J,* Vol. 16, No. 3, July 1998.

[13] Onn Shehory. A Scalable Agent Location Mechanism. In *Intelligent Agents VI. Agent Theories Architectures, and Languages*. Lecture Notes in Computer Science, Vol. 1757/2000, Pages 162-172. 2000.

[14] N.J.E. Wijngaards, B.J. Overeinder, M. van Steen, and F.M.T. Brazier. Supporting Internet-Scale Multi-Agent Systems. In *Data & Knowledge Engineering,* Vol. 41, Issues 2-3, Pages 229-245, June 2002.

[15] Ralph Deters. Scalability & Multi-Agent Systems. 2001.

[16] Staffan Hägg. A Sentinel Approach to Fault Handling in Multi-Agent Systems. In *Proceedings of the Second Australian Workshop on Distributed AI, in conjunction with Fourth Pacific Rim International Conference on Artificial Intelligence (PRICAI'96),* 1996.

[17] Eric Platon, Shinichi Honiden and Nicolas Sabouret. Challenges in Exception Handling in Multi-Agent Systems. In *Proceedings of the 2006 international*

workshop on Software engineering for large-scale multi-agent systems (SELMAS '06), May 22-23, 2006.

[18] Sanjeev Kumar and Philip R. Cohen. Towards a Fault-Tolerant Multi-Agent System Architecture. In *Proceedings of the fourth international conference on Autonomous agents*, Pages 459-466, 2000.

[19] Mark Klein, Juan-Antonio Rodriguez-Aguilar, Chrysanthos Dellarocas. Using Domain- Independent Exception Handling Services to Enable Robust Open Multi-Agent Systems: The Case of Agent Death. *Autonomous Agents and Multi-Agent Systems*, Volume 7, Pages 179-189, 2003.

[20] Olivier Marin, Pierre Sens, Jean-Pierre Briot, Zahia Guessoum. Towards Adaptive Fault Tolerance For Distributed Multi-Agent Systems. In *Proceedings of ERSADS'2001*, May 2001.

[21] N. P. Yu. Modeling of Suppliers' Learning Behaviors in an Electricity Market Environment. *M.S. thesis, Dept. EE. Eng, Iowa. State University, Ames*. 2007.

APPENDIX A

A MODEL OF AN INTELLIGENT MULTI-AGENT SYSTEM

Agents reside and are executed in Places. There are five kinds of Agents: User Agent, Intermediary Agent, Knowledge Agent, Notice Agent, and Update Agent. Agents in the systems are Intelligent Agents. Knowledge Base based on COKB (Computational Object Knowledge Base) stores the knowledge relating to a field. The knowledge consists of concepts, hierarchy, relations between concepts, operators, functions and rules. Storage stores facts about states of local environment in the Place and facts about states of global environment in the system.

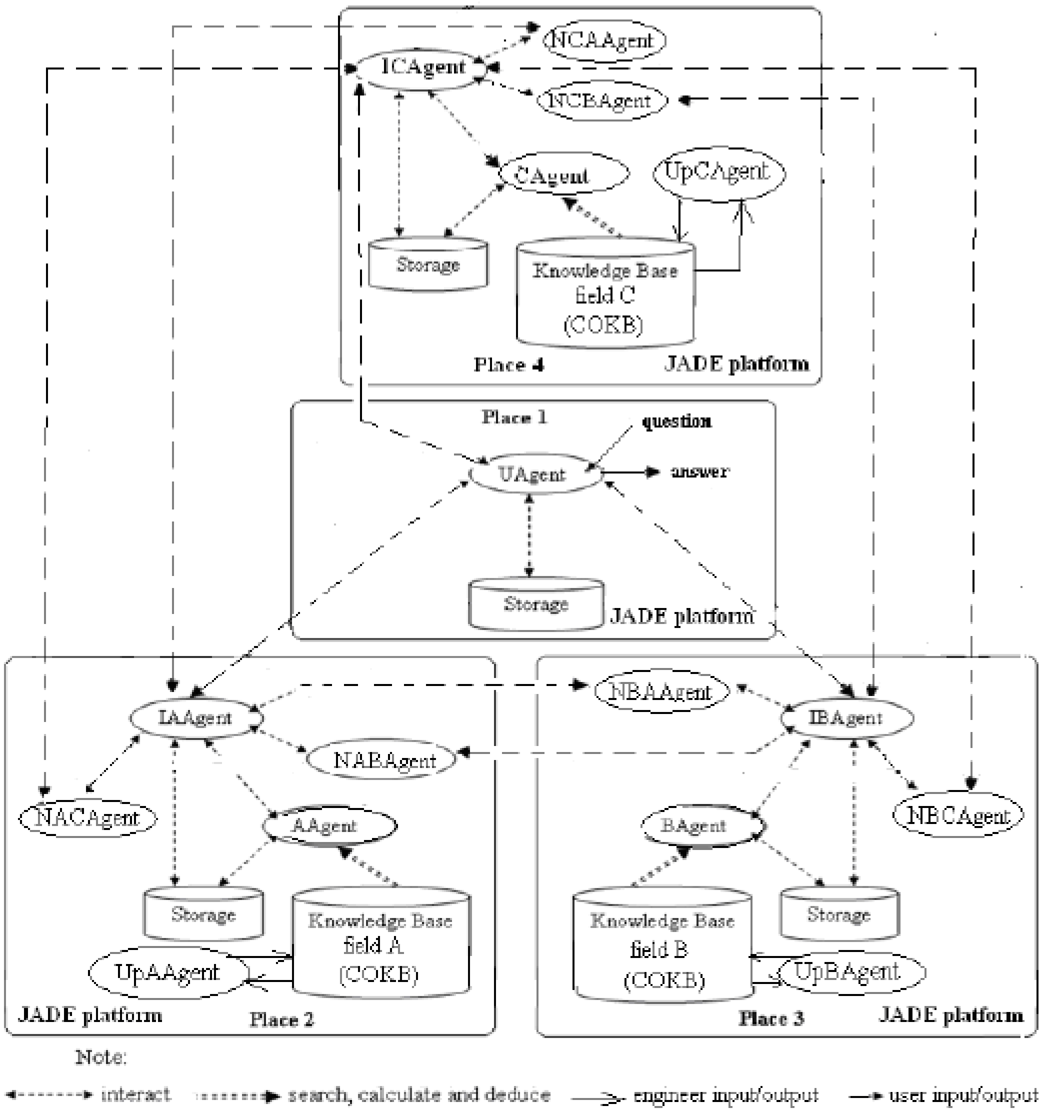
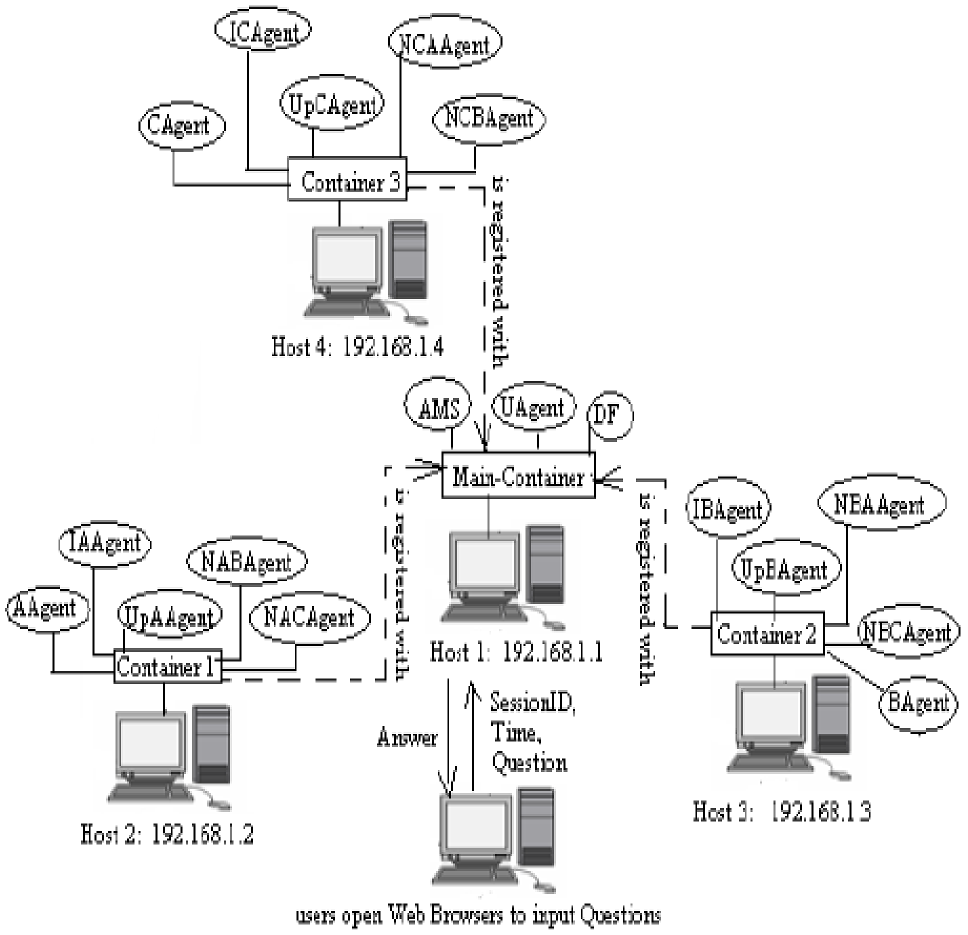
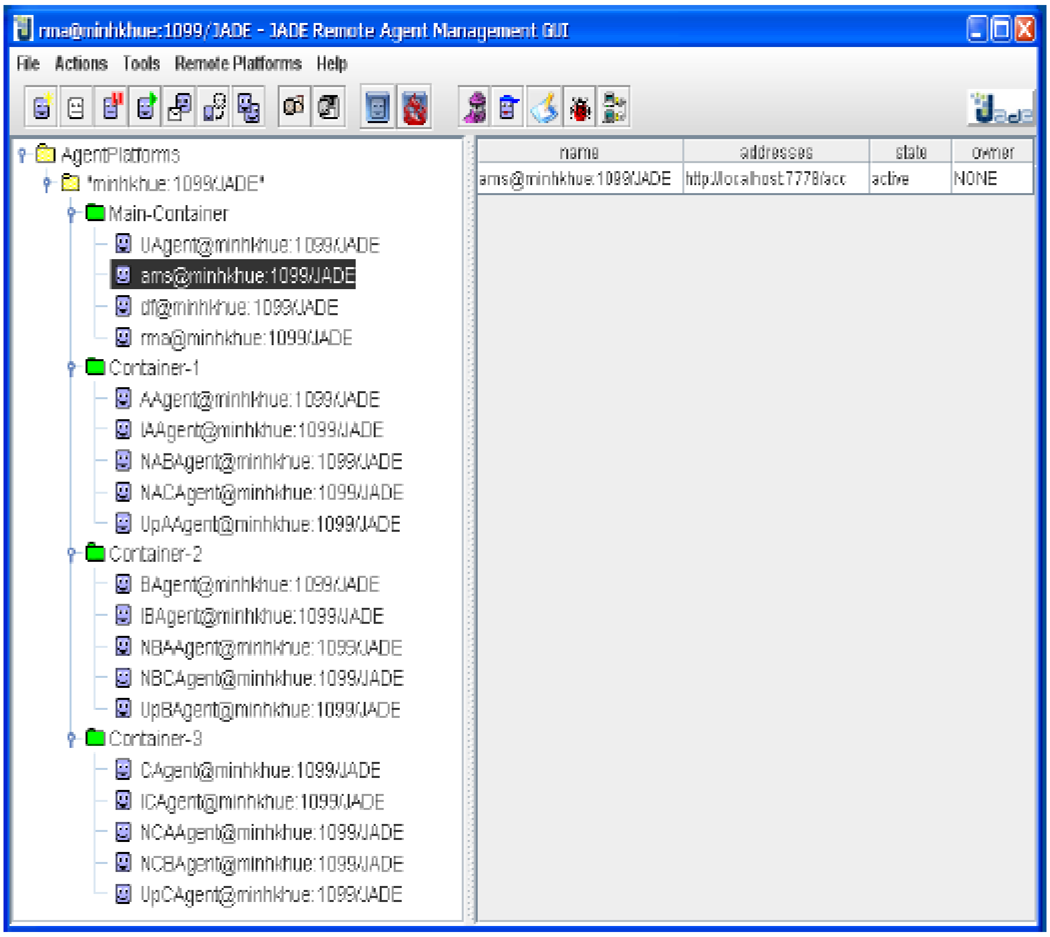


Figure showing Agent Structure and Relationship

In the model, we organize four Places: Place 1, Place 2, Place 3, and Place 4. Place 1 is used to interact with users. It consists of UAgent and Storage. UAgent is a User Agent. Storage stores states of local environment of Place 1. Place 2 is used to deal with the knowledge in field A. It consists of IAAgent, AAgent, NABAgent, NACAgent, UpAAgent, Knowledge Base A, and Storage. IAAgent is an Intermediary Agent in field A. AAgent is a Knowledge Agent in field A. NABAgent is a Notice Agent of Place 3. NACAgent is a Notice Agent of Place 4. UpAAgent is an Update Agent of the field A. Knowledge Base A stores the knowledge in field A. Storage stores states of local environment in Place 2 and states of global environment in the system. Place 3 is used to deal with the knowledge in field B and Place 4 is used to deal with the knowledge in field C. They are similar to Place 2. Agents can interact with agents in a Place or agents in different Places.



JADE GUI:



- AgentPlatforms
  - "minhkhue:1099/JADE"
    - Main-Container
      - UAgent@minhkhue:1099/JADE
      - ams@minhkhue:1099/JADE
      - df@minhkhue:1099/JADE
      - rma@minhkhue:1099/JADE
    - Container-1
      - AAgent@minhkhue:1099/JADE
      - IAAgent@minhkhue:1099/JADE
      - NABAgent@minhkhue:1099/JADE
      - NACAgent@minhkhue:1099/JADE
      - UpAAgent@minhkhue:1099/JADE
    - Container-2
      - BAgent@minhkhue:1099/JADE
      - IBAgent@minhkhue:1099/JADE
      - NBAAgent@minhkhue:1099/JADE
      - NBCAgent@minhkhue:1099/JADE
      - UpBAgent@minhkhue:1099/JADE
    - Container-3
      - CAgent@minhkhue:1099/JADE
      - ICAgent@minhkhue:1099/JADE
      - NCAAgent@minhkhue:1099/JADE
      - NCBAgent@minhkhue:1099/JADE
      - UpCAgent@minhkhue:1099/JADE

name	addresses	state	owner
ams@minhkhue:1099/JADE	http://localhost:7778/acc	active	NONE