FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Smart Reconfiguration of Distribution Grids using Agent-based Technology

Matheus Macedo Lopes

FOR JURY EVALUATION

Dissertation conducted under the Master's in Electrical and Computers Engineering Program - Major Energy

Supervisor: Prof. Vladimiro Miranda , Ph.D.

Co-Supervisor: Prof. Diego Issicaba , Ph.D.

July 25, 2016

Resumo

As manobras de isolamento para reconfiguração em redes de distribuição de média tensão são tradicionalmente manuais ou dependem de decisões tomadas pelos operadores de rede. A abordagem proposta assume uma arquitetura onde os agentes interagem em um ambiente de rede de distribuição simulado a partir do estabelecimento de metas projetadas seguindo o paradigma de orientação mulit-agente. A aplicação é implementada de tal forma que agentes AgentSpeak interagem entre eles através de uma comunicação baseada em ato de fala/comunicação, bem como com um ambiente desenvolvido em linguagem JAVA.

Neste contexto, esta tese propõe a modelagem e verificação de soluções baseadas em agentes para apoiar as operações de reconfiguração em redes de distribuição em nível de média tensão. A metodologia foi utilizada para apoiar as actividades dos operadores de redes de distribuição por meio de planos de restabelecimento de energia para ajudar em casos de falhas permanentes. As abordagens empregadas para arquitetura de agentes para a reconfiguração foram baseadas em modelo hierárquico e uma abordagem totalmente descentralizada. A capabilidade dos agentes foram desenvolvidas prevendo as possiveis aplicações do sistema de distribuição com foco em procedimentos de gestão des interrupções de service. As abordagens foram testadas em um alimentador teste trifásico do IEEE de 123 nós. Os resultados são de interesse para a comunidade acadêmica promovendo discussões científicas sobre os níveis de descentralização na entrega de energia elétrica em nível de distribuição.

Abstract

The isolation and switch state maneuvers for reconfiguration of distribution networks at medium voltage levels are traditionally manual or dependent on decision makings devised by distribution system operators. It assumes a architecture where agents interact within a simulated distribution grid environment from the establishment of abstract goals to be achieved of agent intentions. The integrated application is implemented such that AgentSpeak agents will interact through speech-act based communication as well as with a shared environment coded in JAVA language.

In this context, this thesis proposes modeling e verifying agent-based solutions to support power distribution reconfiguration operations at medium voltage level. The methodology was employed to assist activities of distribution system operators by means of energy reestablishment plans to help with permanent-system failures. The agent-based reconfiguration approaches employed either a hierarchical and a fully decentralized approach. Agent capabilities have been developed envisioning distribution system applications focusing on outage management procedures. The approaches have been tested on a IEEE 123 test feeder. Results are of interest for the scientific community and foster discussions on levels of decentralization in distribution system delivery.

Acknowledgments

It is with deep appreciation that I extend this thanks to all those who help me achieve the completion of these work.

To my friends at FEUP and colleagues –
It is your camaraderie which makes the experience most meaningful and lasting.

To my supervisor, Dr. Vladimiro Miranda –
Who provided me with the opportunity of developing this thesis in Brazil, close to my family

My advisor, Dr. Diego Issicaba –
for your guidance, support and patience.

And to my mother & sisters, Liza & Nayara –
For your constant encouragement, unwavering support, and unconditional love.

Matheus Macedo Lopes

*“Defeating racism, tribalism, intolerance
and all forms of discrimination will liberate us all”*

Ban Ki-moon

Contents

1 Introduction **1**
- 1.1 Motivation and Subject background . . . 1
- 1.2 Objectives . . . 2
- 1.3 Document Structure . . . 2

2 State of the art **5**
- 2.1 Power Distribution Systems . . . 5
  - 2.1.1 Reconfiguration Problem . . . 7
  - 2.1.2 Automation and IEC 61850 . . . 9
- 2.2 Agent-Based Systems . . . 10
  - 2.2.1 Basic Concepts and Definitions . . . 12
  - 2.2.2 The BDI Architecture and The Procedural Reasoning System . . . 13
  - 2.2.3 Agent Programming Languages and Development Environments . . . 17
  - 2.2.4 Agent Communication . . . 18
  - 2.2.5 MAS applications to Power Engineering . . . 19
- 2.3 Conclusion . . . 20

3 Developed Approach **23**
- 3.1 System Component and MAS Modeling . . . 23
  - 3.1.1 Distribution Grid Modeling . . . 24
  - 3.1.2 The JaCaMo MAS Modeling . . . 29
- 3.2 MAS Capabilities and Agent Plans Towards Reconfiguration . . . 35
  - 3.2.1 MAS Structure for Automatic Configuration . . . 36
  - 3.2.2 Agent Plans for Automatic Reconfiguration . . . 39
- 3.3 Final Remarks . . . 51

4 Simulation and Results Analysis **53**
- 4.1 MAS Initialization and building phase . . . 53
- 4.2 MAS Reconfiguration Approach: Hierarchical Solution . . . 60
- 4.3 MAS Reconfiguration Approach: Decentralized Solution . . . 70
- 4.4 Conclusion . . . 76

5 Final Remarks **77**
- 5.1 Conclusions . . . 77
- 5.2 Future Works . . . 77

A simulationCAll and dMS agents Main functions **79**

List of Figures

2.1 Example of an electric power system from generation to MV distribution level . 6
2.2 Simplified diagram of a power distribution feeder 7
2.3 Centralized Control 8
2.4 SCADA Network 10
2.5 IEC 61850-3 Substation device communication model 11
2.6 Functional programs 11
2.7 Basic Agent system structure 12
2.8 Typical structure of a multi-agent system 13
2.9 Typical Goal-Based Agent structure 14
2.10 The Procedural Reasoning System (PRS) 16
2.11 Message passing between Two Agents 19

3.1 OpenDSS stand-alone and COM simulation structure 24
3.2 IEEE123 Test Feeder 25
3.3 Terminal Definition 26
3.4 Wiring and bus representation 27
3.5 OpenDSS Executive solution modes and features 27
3.6 API method explained 28
3.7 JaCa approach 29
3.8 JaCaMo approach 30
3.9 Agent reasoning cycle 34
3.10 Artifact Model 34
3.11 Structural Specification UML diagram 38
3.12 simulationCall structural specification UML diagram 39
3.13 Ag_15 process of creation 41
3.14 busAgents final formation 41
3.15 busAgents fault coordination 46
3.16 dMS reasoning plan generator 48
3.17 busAgents fault coordination 49

4.1 JaCaMo framework console 53
4.2 Summary of steady state simulation performed inside the framework 55
4.3 Sensing summary results for a busAgent 56
4.4 Power grid created by Agent's coordinated communication an active Agents . . . 57
4.5 Steady state Loss density 58
4.6 Steady state bus Voltages per phases 58
4.7 Steady state bus Voltages 59
4.8 Steady state bus Voltages per phases 59

4.9 Fault location at zone 2 60
4.10 Fault belief trigger directed to busAgent 62 60
4.11 Short-circuit analyses after dMS reasoning upon fault 62
4.12 Agents acknowledging changes in the environment over fault state 62
4.13 dMS reported state from busAgents 63
4.14 Automatic zone isolation performed using agents speech act 63
4.15 dMS reasoning upon fault report 63
4.16 Shortest path grid plan 64
4.17 dMS reasoning upon fault report 65
4.18 TIEEE123test feeder shortest path grid 66
4.19 The reconfiguration achievement summary 66
4.20 The reconfiguration shortest-path validation 67
4.21 Short-circuit results 68
4.22 Post zone isolation result 68
4.23 Reconfiguration results 69
4.24 Power flow results 70
4.25 Fault perceptual belief Trigger form decentralized approach 70
4.26 Post faulted state 72
4.27 Fault perceptual belief Trigger form decentralized approach 72
4.28 Grid state after first reasoning cycle 74
4.29 Grid state after second reasoning cycle 75
4.30 Final Grid state 75

List of Tables

2.1 Bibliographic survey of multi-agent systems applied to power engineering problems 20

3.1 Switch Data table 25
3.2 Grid Line codes 26
3.3 Types of literals in plan context 32
3.4 Types of triggering events 32

A.1 simulationCall and dMS Agents Artifact List of Operation 80
A.2 Bus Agent Artifact List of Operation 81

List of Acronyms

AA	Agents-Artifacts
ACSR	Aluminium-conductor steel-reinforced cable
AOP	Agent-Oriented Programming
API	Application Programming Interface
AR	Automatic Reconfiguration
BDI	Believe-Desire-Intention
COM	Component Object Model
CSV	Comma-Separated Value
dll	Dynamic-Link Library
HV	High voltage
IDE	Integrated Development Environment
IED	Intelligent Electronic Devices
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
JCL	Java Class Library
LAN	Local Area Network
LV	Low Voltage
MAS	Multi-Agent Systems
MV	Medium voltage
OpenDSS	Open Distribution System Simulator
RTU	Remote Terminal Units
SCADA	Supervisory Control and Data Acquisition

Chapter 1

Introduction

This chapter brings the overall motivation that entail the need for further studies toward Automatic Reconfiguration (AR) solutions on distribution grids. The main objectives are stated and a revised documentation structure is summarized herein.

1.1 Motivation and Subject background

Distribution power grids are designed to provide electricity with a certain level of adequacy and security in order to assure service continuity and quality. Like most of the systems developed by humanity, the electrical power grids evolve on the basis of drive sectors such as economical, environmental and social.

Recently, initiatives especially concerned with these systems have been established and referred as "GRID 2030" [3], the IntelliGrid Initiative [18, 21], and the European Smart Grids Technology Platform [26], among others [23, 25, 55] in which the main concerns are leaned towards the power grid paradigm shift regarding the Smart Grids concept.

In short, these initiatives promote decentralized control implementations and management solutions, the integration of renewable and distributed energy resources, as well the modernization of the power grids. As a gradual process, the technical challenges embrace several power engineering related fields of expertise as power electronics, communication, information technology, and software engineering.

Distribution grid control and operation might be subjected to substantial changes under the general terms of the listed initiatives. As a matter of fact, most of the supply interruptions occurrences are caused by problems at the distribution level, which lacks monitoring and control devices in comparison with transmission grids.

Under this context, agent-based technology provides suitable a approach that allows a soft transition from actual distribution grids to smart distribution grids approach. Therefore,this project aims at modeling smart distribution solutions to grid reconfiguration using agent-based technology assuming an architecture where agents interact with each other with the overall goal of improving continuity, quality and security of supply, while maintaining electrical quantities within regulatory limits.

1.2 Objectives

This project aims at modeling smart distribution solutions to grid reconfiguration using agent-based technology for cases of restoration. It assumes a architecture where agents interact within a simulated distribution grid environment from the establishment of abstract goals to the achievement of agent intentions. The integrated application will be implemented such that AgentSpeak agents will interact through speech-act based communication as well as with a shared environment implemented in JAVA language.

The specific objectives of the work are the following:

- **Representation of distribution grid in an agent simulation environment** — A simulation model must be designed to simulate the system operation and provide information regarding steady-state and fault conditions. It requires modeling power distribution components as computational objects in order to integrate system state transitions at the MAS simulation model level.

- **Design AgentSpeak models and operations upon grid components in order to control behavior towards grid reconfiguration** — by analyzing electrical quantities of buses and lines under normal and abnormal conditions, agent reasonings and actions must be performed to achieve goals towards grid reconfiguration.

- **Implement, test and verify agent-based reconfiguration solutions, either following a herarchical approach or decentralized approach** — Reconfiguration solutions are tested and analyzed considering a hierarchical approach where automatic decision making is centralized in an entity which solves the problem through a minimum-path algorithm, as well a decentralized approach where grid self-organization is reached through interactive agents widespread in the distribution grid.

1.3 Document Structure

This first part of the work presents the introduction, which explains the project context, problem statement, objectives, as well as the outline of the thesis. The following chapters are structured as follows:

Chapter 2 addresses a state the of art regarding the main topics involved in the proposed approach. Theories, information and literature review are discussed. Subtopics under this chapter include information on distribution network reconfiguration and the definitions of agent and MAS, emphasizing their design, concepts and implementation.

Chapter 3 discusses the methodology adopted to conceive the computational model, the stages taken in order to represent distribution system components, the integration of a simulation tool to the MAS framework, and the reconfiguration solutions.

Chapter 4 results and discussion about the work are thouroghly described.

Chapter 5 conclusion of the project followed by suggestions of future works are presented in this chapter.

Chapter 2

State of the art

In this chapter a summary of the state of the art is presented referring the known methods and solutions related to this work. As introduced in the first chapter, this thesis involves knowledge about particular topics of power engineering and computational sciences. Hence, the main objective of this chapter is to provide a background and state of art of these two fields of study, emphasizing directly to the subjects of interest.

The contents of this chapter cover topics which entail power distribution delivery, grid reconfiguration as a problem of altering the state (open or close) of switching devices, as well considerations toward automation. In this context, the MAS approach is placed to integrate a notion of intelligence that supports system operation, contributing to the formalization of an idea of smartness to be embedded in actual networks.

Therefore, in section 2.1, power distribution systems are described and contextualized alongside their technical challenges for the purpose of reconfiguration. In section 2.2 and 2.3, autonomous agents, MAS and applications to power distribution engineering are discussed.

2.1 Power Distribution Systems

A typical power system can be divided into three parts: generation, transmission and distribution, as depicted in Fig. 2.1. In summary, generation and the transmission systems typically use high voltage (HV) values for efficiency, transformers are responsible for boosting/bucking voltage values and are located in different parts of the system, and distributions systems deliver power to end users (loads), either at medium voltage (MV) or in low voltage (LV) systems [51]. This paperwork will be focusing on the MV systems and applications.

Transmission systems operate with the support of several monitoring and data acquisition equipments [59], while distribution systems are more vulnerable to outages due to the fewer of number these equipments. Power outages in most of the cases are temporary, meaning that the continuity of supply can be shortly recovered by actuation of reclosers – a circuit breaker equipped with a mechanism that can automatically close after it has been opened due to a fault [4]. Reclosers are used on overhead distribution systems to interrupt temporary faults. However, the occurrence

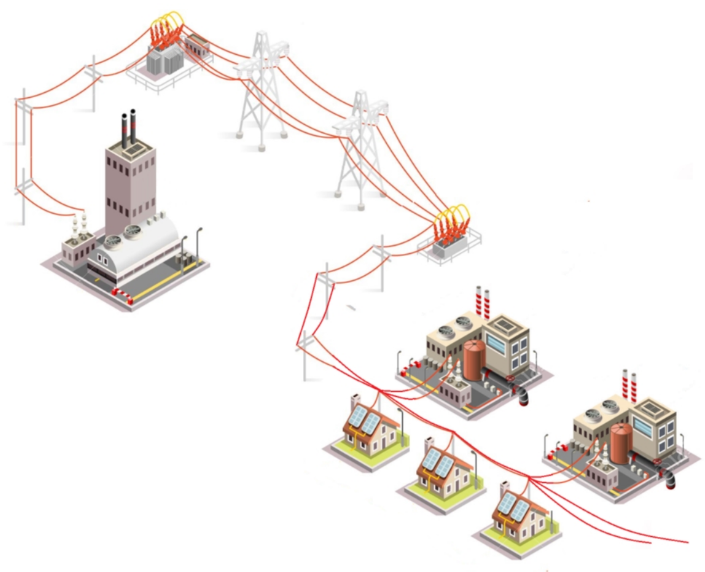


Figure 2.1: Example of an electric power system from generation to MV distribution level

of permanent faults in the electrical network is also a recurrent problem, for both, consumers and power utilities [43, 59].

The majority of distribution grids are designed upon primary topology systems. Due to the size of distribution grids, it is an often lower cost solution that intrinsically helps with managing delivery of supply in fault occurrences, minimizing the impact of outages caused by permanent faults in a segment of the grid. The utility considerations rely on: capacity of the grid, voltages and sources, size of metering equipment, relaying requirements and station batteries [60].

The primary distribution system is made up of circuits called primary feeders or distribution feeders. These feeders include the main feeder, usually a three-phase, four-wire circuit, and branches or laterals, which can be either three-phase or single-phase circuits [19]. In these systems, reclosers are used for both protection and for topology management. A schematic diagram of a simplified primary circuit of a distribution system together with reclosing devices is shown in Fig. 2.2.

Generally power distribution feeders provide power for both primary and secondary circuits, in which the primary feeder usually adopts reclosing devices positioned along the feeder. This arrangement minimizes the extent of time that primary feeders are out of service if a fault occurs. Thus the reclosing of these devices is set to limit the outage to the smallest number of affected loads. This can be achieved by coordinating all fuses and reclosers on the primary feeder [64].

Particuarly, when a feeder is subject to failure, a coordinated action of opening and closing switches can be performed, holding back permanent failure impacts. By reconfiguring the switches and reclosers installed on the distribution feeders it is possible to quickly isolate a faulted section and re-establish service to important customers. A reconfigurable grid is then considered a system comprised of sensors and automated controls used to isolate faults and to reconfigure the distribution network to minimize the outage impact to the customers. Grid reconfiguration comprise a complex problem, one to be modeled and addressed in this work following a MAS

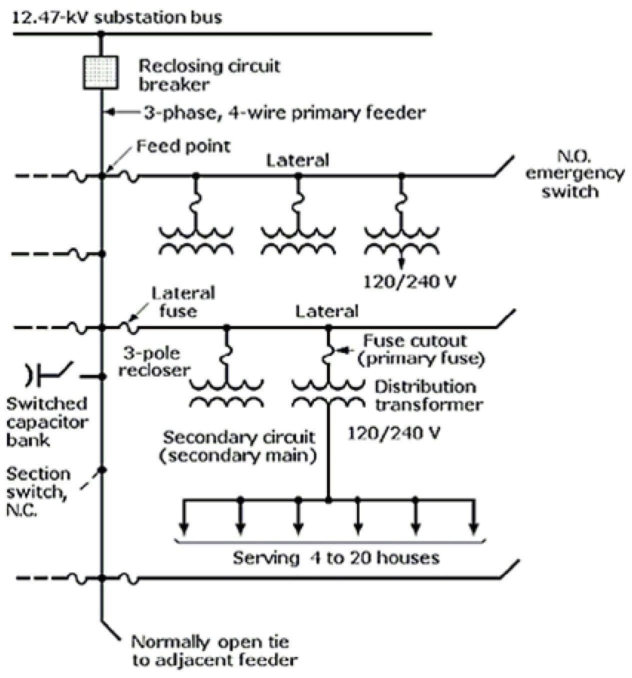


Figure 2.2: Simplified diagram of a power distribution feeder [19]

approach.

2.1.1 Reconfiguration Problem

The concept of reconfiguration was raised long ago [15], and recently has been implemented in many places with the development of switch technologies [40, 68, 77]. Nevertheless, choosing the right combination of opening/closing statuses of switches to optimize some performance or index criteria while keeping a radial topology is still a challenge at distribution level.

Network reconfiguration in distribution systems is then possible by changing the status of sectionalizing switches or reclosers, enabeling topology variation, and it is usually done with the purpose of loss reduction, network overload mitigation, voltage profile improvement and service restoration [6]. These very topics have been targets of research, with the usage of a variety of methods, such as:

- **Branch exchange procedures [6, 56]** — for loss reduction and load balancing;
- **Stochastics optimization [14]** — probabilistic technique for approximating the global optimum, for network reconfiguration;
- **Convex optimization [71]** — Mixed-integer minimization with convex quadratic constraints methods for network reconfiguration;
- **Heuristics and metaheuristic [6, 16]** — Variable scaling hybrid differential evolution methods for solving network reconfiguration of distribution systems and genetic algorithms for distribution systems loss minimum reconfiguration;

- **Mixed methods [41,45,50,57,70]** — Reconfiguration of distribution network mixing fuzzy logic, genetic algorithms, evolutionary algorithms, cloud theory, point estimate and stochastic methods, and the list goes on according to what can be found in IEEE digital library.

Some of these approaches are extensively used due to their broad applicability, e.g. for black-box models. Although every one of these techniques shows its practicality, for the most part, data acquisition approach is used and the execution of its algorithms are made in centralized manner, as depicted in Fig. 2.3.

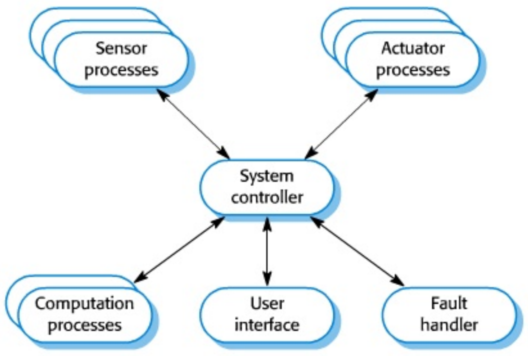


Figure 2.3: Centralized Control [37]

In a centralized control model, one component of the system is designated as the controller and is responsible for managing the execution of other components [37]. A clear example of that model is the Supervisory Control and Data Acquisition (SCADA) systems that collect data from devices and concentrate the analysis and processing of data in a control central station.

Since that centralized models in one hand have the role of decision maker and full control over the organization and the decisions that the organization must follow, on the other hand, as far as the system expands the larger an organization becomes, the more information has to comprehend and consider, eventually achieving processing limits. Furthermore, even if arguably one of the primary objective for power grid innovations are to improve service, whether or not it is decentralized and to what degree, at distribution level, centralized mechanisms for ensuring in-system quality and security are not easy to achieve nor affordable due its scale [42, 80].

When it comes to decentralization, systems achieve different dynamics, taking from the conceptual logic action that allows local actuators to take control of specific functions of an organization and easily convey information back and forth. It also allows for direct communication among elements responsible of coordinate and regulate the system [66]. Also some of the differences when compared to centralized system can be summarized:

- Scaling of the system are less costly processing-wise, that is, adding new equipment without complication in algorithm design;
- Failing of an equipment would not result in the failure of the system;

- Better adaptability to environment;
- Less memory and processing requirements;
- Better options for exception handling.

The downside of a fully decentralized system is the coordination dependency, regarding the system as a whole, as it can be hard to iterate on top of it since this also usually requires some kind of consensus from remote monitoring and control equipments [17]. Effective decentralized models usually have some intermediate devices who serves as a quasi-dictator to ensure progress with a certain degree of quality.

DMS usually centralize operations command using traditional SCADA systems, which provide the capability and information to hanfle operations and maintenance, as well as increasing system and staff efficiency. To ensure grids remain reliable over a wide range of operating conditions, many power utilities focus on system that promotes productivity, aiming at reducing costs and increasing customer satisfaction, though often involving upgrades on aging infrastructures. Substation automation is a rapidly increasing area of interest and benefit to utilities, that may include remote access to intelligent electronic devices (IEDs), relays, event data, diagnostic information, video for security, metering, switching, volt/VAR management, and so forth.

2.1.2 Automation and IEC 61850

Transmissions system typically adopts a distributed network structure to achieve real-time monitoring and control failure alarming and verification of protections values [27, 54]. Automation systems have integrated telecommunication and relays remote monitoring controls to assure stability in power supply and load management, as shown in Fig. 2.4.

Figure 2.4 displays an overview of SCADA networked architecture, where of the remote stations represents a substation attached with other organization network on differing network segments. From the control station, by making use of the human machine interface (HMI), decisions can be passed down towards HV/MV substations through SCADA communication protocols [38].

At the substation level, remote control is usually achieved either by hard-wired circuits to remote terminal units (RTUs) and IEDs or through substation automation. The first solution is explored by the form of exchanging control information and monitoring object values with the control station through SCADA communication protocols. The second solution is explored by establishing a local area network (LAN) within the substation using standard communication protocols, using the well known International Electrotechnical Commission (IEC) 61850 substation automation standards.

The substation device communication model, as shown in Fig. 2.5, consists of an embedded communication computer (DA-681) [52] that serves as a communication processor and uses gateway protocol to handle multiple devices, which use different protocols for front-end data computing (SCADA control station to substation) and protocol conversions. Throughout IEC 61850-3 communication protocols standards, device network is capable of providing protection against

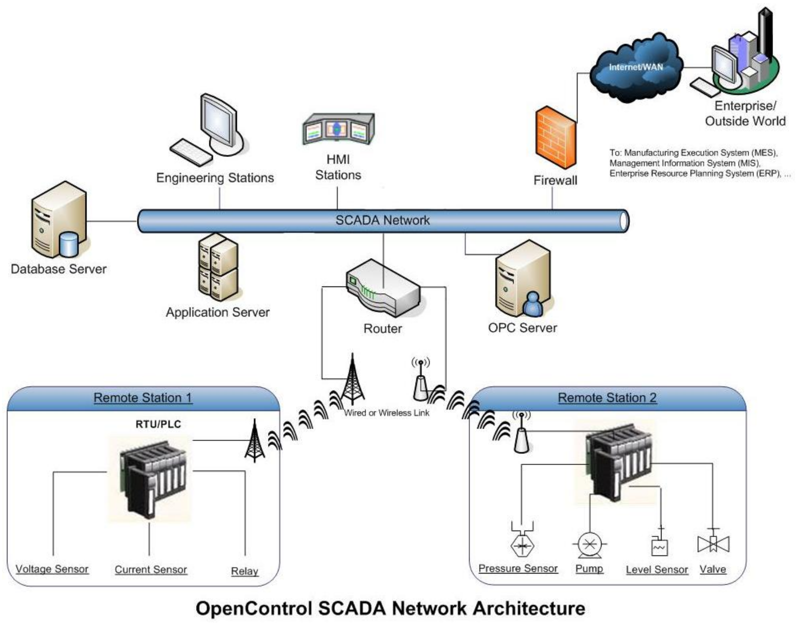


Figure 2.4: SCADA Network [13]

electrical surges and environmental threats. Also, it serves as a backed communication host and central controller for data analysis, processing, transmission back to the control center.

To date, applying the same data acquisition and monitoring approach for distribution networks is not easy or feasible. However, extensions of IEC 61850 regarding distribution automation have been cited as the following approaches are suggested:

- IEC 61850-based feeder terminal unit modeling and mapping [33];
- Extending IEC 61850-7-420 for distributed generators with fault current limiters [74];
- Distributed energy resources (DER) object modeling with IEC 61850 [75];
- IEC 61499 open control architecture towards intelligent smart grid devices with IEC 61850 interoperability [76];
- Standard function blocks for flexible IED in IEC 61850-based substation automation [81].

In account of these propositions and looking into the solutions of automation and communications applied to power distribution delivery, agent-Based systems are considered an eligible modeling and design tool to achieve decision making decentralization over distribution system operation.

2.2 Agent-Based Systems

In order to better understand the overall concept behind the terms "Agents" and "Multi-Agent-Systems" (MAS), let us first consider how agent systems can relate to other types of softwares.

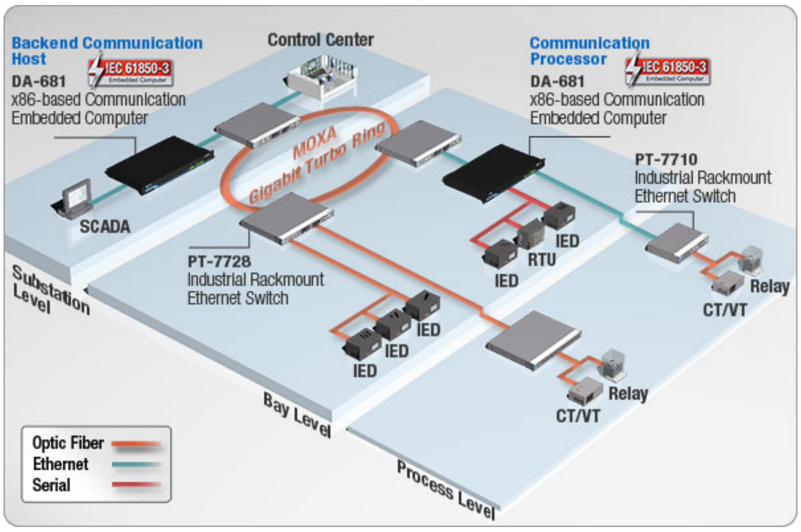


Figure 2.5: IEC 61850-3 Substation device communication model [52]

Functional programs/programming are possibly the simplest type of software from the development and engineering point of view [10].

In functional programming, programs are executed by evaluating expressions avoiding the use of mutable state. Mathematically, we can think of them as functions from some domain of possible inputs, source code programs for example, to some range of possible outputs (bytecode, object code, etc), as depicted in Fig. 2.6.

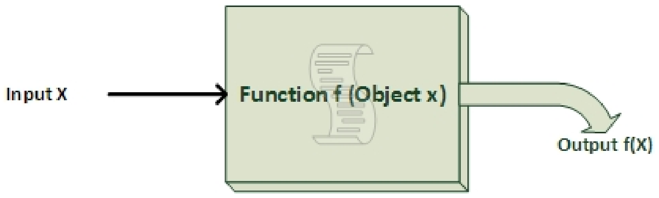


Figure 2.6: Functional programs

From the standpoint of software engineering development, the established techniques of such programs are typically straightforward to design and implement. In spite of that, according with the nature of most of programs, the simply input-compute-output approach is not valid as operational structure. In particular, most of systems are based on long-term ongoing interaction with their environment, i.e. they do not simply compute some functions of an input and then terminate. That is, systems that works that way are designed as reactive system [10].

Hence, the agent programming structure stands as a subset of reactive systems in which it also exhibits some degree of autonomy that the developer delegate in form of tasks. In turn, the system itself determines how the best procedures to achieve this tasks based on the environment state.

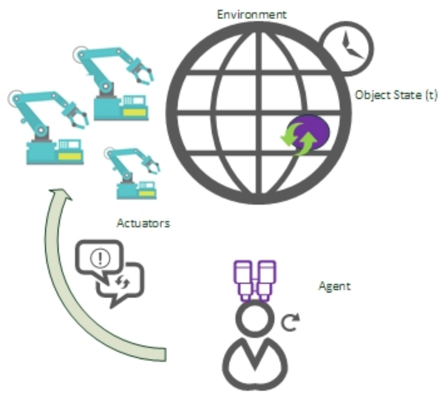


Figure 2.7: Basic Agent system structure

2.2.1 Basic Concepts and Definitions

One way to characterized agents are to consider them to be subsystems that are situated in some environment. What defines their role is then the capability of sensing this environment via sensors and have a repertoire of possible actions that they can perform via effectors or actuators in order to modify the environment. The relationship between an agent and its environment is illustrated in Fig. 2.7.

The questions raised over the view of agent systems lies along these lines: how to go from sensor input to action output and how to decide what to do based on the information obtained via sensors. The two concepts related to that are the percepts, an information received by some sensor, and the agent constrained delegations, which is something that leads to plan of actions. The term percept refers to the agent's perceptional inputs at any given instant.

The environment that an agent occupies may be physical or a simulated environment where decisions about actions are translated into actual actions usually achieved via some sort of application programming interface (API). Thus, while agents can perform actions capable of changing their environment, they generally cannot completely control it. Very often this is because of coordination between agents who exhibit control over their sphere of influence of the environment.

Apart from the environment aspect, other properties are expected from rational agenst to conduct, such as [10]:

- **Autonomy**— Agents that reason upon delegated goals, and then decides how best to act in order to achieve these goals;
- **Proactiveness**— meaning being able to exhibit goal-directed behavior;
- **Reactivity**— in the sense of being responsive to changes in the environment;
- **Social ability**— the ability of agents to cooperate and coordinate activities with other agents, in order to accomplish their goals.

Agents occupying an environment in isolation, in practice, are rare. The more common case is for agents to inhabit an environment which contains other agents, giving a multi-agent system (MAS).

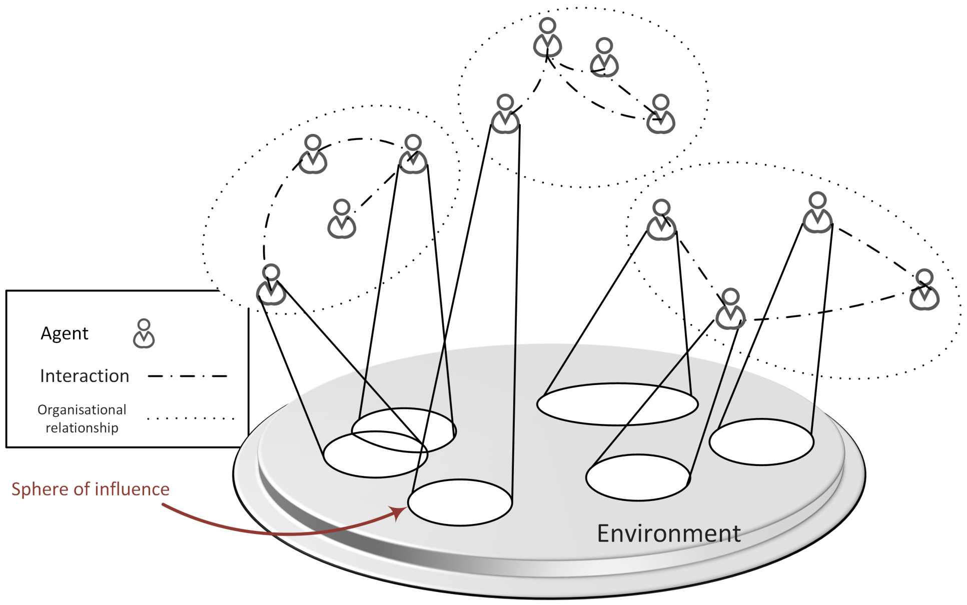


Figure 2.8: Typical structure of a multi-agent system

Each agent has internal sets mechanisms that allows it to reason upon his current "mental" state and the environment. These sets mechanisms define the agent architecture. The next section describes the belief-desire-intention (BDI) architecture and its associated procedural reasoning system (PRS), both approached in this work.

2.2.2 The BDI Architecture and The Procedural Reasoning System

A MAS is composed of multiple interacting intelligent agents within an environment and can be used to solve problems that are difficult or not possible for an individual agent to solve. [53].

An intelligent agent, for the artificial intelligence (AI) community, directs its activity towards achieving goals. A simple agent program can be defined mathematically as a function [62], which maps every perceptual sequence that leads to specific actions (f:$Percepts_n$→Action).

The agent function is an abstract concept that can possibly incorporate various principles of decision making, such as, weighting the utility of individual options, deduction over logic rules, fuzzy logic and so on [63].

The autonomous agent that carries out tasks on behalf of users differ from these single agent reasoning and instead, maps every possible percept to an action. Based on their degree of perceived intelligence and capability [62, pags. 46-54], it is possible to classify them into five classes:

- **Simple Reflex Agents**— The agents act relying on the current percept and the agent function is based on the condition-action rule: if condition then action.

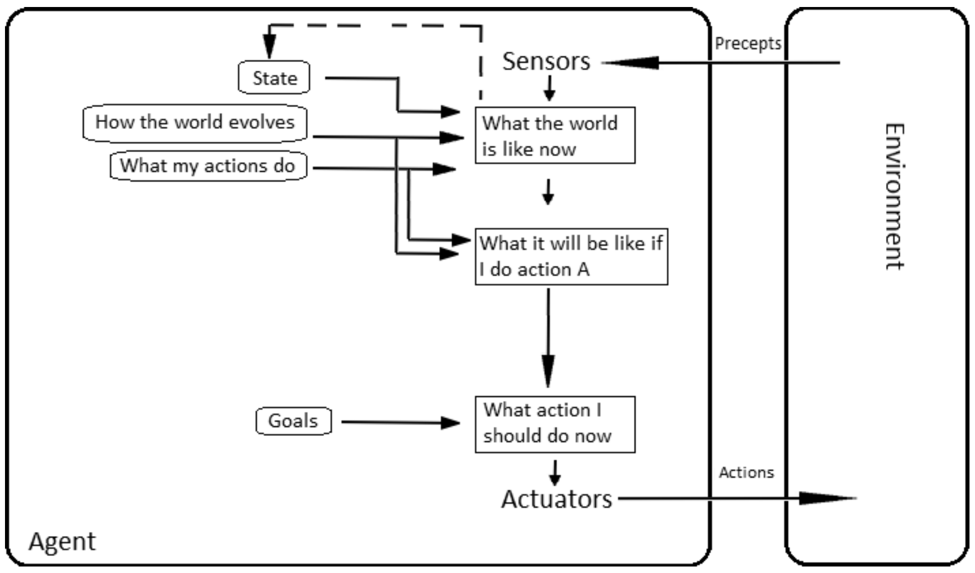


Figure 2.9: Typical Goal-Based Agent structure [62, pag. 52].

- **Model-Based Reflex Agents**— Agents maintain some sort of internal model that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state.

- **Goal-Based Agents**— Goal-based agents further expand on the capabilities of the model-based, by using "goal" information. Goal information describes situations that are desirable.

- **Utility-Based Agents**— the actions for these type of agents are chosen in order to maximizes the expected utility of the action outcomes, that is, what the agent expects given the probabilities and utilities of each outcome.

- **Learning Agents**— initially operate in unknown environments and slowly become more competent than its initial knowledge alone might allow. The most important distinction is the “learning element”, which is responsible for making improvements.

As an extension to the goal-based agent type, illustrated in Fig. 2.9, many agent architectures adopted the Belief-Desire-Intention model, created by [11] to explain human intentional behavior. The BDI model is presented as computer programs that have mental attitudes such as beliefs, desires and intentions [48]. The architecture is closely associated with intelligent agents, but does not ensure all the characteristics associated with them.

“**Beliefs**– are information the agent has about the world.

Desires– are all the possible states of the sequence of events that the agent might like to accomplish. Having a desire, however, does not imply that an agent acts upon it: it is a potential candidate of the agent’s actions.

> **Intentions**– are the state sequence of events that the agent has decided to work towards. Intentions may be goals that are delegated to the agent, or may result from considering options: we think of an agent looking at its options and choosing between them.” [10, chap. 2,pag. 17]

Ultimately, the BDI software model is an attempt to solve a problem that has more to do with plans and planning than it has to do with the programming of intelligent agents. Planning can be viewed from different perspectives:

> “Planning concerns the process by which people select a course of action – deciding what they want, formulating and revise plans, dealing with problems and adversity, making choices, and eventually performing some action.” [61, pag.1]

putting planning as two stages process (planning and control):

> “We define planning as the predetermination of a course of action aimed at achieving some goal. It is the first stage of a two-stage problem solving process. The second stage entails monitoring and guiding the execution of a plan to a useful conclusion.” [34, pag. 275]

BDI agent type architecture are then known by taking agent actions on the basis of the agent goals desirability – a function that ranks alternatives according to their utility – together with the cost of achieving them, weighted by the probability of success. The function helps agents choosing actions and behave more adequately, or even rationally in the basis of the model, providing more than a simple binary decision [62].

2.2.2.1 The Practical Reasoning of the BDI system

Given the introduction to the BDI model and knowing the key data structures of it (beliefs, desires and intentions), the yet to be answer question is: how does an agent go from what he already have to its actions.

The decision-making method underlying the BDI model is known as practical reasoning, the process of figuring out what to do or reasoning directed towards actions. In terms of implementation, intentions are manifested by means of executing one or more plans, which are courses of actions and may include triggers of additional plans.

Goals are desires that have been chosen to be actively pursued by the agent. Another central concept regards event, generated internally or externally to the agent, in which are triggers that may activate plans, update beliefs or modify goals.

Using this concepts, a practical reasoning system (PRS) model implies in adopting intentions and decide how to act in order to achieve the adopted intentions. Viewed by the architectural point of view, the PRS, originally developed at the Stanford Research Institute, was perhaps the first agent architecture to explicitly embody the BDI model. This architecture is shown bellow.

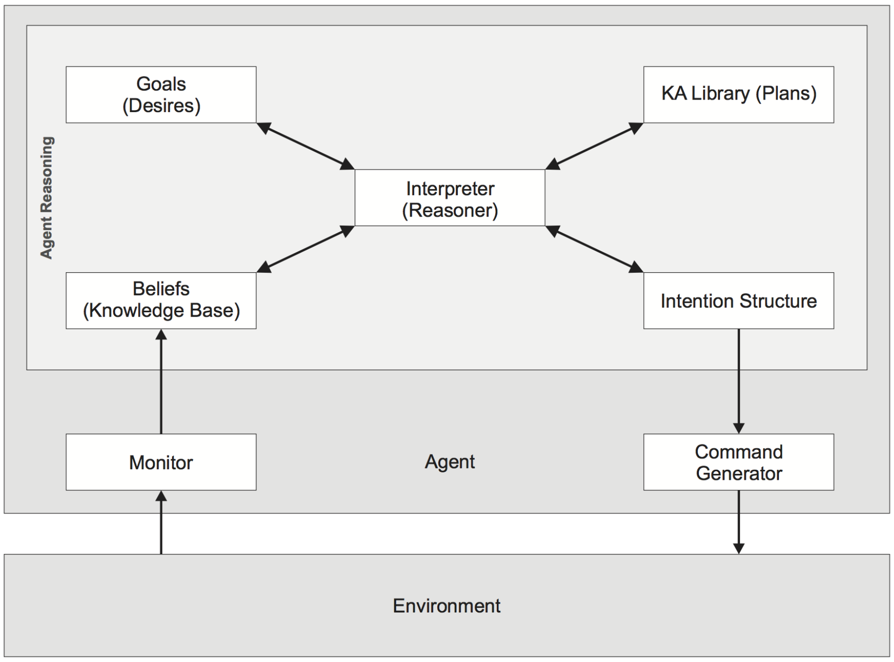


Figure 2.10: The Procedural Reasoning System (PRS) [10]

In the PRS agent architecture an interpreter manage beliefs, goals, plans and intentions. Basically what the interpreter do is update beliefs from observations of the environment, generate new desires (tasks) on the basis of new beliefs, and select from the set of active desires some subset to act as intentions. Hence, the interpreter must select an action to perform on the basis of the agent's current intentions and knowledge .

From the agent viewpoint no planning are made at first in a reasoning cycle . Instead, the PRS is equipped with a library of pre-compiled collection of plans, top-level goals and initial beliefs. The plans are manually constructed, in advance, by the agent programmer. Top-level goals are redirected onto an intention stack, which is responsible to store all achievement pending goals and the beliefs are represented as first-order logic atomic formulas, in which the precise form of atomic formulas depends on the logic under consideration with respect to a given model [35, 36].

> "In mathematical logic, an ***atomic formula*** (also known simply as an atom) is a formula with no deeper propositional structure, that is, a formula that contains no logical connectives or equivalently a formula that has no strict subformulas. Atoms are thus the simplest well-formed formulas of the logic. Compound formulas are formed by combining the atomic formulas using the logical connectives" [49].

The agent then searches through a plan library to see which plans have a goal on top of the intention stack , some will have their pre-condition satisfied and then proceed to post-condition . At this point, the process of selecting a particular plan is result of utility ordering, the chosen plan is then executed. Whether a particular plan to achieve a goal fails, then the agent is able to select another plan to achieve the associated goal from a set of candidate plans.

2.2.3 Agent Programming Languages and Development Environments

From the devising Agent-systems standpoint, the programming language play a fundamental role to the project design and execution, in the sense of providing practical programming languages and tools that are appropriate for the implementation of such systems [9], also for those who rely on its interpretation and extensibility whatever the required field is.

Most agent programming languages rely on platform which implements its semantics. However, these frameworks are not embed with one specific programming language. Instead, they provide general techniques for relevant aspects such as agent communication and coordination [9]. Also, the languages that are in a mature state are aided and carried along by some Integrated Development Environment (IDE), intended to enhance the productivity of programmers.

In spec of that, the basic requirements expected of agent-oriented programming languages rely on managing the inherent complexity of MAS and helping with their development, the research community has produced a number of methodologies [7]. To address some of the existing approaches situated along the lines of declarative, imperative, and hybrid infrastructures, the extent to which researchers have contributed to their development [9] are shown as follows:

Declarative Languages are partially characterized by their strong formal nature, which focuses on what the program should accomplish without specifying how the program should achieve the result [69], some driven on logic others on formalism such as calculus. Languages that fits the profile are: FLUX, Minerva, Dali, ResPect, CLAIM [9].

Imperative Languages is a type of imperative programming in which the program is built from one or more subroutines or functions explicitly listing commands or steps that must be performed. to agent-oriented programming spectrum [69], pure imperative languages are not very common, mainly due most of the agent-oriented abstractions related to design be declarative in nature. however to the non-agent oriented approach, imperative languages are used to develop MAS. An example of an essentially imperative language that incorporate agent-specific abstractions, is the language available with the framework JACK [9]

Hybrid Approaches Various well-known agent languages combine declarative and imperative features (been declarative while at the same time providing some specific constructs allowing the use of code implemented in some external imperative language). The languages that somehow illustrate the hybrid approach are: 3APL, Jason, IMPACT, Go!, and AF-APL [9].

Consequently, by examining the alternatives provided by the survey [9], similar from the presented approaches but now focusing on the elements interrelated with environments development, which must be associated to agent oriented programming and since we are interested in well-defined BDI agents. As the programming languages definitions above suggests, an hybrid approaches shown to be a good candidate to be the utilized as middleware to conduct the premises of this thesis.

Therefore, one alternative would be to explore Jason, an built-in framework for the Eclipse Mars IDE which allows BDI modeling. In fact, the framework present itself as an open source interpreter of an extended version of AgentSpeak, which in terms is a declarative agent-oriented programming language that applies the BDI model. If well-utilized, Jason allows a high-level representation of the agent's AgentSpeak reasoning at the same time make a sophisticated use of object-oriented programming implemented in JAVA [38].Besides, Jason is the standard instance utilized in the CArTAgO documentation [5], a singular and sophisticated common artifact infrastructure to model agent open environments.

The proper use of these two frameworks will be described further in Chapter 3, where Jason is used to define speech acts in AgentSpeak and CArTAgO to interact with the power grid objects in an environment.

2.2.4 Agent Communication

The BDI model abstracts the agent coordination at the environment level, action as a mechanism that handles duplicated activities while preventing two processes simultaneously accessing the same non-shareable resource.

Autonomous intelligent agents that interact within open, distributed, and decentralized environments need to regulate in a collaborative manner their activities in order to facilitate the process of achieving conflicting tasks/activities [67]. Coordination, at the agent level, is the process of managing these interactions by identifying and solving the interdependence between such activities. That said, for a mechanism like BDI to work effectively within a systems, Agents need to communicate.

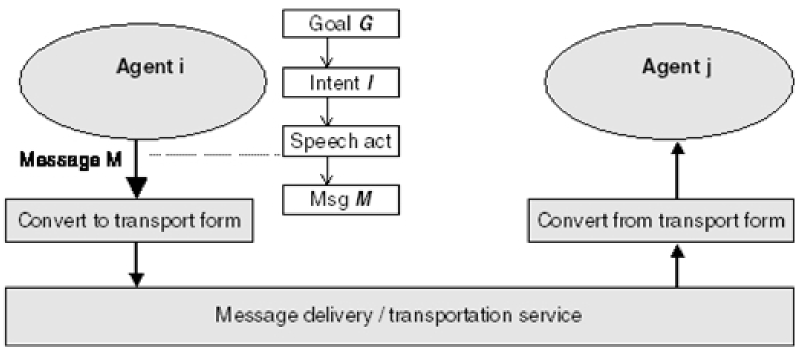
For the ability of social interaction be achieved between agents, it then first necessary that the agents use a common terminology to be understood by all those participating in the speech act. For example, the request of closing switches of "switchAgents", those participating in the discussion should know, for example, what means the term switch or its representation, so the switchAgents knows why its necessary to close it. Meaning that the agents need to know a way to interpret their messages and the means to answer it.

With the intent of standardize some aspects of MAS, including communication, the Institute of Electrical and Electronics Engineers (IEEE) has decided to create an organization to take care of all these details, and decided to call it the Foundation for Intelligent Phisical Agents (FIPA).

FIPA promotes agent-based technology by setting standards based on interoperability between MAS and other technologies, dealing with Agent Communication Language (ACL) messages, message exchange interaction protocols, speech act theory-based communicative acts and content language representations [30].

The agent communication language (ACL) defined by FIPA is very similar to knowledge query and manipulation language (KQML),*"KQML is a programming language and protocol for communication among software agents and knowledge-based systems"* [29, pag. 456]. Code illustrated bellow, source [28].

```
(ask-one
     :content (PRICE IBM ?price)
     :receiver stock-server
     :language LPROLOG
```


Agent i
Goal G
Intent I
Speech act
Msg M
Agent j
Message M
Convert to transport form
Convert from transport form
Message delivery / transportation service
Figure 2.11: Message passing between Two Agents [30]

)

ACL messages can be interpreted as objects (in terms of object-oriented languages, its performatives can be viewed as a message class containing a number of parameters (attributes of the message objec, e.g. achieve, ask-one, tell) [79]. Performatives can also be represented as *tags* that provide a pre-description about the purpose of the communicative act. For example, when an agent wants to report the result of some task to another agent, the performative *Inform* can be used, but when you need a proposal, it is more convenient to send a message with performative *Call For Proposes* (CFP).

Taking the code message exemplified above, the KQML performative is *ask-one*, the content is (price ibm ?price), the a common terminology assumed by the query is identified by the token *nyse-ticks*, the receiver of the message is a server identified as *stock-server* and the query written language is LPROLOG.

All the speech acts, along with their description are found in [30]. Although, not always the format shown is suitable or appropriate for exchanging messages between Agents or between Agents and applications. In such case, the default ACL messages can be translated to a Extensible Markup Language (XML) [31], a standard universally accepted structure that if adopted presents no meaning loss.

2.2.5 MAS applications to Power Engineering

Although a complete review on all power engineering applications to MAS is out of the scope of this work, it is important to emphasize that thoroughly literature revision on agent-based systems applied to power engineering has been found in [38], where a extensive analysis based on surveys of the IEEE Power Engineering Society's Multi-Agent Systems Working Group [46, 47] were made, regarding conferences publications (Intelligent Systems Applications to Power Systems (ISAP) conferences), publications covering Journals of IEEE transactions, the Institution of Electrical Engineers (IEE),the Institution of Engineering and Technology (IET), the Electric Power Systems Research (EPSR) as well as International Journals of Electrical Power & Energy Systems (IJEPES).

Based on what has been stated and further analyzed in the document cited, until the year of 2012, the extent of work object of citation cover up to 100 publications related to MAS applied on various sectors related to power Delivery, Generation, Transmission and Distribution. The distribution portion reached 23 related documents, according to Table 2.1.

Table 2.1: Bibliographic survey of multi-agent systems applied to power engineering problems

	ISAP 2001–2003	ISAP 2005–2007	ISAP 2009–2011	IEEE/IEE IET Journals	EPSR/IJEPES Journals	Total
Generation & Transmission	5	15	15	39	3	77
Distribution	1	5	6	8	3	23
Total	6	20	21	47	6	100

In comparison with other research fields, these figures suggest that the relationship between agent technologies and power engineering is not yet at a mature state. This is probably related to the inherently complexity of power engineering problems and how agent-based systems have conceptually evolved in the past few years together. Also, it might be related to the interdisciplinary requirements to building such MAS applications and the complexity aggregated to adding new paradigms to power delivery.

Concerning the review, most of cited applications reinforced the use of decentralized systems. Furthermore, the a great amount of work that explores decentralized approaches happens to be featured by Multi-Agent-Systems. These efforts cover from specific solutions proposed to improve the protection until abstract frameworks devised to manage entire power systems.

Amongst the applications to MAS, the works in [2, 10, 10, 38] were the main references utilized in this work, where a block-oriented agent-based architecture is completely designed using AgentSpeak and CarTAgO to the purposes of operation control.

2.3 Conclusion

A review about the current status of power grids as well as power distribution system operation and control has been made, focusing on aspects from the distribution viewpoint that entails operation, management, automation and reconfiguration. Regarding agent systems, it has been shown that in order to develop such systems, it is necessary to consider functional particularities of agent structures, agents reasoning systems as well as agent communication processes. It is important to emphasize the BDI architecture, which is one of the options that enable viewing an agent as a goal-directed entity that acts in a rational manner.

An introduction on how agent-based system can be implemented in terms of languages and development environments has been briefly stated. This was discussed addressing some of the exiting approaches and their corresponding platforms. The Jason declarative agent-oriented modeling, or AgentSpeak, along with CArtAgO framework for agent systems modeling are introduced as platforms that provides an integrated Agent architecture within an open simulated environment.

This brought up the potential utility for their applications, motivating the further developments described in the next chapters

Chapter 3

Developed Approach

This chapter introduces the modeling of the developed approach for AR using MAS coordination. Reconfiguration solutions have been modeled either considering a hierarchical approach, where decision making is centralized in an entity provided with a Dijkstra's shortest paths algorithm application, as well as a decentralized approach, where grid self-organization is reached through agent interactive procedures.

For this accomplishment, a simulation model has been designed to simulate the system operation and provide information regarding steady-state and fault conditions. This required the modeling of network components as computational objects aiming at integrating system state transitions at the MAS simulation model level. Then after, agent capabilities, plans of action and reasonings have been modeled using AgentSpeak formulations, while environment modeling has been addressed through agent-artifacts representations. All these models have been included in a simulation platform, where decentralized solutions can be tested and verified, and the independence of agent/component processes is guaranteed by the JaCamo framework.

The chapter is organized as follows. In section 3.1, distribution grid modeling and MAS modeling is thoroughly described, focusing on how components are represented, included and integrated in the simulation platform. In section 3.2, MAS capabilities and planning procedures are described with emphasis on how AR is modeled and achieved in the developed approach. In section 3.3, conclusions and final remarks are outlined.

3.1 System Component and MAS Modeling

The global simulation model is constituted by a set of AgentSpeak plans that coordinate the speech act and agent interactions within the environment imposed to it. Moreover, it includes a Java Application Programming Interface (API) method elaborated and supported with a set of Java based libraries allowing relations with external programming platforms chosen to perform grid calculations. Lastly, it includes a built-in MAS framework platform.

As the MAS abstraction suggests, before creating the agent level representation of the problem, an environment must to be defined. Being power distribution grids the object of study in this work, the adopted solution to fulfill this requirement was by making usage of an external power grid simulation software and finding means to integrate it to the MAS framework. The choice was for the Electric Power Research Institute (EPRI) freeware *Open Distribution System Simulator* (OpenDSS) [24].

The Integrated Development Environment (IDE) used as basis to run the MAS development workspace framework was *Eclipse Mars 4.3* [73] and the framework chosen was *JaCaMo* [8], that is essentially a MAS development tool consisting of a set libraries/classes that implement the BDI model, the AgentSpeak coordination with the model, the environment and its features.

3.1.1 Distribution Grid Modeling

OpenDSS is a script-driven power grid simulation tool used in distribution planning and analysis of multi-phase AC circuit systems [22]. The software can be executed as both stand-alone mode or be called through an in-process Component Object Model (COM) server dynamic-link library (dll), as shown in Fig. 3.2, that was designed to be driven from a variety of existing 3rd party analysis program that can handle COM. Users commonly drive the engine with Mathworks MATLAB, Python, C#, Java, and other programming languages platforms.

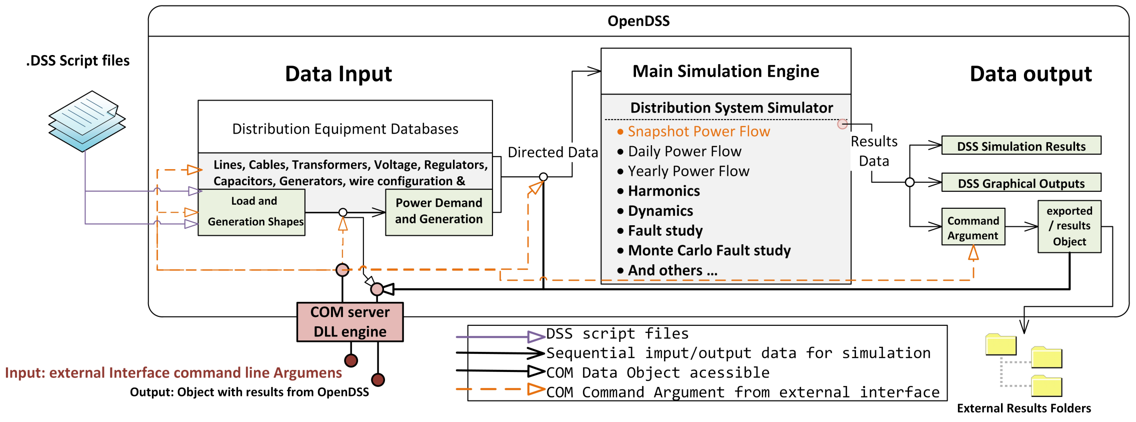

OpenDSS
.DSS Script files
Data Input
Distribution Equipment Databases
Lines, Cables, Transformers, Voltage, Regulators, Capacitors, Generators, wire configuration &
Load and Generation Shapes
Power Demand and Generation
Directed Data
Main Simulation Engine
Distribution System Simulator
• Snapshot Power Flow
• Daily Power Flow
• Yearly Power Flow
• Harmonics
• Dynamics
• Fault study
• Monte Carlo Fault study
• And others ...
Results Data
Data output
DSS Simulation Results
DSS Graphical Outputs
Command Argument
exported / results Object
COM server DLL engine
Input: external Interface command line Argumens
Output: Object with results from OpenDSS
DSS script files
Sequential imput/output data for simulation
COM Data Object acessible
COM Command Argument from external interface
External Results Folders

Figure 3.1: OpenDSS stand-alone and COM simulation structure.

The COM interface allows one to develop or abstract algorithms in another programming platform and then drive the OpenDSS dll engine to do something that it is not implemented within the main script. The external algorithms rely on the OpenDSS to represent the distribution system behaviors while adjusting variables to be optimized.

In order to characterize the distribution power grid, the default OpenDSS script files for IEEE123 test feeder were used to define the electrical and physical input data as well as the load shape and wiring data and line codes, according to [20]. The power grid is shown in Fig. 3.2.

> "The IEEE 123 node test feeder operates at a nominal voltage of 4.16 kV. While this is not a popular voltage level it does provide voltage drop problems that must be solved with the application of voltage regulators and shunt capacitors. This circuit is

characterized by overhead and underground lines, unbalanced loading with constant current, impedance, and power, four voltage regulators, shunt capacitor banks, and multiple switches. according to [20, pag.1]."

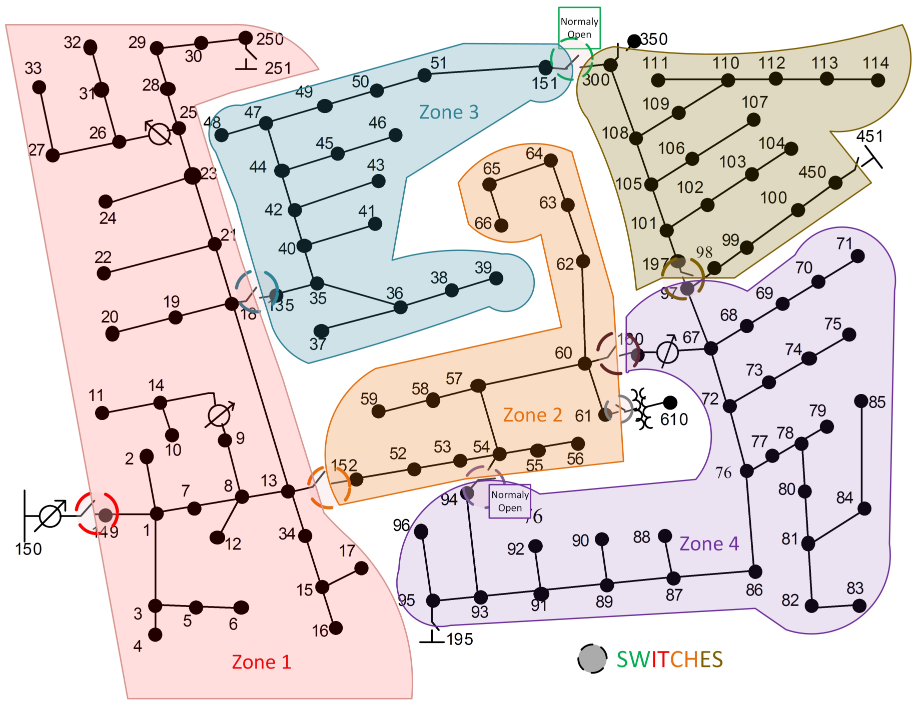


Figure 3.2: IEEE123 Test Feeder

From AR conceptual analyses, the series of switches and lines that composes the distribution system are of crucial importance for modeling the MAS simulation environment. Therefore, the problem itself holds them accountable for grid alterations and system behavior variations. Switch starting status and basic considerations are represented according to Table 3.1.

Table 3.1: Switch Data table

Represented in the the Model	Three Phase Switches		
Yes/No	Node A	Node B	Normal
Yes	13	152	closed
Yes	18	135	closed
Yes	60	160	closed
Yes	61	610	closed
Yes	97	197	closed
Yes	150	149	closed
No	250	251	open
No	450	451	open
Yes	54	94	open
Yes	151	300	open
No	300	350	open

Some non-representations of switches in the model is justified based on fundamental aspects related to validation data, as if they were not worth of consideration for not presenting load shape from the second end attached bus, or no load at all, as well as not presenting relevancy from the radial circuit redundancy aspect. From a OpenDSS scripting language syntax that defines circuit elements as shown in the algorithm 3.1, considerations on how switches are represented must also be referred, in the sense that normally open and closed switches can be characterized as short lines that after being stated can be enabled or disabled.

Algorithm 3.1: OpenDSS scripting language syntax for Switch /Line Definition

```
!example of line
New Line.L109 Phases=1 Bus1=109.1 Bus2=110.1  LineCode=9 Length=0.3
!example of switches definition as line
New Line.SW1 phases=3 Bus1=150 Bus2=149 r1=1e-3 r0=1e-3 x1=0 x0=0 c1=0 c0=0 Length
    =0.001
```

Each electrical element in the distribution system has one or more terminals, associated to one or more conductors. Each conductor conceptually contains a disconnect switch and a fuse, which can be enabled or disabled, as shown in Fig. 3.3. Analyzing the OpenDSS grid model, the general

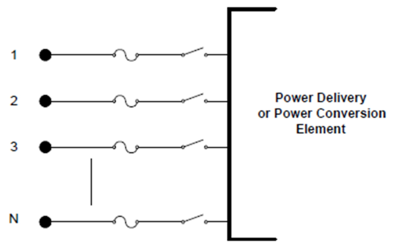


Figure 3.3: Terminal Definition [22]

bus wiring and line code vary according to the documented configuration model [20], described as in Table 3.2 and represented in Fig. 3.4.

Table 3.2: Grid Line codes

Config.	Phasing	Phase Cond. ACSR*	Neutral Cond. ACSR*	Spacing ID
1	1 2 3 N	336,400 26/7	4/0 6/1	500
2	3 1 2 N	336,400 26/7	4/0 6/1	500
3	2 3 1 N	336,400 26/7	4/0 6/1	500
4	3 2 1 N	336,400 26/7	4/0 6/1	500
5	2 1 3 N	336,400 26/7	4/0 6/1	500
6	1 3 2 N	336,400 26/7	4/0 6/1	500
7	1 3 N	336,400 26/7	4/0 6/1	505
8	1 2 N	336,400 26/7	4/0 6/1	505
9	1 N	1/0	1/0	510
10	2 N	1/0	1/0	510
11	3 N	1/0	1/0	510

ACSR* stands for Aluminium-conductor steel-reinforced cable

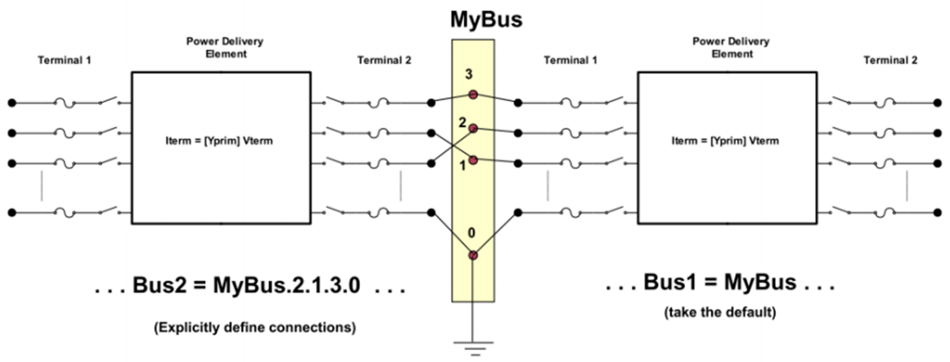


Figure 3.4: Wiring and bus representation [65]

As Java represents the main programming language of the overall MAS concept model, the OpenDSS integrates the model supported by an API method. The API makes possible for applications like OpenDSS to run on top of a framework such as JaCaMo. That being stated, the OpenDSS package API gives access to the program Executive and Circuit element models as featured in Fig. 3.5.

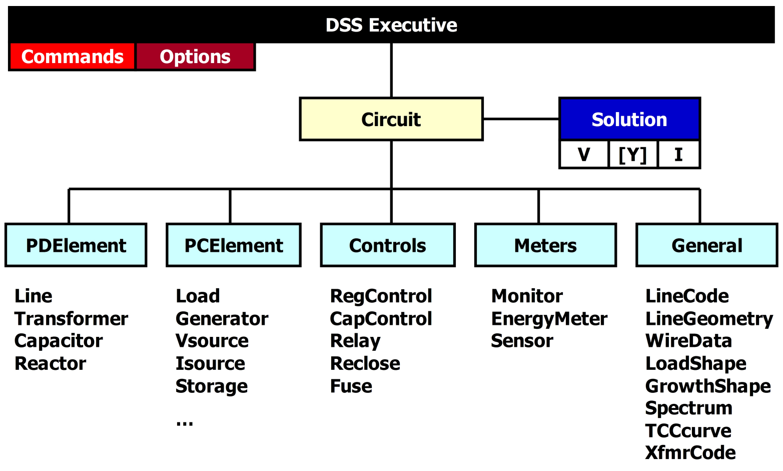


Figure 3.5: OpenDSS Executive solution modes and features [22]

APIs are essentially developed to perform tasks which allow communication with services and their integration to other services. Third-party programs can use API to take advantage or extend the functionality of the existing services [12]. When an API is related to a software library, the classes comprised inside it define the program behavior while the library is an actual implementation. Therefore, an exported or external package can define a specific API library class, containing Java interfaces and classes, as it is shown in the following diagram Fig. 3.6.

From the OpenDSS API method perspective, the services in it are expressed as functionalities related to the simulation tool intended to make part of the built-in MAS framework. In basic

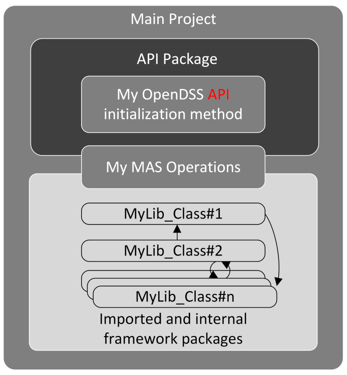


Figure 3.6: API method explained

terms, the API method sets the requirements that govern how the framework call-method can talk to the simulation engine and designates what makes possible to move information between the two platforms — for instance, by passing text commands and receiving return calls from delegated DSS commands. The OpenDSS API executive initiation set is shown in Algorithm 3.2.

Algorithm 3.2: API engine initiation

```
package toll
  import OpenDSS.*;  // API library class package
  public class artSimTool extends Artifact { ...
    /**initializing API executive interface classes**/ ...
    private static  OpenDSS.IDSS dss =  ClassFactory.createDSS();
      OpenDSS.IText               dssText      = dss.text();
      OpenDSS.ICircuit            DSSCircuit   = dss.activeCircuit();
      OpenDSS.ILines              DSSLine      = DSSCircuit.lines();
      OpenDSS.ICktElement         DSSelementCk = DSSCircuit.activeCktElement();
      OpenDSS.IDSSElement         DSSelement   = DSSCircuit.activeDSSElement();
      OpenDSS.ISolution           DSSSolution  = DSSCircuit.solution();
      OpenDSS.IBus                DSSBus       = DSSCircuit.activeBus();
      OpenDSS.ISwtControls        DSSSwt       = DSSCircuit.swtControls();....
```

From the framework perspective, the act of passing command and retrieving oriented objects data are shortly exemplified as follows:

Algorithm 3.3: API method directives

```
public void runSimulation(List<String>cmd,int sysChose) throws IOException{
  /************      COM handling  setting up directives  ***************/
      dss.dataPath(System.getProperty("user.dir"));
      dss.allowForms(false); // disallow the engine GUI calling
  /********    initialize simulation tool directives handling  ********/
      dss.start(0);
      dssText.command("clear"); // clear buffer for new simulation
  ...
         pathFind = System.getProperty("user.dir").replace("\\", "/" )+"/lib/";
String origin = pathFind + "123Bus/ieee123.dss"; // set dss script path
/********************      RUN simulation      **********************/
dssText.command("compile ["+origin+"]");
  //if solution is valid print the event log is Generated
  if(DSSSolution.converged()) {
  // Acessible Objects Power Flow solution
  String V = Arrays.deepToString(((Object[]) DSSSolution.eventLog()));
  int gridBusSize = DSSCircuit.numBuses();
      dssText.command("Show Elements");   // Object String value
      dssText.command("Export Voltages"); // Object String value
      dssText.command("Export Currents"); // Object String value
  ... }
}
```

Post-processing methods have also been created, fundamentally to task direct and redirect outputs and exported command-generated files such as Comma-Separated Value (CSV) and text files.

3.1.2 The JaCaMo MAS Modeling

The framework basis is a programming model named JaCa specifically created to implement the BDI model of AgentSpeak. The model unites the two frameworks named upon the acronym that stands for JAson-CArtago, Jason [10] is adopted as programming language to implement and execute the agents model and CArtAgo [5] as the framework to program and execute the environments.

A MAS designed based on JaCa principles is modeled as a set of agents which work and cooperate inside a common environment. Implementing the application means programming the agents on the one side, including the actions control logic to be executed relying on the environment changes, and on the other side the environment itself. That all stated, the notion of environment is solely based on real world concept to be explored according to the user problematic [58], i.e. *the environment here is part of the software system* to be developed .

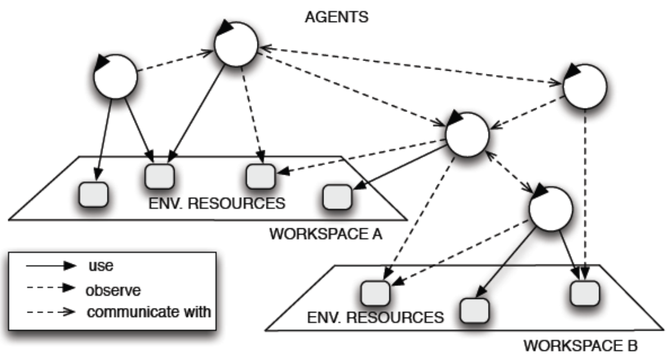


Figure 3.7: JaCa approach [8]

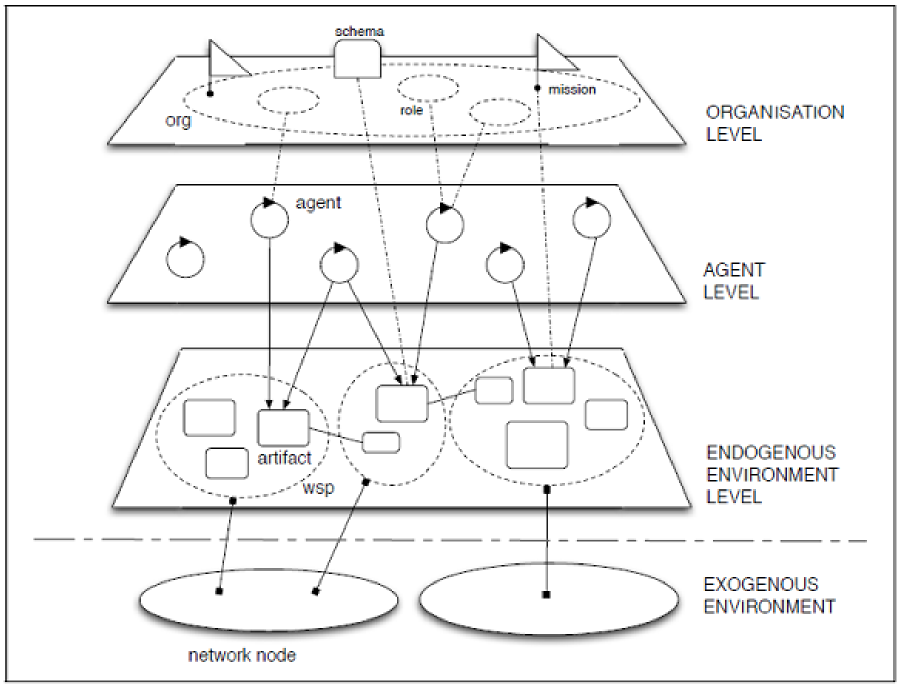


Figure 3.8: JaCaMo approach [8]

Now, a designed JaCaMo system is given by a Moise organization of autonomous BDI agents, implemented in Jason, working in shared artifact-based environments implemented in CArtAgO, as depicted in Fig. 3.8. Therefore, the JaCaMo framework approach establishes the coordination between three models JAson-CArtago-MOise, in which each represents substantial contribution to the framework as a whole, that is:

Jason library implements AgentSpeak development workspace and extends it by interpreting the running code of Jason's Agent language to be compiled by JaCaMo runnable file (. jcm).

Cartago library are responsible to define the environment and link to the Agents dimension through Artifacts operation, using a proper set of commands to define each operation and observable proprieties of the environment.

Moise library [39] used to orchestrate the organization between agents and environment observable proprieties.

The JaCaMo MAS approach has been conceived by utilizing the default Moise organizational model of autonomous BDI agents, and a set of Jason AgentSpeak language codes working in a shared Artifact-based environment dimension created, implemented in CArtAgO artifact extensions by the same means Java methods are implemented.

AgentSpeak is a discipline of computational sciences and some details in terms of approach, at the modeling development, might differ from power engineering point of view as the process of learning and implementing a MAS model goes. Therefore, as an attempt to introduce the reader some of the details and basis of how the solutions are actually created, a brief explanation of the Jason language and interpreter will be presented, skipping the complex issues for the ease of the reader. For more information, a proper reference can be found in [10].

The main idea to be addressed when comes to model agents, is talk of computer programs as if they have a "mental state". Thus, when we talk about a BDI system, we are talking about computer programs with computational analogues of *beliefs, desires and intentions*. Unlike in classical logic, the concept to be learned here is of modalities of truth, rather than things that are stated to be true in absolute terms. Therefore, the basic steps of an agent abstraction based on the PRS model, outlined in Chapter 2, is the usage of deliberation and means-ends reasoning on the basis of the control loop, in which an agent continually:

- Looks at the world and updates beliefs based on events that occur in it, Agents receive events either external from the environment (perceptual data) or internally generated (beliefes);
- Deliberates to decide what intention to achieve, by tring to handle events by looking for plans (desires) that match;
- Uses means-ends reasoning to find a plan to achieve this intention;
- Executes the plan.

Belief sets are then part of agents structure, in which represents a collection of literals, where each one of them is represented by predicates in a symbolic form:

`voltage(Phase1,1.02)`$[source(self)|source(percept)]$`.`

This means that an agent with such *belief*, acquired either from what he is sensing of its environment sphere of influence or what himself somehow concludes, determine as true "Phase1 has voltage 1.02 p.u." until it changes – p.u. here is implicit and means voltage electrical quantity per unit system. However, in the context of agents, this will never be known unless it is stated as a belief. That is, the existence of the literal *voltage* in the agent's *belief* base means that the agent relate the internal terms that defines it as true.

The *desires* are previously devised by the MAS developer in advance, by giving the agent information about how to respond to events and how to achieve goals. A desire structure then comprehends: an event or *trigger* that the plan can handle, a *context* (rule) that defines the condition under which the plan can be used and a *body* (course of action) or the the actions to be carried out if the plan is chosen, as follows:

`+!triggering_event`$[source(percept|agent_n|self)]$`:` context <- body.

The triggering events are identifiable changes in *beliefs* or *Goals*, and such changes can trigger the execution of plans. Contexts will then represents logical conditions which defines a plan applicability or not, i.e. candidate for execution, and bodies are the course of actions (actuation commands). The different types of triggering events and context literals are shown in Tables 3.4 and 3.3.

Table 3.3: Types of literals in plan context

Notation	Description
literal	The agent believes literal is true
∼ *literal*	The agent believes literal is false
not *literal*	The agent does not believe literal is true
not ∼ *literal*	The agent does not believe literal is false

"Goals: predicate formulae prefixed with "!" identify what the agent wants to do. Whereas beliefs, in particular those of a perceptual source, express properties that are believed to be true of the world in which the agent is situated, goals express the properties of the states of the world that the agent wishes to bring about. Goals are equivalent to method calls in object-oriented programming. The mapping is achieved through the use of events and associated event handler, known as plan rules." [10, adapted from pag.40]

Table 3.4: Types of triggering events

Notation	Description
+*literal*	Belief addition/Variation
-*literal*	Belief deletion
+!*literal*	Achievement goal addition
-!*literal*	Achievement goal deletion

In means-ends analysis, the Agents reasoning begins by envisioning the end, or ultimate goal, and then determines the best strategy for attaining the goal in his current situation. Therefore, the desires are represented as the set of plans that match the event or the possible options of executions in with differentiate between them by triggering rules they carry. That said, the Agent will then find a plan to achieve this intention by trying to handle events by looking for plans that match a certain condition of the environment. An example of handling desires upon a belief perception of the environment is devised as the algorithm suggests:

Algorithm 3.4: *belief* environment change percept illustrate

```
\\ belief triggered if environment percept changes and if its locate in Phase1
 +voltage(A,B)[Source(percept)]: A == "Phase1" <-
            .print("Testing Voltage Adequacy Phase:",A);
            !testBelifadequacyVoltage(A,B). \\passing  belief to a course of action
```

What the Agents perception of the environment creates when changed and if matched its pre-conditions, for this particular case where the belief is triggered only if (A) condition relates with "Phase1", basically is calling a desire loop – triggered when first called, passing from an idle goal (false) to a active goal (true) – in which will further decide whats the best action to take.

Algorithm 3.5: Completion and retention of Agent possible intention

```
/*******Decisions Loop: ***********/
+!testBelifadequacyVoltage(A,B): true <- !testBelifadequacyVoltage(A,B).
+!testBelifadequacyVoltage(A,B): B>0.95 & B<1.05
        <- .print("Report only: Voltage Levels whiting Regulatory Standard Range").
+!testBelifadequacyVoltage(A,B):  B<0.95 | B>1.05
        <- .print(">> FALT ALERT!  voltage out of tolerance Range <> ",B);
            handle_Situation_by_Calling_Actuator_internal_Function(A);
           .send(dMS,tell,Occurrence(A,B)).
```

The process of actuating upon environment conditions, here handling a possible undervoltage or overvoltage, is defined as *context* (desires conditions). That is, if B – the voltage value sensed from the environment– are withing 0.95 and 1.05 (p.u.) the Agent choosing desire will be reporting the environment current state, otherwise if B presents below or above the tolerable condition the course of action to be taken will be the one in with deliberation orders to the agents actuator, by calling its own fix routines and sending a message with *Occurrence* label to be interpreted by the *dMS* Agent.

In very short terms, at every reasoning cycle of an programmed agent, the interpreter updates the events list, which may be generated from perception of the environment, or from the execution of intentions (when goals are specified in the body of plans). Beliefs are either updated from perception or from plan controlled operations and whenever there are changes in the agents beliefs, this implies the insertion of an event in the set of events. Check the PRS scheme illustrated in Fig. 3.9.

The Agent-Artifact Model

All elements beyond the scope of the Agents in a MAS application are typically considered to be part of the environment. Such elements can be relented to databases, constructed communication infrastructures, topology of a spacial domain and so on [78].

By definition, CArtAgO means Common ARTifact infrastructure for AGents Open, allowing users to model and and execute virtual environments for Artifact-based MAS in open workspaces were agents can join and work together. CArtAgO is based on the Agents-Artifacts (AA) multi-agent systems environment modeling and designing abstraction. AA introduces agents as computational entities performing some kind of task or goal-oriented activity, and artifacts as resources and tools dynamically constructed, used and manipulated by Agents to support their individual and collective activities [1].

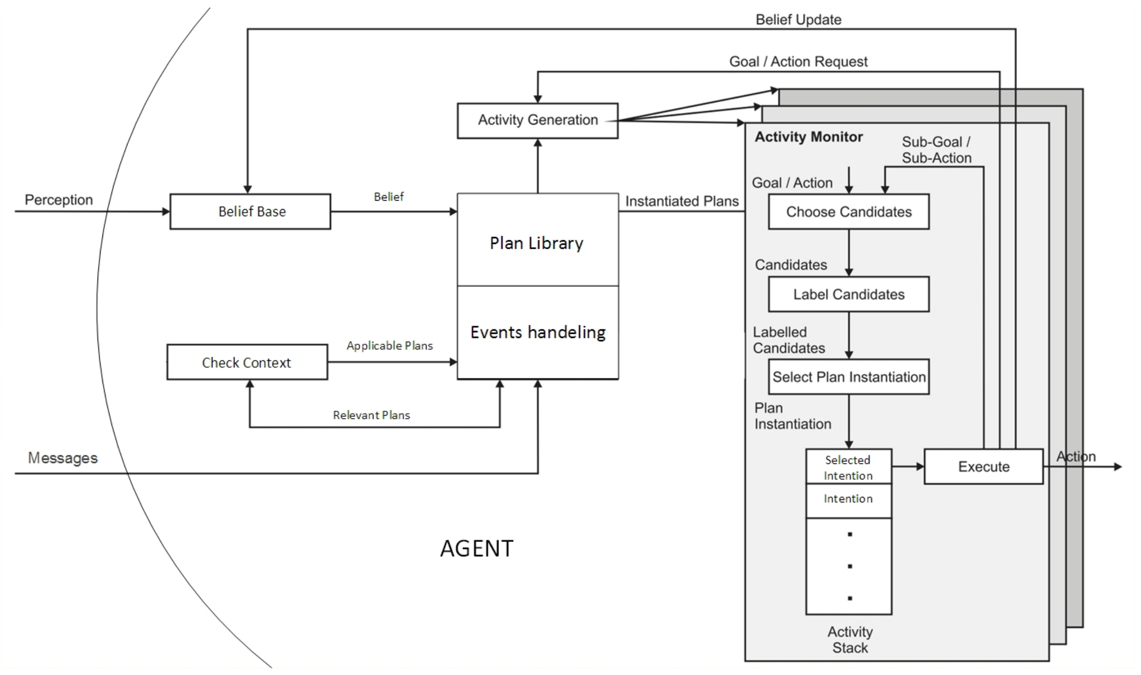


Figure 3.9: Agent reasoning cycle. after [10, pag. 68]

Developers of MAS have then a simple framework to design and implement Agents computational environment [1]. Artifacts, as the encapsulated representation suggests in Fig. 3.10, have been proposed as first-class abstractions to model engineer computational environments. Knowing that, environments can plays a fundamental role to the overall application, as Artifacts handles responsibilities impacting on the design and development.

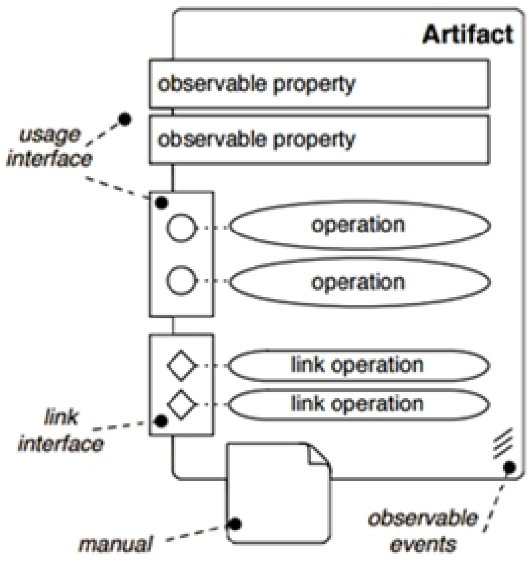


Figure 3.10: Artifact Model [78]

The workspace environment, portrayed as endogenous environment level in Fig. 3.8, can be populated by one or more workspace entities (Artifact/Agent). These entities form the abstraction layers that rely upon the observable environment proprieties or usage interface. Defining Artifact's usage interface means then setting observable properties and programming a set of operations that mimic the user intended goal or actuation command. Operation or variability in the simulated environment can generate updates to the observable properties and specific observable events. The last entity concerning the environment dimension is the manual, an entity used for representing the description of the functionalities provided by an Artifact.

The Application of the AA concept to the MAS developed, implicate in the creation of methods that would help agents to coordinate communication and their behaviors, as the Algorithm 3.6 suggests.

Algorithm 3.6: Artifact Example

```
 @OPERATION public void init(...) throws InterruptedException {
   defineObsProperty("statusVolt", new Atom("Coud Not reach Voltage sensor"));
   defineObsProperty("statusCurrent", new Atom("Coud Not reach Current sensor"));
   defineObsProperty("zone", new Atom(Zone));
   defineObsProperty("voltagePhase1", 0.00);
   defineObsProperty("voltagePhase2", 0.00);
   defineObsProperty("voltagePhase3", 0.00);
   ...}
 @OPERATION public void incSim() {}
 @OPERATION public void getBusRealName(OpFeedbackParam<String> name) {...}
 @OPERATION public void attachedTo() {...}
 public double randDouble(double bound1, double bound2) {...}
 public static String readLine(int line, String File){...}
...
```

The artifcat modeling, illustrated inFig. 3.10, is defined by a set of observable proprieties that are devised according with the characteristics of the problem and the environment observed. Also, it includes a set of operations which can be either to actuate on the environment or to aid agents to archive their tasks. This is possible since the operations and agents share the same level of abstraction, and the ways operations are implemented depends only on how the user abstracts its platform. This link between agents and operations is generally provided in case agents use AgentSpeak (and proper agent language to communicate with their artifact) as well as artifacts use Java language.

3.2 MAS Capabilities and Agent Plans Towards Reconfiguration

Achieving an acceptable integration within the simulated tools required a considerable level of abstraction when it comes to specifying the *goals* –intentions– over the activities behind reconfiguration operations. In this context, Jason AgentSpeak has been used to model the basis of agent-agent communication and the interaction at agent-artifact level. CArtAgo has been first used to define an artifact that held the API method, which in turn links, initiate and drive OpenDSS via COM engine inside JaCaMo framework. The artifact based library has been also applied to implement operations to be directed through the COM engine and operate on behalf of agents, supporting them to achieve their *goals*. The API was written in native Java and the set of operation is modeled in CarTAgO.

3.2.1 MAS Structure for Automatic Configuration

This section describes the MAS structure utilized for both hierarchical and fully decentralized AR approaches. Such modling required the specific use of each JaCaMo libraries, which come with the necessity of representing the external distribution grid environment. That stated, in order to put the reconfiguration into perspective, simplification and certain assumptions to make the scenario suitable are summarized as follows:

- Power flow snapshots have been utilized to model network voltages and currents in steady-state;
- Short-circuit analyses have been utilized to model network voltages currents for falty conditions;
- All switches have capacity to support short-circuit conditions and are able to be remotely controled using utility communication systems. Also, fault passage indicators are provied near each switch.
- A communication system is avaiable to support the operation procedures.

The agent types considered in the MAS are classified as:

- Ag_busNumber*: Represent at most each and every existing bus in the system, bus agents will be responsible to be constantly reasoning upon their status and regarding failure it must be capable of autonomously handle it. They can be modeled to handle operations in an actual network bus or manage a set of buses in a zone.
- simulationCall Agent: Responsible to coordinate the simulation (using steady-state and short-circuit analisis) and in build the MAS infrastructure (grid environment and bus Agents).
- dMS (Distribution Management System) Agent: responsible for management and simulation delegations such as to ask for simulationCall to perform fault simulations and to reason about grid health by constantly keep contact with busAgents, which in turn provide access to electric quantities (voltage and currents).

In the agent-abstraction level, using JaCaMo layered structural approach (see Fig. 3.8), the following considerations must be emphasized:

Cycle.1 The Exogenous Environment Characterization: The MAS framework integration with the OpenDSS tool must be established allowing a simulated open workspace grid. Hence, the environment is set by an agent created for this very reason – here referred as *simulationCall* – and responsible for grid building and busAgents initialization. This is devised by:

- Phase 1: The initialization of the external simulation tool using an Java API;
- Phase 2: Building the distribution grid and run the steady-state simulations, whose results are stored in text and CSV files.

Cycle.2 The Endogenous Environment Characterization: When integration is established and the grid objects are observable, agents can now take place in the workspace where its objects are. This cycle features 2 sub cycles:

- Phase 1: Each agent is responsible for at least a bus and an artifact-sensor, exempt agents responsible for the simulations (Agents: simulationCAll and dMS). When instumentation is available, each agent must be capable of identifying:
  - (A) Voltages - Status Phases Phase 1, 2 and 3 ;
  - (B) Currents - Status Phases Phase 1, 2 and 3;
  - (C) Zone - 1, 2 ...;
  - (D) Switches zone status On/Off of the zone;
  - (E) Identifiable neighboring agents such that a bound amongst then is establhised by communication act.
- Phase 2: Setting/coordinate sensorial analyses for bus agents, where each bus agent must be capable of sensing corresponding voltages and currents and perform further analysis upon its values such as:
  - (A) Voltages - Overvoltages or undervoltages, and so forth;
  - (B) Currents - Overcurrents, line overloading, and so forth;
  - (C) Zone - read zone and interpret switches positioning;
  - (D) Switches - capability of handling fault conditions either if zone switches are energized or not;
  - (E) In adversity, capacity of handling isolation and coordinate solution along side neighboring agent.

Cycle.3 The Reconfiguration Characterization: If a fault condition for an section of the grid is assigned, the MAS system must be capable of autonomously adjust – locally and systemically – handling zone and perform line isolation determining courses of action to reconfigure the grid, assuring continuity of supply to loads and secondary feeders. These activities rely on:

- Phase 1: Perform fault isolation: after fault, a local busAgent must handle communicating acts with busAgents attached to switches in order to isolate the zone.

- Phase 2: Perform reconfiguration coordination: once the local faulted agent confirms the zone is properly isolated, other agents start a coordinating to find ways to reconfigure.
- Phase 3: Finalize the reconfiguration by reading bus status againi f reconfiguration is performed adequately. Otherwise the system remains in a faulted state and looking for solutions.

Having these structures in mind, a specification diagram using Unified Modeling Language (UML) was developed to aid designing and visualizing the MAS. This diagram maps Agents relation also giving specifications to the created MAS as shown in 3.11.

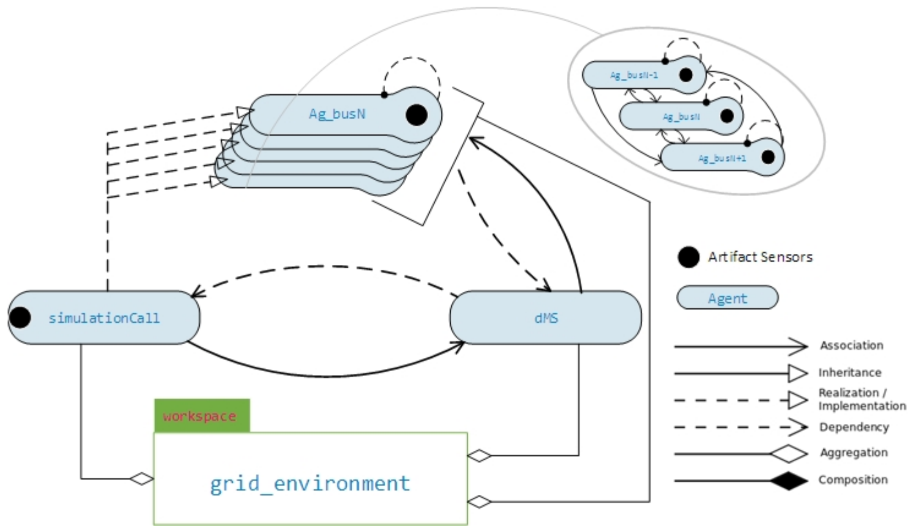


Figure 3.11: Structural Specification UML diagram

In the interactions shwon, proposed action plans approval by the *dMS* before any action is not necessarily imposed or either reinforced. Agents are programmed to report all activities back to the *dMS*, which has the possibility of overriding decisions. These directives might be of great relevance to gathering data about the system operations, testing protection and control settings derived from the executed plans, tuning and validating agent decisions, as well as improving its degree of acceptance. The dependencies in the model are related to the necessity on acting toward grid equipment state, and the fact that the *dMS* share the same simulation Artifact as simulationCall.

The *simulationCall* holds a communication role as well as goal-oriented activities to perform distribution grid simulations, only possible due the link to an .asl file, that represents the main Agent module and where algorithms run the Agent-Artifact concept through CArtAgO framework. To give a more comprehensible insight to what the simulationCall does, a UML diagram of the agent is presented in Fig. 3.12.

On the UML diagram it is possible to verify the design of simulationcall agent. In order to build a grid environment in which practically all data can be acted upon through artifact operations, agent-artifact main operations are the follwing: *start(_)* to start the runs steady-state simulation,

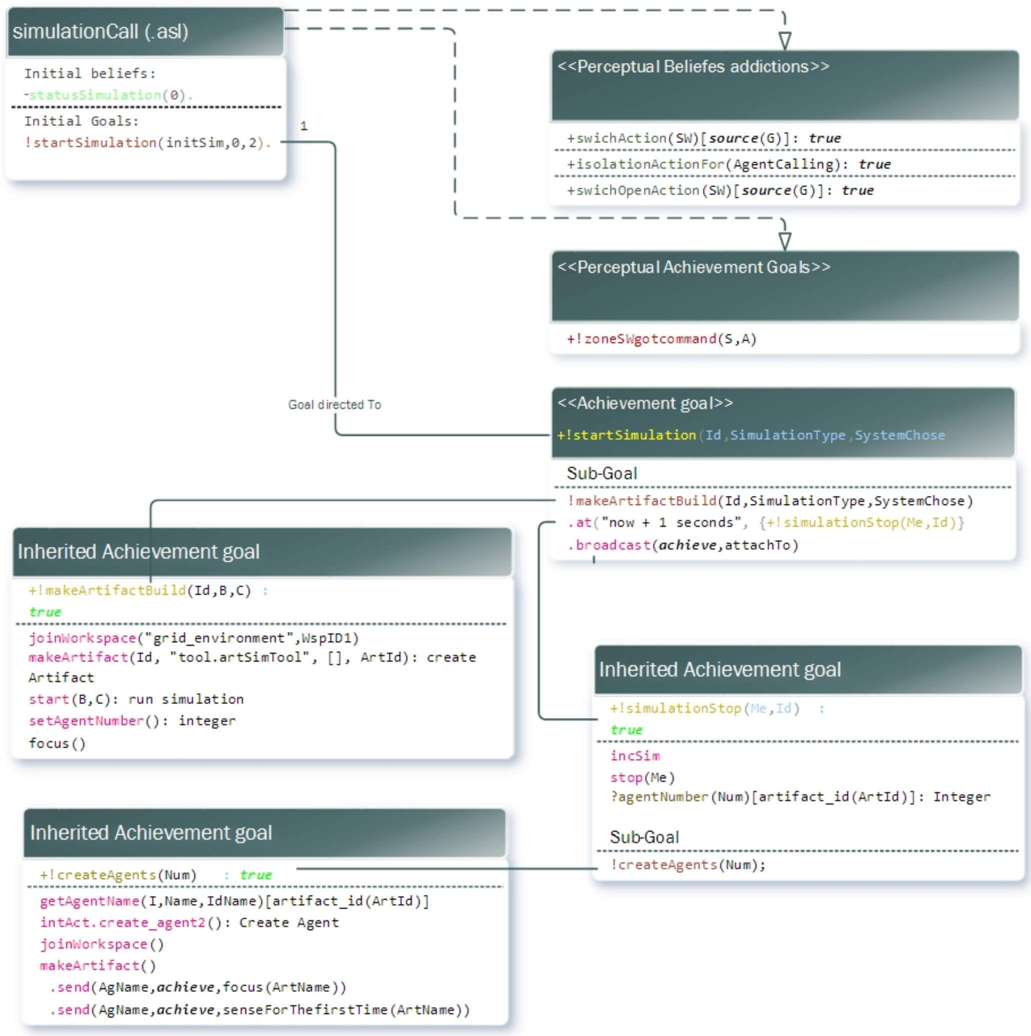

simulationCall (.asl)
Initial beliefs:
-statusSimulation(0).
Initial Goals:
!startSimulation(initSim,0,2).
1
<<Perceptual Beliefes addictions>>
+swichAction(SW)[source(G)]: true
+isolationActionFor(AgentCalling): true
+swichOpenAction(SW)[source(G)]: true
<<Perceptual Achievement Goals>>
+!zoneSWgotcommand(S,A)
Goal directed To
<<Achievement goal>>
+!startSimulation(Id,SimulationType,SystemChose
Sub-Goal
!makeArtifactBuild(Id,SimulationType,SystemChose)
.at("now + 1 seconds", {+!simulationStop(Me,Id)}
.broadcast(achieve,attachTo)
Inherited Achievement goal
+!makeArtifactBuild(Id,B,C) :
true
joinWorkspace("grid_environment",WspID1)
makeArtifact(Id, "tool.artSimTool", [], ArtId): create Artifact
start(B,C): run simulation
setAgentNumber(): integer
focus()
Inherited Achievement goal
+!simulationStop(Me,Id) :
true
incSim
stop(Me)
?agentNumber(Num)[artifact_id(ArtId)]: Integer
Sub-Goal
!createAgents(Num);
Inherited Achievement goal
+!createAgents(Num) : true
getAgentName(I,Name,IdName)[artifact_id(ArtId)]
intAct.create_agent2(): Create Agent
joinWorkspace()
makeArtifact()
.send(AgName,achieve,focus(ArtName))
.send(AgName,achieve,senseForThefirstTime(ArtName))

Figure 3.12: simulationCall structural specification UML diagram

setAgentNumber(_) to retrieve the number of buses in the grid, *getAgentName(_)* a list with the bus names to assign to additional agents and *stop(_)* to stop simulation and start post processing to create new agents to integrate the grid environment.

Predicates that comes with "+!" represents achievement goals and will be triggered by the ones containing only "!", subsequently when the goal is complete the agent return to the previous achievement goal, following the main sequence: simulationCall initialization $\rightarrow$!startSimulation(_) $\rightarrow$ +! +!startSimulation(_) $\rightarrow$!makeArtifactBuild(_) $\rightarrow$ +!makeArtifactBuild(_) $\leftarrow$ +!startSimulation(_) $\rightarrow$ +!simulationStop(_)$\leftarrow$ +!startSimulation(_).

3.2.2 Agent Plans for Automatic Reconfiguration

To design the agent-based model, the fist step taken has been defining the Agents *simulationCall* and *dMS* inside the framework, and a workspace layer to represents the environment space where grid elements and agents will be located. The code characterization of these step if shown in the Algorithm 3.7.

Algorithm 3.7: Defining MAS workspace in JaCaMo

```
mas masReconfig{
    agent simulationCall : simulationDSSSystem.asl
    agent dMS : dms.asl
       workspace grid_environment { agents : simulationCall, dMS    }
       class-path:  lib, src/env, src/int/intAct/create_agent2.java
       asl-path   :  src/agt, src/agt/inc
}
```

A MAS capable of automatic reconfiguration must have basic features to handle grid elements with actuation orders. To perform that, the concept of Artifacts was explored to mimic actuating (Opening and closing switches) and sensing grid element electric quantities such as Voltages, currents and power flow. Furtmermore, as the problem suggests the agent simulationCall must be responsible for providing an AA model for each busAgent that through artifact operations allows access to some of the grid elements operations. Therefore, the MAS distribution grid initiation can be conducted as follows:

1. Initializing the **simulationDSSSystem.asl** file giving *"life"* to simulationCall;
2. A previous devised achievement *goal* is initialized "`!startSimulation (_,_,_).`", and *simulationCall* proceed to perceive the environment objects;
3. The previous goal plan written contains two sub-goals that are subsequently executed:
    - `!makeArtifactBuild(_,_,_)` : responsible for creating the Artifact that holds the OpendDSS API;
    - `!simulationStop(_,_)`: Cease the simulation and starts post possessing;
4. The system must be now ready to receive other agents and once the grid objects are identifiable, another sub-goal is ignited with the responsibility of creating, placing and initializing busAgents within the workspace. This is achieved by the following speech act:
    - .send(AgName,achieve,`focus(ArtName)`) : Order new busAgents to attach to the workspace and bound to Artifact-sensor;
    - .send(AgName,achieve,`senseForThefirstTime(ArtName)`): Order new busAgents to initiate sensing their bus objects, retrieved from the steady state results simulation (voltages and currents);

By interpreting these architecture design guidelines, schema and descriptions, the agent capabilities and plans of actions taken are illustrated in the Fig. 3.13. A complete list of operations used to implement *simulationCall* and *dMS* Agents can be found in Appendix A.

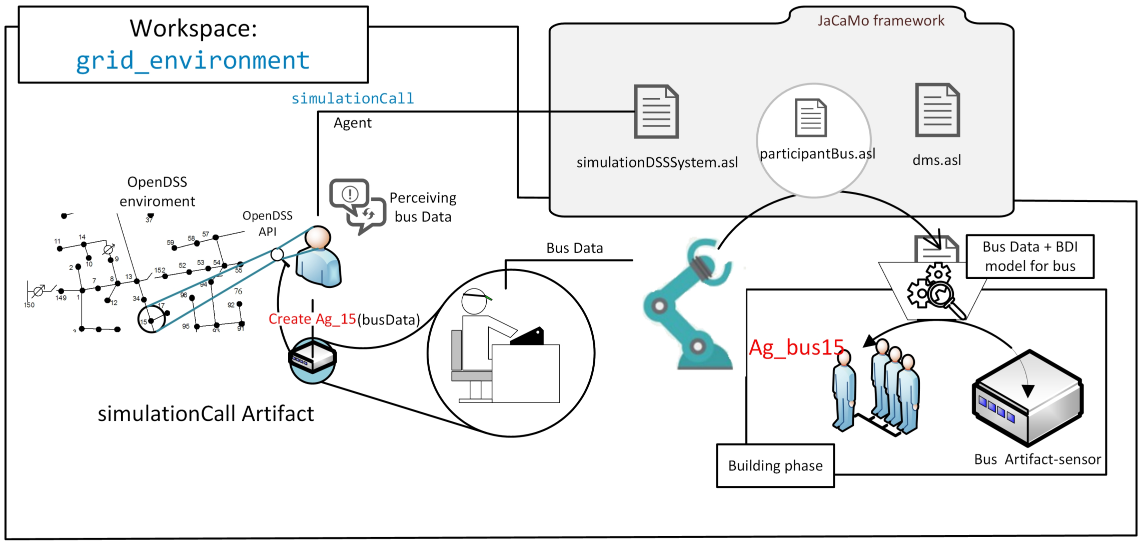


Figure 3.13: Ag_15 process of creation

The afterwards results of the building phase consists of: for each bus (at most), there is a busAgent and artifact-sensor, responsible for sensing the basic proprieties such as voltages, currents and power flows, as pictured in Fig.3.14. The bound with the sensor is established after a communication order from the *simulationCall* to focus on its artifact and start handling data, as explained before in subsection 3.2.2.

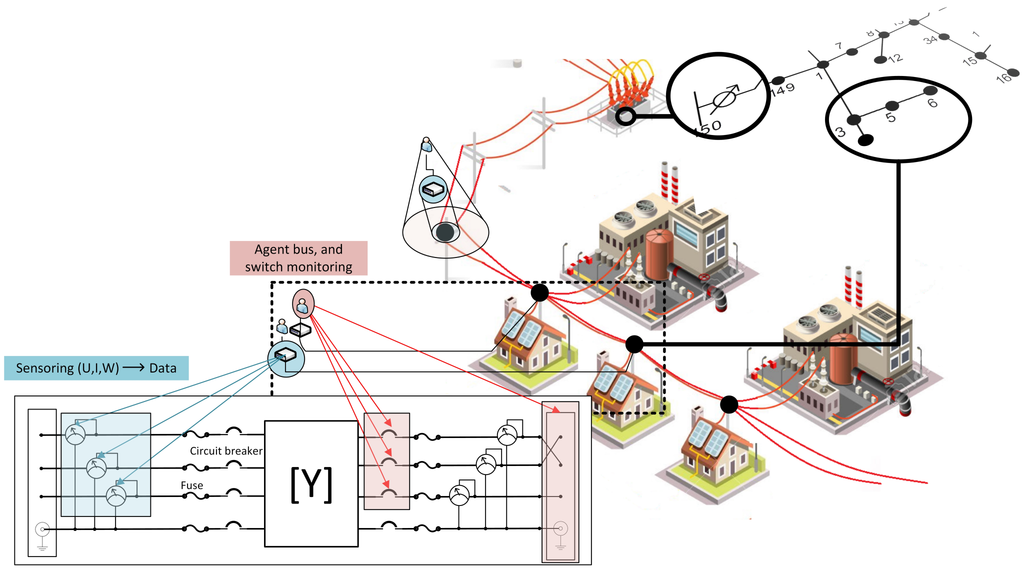


Figure 3.14: busAgents final formation

By definition, the referred agent capabilities are agent oriented “functions” related to agent types and characterized by a set of associated plans. Altogether agents handle decisions towards the grid state respecting the complexity and dynamism of the power distribution systems. Hence, simple but effective rules must represent the main contents of the plans. That is, excess of complexity in plan designing is not considered to be directly related to smartness. Also, excess of complexity might lure engineers related to the field out from implementing such systems.

The capabilities and plans have been designed aiming the balance between critical matters to our purposes and the existing encapsulated schemes from the literature or currently being at evaluation. Therefore, agent plans have been employed respecting the complexity and the certain degree of dynamism imposed by the power distribution simulated environment.

The originated plans in this section are presented as a fragmented lower level abstraction of the set of plans implemented in Jason syntax on the JaCaMo framework. The form in which plans are introduced are inspired by the model of *meta-plan* proposed in [38, chap. 3 pag. 75]

3.2.2.1 Participant Bus Beliefs & Plans

Reinforcing the concept, beliefs in nature can be acquire from initial statement on belief base of external stimulation (*precepts*), *plans* are course of actions that can or can not take place based on Agents decision and they are stated whiting possible Achievement *goals*. That stated, the main set of plans, or the more relevant ones, that define most of the busAgent interactions within the environment and between agents are presented as follows:

The initial beliefs of a busAgents are:

Algorithm 3.8: Initial beliefs of busAgents

```
/** first  term stands for the numerical representation of the Phase (1,2 or 3)
    second term stands for the per unity correspondent voltage value [0.00, 1.99]*/
voltage(0,0.00).
/** first  term is used as pointer to these belief
    second term represents the possible circuit phase (1,2 or 3)
    third term current value
    4th and 5th terms represents atoms used to aid internal operations */
current(1,0,0.000,"null","null").
/** Boolean term used to define bus state 0 - normal state 1-faulted state */
simulationStatus(0).
/** integer term used to define bus zone [0-7] */
zone(None).
```

The perceptual beliefs along the busAgent code, are the ones that will be representing the overall *"mindset"* of these agent-type, refreshing at the end of each reasoning cycle, acquired either by communication or perception of the grid elements within the sphere of influence of the Agent. All represented in Algorithm 3.9:

Algorithm 3.9: perceptual beliefs of busAgents

```
\\acquired by the user or communication
+showConsole : true <-(...)//** used to show bus report of its acts
+sendResponse : true<-(...)//** uses dMS link to COM to show OPnenDSS bus plots
\\acquired by the user to perform fault simulation near related bus
+callFalt : true <- (...) /**decentralized solution- AgentSpeck solution*/
+callFalt2 : true <- (...)/**semi-decentralized solution, over of dMS approval*/
\\acquired by communication, used to receive and send messages amon}g busAgents
+voltage(_,B): B\==0 <- (...) \**rection on environment changes*\
+current(1,0,_,C,_):  C > 0.000 <-(...) \**rection on environment changes*\
+senseAgain(...) : true <- (...) /** command sensors in each cycle */
+performPlan(...)[source(G)]:true <-(...)/**receive and execute plans(dMS)*/
+planForSWagent(...)[source(G)]:true<-(...)/**for busAgents with switch attached*/
+tellSwAgent(...) : true <- (...) /**for busAgents with switch attached*/
+gotActionGoingOnAg(...)[source(G)] : true <- (...)
+gotCommunicationFault(...)[source(G)] : true <- (...)
+gotCommunicationFaultResposta(...)[source(G)] : true <- (...)
+tickFalt(...):true<-(...)/** define speech act for reconfiguration coordination*/
+tickAt(...) : true <- (...) /** define bound between neighboring busAgents*/
```

The following plans represents the main set of Achievement *goals* and perceptual beliefs that define busAgents features such as: how they use a Artifact-sensor on their behalf and how they reason on the data handle by them. The first plan reflects the ideas of integrating an agent to an open environment, in which there will be grid artifacts objects available to focus:

Plan 1: Achievement goal → +!focus(_)

Description: Joins the workspace and establish bound between Artifact and busAgent
Context: The artifact have the operations that allow the busAgent perceive its objects of interest, and using proper Java functions together with CArTaGo Artifact operations is possible to read and write OpenDSS script files, CSV and text files with the simulations results
Functionality: Equip Agents with pseudo sensors and actuator
Trigger: `!focus(Artifact_id)`
Outgoing messages: Confirmation of entering workspace dimension and Artifact-Agent bound and print info action/msg on framework prompt
Actions: `joinWorkspace(_,_), lookupArtifact(_,Artifact_id);`
Used data: Artifact id passed on in the build phase by its creator
Produced data: command prompt printed messages of actions taken
Goal: focus(Artifact_id): internal Jason function

```
begin
    - joinWorkspace("grid_environment",WspID1);
    - lookupArtifact(A,ToolId);
    - ?current_wsp(_,NameEnvi,_);
    - .print(Msg)[artifact_id(ToolId)];
    - focus(ToolId);
```

The plan that follows, Plan 2, give us an introduction to the perceptual *beliefs* used to update each busAgent state at different cycles/simulation that might occur.

Plan 2: Achievement goal → +!perceiveData

Description: Start resoning upon the sensoring cycle

Context: At these point the busAgent already have information data of the bus atachet to him, its now his task to analyze these values and take action

Functionality: Give Agent awerness on what is aceptable or not for a bus to function normally // **Trigger:** `perceiveData`.

Incoming messages: analysys voltage values it self and repond back

Outgoing messages: send voltage values to dMS , Inform message from its own analyses and communication with dMS, print info action/msg on framework prompt

Percepts: `+voltage(1,_,_),+voltage(2,_,_),+voltage(3,_,_) +current(1,_,_,"Phase 1",_,_), +current(1,_,_,"Phase2",_,_), +current(1,_,_,"Phase 3",_,_)`

Actions: `.send("dMS",achieve,simulationStatus(H)).`

Used data: Data acquired by Agent Artifact

Produced data: command prompt printed data aquried by data manupulation

Goal: !statusSystem(H)

```
begin
    begin
        - Trigger changes in belief:
        - +voltage(_,_,_)
        - +current(_,_,_,_,_,_)
    - Establish communication with dMS by reporting Voltage values
```

Furthemore, in plan 3, it is explained how busAgents reason upon the data assigned to them.

Plan 3: Perceptual belief → voltage(_,_,_) & current(_,_,_,_,_,_)

Description: Used to acknowledge busAgents if its Voltage and Current values are within regulatory standard range. Also giving a hint to them of the grid state

Context: In order to change beliefs toward voltage an current obtained values, these agent type, use a mechanism to test belief under voltage and current labels

Functionality: Used to change belief on Voltage and Current Values and subject them to test

Trigger: `+voltage(_,_,_) & +current(_,_,_,_,_,_)`

Actions: `!testBelifadequacyVoltage(_,_,_), !testBelifadequacyCur(...)`

Used data: Voltages and Currents data

Produced data: Log of the attempts

```
begin
    +voltage(A,B,ToolId): B/==0 <- !testBelifadequacyVoltage(A,B,ToolId). +current(A,B,C,F,H,ToolId): C
    > 0.000 <- !testBelifadequacyCur(A,B,C,F,H,ToolId).
```

Finally, in Plan 4 and 5, it is provided agent means to know believes corresponding to measurements as current and voltage values.

Plan 4: Achievement goal → +!testBelifadequacyVoltage(_,_,_)

Description: Decision loop used to test busAgents Voltage values
Context: Inherited plan solution from voltage belief changing state
Functionality: Chose a plan depending on Voltage values
Trigger: `!testBelifadequacyVoltage(_,_,_)`
Outgoing messages: Report on choosing plan
Percepts: `+showConsole`
Actions: `Report state`
Used data: Voltages data
Produced data: Log of the attempts

```
begin
    -Decision loop:
    +!testBelifadequacyVoltage(_,B,_): true <- !testBelifadequacyVoltage(_,B,_).
    +!testBelifadequacyVoltage(_,B,_): B<0.95 | B>1.05 <- .print(ReportOnFalt); +showConsole.
    +!testBelifadequacyVoltage(_,B,_): B>0.95 & B<1.05 <-.print(ReportOnAdequacy).
```

Plan 5: Achievement goal → +!testBelifadequacyCur(_,_,_,_,_,_)

Description: Decision loop used to test busAgents Voltage values
Context: Inherited plan solution from current belief changing state
Functionality: Chose a plan depending on Voltage values
Trigger: `!testBelifadequacyCur(_,_,_,_,_,_)`
Outgoing messages: Report on choosing plan
Percepts: `+showConsole`
Actions: `Report state`
Used data: Voltages data
Produced data: Log of the attempts

```
begin
    -Decision loop:
    +!testBelifadequacyCur(_,_,C,_,_,_): true <- !testBelifadequacyCur(_,_,_,_,_,_).
    +!testBelifadequacyCur(_,_,C,_,_,_):(C*1/6)<100.0 & (C*1/4) > 100.0 <- ...Report&act
    +!testBelifadequacyCur(_,_,C,_,_,_):(C*1/4) < 100 & (C*1/6) < 100 & C > 10 <- ...Report&act
```

3.2.2.2 Reconfiguration Planning: Hierarchical Solution

In case of fault conditions, the agents start a coordination cycle to achieve reconfiguration by exploring a hierarchical approach using a *dMS* to delegate toplans to be followed by busAgents in order to achieve reconfiguration. The protocols, procedures and and strucuted interactions for this case are examplified for a short-circuit near bus 60 of the network shown in Fig. 3.2:

In the figure 3.15, it is shown the moment the nearest busAgent acknowledges the occurrence of a fault and starts action upon it. Using directional fault passage indicators, it is possible to identify the zone where the fault occurred. Basically, if the fault current has entered the zone, but not get out of the zone, the busAgents can identify that the fault occurred inside the zone. Otherwize, if the circuit is de-energized and the current got into and out the zone, then a fault occured in other zone. With this reasonings, messages are conveyed to the agents attached to switches to negotiating to isolate the zone. At the same time the dMS also receives a de-energization messages and starts reasoning on behalf of the other agents that reports the state back to dMS as well. Once the

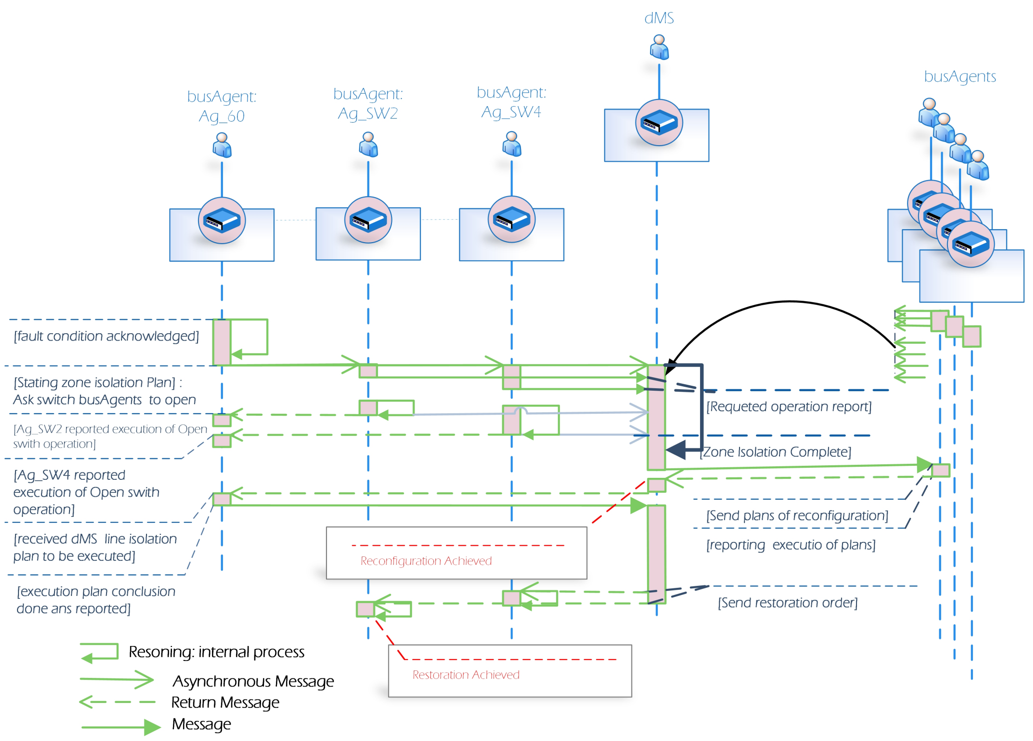

dMS
busAgent: Ag_60
busAgent: Ag_SW2
busAgent: Ag_SW4
busAgents
[fault condition acknowledged]
[Stating zone isolation Plan] : Ask switch busAgents to open
[Ag_SW2 reported execution of Open swith operation]
[Ag_SW4 reported execution of Open swith operation]
[received dMS line isolation plan to be executed]
[execution plan conclusion done ans reported]
[Requeted operation report]
[Zone Isolation Complete]
[Send plans of reconfiguration]
[reporting executio of plans]
[Send restoration order]
Reconfiguration Achieved
Restoration Achieved
Resoning: internal process
Asynchronous Message
Return Message
Message

Figure 3.15: busAgents fault coordination

dMS receive the isolation zone confirmation it starts a subroutine to find ways to best reconfigure the grid. When the plans for configuration are set the dMS delegates functions to agents to reconfigure. When the process is done it is expected that the reconfiguration is achieved and plans for calling crews are initialized to take care of the failed component.

The dMS agent runs a Dijkstra Shortest Path algorithm to find the best solution to achieve reconfiguration aiming in reducing the number of non energized zones and buses. The criterion of number of buses was chosen for the sake of the example, though other criterion could be specified. Afterwards the dMS send plans to be followed by the busAgents.

Dijkstra Shortest Path is an algorithm for finding the shortest paths between nodes in a graph and exists in many variants. The one used for the course of reconfiguration fixes a single node as the *"source"* node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree. The idea behind Dijkstra algorithm can be depicted assuming the needed path is from agent A to B:

- Start with a message consisting of the name of an agent node A.
- Send a copy of this message along each line spreading from A.
- When a message reaches a node agent K, its name is add to the path.
- Along each line which spreads from node K and was not used yet, a copy of the message is sent.

- The first message that arrives at node B (and any other node) contains the shortest path from A to it.

The use of the algorithm here comes with the idea of finding the best path option from the source bus to normally open switches in order to minimizing the impact of an possible permanent fault. This can take into account several criteria, so for the sake of choosing one we aim to maximize the number of nodes to be energized. By making use of the data obtained by the agents, such as coordinates and length, and to implement an method that use these data, a Java - based dynamic graphic library *GraphStream* [72] was chosen.

The basic usage of this class takes place in 4 steps [72]:

1. Definition of a Dijkstra instance with parameters needed for the initialization.
2. Initialization of the algorithm with a graph through the initiation(graph) method from the Algorithm interface.
3. Computation of the shortest path tree with the compute(_) method from the Algorithm interface.
4. Retrieving of shortest paths for given destinations with the getShortestPath(Node) method for instance.

The creation of the Dijkstra instance is done with the dijkstra(Element, String, String) constructor by giving 3 parameters: First, the type of element that is consider for the computing of shortest paths (Dijkstra.Element.node). Second, the key string of the attribute used for the weight computation. The third parameter is the "id" of the source node the shortest tree will be constructed for. The source of all these information along with the example of how to use the algorithm in the library can be found in [32, 72] and the devised algorithm for these approach can be found in [44].

The application of the algorithm is defined by an simple search for solution, consisting in seating a fix node as *source*. For instance, in the IEEE123 test feeder the source node is bus number 150. The basic solution is the find the shortest path from the source to all switches in the grid. A post processing method was devised to read the list created with all shortest path assuming all switches are in close state, by their names witch contains normally open (NO) or normally (NC), the shortest path to a normal open switch define the plan form a closing to be closed by the agent attached to it, as depicted in Fig. 3.16.

3.2.2.3 Reconfiguration Planning: Decentralized Solution

Although the hierarchical solution might present fair results, we also deployed the idea of a fully decentralized solution where busAgents are responsible to the main reconfiguration procedures. This reinforces the idea of a smart system that can handle adversities without a quasi-dictator *dMS* agent. To achieve that the implemented functionalities were:

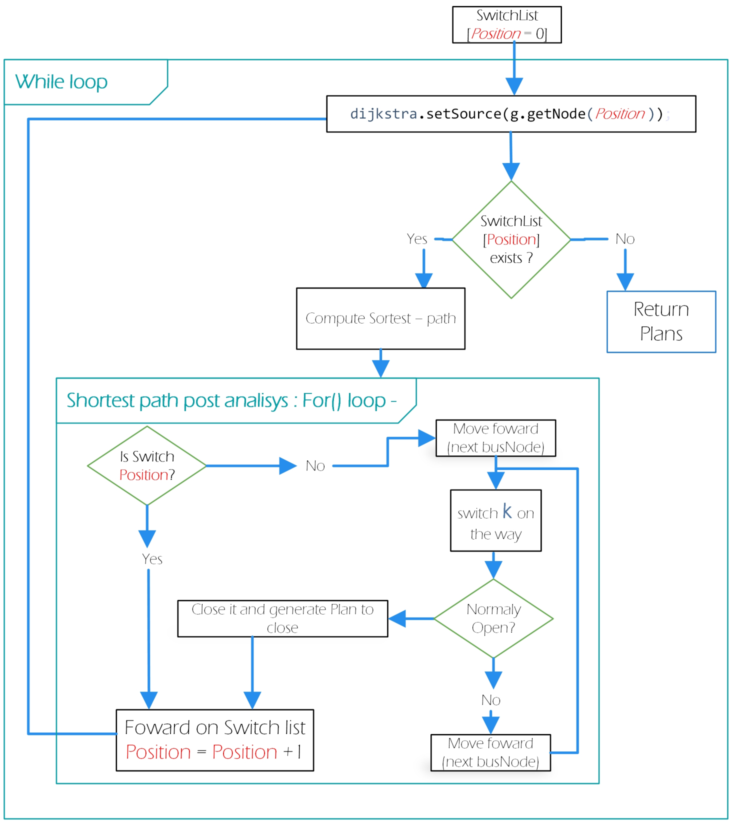

SwitchList
[Position = 0]
While loop
dijkstra.setSource(g.getNode(Position));
SwitchList [Position] exists ?
Yes
No
Compute Sortest – path
Return Plans
Shortest path post analisys : For() loop -
Is Switch Position?
No
Yes
Move foward (next busNode)
switch k on the way
Normaly Open?
Close it and generate Plan to close
No
Foward on Switch list Position = Position + 1
Move foward (next busNode)

Figure 3.16: dMS reasoning plan generator

*Step*1 Fault identification: Similarly to the hierarchical approach, the nearest bus react to fault current indications initiating speech act actions in order to isolate the faulted zone. The de-nergized zones are also isolated to aid reconfiguration.

*Step*2 The reconfiguration system: One by one, agents responsible for switches will are asked by neighboring busAgents to close if they are neighbors of an energized zone, excluding the faulted zone. This enables the full an decentralized solution.

*Step*3 The restoration: After a signal indicates the faulted component isrepaired, another procedure is made in order to restore the system to its initial state.

The basic idea behind these approach is to find a simple but suitable solution for reconfiguration. As shown in the Fig. 3.17, the starting point of the simulation represents the moment when the nearest busAgent to fault acquire a belief of a fault state, and right after proceeds to communicate to other busAgent attached to switches, asking them to change status from normally close to open. Afterwards, course of actions involve the other agents, that is, all busAgents will participate of an speech acts in order to close switches attached to energized zones, one by one until a reconfiguration is done and most of the custumers are supplied. The steps of the AgentSpeak implementation are found in algorithm 3.10.

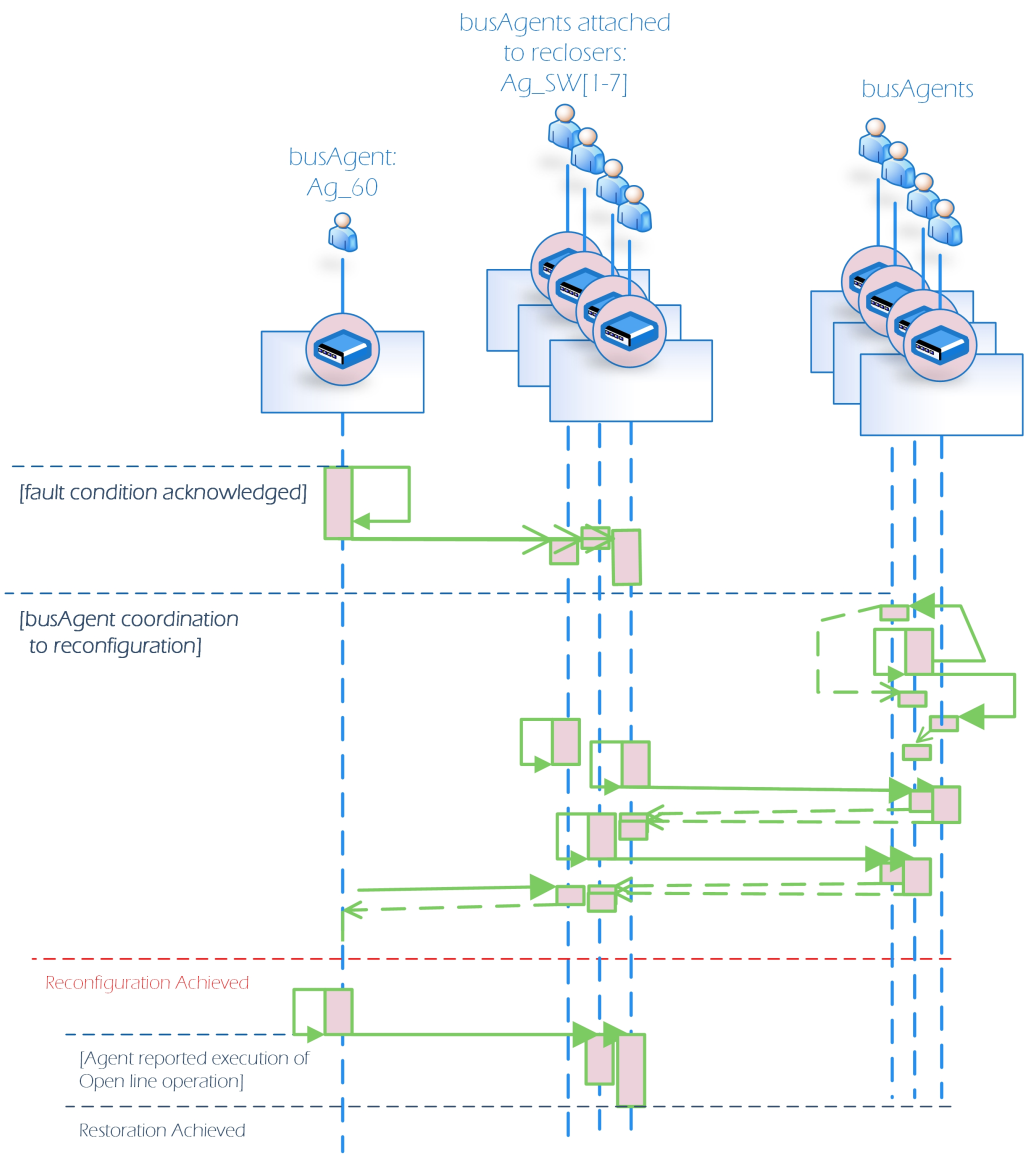


Figure 3.17: busAgents fault coordination

Algorithm 3.10: AgentSpeak for decentralized reconfiguration

```

/**(Step 1: From busAgent)**/  +callFalt2: true <- \\ perceptual belief triggered
  getBusRealName(RealName)[artifact_id(S)]; getFaltLine(FaltLine)[artifact_id(S)];
  ?zone(BusZone)[artifact_id(S)];
  .print("**Falt simulation with in line",FaltLine," bus number: -->",RealName)
  .send("dMS",tell,fautSimulation(RealName,FaltLine,BusZone));. \\ belief trigger
/**(Step 2: From simulation Agent)**/
+fautSimulation(BusFalt,LineOfFalt,BusZone)[source(G)]; \\ perceptual belief
   \\run OpenDSS short-circuit simulation given the Bus and line of fault
    .wait(5000);  performFalt(BusFalt,LineOfFalt)[artifact_id(ArtId)];
    .print("Falt occurency, reported location of falt by ---> Agent",G);
    +statusSimulation(0,1); \\ changing belief --> Normal to Fault state
    .wait(3000); .print("****       OPENING ALL SWITCHES             ****");
   openALLsw[artifact_id(ArtId)];       ...
/**(Step 3:)**/                                 ...
  .broadcast(tell,senseAgain(_)); \\ tell all bus Agent to acknowledge their state
  .print("****Atempting Falt restoration by Agents speach Act coordination****");
  .wait(5000);    .broadcast(achieve,faltCommunication);
                      ...
/**(Step 4:)**/
.print("Reconfiguration Performed Successfully... Awaiting Restoration Agreement");
                      ...
 +procedReconfig_and_restoration(G) \\ trigger belief
 +getPlots("1");
 .send("Ag_1",tell,showConsole);.
```

Both solutions present fundamentally the same results and uses the same basis of Agent communication though out speech acts. What essentially differentiates one from another is that in the first solution the MAS is somewhat centralized, while the alternative solution presents a fully decentralized approach.

The basic explanation on how agents manage to receive massages proceed the reasoning at ate the same time that its sending messages, is due a the multi-threaded asynchronous nature adopted in JaCaMo approach. One way to characterize that is comparing with other processing methods, as below.

```
    ----    : Time spend
 A,B and C : Processes (receive, send, others...)
Synchronous (one thread):
thread ->   |----A-----||-----B-----------||-------C------|
Synchronous (multi-threaded):
thread A -> |----A-----|
                        \
thread B ------------>   ->|-----B-----------|
                                              \
thread C ----------------------------------->   ->|-------C------|

Asynchronous (one thread):
         A-Start ---------------------------------------- A-End
           | B-Start ----------------------------------------|--- B-End
           |   |      C-Start -------------------- C-End      |      |
           V   V       V                            V          V      V
1 thread-> |-A-|---B---|-C-|-A-|-C-|--A--|-B-|--C--|---A-----|--B--|

Asynchronous (multi-Threaded):
 thread Agent A ---------> |----A-----|
 thread Agent B ---->     |-----B-----------|
 thread Agent C ------>     |-------C----------|
```

The next chapter will then expose the overall results obtained from both solutions.

3.3 Final Remarks

Considerations toward the software and tools utilized to implement a MAS approach for reconfiguration in the IEEE 123 test feeder are presented. Also, it was presented the model adopted to conceive power grids on OpenDSS context as well as example of representations of line, switches, busses and terminal were addressed. This was followed by the description of the approach to to integrate the grid model and simulation with an MAS framework supported by the use of an API package.

Concerning the MAS model, the JaCaMo approach for agent-based systems has been unfold, addressing the main topics of its representations, the BDI structure and the model of AgentSpeck adopted, all expressed in basic Jason modeling and the Agent-Artifact model supported by CArTaGo. When comes to the application capabilities, it was developed schemes to devise a MAS capable to simulate a power grid environment along with agent planing and methodologies to achieve reconfiguration within the simulation platform. All this considered solutions for reconfiguration either by a semi decentralize model using a dijkstra shortest path algorithm and a fully decentralized model.

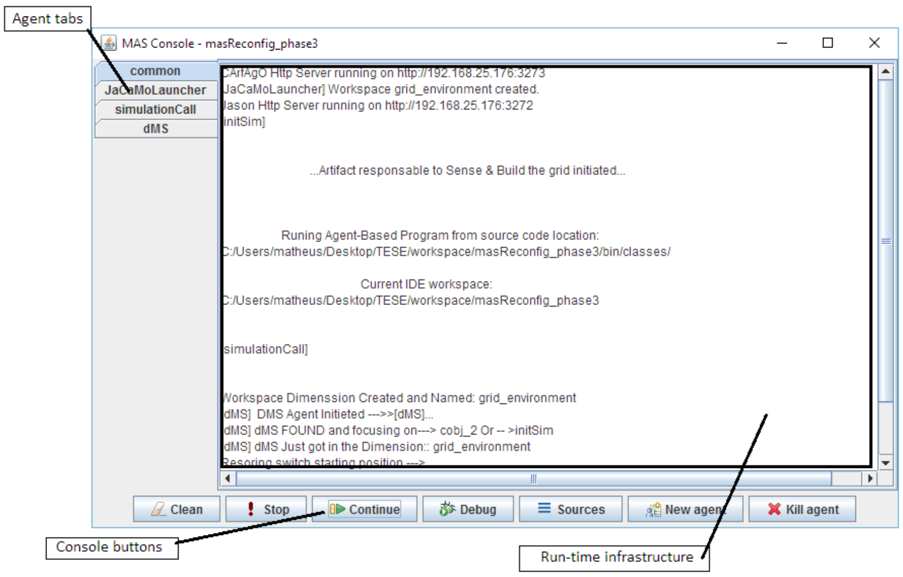
Chapter 4

Simulation and Results Analysis

This chapter presents simulations and result analysis for the MAS approach to model AR, either considering a hierarchical or fully decentralized control philosophy. The chapter is structured as follows. In section 4.1, system initialization and building phase are presented for the purposes of testing and validating environment and grid simulations. In section 4.2, simulation cycles are presented in order to analyse reconfiguration solutions achieved using both control philosophies.

4.1 MAS Initialization and building phase

The *JaCaMo* framework console is a buit-in optional Graphic User Interface (GUI) used to present the run-time simulation of the system also providing basic tools to aid the user in achieving a compressible view over devised MAS solutions. JaCaMo's console is illustrated in Fig. 4.1.



```
CArtAgO Http Server running on http://192.168.25.176:3273
JaCaMoLauncher] Workspace grid_environment created.
Jason Http Server running on http://192.168.25.176:3272
initSim]


            ...Artifact responsable to Sense & Build the grid initiated...



         Runing Agent-Based Program from source code location:
C:/Users/matheus/Desktop/TESE/workspace/masReconfig_phase3/bin/classes/

                         Current IDE workspace:
C:/Users/matheus/Desktop/TESE/workspace/masReconfig_phase3


simulationCall]


Workspace Dimenssion Created and Named: grid_environment
dMS]  DMS Agent Initieted --->>[dMS]...
dMS] dMS FOUND and focusing on---> cobj_2 Or -- >initSim
dMS] dMS Just got in the Dimension:: grid_environment
Resoring switch starting position --->
```

Figure 4.1: JaCaMo framework console

.

In JaCaMo framework console, Agent-Agent and Agent-Artifact interactions are can be depicted in particular window tab dedicated to a given agent, or in a common window where all interactions are marked by console messages. The example below transcripts messages shown in the window console:

```
begin
    ... [simulationCall] dMS FOUND and focusing on--> cobj_2 Artifact
    [dMS]DMS Agent Initiated
    ...
```

The course of actions taken by the agents can be observable through out the run-time infrastructure as simulation goes. Also, log messages of each participant on the environment will be accessible by a tab with the *Ag_Nº* tag, giving the user access to each action made either by the Agent itself or its attached Artifact. In addition, some of the console buttons give the user the power to direct interaction within the environment and the agents as well as the possibility of adding new agents and triggering beliefs and goals externally.

In agent simulatiom, visualization is a matter of utmost importance. JaCaMo console allowed validating the integration of OpenDSS with MAS platform, besides supporting modeling and improving the applications. With direct link to OpenDSS, all its structures, including validated IEEE data for test systems, can be used as object types. The customized console permited also verifying and correcting problems during the building, where workspace and agent types are instantiated.

The building phase consists on running the JaCaMo file (.jcm), shown in algorithm 3.7, containing the MAS initiation course. This works also as a form of validating each step taken in the simulation, printing messages and log commands placed intentionally to aid verifying the evolution of simulation process and execution of internal/external functions.

Once the run button is pressed, the artifact created by *simulationCall*, named *initSim*, run the API and start processing:

```
[initSim] ..Artifact responsible to Sense & Build the grid initiated..
```

Afterwards, the simulationCall Agent acknowledges its artifact ant takes he first actions:

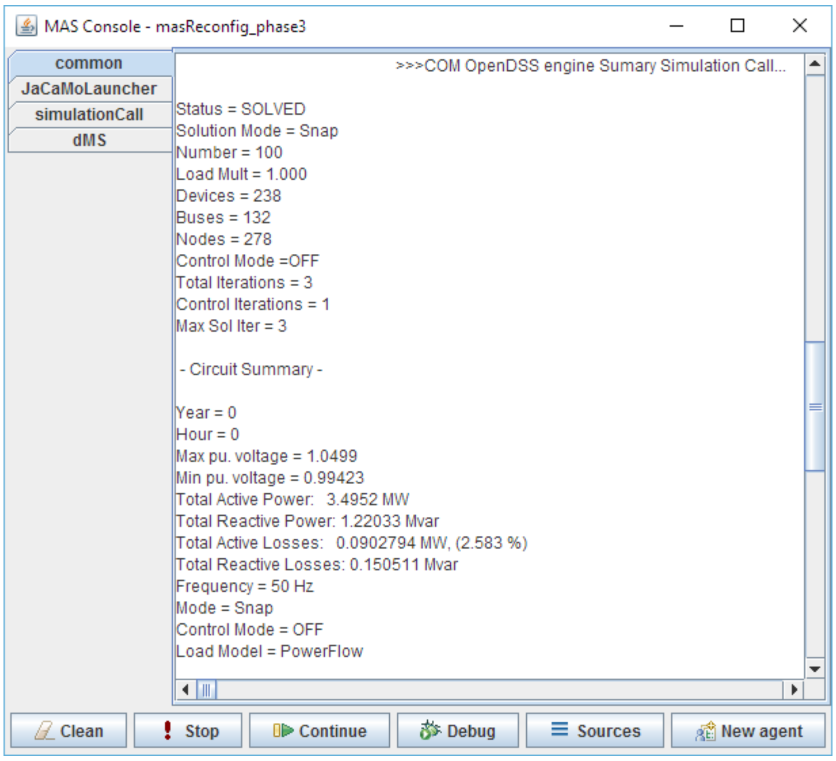
1. Identify the workspace:

```
begin
    [simulationCall] Workspace Dimension Created and Named: grid_environment
```

2. Trigger goals to perform stead-state simulation:

```
begin
    [initSim] Runing Case: IEEE123 Distribution system
    [initSim] »COM OpenDSS engine::: Running Simulation
```

3. Log the simulation summary, as shown in Fig. 4.2.**begin**



```
>>>COM OpenDSS engine Sumary Simulation Call...

Status = SOLVED
Solution Mode = Snap
Number = 100
Load Mult = 1.000
Devices = 238
Buses = 132
Nodes = 278
Control Mode =OFF
Total Iterations = 3
Control Iterations = 1
Max Sol Iter = 3

 - Circuit Summary -

Year = 0
Hour = 0
Max pu. voltage = 1.0499
Min pu. voltage = 0.99423
Total Active Power:   3.4952 MW
Total Reactive Power: 1.22033 Mvar
Total Active Losses:   0.0902794 MW, (2.583 %)
Total Reactive Losses: 0.150511 Mvar
Frequency = 50 Hz
Mode = Snap
Control Mode = OFF
Load Model = PowerFlow
```

Figure 4.2: Summary of steady state simulation performed inside the framework

4. Start processing actions to filter and organize returned COM objects and external files: **begin**

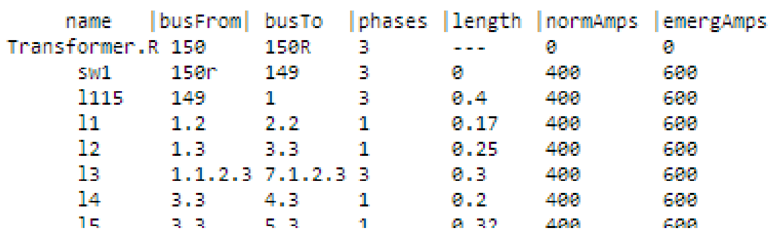
$[initSim]$ »>COM OpenDSS engine::: Retrieving Result Objects...

$[initSim]$ Lines and switches : $[l115, l1, l2, l3, l4, l5, l6, ..., sw1, sw2, sw3, sw4, sw5, sw6, sw7, sw8]$

$[initSim]$ Busses: $[150, 150r, 149, 1, 2, 3, 7, 4, 5, 6, 8, 12, 9, 13, 9r, 14, 34, 18, 11, 10, ...]$

$[initSim]$ Tranformer and regulators: $[Transformer.reg1a, RegControl.creg1a...]$

$[initSim]$»>Post Processing Results ...



```
      name    |busFrom| busTo  |phases |length |normAmps |emergAmps
Transformer.R 150     150R     3       ---     0         0
         sw1  150r    149      3       0       400       600
        l115  149     1        3       0.4     400       600
          l1  1.2     2.2      1       0.17    400       600
          l2  1.3     3.3      1       0.25    400       600
          l3  1.1.2.3 7.1.2.3  3       0.3     400       600
          l4  3.3     4.3      1       0.2     400       600
          l5  3.3     5.3      1       0.32    400       600
```

...

Coordinates , Bus: 88 X : 10813.263337 Y : 2171.336913

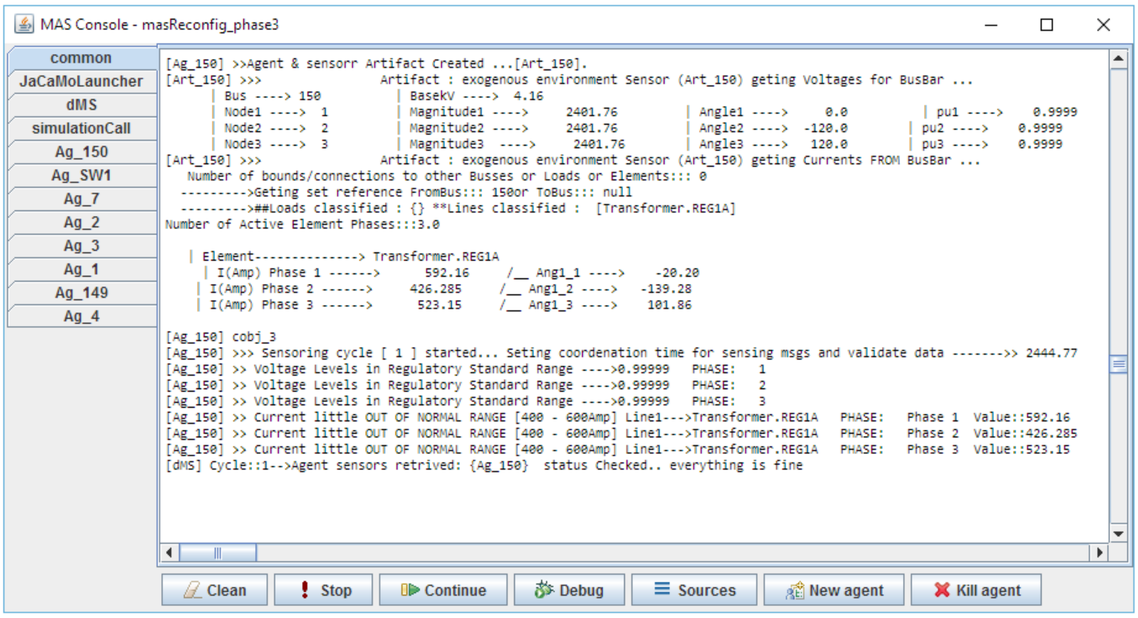
Coordinates , Bus: 89 X : 9833.451402 Y : 279.28628

Coordinates , Bus: 197 X : 11067.378502 Y : 6097.64203

...

5. After the artifact corresponding to *simulationCall* agent returned all required simulation data, bus agents are created. The steady-state simulation allows sensoring artifacts to be instantiated and associated to their interrelated bus agents, as depecited in Fig. 4.3.

```
begin
    [simulationCall]»simulation ended successfully :)
    [simulationCall]>Agent action terinated 00 hours, 00 mins, 52 seconds
    [simulationCall] Exogenous Environment set for the Power Distribution Study System.....
```



```
[Ag_150] >>Agent & sensorr Artifact Created ...[Art_150].
[Art_150] >>>              Artifact : exogenous environment Sensor (Art_150) geting Voltages for BusBar ...
      | Bus ----> 150          | BasekV ---->  4.16
      | Node1 ---->  1         | Magnitude1 ---->     2401.76         | Angle1 ---->     0.0          | pu1 ---->    0.9999
      | Node2 ---->  2         | Magnitude2 ---->     2401.76         | Angle2 ---->  -120.0         | pu2 ---->    0.9999
      | Node3 ---->  3         | Magnitude3  ---->     2401.76        | Angle3 ---->   120.0         | pu3 ---->    0.9999
[Art_150] >>>              Artifact : exogenous environment Sensor (Art_150) geting Currents FROM BusBar ...
   Number of bounds/connections to other Busses or Loads or Elements::: 0
  --------->Geting set reference FromBus::: 150or ToBus::: null
  --------->##Loads classified : {} **Lines classified :  [Transformer.REG1A]
Number of Active Element Phases:::3.0

   | Element--------------> Transformer.REG1A
     | I(Amp) Phase 1 ------>      592.16     /__ Ang1_1 ---->    -20.20
    | I(Amp) Phase 2 ------>     426.285     /__ Ang1_2 ---->   -139.28
    | I(Amp) Phase 3 ------>      523.15     /__ Ang1_3 ---->    101.86

[Ag_150] cobj_3
[Ag_150] >>> Sensoring cycle [ 1 ] started... Seting coordenation time for sensing msgs and validate data ------>> 2444.77
[Ag_150] >> Voltage Levels in Regulatory Standard Range ---->0.99999   PHASE:   1
[Ag_150] >> Voltage Levels in Regulatory Standard Range ---->0.99999   PHASE:   2
[Ag_150] >> Voltage Levels in Regulatory Standard Range ---->0.99999   PHASE:   3
[Ag_150] >> Current little OUT OF NORMAL RANGE [400 - 600Amp] Line1--->Transformer.REG1A   PHASE:   Phase 1  Value::592.16
[Ag_150] >> Current little OUT OF NORMAL RANGE [400 - 600Amp] Line1--->Transformer.REG1A   PHASE:   Phase 2  Value::426.285
[Ag_150] >> Current little OUT OF NORMAL RANGE [400 - 600Amp] Line1--->Transformer.REG1A   PHASE:   Phase 3  Value::523.15
[dMS] Cycle::1-->Agent sensors retrived: {Ag_150}  status Checked.. everything is fine
```

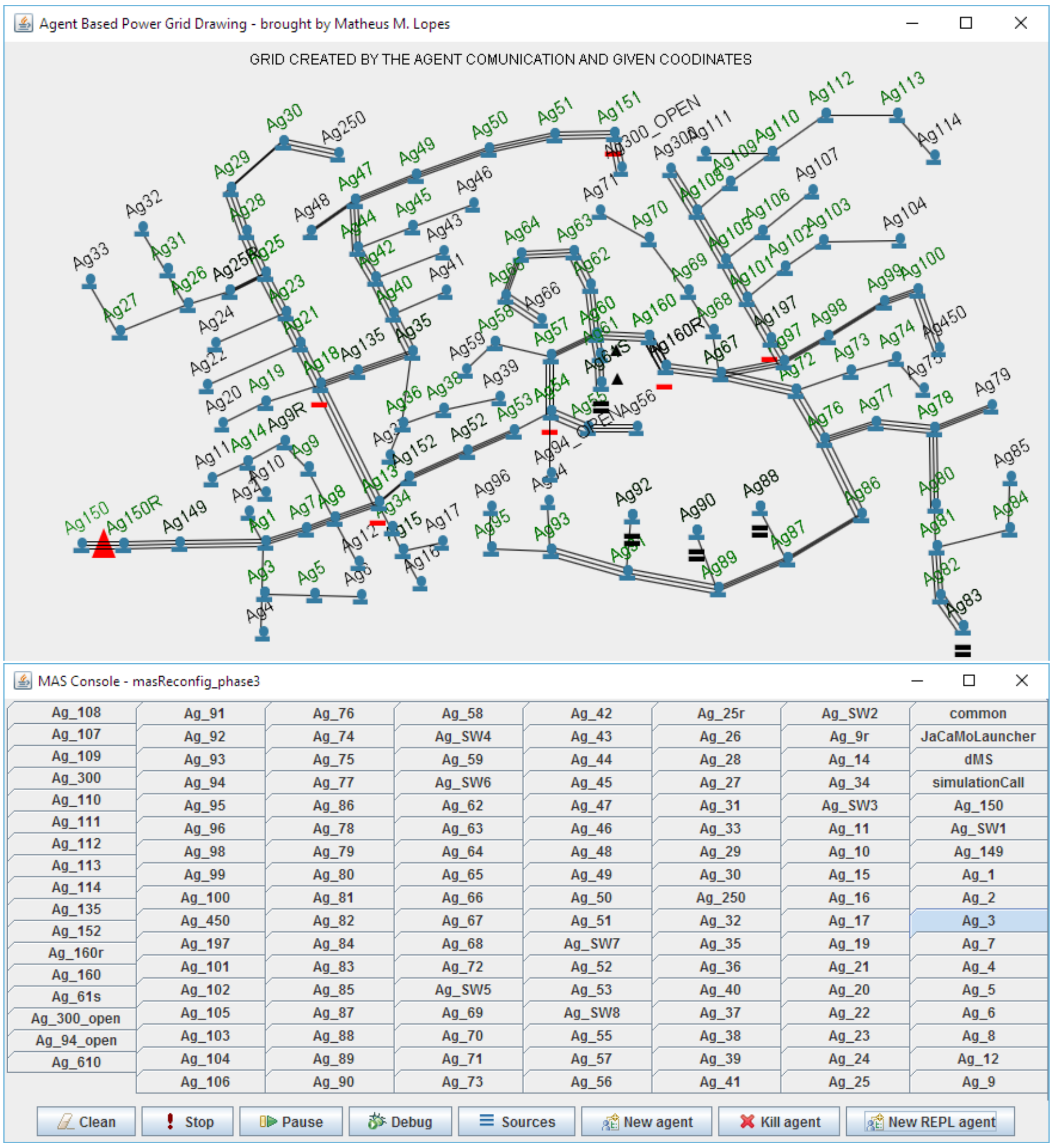
Figure 4.3: Sensing summary results for a busAgent

6. After agent and artifact initiatlization, busAgents start interactions and communications in order to try identifying its neighbours. This interactions are logged as shown below.

```
begin
    [Ag_1] I am attachedTo [Ag_3]
    [Ag_1] I am attachedTo [Ag_7]
    [Ag_1] I am attachedTo [Ag_2]
    [Ag_1] I am attachedTo [Ag_149]
    [Ag_1] Message received From Agent {Ag_7} :
    Reporting Executed Action/Or something —> Confirmed Attachment by tick ...
    [Ag_1] Message received From Agent {Ag_3}:
    Reporting Executed Action/Or something —> Confirmed Attachment by tick ...
    [Ag_1] Message received From Agent {Ag_149}
    Reporting Executed Action/Or something —> Confirmed Attachment by tick ...
    [Ag_1] Message received From Agent {Ag_2}
    Reporting Executed Action/Or something —> Confirmed Attachment by tick ...
```

Concurrently, similar processes are devied by the *dMS* that is initialized alongside its corresponding artifact, and is included in the environment workspace.

As form of evaluation and registering, once the integration process is finished and all the busAgents are placed in the environment, the dMS starts a subroutine that use the bus attachment communication results to draw the grid. Fig. 4.4 exposes the resultant drawing which includes a tab list of active Agents in the environment.



Ag_108	Ag_91	Ag_76	Ag_58	Ag_42	Ag_25r	Ag_SW2	common
Ag_107	Ag_92	Ag_74	Ag_SW4	Ag_43	Ag_26	Ag_9r	JaCaMoLauncher
Ag_109	Ag_93	Ag_75	Ag_59	Ag_44	Ag_28	Ag_14	dMS
Ag_300	Ag_94	Ag_77	Ag_SW6	Ag_45	Ag_27	Ag_34	simulationCall
Ag_110	Ag_95	Ag_86	Ag_62	Ag_47	Ag_31	Ag_SW3	Ag_150
Ag_111	Ag_96	Ag_78	Ag_63	Ag_46	Ag_33	Ag_11	Ag_SW1
Ag_112	Ag_98	Ag_79	Ag_64	Ag_48	Ag_29	Ag_10	Ag_149
Ag_113	Ag_99	Ag_80	Ag_65	Ag_49	Ag_30	Ag_15	Ag_1
Ag_114	Ag_100	Ag_81	Ag_66	Ag_50	Ag_250	Ag_16	Ag_2
Ag_135	Ag_450	Ag_82	Ag_67	Ag_51	Ag_32	Ag_17	Ag_3
Ag_152	Ag_197	Ag_84	Ag_68	Ag_SW7	Ag_35	Ag_19	Ag_7
Ag_160r	Ag_101	Ag_83	Ag_72	Ag_52	Ag_36	Ag_21	Ag_4
Ag_160	Ag_102	Ag_85	Ag_SW5	Ag_53	Ag_40	Ag_20	Ag_5
Ag_61s	Ag_105	Ag_87	Ag_69	Ag_SW8	Ag_37	Ag_22	Ag_6
Ag_300_open	Ag_103	Ag_88	Ag_70	Ag_55	Ag_38	Ag_23	Ag_8
Ag_94_open	Ag_104	Ag_89	Ag_71	Ag_57	Ag_39	Ag_24	Ag_12
Ag_610	Ag_106	Ag_90	Ag_73	Ag_56	Ag_41	Ag_25	Ag_9

Figure 4.4: Power grid created by Agent's coordinated communication an active Agents

Since actions inside the platform are linked to OpenDSS, then it is possible to plot steady-state and fault analysis results inside the MAS platform, including voltages, Currents, Power flows and losses. As examples, visualizations are shown in the figures ahead, including power losses through components (Fig. 4.5), power flow through components (Fig. 4.6) and steady-state voltages (Fig.

4.7, Fig. 4.8).

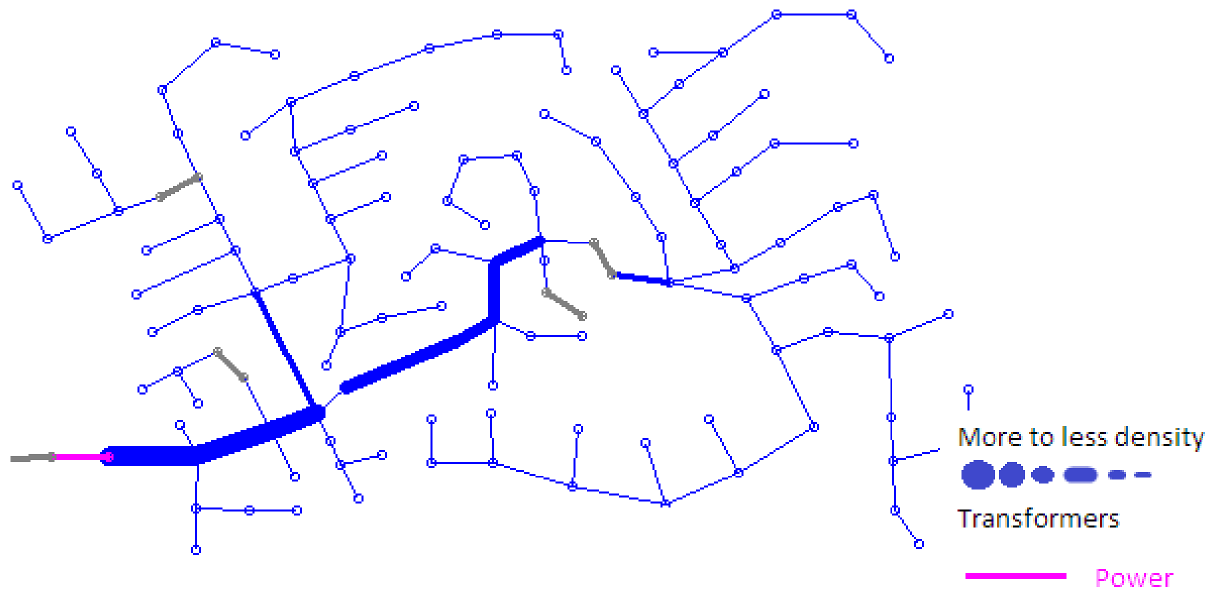


Figure 4.5: Steady state Loss density

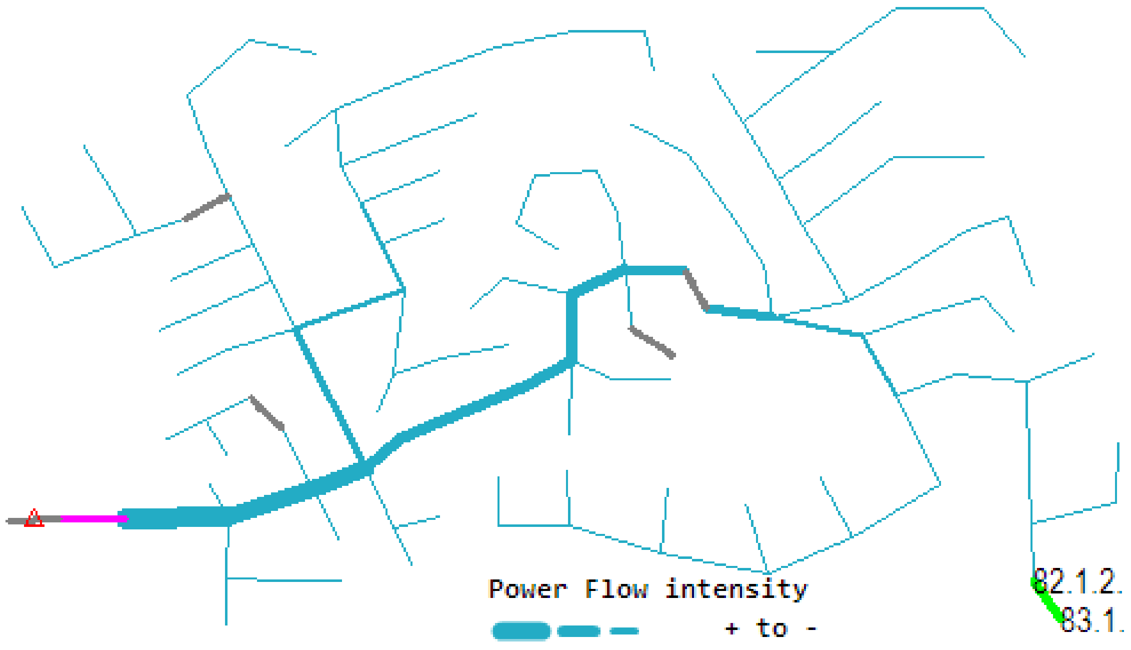


Figure 4.6: Steady state bus Voltages per phases

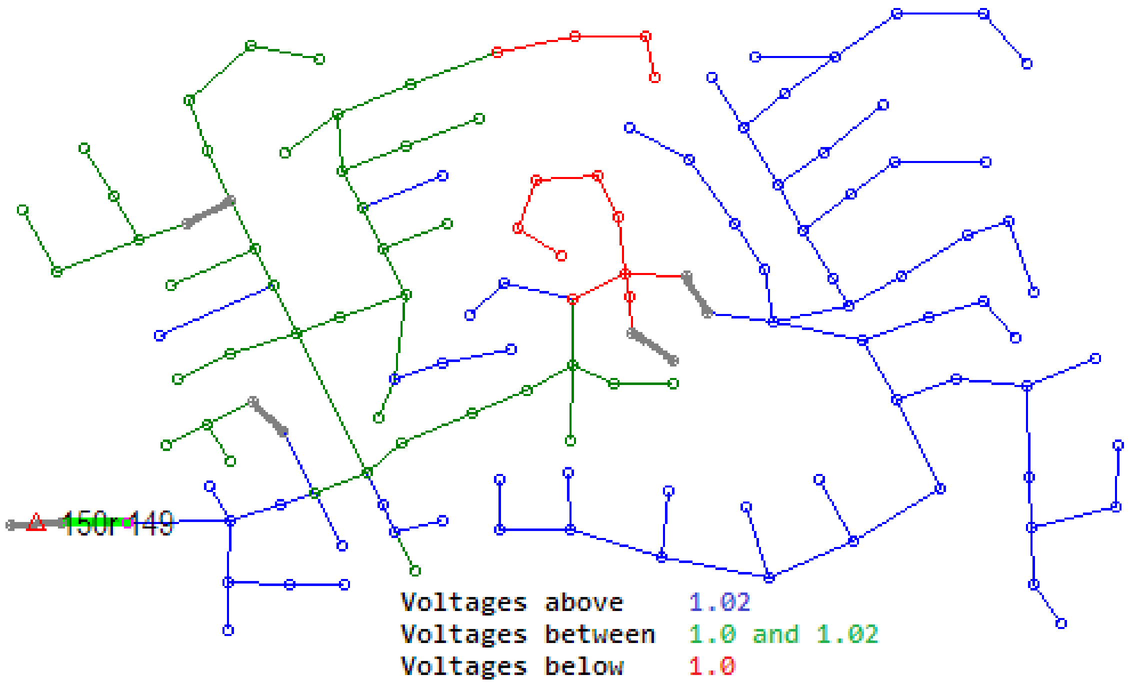


Figure 4.7: Steady state bus Voltages

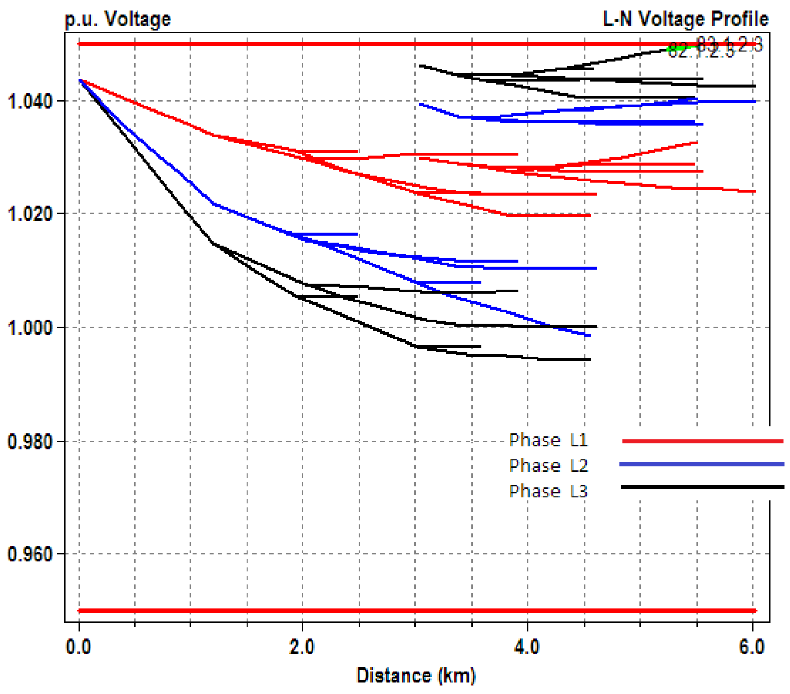


Figure 4.8: Steady state bus Voltages per phases

These are resultant of a plotting script developed to present the plots the way its presented herein. The method to set coordinates as well as the plot script can be found in [44].

4.2 MAS Reconfiguration Approach: Hierarchical Solution

The MAS reconfiguration has been tested in the IEEE 123 test feeder shown in Fig.3.2. This test system is composed of 132 buses, 237 lines, 1 power transformers and 5 current regulators, and a total of 3.4952 MW + j1.22033 MVAr customer load, their position and other useful information are found in [20]. Due to its 8 switches, the test system can be divided in 5 zones. Reconfiguration procedure have been tested for a large variety of outage locations. For the sake of an example, results are shown for a given failure at bus 62 in zone 2.

The hierarchical approach, in which the *dMS* partially dictates the reconfiguration plan, starts with zone isolation performed by busAgents independently. As shown in Fig. 4.9, the isolation is performed by the nearest agent of affected lines.

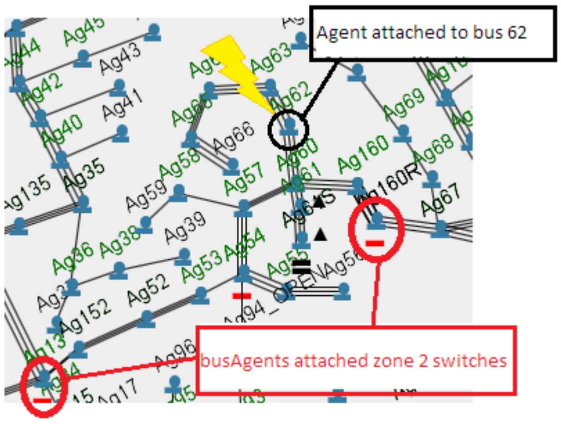
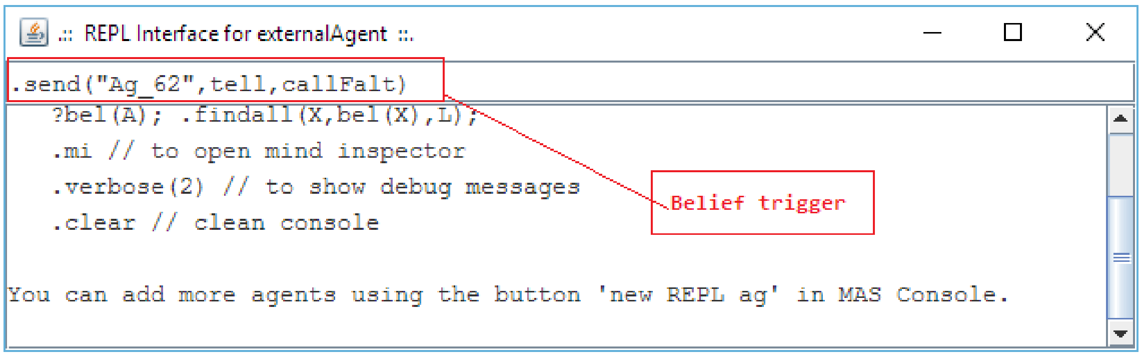


Figure 4.9: Fault location at zone 2

An event/belief called *callFalt*, shown in Fig. 4.10 and unfold in Algorithm 4.1, triggers a plan to handle a fault condition, starting communication between the agents within the zone. Afterwards the *dMS* coordinates the plan to reconfiguration.



```
.send("Ag_62",tell,callFalt)
   ?bel(A); .findall(X,bel(X),L);
   .mi // to open mind inspector
   .verbose(2) // to show debug messages
   .clear // clean console

You can add more agents using the button 'new REPL ag' in MAS Console.
```

Figure 4.10: Fault belief trigger directed to busAgent 62

Algorithm 4.1: callFalt belief

```
+callFalt: true <-
/**For the record*/:
//           actions preceded by dot (.)  = .internal_JaCaMo_function
//           actions without dot (.)   =  artifactOperation(devised, byMe)
        .my_name(Me);
         getBusRealName(RealName)[artifact_id(_)];
         getFaltLine(FaltLine)[artifact_id(_)];
        ?zone(BusZone)[artifact_id(_)];
        .print("******Falt simulation with line Bus Agent ----->",RealName);
        .print("sending DSM request for Plan Of Action of Falt in that zone");
        .send("dMS",tell,getFalts(RealName,FaltLine,BusZone))
```

When the *belief* is acquired by *busAgent 62*, it proceeds to handle the isolation plan and sends *percep/belief* to *dMS* allowing it to acknowledge the fault. The triggering *belief* is depicted in Algorithm 4.2.

Algorithm 4.2: dMS perceptual fault belief

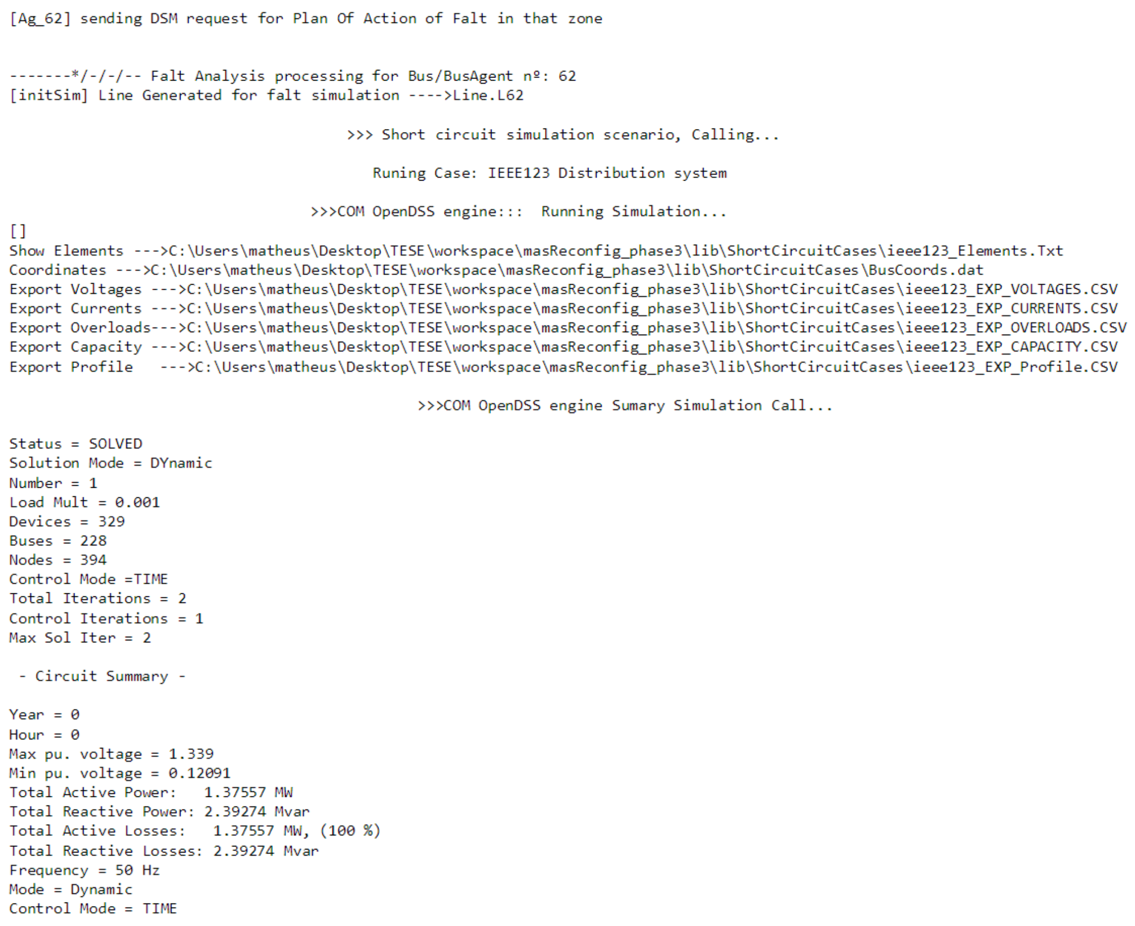
```
+getFalts(BusFalt,LineOfFalt,BusZone)[source(G)]: true <-
           performFalt(BusFalt,LineOfFalt[artifact_id(ArtId)];
          .print("Falt occurency, reported location of falt by --->",G);
          +statusSimulation(0,1);
           stop(_)[artifact_id(ArtId)]; .wait(300);
           getBusData("1",_,_,_,_,BPstring)[artifact_id(ArtId)];
          .broadcast(tell,senseAgain(BPstring));
          +drawGridplotFalt(ArtId);
          !procedReconfiguration(BusZone,Sw1,Sw2,ArtId,G,LineOfFalt)...
```

The agent *dMS* first actuates to perform short-circuit analysis and after triggers the status of the grid *belief* from normal state to faulted state. Afterwards is proceeds by telling all bus agents to acknowledge their state and reason upon it:

$$\texttt{statusSimulation}(0{,}0)[source(self)] \rightarrow \texttt{statusSimulation}(0{,}1)[\text{source(self)}]$$

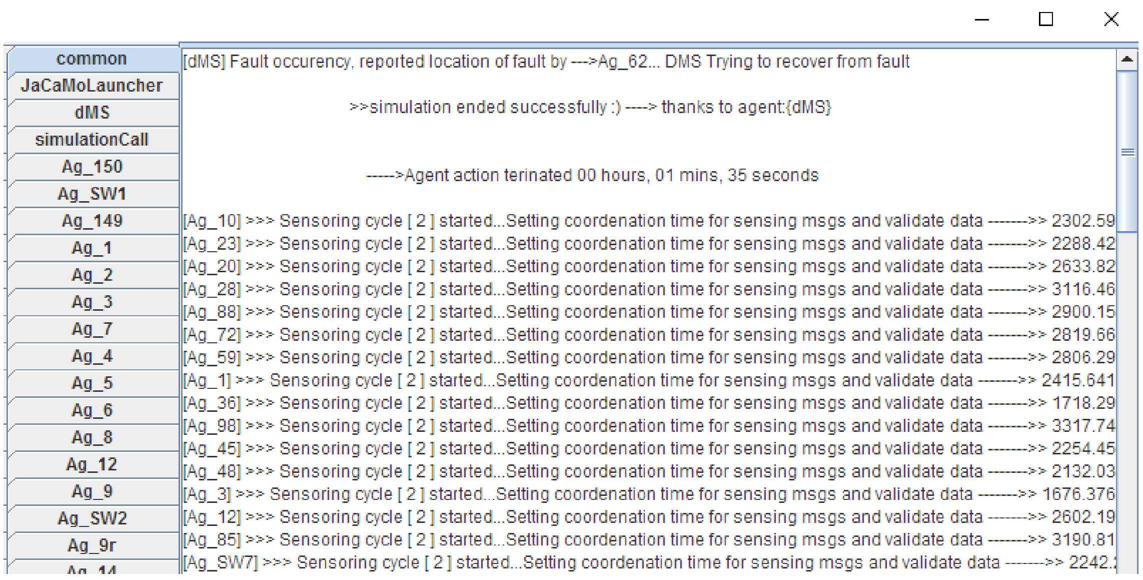
$$\texttt{.broadcast}\ (\textit{tell}, \text{senseAgain(\_)})$$

The results of these course of action is shown in Fig. 4.11, 4.12 and 4.13, where the short-circuit analysis is initialized. BusAgents perceptions, under the grid environmental, change followed by the communication task and reporting their status to dMS.



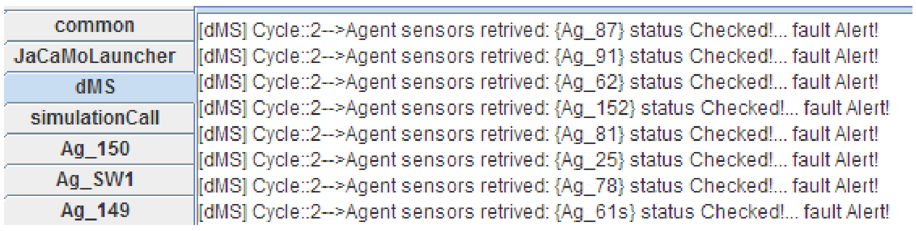
```
[Ag_62] sending DSM request for Plan Of Action of Falt in that zone


-------*/-/-/-- Falt Analysis processing for Bus/BusAgent nº: 62
[initSim] Line Generated for falt simulation ---->Line.L62

                                     >>> Short circuit simulation scenario, Calling...

                                        Runing Case: IEEE123 Distribution system

                                 >>>COM OpenDSS engine:::  Running Simulation...
[]
Show Elements --->C:\Users\matheus\Desktop\TESE\workspace\masReconfig_phase3\lib\ShortCircuitCases\ieee123_Elements.Txt
Coordinates --->C:\Users\matheus\Desktop\TESE\workspace\masReconfig_phase3\lib\ShortCircuitCases\BusCoords.dat
Export Voltages --->C:\Users\matheus\Desktop\TESE\workspace\masReconfig_phase3\lib\ShortCircuitCases\ieee123_EXP_VOLTAGES.CSV
Export Currents --->C:\Users\matheus\Desktop\TESE\workspace\masReconfig_phase3\lib\ShortCircuitCases\ieee123_EXP_CURRENTS.CSV
Export Overloads--->C:\Users\matheus\Desktop\TESE\workspace\masReconfig_phase3\lib\ShortCircuitCases\ieee123_EXP_OVERLOADS.CSV
Export Capacity --->C:\Users\matheus\Desktop\TESE\workspace\masReconfig_phase3\lib\ShortCircuitCases\ieee123_EXP_CAPACITY.CSV
Export Profile   --->C:\Users\matheus\Desktop\TESE\workspace\masReconfig_phase3\lib\ShortCircuitCases\ieee123_EXP_Profile.CSV

                                             >>>COM OpenDSS engine Sumary Simulation Call...

Status = SOLVED
Solution Mode = DYnamic
Number = 1
Load Mult = 0.001
Devices = 329
Buses = 228
Nodes = 394
Control Mode =TIME
Total Iterations = 2
Control Iterations = 1
Max Sol Iter = 2

 - Circuit Summary -

Year = 0
Hour = 0
Max pu. voltage = 1.339
Min pu. voltage = 0.12091
Total Active Power:   1.37557 MW
Total Reactive Power: 2.39274 Mvar
Total Active Losses:   1.37557 MW, (100 %)
Total Reactive Losses: 2.39274 Mvar
Frequency = 50 Hz
Mode = Dynamic
Control Mode = TIME
```

Figure 4.11: Short-circuit analyses after dMS reasoning upon fault



```
[dMS] Fault occurency, reported location of fault by --->Ag_62... DMS Trying to recover from fault

                    >>simulation ended successfully :) ----> thanks to agent:{dMS}


                     ----->Agent action terinated 00 hours, 01 mins, 35 seconds

[Ag_10] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 2302.59
[Ag_23] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 2288.42
[Ag_20] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 2633.82
[Ag_28] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 3116.46
[Ag_88] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 2900.15
[Ag_72] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 2819.66
[Ag_59] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 2806.29
[Ag_1] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 2415.641
[Ag_36] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 1718.29
[Ag_98] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 3317.74
[Ag_45] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 2254.45
[Ag_48] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 2132.03
[Ag_3] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 1676.376
[Ag_12] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 2602.19
[Ag_85] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 3190.81
[Ag_SW7] >>> Sensoring cycle [ 2 ] started...Setting coordenation time for sensing msgs and validate data ------->> 2242.
```

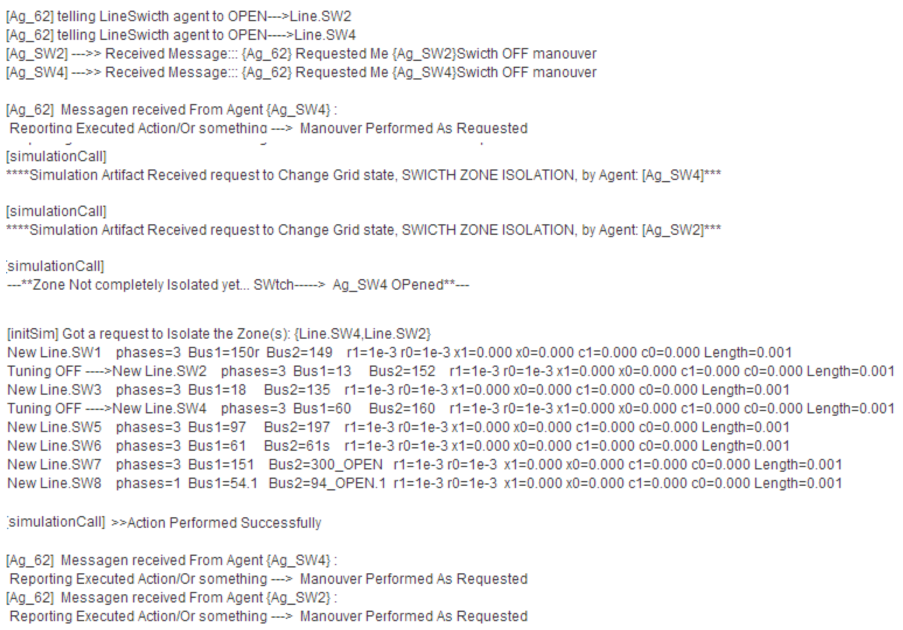
Figure 4.12: Agents acknowledging changes in the environment over fault state



```
[dMS] Cycle::2-->Agent sensors retrived: {Ag_87} status Checked!... fault Alert!
[dMS] Cycle::2-->Agent sensors retrived: {Ag_91} status Checked!... fault Alert!
[dMS] Cycle::2-->Agent sensors retrived: {Ag_62} status Checked!... fault Alert!
[dMS] Cycle::2-->Agent sensors retrived: {Ag_152} status Checked!... fault Alert!
[dMS] Cycle::2-->Agent sensors retrived: {Ag_81} status Checked!... fault Alert!
[dMS] Cycle::2-->Agent sensors retrived: {Ag_25} status Checked!... fault Alert!
[dMS] Cycle::2-->Agent sensors retrived: {Ag_78} status Checked!... fault Alert!
[dMS] Cycle::2-->Agent sensors retrived: {Ag_61s} status Checked!... fault Alert!
```

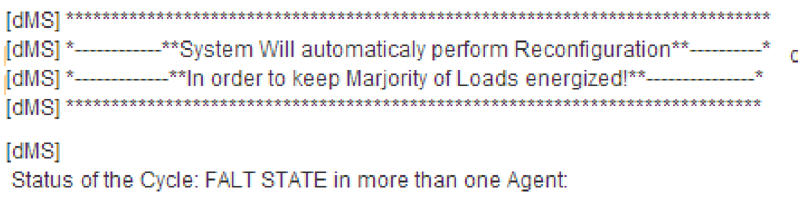
Figure 4.13: dMS reported state from busAgents

Although the course of action seems to be sequential, the execution timing for each participant in the speech act has an asynchronous feature, meaning that the order of action was not a concern when comes to implementing such systems, but rather a matter of acting upon conditions. Therefore, the Ag_62 requests action on isolation to agents attached to switches in the faulted zone right after sensing its short circuit current. This is achieved concurrently as shown in Fig. 4.14. The *dMS* also receives a request to actuate upon fault as shown in Fig. 4.15.



```
[Ag_62] telling LineSwicth agent to OPEN--->Line.SW2
[Ag_62] telling LineSwicth agent to OPEN---->Line.SW4
[Ag_SW2] --->> Received Message::: {Ag_62} Requested Me {Ag_SW2}Swicth OFF manouver
[Ag_SW4] --->> Received Message::: {Ag_62} Requested Me {Ag_SW4}Swicth OFF manouver

[Ag_62]  Messagen received From Agent {Ag_SW4} :
 Reporting Executed Action/Or something --->  Manouver Performed As Requested

[simulationCall]
****Simulation Artifact Received request to Change Grid state, SWICTH ZONE ISOLATION, by Agent: [Ag_SW4]***

[simulationCall]
****Simulation Artifact Received request to Change Grid state, SWICTH ZONE ISOLATION, by Agent: [Ag_SW2]***

simulationCall]
---**Zone Not completely Isolated yet... SWtch----->  Ag_SW4 OPened**---


[initSim] Got a request to Isolate the Zone(s): {Line.SW4,Line.SW2}
New Line.SW1    phases=3  Bus1=150r  Bus2=149    r1=1e-3 r0=1e-3 x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001
Tuning OFF ---->New Line.SW2    phases=3  Bus1=13     Bus2=152    r1=1e-3 r0=1e-3 x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001
New Line.SW3    phases=3  Bus1=18     Bus2=135    r1=1e-3 r0=1e-3 x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001
Tuning OFF ---->New Line.SW4    phases=3  Bus1=60     Bus2=160    r1=1e-3 r0=1e-3 x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001
New Line.SW5    phases=3  Bus1=97     Bus2=197    r1=1e-3 r0=1e-3 x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001
New Line.SW6    phases=3  Bus1=61     Bus2=61s    r1=1e-3 r0=1e-3 x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001
New Line.SW7    phases=3  Bus1=151    Bus2=300_OPEN   r1=1e-3 r0=1e-3  x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001
New Line.SW8    phases=1  Bus1=54.1   Bus2=94_OPEN.1  r1=1e-3 r0=1e-3  x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001

simulationCall] >>Action Performed Successfully

[Ag_62]  Messagen received From Agent {Ag_SW4} :
 Reporting Executed Action/Or something --->  Manouver Performed As Requested
[Ag_62]  Messagen received From Agent {Ag_SW2} :
 Reporting Executed Action/Or something --->  Manouver Performed As Requested
```

Figure 4.14: Automatic zone isolation performed using agents speech act



```
[dMS] ****************************************************************************
[dMS] *------------**System Will automaticaly perform Reconfiguration**----------*   c
[dMS] *-------------**In order to keep Marjority of Loads energized!**---------------*
[dMS] ***************************************************************************

[dMS]
 Status of the Cycle: FALT STATE in more than one Agent:
```

Figure 4.15: dMS reasoning upon fault report

At the moment, the dMS receive confirmation of zone isolation, line switch 2 and 4 both open, and their correspondent agent sends a message to *dMS* which confirms zone isolation. Afterwards a sub-routing with the shortest path algorithm is responsible to find a solution for reconfiguration. The reasoning goes like:

```
begin
    [dMS] DMS awaiting switch agents completion of actions...
    [dMS] Fault scenario completed...
    [dMS] DMS Agent Calling graphic API to generate Plan of Action For reconfiguration
```

When the dMS initialize the algorithm API method, the graph shown in Fig. 4.16 is depicted in a new window. The *grasphstream* library allows one to draw a dynamic node-based graph with the obtained data from the building phase, including coordinates (x,y) of the nodes and the length of the lines attached to them. The grid is interactive given a visual impression on how the decision plan is created.

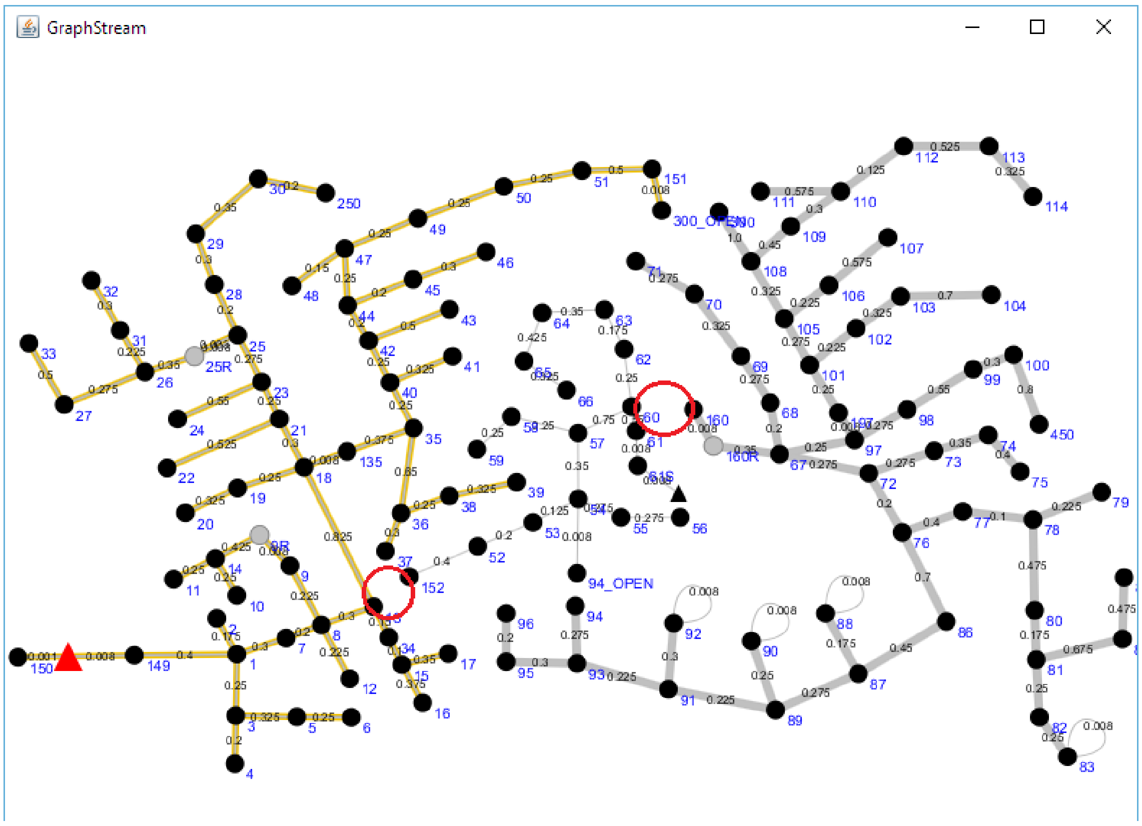
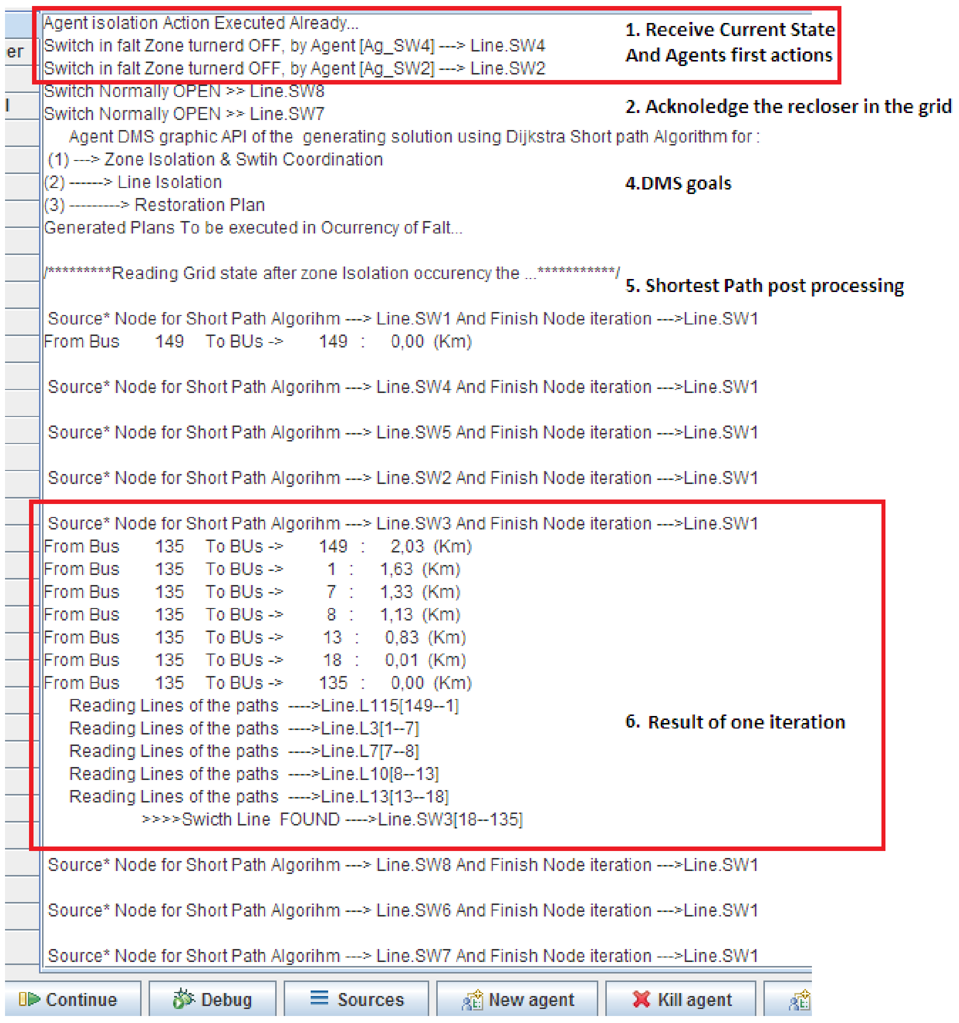


Figure 4.16: Shortest path grid plan

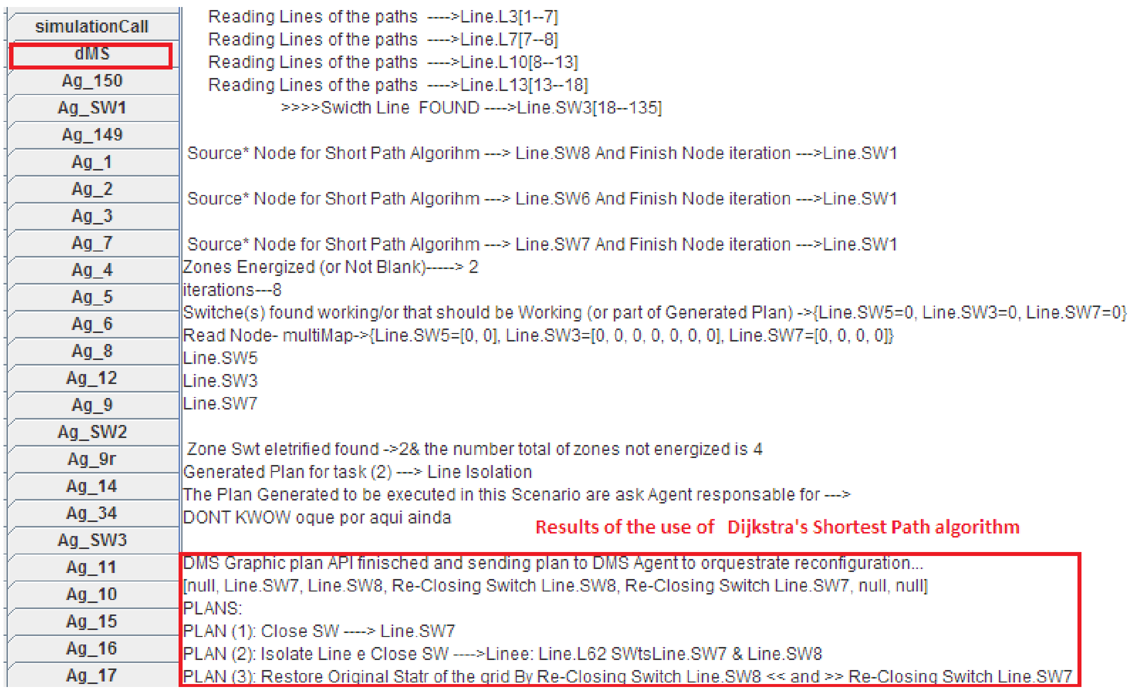
As the algorithm iterates as suggested below, the dMS reports the process of creating a plan as shown in 4.17. Afterwards dMS proceeds by reasoning until the solution is found (4.16), as indicated in Fig.4.19 by implementing/acting upon the generated plans, as shown in the Fig. 4.20.

```
begin
    [$dMS$] Switch(s) found working/or that should be Working - $Line.SW3 = 0$
    [$dMS$] Read Node multiMap->{Line.SW3=[0]}
    [$dMS$] #Switch Plan1 : Got to be Close—-> Line.SW3
    [$dMS$] —-> Geting NOT Energyzed Switch Zones...
    [$dMS$]Zone ->1 Zone Swt with No Power found ->Line.SW4
    [$dMS$]Zone ->2 Zone Swt with No Power found ->Line.SW5
    [$dMS$]Zone ->3 Zone Swt with No Power found ->Line.SW2
    [$dMS$]Zone ->5 Zone Swt with No Power found ->Line.SW6
    [$dMS$] Zone Swt electrified found ->2 & not energized : 4
    [$dMS$]Calculating Solution Using Dijkstra short path algorithm and Agents coordination...
    [$dMS$]Setting source Node...Ag/Bus 150 —> Transformer
    [$dMS$]Generation DMS plan for isolation NORMALY OPEN SWITCH coordination
```



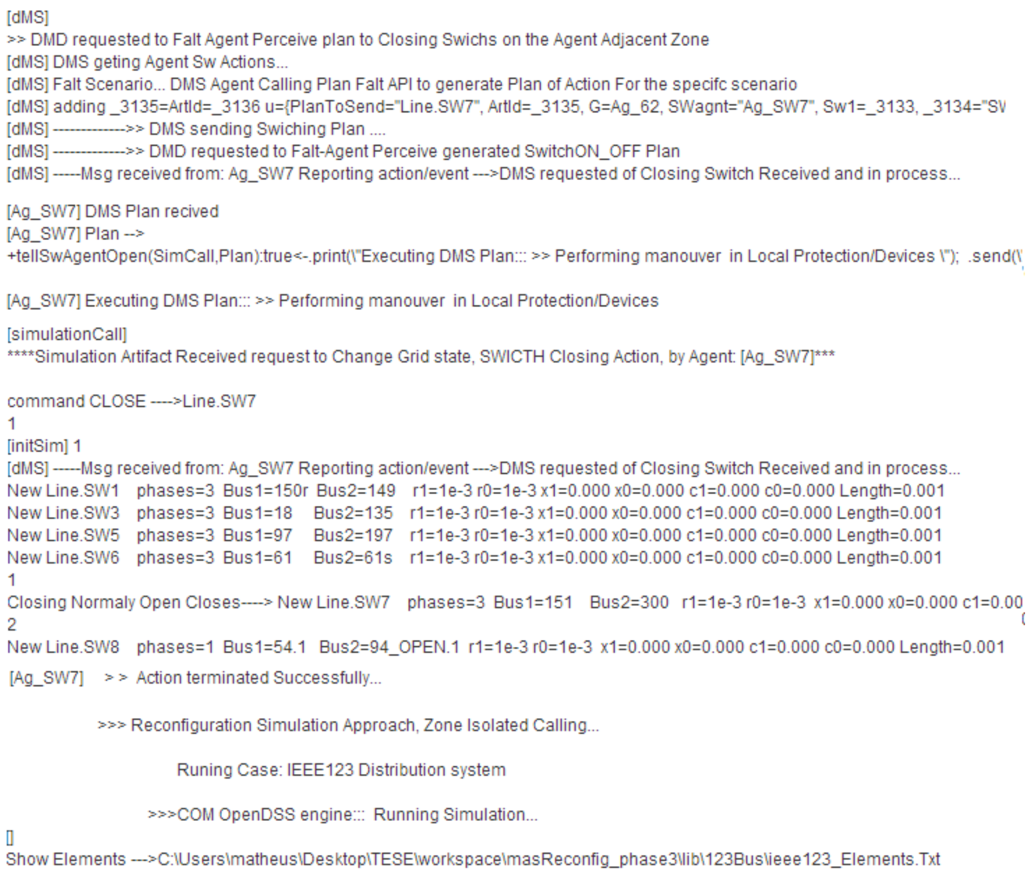
```
Agent isolation Action Executed Already...
Switch in falt Zone turnerd OFF, by Agent [Ag_SW4] ---> Line.SW4
Switch in falt Zone turnerd OFF, by Agent [Ag_SW2] ---> Line.SW2
Switch Normally OPEN >> Line.SW8
Switch Normally OPEN >> Line.SW7
    Agent DMS graphic API of the  generating solution using Dijkstra Short path Algorithm for :
 (1) ---> Zone Isolation & Swtih Coordination
(2) ------> Line Isolation
(3) ---------> Restoration Plan
Generated Plans To be executed in Ocurrency of Falt...

/*********Reading Grid state after zone Isolation occurency the ...***********/

 Source* Node for Short Path Algorihm ---> Line.SW1 And Finish Node iteration --->Line.SW1
From Bus      149    To BUs ->      149  :    0,00  (Km)

 Source* Node for Short Path Algorihm ---> Line.SW4 And Finish Node iteration --->Line.SW1

 Source* Node for Short Path Algorihm ---> Line.SW5 And Finish Node iteration --->Line.SW1

 Source* Node for Short Path Algorihm ---> Line.SW2 And Finish Node iteration --->Line.SW1

 Source* Node for Short Path Algorihm ---> Line.SW3 And Finish Node iteration --->Line.SW1
From Bus      135    To BUs ->      149  :    2,03  (Km)
From Bus      135    To BUs ->        1  :   1,63  (Km)
From Bus      135    To BUs ->        7  :   1,33  (Km)
From Bus      135    To BUs ->        8  :   1,13  (Km)
From Bus      135    To BUs ->       13  :   0,83  (Km)
From Bus      135    To BUs ->       18  :   0,01  (Km)
From Bus      135    To BUs ->      135  :   0,00  (Km)
    Reading Lines of the paths  ---->Line.L115[149--1]
    Reading Lines of the paths  ---->Line.L3[1--7]
    Reading Lines of the paths  ---->Line.L7[7--8]
    Reading Lines of the paths  ---->Line.L10[8--13]
    Reading Lines of the paths  ---->Line.L13[13--18]
              >>>>Swicth Line  FOUND ---->Line.SW3[18--135]

 Source* Node for Short Path Algorihm ---> Line.SW8 And Finish Node iteration --->Line.SW1

 Source* Node for Short Path Algorihm ---> Line.SW6 And Finish Node iteration --->Line.SW1

 Source* Node for Short Path Algorihm ---> Line.SW7 And Finish Node iteration --->Line.SW1
```

Figure 4.17: dMS reasoning upon fault report



```
  Reading Lines of the paths  ---->Line.L3[1--7]
  Reading Lines of the paths  ---->Line.L7[7--8]
  Reading Lines of the paths  ---->Line.L10[8--13]
  Reading Lines of the paths  ---->Line.L13[13--18]
            >>>>Swicth Line  FOUND ---->Line.SW3[18--135]

 Source* Node for Short Path Algorihm ---> Line.SW8 And Finish Node iteration --->Line.SW1

 Source* Node for Short Path Algorihm ---> Line.SW6 And Finish Node iteration --->Line.SW1

 Source* Node for Short Path Algorihm ---> Line.SW7 And Finish Node iteration --->Line.SW1
Zones Energized (or Not Blank)-----> 2
iterations---8
Switche(s) found working/or that should be Working (or part of Generated Plan) ->{Line.SW5=0, Line.SW3=0, Line.SW7=0}
Read Node- multiMap->{Line.SW5=[0, 0], Line.SW3=[0, 0, 0, 0, 0, 0, 0], Line.SW7=[0, 0, 0, 0]}
Line.SW5
Line.SW3
Line.SW7

 Zone Swt eletrified found ->2& the number total of zones not energized is 4
Generated Plan for task (2) ---> Line Isolation
The Plan Generated to be executed in this Scenario are ask Agent responsable for --->
DONT KWOW oque por aqui ainda

DMS Graphic plan API finisched and sending plan to DMS Agent to orquestrate reconfiguration...
[null, Line.SW7, Line.SW8, Re-Closing Switch Line.SW8, Re-Closing Switch Line.SW7, null, null]
PLANS:
PLAN (1): Close SW ----> Line.SW7
PLAN (2): Isolate Line e Close SW ---->Linee: Line.L62 SWtsLine.SW7 & Line.SW8
PLAN (3): Restore Original Statr of the grid By Re-Closing Switch Line.SW8 << and >> Re-Closing Switch Line.SW7
```

Figure 4.18: TIEEE123test feeder shortest path grid



```
[dMS]
>> DMD requested to Falt Agent Perceive plan to Closing Swichs on the Agent Adjacent Zone
[dMS] DMS geting Agent Sw Actions...
[dMS] Falt Scenario... DMS Agent Calling Plan Falt API to generate Plan of Action For the specifc scenario
[dMS] adding _3135=ArtId=_3136 u={PlanToSend="Line.SW7", ArtId=_3135, G=Ag_62, SWagnt="Ag_SW7", Sw1=_3133, _3134="SV
[dMS] ------------->> DMS sending Swiching Plan ....
[dMS] ------------->> DMD requested to Falt-Agent Perceive generated SwitchON_OFF Plan
[dMS] -----Msg received from: Ag_SW7 Reporting action/event --->DMS requested of Closing Switch Received and in process...

[Ag_SW7] DMS Plan recived
[Ag_SW7] Plan -->
+tellSwAgentOpen(SimCall,Plan):true<-.print(\"Executing DMS Plan::: >> Performing manouver  in Local Protection/Devices \");  .send(\

[Ag_SW7] Executing DMS Plan::: >> Performing manouver  in Local Protection/Devices

[simulationCall]
****Simulation Artifact Received request to Change Grid state, SWICTH Closing Action, by Agent: [Ag_SW7]***

command CLOSE ---->Line.SW7
1
[initSim] 1
[dMS] -----Msg received from: Ag_SW7 Reporting action/event --->DMS requested of Closing Switch Received and in process...
New Line.SW1   phases=3  Bus1=150r  Bus2=149   r1=1e-3 r0=1e-3 x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001
New Line.SW3   phases=3  Bus1=18    Bus2=135   r1=1e-3 r0=1e-3 x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001
New Line.SW5   phases=3  Bus1=97    Bus2=197   r1=1e-3 r0=1e-3 x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001
New Line.SW6   phases=3  Bus1=61    Bus2=61s   r1=1e-3 r0=1e-3 x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001
1
Closing Normaly Open Closes----> New Line.SW7   phases=3  Bus1=151   Bus2=300   r1=1e-3 r0=1e-3  x1=0.000 x0=0.000 c1=0.00
2
New Line.SW8   phases=1  Bus1=54.1   Bus2=94_OPEN.1  r1=1e-3 r0=1e-3  x1=0.000 x0=0.000 c1=0.000 c0=0.000 Length=0.001
 [Ag_SW7]    > >  Action terminated Successfully...

               >>> Reconfiguration Simulation Approach, Zone Isolated Calling...

                              Runing Case: IEEE123 Distribution system

                          >>>COM OpenDSS engine:::  Running Simulation...
[]
Show Elements --->C:\Users\matheus\Desktop\TESE\workspace\masReconfig_phase3\lib\123Bus\ieee123_Elements.Txt
```

Figure 4.19: The reconfiguration achievement summary

To this point, the plan is to close the *SW7* and the other switches not including the ones of faulted isolated zone. By default *Line.SW7* is a normally open switch. The reconfiguration is then achieved and the final result can be shown in 4.20. In order to analzyse the reconfigured grid scenario, another power flow analysis is trigger for the agents to sense.

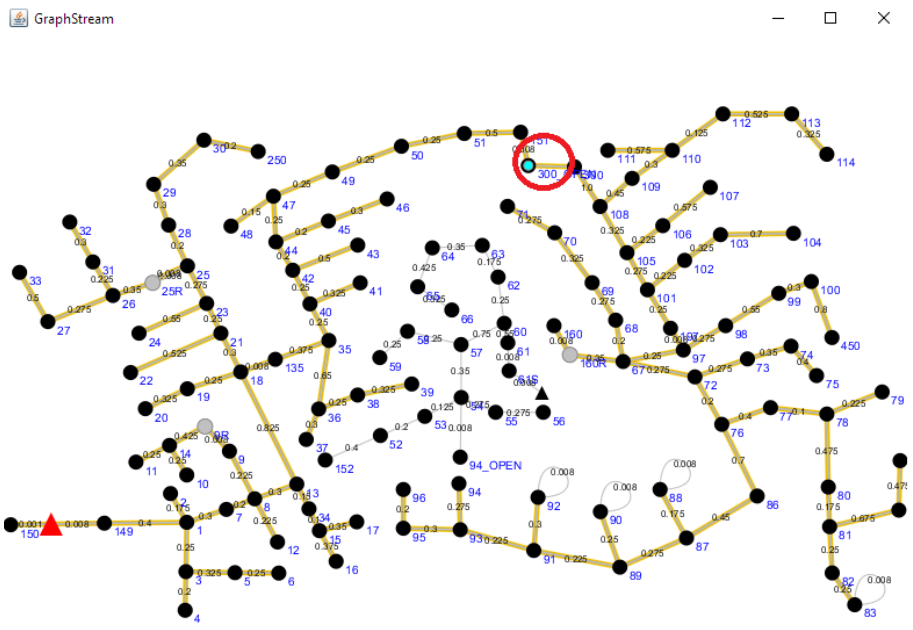


Figure 4.20: The reconfiguration shortest-path validation

The reconfiguration cycle is now complete, as a minimum number of customer nodes are non energized. One way of evaluating the simulation cycles presented is by resorting to the plot results generated by the OpenDSS. These evaluations are briefly put in the next topic.

The graph in figure 4.21 is refereed to the grid state right after the short-circuit simulation and when busAgents start the course for reconfiguration reconfiguration in order to change that state. Figure 4.22 is the second state of the simulation and when dMS start planing to reconfigure. Figure 4.23 shows the voltage profiles in the of reconfigured grid and lastly in figure 4.24 it is the power flow intensity depicted.

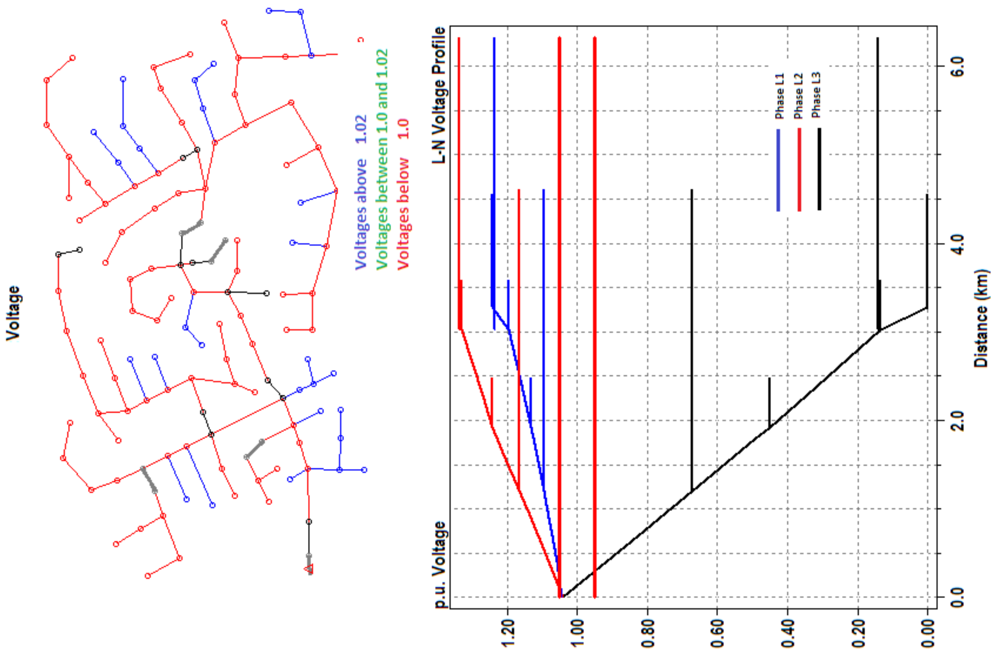


Figure 4.21: Short-circuit results

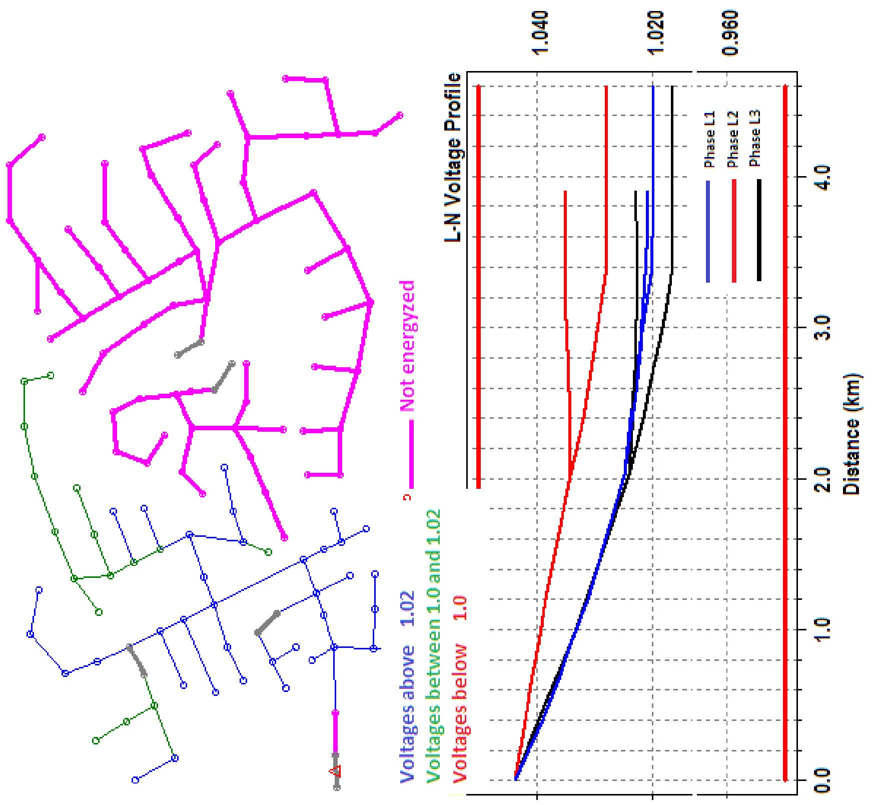


Figure 4.22: Post zone isolation result

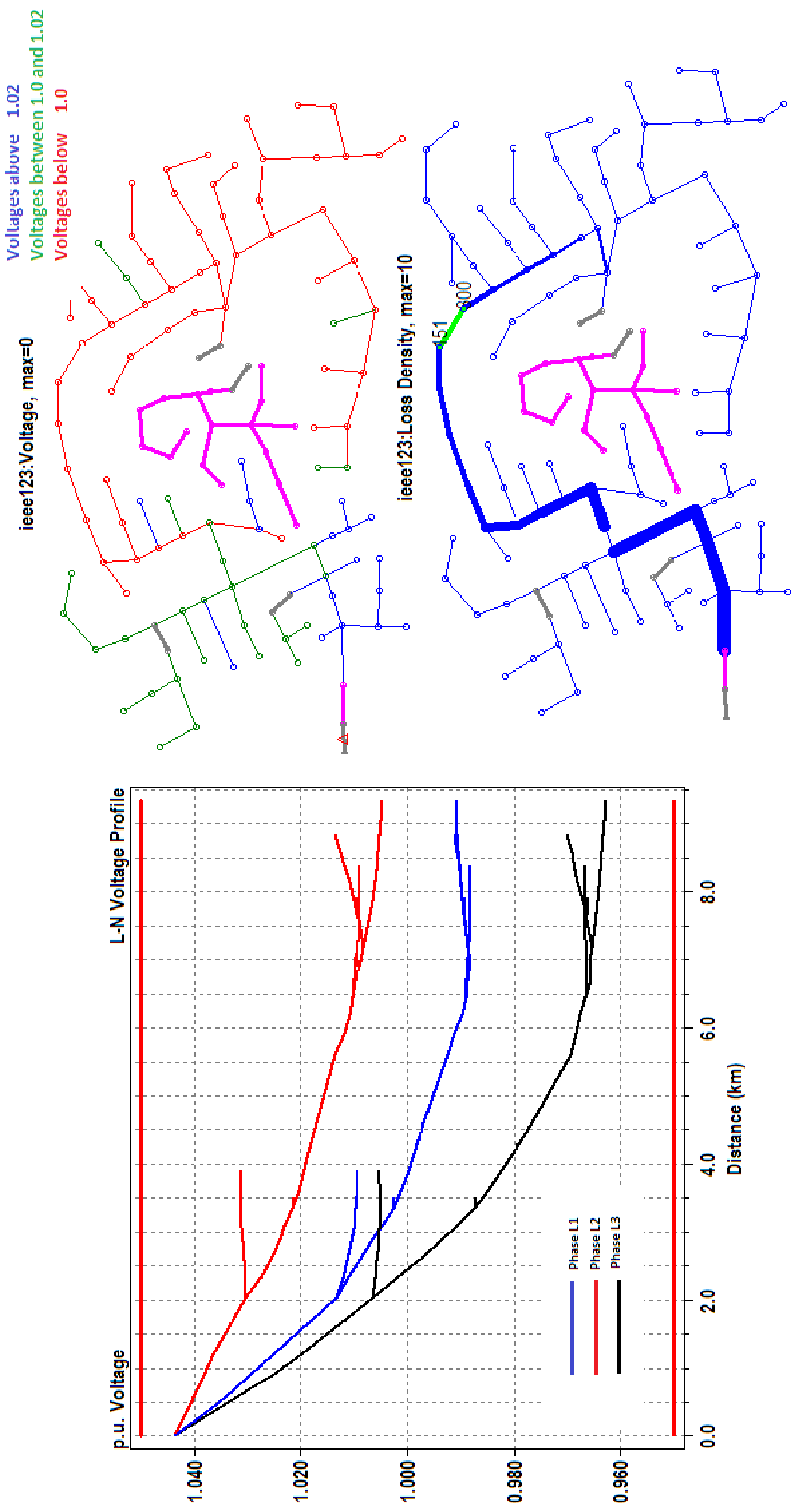


Figure 4.23: Reconfiguration results

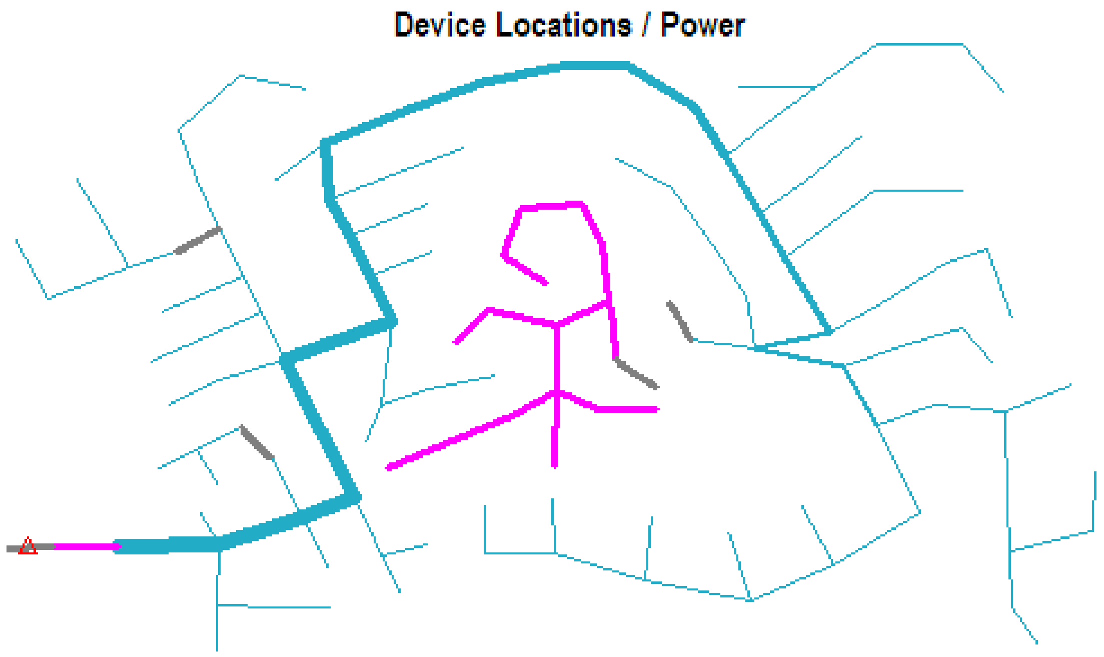
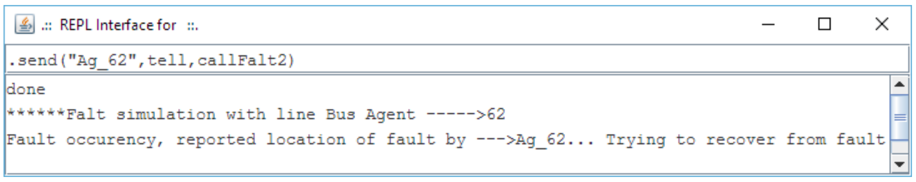


Figure 4.24: Power flow results

These figures represents all grid states the agents inside the framework were capable of sensing and reasoning upon gird objects.

4.3 MAS Reconfiguration Approach: Decentralized Solution

The decentralized solution is taken following the same considerations as the hierarchic solution. It is considered the case scenario in which a fault is simulated for a line near bus 62, as depicted in Fig. 4.25. The only alteration in comparison to the previous case is the inclusion of agent-agent interaction to avoid depending on plan executed by *dMS*.



```
.send("Ag_62",tell,callFalt2)
done
******Falt simulation with line Bus Agent ----->62
Fault occurency, reported location of fault by --->Ag_62... Trying to recover from fault
```

Figure 4.25: Fault perceptual belief Trigger form decentralized approach

From the start, when a trigger event caring a signal of fault is acquired by agent 62, the immediate response is to run a short-circuit simulation. The *trigger* is shown in Algorithm 4.3:

```
begin
    [Ag_62]******Fault simulation with line attach to Bus ——>62
    [Ag_62]———-*/-/-/– Falt Analysis processing for Bus/BusAgent nº: 62
    [initSim]*Line Generated for falt simulation —->Line.L62
```

Algorithm 4.3: callFalt2 belief

```
+callFalt2: true <-
      .my_name(Me);
        getBusRealName(RealName)[artifact_id(S)];
        getFaltLine(FaltLine)[artifact_id(S)];
        ?zone(BusZone)[artifact_id(S)];
       .print("******Falt simulation with line Bus Agent ----->",RealName);
       .send("simulationCall",tell,getFalts2(RealName,FaltLine,BusZone));
```

The subroutine generates the short circuit-scenario with the same simulation summary result as shown in Fig. 4.11 page 62. The course of action taken in AgentSpeak is depicted in Algorithm 4.4.

Algorithm 4.4: callFalt2 belief

```
+getFalts2(BusFalt,LineOfFalt,BusZone)[source(G)]: true <-
             performFalt(BusFalt,LineOfFalt)[artifact_id(ArtId)];
             +statusSimulation(0,1);
             stop(_)[artifact_id(ArtId)];
             getBusData("1",_,_,_,_,BPstring)[artifact_id(ArtId)];
             .broadcast(tell,senseAgain(BPstring));
```

A command action, such as ".*broadcast(tell,senseAgain(_))* ", gives the agent the power to acknowledge a fault state after short-circuit. What differentiates this approach from the previous one is when a fault occur the busAgent 62 initiates a speech act between all the agents attached to normally close switches and ask them to open, instead of recurring to the *dMS*. The communication occur as shown below, as illustrated in Fig. 3.17 page 49.

```
begin
    [simulationCall] attempting to Fault restoration by Agents speech Act coordination [simulationCall]
    Received request to open switch from [Ag_SW1]
    [initSim] switching OFF :Line.SW1 phases = 3Bus1 = 150rBus2 = 149...
    [simulationCall] Received request to open switch from [Ag_SW2]
    [initSim] switching OFF : Line.SW2 phases=3 Bus1=13 Bus2=152 Bus2=135..
    [simulationCall] Received request to open switch from [Ag_SW3]
    [initSim] switching OFF : Line.SW3 phases=3 Bus1=18 Bus2=135
    [simulationCall] Received request to open switch from [Ag_SW4]
    [initSim] switching OFF : Line.SW4 phases=3 ..
    [simulationCall] Received request to open switch from [Ag_SW5]
    [initSim] switching OFF : Line.SW5 ....
    [simulationCall] Received request to open switch from [Ag_SW6]
    [initSim] switching OFF : New Line.SW6 ..
```

At this point, a suitable scenario is found where the agents achieve a reconfiguration, as illustrated in Fig. 4.26. Then after, the agents start communicating with its neighboring agents asking if they are attached to energized switches. If the response is affirmative, the neighboring switch is closed, assuming restrictions for the connection are none. These activities go until all possible zones are energized as illustrated below.

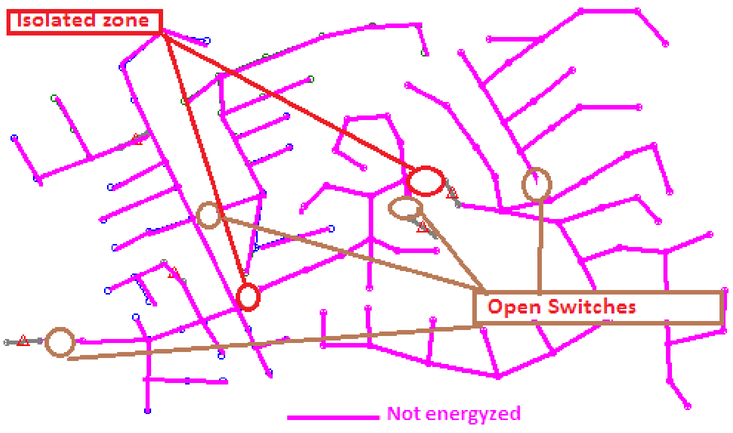


Figure 4.26: Post faulted state

An example of the speech act is demonstrated bellow. It assumes busAgent 149 and 19 (according to Fig. 4.27) are close to busAgents and attached to switches (busAgent 150R and 18). For the first reasoning cycle, all agents ask their neighbors if they are attached to an energized switch. Considering busAgent 20 close to the busAgent 19 asks if he is close to energized switch. The cprresponding speech act is depicted as follows:

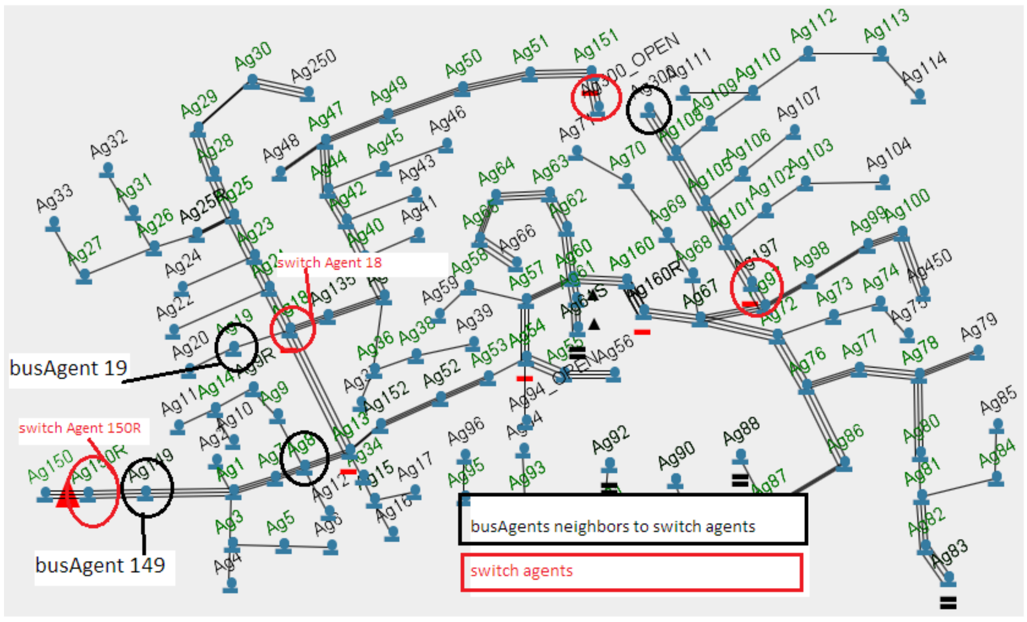
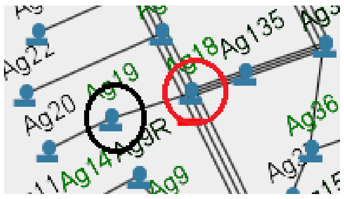


Figure 4.27: Fault perceptual belief Trigger form decentralized approach

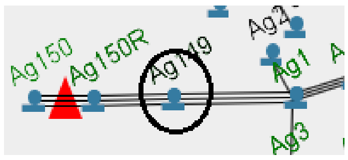
begin



```
[Ag_20] Asking {Ag_19} if neighbors Agents are close to energized switch
[Ag_20] Agent *{Ag_19}* Asked :Are you close to energized switch!?...
[Ag_20] ***Received answer From Agent {Ag_19} : no
```

Same reasoning circle for the busAgent 1:

begin



```
[Ag_1] Asking ∗{Ag_7}∗ if neighbors Agents are close to energized switch
[Ag_1] Agent {Ag_3} Asked :Are you close to energized switch!?...
[Ag_1] Agent {Ag_149} Asked :Are you close to energized switch!?...
[Ag_1] Asking ∗{Ag_2}∗ if neighbors Agents are close to energized switch
[Ag_1] Agent {Ag_7} Asked :Are you close to energized switch!?...
[Ag_1] ***Received answer From Agent {Ag_2} : no
[Ag_1] ***Received answer From Agent {Ag_7} : no
[Ag_1] ***Received answer From Agent {Ag_3} : no
[Ag_1] ***Received answer From Agent {Ag_149} : yes
[Ag_1] Asking to close...
[Ag_149] Asking to close...
[simulationCall] Received request to Change Grid state, CLOSE SWITCH, by Agent: {Ag_149}
[initSim] Got a request to close switch
[initSim] Restoring switch starting position> New Line.SW1 phases=3 Bus1=150r Bus2=149....
[simulationCall] »Action Performed Successfully
```

As the speech act interactions above suggests, the grid is restoring supply at a sequential manner. Let us analyses what happens in the reasoning cycle where the agent 1 receives the message that its neighbor is energized in one end of its attached switch. The resulted configuration is shown in Fig. 4.28.

For the ongoing reasoning cycle the ideas is the same:

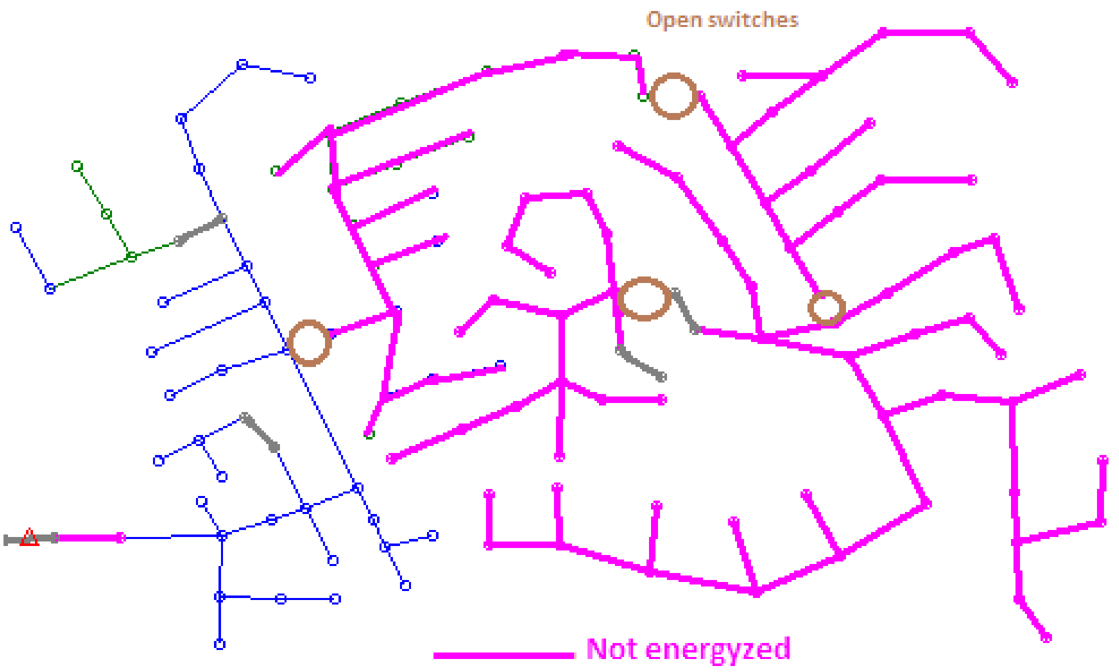
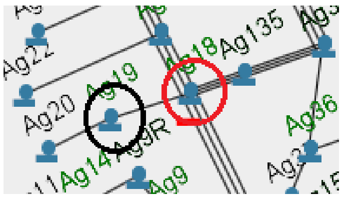


Figure 4.28: Grid state after first reasoning cycle

begin



[*Ag*_20] Asking {Ag_19} if neighbors Agents are close to energized switch
[*Ag*_20] Agent *{Ag_19}* Asked :Are you close to energized switch!?...
[*Ag*_20] ***Received answer From Agent {Ag_19} : yes
[*Ag*_1] Asking to close...
[*Ag*_19] Asking to close...
[*simulationCall*] Received request to Change Grid state, CLOSE SWITCH, by Agent: {*Ag*_149}
[*initSim*] Got a request to close switch
[*initSim*] Restoring switch starting position> New Line.SW1 phases=3 Bus1=18 Bus2=135....
[*simulationCall*] »Action Performed Successfully

The grid state at this point is the same illustrate in Fig. 4.29. This acts will continue until the grid reaches its maximum reconfigurable state. The continuation of these acts will then result in a reconfigured grid as shown in Fig. 4.30

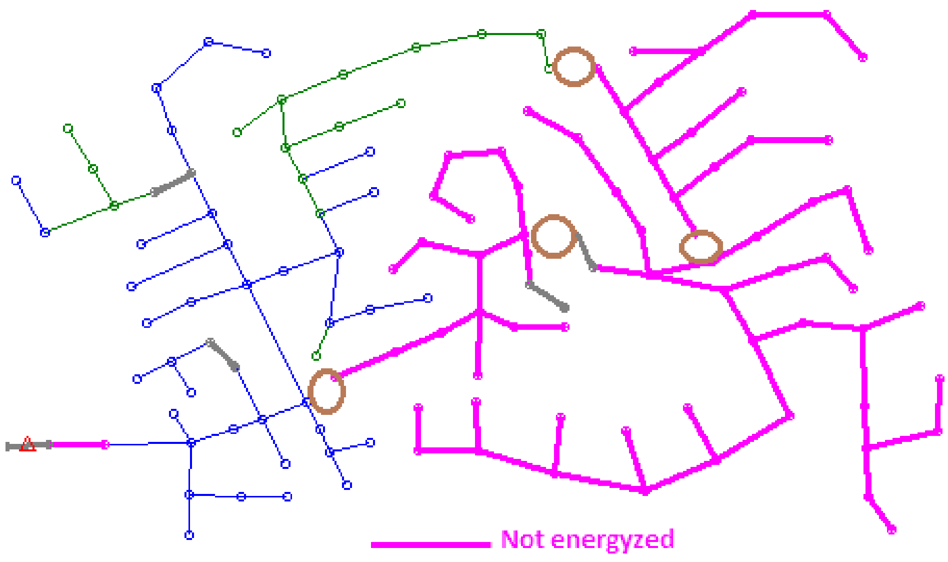


Figure 4.29: Grid state after second reasoning cycle

As the simulation goes, and given the same simulation scenario, the overall results regards the grid final configuration can be compared to the results obtained in the previous solution, shown in 4.30.

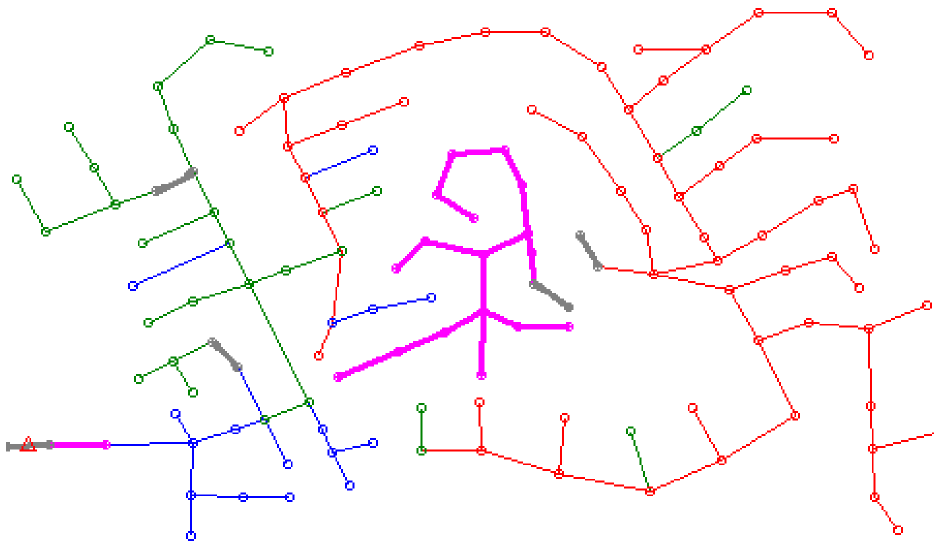


Figure 4.30: Final Grid state

As final remarks one can observe that both approaches provide satisfactory solution either following centralized or decentralized approach.

4.4 Conclusion

In the first section of these chapter, a brief review on the MAS framework basic user interface functionalities and an idea of how it works. This is follow by an introduction to the MAS building phase, regarding its first steps from the initiation to an simulated distribution grid environment. This empathes how agent and artifact coordinate to achieve the integration.

In the following section, consideration on the grid to be analyzed have been made referring its basic structural data. Proceeding to first simulation case, assuming a hierarchical approach, the simulation has been conducted by analyzing the capabilities of the application to handle a fault in a line of the grid environment, where the faulted nearest bus agent is responsible to initiate the process of reconfiguration.

The ways the agents within the environment handles a fault isolation solution and further a reconfiguration was by cooperation among busAgents and a dMS agent. The dMS was responsible to generate a plan that better fits the scenario by utilizing an shortest-path algorithm, resulting in reconfigured system minimizing the non-customer supply.

To the second solution a different approach is taken where the system no longer rely on a dMS plan solution to achieve a reconfiguration. In the procedure, busAgents take the lead and do it by themselves by communicating with neighboring agents, and achieve a reconfigured grid.

Chapter 5

Final Remarks

This chapter completes the thesis by stating its conclusions and giving a series of research topics to be explored in the future.

5.1 Conclusions

This thesis has been conducted in a context which promotes the use of agent-based technology to model a smart approach for the purpose of reconfiguration. For that matter, a framework for MAS that combines three separate state of the art technologies was choosen to suit this purpose: Jason, CArTaGo, Moise. Also an OpenDSS distribution grid model was integrated in the simulation platform, including all structures responsible to represent a three-phase distribution grid.

The first proposed approach implemented a centralized model which uses a graph-theoretic distribution restoration that applies the shortest-path algorithm search technique to find the network topologies capable of minimizing the number of out of service loads. This provides an optimal solution, that is a restoration plan with minimum switching operations for a restoration course.

The second proposed approach provided a simple and effective method to handle outage management and was implemented based on a decentralized model for reconfiguration. The fundamental ideas implies on having agents widespread on the network which communicate to ask permission for zone energization. Power flow calculations with detailed network model are performed to ensure that the network topologies suggested either by the proposed algorithm or the decentralized approach will be operated within the electrical and operation limits.

5.2 Future Works

This thesis scope is focused on outages and automatic systems, consequently in the followings future research topics are described and discussed:

1. Aggregate other types of features such as, load shedding, islanding, voltage regulation, integrate photovoltaic and so forth;

2. Another methods an algorithms can be combined with a load shedding scheme to enhances system capabilities;

3. A comprehensive and detailed relaying system together with a communication system modeling should be investigated in the future.

4. Integrate communications constraints to the model, concerning applied technology, be that : optical fibber, wirelesses access points (WAPs) or other.

5. Integrate methods to represent a qualitative study of the solutions through service quality indexes.

Appendix A

simulationCAll and dMS agents Main functions

Table A.1: simulationCall and dMS Agents Artifact List of Operation

Function	Description
@OPERATION init()	Initialize artifact operations
@OPERATION start()	Instantiate the COM OpenDSS API and Power flow
@OPERATION cmd()	Store OPendDSS command written in AgentSpeak
@OPERATION cmdView()	Show MAS specific agent behaviour
@OPERATION cmdClear()	Clear console
@OPERATION setAgentNumber()	Post processing: return number of busses
@OPERATION getAgentName()	Post processing: return bus name
@OPERATION getBusData()	Post processing: return Bus data object
@OPERATION incSim()	Increment simulation cycle
@OPERATION inc()	Increment simulation fault count
@OPERATION incClear()	clear inc
@OPERATION openALLsw()	Open switch actuator
@OPERATION plotGrid()	Plot grid
@OPERATION getActionForZone()	Actuator: pass OpenDSS command
@OPERATION zoneIsolationActionFor()	Actuator: pass OpenDSS command
@OPERATION lineIsolationActionFor()	Actuator: pass OpenDSS command
@OPERATION performIsolationPF()	Actuator: pass OpenDSS command
@OPERATION performReconPF()	Actuator: pass OpenDSS command powerflow
@OPERATION callPlanAPI()	Actuator: pass OpenDSS command Power flow
@OPERATION getActionSwtOpen()	speech action
@OPERATION generatedPlan()	speech action
@OPERATION reconfigurationPF()	Actuator: pass OpenDSS command Power flow
@OPERATION performFalt()	Actuator: pass OpenDSS command short-circuit
@OPERATION callForIsolationZonePlan()	Speech act: for isolation
@INTERNAL_OPERATION void waitIngtime()	Internal function to handle thread time internaly
runSimulation()	method: post processing power flow simulation handling objects
readFalt()	method: CSV file with fault analysis
readLine()	method: read line of text file
getCordinate()	method: read coordinates OpenDSS file
class LinesDrawing	class that implement grid drawing
LinesReverseIsolation()	method: restore dss files for new MAS simulation
SwitchConfig()	method: handle switch OpenDSS file
SwitchAlloff()	method: handle switch OpenDSS file
SwitchConfigRestore()	method: restore dss files for new MAS simulation
readZones()	method: read text file with zones
getLineOfFalt()	method: set fault line
setLineOfFalt()	method: get fault line

Table A.2: Bus Agent Artifact List of Operation

Function	Description
@OPERATION init()	Initialize artifact operations
@OPERATION incSim()	Increment agent fault reasoning cycle
@OPERATION getBusRealName()	Increment agent simulation cycle
@OPERATION attachedTo()	Actuator: implement speech act action
@OPERATION faltCommunication()	Actuator: implement speech act action
@OPERATION isclosetosw()	Actuator: implement speech act action
@OPERATION senseCurrents()	Actuator: implement sensing feature
@OPERATION senseVoltages()	Actuator: implement sensing feature
@OPERATION busPath()	Actuator: implement post processing
@OPERATION incClear()	clear incSim
@OPERATION inc()	clear inc
@OPERATION getZone()	Actuator: implement read zone in the environment
@INTERNAL_OPERATION void waitIngtime()	Internal function to handle thread time internaly
SwitchRecon2()	method: post processing from actuator
randDouble()	method: post processing from actuator
readLine()	method: read line of text file
readResultfileVolt()	method: read line of CSV file
readResultfileCurr()	method: read line of CSV file
getNodes()	method: read line of grahp file

Bibliography

[1] CArtAgO. URL: http://cartago.sourceforge.net/.

[2] cartago_by_examples.

[3] Transforming the Grid to Revolutionize Electric Power in North America Transforming the Grid to Revolutionize Electric Power in North America. 2003.

[4] Amir Abiri-Jahromi, Mahmud Fotuhi-Firuzabad, Masood Parvania, and Mohsen Mosleh. Optimized Sectionalizing Switch Placement Strategy in Distribution Systems. *IEEE Transactions on Power Delivery*, 27(1):362–370, 1 2012. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6072299, doi:10.1109/TPWRD.2011.2171060.

[5] asanti aricci. *CArtAgO By Example*, volume 2.0.1. 2010.

[6] M.E. Baran and F.F. Wu. Network reconfiguration in distribution systems for loss reduction and load balancing. *IEEE Transactions on Power Delivery*, 4(2):1401–1407, 4 1989. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=25627, doi:10.1109/61.25627.

[7] Carole Bernon, Massimo Cossentino, and Juan Pavón. An Overview of Current Trends in European AOSE Research. *Informatica*, 29(4):379–390, 2005.

[8] Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. JaCaMo Project. URL: http://jacamo.sourceforge.net/.

[9] Rafael H. Bordini, Lars Braubach, Mehdi Dastani, Amal El Fallah-Seghrouchni, Jorge J. Gómez-Sanz, João Leite, Gregory M. P. O'Hare, Alexander Pokahr, and Alessandro Ricci. A Survey of Programming Languages and Platforms for Multi-Agent Systems., 2006.

[10] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. 2007. doi:10.1002/9780470061848.

[11] Michael E Bratman, John Broome, Sarah Paul, and Jeffrey Seidman. Intention, Practical Rationality, and Self-Governance. *Ethics*, 119:411–443, 2009.

[12] PROFFITT BRIAN. What APIs Are And Why They're Important. URL: http://readwrite.com/2013/09/19/api-defined/.

[13] buraq group. scada network. URL: https://www.csiac.org/journal-article/the-efficacy-and-challenges-of-scada-and-smart-grid-integration/.

[14] Hong-Chan Chang and Cheng-Chien Kuo. Network reconfiguration in distribution systems using simulated annealing. *Electric Power Systems Research*, 29(3):227–238, 5 1994. URL: http://linkinghub.elsevier.com/retrieve/pii/0378779694900183, doi:10.1016/0378-7796(94)90018-3.

[15] H.-D. Chiang and R. Jean-Jumeau. Optimal network reconfigurations in distribution systems. I. A new formulation and a solution methodology. *IEEE Transactions on Power Delivery*, 5(4):1902–1909, 1990. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=103687, doi:10.1109/61.103687.

[16] Chiou J Chang C Su C. Variable Scaling Hybrid Differential Evolution for Solving Network Reconfiguration of Distribution Systems. *IEEE Transactions on Power Systems*, 2005. URL: ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1425559, doi:10.1109/TPWRS.2005.846096.

[17] Gordon R. Clarke, Deon. Reynders, and Edwin Wright. *Practical modern SCADA protocols DNP3, 60870.5 and related systems*. Elsevier, 2004.

[18] Consortium of Electric Infrastructure to Support a Digital Society (CEIDS). Overview of IntelliGrid Architecture. URL: http://www.intelligrid.info/IntelliGrid{_}Architecture/Overview{_}Guidelines/Frm{_}Overview.htm.

[19] Edvard Csanyi. Important Primary Distribution (Radial and Loop) System Considerations. URL: http://electrical-engineering-portal.com/author/edvard.

[20] Distribution System Analysis Subcommittee. IEEE 123 Node Test Feeder Letterhead.

[21] Don Von Dollen, Joe Hughes, and Paul Haase. INTELIGRID : A Smart Network of Power. *EPRI Journal*, page 32, 2005. URL: http://mydocs.epri.com/docs/CorporateDocuments/EPRI{_}Journal/2005-Fall/1012885{_}IntelliGrid.pdf.

[22] Roger C Dugan. Reference Guide The Open Distribution System Simulator (OpenDSS). 2013.

[23] Venizelos Efthymiou, George Huitema, and et. al. The Digital Energy System 4.0. *ETP SG*, page 72, 2016. URL: http://www.smartgrids.eu/documents/

ETP{%}20SG{%}20Digital{%}20Energy{%}20System{%}204.0{%}202016.pdf.

[24] Electric Power Research Institute. Simulation Tool – OpenDSS. URL: http://smartgrid.epri.com/SimulationTool.aspx.

[25] European Technology and Innovation Platform (ETIP). National and Regional Smart Grids initiatives in Europe. *European Technology and Innovation Platform (ETIP)*, 2(Second Edition):52, 2016. URL: http://www.smartgrids.eu/documents/ETP{%}20SG{%}20National{%}20Platforms{%}20Catalogue{%}202016{%}20edition.pdf.

[26] European Technology Platform. Smart Grids European Technology Platform. URL: http://www.smartgrids.eu/.

[27] M. Ferdowsi, A. Lowen, P. McKeever, A. Monti, F. Ponci, and A. Benigni. New monitoring approach for distribution systems. In *2014 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*, pages 1506–1511. IEEE, 5 2014. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6860997, doi:10.1109/I2MTC.2014.6860997.

[28] T. Finin. The KQML Language. URL: http://www.csee.umbc.edu/csee/research/kqml/papers/desiderata-acl/section3.4.html.

[29] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. KQML as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management - CIKM '94*, pages 456–463, New York, New York, USA, 1994. ACM Press. URL: http://portal.acm.org/citation.cfm?doid=191246.191322, doi:10.1145/191246.191322.

[30] FIPA. FIPA Agent Communication specifications, 2016. URL: http://www.fipa.org/repository/aclspecs.html.

[31] FIPA TC Agent Management. FIPA ACL Message Representation in XML Specification, 2002. URL: http://www.fipa.org/specs/fipa00071/SC00071E.html.

[32] project graphstream. Dijkstra's Shortest Path Algorithm. URL: http://graphstream-project.org/doc/Algorithms/Shortest-path/Dijkstra/1.0/.

[33] Guozheng Han, Bingyin Xu, and Jiale Suonan. IEC 61850-Based Feeder Terminal Unit Modeling and Mapping to IEC 60870-5-104. *IEEE Transactions on Power Delivery*, 27(4):2046–2053, 10 2012. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6293924, doi:10.1109/TPWRD.2012.2209685.

[34] Barbara Hayes-Roth and Frederick Hayes-Roth. A Cognitive Model of Planning*. *Cognitive Science*, 3(4):275–310, 10 1979. URL: http://doi.wiley.com/10.1207/s15516709cog0304{_}1, doi:10.1207/s15516709cog0304{_}1.

[35] Peter G. Hinman. *Fundamentals of mathematical logic*. A.K. Peters, 2005.

[36] Wilfrid. Hodges. *A shorter model theory*. Cambridge University Press, 1997.

[37] Ian Sommerville. Centralized Control, 2008. URL: https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/Architecture/ArchPatterns/CentralControl.html.

[38] Diego Issicaba, João Abel Peças Lopes, Mauro Augusto da Rosa, and Senior Researcher. Block-Oriented Agent-Based Architecture to Support the Power Distribution System Operation System Design and Environment Model. 2013.

[39] J. F. Hübner, O. Boissier, and A. Ricci R. Kitio. Instrumenting multi-agent organisations with organisational artifacts and agents: "giving the organisational power back to the agents. *Journal of Autonomous Agents and Multi-Agent Systems*, 20(3):369–400, 2010.

[40] M.A. Kashem, V. Ganapathy, and G.B. Jasmon. Network reconfiguration for load balancing in distribution networks. *IEE Proceedings - Generation, Transmission and Distribution*, 146(6):563, 1999. URL: http://digital-library.theiet.org/content/journals/10.1049/ip-gtd{_}19990694, doi:10.1049/ip-gtd:19990694.

[41] Abdollah Kavousi-Fard, Taher Niknam, and Mahmud Fotuhi-Firuzabad. A Novel Stochastic Framework Based on Cloud Theory and-Modified Bat Algorithm to Solve the Distribution Feeder Reconfiguration. *IEEE Transactions on Smart Grid*, 7(2):1–1, 2015. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7123178, doi:10.1109/TSG.2015.2434844.

[42] Johan Kensby and Rasmus Olsson. Building Automation Systems Design Guidelines for Systems with Complex Requirements.

[43] Edwin Bernard Kurtz, Thomas M. Shoemaker, and James E. (Engineer) Mack. *The lineman's and cableman's handbook*. McGraw-Hill, New York, 9th edition, 1997.

[44] Matheus Lopes. M. IEEE123testfederForAR, 2016. URL: https://github.com/matheusmlopess/matheusmlopess/blob/master/IEEE123testfederForAR.

[45] Ahmad Reza Malekpour, Taher Niknam, Anil Pahwa, and Abdollah Kavousi Fard. Multi objective Stochastic Distribution Feeder Reconfiguration in Systems With Wind Power Generators and Fuel Cells Using the Point Estimate Method. *IEEE Transactions on Power Systems*, 28(2):1492, 5 2013. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6338327, doi:10.1109.2012.2218261.

[46] Stephen D. J. McArthur, Euan M. Davidson, Victoria M. Catterson, Aris L. Dimeas, Nikos D. Hatziargyriou, Ferdinanda Ponci, and Toshihisa Funabashi. Multi-Agent Systems for Power Engineering Applications—Part I: Concepts, Approaches, and Technical Challenges. *IEEE Transactions on Power Systems*, 22(4):1743–1752, 11 2007. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4349106, doi:10.1109/TPWRS.2007.908471.

[47] Stephen D. J. McArthur, Euan M. Davidson, Victoria M. Catterson, Aris L. Dimeas, Nikos D. Hatziargyriou, Ferdinanda Ponci, and Toshihisa Funabashi. Multi-Agent Systems for Power Engineering Applications—Part II: Technologies, Standards, and Tools for Building Multi-agent Systems. *IEEE Transactions on Power Systems*, 22(4):1753–1759, 11 2007. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4349107, doi:10.1109/TPWRS.2007.908472.

[48] Hugh J. McCann and M. E. Bratman. Intention, Plans, and Practical Reason. *Noûs*, 25(2):230, 4 1991. URL: http://www.jstor.org/stable/2215590?origin=crossref, doi:10.2307/2215590.

[49] Elliott. Mendelson. *Introduction to mathematical logic*. CRC Press, 2010.

[50] Alexandre Mendes, Natashia Boland, Patrick Guiney, and Carlos Riveros. Switch and Tap-Changer Reconfiguration of Distribution Networks Using Evolutionary Algorithms. *IEEE Transactions on Power Systems*, 28(1):85–92, 2 2013. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6203628, doi:10.1109/TPWRS.2012.2194516.

[51] Jefferson Morais, Yomara Pires, Claudomir Cardoso, and Aldebaro Klautau. An Overview of Data Mining Techniques Applied to Power Systems.

[52] Moxa Newsletter. Power Automation. URL: http://www.moxa.com/Event/Sys/2009/IEC{_}61850-3/Application.htm.

[53] Muaz Niazi and Amir Hussain. Agent-based computing from multi-agent systems to agent-based models: a visual survey. *Scientometrics*, 89(2):479–499, 11 2011. URL: http://link.springer.com/10.1007/s11192-011-0468-9, doi:10.1007/s11192-011-0468-9.

[54] James. Northcote-Green and Robert Wilson. *Control and automation of electrical power distribution systems*. Taylor & Francis, 2007.

[55] Ricardo Pastor, Dag Eirik Nordgård, and et. al. Progress and Challenges on Asset Management for Future Smart Grids WORKING GROUP 1: NETWORK OPERATION AND ASSETS 2016. *ETP SG*, page 16, 2016. URL: http://www.smartgrids.eu/documents/ETP{%}20SG{_}Asset{%}20Management{_}White{%}20Paper{_}2016.pdf.

[56] Luciano L. Pfitscher, Daniel P. Bernardon, Luciane N. Canha, Vinicius F. Montagner, Lorenzo Comasseto, and Maicon S. Ramos. Studies on parallelism of feeders for automatic reconfiguration of distribution networks. In *2012 47th International Universities Power Engineering Conference (UPEC)*, pages 1–5. IEEE, 9 2012. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6398413, doi:10.1109/UPEC.2012.6398413.

[57] K. Prasad, R. Ranjan, N.C. Sahoo, and A. Chaturvedi. Optimal Reconfiguration of Radial Distribution Systems Using a Fuzzy Mutated Genetic Algorithm. *IEEE Transactions on Power Delivery*, 20(2):1211–1213, 4 2005. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1413376, doi:10.1109/TPWRD.2005.844245.

[58] A. Santi Ricci and M. Piunti. Action and perception in multi-agent programming languages: From exogenous to endogenous environments. In *In Proceedings of International Workshop on Programming Multi-Agent Systems (ProMAS-8)*, 2010.

[59] Richard C. Dorf. *The Electrical Engineering Handbook,Second Edition*. CRC Press LLC, 2 edition, 1997. URL: https://books.google.com.br/books?id=qP7HvuakLgEC{&}redir{_}esc=y.

[60] David D P E Roybal. Primary and Secondary Electrical Distribution Systems Eaton Electrical Cutler-Hammer Products. 2006.

[61] Daniel M. Russell. Planning and understanding: A computational approach to human reasoning. *Artificial Intelligence*, 23(2):239–242, 7 1984. URL: http://linkinghub.elsevier.com/retrieve/pii/0004370284900110, doi:10.1016/0004-3702(84)90011-0.

[62] Stuart J. (Stuart Jonathan) Russell, Peter Norvig, and John. Canny. *Artificial intelligence : a modern approach*. Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 2003. URL: http://aima.cs.berkeley.edu/.

[63] Tomaas. Salamon. *Design of agent-based models : developing computer simulations for a better understanding of social processes*. Tomas Bruckner, Repin, 2011. URL: http://www.designofagentbasedmodels.info/.

[64] Neil. Sclater and John E. Traister. *Handbook of electrical design details*. McGraw-Hill, 2003.

[65] Jason Sexauer and Opendss User. New User Primer The Open Distribution System Simulator (OpenDSS). 2012.

[66] R. O. Sinnott, D. W. Chadwick, T. Doherty, D. Martin, A. Stell, G. Stewart, L. Su, and J. Watt. Advanced Security for Virtual Organizations: The Pros and Cons of Centralized vs Decentralized Security Models. In *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 106–113. IEEE, 5 2008. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4534208, doi:10.1109/CCGRID.2008.67.

[67] Ben Lithgow Smith, Valentina Tamma, and Michael Wooldridge. AN ONTOLOGY FOR COORDINATION. doi:10.1080/08839514.2011.553376.

[68] D.M. Staszesky, D. Craig, and C. Befus. Advanced feeder automation is here. *IEEE Power and Energy Magazine*, 3(5):56–63, 9 2005. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1507027, doi:10.1109/MPAE.2005.1507027.

[69] Michael Stevens. *Programming paradigms and an overview of C - COMP3610 - Principles of Programming Languages: Object-Oriented programming*. Australian National University, 2011. URL: http://cs.anu.edu.au/.

[70] Ramadoni Syahputra, Imam Robandi, and Mochamad Ashari. Reconfiguration of distribution network with DG using fuzzy multi-objective method. In *2012 International Conference on Innovation Management and Technology Research*, pages 316–321. IEEE, 5 2012. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6236410, doi:10.1109/ICIMTR.2012.6236410.

[71] Joshua A. Taylor and Franz S. Hover. Convex Models of Distribution System Reconfiguration. *IEEE Transactions on Power Systems*, 27(3):1407–1413, 8 2012. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6153415, doi:10.1109/TPWRS.2012.2184307.

[72] graphstream Team. Graphstream, 2016. URL: http://graphstream-project.org/.

[73] The Eclipse Foundation. Eclipse Technology. URL: https://eclipse.org/.

[74] T. S. Ustun, C. Ozansoy, and A. Zayegh. Extending IEC 61850-7-420 for distributed generators with fault current limiters. In *2011 IEEE PES Innovative Smart Grid Technologies*, pages 1–8. IEEE, 11 2011. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6167079, doi:10.1109/ISGT-Asia.2011.6167079.

[75] T.S. Ustun, C. Ozansoy, and A. Zayegh. Distributed Energy Resources (DER) object modeling with IEC 61850–7–420, 2011.

[76] Valeriy Vyatkin, Gulnara Zhabelova, Neil Higgins, Karlheinz Schwarz, and Nirmal-Kumar C Nair. Towards intelligent Smart Grid devices with IEC 61850 Interoperability and IEC 61499 open control architecture. In *IEEE PES T&D 2010*, pages 1–8.

IEEE, 2010. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5484272, doi:10.1109/TDC.2010.5484272.

[77] J. Wang, Pedro M. S. Carvalho, and J. Kirtley. Emergency reconfiguration and distribution system planning under the Single-Contingency Policy. In *2012 IEEE PES Innovative Smart Grid Technologies (ISGT)*, pages 1–5. IEEE, 1 2012. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6175678, doi:10.1109/ISGT.2012.6175678.

[78] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, 10 2006. URL: http://link.springer.com/10.1007/s10458-006-0012-0, doi:10.1007/s10458-006-0012-0.

[79] Michael J. Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[80] Felix F Wu, Khosrow Moslehi, and Anjan Bose. Power System Control Centers: Past, Present, and Future. doi:10.1109/JPROC.2005.857499.

[81] Lin Zhu, Dongyuan Shi, and Xianzhong Duan. Standard Function Blocks for Flexible IED in IEC 61850-Based Substation Automation. *IEEE Transactions on Power Delivery*, 26(2):1101–1110, 4 2011. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5674123, doi:10.1109/TPWRD.2010.2091154.