SIMULATING MULTI-AGENT DECISION MAKING FOR A SELF HEALING SMART GRID

A Dissertation
Submitted to the Graduate Faculty
Of the
North Dakota State University
Of Agriculture and Applied Science

By

Steve Martin Bou ghosn

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:

Computer Science

April 2013

Fargo, North Dakota

North Dakota State University
Graduate School

Title

Simulating Multi-Agent Decision Making for a Self Healing Smart Grid

By

Steve Bou ghosn

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

DOCTOR OF PHILOSOPHY

SUPERVISORY COMMITTEE:

Dr. Kendall Nygard
Chair

Dr. Anne Denton

Dr. Jun Kong

Dr. Rodney Traub

Approved:

04/26/2013 — Date

Dr. Brian Slator — Department Chair

ABSTRACT

Dynamic real-time power systems like the national power grid operate in continuously changing environments such as adverse weather conditions, power line malfunctions, device failures, etc. These disruptions can lead to different fault conditions in the power system, ranging from a local outage to a cascading failure of global proportions. It is vital to be able to guarantee that all consumers with critical loads won't be seriously affected when these outages occur, and to also be able to detect potential faults early on, to prevent them from spreading and creating a generalized failure. In order to achieve this, the power grid must be able to perform intelligent behavior to adapt to ever changing conditions and also to self-heal itself in the event that a fault condition occurs.

The Smart Grid must continuously monitor its own status and if an abnormal state is detected, it must automatically perform corrective actions to restore the grid to a healthy state. Due to the large scale and complexity of the Smart Grid, anticipating all possible scenarios that lead to performance lapses is difficult [2]. There is a high degree of uncertainty in accurately estimating the impact of disruptions on the reliability, availability and efficiency of the power delivery system. The use of simulation models can promote trust in Smart Grid solutions in safe and cost effective ways.

In this work, we first present an innovative framework that can be used as a design basis when implementing agent based simulations of the smart grid. The framework is based on two primary concepts. First, the electrical grid system is separated into semi-autonomous units or micro-grids, each with their own set of hierarchically organized agents. Second, models for

automating decision-making in the grid during crisis situations are independently supported, allowing simulations that can test how agents respond to the various scenarios that can occur in the smart grid using different decision models. Advantages of this framework are scalability, modularity, coordinated local and global decision making, and the ability to easily implement and test a large variety of decision models.

ACKNOWLEDGEMENTS

I would like to express my deep appreciation and gratitude to my advisor, Dr. Kendall Nygard, for all his guidance and support through all these years. I also would like to thank all the people from the Smart Grid research group that was hosted by Dr. Nygard, all the discussions and collaborations with them had a meaningful influence on this work.

I would specially like to thank Ryan MCulloch and Davin Loegering, their contributions to this work are especially noteworthy. Ryan influenced this work greatly through all the discussions we had in our research meetings, including the design and implementation of the framework. Davin consistently collaborated with me and contributed greatly to the design and development of the simulator. Without their collaborations and support I'm not sure I would have been able to achieve these goals.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

In this chapter we first present some background information on the smart grid and the different disruptions that can happen in it that make supporting self healing capabilities vital. We then present some background on agents and multi agent system design as those concepts are used intensively throughout this work. Finally we explain the motivation for the creation of a simulation system to assess the self healing smart grid and we present some general guidelines that should be used for such simulation system, this serves as a prelude for chapter 2 where we implement our own simulation design based on those guidelines.

1.1. Vulnerabilities of the Current Power Grid and Advantages of a Self Healing Smart Grid

Dynamic real-time power systems like the national power grid operate in continuously changing environments and are exposed to different faults, like those created by natural calamity, device failures or operator failures. Once a fault occurs in a power system, it is necessary to isolate the faulted components from the rest of the network as soon as possible in order to minimize the detrimental effect of the local failure on the global system. A power failure can range in magnitude and impact from a relatively modest curtailment of power caused by a local power disruption to a catastrophic regional blackout caused by a violent storm or terrorist attack.

Previous experience shows that we are not well prepared or lack in adequate mechanisms to deal with these faults during emergency situations. For example, clinical operations were negatively affected when normal power was lost during the Houston floods of 2001, the northeastern United States blackout in 2003, and major hurricanes Charlie, Francis, Ivan and Jean in 2004 and Katrina and Rita in 2005. Power system control and management should be improved in order to detect those situations beforehand and avoid them, or the grid must be able to heal itself quickly when those events occur. In other words, the grid must support real-time situation awareness. Situation awareness (SA) is the capacity to make sensible decisions based on current environment conditions, to understand connections among people, places and events in order to anticipate their trajectories and act effectively.

Although there are variations in power system protection mechanisms throughout the world, the protection schemes have to detect faults and disconnect only faulted components from the network by opening or closing the appropriate circuit breakers. There are many problems that can occur in the power grid and disrupt its normal functioning, the most common types of problems that can occur are:

- **Power Outages:** Also known as power failures or blackouts, power outages are short or long term losses of power in a certain area. Causes include damaged power lines, or faults in any of the devices responsible for power transmission and distribution.

- **Voltage Instability**: Voltage instability refers to a condition in which a power system is unable to maintain voltages within acceptable ranges, particularly after a voltage collapse somewhere in the system. Assessment of voltage stability can be carried out with a variety of models, including neural networks, fuzzy logic systems, genetic search, and statistical methods.

- **Power Flow Unbalances**: Electric power generation and consumption must be kept in balance, since it generally is not practical to store electrical energy. If a component has a partial or complete failure and redirects its load to other components that are not ready or capable of receiving the load, an overload condition results. Subsequent redirection of the load can cascade into other components and cause further failures. Such chain reactions of failures can be very damaging and quickly disable large portions of the power system. This erroneous behavior is called a Cascading Failure, because the chain effect it generates resembles the expansion of a ripple in a pond.

- **Security:** The security of cyber and communication networks is essential for the reliable operation of the grid. The more heavily power systems rely on computerized communications and control, the more dependent system security becomes on protecting the integrity of associated information systems. A power system that is compromised or rendered inoperative due to security failures is helpless to serve its intended purposes.

> Possible attacks on communication networks serving the grid can come in many forms. For example, wireless components are subject to jamming or tampering at the physical layer; routing information can be manipulated at the network layer, and flooding can overwhelm a network at the transport layer. An intelligent adversary can devise many types of attacks to breach security in these types of communication networks. Most existing control systems, originally designed for use with proprietary, stand-alone communication networks, were indirectly connected to the Internet without added technologies to ensure their security [5].

A smarter grid should successfully solve each of the problems mentioned above, by providing the most effective and optimal solution. In the case of power outages, the current system consists of a few central power generators broadcasting into a large number of users using static links. The smart grid will replace this scheme by a dynamic power rerouting system, that is, it will be capable of dynamically routing and rerouting power to utilize optimal paths to deliver that power based on the current system state. Another problem in the current power grid is that if an electrical component fails and creates an outage, the detection of the failure will be very slow. Power companies might not become aware of the failure until a customer reports the outage. This slow detection and response results on many users being out of power for some time, and could have critical consequences in some cases. A similar situation caused the cascading power failure that blacked out 45 million people in the northeast region of the United States in 2003. In the smart grid, the smart devices and components will be able to monitor the environment, detect the failure, notify the power company and coordinate to use alternate

devices/resources and reroute the power properly in order to minimize the impact on the consumers, while the damaged components are repaired.

Concerning the problem of voltage instability, agents associated with smart controllers at generation sites can monitor voltage levels, run decision models to produce a strategy for dealing with the problem, and take action as needed to adjust voltage levels. Finally, in order to avoid the overloading of network components, smart agents and devices should coordinate to ensure that supply and demand for power are synchronized. This can prevent cascading failures in many cases by utilizing intelligent agents to adapt and coordinate to make sure the excess load is handled in a high-performance way.

1.2. Software Agents and Multi-Agent System Design

A software agent is an encapsulated software system situated in an environment where it can conduct flexible and autonomous actions to meet its design objectives [1]. The key characteristic of a software agent that differentiates it from other programming paradigms is that within a context, it persistently evaluates a suite of options that are available and chooses among them the most appropriate one and acts upon it.

Multi Agent System (MAS) are composed of multiple interacting intelligent agents. In general, an agent could be software agent, hardware agent, or human teams, but in this research, the focus is mainly on software agents. The agents are provided with different roles and capabilities and are distributed. Knowledge can be shared between agents in a MAS using different standard communication protocols, such as KQML, FIPA's Agent Communication Language (ACL). Examples of applications for which MAS is suited could be an on-line trading

system, Unmanned Air Vehicle (UAV) system, disaster response system, social structure, auction system, distributed network, etc.).

In the smart grid, large volumes of data are available at real time, moving data in such a time critical system is inefficient. A MAS can efficiently solve the problem due to its decentralized nature with local views or knowledge. The data are processed locally and only the results are communicated in the system by protocols, such as KQML and ACL. There haven't been many research works that have attempted the design, implementation and testing of a MAS for the smart grid.

In the Smart Grid framework, to process real time information from the environment and prevent, detect and handle faults, outages, and power flow issues, we require agents with differing roles that can adapt and cooperate to achieve global goals. Agents are defined for monitoring how users consume energy, the status of other agents and devices, brokering the completion of computational tasks among available resources, performing negotiations related to power demand/supply exchanges within a micro-grid, monitoring the power quality in the system to detect voltage irregularities, outages or power flow issues, among many other tasks.

A correspondence must be defined between the capabilities of a software agent from a programming perspective, and the roles that must be fulfilled in a responsive and adaptive Smart Grid framework. The agents in the Smart Grid are heterogeneous and cooperative. Heterogeneity arises from the various agents' capabilities: sensing widely differing things, activating diverse controllers, negotiating the completion of varied tasks, and finding acceptable solutions to alternative models. They are cooperative in that they all contribute to a larger overall objective, even if they have their own localized goals to achieve.

When working with agents in a large distributed system like the Smart Grid, it is advantageous to establish hierarchical relationships among the agents, where the agents on higher levels supervise those in the levels below. With this arrangement, an agent can either be entirely autonomous (acts on its own) or semi-autonomous (works cooperatively with other agents and under direction from a supervisor). A possible agent design could consist two hierarchical layers, with the agents in the second layer being supervised by the agents in the first layer. In cases where a supervisor is unavailable, a second layer agent acts autonomously. The first layer agents typically run on hosts with more processing power and memory than subordinate agents. This supports data fusion, complex calculations, and decision support models at the first level.

A satisfactory agent design model would require at least the following first level objects:

- **Device Agents**: These agents are responsible for controlling a particular device in the electric grid, for example circuit breakers, voltage regulators, transformers, relays, etc
- **User Agents**: These agents act as the interfaces between users and the system, monitoring the activities and intentions of users and communicating with other agents in the system on behalf of them.
- **DER Agents**: DER stands for distributed resource agent. These agents are only used within Micro-grids, if the supply coming from the main distribution circuit is not sufficient for the critical load of the users of that local area, then the user agents will start a negotiation process with the DER agents in order to compensate for the power shortage by purchasing power from those DERs.

- **Control Agents**: These agents are in charge of monitoring the system in order to prevent failures by detecting erroneous states that could lead to failures, or detecting the actual failures. They report all the information to their supervisor agents that decide the optimal strategy to follow. Some of the important system status info to be reported are operating frequency and voltage, generator synchronization status, quality of power level, devices not operating properly, etc
- **GUI Agents**: Are in charge of monitoring all activity in the system and creating a graphical visualization of all meaningful events in the system, in real time.

The second level objects come in one class and are named Management Agents, they are in charge of collecting all information from their subordinate agents and then using that data to make informed decisions regarding collaborative strategies that its subordinate agents should follow in order to accomplish a certain task, like dealing with a fault condition. This decision often involves the management agent allocating different roles to its subordinate agents in a way that optimizes the goal to be achieved. In order to do this the management agent bases its decision in some role allocation model, for example a linear approximation model like the one discussed in [6]. As an example of this, if the control agents detect power flow irregularities, once the management agent receives the information it could contact the device agents controlling some parts of the system and force them to shut down or disconnect the devices from the system, in order to avoid a cascading power failure.

1.3. Smart Grid Simulation Motivation and Design Guidelines

Due to the large scale and complexity of the Smart Grid, anticipating all possible scenarios that lead to performance lapses is difficult [2]. There is a high degree of uncertainty in accurately

estimating the impact of disruptions on the reliability, availability and efficiency of the power delivery system. These uncertainties result in hesitation on the part of decision makers in committing to smart systems for grid management. Most current research is focused on efforts to mitigate those uncertainties.

The use of simulation models can promote trust in Smart Grid solutions in safe and cost effective ways. The more accurately a software system can emulate the behavior and performance of an Smart Grid architecture, the better we will understand the advantages and possible shortcomings of a proposed infrastructure. Smart Grid simulations should adopt an incremental approach, starting with a small and manageable local subsystem with the intent of scaling gradually to be more comprehensive. One way to implement this kind of design approach, is to begin with simulating a local micro-grid, but with a scalable design that can grow hierarchically into a more complete model. The authors in [10] follow a similar approach.

To create an innovative and effective simulation system, it is important to have as main design goals the production of a software system that is flexible, adaptable and scalable. The intent is for the simulation environment to support multiple agents with alternative knowledge bases, running various decision models from multiple perspectives and levels of abstraction.

Key characteristics and design guidelines for the simulation are as follows:

- **Encapsulation of device models:** Device objects/classes that model working components in the grid are designed as communicating black boxes with well-defined interfaces. This allows the simulation to function consistently irrespective of the detailed implementation of the classes. An initial device implementations could be relatively simple, just supporting basic features, but as the system evolves, we should

be able to migrate to a new implementations using MATLAB/SIMULINK to accurately reproduce the electronics behavior of the devices for increased accuracy.

- **Loose coupling:** All agent nodes function as loosely coupled entities that interact only through the agent communication language. This communication approach carries computational overhead, but avoiding alternatives like shared memory communication provides great flexibility to accommodate any kind of component, ranging from a device like a relay or circuit to a full micro-grid.
- **Management of complexity**: The high degree of complexity of the Smart Grid is managed through the modularity offered by defining individual agents for the functions of the software. This provides a design that is flexible, adaptable, evolvable and easy to maintain and scale.
- **Cross-cutting Concerns:** From the viewpoint of software engineering, some requirements and properties cannot be modularized because they are inherently crosscutting concerns. These concerns resist modularization because they are inherently distributed and can have a negative effect on the degree of coupling in the software. The primary cross-cutting concerns in the Smart Grid are security, trust, authentication and policy enforcement.

To support these concerns, we insist upon consistent protocols throughout the system for identifying agents, safeguarding the knowledge bases of the agents, establishing trust relationships, and enforcing the two-level hierarchy of agent roles. If encapsulated into modules themselves, crosscutting concerns result in duplication of code across modules and create interdependences. This can result in a large number of updates to modules that are not directly related and create a negative effect

on the degree of coupling in the software. Cross-cutting concerns can be inimical to the goals of making the software easy to maintain, reuse, and evolve.

Aspect-oriented programming (AOP) is an approach to dealing with cross-cutting concerns by encapsulating them in modules called aspects. The code for a cross-cutting concern is called advice, is contained within the aspect, and is invoked at well-defined locations called join points. By using aspects in our design, we avoid most of the problems induced by cross-cutting concerns.

- **Knowledge Representation:** To support intelligent decision making, the agents must maintain a knowledge base. A base approach is to use the Belief-Desire-Intention (BDI) model for representing and utilizing knowledge. This separates the activity of choosing an intelligent plan from the actions that are required to execute a plan. Initially, the plans themselves are sequences of actions that are pre-specified and prescribed to accomplish self-healing in the grid. Other ontologies that use facts that an agent knows to be true and models relationships among facts and reasoning are expected to be added and included as needed.

- **Decision Support Models:** In many situations, agents must use decision support models to assess a current situation and determine a course of action to recommend. The Smart Grid is a vast heterogeneous system with a large number of nodes. Some communication channels may be disabled or unreliable. Robustness needs suggest that the decision-making be very distributed. Thus, the obtaining of complete information by an agent for decision-making purposes could incur delays, be very expensive, or even impossible. Thus, we turn to decision support models that incorporate uncertainty into decision making with partial information. A possible

suite of decision models that could be used includes fuzzy logic, integer linear program, and statistical hypothesis testing.

Fuzzy logic is a problem-solving control system methodology that has been successful in systems ranging from simple, small, embedded micro-controllers to large, networked, multi-channel data acquisition and control systems. It can be implemented in hardware, software, or a combination of both. Fuzzy logic deals with imprecision and incompleteness of information by formalizing the degree to which observations fit a classification. Decision trees can be built under fuzzy logic for decision making with partial information. This approach provides a straightforward way to arrive at definite conclusions based upon vague, ambiguous, imprecise, noisy, or missing input information.

The fuzzy decision support technique uses logic rules (e.g., IF A AND B THEN C) to make decisions. In applying fuzzy decision making for self-healing, a goal-seeking agent uses data and evidence of faults to populate the logic rules. Figure 1 shows a block diagram that illustrates the approach. The fuzzification step places data on factors like voltage levels, instability measures and known outages into ranges (e.g., low, medium, high).

The inference engine evaluates the decision rules as being true or false, resulting in the identification of a self-healing decision, such as rerouting power. As time passes and knowledge improves, the agents in the system can reason and make decision choices such as setting breakers, reducing loads, and rerouting power that seek to optimize their goals.

Integer linear programming is a closed-form optimization method that is capable of finding optimal allocations of resources. In our application, a method similar to that reported in [6] should be used to allocate combinations of tasks to agents that are capable of carrying

out specific actions to correct grid performance lapses. When tasks explicitly require collaboration, this model is capable of identifying the optimal suite of agent resources to be brought on a problem, including the roles that they should assume.

Statistical hypothesis testing provides a time-honored method of supporting decision-making in data-rich environments. For each identified problematic situation in the grid, we have a suite of alternative hypotheses available, each of which can be statistically tested as to whether the evidence supports the truth of the hypothesis. When a specific hypothesis is supported with a given level of confidence, an associated corrective action than agents can carry out is triggered.

Many other decision models can be incorporated. For example, we are considering Bayesian belief networks for situations in which it is possible to reliably estimate probabilities for alternative explanations of the data. Other possibilities include search heuristics or meta-heuristic methods such as the Distribution Tree Problem (DTP), Minimum Spanning Trees (MST), Simulated Annealing (SIMA) and Tabu search (TS) techniques Our simulation should be designed so that agents are independent from the decision model in use, to allow easy experimentation with alternative models.

The designed system should also be able to simply "call for help," enlisting support from distributed energy resources to help balance system needs. Agents that receive real-time price feeds and other data from the utilities, will have a basic set of knowledge-based rules on control decisions, and make the decisions that are executed and integrated in the environment to self heal the grid under uncertain conditions.

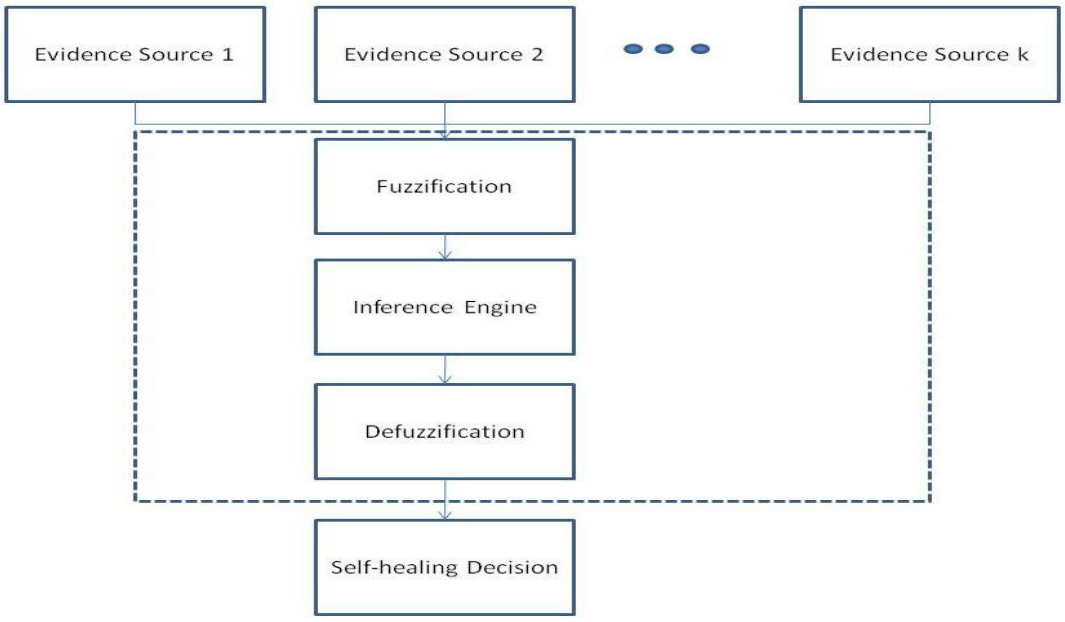

Figure 1: The Fuzzy Decision Making Process

Our main goals in this research work are first implementing a framework that addresses most of the shortcomings of traditional centralized and decentralized schemes by utilizing a hierarchical distributed control scheme. We also want the framework to supports direct comparisons among different decisions models, by implementing them in a separate independent layer from the agents.

To verify that the framework is feasible, we are implementing a simulator based on it. The simulator should successfully allow researchers to configure, test and compare different decision models for the various self-healing scenarios that can occur. The simulator should also provide an application programming interface that allows researchers to develop their own models and easily integrate them in the simulator for testing.

The rest of this work is organized as follows: Chapter 2 discusses the framework and also the design and implementation of the multi agent simulator system. Chapter 3 focuses on how the Decision Models work within the simulator and presents design guidelines for Decision Model developers. We also present the implementation of a default decision model and two others based on Tabu Search and Genetic Algorithms. On Chapter 4 we test and compare the different decision models presented in Chapter 3 on fairly large Macrogrid samples implemented using the simulator and running on Amazon EC2 cloud services. We also present detailed results for the tests performed. Finally, on Chapter 5 we present our conclusions and future work.

2. SIMULATION DESIGN AND IMPLEMENTATION

In this chapter we first present a discussion of related research work and discuss how our approach seeks to address some of the shortcomings of those works. We then present a detailed discussion of the framework and the design of a multi agent system based on the framework. We then present an actual implementation of a simulator based on the framework and discuss its main features and usage. Finally we present the different performance measures to be used for the comparison of different testing scenarios in the simulator.

2.1. Related Work and Main Objective

In the previous section different guidelines for an agent-oriented Smart Grid Simulation design were presented. The main conclusions for the design were that it should involve multiple interacting agents at supervisory and subservient levels, where each device, assessment, prediction, modeling, response, and decision-making procedure is associated with an agent. The agents are distributed and are capable of communicating and collaborating with each other and taking automatic actions in order to correct problematic conditions in the electrical grid. The simulation is designed with potential for scalability, to be able to gradually handle larger and more complex distributed systems.

Recently developed Smart Grid simulators and analysis tools include GridLAB-D and the Graphical Contingency Analysis (GCA). Both are projects developed at the U.S. Department of Energy's (DOE) Pacific Northwest National Laboratory (PNNL). GridLAB-D is a sophisticated simulator that provides detailed information of the power grid’s state, including power flow, end-use loads, and market functions and interactions. The GCA is a visual analytic software tool that aids power grid operators in making complex decisions. By using human friendly visualizations

and classifications of critical areas and by allowing the operators to simulate possible actions and their consequences, the tool helps human operators to analyze large amounts of data and make decisions in a reasonable amount of time. Due to the large amounts of data representing the grid status at any given time, even when aided by simulation and analysis tools, there are still limitations on how quickly human operators can make efficient decisions in near real time.

Because of the limitations of human operators in comparison with automated control, there is considerable research being done on how to fully automate control of the electrical grid by using software agents. There have been several attempts to create simulation systems for a smart grid using multi agent systems[7][8][9]. In one of the earliest works [10], the authors created an accurate hardware simulation of a simple Micro-grid using MATLAB and Simulink to model the functionality of low level electrical circuits.

Their agent implementation was very simple, with voltage monitoring to activate a circuit breaker and secure critical loads, with no complex decision making. Their focus was on showing that a Micro-grid can be managed as part of the global grid and is still able to work autonomously in an islanded mode, these autonomous subunits of the grid are also called an IDAPS (intelligent distributed autonomous system).

Their simulation of agent interaction and collaboration was not thoroughly tested, evaluated or analyzed. Their contribution was valuable in showing that the smart grid could be modularized into smaller independent units with their own agents and that those modules could work autonomously and as part of the whole.

In [11] a centralized multi-agent framework for power system restoration was designed. In [12] and [13], improvements were done to the framework while still maintaining a centralized design. Multi-Agent systems that utilize centralized control are typically not able to respond

quickly enough to perform global decisions and actions in near real time. Thus, such systems fall short of being able to address critical situations like a cascading failure that could have catastrophic consequences if not dealt with promptly.

In [14] and [15], new multi-agent frameworks for the power grid based on decentralized multi-agent systems are presented. The frameworks presented in [14] and [15] are decentralized multi-agent systems, but have the disadvantage of only allowing nodes to communicate with their neighbors. When nodes only acquire information from their neighbors, it greatly limits the quality of the decisions that the nodes can make, due to insufficient data. In [16] a hybrid multi agent framework that combined centralized and decentralized architectures was proposed. All of these approaches are topology dependent, with the exception of [15] and [16], which used a topology independent framework. By allowing the framework to work irrespective of the physical structure of the grid, a high level of flexibility and scalability can be obtained. Other agent-oriented Smart Grid designs are described in [18] and [19].

A major goal of our simulator is to support the appropriate decision models for the various self-healing scenarios that can occur. Therefore, we have based it on a topology independent framework that places the physical aspects of the power grid and the actual agents in separate independent layers. This addresses the previously unmet need to allow researchers to easily configure the type of decision model used by the agents, to compare how agents perform in the same case scenario when using different reasoning processes. More importantly, we also address the need for researchers to develop their own models and easily integrate them in the simulator for testing. Other important features are provided in our approach including innovations in supporting power grid topology, dynamic agent generation, and scalability.

The framework we present in this research work addresses many of the shortcomings of traditional centralized and decentralized schemes by utilizing a hierarchical distributed control scheme. It also supports direct comparisons among different decisions models, by implementing them in a separate independent layer from the agents.

There are a number of important questions regarding smart electrical grids that we hope to answer using the proposed simulator. What decision model would be most appropriate for handling self healing in particular situation in the smart grid? What decision model can guarantee that the reliability and efficiency is maximized for a given power system? How can decisions be made in a manner quick enough to avoid potential cascading failures while still deliberate enough to maintain the efficiency in the system overall? The previous smart grid simulation research work, despite their valuable contributions, cannot provide satisfactory answers to these questions.

Moreover, previous simulations haven't been designed with the flexibility to allow researchers to simulate different test cases. For example, most simulation efforts have been attached to a static topology. They cannot be used as versatile systems that allow researchers to design their own topology or decide on the mapping of intelligent agents upon that topology.

The simulation system that is implemented in this work is a flexible and fully scalable system. It allows researchers to create their own grid topologies and intelligent agent mappings through a convenient and easy to use graphical interface. The simulation can also be manipulated at run time through actions such as shutting down devices or forcing component failures. Finally, and perhaps most importantly, the type of decision model used by the different agents can be configured, and entirely new models can be plugged in and tested in the simulator. There is

considerable potential for the simulations described to help in building a smarter electrical Grid architecture.

2.2. Agent Model Design and Implementation

In this section the agent design used in the implementation of the simulation system is described. It consists of a three layered design, only the two upper layers of which are agent based. The bottom layer, the physical layer, is a simple hardware simulation. The main aim with this bottom most layer is to mimic the behavior of the electrical components themselves. In other words, this layer simulates the different components of the grid (relays, transformers, capacitors, power lines, consumers and generators) running on their own without any intervention or added intelligence. This separation between the intelligent agent layer and the physical simulation allows researchers to run two basic scenarios in the simulator, one, at the hardware level like the grid would normally operate, and another, using intelligent agent support, to compare the results.

It should be noted however that the physical layer implemented is not a perfectly accurate hardware simulation like the ones attempted in other research works like [10]. In this simulator we treat the physical components as essentially abstract black boxes. We are simply concerned about the basic power, voltage, and current metrics that are being inputted to and outputted from devices. Our focus is on the agent based upper layers, as our work is primarily concerned with how agents might solve fault situations, such as power outages, by assuming special roles and collaborating with one another. Knowing the specific details of the fault on an individual electrical component is not as important to us as the strategy to handle the fault itself and the decision models used to generate that strategy.

In the middle layer of our design structure are the supervised agents like distributed energy resources(DER) agents, User agents, Device agents, and Control Agents, these agents collaborate with each other to achieve global goals.

- **User Agents:** act on behalf of consumers, for example: if there is an outage they engage in a negotiation with DER agents to obtain a power lease to address that outage.
- **DER Agents:** act on behalf of generators that are independent from the main power distribution center. These generators are usually small companies or special consumers that also participate in the power market, like a small wind power generator or geothermal generator, etc. DER agents act on behalf of their generators by engaging in power supply negotiations with users.
- **Device Agents:** are in charge of performing operations on devices. For example: rerouting power by changing the distribution of power send through the lines of a relay, disabling a certain power line on that same relay, shutting down a transformer or closing a circuit breaker to secure some critical load or isolate some portion of the grid, etc.
- **Control Agents:** are in charge of constantly monitoring a section of the grid and collecting all data and information from all devices and power lines in that area. These results are sent in real time to a fusion point and aggregated there. These results after fusion represent the status of the system at a certain moment in time. That fusion point where all data is received and aggregated is in the management agent which will be addressed shortly.

In order for the agents in the middle layer to talk to the physical components in the first layer it is fundamental to have some middleware that facilitates the communication. This is achieved by encapsulating the bottom layer within an agent called the environment agent. The

environment agent contains all information about the grid topology and all device status information at any given time.

Middle layer agents have to query the environment agent to know what's going on in the physical grid. They must also send a write/modify request to the environment agent when they want to alter the behavior of the physical grid. This simulates (at an abstract level) how the agents would be integrated with smart meters and other sensors in a real grid.

The top most layer is the administrative layer and hosts the management agents, these are the agents that make the high level decisions. The control agents send all data representing the status of the system at a given time to the management agent. At which point, the latter organizes and analyzes that data, and attempts to detect situations in the grid that require self healing. If such situation is detected, it creates a strategy to heal the system. This strategy is expressed as a set of corrective behaviors that are basically a set of different roles to be performed by the middle layer agents.

Like the environment agent, the management agent has knowledge of the micro-grid topology. It knows, for example, how components are physically connected, but it does not know if a line has failed or its current voltage level value. That information is sent in real time to the management agent by the control agents.

The management agent is the main decision maker in our simulation. That said, the other agents in the middle layer still have certain autonomy to carry the high level decisions generated by the management agent. In other words they must make the lower level decisions needed to carry out the higher level actions. How decision models work and the different decision levels are explained in more details in Chapter 3.

In figure 3, we can see a basic graphic representation of the communication flow between these layers that illustrates the kind of intelligence that is integrated into each layer. The management agent is on top and on the bottom we have the basic physical components simulated in the environment agent with no intelligence. Then a middle layer that has some intelligence but not to the degree of the management agent.

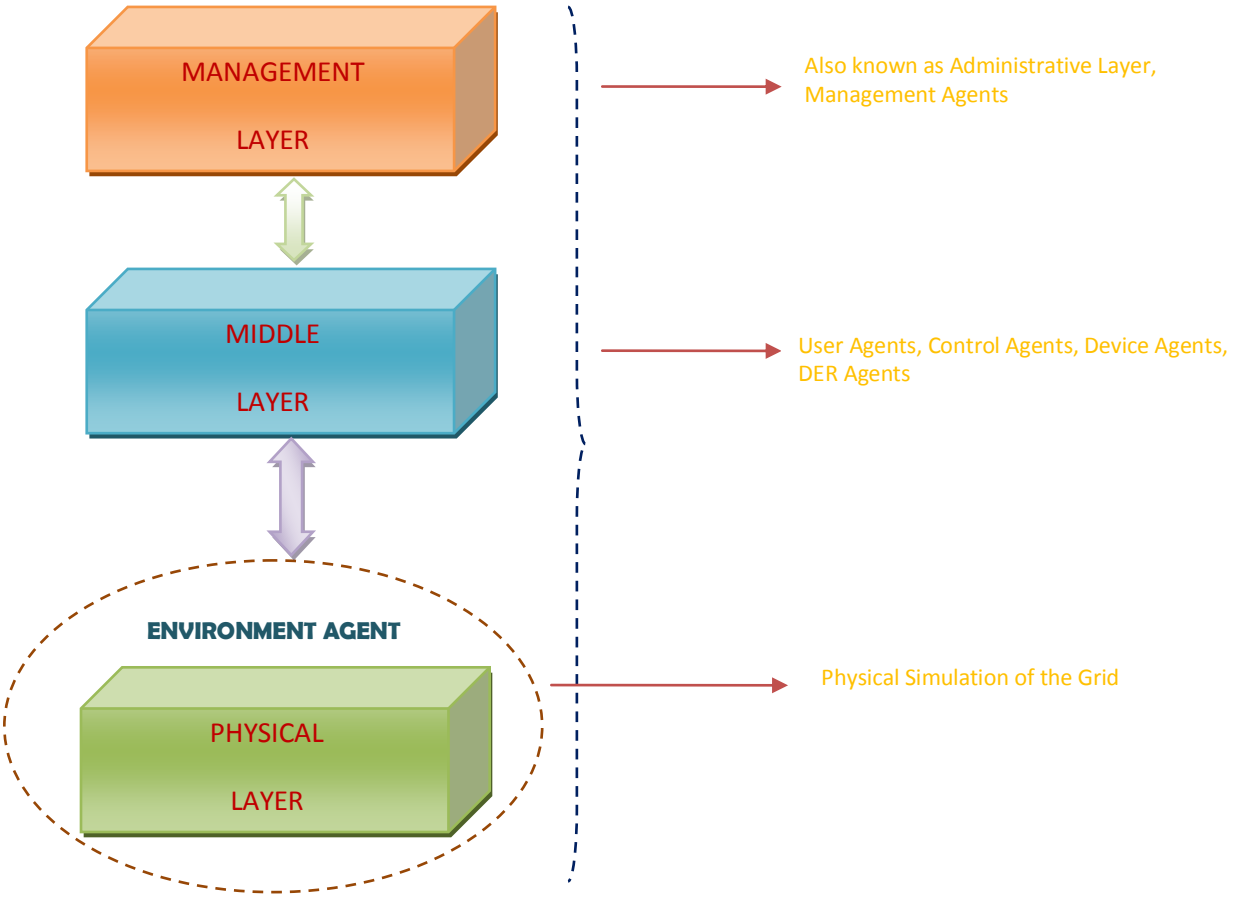

Figure 2: Multilayered MAS Design

The agent framework used to develop and run this MAS is the Java Agent Development Framework (JADE). The reason for choosing JADE is due to it being based on strong agent

communication standards like the IEEE standard on Foundation for Intelligent Physical Agents (FIPA), which can guarantee portability and interoperability between many different systems and platforms. Other reasons are its convenience and ease of use. JADE runs agents in containers, which are agent groupings, related agents usually run in the same container.

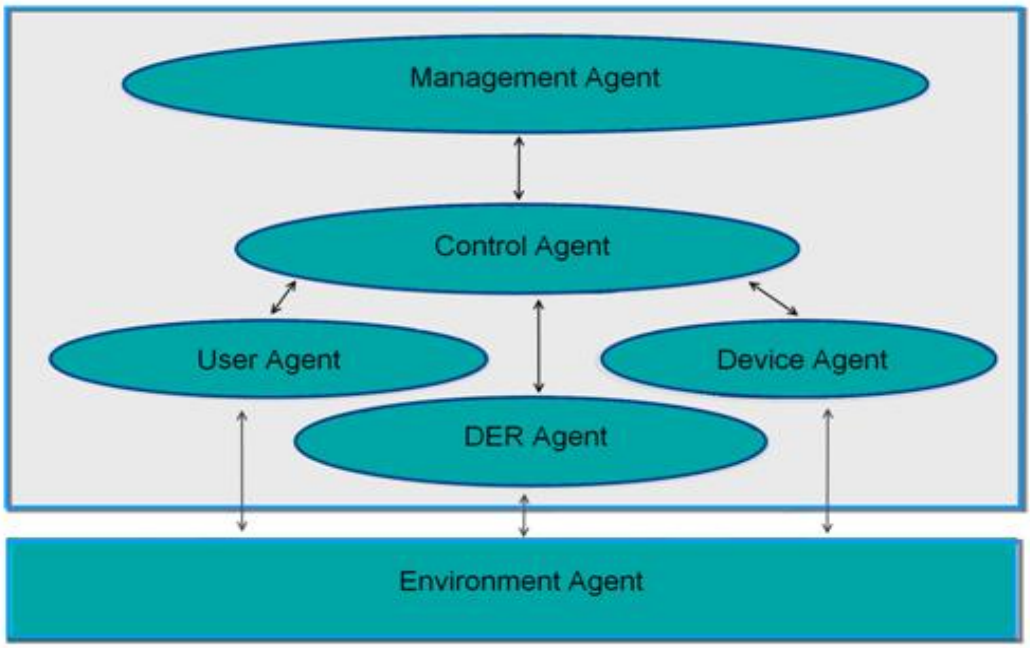

Figure 3: Flow of Communication Between Agents in the MAS

2.3. Simulation Framework

This section is divided as follows: In section 2.3.1 we discuss the Distributed Control aspects of the simulation framework, its advantages and also its possible challenges. In section 2.3.2 we discuss the Decision Model Independence aspect of the framework, and a discussion of its advantages and possible challenges also follows.

2.3.1. Distributed Control

To address the large size and complexity of the smart grid, we break the problems that were handled by a centralized controller into smaller problems handled by multiple distributed controllers. It is critical to utilize a granularity that allows the units to work independently and autonomously as well as to integrate and coordinate with each other in order to guarantee that the system works efficiently as a whole.

Our design is inspired by the concept of an Intelligent Autonomous Distributed Power System (IDAPS) that was proposed by the Advanced Research Institute of Virginia Tech., IDAPS is essentially a Micro-grid that contains sufficient intelligence and resources to be fully autonomous, yet functions within the global grid. We specifically design Micro-grids so that they are capable of disconnecting themselves from the rest of the grid under certain situations and work autonomously in islanded mode [10].The intelligence in the Micro-grid is handled by the multi-agent system associated with the Micro-grid and its quality depends directly on the multi-agent design employed.

The primary advantages of this framework using distributed control are modularity, scalability and effective local and global decision making

- **Modularity:** The complexity of large systems such as the smart grid can be managed with a divide and conquer approach. As much as possible, each autonomous unit is responsible for managing and solving local problems that pertain to the unit itself. When a critical situation occurs in the unit it is simple to isolate that unit from the global grid to prevent the crisis from propagating to other sections of the grid, as is often the case with cascading failures.

Modularity also allows the different agents within the Micro-grid to tune themselves and adapt to use the decision models and strategies that optimize resource usage and maximize efficiency for that particular Micro-grid, with its own topology, organization and operation.

- **Scalability:** Separation into well-defined independent autonomous units simplifies the process of testing at the unit level. Integration testing, referring to testing how the modules coordinate and integrate together, is also facilitated. The design supports the scalability of the system, because every Micro-grid enjoys autonomy in that all the subordinate agents in the Micro-grid report to their management agents and communication is done exclusively between elected management agents in each Micro-grid. This means that if we verify that a number of units work properly when tested independently and also work properly together when tested through integration testing, that is good evidence that adding more units would also scale well.

The main purpose of our design is to allow for easy scaling of the system. With our approach we can create a larger Macrogrid by connecting several Microgrids together. It is convenient for a user to create a simulation of a large grid by developing several different Microgrids in the simulator, followed by connecting them together.

Once a user has generated several Microgrids and established connections between them, the simulator should give him/her some flexibility on how to run those Microgrids. Many Microgrids could be set to run on the same computer or one Microgrid per computer. Therefore, the only limits to the scalability of the system are the resources available to the user. The simulator itself shouldn't have specific limits, since different

instances of it can be run to support each Microgrid, and the Microgrids if connected can communicate and interact with each other.

- **Local and Global Decision Making**: In designing a distributed control system, it is often very difficult to coordinate effective local and global decision making. For example, in distributed systems, if agents are only able to communicate with neighboring agents, there is a severe restriction on the quality and quantity of the data that can be collected. For local decisions this restriction of data is acceptable, for the decisions are correspondingly limited in scope, but for global decisions this data restriction can easily result in ineffective and potentially harmful decisions. However, centralizing decision making and forcing the nodes to report to and be controlled by a single entity is also problematic due to excessive communication requirements. Our design combines the two extreme approaches by allowing local Microgrid-based decisions to be handled locally, while still supporting certain global interactions.

Separating the extremely large and complex electrical grid system into small independent Microgrids facilitates effective local decision making. These decisions are handled by the management agents contained in every Microgrid. These agents constantly receive data reflecting the status of the local Microgrid in near real time, and therefore are enabled to make decisions that optimize the local performance of the Microgrid.

However, there are circumstances under which it is desirable for Microgrids to adjust their level of autonomy. This tuning of autonomy levels depends greatly on the

specific decision model in effect in the management agents. We describe this in more details in later sections.

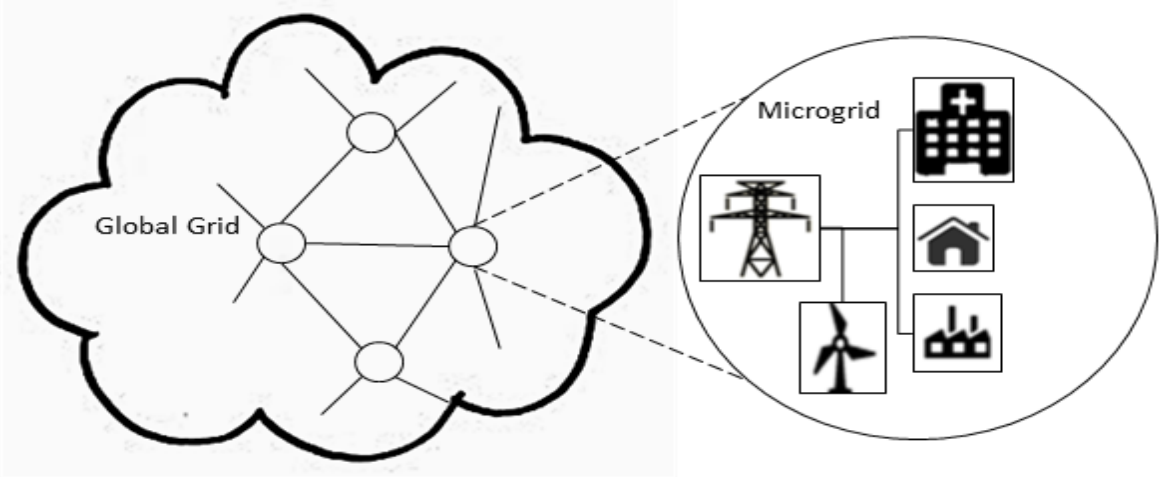

Figure 4: Local and Global Control Interactions

In the electrical grid, it is highly desirable to distribute decision-making. Our design allows each Micro-grid to essentially act as a single node in the larger grid. For example, If an individual Microgrid requires electrical power from outside, it can communicate and negotiate with its connected neighbors and establish a contract for that power. Under outage conditions, a Microgrid node can island itself from its neighbors to avoid propagation of the disruption.

A key advantage of a Smart Grid is the ability to access an open market for power with speed and agility [21]. This means that a Microgrid node that requires additional power can access the power market, select its preferred provider based on attributes such as price or location, and then negotiate an automated contract with the provider in near real time. This

ability can make significant advances in removing inefficiencies that are pervasive in the standard grid.

2.3.2. Decision Model Independence

We envision that the true promise of the Smart Grid lies in the development of multiple types of decision models that carry out their calculations automatically and trigger actions that are appropriate to the situation with little or no human intervention. The following types of decisions are candidates for automation:

- **Power Rerouting**: When devices or power lines fail, models that are equipped with details of the network topology and distribution costs and parameters can be charged with rerouting power along alternative pathways. The decisions must avoid exceeding the capacities of the available lines and devices, honor reliability requirements, and head off possibilities for cascading failures.

- **Resource Allocation**: When it is critical to rapidly access new or reserve power supply sources, the decisions must consider many available combinations and prioritize them in terms of their advantages and disadvantages. Cost, transmission distance and routing options, risks and reliability, and contract terms are all factors that must be included in parameterizing these models.

- **Dynamic Pricing**: When it is advantageous to shift power sources or limit power consumption at prescribed times to achieve cost savings, dynamic pricing models can negotiate and establish new power supply schedules at reduced cost.

An important new innovation of our simulation framework is that of supporting the decision models independently within the design. By placing the decision model agents on an independent layer, the monitoring and action-oriented agents can carry out their functions with no encumbrance from extensive special interactions and communications. This is accomplished through middleware that adheres to communication standards between the decision support agents and the others.

2.3.2.1. Decision Process

The management agents play a central role in the decision-making process. Management agents receive extensive near-real time data that characterizes the state of the system. The management agents are equipped with meta-level reasoning capabilities that determine if the current state of the grid is acceptable or in need of healing. In the latter case, it generates a strategy that is evaluated by calling upon decision model agents that carry out and return results aimed at corrective actions.

Multiple decision models have been designed and prototyped and are at some stage of maturity. These include the following:

- **Integer Linear Programming**: These models are capable of identifying optimal rerouting options and resource allocations. They are designed for decomposition so that local management agents can invoke only the portions of the global model that pertains to their Microgrid.

- **Fuzzy Logic**: These models are based on fuzzy set membership functions that capture degrees of fit with key resource allocation parameters, such as cost, distance, risk and reliability. The fuzzy sets drive a rule-based expert system that produces the suggested allocations. Although heuristic in nature, this type of decision model can function very quickly and easily in near-real time decision making.

- **Bayesian Belief Networks**: These models are based on probabilities associated with system states. These models are capable of polling the grid for additional information that form the basis for producing posterior probabilities with enhanced accuracy.

- **Market-driven Pricing Models**: These models work within a market economy in which energy resources are traded. This type of model includes dynamic pricing based on smart building and smart meter infrastructure, and provides an area of great promise in improving grid performance and efficiency.

In general, state variable data is made available via middleware in an Application Programming Interface (API). A key advantage of this approach is that researchers are free to readily test the models above as well as any other developed decision model. The model builder must convert data from the API to match the data types of the decision model. This method is analogous to the presentation layer in the OSI networking model. After a decision model is invoked, it has defined a set of actions and corrective behaviors that can ultimately be carried out by middle layer agents.

2.3.2.2. Adjustable Agent Autonomy

In some cases it is desirable for certain agents to be only semi-autonomous in that they make decisions only in a context controlled by a management agent. For example, a semi-autonomous user agent may have preferences and configurations that can be compromised by a decision model for the betterment of the Microgrid. Fully autonomous agents must adhere to their settings regardless of decision model recommendations. Semi-autonomous agents have a "wait" state in which they take actions to deal with a problem only upon a directive from a management agent.

Management agents can strategically choose to elevate the autonomy level of a semi-autonomous agent, based on the current decision model and system state. For example, a component of a management agent strategy to heal the system could be to promote some semi-autonomous agents to fully autonomous status either temporarily or permanently. The inherent flexibility of adjustable autonomy is a powerful capability that allows an agent-oriented system to respond to events that cannot be foreseen.

It is critical that an agent-based system for a large complex system such as the Smart Grid support only agents with a high level of trust, to alleviate suspicions expressed by people that agent decisions could go awry. Adjustable, situated autonomy increases the level of trust. Thus, we believe that adjustable autonomy is an important element of decision making in the Smart Grid.

The key advantage of Decision Model Independence is the ease in experimenting with and evaluating alternative decision models. Any single scenario can be evaluated with multiple models for side-by-side comparisons.

2.4. Simulation Implementation and Main Features

In this section we will discuss the main features of the simulation system, and we will explain the innovations that the simulator introduces when compared with previous smart grid simulation attempts. We will also give a tour of the simulator, its basic configuration and use and in doing so display the potential benefits it could bring to researchers testing self healing scenarios on a smart grid architecture.

Before describing the main features of the simulator we should briefly describe the modes in which it operates, as the different features apply to different modes. The simulation works in two distinct modes, design and run. In design mode, researchers configure the simulator with the details of their desired test scenario. While there are a variety of configuration options, at the very least a researcher will need to design or load a microgrid topology as well as determine the agent assignments to different components or sections of that microgrid. In this mode researchers can also save all that design information to a XML file for themselves or others to use later. This way others can verify results or run other tests on the exact same topology. In run mode, we load the test case created in design mode and actually run it in the simulator. This means that the simulation has actually started, the physical infrastructure is emulated and the agents are active, monitoring and collaborating.

The main features of the simulation system are: Physical Layer Independence, Power Grid Topology Design Flexibility, Full Scalability, Flexible Agent Mapping and Dynamic Agent Generation, Decision model Independence and Active Control of Testing Environment.

- **Physical Layer Independence:** As was described before, the physical hardware simulation is in a different layer than the agents, this allows us to achieve some independence between both. By separating them it allows us to run the simulator as a purely hardware simulation of the physical grid without any external intervention or intelligence. It is then possible to run the same configuration using agents and intelligence, and then compare to what degree the agents are improving the efficiency of the self-healing process of the power grid.
- **Power Grid Topology Design Flexibility:** One of the main goals for the simulator was to design it in such a way that it would be flexible enough to allow researchers to test any kind of grid topology. Because of that, we constructed a simple yet flexible representation of a microgrid as a set of nodes connected by arcs. The arcs represent power lines and the nodes can be consumers, generators or devices (like relays or transformers). This design allows researchers to generate a large variety of topologies.

  Topology design is done in design mode. In design mode, the simulator allows the design of a microgrid through the use of GUI tools. A toolbar along the left-hand side of the window allows the user to drag and drop a variety of nodes onto the main grid layout area and connect them with power lines. The nodes can be consumers, generators, or devices and components like relays or transformers. The simulator is designed flexibly enough that new types of components can be easily incorporated into the system if required. After a microgrid topology is designed, all the topology design information can be saved to an XML file for later use.

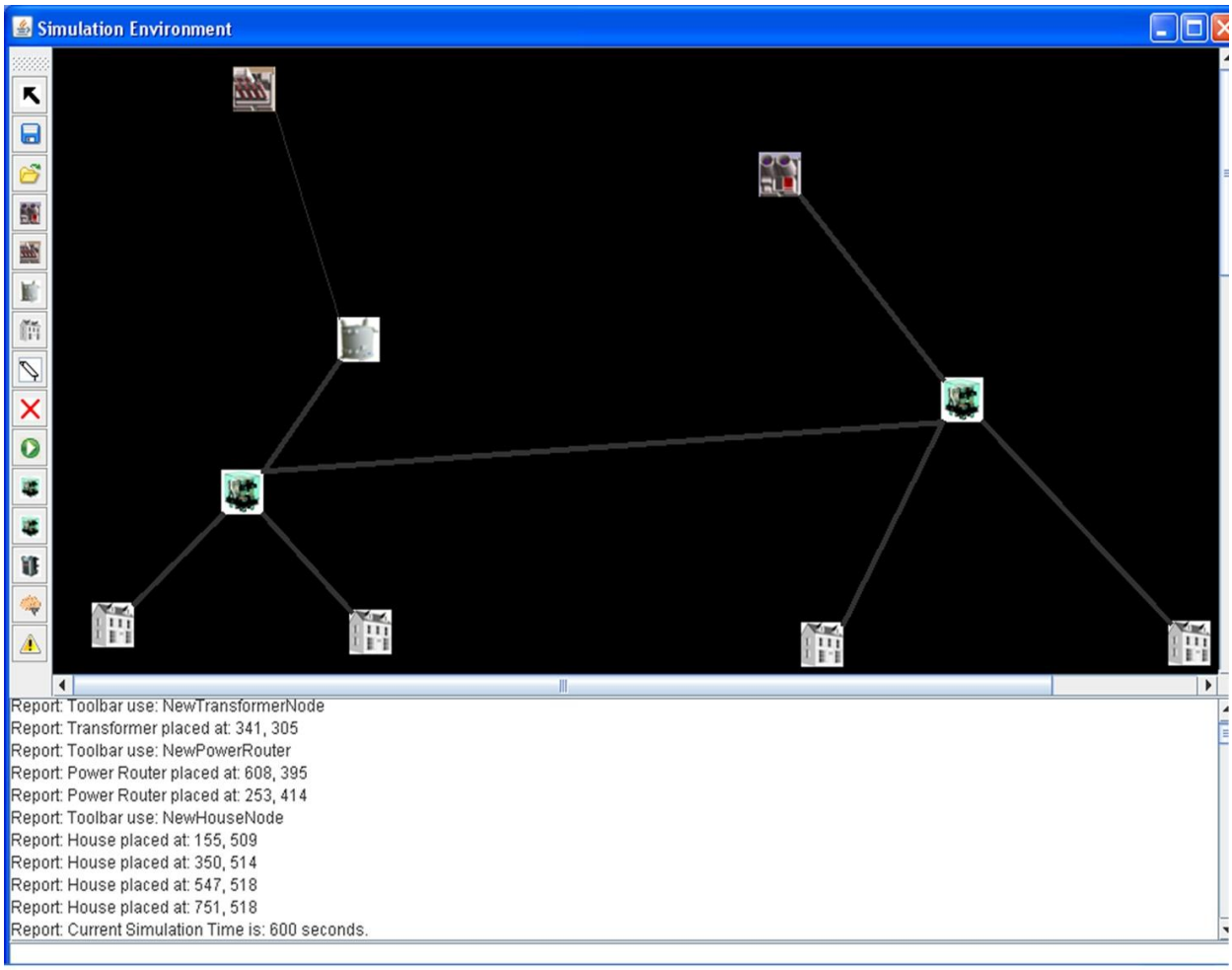

Figure 5: Simulation GUI Design Mode

- **Full Scalability:** The main reason for choosing to design at the microgrid level is to allow for easy scaling of the system. By using this model we can create a larger macrogrid by simply connecting several microgrids together. In other words, if a user wants to create a simulation of a larger grid, all that he/she needs to do is design several different microgrids in the simulator and then connect them together. A researcher can indicate in the microgrid he/she designs the external connection points, and then specify for each connection point what precise microgrid it is linking to. Each of the microgrids designed in the simulator run in an independent JADE container.

```
<Simulation.SimulationComponent>
  <position>
    <x>341</x>
    <y>305</y>
  </position>
  <inputPowerLines>
    <Simulation.PowerLine>
      <distance>155</distance>
      <maximumVoltage>2000.0</maximumVoltage>
      <maximumCurrent>2000.0</maximumCurrent>
      <maximumPower>2000.0</maximumPower>
      <startPosition>
        <x>294</x>
        <y>158</y>
      </startPosition>
      <endPosition>
        <x>342</x>
        <y>306</y>
      </endPosition>
      <name>line1</name>
      <idComponent1>Coal0</idComponent1>
      <idComponent2>Transformer2</idComponent2>
      <lineFailed>false</lineFailed>
      <lastPowerValueBeforeFailure>0.0</lastPowerValueBeforeFailure>
      <traceableFailed>false</traceableFailed>
      <trackInputPower>0.0</trackInputPower>
    </Simulation.PowerLine>
  </inputPowerLines>
  <type>DEVICE</type>
  <node class="Simulation.Transformer">
    <name>Transformer2</name>
    <numStates>2</numStates>
    <nodeFailed>false</nodeFailed>
    <queueVoltage class="linked-list"/>
    <queueCurrent class="linked-list"/>
    <queuePower class="linked-list"/>
    <GS/>
    <subtype>Transformer</subtype>
    <maximumVoltage>1000.0</maximumVoltage>
    <maximumCurrent>1000.0</maximumCurrent>
    <maximumPower>1000.0</maximumPower>
    <failureProbability>0.2</failureProbability>
    <deviceFailed>false</deviceFailed>
```

Figure 6: Section of XML File Storing Design Information

Once a user has generated several microgrids and established connections between them, he/she has some flexibility provided by the JADE platform and the simulation framework, on how to run those microgrids. Many microgrids could be set to run on the same computer or one microgrid per computer. Therefore, the only limits to the scalability of the system are the resources available to the user. The simulator itself has no specific limits, since different instances of it can be run to support each microgrid, and the microgrids if connected can communicate and interact with each other.

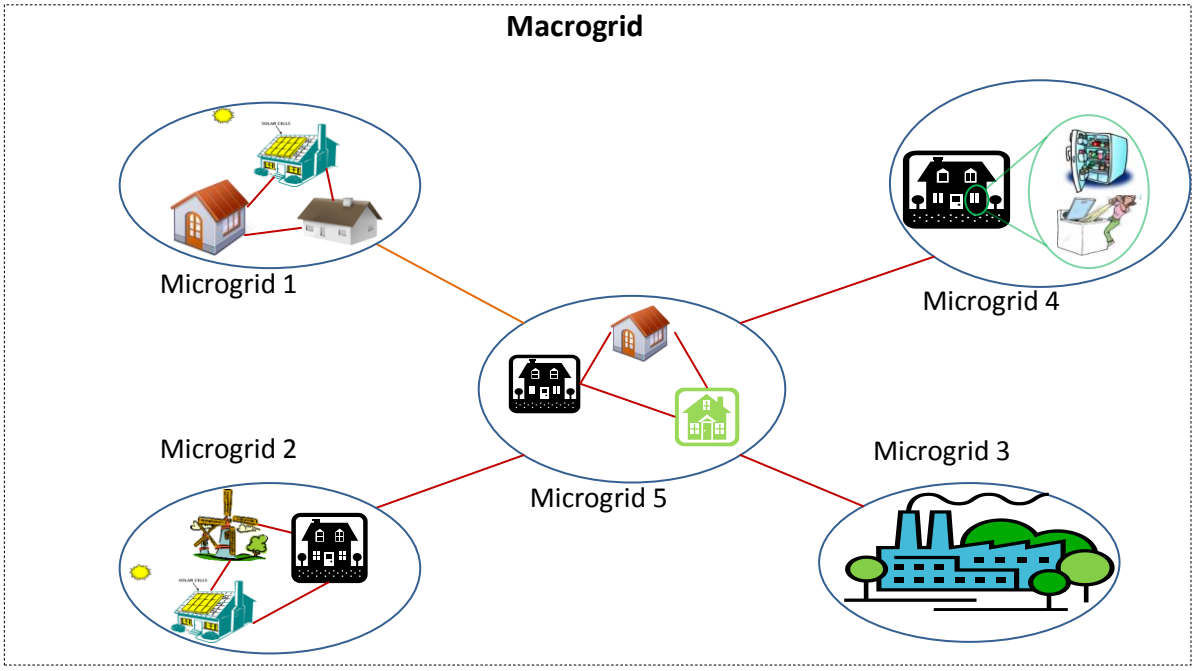

Figure 7: Macrogrid Generated by Connecting Different Microgrids

- **Flexible Agent Mapping and Dynamic Agent Generation:** After a user is done designing the topology of their microgrid and specifying design parameters, he/she can work on designing their desired agent support. Users can use the toolbar to decide where to place agents, if a user wants certain devices to be monitored/manipulated by agents all he/she has to do is select the agent tool in the toolbar and click on desired component to associate an agent with it. If a user wants an agent associated with a certain consumer or generator, the mapping can be done easily in the same way. This allows researchers to make a test while having an agent assigned to monitor a certain device in one run, and then retest the same scenario without having an agent monitoring that same device, to be able to see how that particular change affects the results.

  One of the main features of our simulation is that the agents are dynamically generated in the supporting agent platform (JADE). When the user shifts the simulation from design to run mode the system will start the simulation of the physical grid and will automatically launch the JADE framework dynamically creating the agents that the user mapped. The system automatically determines what kind of agent is required for each node. If the agent was placed on a consumer it knows it should create a user agent, on a generator a DER agent, on a device a device agent, etc. The agent-component mappings can also be saved to the XML file storing design information.

  In the case of a control agent, since it monitors a complete section or area in the grid instead of a single component, which components it monitors is determined by associating it to the nearest power line and then using a ***hopRange*** parameter that can be set by the user, to decide the range within which components are monitored by that

control agent. Only components within a ***hopRange*** number of hops from the original power line are monitored.

Every microgrid created in the simulator runs in a separate JADE container. Agents that the user specified in design mode are created dynamically and placed in the microgrid JADE container, the system automatically creates an environment agent for the microgrid and at least one management agent.

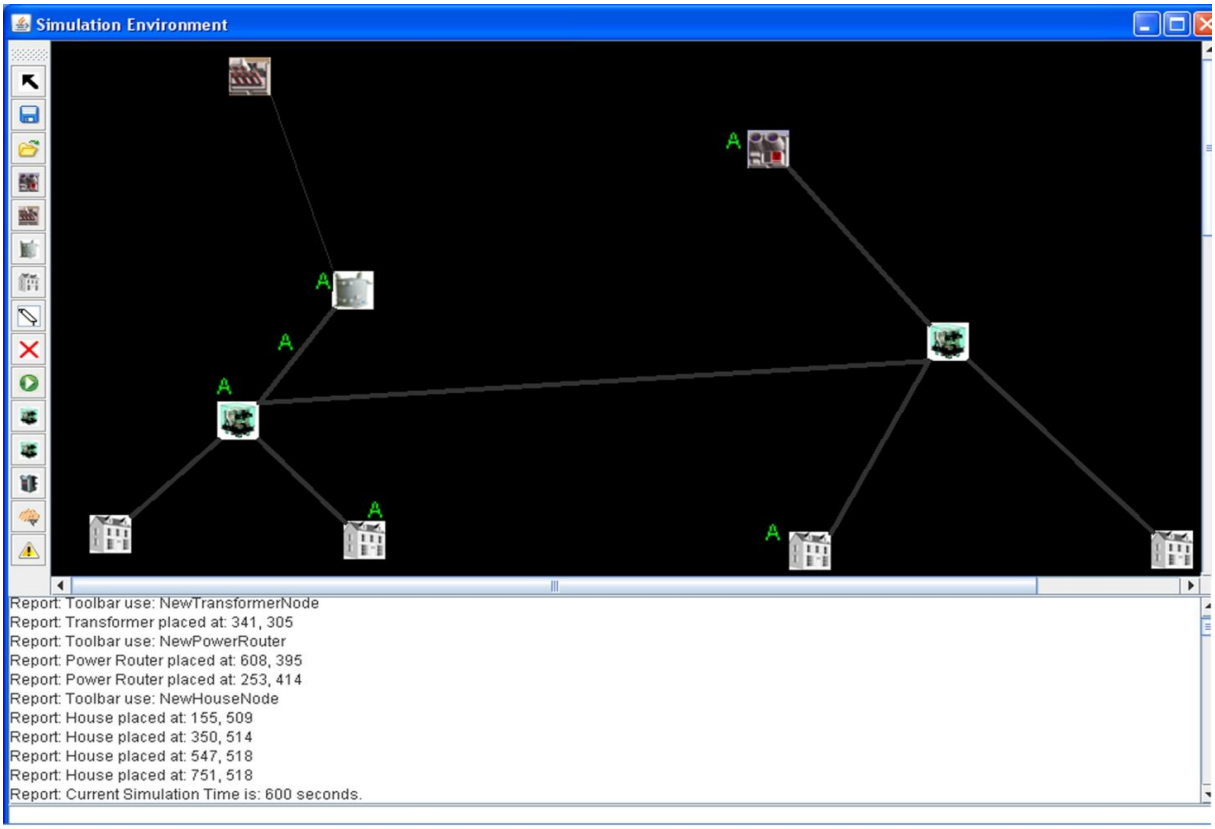

Figure 8: Design GUI Showing Agent Associations

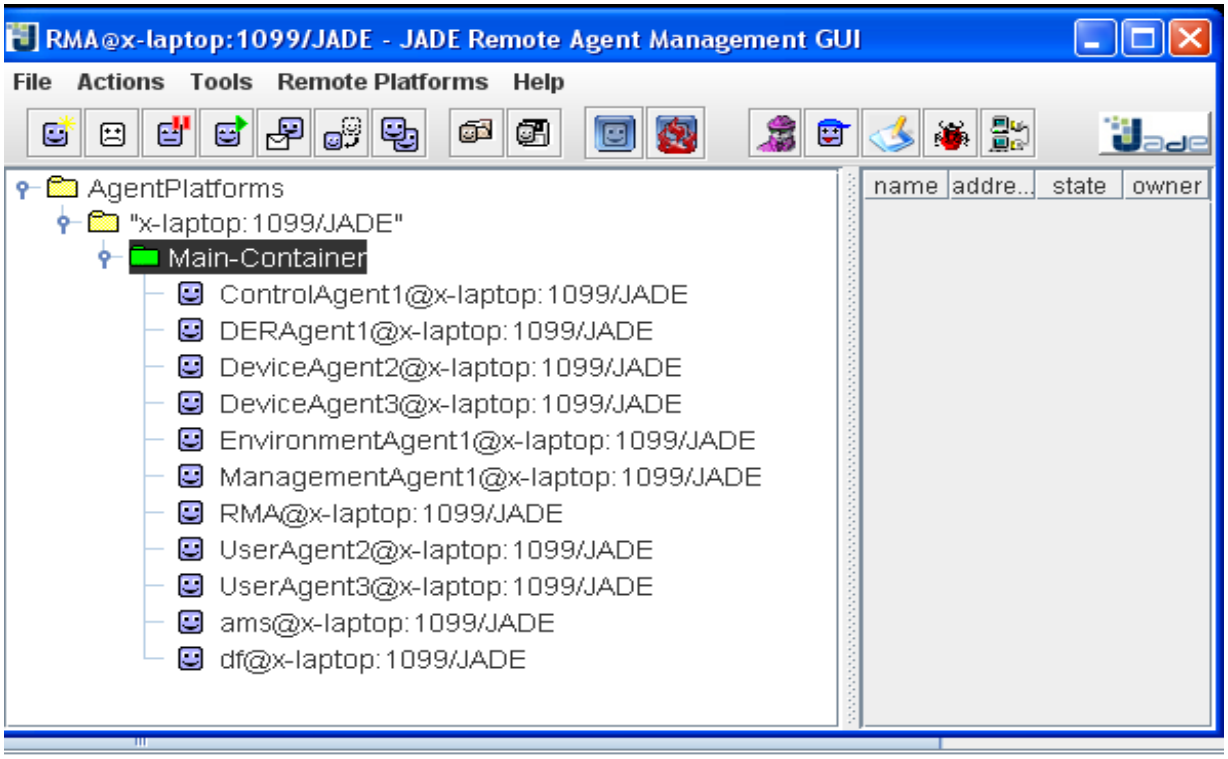

Figure 9: Jade Running with a Container Created for the Microgrid and the Agents Dynamically Generated

- **Decision Model Independence:** The most important innovation of this simulation system when compared to previous efforts is that of having the decision models independent of the rest of the design. In other words, any number of custom decision models can be configured in the simulator. For example a management agent could be configured to use ILP(Integer Linear Programming) as its decision model to test a particular scenario and evaluate the efficiency and performance of how agents respond when using that decision model, and then test the same situation by configuring a heuristic based decision model or one based on Naïve Bayesian, and then compare the results. Ultimately researchers can develop their own decision models and plug them into the system.

Standard interfaces will be available that will allow researchers to design their own models and smoothly plug them in, without a need of knowing the details of the internal implementation of the simulator. We have already implemented models based on Tabu Search, Fuzzy Logic and Genetic Algorithms. We have thoroughly tested the Tabu Search and GA models, results for the experimentation with these models will be presented later on Chapter 4. As part of our future research, we are planning to integrate the ILP decision model described in [4] into our simulator.

The decision models for lower level agents are also configurable. For example DER and user agents use a power negotiation model to establish power leases, that model is configurable, the default negotiation model used is very simple and based on the lowest price, but new more complex power negotiation models can be created and plugged in, to test them in the simulator.

- **Active Control of Testing Environment:** Our simulator allows users to both, configure some parameters and settings when designing a physical microgrid, and also change those parameters dynamically during run mode. Some examples of these parameters are the maximum power a given generator can produce, the power demands of consumers, or the capacities of power lines. Parameters can be set/modified at both design and run time. For example, in design mode users can set an initial power, voltage, or current that a component can be outputting when the simulation starts, but in run mode it is possible to modify and configure all voltage, power or current settings of a component in real time.

  Parameters set at design mode can be saved with the rest of the configuration. When in run mode, a user can also generate different failures in the grid, such as forcing

a power line failure or a relay to malfunction. These actions can be used to observe the resulting effect and in particular the self-healing operations taken by the decision model that is currently configured.

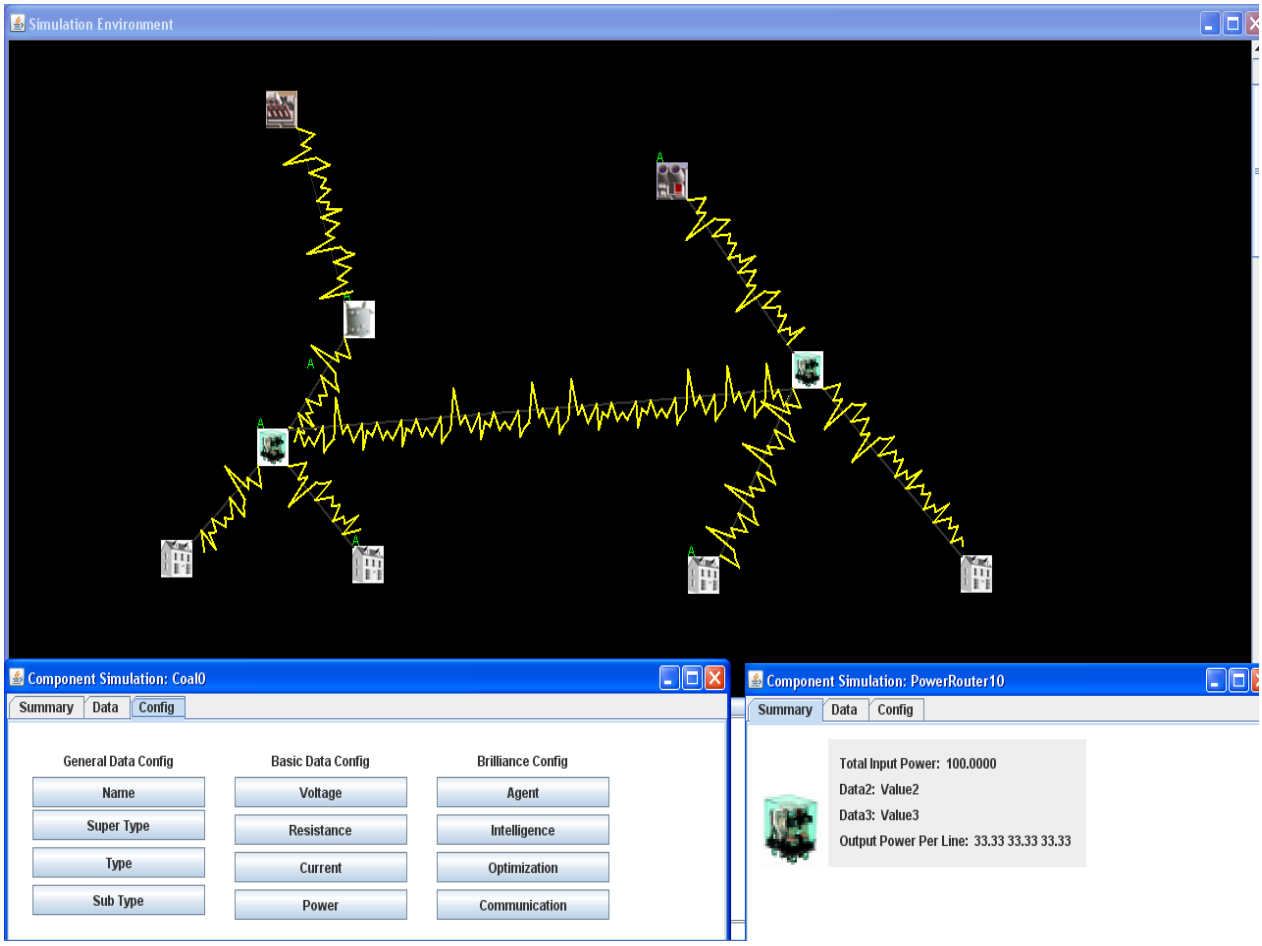

Figure 10: Simulation in Run Mode

2.5. Performance Measurements

In order to evaluate the performance of the simulator we need to establish a set of standard performance measurements. These measurements should allow us to evaluate the different scenarios and help us clearly identify the potential advantages and disadvantages from the viewpoint of the different stakeholders that might want to test/observe the smart grid behavior. It is important to evaluate for example, how efficient the system is in supplying power and preventing and minimizing outages, just as important as knowing the overall cost that the system is incurring to achieve that, irrespective of whether the level of outages is high or not. Having those criteria in mind, we have established the following main performance measures that are used to evaluate the performance of a given decision model when applied to a particular scenario in the grid:

- **Outage Time:** The total amount of time any consumers in the system experience power outages. By minimizing the time during which consumers experience unsatisfied demands we are not just guaranteeing the higher quality of the service for consumers, but we are also reflecting on the performance and efficiency of the grid. A decision model that effectively diminishes outage time for consumers is able to perform load/power rerouting in fast and efficient ways. It is also able to prevent and deal with device failures, since a potential device failure almost always will lead to extended power outages or in the worst case a more generalized failure like a cascading failure. This time is measured in simulation units; simulation units are flexible and can be fine tuned by the researcher performing the experiment. A simulation time unit could be an amount of seconds or milliseconds, in some special cases it could even be minutes.

- **Power Cost:** The overall cost of the system during a period of time measured in simulation units. When evaluating the cost of the system we include both the cost for generators to produce a certain amount and type of power, as well as the cost of outages experienced by consumers. On the generator side, different generators produce different types of power and have different expenses which translate into different cost levels. On the consumer side, the longer the time a given consumer experiences unserved demands, the more the cost. Also, we cannot treat all consumers the same, some consumers have higher priority than others and incur much higher expenses when experiencing an outage, a clear example is comparing a normal household with a hospital, they are both consumers and while a small outage wouldn't cost much to the household it could be extremely critical and expensive for the hospital. One could argue that the cost measurement indirectly reflects the time outage performance measure, since the cost of consumer outages is included in the overall cost amount. That cost is expressed individually as total outage time for individual customer multiplied by the cost per outage time unit for that same consumer. While it is true that this measurement includes the other to a certain degree, it still doesn't fully account for it, the power cost measurement directly reflects how decision models do resource mapping, but it is not an accurate way to know how efficiently a decision model performs power rerouting and device failure prevention/handling. There are some situations where researchers might not care about cost and intend to evaluate how a decision model handles power routing, performance and reliability in the grid; for those situations it would be much more convenient to do evaluations based on the time outage measure rather than basing it on the power cost

measure. Using the latter measure for those scenarios could give misleading results; a decision model could be hiding its poor performance strategies by its use of a remarkable resource allocation strategy.

- **Power Quality:** The overall quality of the power supply service in the power grid also expressed as the overall level of consumer satisfaction. Minimizing consumer power outages doesn't guarantee maximum consumer satisfaction per se, this is rather obtained when we supply those consumer outages while trying to comply as much as possible with consumer preferences. Some consumers have a price range that they are willing to pay for power, and are only willing to pay a higher price if a real emergency occurs, if we often force the consumer to pay a higher price for power than he initially compromised himself/herself to pay, that will directly affect that consumer satisfaction level. There are many criteria affecting consumer satisfaction, all of these combine together to generate a utility score that would reflect the overall quality of the system service.

  The main criteria for customer satisfaction are the following:

  - Price Limit Preference: If a consumer doesn't frequently have to pay a power price higher than he agreed to, he will be more prone to be satisfied.
  - Generator Type Preference: Many consumers have preferences for generator types, for example: Some consumers might be willing to pay a higher price to use green energy generators or any other environmentally friendly generators instead of those that could potentially damage the environment like coal or nuclear. In some situations the use of those generators can't be avoided and it is still preferred than having the consumer experience outages, but to the extent that we

supply consumers using the kind of generators they prefer, to that extent that consumer's satisfaction level will be higher.

- Generator Ranking Preference: Many consumers have a preference for generators that have proven to be reliable in the past. What use is for a consumer to have its outage being supplied by a particular generator which has a high probability of suspending that supply in the near future. That is why consumers are able to give a ranking to generators based on their performance and reliability. These rankings work just like sellers of other items/products are evaluated in cyber marketplaces like EBay or Amazon. Customers who have experienced a particular generators power services are allowed to rate the service for other consumers to see. Therefore consumers are allowed to set their preference to be supplied by generators that are above a certain level of quality rating. To the extent that we assign consumers with outages to generators that are trustworthy to them, to that same extent that consumer's satisfaction level will be higher.

All of these criteria and others are combined to create a total consumer satisfaction score by feeding them to a special utility function. The resulting score will be the power quality performance value of the system.

- **Variability:** To perform power routing or resource allocation in the system there are some changes or variations that need to occur in order to execute the required operations. For example in order for a certain generator to supply a certain consumer it is not enough that the generator has enough available power or that there is a valid physical path connecting both nodes, in order to route the power from the generator to the consumer, the system has to perform a series of changes like for example increasing the voltage,

power or current in the generator, and make sure that all relays within the path between the consumer and generator are set to the proper values, to guarantee the power is routed safely and efficiently.

All these performed changes bring a risk to the stability of the system, for example when altering the amount of power a relay outputs on each line to do power rerouting, since the device might have a maximum power load it can handle, the changes in the output rates could affect the power being transmitted through the relay to a different consumer, requiring further action to also restore that path. It's easy to see how the more changes that are made to the system, the more possibilities for these to cascade into further problems and the requirement for even further changes, not to mention that some of these changes could potentially be expensive for individual nodes in the real world. Therefore it is important to measure not only how efficient a particular decision model can perform power routing and resource allocation, but also to which extent it can decrease the overall system variability while achieving so.

- **System Failure Prevention Level:** This performance measures evaluates the degree to which the system operation minimizes the probability of occurrence of a generalized failure like a cascading failure. At any given time this depends directly on the loads being transmitted in the different power lines and their capacities, attributes and probabilities of failure. The same applies to all devices being used at the time, the characteristics and status of each device determine its probabilities to fail sometime in the short term or long term future and that directly affects the probability of getting a generalized failure. Failure probabilities of individual devices or power lines are directly proportional to a generalized/global failure probability.

- **System Failure Propagation Range:** One of the biggest dangers that should be prevented by the decision models is a generalized malfunction; a failure that propagates in the system should be stopped promptly before it cascades into a critical situation like a global blackout. This performance measure takes the average of failure propagation during the system test lifetime. The failure propagation in a particular situation is measured by the extent of node levels through which the failure manages to propagate before the situation is solved by either shutting down a device or closing a circuit breaker.

For example: if a relay fails it might transfer an unbalanced power load to every node it is transferring power to, those nodes will most probably also fail and also transfer their improper balanced loads to the nodes connected to themselves. If the failure is stopped at the level consisting of all adjacent nodes to the relay that failed, the failure propagation would simply be one. For each time the compromised load is transferred to a new set of adjacent nodes we add a value of two to the power of the level number to reflect the fact that as a failure cascades further, the more it propagates the more intense its damaging effects are, this is discussed in details on Chapter 3 dealing with Decision Models design. Decision models should be able to achieve a low system failure propagation range score to be able to guarantee safe operability of the grid.

3. DECISION MODELS DESIGN AND IMPLEMENTATION

In this chapter we first present a detailed discussion concerning the design of decision models and how the decision process works. We then discuss our implementation of a default decision model. Finally we present two other decision model implementations based on Tabu Search and Genetic Algorithms.

3.1. Decision Model Design

Decision models are at the core of our simulation, and as such there are many levels of decisions that can be taken by agents. The highest level decisions are taken exclusively by the management agents. Management agents can use different decision models depending on the testing configuration. It is possible for a researcher to test a certain scenario using for instance an Integer Linear Programming (ILP) decision model and then compare it against an alternative decision model like Naive Bayesian.

3.1.1. Decision Process Design

The decision process works in the following way: The management agent receives all collected data representing the status of the system at a given time. Within the decision framework there is a special layer called the data presentation layer, this layer is in charge of taking all data collected from control agents, and structuring it in such a way as to make it compatible with the selected decision model. A decision model based on Bayesian logic might need the information presented in a different way than a one based on ILP.

If a researcher creates a new decision model and wants to integrate it into the simulator he/she must either conform their decision model to an existing data presenter or create a data presenter that is appropriate for their model. In this way the data presenter layer of the decision model can be compared to the data presentation section of the application layer in the popular ISO networking model.

Once the Data Presenter has formatted the data, the management agent determines if the status of the power grid is normal or if there is a need for self-healing. In the latter case, it uses the desired decision model to generate a strategy that consists of the best corrective actions that can be taken to solve the detected problem. After generating the corrective actions the decision model must then translate the results obtained into a set of corrective behaviors which consist of mappings of various middle layer agents to appropriate roles. The final output that the decision model generates is a set of actions and which agent is responsible for accomplishing them, as that is the format that our MAS understands. The layer within the decision model that is in charge of converting the generated results to this format is called the Results Translator. Figure 11 illustrates the basic decision process design and Figure 12 shows the complete decision model process design.

Figure 13 shows a section of a power system, an example of a set of corrective behaviors that could be used to increase the power from the wind generator to consumer 2 by 100 kW for that power system could be:

DERAgent2 Generate +100 kW

DeviceAgent2 Set Line 3 in Relay +100 kW

DeviceAgent3 Set Line 5 in Relay +100 kW

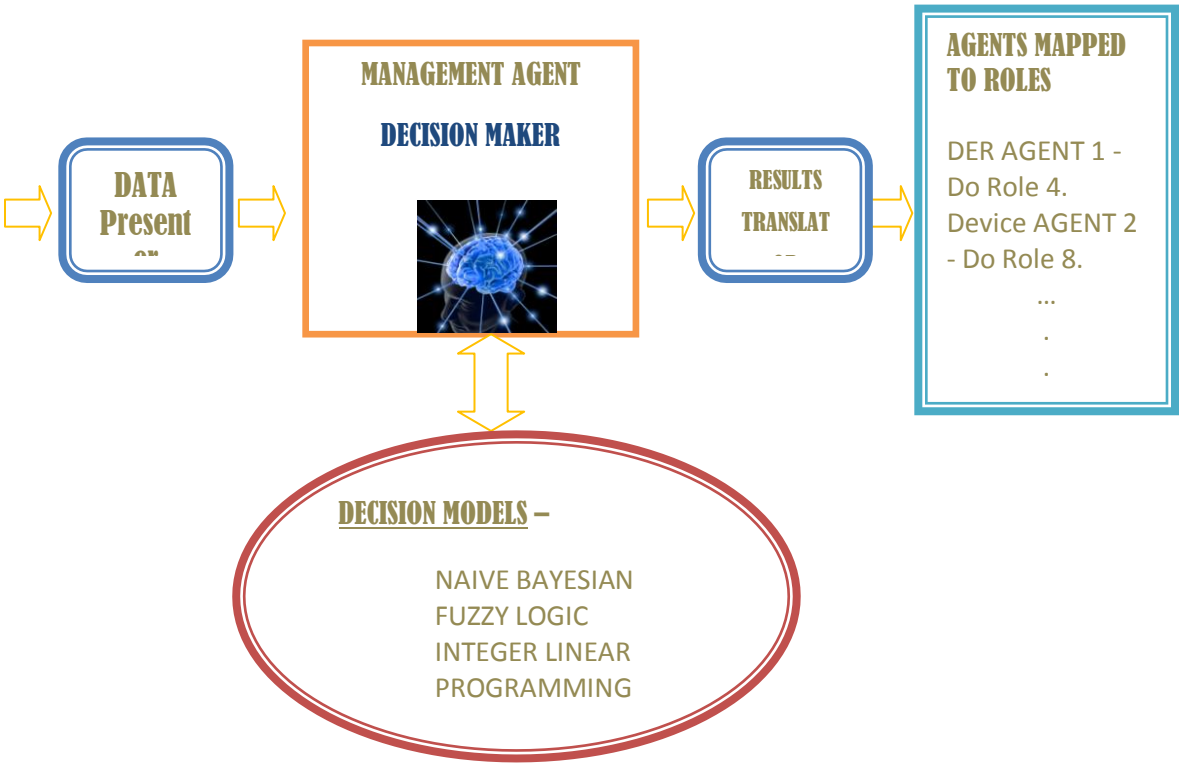

Figure 11: Decision Model Process Design Used in the Simulator

3.1.2. Agent Dependencies

The corrective behaviors generated by the decision model constitute a transaction, and some of the steps composing it might be preemptive, therefore they often must be carried out in a specific order to accomplish their task. Because of this, the management agent sends commands to the different agents and then waits for a confirmation from them before sending the next command to the next agent in the transaction. There are other transactions where the steps are independent and in that case the management agent can send all those commands in parallel. In both cases, if any step of the transaction fails this may compromise the whole transaction.

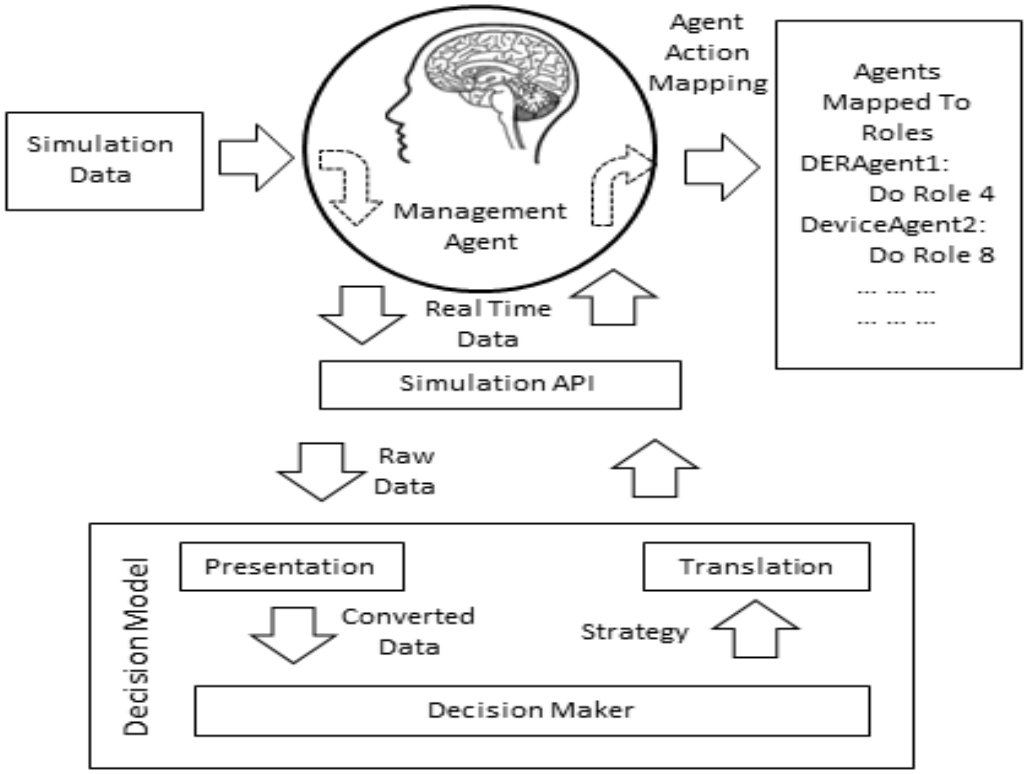

Figure 12: Decision Model Design & Decision Process

An incomplete transaction can leave the smart grid system in a faulty state that can create all sorts of problems and inefficiencies. In these cases the whole transaction must be aborted gracefully and all completed changes must be undone in order to return the system to a valid state. The requirements for corrective behavior execution are in fact very similar to the atomicity, consistency, isolation, and durability (ACID) properties of transactions processing.

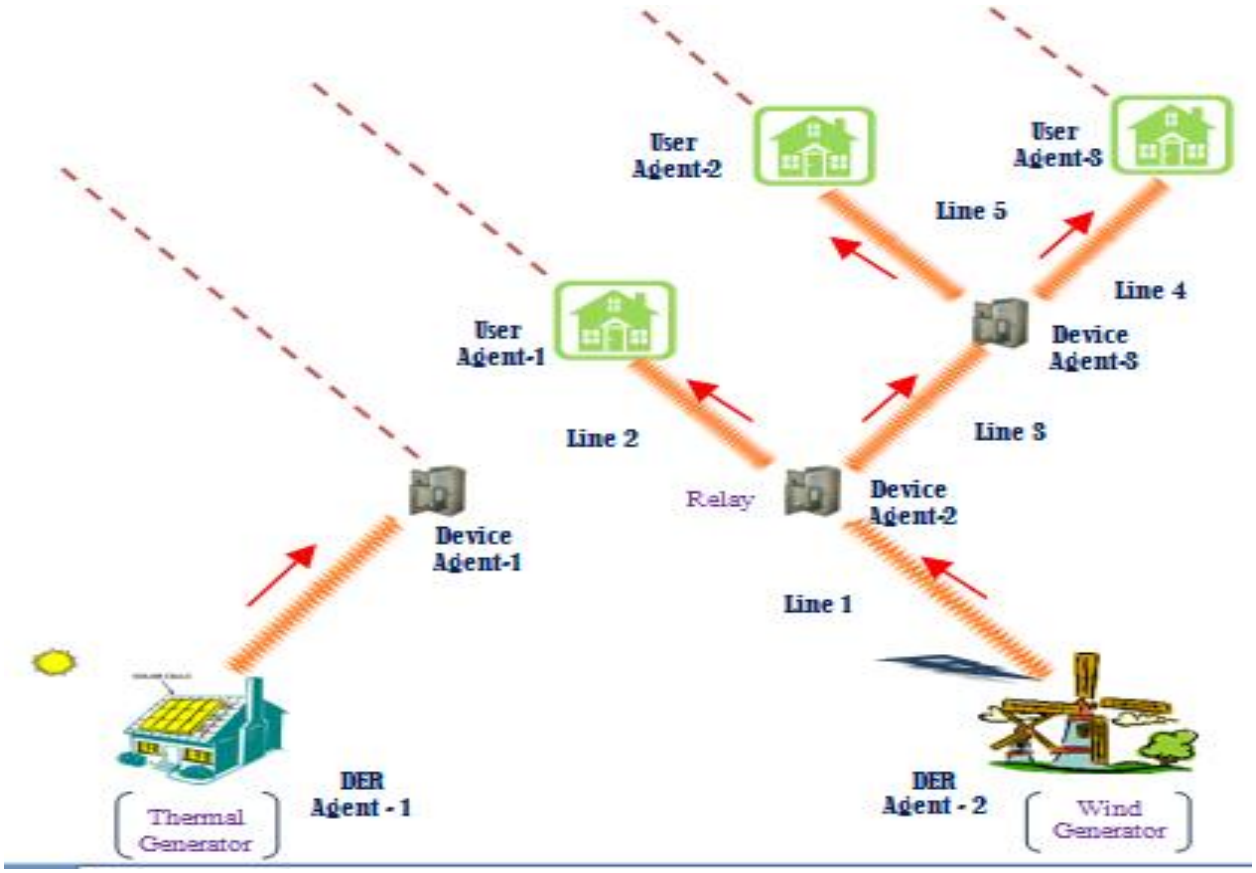

Figure 13: Subsection of a Simple Power Network

3.1.3. Agent Autonomy

Decision models can accommodate the concept of autonomous and semi-autonomous agents. Fully autonomous means that a given agent is able to make decisions on its own whereas semi-autonomous agents can only make autonomous decisions within the constraints of following the higher level commands from the management agent.

For example in a centralized integer linear programming decision model we need to guarantee that resources will be used efficiently and that a global optimal will be reached. Therefore, we require a high degree of control over the operation of all agents, hence we use the

semi-autonomous model to control DER and User agents, as we must make sure that a certain User agent leases power from a specific DER in order to guarantee a global optimal. In a decision model with a lesser degree of control, the management agent could focus mainly on rerouting power during outage situations while the DER and User agents perform power negotiations in parallel, in a fully autonomous way. It's important to differentiate decision levels, the corrective behaviors generated by the management agent are higher level decisions and even though the underlying agents are forced to carry them on, they do have some flexibility on how to carry those actions. Many of these higher level actions can be broken into smaller sub-actions to be carried by different agents with different roles.

Take for example the case of when a user has an outage of 200 kW and is trying to contact a DER agent to lease power. If the user agent is autonomous, it first broadcasts its power demand to all DERS, it gets many replies from DERS and then based on a power supply negotiation chooses a given DER as its power supplier. Before replying and offering its services to the user, a DER agent first determines if it is eligible by verifying a set of conditions, such as if their generator can handle the required load. If the initial conditions are met, they query the management agent to determine if there is existing physical infrastructure to route power between themselves and the user. Only after getting confirmation of satisfiability, they contact the user, then the process continues with the user choosing the appropriate DER (in some cases more than one) and contacting it with a power lease request.

After receiving the power lease request, the DER has to contact the management agent again, this time to prepare the infrastructure to transfer that power to the user. It's not enough for the DER agent to simply send a request to the generator it manages and ask it to emit an extra amount of power of 200 KWatts. Because if the relays between the generator and the expected

consumer are not set to the appropriate routing settings, the extra power generated could end up not going to the intended destination/consumer. That is why the DER agent sends a routing request to the management agent and the management agent creates a set of corrective behaviors (just like when performing self healing decisions) to prepare the exchange infrastructure. Again, each of these steps can be dependent on each other, so the management agent waits for a given step/action to be confirmed before carrying the next.

Following the same (ACID) rules of a transaction processing, if any of the steps fail, we must abort the transaction and roll back any changes performed into the power grid by the previously executed steps, failure to do so would leave the power grid in an inconsistent state.

A decision model that requires a higher level of control such as a centralized ILP can generate behaviors like DERAgent1 lease 200 kW of power from UserAgent2. That behavior will break into all steps mentioned before, except that there is no broadcast from users to all DERs since the resource mappings are already established, also no need for DERs to verify there is a physical infrastructure connecting them to the consumer as the management agent already handled that. All other steps including the steps to establish a routing path are still required.

3.1.4. Decision Model Guidelines

From the experience of implementing and designing our basic decision model we determined some useful guidelines for the implementation of future decision models. With the flexibility of our simulator it's best to create decision models with a specific purpose, handling one to a handful of issues. Designing for a particular task can be shared through publication to be tested by other researchers or also be implemented in unison of other researcher's models that handle other problems effectively for a more complete decision model. The scope of issues is

vast and by allowing specializations in decision models like cascading failures handling or resource allocation, power rerouting, and others, the smart grid community can produce advanced decision models that handle a majority of issues in the smart grid. Decision Models communicate with the simulator using a Java based API. Using the API of the simulator anyone can effectively use any internal implementation as long as they can convert appropriate variables from and to our API. Creators of the decision model will have to choose early on which middle-layer agents will work with a high degree of autonomy and which will be semi-autonomous and work under full control of the management agent. It's really a compromise either way. Again, the decision models will specify each agent's autonomy, for now this is implemented in the following 2 ways:

1) The corrective behaviors generated by the decision model are themselves restrictive and ask precise actions from those agents
2) Agents are setup to not take initiative on their own when a problem occurs, but rather wait for commands from the management agent.

Features that are supported in the system framework to save some trouble in testing and implementation of the decision model are the ability to rollback to a pre-decision state when a decision fails to properly execute, a healthy log system to later analyze the decisions made, and methods of providing feedback to help evolve our API and the decision process to keep it relevant in simulating the smart grid.

3.2. Default decision model implementation

The default decision model that was implemented is based on the autonomous paradigm, that is it doesn't strive for a high degree of control, the corrective behaviors we generate don't restrict the autonomy of User and DER agents. We wanted a simple model to test the functionality of the simulator, it doesn't try to guarantee the most optimal results for the system (global optimal), it just aims to solve the situation at hand (local optimal). Solving the local problem in one section could potentially create a problem in another section, but we are assuming that even if new problems are created, since the management agent is solving issues in real time, at some point a balance will be found where the grid is working without outages, even if this state is not reached in the most efficient way. We have also worked on Tabu Search and Genetic Algorithm decision models, those will be described in sections 3.4 and 3.5. We have also compared and performed extensive testing and experimentation of all of these decision models as will be described in Chapter 4. When developing future models like linear integer programming we will work on aiming for optimal performance. All the results from the different decision models can be compared using as basis the performance measures described in Chapter 2.

As mentioned above the simple decision model we implemented allows many agents to work autonomously, more precisely, while the management agent is trying to solve an outage, the user agents and DER agents are also trying to solve the outage by performing power negotiations in parallel. They can establish a power lease to supply the unsatisfied demand until the original power supply is restored, that way minimizing consumer outage times.

These autonomous behaviors of the agents decrease the control on the system, but have the advantage of improving performance by having agents implement different strategies in parallel to solve consumer outages or other grid problems.

To evaluate the default decision model we will focus on two situations that can occur in the grid that require self-healing. The first is a simple user outage, caused generally by a power line or a device failing. This usually requires the main electrical company to reroute power in an efficient way, in order to solve the outage. The second situation deals with a cascading failure. If a device fails due to receiving a higher power/voltage/current than expected, the excess load could be transferred to the next component which could not be ready for it, causing the same situation to repeat with the next component, creating a cascading failure that can propagate quickly through the system and compromise the large portions of the grid. The latter situation should be handled with higher priority than the former one, since if we don't respond in a timely manner the failure could critically damage large portions of an electrical grid.

3.2.1. Simple Outage Example

In the case of dealing with solving customer outages, power should be rerouted in an efficient way. Whenever the management agent receives system status data, it will first loop through all consumers and find if any of them is experiencing an outage. If a consumer with an outage is found, the management agent will determine all power paths leading into this consumer, in other words it will look at all input power lines that are actually delivering power to this consumer or supposed to, then it will trace the power to the generator producing it.

As the next step, it chooses one of these power paths that contains a line that has experienced a failure. So its job now is rerouting the power that was supposed to go from the failed line in an alternative way.

The process the management agent follows to reroute power and solve outages works in the following way:

- Identifies the failed power path
- Detects all potential paths from the consumer to the generator, not necessarily valid power paths.
- Searches through all those paths and chooses the first path that has a common relay component with the original power path, always the common relay closest to the consumer is chosen. Detecting a common relay component is key, as power routing can only be done using relays.
- Once the common relay is detected, it adjusts the settings in the relay to increase the power routed through the line linking to the new valid path found, by the amount of power originally intended for the failed line.
- In the same way, it decreases it by that same amount for the line linking to the original power path containing the failed line.
- Traverses all relays between the common component and the consumer in the power path and makes sure that their routing settings are congruent with the changes. For the new path, every relay has to be set to increase the power by the adjusted amount for each power line in the path until we reach the consumer.

When looking at figure 14, we can see a very simple example of how this works if one of the power lines supplying power to consumer 1 fails. The power path containing the failure is the one starting on the main distribution center (MDC) and passing through relays 1 and 2 and reaching consumer 1. When listing all the different paths from any existing generator to consumer 1 (not including the failed one) we only find one, that starts from the MDC and goes through relays 1 and 3 into consumer 1, in this simple example there is no other alternative path. We don't count the path coming from the wind generator through relay 5 into consumer 1 because as explained before, our decision model is trying to reroute power only from the generators that belong to the main electrical company. Note that in a more complex power network the electrical company could have many generators supplying consumers.

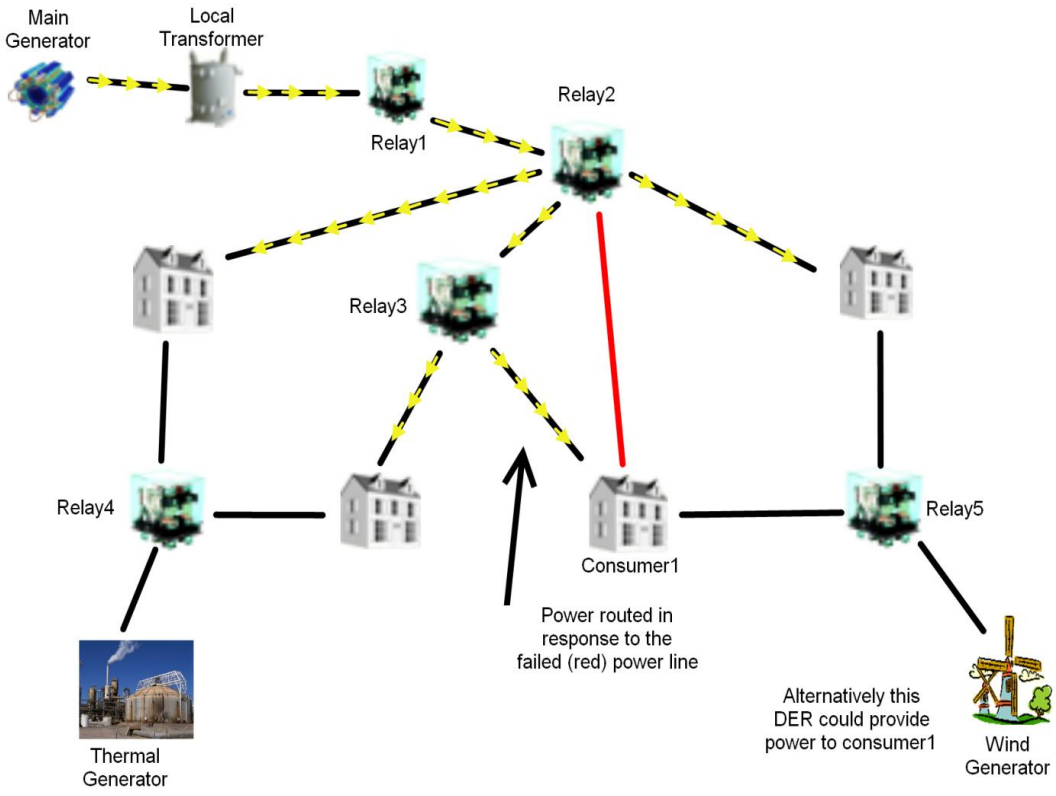

Figure 14: Power Rerouting Process for a Simple Instance of Case 1

The closest common relay between the original power path and the new found path is relay 2. Therefore to successfully reroute the power, relay2 must be set to route all power that was originally set to go though the failed line, instead to the line in the other path that goes to relay 3. Then relay 3 must be configured so that it knows that all the extra incoming power should be routed through the power line going into consumer 1.

At the same time that the management agent is evaluating options and creating this strategy, the user agents associated with affected consumers are doing their part in order to fulfill the power demands. A user agent that detected an outage in its consumer will broadcast a message to all DERs in the microgrid and tell them about its power needs. Then the usual power negotiation process takes place.

Since the user agents and the management agent are both trying to solve the problem at the same time, this can sometimes lead to conflicts. If the management agent manages to reroute power properly but a user agent has already started a transaction with a DER, then we must halt the transaction in mid place because we don't want to use power from extra resources if the main electrical source is already generating that power, it would be wasted if not rerouted. So in that case we abort the transaction and rollback any changes, as explained in previous chapters. There is also the possibility that the management agent takes too much time to solve the outage and a user agent already gets a lease from a DER, this is desired behavior because it minimizes consumer waiting time. But once the power from the MDC is restored, we must cancel the power lease and undo the transaction changes.

There are many reasons why a management agent could take too much time before rerouting, one could be a self healing strategy that has to be aborted because for example a device failed and then the next possible strategy possible has to wait for a resource (for instance a relay) that is currently locked because it has reached its maximum capacity. Power lines and devices have capacities, exceeding power levels on those lines could make them fail, causing potential cascading failures. Also, if there is no way to reroute power from the main distribution circuit, then users should lease power from DERs indefinitely as it's the only available solution.

Note than in this default decision model DERs act like resources to be used when the main electrical company can't supply all required power. But in other models like ILP or Tabu Search we might decide to use a given resource rather than rerouting power from the main electrical circuit based on a utility function that takes into consideration the cost of generating power from every given source and aims at optimizing the system performance and resource utilization, as described in the model presented in [4].

3.2.2. Cascading Failure Example

In the second case, when a device has failed creating a potential cascading failure, it's really important to stop the failure propagation by shutting down certain components. The first action the management agent must take is to send a broadcast to all device agents managing the circuit breakers in charge of isolating the microgrid from the external global grid. All those circuits must be closed first to prevent any failures from propagating outside the microgrid, as we don't want a cascading failure of global proportions. The next corrective actions are sending shutdown commands to certain devices in the network to stop the propagation within the microgrid itself.

This process is trickier than it would initially appear to be, it would seem as simple as sending a shutdown request to the device that failed, but during the time that the management agent makes its decision and sends the corrective shutdown action, the failure could have already propagated to the next level, meaning every device connected to the failed device could have potentially failed as well. Waiting for more data from control agents and trying to make further decisions could risk vital time and allow the failure to keep spreading to further levels. Every level the failure spreads makes it more complex to fix than the previous one, because we need to send requests to shutdown more devices, and the more devices we miss the more the failure spreads.

For example, imagine a simple network where every component connects to two other components, every level that the failure proceeds we will have 2 to the power L devices to shutdown, where L is the level number. It's easy to see how the complexity can increase in a large microgrid where each component could connect to any number of other components, making the situation much harder to solve.

The default decision model handles this situation by trying to stop the failure at a propagation level different than the current one from the start. In other words, instead of targeting the component level itself, it tries to stop the failure on levels further ahead. This is regulated by a parameter in the decision model. If after analyzing the next status data of the system, the failure is still propagating, the parameter will be adjusted to attempt to stop the failure at a level that is farther ahead. The amount the distance between failure point and correction level is increased when a wrong estimation occurs is also based on a tuning parameter.

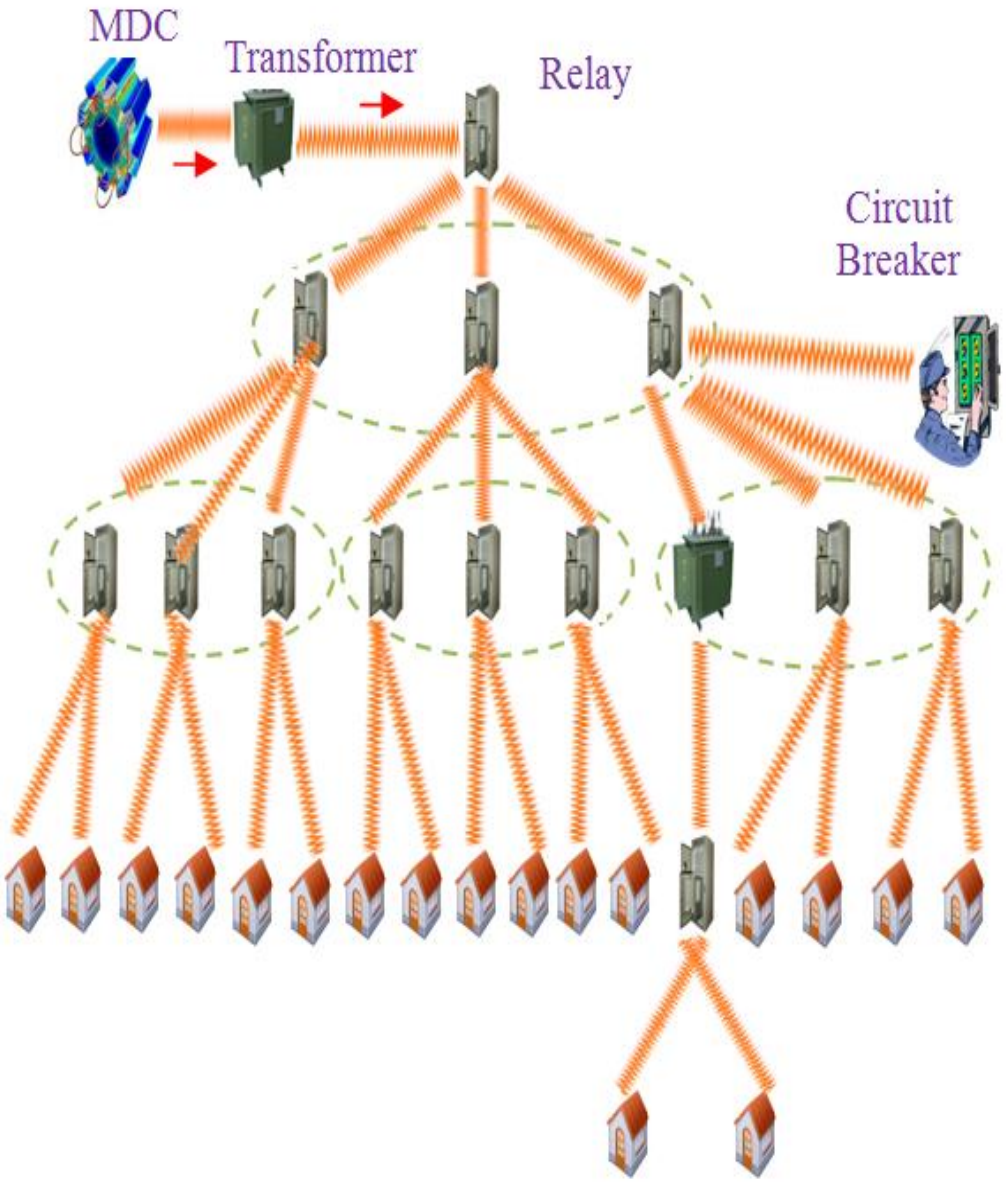

Figure 15: Section of a Simple Power Network Showing Propagation Levels

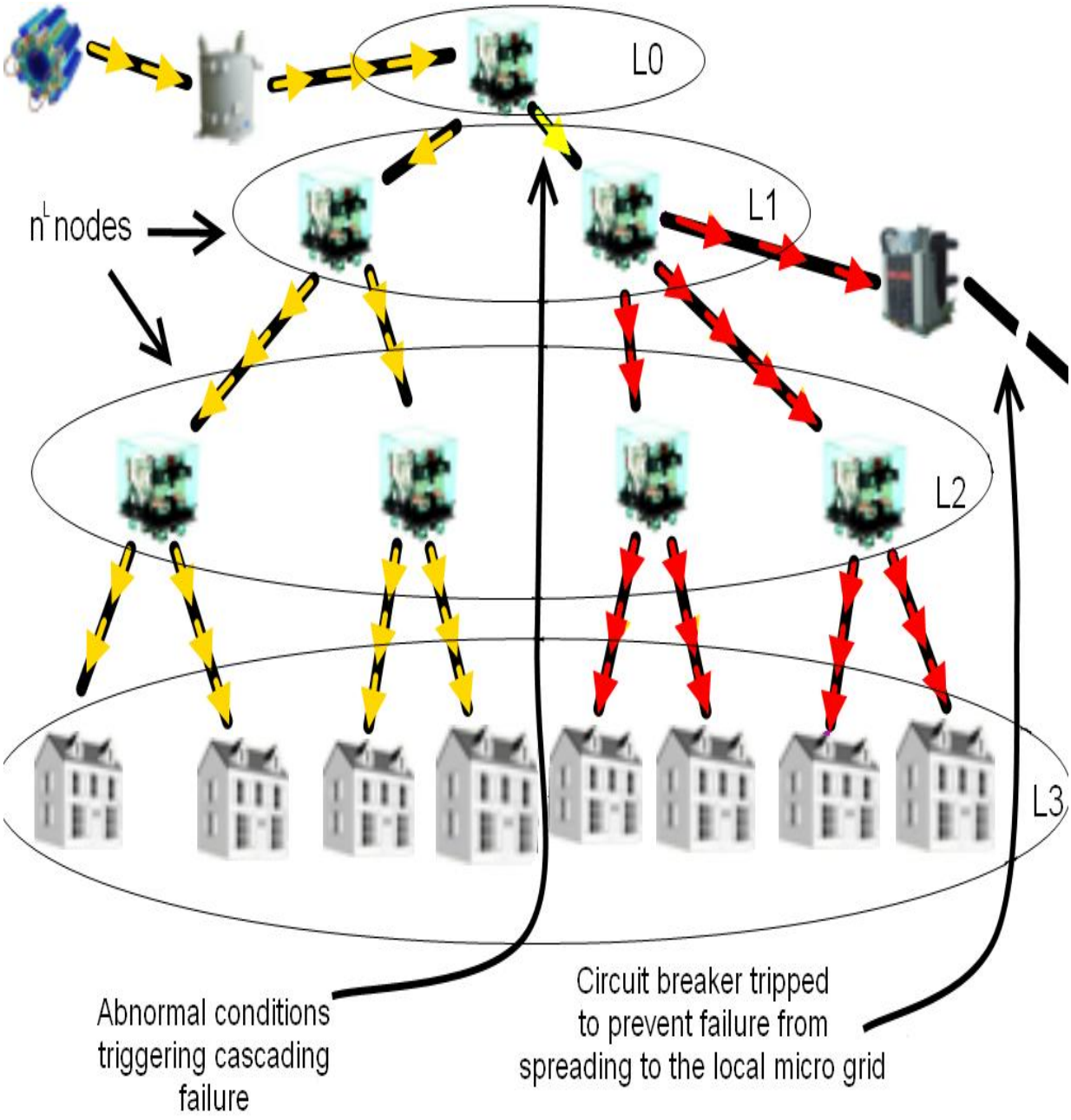

Figure 16: Power System Illustrating Propagation Levels for Case 2

3.3. Decision Model Application Programming Interface

In this section we will describe the simulation API(Application Programming Interface) in details, this is the interface to be used by researchers designing their own decision models to be integrated and tested in the simulator. It is important to note that every decision model must be written in java and inherited from the ***DecisionModel*** class built in the simulator. These allows the different child decision models to use the existent support functions that are common to all decision models and are already implemented in the parent class ***DecisionModel***. Also, every decision model must override the ***MakeSelfHealingDecisions*** method from the parent class and replace it with their own implementation suited for that decision model. The main functions that are essential in the API are:

ArrayList<ConsumerInfo> getAllConsumers(SimulationEnvironment se, ArrayList<ComponentInfo> componentsStatus)

ArrayList<ConsumerInfo> getConsumersWithOutages(SimulationEnvironment se, ArrayList<ComponentInfo> componentsStatus)

These two functions return a list of structures describing all the consumers in the system or only the consumers having outages respectively. Most of the time, decision models will just be interested in knowing which consumers have actual outages, in order to generate a self healing strategy to solve those outages, but in some situations knowing information about all consumers might be desirable as in the case of resource allocation optimization. In some cases decision models might also try to improve the performance in the grid even if there are no situations requiring direct self healing per se.

The ***ConsumerInfo*** structure returned contains information about each individual consumer. Among the fields contained in that structure we have the consumer id, which is a unique string identifying that consumer; It's overall required power at any given time for both critical and non critical power loads; The current power outage value for both critical and non critical loads; The cost that the consumer incurs for every simulator time unit that it remains in an outage state. There is also a preferred generator type field that expresses what type of generator the consumer would prefer to be served from when having outages, for example wind or green energy generators. Also, a price limit established by the consumer that determines the maximum price this consumer is willing to pay to a generator supplying it during outages. Last but not least, a generator score preference, so that whenever possible only generators that have been ranked by the community of consumers as having a certain level of reliability will be elected by this consumer.

In order to retrieve information about the generators that could possibly be assigned to supply those consumers, decision model designers can use the functions:

ArrayList<GeneratorInfo> getAllGenerators(SimulationEnvironment se, ArrayList<ComponentInfo> componentsStatus)

ArrayList<GeneratorInfo> getAvailableGenerators(SimulationEnvironment se, ArrayList<ComponentInfo> componentsStatus)

These two functions return a list of structures describing all the generators in the system or only the generators that have any power currently available respectively. Most of the time, decision models will just be interested in knowing which generators have any available power, since only those generators can be assigned to consumers with unserved demands during normal self healing situations. But in some situations knowing information about all generators might be important, for instance this function can become really useful if the decision model attempts to optimize system performance by reassigning resources.

The ***GeneratorInfo*** structure returned contains information about each individual generator. Among the fields contained in that structure we have the generator id which is a unique string identifying that particular generator. The maximum amount of power that this generator can produce at any given time and also the amount of power that is currently available. The current power available is basically the maximum power amount that can be generated minus the amount of power currently being generated and shared. Another field shows the cost/price that the generator charges for supplying power, it is measured in Kw/hr. There is also a generator type field that indicates what kind of generator this is, whether it's a nuclear, coal, solar panel, wind generator among others. Finally, a field is also included with a reliability ranking for the generator, this ranking is determined based on the votes of the community of consumers in the grid, as explained before.

Once the decision model obtains the list of consumers with outages and the list of available generators it still cannot directly map a given generator or generators to a particular consumer. This is because even if the generator or set of generators have available power that is sufficient to solve the outage of that consumer, there could be other issues like not having a valid physical path connecting an individual generator to that particular consumer.

In order to help on determining which generators are eligible for a particular consumer the following two methods can be used:

boolean isTherePathBetweenConsumerGenerator(String consumer, String generator, SimulationEnvironment se, ArrayList<ComponentInfo> componentsStatus)

ArrayList<PathInfo> findAllPathsBetweenConsumerGenerator(String consumer, String generator, SimulationEnvironment se, ArrayList<ComponentInfo> componentsStatus)

The first method allows us to test if there is at least one valid energy path connecting a consumer and a generator. Once we have determined that they are connected, we can use the second method to obtain information about all paths that could be used to transfer the power from the generator to the consumer. The decision model has the task of using the most appropriate path that would maximize the intended goals of optimizing the power grid. Information about different paths that would help the decision model is contained in a special structure called a ***PathInfo*** structure, a description of all the information fields in the structure follows.

The information contained in the ***PathInfo*** structure consists first of the total distance that the particular path spans. Also the maximum power, voltage and current capacity of the path, meaning that at least one device or line within the path doesn't have the capacity to transfer a higher amount of power, voltage or current at a given time without failing. Another field contains the total number of lines and nodes that conform the path. There is also a failure probability reflecting the potential for this path to fail in the future, the failure probability is usually equal to the failure probability of the device or line within the path that is most prone to fail.

The API contains many other useful methods in order to do things like retrieve all information about all devices or just devices that have failed in the system. As researchers start designing their own decision models, they can request their required functionalities, and all those functionalities that are continuously being used by researchers can be incorporated into the API and be supported natively within the simulator core.

The decision model will output its results by generating a set of strings with commands or actions for specific agents, all of these commands together conform the self healing transaction. The commands are simple, for example to assign a given generator to a consumer the command would be in the format "***assign consumerId generatorId powerAmount***", meaning assign the given generator to supply the given consumer for that power amount. Those command strings are supplied to the API where the results translator will convert them into a self healing strategy in the format that the simulator understands consisting of middle layer agents and roles.

There are many advantages to this approach of inheriting new designed decision models from the parent class ***DecisionModel*** built in the simulator and having the decision model within the management agent as opposed to having it encapsulated within a different agent. One of the main advantages is a decrease in network communications overhead, since having the decision model within the management agent will save all the need of sending external agent messages whenever they need to communicate. This is crucial, because the state of the power grid is being sent in real time to the management agent and it should be able to analyze the data, detect and diagnose any faults and generate a self healing strategy very quickly.

Another fundamental advantage is that every new decision model through inheritance is already provided with all the required common functionality, by calling this standard API decision models designers can save the time of implementing that functionality themselves. It makes no sense to have different decision model developers to implement the same functionality; therefore having this support allows them to focus on the more specific goals of their models. At the same time, due to inheritance, any performance improvements done to the core parent ***DecisionModel*** class will automatically cascade to all the different decision models derived from it, which is a very powerful feature of inheritance.

Another big advantage of this scheme is that if new researchers require new functionality not present in the current API, this new functionality can be added and provided without changing the old methods, by simply adding the new ones. That means that the existing decision models won't be affected by the changes and would need no modifications to run in the extended API. So another main advantage of this scheme is that it is very extendable and supports legacy decision models.

3.4. Tabu Search Decision Model Implementation

In this chapter we first present some background information on Tabu Search. We then discuss in details our Tabu Search implementation including two different solution encodings, the workings of the objective function and the different memory strategies that we used to explore the search space.

3.4.1. Tabu Search Introduction/Background

Local search heuristic algorithms have been used extensively to try to solve all kinds of optimization problems, like for example the travelling salesman problem. The problem with local search techniques is that more often than not the solutions get stuck in a local optimal, instead of finding the global optimal. In order to make the exploration of the search space more efficient it is important to keep track of other information besides local values such as the current best solution or the current value of the objective function, we must keep track of different information related to the exploration process itself.

The set of all this information can be tracked by a memory module/component dedicated specifically to this task. Tabu Search combines a local search with the efficient use of memory to keep track of relevant exploration process information, this use of memory to avoid getting stuck in local optima is one of its essential properties.

Tabu Search was first presented by Glover and Hansen in [21] [22][26] and later further formalized in [23], [24], [25]. There have been many research studies and tests that have proven the efficacy of using Tabu search when compared to other optimization techniques. Tabu Search is actually a heuristic with its own rules that orients and guides another local search procedure, so it is usually considered a meta-heuristic.

Within the memory structures used in the Tabu Search there is a special ***tabu*** *list*, this list is used to temporarily ban certain solutions, this is done in order to filter the solutions that will be admitted into the neighborhood during a certain interaction of the search. The purpose is to direct the exploration process to different potential areas and to also make sure not to leave a promising area too soon. In order to achieve this its fundamental to have a proper procedure for keeping an

optimal balance between both Intensification and Diversification; Intensification allows the thorough exploration of promising search areas while Diversification guarantees that the search won't just dwell on the same areas and miss other areas with promising potential. The memory structure in Tabu Search is separated into three main levels which are aimed at warranting that balance:

- **Short-term memory:** Its role is to forbid some moves that would return the search to situations/solutions recently visited. Every solution in the short term ***tabu*** list is assigned an expiration time and cannot be revisited until its ban has been lifted when that time is due. The short-term memory also intensifies the search by focusing on the moves that have higher evaluation scores. Short-term memory by itself has been proven to be more efficient and achieve better results than standard local search methods when dealing with problems of moderate complexity. For complex problems it is very important to have efficient strategies for Intensification and Diversification so intermediate and long-term memory structures are required [27].
- **Intermediate-term memory:** A list of rules intended to bias the search towards promising areas of the search space. In other words, it helps intensify the search in those areas that seem attractive.
- **Long-term memory:** These Rules promote diversity in the search process; it allows the search to shift to unexplored areas whenever it is stuck in the same areas for a long time.

Intermediate and long-term memory structures are the main components in charge of handling the intensification and diversification of the search. An intermediate-term memory

structure could use a strategy that forbids solutions with particular properties or prevents certain moves, for a certain amount of time. This is very important when working in large search spaces, in those situations it is easy to get stuck in a local area despite having a ***tabu*** list prohibiting entire solutions, therefore it is useful to have a memory structure forbidding some attributes of a solution instead of the solutions themselves [28].

Another important concept for a Tabu Search implementation is that of an Aspiration Criterion. When we use memory strategies that forbid certain general attributes of solutions instead of the solutions themselves, many times among the many solutions we really intend to prohibit we might also be eliminating solutions that are actually promising. An aspiration criteria is set to help distinguish these high quality solutions and still consider them even if they possess traits that are currently forbidden. An example of a commonly used aspiration criterion is to allow solutions which have a higher score than the currently-known best solution.

3.4.2. Tabu Search Implementation

There are two possible implementations we have devised for the tabu search decision model, each with their own advantages and disadvantages. In our tabu decision models we focus mainly on handling consumer outages by doing intelligent resource allocation, that is choosing appropriate mappings between consumers currently experiencing outages and available generators, this mapping process is not that simple considering that a given consumer can get its unserved demand supplied by many different generators. Besides the resource allocation problem we also focus on the prevention of cascading failures and also on handling them in the event that they still occur despite the prevention measures.

3.4.2.1. Solution Encoding Implementation A

In order to encode a solution that reflects proper resource allocation to consumers that are currently experiencing outages, we first obtain the list of ***ConsumerInfo*** structures for all consumers with outages in the system, every consumer will be identified with their index position within the array list of consumers with outages. That index will be the unique identifier for a consumer through the complete tabu search exploration during that particular self healing decision attempt. Every time the decision model is invoked for a new decision we get the new updated list of consumer outages and hence the ids will be different. A solution consists of a set of values, each value in the solution is an integer that either represents an index of a consumer into the array of consumers with outages or the index of a generator within the array of generators that currently have available power to share. The decision model can retrieve the list of available generators from the simulator by using the API methods previously described.

The way the mapping between consumers and generators is established in this implementation is by dividing the solution slots in groups of consumers and having generator ids in the slots, the generator ids in a given consumer group specify a mapping between that given consumer and those generators, meaning that each of the generators will supply at least part of the power outage demand of that consumer, not necessarily all of it as the generator might not have sufficient power. We define a fine tuning constant to determine the maximum number of generators we can use to satisfy the demand of a given consumer, we denote it as ***maxGeneratorsPerConsumer***. It is important to define this tuning constant as the solution size will depend directly on it and we want to limit the solution size in order to avoid large search spaces and reduce the amount of exploration we have to do.

The way the mapping between consumers and generators is interpreted in the solution is the following, the first generator would attempt to satisfy the full demand of the consumer but if power resources aren't sufficient then the next generator in the list will continue, until the demand for that consumer is satisfied. Ideally the consumer demand should be satisfied by the generators assigned in its group. The number of generators assigned to a particular consumer can't exceed ***maxGeneratorsPerConsumer*** fine tuning constant that we previously defined. Solutions can be low quality whenever particular consumers don't have their full demand fulfilled or when we assign generators to a consumer that don't really have a feasible way of communicating with it, for example if there isn't a valid physical infrastructure to transfer the power between them. There are many other different aspects affecting the quality of the solution, we review those later in this chapter when we discuss the objective function.

The solution size for this implementation is expressed by ***maxGeneratorsPerConsumer*** multiplied by the number of consumers with outages. It is possible to have less generators than ***maxGeneratorsPerConsumer*** allocated to a consumer in a given solution, in that case the remaining slots of that consumer group are filled with blanks.

Figure 17 illustrates how solutions are encoded in this implementation when ***maxGeneratorsPerConsumer*** is 3. In the example, generators 5 and 6 are assigned to consumer 1, only generator 3 is assigned to consumer 2 , generators 12, 8, 6 are assigned to consumer 3 and generators 17, 11 and 6 are assigned to consumer 4. Note that it is valid to have generator 6 assigned to all these different consumers because it is always appearing at the end of a consumer group meaning that power could still remain in that generator after the demand of a consumer is fulfilled. On the other hand generators appearing in any slot of a consumer group but the last, can't appear again in other consumer groups since we are assuming that every generator uses its

full capacity when supplying a consumer. Figure 18 shows an example of an invalid solution that breaks the rules of the representation.

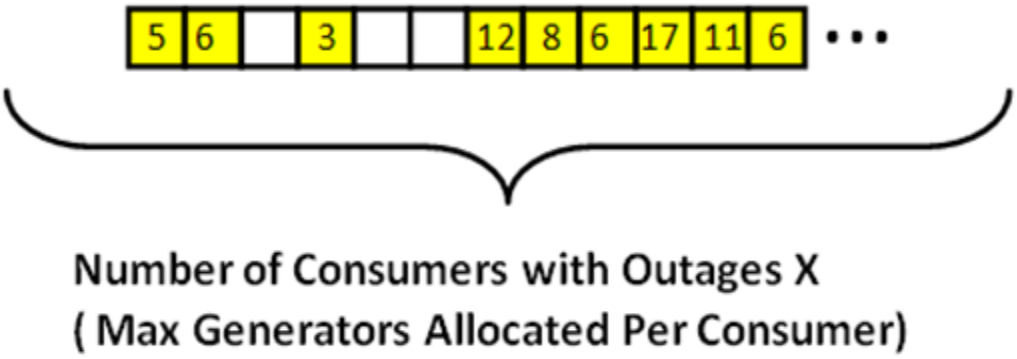

Figure 17: Solution Encoding A, Using Ordered Placement

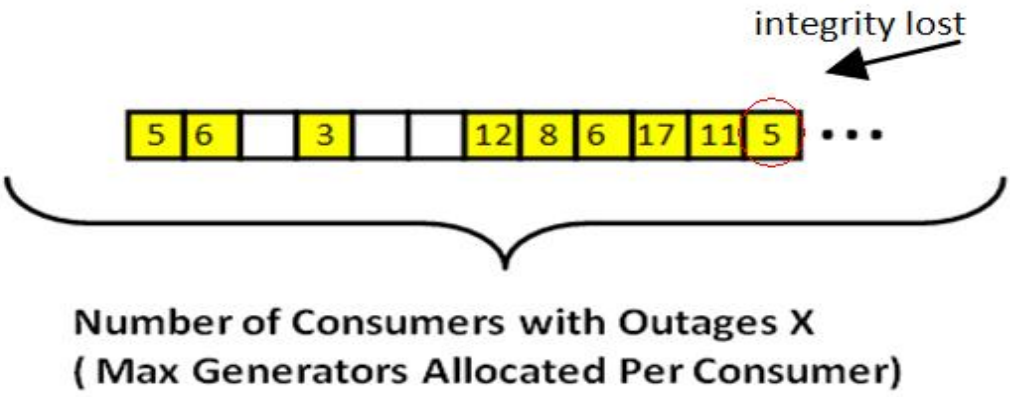

Figure 18: Solution Encoding A, Invalid Representation

3.4.2.2. Solution Encoding Implementation B

Just like in the previous encoding, the indices within a solution are relative to their position on the list of information structures for all consumers with outages in the system and the list of all available generators in the system, both of which are directly obtained by invoking the proper methods within the API. Those indices will be the unique identifier for a consumer/generator through the complete tabu search exploration during that particular self healing decision attempt. Again a solution consists of a set of values that are grouped as mappings between consumers and one or more generators.

In implementation A we compromised some fairness when initializing solutions in exchange for a reduced search space. It is possible to reformulate encoding A by adding support for unordered placement of the consumers and be able to obtain higher distribution fairness. That can be done by using the same division into groups per consumer, but instead of placing the groups in order by consumer index we place the groups in any random order. In order to know which customer is mapped to a particular group, we add a header to the solution. The header consists of as a many slots as there are consumers with outages and each slot stores the index of a consumer. The consumer index values are placed in the exact order that we should follow to interpret the group mappings to consumers. Figure 19 shows an example of the solution encoding B.

In the example ***maxGeneratorsPerConsumer*** is set to 4. Generators 5 and 6 and 1 are assigned to consumer 3, generators 17, 11, 1 and 9 are assigned to consumer 2, generators 12, 10, and 2 are assigned to consumer 1 and generators 8 and 2 are assigned to consumer 4.

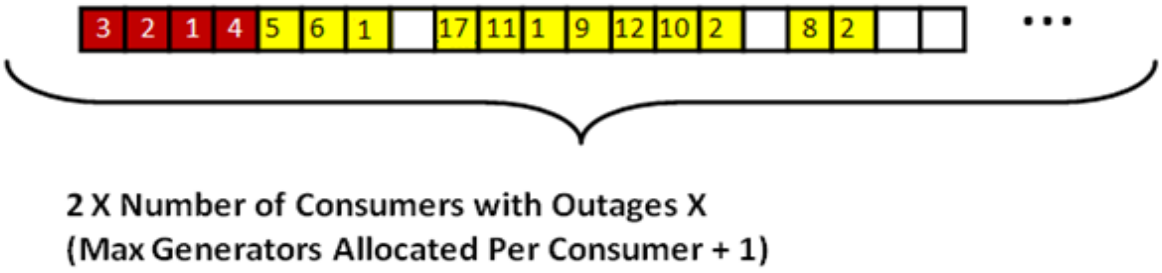

Figure 19: Solution Encoding B Example

In this implementation the consumers groups can appear on any order, this is to guarantee fairness when we initialize the solutions. The solution size is expressed by ***maxGeneratorsPerConsumer*** multiplied by the number of consumers with outages plus the header size (which is also equal to the number of consumers with outages). Having a header to specify ordering seems to increase the complexity and size of the search space as opposed to having groups sorted in ascending order based on the consumer index, but having the solutions sorted would give preference to consumers with earlier indexes and affect the fairness of the resource allocation. Choosing a particular implementation involves a compromise between absolute fairness and smaller solution size. In our testing we experimented with both implementations, these experimental results are detailed in chapter 4.

3.4.2.3. Solution Initialization

The procedure to initialize a solution differs slightly depending on which of the two solution encodings we intend to use. For solution encoding A we initialize the solutions by

generating ***maxGeneratorsPerConsumer*** random numbers in the range [0 : ***numGeneratorsAvailable***] for every consumer group. Whenever we obtain the value 0, this will represent a consumer that is assigned to less than the maximum number of generators and as described in the previous section these appear as trailing blank slots at the end of the consumer group. For solution encoding B we must first determine the consumer group order, to do this we generate random numbers in the range [1 **:** ***numConsumersWithOutages***] and store the obtained values as the header. We then proceed to do the same process we did for encoding A.

Since values are determined randomly, sometimes the results can be invalid for the representation, to handle that we adjust the solution every time we get an invalid value to restore its integrity. For example, if a generator that was already used in a previous consumer group and wasn't appearing in the last slot is reused, that breaks the integrity of the solution as that kind of arrangement is against the rules we defined for the encoding. To fix that situation we would generate alternate random generators to replace that one until one that is suitable is found.

3.4.2.4. Objective Function

The objective function evaluates particular solutions and assigns them a quality score which is used by the tabu search to know how to guide and orient the search. Since the simulator evaluates the performance of decision models based on the performance measures described in section 2.5, it makes sense to base the score determined by the objective function on the same criteria. So the objective function computes separate scores for each of the performance measures evaluated in the simulator and then combines the score into one value that is returned to the tabu search as the total overall quality score of that solution.

It is important to note that when we evaluate a solution we move from left to right and assign consumers to generators in the order the groups appear in the solution, the assignment is done by making sure that we don't assign an extra generator to a consumer until all other consumers have been given an assignment chance. That is we will handle one mapping for each group instead of handling all the mappings for one group right away. In other words we first map Consumer 1 to the first generator in its group, then we proceed with Consumer 2 and so on. We won't map Consumer 1 to the second generator in its group until all other consumers have had their first generator assigned and evaluated.

This is done to ensure some fairness in the distribution of resources to consumers, otherwise the consumers with lower indexes would have always resource allocation preference. Even when using solution encoding B where the consumer order is flexible, it is not desirable to handle all mappings to a given consumer at one time, since that would give resource allocation preference to that consumer every time we evaluate a solution in that Tabu Search run.

A description of how each performance measure score is computed follows:

- **Outage Time:** When evaluating solutions that are high quality it is very hard to estimate what will be the impact on the system in terms of the total time consumers have to wait to get their outage solved. But if the solution is lower quality, meaning part or all of the demands of power outages for some consumers are not satisfied, we can infer that this will increase the total outage time in the system. If that solution was used the consumers would still have outages for at least until another cycle when the status of the grid is sent to the management agent again and the Tabu search decision model is called to determine a new self

healing strategy. Therefore we penalize every solution score with a value proportional to the total unserved demand of all consumers in the solution. By making this a penalizing score we lower the overall value appraisal of solutions that would lead to higher total outage amount time in the system. Despite this being a penalizing score the value is added rather than subtracted since the tabu search aims to the minimization of the solution score when looking for high quality solutions. The exact value used is obtained by dividing the total unserved demand of all consumers in the solution by their total demand, this would result in a real number between 0 and 1. We explain how this number would be combined into an overall score for the solution reflecting all performance measures later in this section.

- **Power Cost:** To compute how this solution will affect the overall cost of the system we take into consideration both the cost for generators to produce a certain amount and type of power, as well as how costly it is for a particular consumer to experience outages, the more critical the consumer the higher the price. For generators, we compute the total amount of power being supplied to consumers by each individual generator in the solution and we multiply that value by the cost of power generation of that generator, then we divide that value by the maximum power that generator could produce. Then, we add the cost incurred by each individual generator to compute the total cost. For consumers we evaluate how much of their power outage will be unserved and then multiply their power outage amount by the cost per outage time unit of that particular consumer and divide by the total power required by that consumer. Finally we compute the total cost for

all consumers represented in the solution. The final cost of this solution is the sum of the total generators cost plus the total consumers cost divided by the maximum total consumer cost plus the maximum total generator cost. Just like when computing outages, this will result in a real number between 0 and 1, to be later combined in the overall solution score.

- **Power Quality:** To evaluate the overall power quality of a solution, that is the level of consumer satisfaction, for every consumer and the generators it is mapped to, we evaluate to what extent they fit all the satisfaction criteria previously described in the performance measures section. In other words how would it satisfy: Price Limit Preference, Generator Type Preference and Generator Ranking Preference, for every consumer a score is generated based on how many of these criteria are satisfied. The overall power quality of the solution is determined by the average of the satisfaction level of all consumers in the solution.

The final result is a real number between 0 and 1 representing how high the satisfaction level in the solution. For this particular performance measure the value that will get aggregated to the other performance measures scores isn't this resulting value but actually (1 - resultingValue). In other words what will get combined is the level of dissatisfaction, the reason for this is that the Tabu search decision model is performing a minimizing search, so better solutions have lower score. So the higher the satisfaction level of consumers for a given solution, the lower it will be the dissatisfaction level and the less it will contribute on increasing the solution score.

- **Variability:** To estimate the variability value of a solution we compare the paths of the assigned generators to its given consumers, we verify how many actions should be performed to establish a power routing path between a consumer and all the assigned supplying generators to it. That total number of actions for all consumers is divided by the maximum possible number of action that could be performed on every device in the paths involved. The result is a real number between 0 and 1, to be later combined in the overall solution score.
- **System Failure Prevention Level:** To estimate the probability of a failure occurring after applying a given solution to the grid, we compare the paths of the assigned generators to its given consumers. Each one of the chosen paths has a probability of failing that corresponds to the device or power line with the highest probability failure located in that path. After failure probabilities have been assigned to all paths in the solution, we aggregate all the probability values into one single combined probability score representing the overall failure level. In situations where a certain consumer could be mapped to its allocated generator using various paths, the path with the lowest failure probability is always chosen. The final result is a real number between 0 and 1, to be later combined in the overall solution score.
- **System Failure Propagation Range:** It doesn't seem feasible handling cascading failures once they have already occurred using the Tabu Search approach, this is because cascading failures have to be responded in real time or the consequences could be potentially catastrophic. The Tabu Search encoding and search don't seem particularly suited for the handling of cascading failures,

even though it can be used to prevent them as explained above. So in order to handle cascading failures once they have occurred as opposed to preventing them we use the same approach we implemented for the default/basic decision model discussed on section 3.2. So when we evaluate solutions using Tabu Search we don't include this criteria in the score.

The final score to be computed for the solution is a number between 0 and 1000. The higher the score, the lower quality the solution is. To combine all the five performance measure scores into this overall score, we simply multiply each one of them by 200 and add them together. By doing that we are giving equal weight (20%) to every measurement. It is also possible to give higher weight to certain performance measures over others to emphasize them in the Tabu Search. For example, a researcher can decide to prioritize the effect that total time outage and power quality have on the score by assigning them weights of 30% each and assigning the other three measurements only 10% each. This will make those measurements have a greater effect on every solution's score. This property of the framework that allows users testing a decision model to prioritize a certain performance aspect when making decisions, is an ideal tool to compare how well different decision models perform when optimizing specific performance measurements.

3.4.2.5. Stopping Criterion

The stopping criterion is simply a number of iterations that the search will run, this is regulated by a tuning constant ***numOfIterations***. The longer the number of iterations the search runs, the more chances to find higher quality solutions but also the more time it takes to make

decisions. By fine tuning this constant an adequate balance between solution quality and performance can be determined.

3.4.2.6. Move Set

The moves that the tabu search performs in the solution consist in selecting a particular consumer and changing one or more of the generators that are currently supplying it. The way generators are replaced is by substituting one of the existing generator indices currently used in the solution by another generator index, as established all the indices are taken with respect to the array of currently available generators. For normal moves, the consumer group, the generator index within that consumer's domain to be replaced and the generator we are replacing it with, are all chosen randomly.

For a given solution x, the neighborhood N(x) that gets generated whenever we do one of these simple generator replacements has a size of ***numGeneratorsAvailable* + 1**, the extra 1 is added because there is a possibility of also replacing that generator with a blank. In practice, the neighborhood of a particular move could be even smaller since some generators couldn't be used as a replacement as they would break the solution's integrity. Every time a move is performed, the neighborhood is generated and all solutions are evaluated using the objective function to find the most optimal solution that will be used for the next Tabu search iteration.

The most optimal solution will be taken unless its currently marked as forbidden in the Tabu list. If that is the case, then the closest solution to the optimal that isn't in the Tabu list will be chosen. Once a solution is chosen, it will be used to generate a new neighborhood for the next iteration.

These kind of moves can of course lead to low quality solutions like one where a consumer doesn't have its full outage demand served or actual invalid solutions where we map consumers to generators that don't have a viable physical path through which to transfer the power to the consumer. Despite that, they are necessary to make sure that we are exploring all areas in the search space. Forcing the moves to valid and high quality solutions only would restrict the search too much. That said there are situations when we want to make special moves that are careful and deliberate to move the resulting solution into a valid one or even purposefully increase its quality. Keeping this balance between normal moves and special moves is a very important issue for the success of the Tabu search exploration, this balancing strategy is regulated by the interaction between the different levels of memory structures, a more detailed discussion on the way the short term memory, midterm memory and long term memory interact to achieve this balance follows.

3.4.2.7. Diversification and Intensification Criteria

In this section we go in to details on how the different layers of memory structures implement intensification and diversification. We also discuss the aspiration criteria used to sometimes override these selection/filtering strategies.

First of all, the short term memory is the most frequent mechanism used to prevent revisiting recent solutions. By placing all recent visited solutions in the tabu list we forbid recently made moves for a short amount of time. When the tabu expires, the ban on the move is removed and we can perform it again.

This kind of strategy helps us make sure we search the current local space intensively, so basically this is part of the intensification strategy, since the moves only alter a solution slightly,

the change is not big enough to really move out of the current explored area, but it does help make sure we are not revisiting always the same places in that area. The reason the moves are only forbidden for a short time is that we sometimes need to revisit the same move/solution since making a different post move could lead to other new solutions in the same local space.

The short term memory only prevents moves that go into recently visited solutions disregarding any concern for whether they are moving us to a valid solution or not, also the short term memory shows no concern on whether the solution is high quality or not. Limiting ourselves to the use of only the short term memory would only allow slight moves that don't change the solution very much and hence intensify local searches, but in many cases it would also promote solutions getting stuck in a local optimal.

Despite the fact that the short term memory provides some level of intensification, sometimes we want to make sure we give special attention to particular areas in the search space that look very promising. In order to do that, we have a midterm memory that is in charge of looking for common patterns between high quality solutions. Once the midterm memory decided that a pattern is frequent enough to correlate it with the quality of those solutions, it can fix that pattern for a certain number of iterations in the Tabu search.

For example, if the midterm memory determines that a common attribute of many high quality solutions is that generator 7 is always assigned to Consumer 2 and generator 3 is always assigned to Consumer 11, then it can force solutions to include those attributes for a certain period of time. This way, we are intensifying the search by fixating on the section of the search space that conforms to that pattern.

Besides the intensification strategies, in order to avoid getting stuck in a local optimal, we need to complement them with proper diversification strategies. Our diversification strategy is implemented in two levels. In the first level we will forbid moves that create invalid solutions, for a certain amount of time, that is certain moves/solutions that are invalid or low quality will be forbidden. For example, if a given solutions is mapping generators to consumers despite the generators not having enough power to supply those consumers or if it maps resources in a way that creates very high cost, these moves will be forbidden selectively to help the search move to a different search space for some time.

This process diversifies the search and allows the exploration of the different areas of the global search space. Note that the long term memory won't be always forbidding bad solutions; it decides whether to forbid certain solutions based on search performance, if many iterations have passed without getting any better solution than the current best, then the long term memory will prevent invalid moves that follow our current pattern to move us to a different search space. Note that level one of the diversification strategy only prevents bad solutions that can be quickly verified to be bad based on power distribution/allocation of resources.

Level two of the long term memory diversification strategy works through a longer period of time and provides some deeper level of diversification by forbidding solutions that are invalid or low quality but are less apparent on first analysis. For example, a solution might assign generators to consumers that satisfy all their power demands in a high quality way on a first analysis, but after further analysis the solution might still be invalid because one or more of the given generators doesn't have a feasible physical path to that consumer, so power transfer is impossible. An unfeasible path could be an inexistent physical path or one that is too unreliable and could lead to faults and outages.

The long term memory will forbid some of the move patterns related to these less obvious situations depending on the search performance. If we are getting stuck on the same solutions for a long time despite of the filtering done by the lower memory levels then it will analyze the solutions more thoroughly and forbid some patterns to make sure we move to more promising search spaces. Note that it can also do the opposite if needed, which is forbid valid solution patterns if the memory system notices that the search is stuck in the same areas that suit a certain pattern of valid solutions, it can forbid those valid moves for some time also for the sake of diversification, so the process works both ways.

3.5. Genetic Algorithms Decision Model Implementation

In this chapter we first present some background information on Genetic Algorithms. We then discuss in details our implementation including two different solution encodings, the workings of the fitness function and the different genetic operators that we used to explore the search space.

3.5.1. Genetic Algorithms Background

Genetic algorithms (GA) are approximate techniques to find solutions to complex optimization problems. These solutions, despite not being exact, are usually very close to the optimal. Genetic algorithms in a way mimic the evolutionary process of nature by trying to select the best out of a group of solutions(natural selection) and by recombining promising solutions into a new one to generate a new "offspring" that could be closer to optimal. Genetic algorithms represent the possible solutions of a problem in the format of chromosomes. A set of chromosomes (candidate solutions) represents a population.

An initial population is determined randomly, then the fitness (the potential of a solution to be the optimal) of all chromosomes/solutions is computed. Based on the fitness of each chromosome, the algorithm then proceeds to either combining different chromosomes to generate new ones or to randomly modify particular chromosomes individually. The former is called crossover or *sexual* reproduction (since it involves a pair of chromosomes) and the latter is called mutation or *asexual* reproduction.

Crossover and mutation are the main operators used by genetic algorithms in order to generate a new evolved population that becomes the input for the next iteration of the algorithm. The idea is that at every iteration the population of solutions should further evolve into fitter solutions that are closer and closer to the optimal, the iterations are usually called generations.

In order to use genetic algorithms to solve a problem, we require first a valid way of encoding a solution so that it can be can be manipulated by the algorithm. For example, a solution could be represented by a group of numbers, characters, boolean values, etc. We also require a way to evaluate how good a given solution is, therefore we must supply a fitness function that analyzes a chromosome and determines how fit it is with respect to the problem we are attempting to solve. Additionally, we require appropriate genetic operators (crossover and mutation) which should be tailored specifically to the problem, because the performance of the GA and the quality of the solutions we obtain will greatly depend on them. Last but not least, we need a termination criterion that could be simply a limit on the number of generations or iterations.

3.5.2. Solution Implementation using Genetic Algorithms

3.5.2.1. Chromosome Encoding

When dealing with possible ways of encoding the solutions for the GA to work we must take into consideration that a good encoding must satisfy the following conditions:

a) The encoding should have the potential to represent all possible solutions in the solution space (including all optimal solutions).

b) The encoding should be appropriate with respect to the genetic operators, meaning that the operators should be able to manipulate the chromosomes in order to explore successfully the solution space and be able to gradually approach the optimal.

Based on this, we could reuse the same encodings presented on section 3.4 for Tabu Search. These two possible encoding implementations are appropriate and can be reused for our genetic algorithm approach.

3.5.2.2. Chromosome Initialization

The procedure to initialize a chromosome differs slightly depending on which of the two solution encodings we intend to use. For solution encoding A we initialize the chromosomes by generating ***maxGeneratorsPerConsumer*** random numbers in the range [0 : ***numGeneratorsAvailable***] for every consumer group. Whenever we obtain the value 0, this will represent a consumer that is assigned to less than the maximum number of generators and will appear as trailing blank genes at the end of the consumer group. For solution encoding B we must first determine the consumer group order, to do this we generate random numbers in the

range [1 **:** ***numConsumersWithOutages***] and store the obtained values as the header. We then proceed to do the same process we did for encoding A.

Since values are determined randomly, sometimes the results can be invalid for the representation, to handle that we adjust the solution every time we get an invalid value to restore its integrity just like we did for the Tabu Search approach.

3.5.2.3. Fitness Function

Since we are using the same solution encodings as the Tabu search approach in section 3.4, it follows that the way to evaluate the fitness of a given solution is by using the same scoring technique as in the objective function discussed for that approach. The goal of the GA is to minimize the score, in order to do this we make use of the fitness function to determine the value of a chromosome. Since each chromosome represents a resource allocation strategy, the fitness function analyzes the chromosomes in order to determine their score based on the performance measures described in section 2.5. So the fitness function basically receives a chromosome as input and returns the overall performance score as output. Again, the process followed to evaluate the chromosomes is the same process used to evaluate solutions that was used for the Tabu Search decision model, so the score of every solution is in the range 0 to 1000.

3.5.2.4. Mutation Operator Implementation

The mutation operator performs mainly swaps between the genes of a given chromosome. These swaps can be both, intelligent or just random; we will discuss both approaches in this section. In order to determine whether to use the random or the intelligent approach we set a parameter indicating a probability, and then based on that probability we

determine if we should use the random or the intelligent approach every time the mutator is called. It's very important to tune this parameter, and in fact extensive time has been taken during testing to find the optimal adjustment of the tuning variable, in order to reach the proper balance between randomness and intelligence.

The problem of following a purely intelligent approach is that it might constrain too much the solutions we get and might make us get stuck on local optima. The problem of following a purely random approach is that for large search spaces the genetic algorithm could take a huge amount of time in order to even fairly approach the global optimal. Therefore as mentioned above it's very important to find a balance between both approaches to obtain the best results.

The first approach is purely random and basically consists of choosing two random numbers, the first in the range [0, ***numConsumersWithOutages***] and the second in the range [0, ***maxGeneratorsPerConsumer***]. So basically we are randomly selecting one of the consumers represented in the chromosome and also one of the generators assigned to it. Then we generate a random third number in the range [0, ***numGeneratorsAvailable***] and replace the generator determined by the second number by the new generator determined by the third number. In other words we are replacing one of the generators assigned to a given consumer by a new random generator, so we are simply changing the value of one gene in the chromosome. This kind of move is ideal for a mutation operator since it only changes the chromosome slightly. Another possible move that can be used when we require a more aggressive mutator is to swap two generator values from two different consumers represented in the chromosome, this changes the chromosome more as it implies changes to two genes instead of one.

The second approach which is more intelligent uses the information obtained from the API to make a more informed choice that would lead to a better quality solution. For example, we could still determine the consumer and the generator to be replaced randomly, but we could decide on the new generator to assign to the consumer based on an intelligent criteria like making sure the new generator fully satisfies the missing demands for that consumer.

Alternatively, we could use intelligence to determine the generator we want to replace by choosing the generator that is supplying the least energy to the consumer; or even choose the consumer to which we are reassigning the generator intelligently by selecting the consumer that has its demands less satisfied and could better benefit from resource reallocation. It is important to find a good balance between random moves and intelligent moves, overusing intelligent moves would constrain the mutations too much and prevent the process from flowing freely into various areas of the solution space, resulting in a local optimal. Therefore in order to properly regulate the use of the different kind of mutation moves we use a tuning constant that can be adjusted during experimentation.

3.5.2.5. Crossover Operator Implementation

There are two crossover operators provided in this implementation, one performing a uniform crossover and another that is based on a cross point approach. Because of the solution encoding we have chosen, we cannot apply any of these two approaches purely, but we need to do extra chromosome processing to ensure the offspring after crossover represents a valid solution within the search space. As usual we use a probability parameter that can be fine tuned in order to determine how often to use each different approach, it's also possible to adjust the parameter so that a particular approach is used exclusively for experimentation. We will now

describe both crossover approaches and how they work for the two different solution encodings we previously defined.

3.5.2.5.1. Uniform Crossover

When applying uniform crossover to the second solution encoding presented, we simply choose a consumer from the first chromosome and move it into the offspring, then choose the next consumer from the second chromosome and also move it into the offspring, repeat this process until all consumer resource assignment info is in the new chromosome 3 resulting from the merge of 1 and 2. The resulting chromosome 3 would have all the information for the consumers placed at odd positions in chromosome 1 and those placed at even positions in chromosome 2. Unfortunately, this merge doesn't necessarily preserve chromosome integrity, because some of the generators that are assigned to consumers would be invalid, for example if the ***maxGeneratorsPerConsumer*** is 3 and the ***numConsumersWithOutages*** is 10 and ***numGeneratorsAvailable*** is 20, and if generators 7, 3, 8 were assigned to consumer 1, that means generators 7 and 3 can't be assigned to any other consumer otherwise we would generate a solution out of the search space of valid solutions. So after the uniform crossover we must perform additional processing to replace all invalid generator assignments and enforce the chromosome integrity of the offspring. Figure 20 shows the process of uniform crossover applied to solution encoding B, for the case when ***numConsumersWithOutages*** is 4, ***numGeneratorsAvailable*** is 18, and ***maxGeneratorsPerConsumer*** is 3.

When dealing with encoding A we can apply the same process of choosing a consumer from each chromosome alternatively, but because the consumers are ordered randomly in this encoding we do it in the following way:

1) We extract the info of the first available consumer in chromosome one, if that consumer isn't yet present in the new chromosome, then we insert the info in the next available section within the new chromosome.

2) If the consumer is already present in the new chromosome then we repeat step 1 by selecting the next consumer until one that isn't present in the new chromosome is found.

3) We then do the same procedure for chromosome 2.

4) We keep repeating this procedure until the new chromosome has all the information for all the consumers and their resource mappings.

This approach guarantees that for every consumer in the resulting chromosome that originated from chromosome 1 there will be a matching consumer originating from chromosome 2. This approach keeps chromosome integrity at the consumer level, but still just like in the previous case we might break chromosome integrity based on the generators assignments. We can use the same process we used for implementation 1 in order to repair the chromosome, Figure 21 shows the process of uniform crossover applied to solution encoding A, for the case when ***numConsumersWithOutages*** is 4, ***numGeneratorsAvailable*** is 18, and ***maxGeneratorsPerConsumer*** is 3.

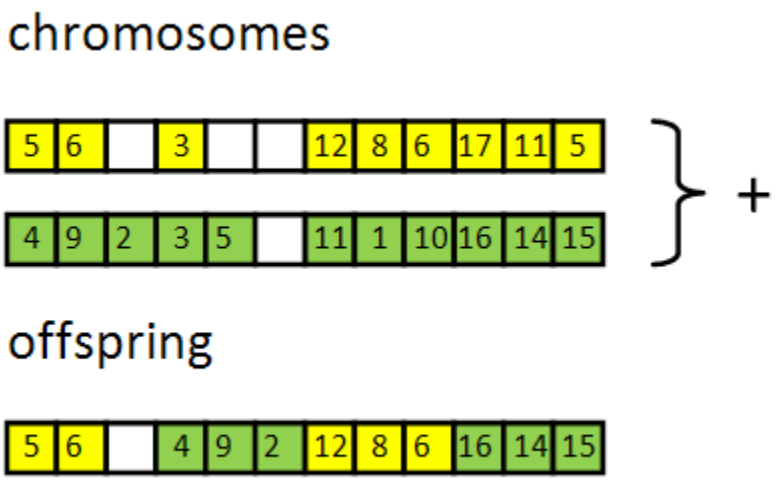

Figure 20: Uniform Crossover Applied to Chromosomes Using Encoding B

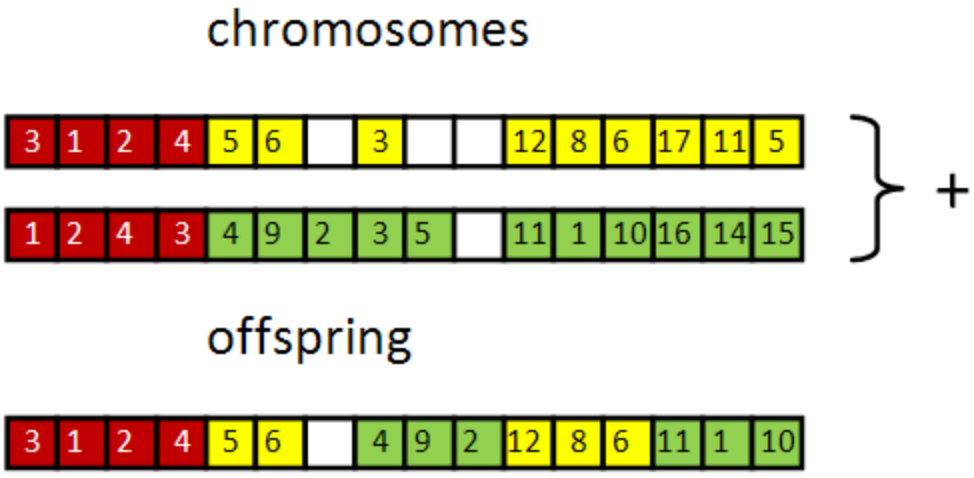

Figure 21: Uniform Crossover Applied to Chromosomes Using Encoding A

3.5.2.5.2. Crosspoint Crossover

When applying crosspoint crossover to solution encoding B, we simply combine the first half of chromosome 1 with the second half of chromosome 2. Therefore the consumers info in the offspring will contain as many consumers from chromosome 1 as from chromosome 2. Like in the case of the uniform crossover, this merge doesn't necessarily preserve chromosome integrity, because some of the generators that are assigned to consumers would be invalid. Therefore, just like after the uniform crossover we must perform additional processing to replace all invalid generator assignments and enforce the integrity of chromosome 3. Figure 22 shows the process of crosspoint crossover applied to solution encoding B, for the case when ***numConsumersWithOutages*** is 4, ***numGeneratorsAvailable*** is 18, and ***maxGeneratorsPerConsumer*** is 3.

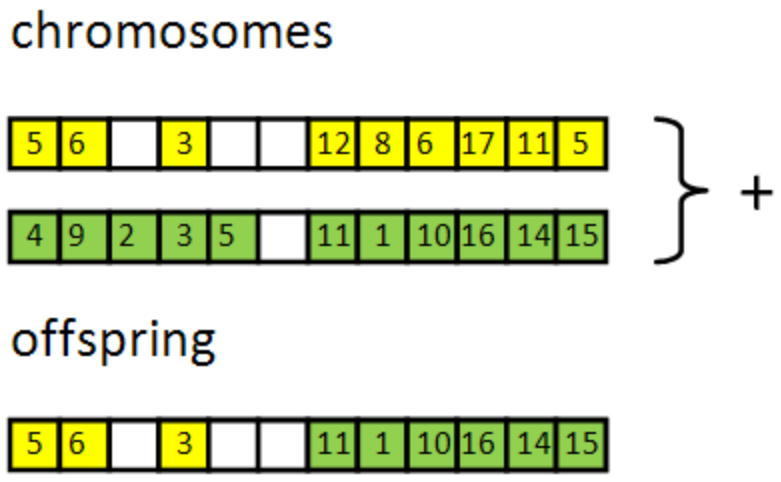

Figure 22: Crosspoint Crossover Applied to Chromosomes Using Encoding B

When dealing with encoding A we can't simply do the previous process, since the consumers are ordered randomly within the chromosome, following the previous approach would break chromosome integrity as we could end up with duplicate consumers. In order to prevent this we apply the process in the following way:

1) We extract the info for ***numConsumersWithOutages*** / 2 consumers from chromosome 1 and insert it within the new offspring.

2) We extract the information of all consumers in chromosome 2 that aren't already present in the offspring.

3) We add that missing consumer info to complete the offspring.

This approach guarantees that for every consumer in the resulting chromosome that originated from chromosome 1 there will be a matching consumer originating from chromosome 2. This method also preserves chromosome integrity at the consumer level, but still just like in the previous case we might break chromosome integrity based on the generators assignments. We can use the same process we used for implementation 1 in order to repair the chromosome, Figure 23 shows the process of crosspoint crossover applied to solution encoding A, for the case when ***numConsumersWithOutages*** is 4, ***numGeneratorsAvailable*** is 18, and ***maxGeneratorsPerConsumer*** is 3.

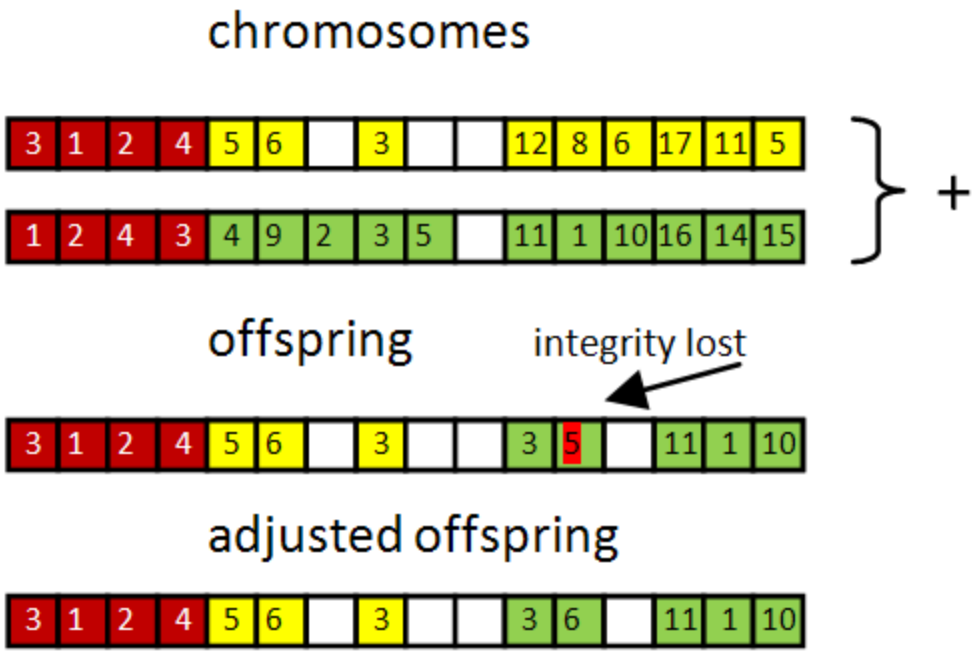

Figure 23: Crosspoint Crossover Applied to Chromosomes Using Encoding A

4. TESTING AND EXPERIMENTATION

In this chapter we first present some preliminary testing we did to verify the basic functionality of the simulator. We then present our detailed testing results when comparing all the different decision models and settings applied to large microgrid samples running in the cloud. We also give some background on Amazon Elastic Cloud services which is the virtualization environment used for the tests.

4.1. Preliminary Testing

In this section we present some preliminary testing we have performed, in order to evaluate the workings of the simulator on a simple test case. The purpose was to evaluate if all the different agent layers, decision levels and model design work properly when integrated in practice. In order to do that we conceived a case sample based on situation 1 from the main self-healing issues a decision model should handle, as explained in section 3.2.1. The management agent is configured to use the default decision model detailed in section 3.2. When evaluating this simple test case, we are only concerned to see if the simulator is able to model the situations, and to confirm that when problems occur, the agents are able to collaborate and interact in order to return the system to a healthy state. In this section we aren't concerned about measuring performance or comparing different decision models. In the next section we experiment using a large interconnected grid sample, we apply different decision models and make a detailed comparison of their performance.

The test sample is used to evaluate if the agents respond well to outage situations. We have designed the topology shown in fig.24 in the simulator, the agent mappings are specified with a letter A over the element they are assigned to. As explained on section 2.4, one of the main features of our simulation is that the agents are dynamically generated in the supporting agent platform (JADE) as soon as the simulation launches into run mode. The system automatically determines what kind of agent is required for each node, if the agent was placed on a consumer it knows it should create a User agent, on a generator a DER agent, on a device a Device agent, on or near a power line a Control agent.

Our test sample consists of a main electrical generator and two alternative generators, the main generator appears on the top and the two alternative generators appear at the bottom. We have 5 consumers which are displayed as houses, each one of these consumers requires between 100 kW of power to 200 kW of power at any given time, the main generator is set to produce 1800 kW of power, the alternative generators are set each to produce a maximum of 500 kW at any given time. Besides the consumers and generators, we have devices between them; In our sample there are 6 relays and one transformer, the relays are required in order to be able to route power.

Every consumer is associated with a user agent and every single device is associated with a device agent, only the two alternative generators are associated with DER agents, as we want the users to establish power negotiations with them whenever there is an outage or the power from the main electrical distribution is insufficient. The control agents are represented with the A symbols that appear on top of power lines, each control agent monitors a section of the grid with all the components belonging to that section.

If there is no control agent monitoring a certain component its status won't be known and its data won't be delivered to the management agent, hence having control agents monitoring all relevant sections is very important. We place enough control agents on our sample to guarantee the monitoring of the whole microgrid.

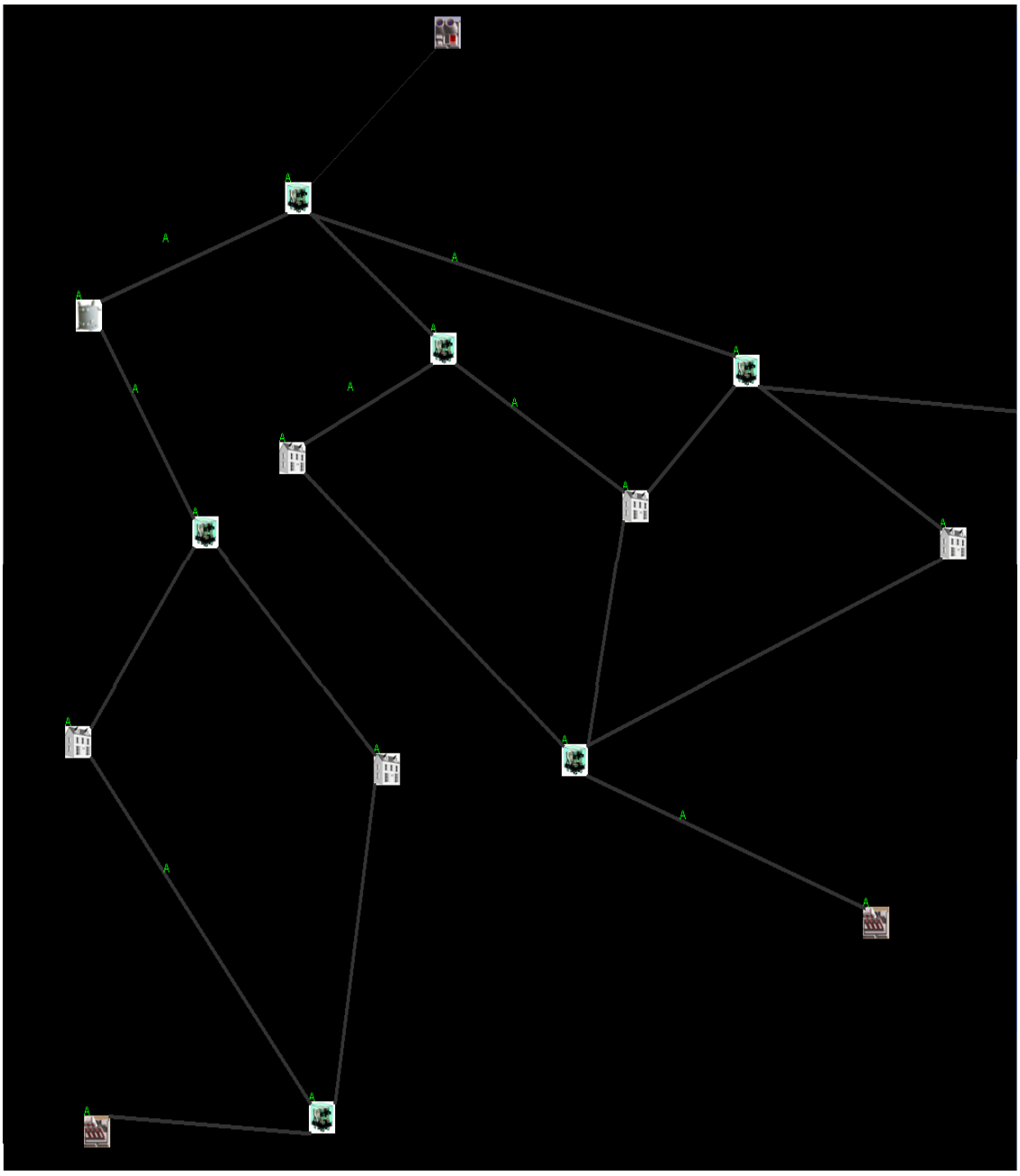

Figure 24: Simple Test Sample Implemented in the Simulator for Preliminary Testing

During our tests, when we initially run the simulator, all consumers have their power demands fully satisfied. We then experiment by forcing the failure of different power lines in the topology. The failure of the power lines causes consumer outages which are represented by red blinking of houses/consumers within the simulator. In our tests we observed that every outage that occurred was promptly corrected by either a change of the power distribution in the relays that is the main distribution circuit intelligently reroutes power so that the missing demand reaches the consumer through an alternate path, or one of the available DERs supplies the missing demand to the consumer. We also observed that DERs would stop supplying power to the user as soon as the power from the main distribution to the consumer was reestablished.

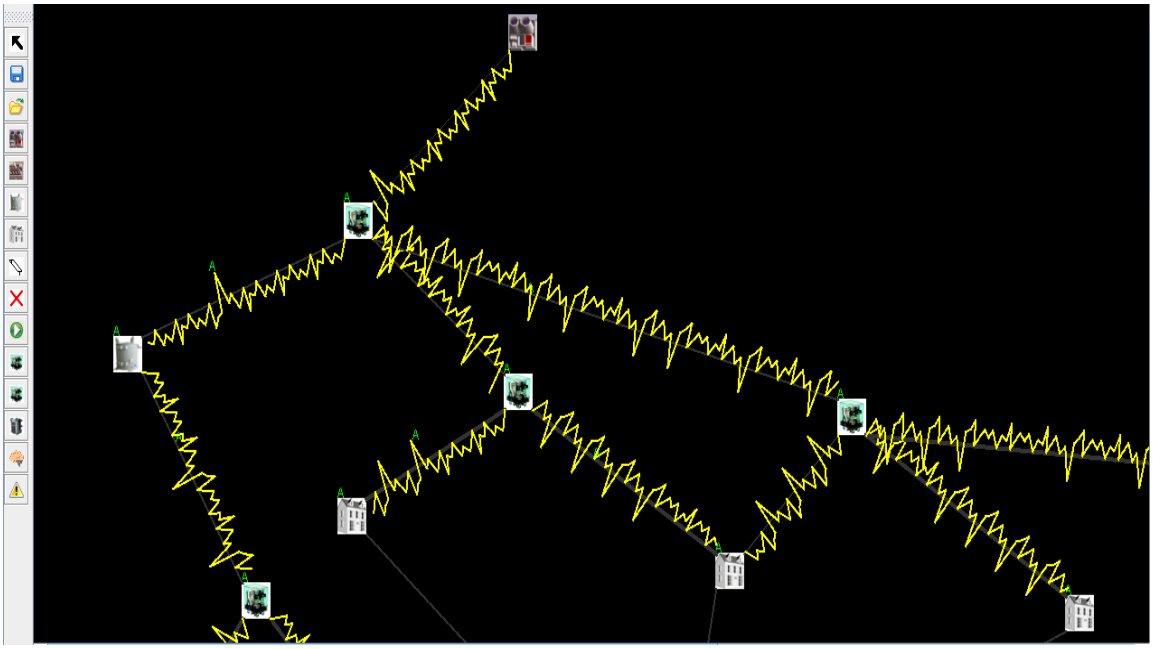

Figure 25: Test Sample Running in Simulator

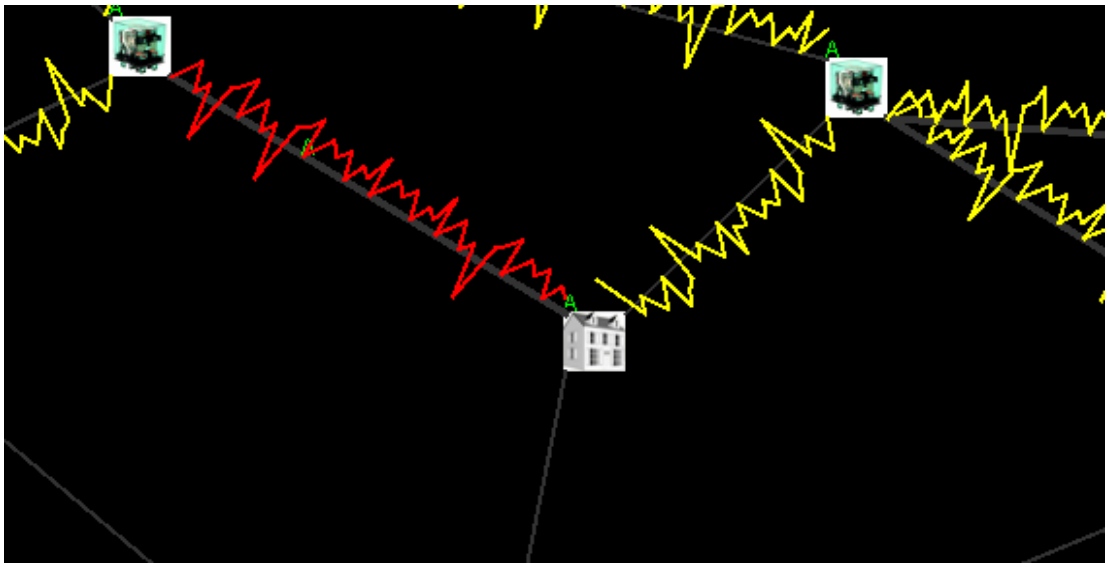

Figure 26: Line Failure Forced During simulation

In the case of the consumers connected to the main electrical company through a single line that failed, that is there was no possible way to reroute power to them, we observed that they would maintain their DER lease indefinitely, as expected. Through our different tests we saw different behavior of the agents, but what was constant was a prompt response to solve outages and a successful return of the system to a healthy state either by power rerouting or power negotiations between DERs and user agents. This shows that the simulator can run our default decision model and perform self healing in the designed microgrid successfully. Figs 28 and 29 show the simulator GUI when an outage has been detected and the response that follows, that is power is routed from alternate sources in order to supply the outage.

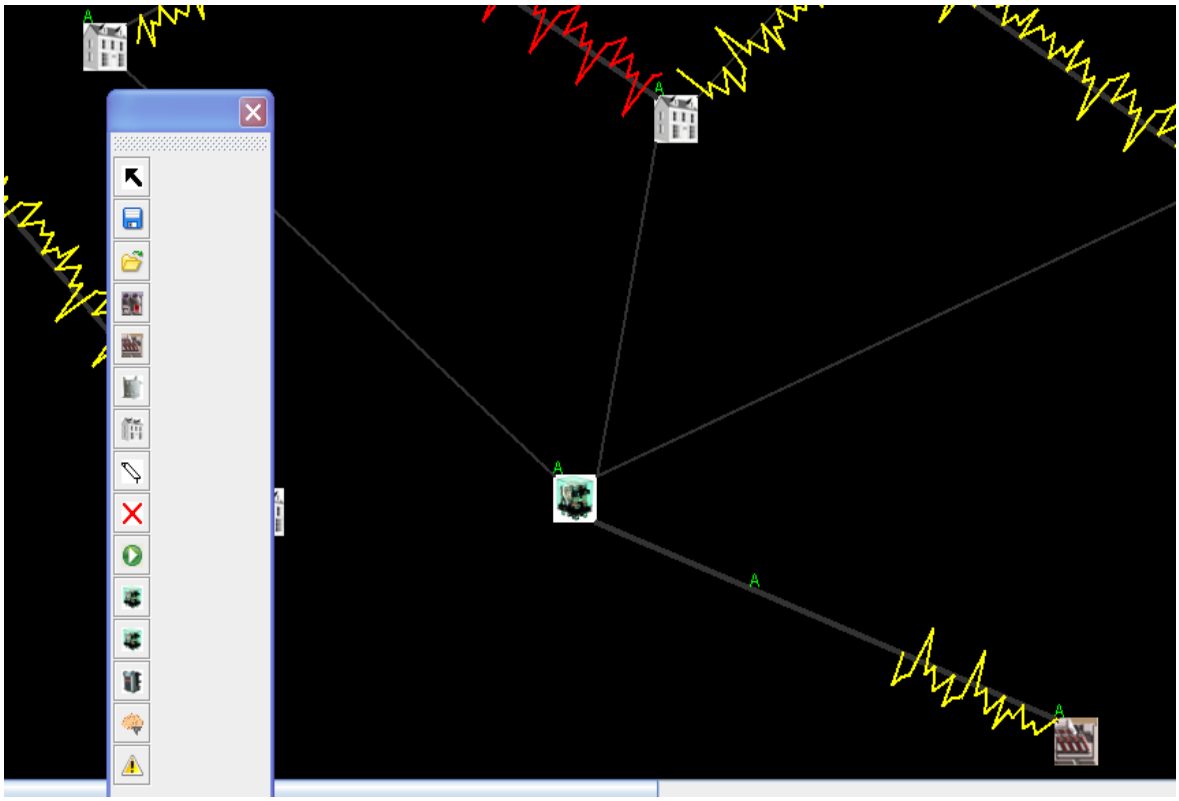

Figure 27: Power from Alternate Resource is Activated to Supply Outage

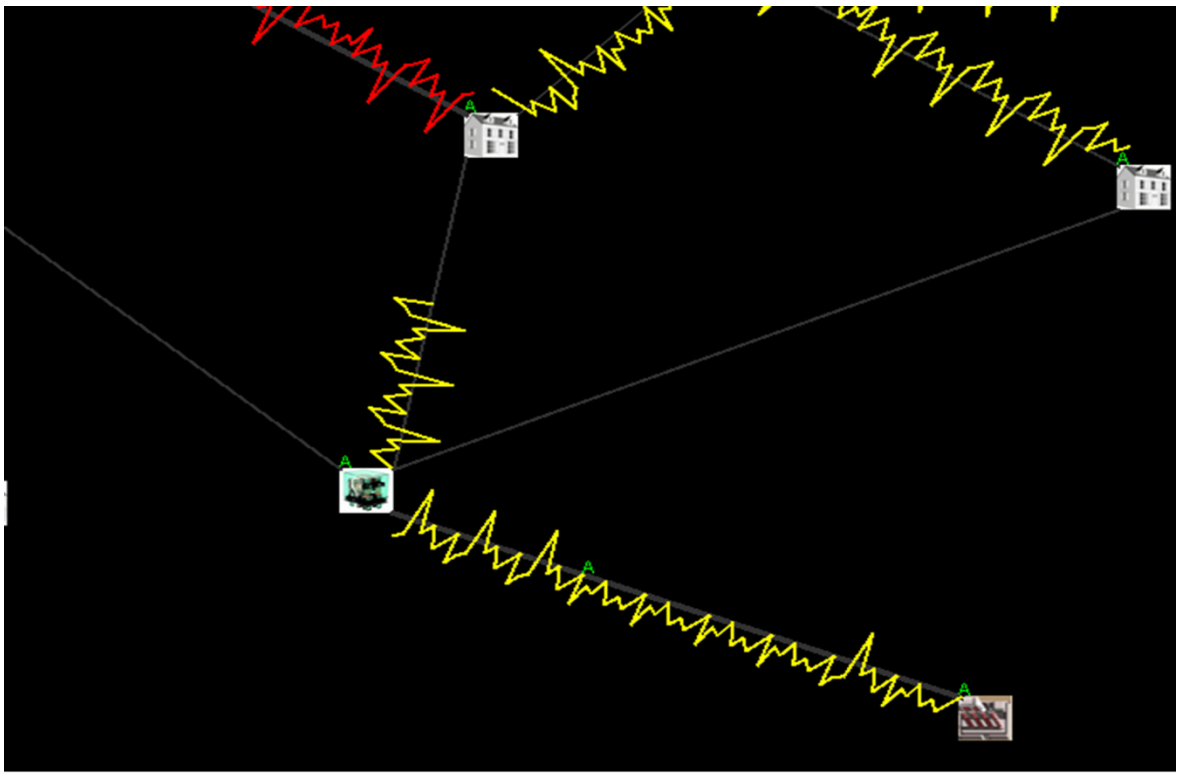

Figure 28: Power from Alternate Source is Routed to Supply Outage

4.2. Testing and Comparing Different Decision Models

For the testing of the different decision models in the simulator we created fairly large scale samples using the scaling capabilities of the simulator. These samples consist of the interconnection of many microgrids to create a larger grid. The microgrids themselves are fairly large containing at least 80 nodes, including consumers, devices and generators. Two kinds of macrogrid samples are generated by interconnecting the microgrids, a 25-microgrid sample and a larger 50-microgrid sample.

As described in previous sections, every microgrid in the simulator runs in its own Jade container, so when running these large samples every microgrid runs on a separate computer with independent resources. In order to implement these tests a computer network environment is required where every independent computer hosts a microgrid container with all its agents. To implement this in our testing we used the Amazon elastic cloud services, as a virtualization environment. This service allowed us to create as many machine instances as we required easily, and also connecting them together and monitoring their resources on demand. In section 4.2.1 we describe the Amazon EC2 service in more details, in section 4.2.2 we describe how the performance measurements described in chapter 2 are normalized into proper quality scores and finally in section 4.2.3 we discuss our testing procedure and detailed results.

4.2.1. Amazon Elastic Cloud Services

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers. It uses a simple web service interface that allows developers to configure capacity and provides them with complete control of the environment and computing resources. Amazon EC2 reduces

the time required to obtain and boot new server instances to minutes, allowing developers to quickly scale capacity, both up and down, dynamically as computing requirements change. Amazon EC2 changes the economics of computing by allowing developers to pay only for capacity that they actually use. It also presents a true virtual computing environment, allowing developers to use web service interfaces to launch instances with a variety of operating systems, load them with their custom application environment, manage the network's access permissions, and work on as many or few systems as desired.

To use Amazon EC2, all that is required is selecting a pre-configured Amazon Machine Image (AMI) which specifies the OS and supporting software installed (like for example Windows 2008 server 32 bit with MS SQL server), configuring security and network access on the instance, then one can launch and monitor as many instances as required. Finally one can connect to any of the instances by using services like SSH or remote desktop connection and upload his data to the virtual machine and configure it.

The main advantages of Amazon EC2 are:

- **Elastic:** Increase or decrease capacity within minutes, not hours or days. One can commission one, hundreds or even thousands of server instances simultaneously.
- **Completely Controlled**: Developers have complete control of the instances and can interact with them as they would with any machine. Instances can be stopped while retaining the data on a separate volume for later use.
- **Flexible**: Developers have a choice of multiple instance types, operating systems, and software packages. For example, the choice of operating systems includes numerous Linux distributions, and Microsoft Windows Server.

- **Reliable**: Offers a highly reliable environment where replacement instances can be rapidly and predictably commissioned. The service runs within Amazon's proven network infrastructure and data centers.
- **Secure**: Includes web service interfaces to configure firewall settings that control network access to and between groups of instances.
- **Inexpensive**: Developers pay only for the resources that they actually consume, like instance-hours or data transfer.

4.2.2. Performance Measurement Quality Scores Estimation

When evaluating the performance of a microgrid we use the performance measures described in chapter 2, but some of these measurements are hard to compare in their raw state, for example when evaluating and comparing total outage time, these values are expressed in simulation time units and often result in huge numbers that are hard to display or compare. Therefore before comparing and analyzing the results we try to normalize every performance measure result to a quality score between 0 and 100. Some of these score are more optimal when high and others when low, for example a high total outage time or total cost score is not good but a high consumer satisfaction score is ideal. In this section we describe the normalization process for each of the performance measures:

- **Total Outage Time:** In order to normalize this value we first compute the total outage for each consumer independently. We divide the life of a consumer in a group of time slots, each time slot continues until there is a change in the supply to the consumer. For example: if a consumer requires 200W of power to supply its needs, and it starts with that power, later on due to a failure it is only getting 100W supplied, and then later on it

receives an extra 50W from another source reducing its outage to 50W. That creates three different time slots, where supply is 200W, 100W and 150W respectively. In order to compute the overall outage time we then group the slots into sets of two neighbors and compute the performance of every pair as the time the consumer experienced outage divided by the total time. The results of every pair get averaged to a real number between 0 and 1 representing the final result for that consumer. That number is multiplied by 100 to obtain the quality score for that independent consumer. To obtain the overall quality score for the microgrid we add the scores for all consumers and divide by the number of consumers in the microgrid.

- **Total Cost:** In order to normalize this value we first compute the total cost for each consumer and generator independently. We divide the life of a consumer and a generator in a group of time slots, each time slot for a consumer continues until there is a change in the supply to the consumer, for the generators each time slot continues until there is a change in the amount of power the generator is generating. For example: if a generator is generating 200W and then increases its load to 240W of power, that starts a new timeslot when its load increases. In order to compute the overall cost for a consumer we compute a fraction for each one of these time slots. The fraction is equal to the current power outage value multiplied by the cost per outage time unit of that consumer divided by the total power required by that consumer. For the generators the fraction is equal to the amount of power currently being generated in that slot multiplied by the cost of power generation per unit of power for that particular generator divided by maximum possible power that the generator can generate. After the simulation ends, we add all the slot fractions together for a particular consumer or generator together and divide by the total

number of slots multiplied by either the cost per outage time unit for consumers or the cost of power generation per unit for generators. We then obtain a real number between 0 and 1, that number is multiplied by 100 to obtain the quality score for that independent consumer or generator. To obtain the overall quality score for the microgrid we add the scores for all consumers and generators and divide by the sum of the total number of consumers and generators in the microgrid.

- **Consumer Satisfaction Level:** To compute the satisfaction level we assign a maximum value of 40 to the price limit preference, a maximum value of 30 to generator type preference and the remaining 30 points to the generator ranking preference. The satisfaction level just like the previous performance measures is computed with a time slot granularity and then results are combined. To compute the price limit score we divide the average price this consumer is currently paying in the present time slot, this is obtained by computing the average of prices paid to each independent generator supplying the consumer in the current time slot. That average is then divided by the maximum price the consumer is willing to pay, if the result is 1, meaning the values are the same, then the score will be 30. If the value is bigger than 1 then for every 10% of the price limit that is in excess there is a penalty of 3 points, therefore if the average price paid is double the price limit then the score will be 0, the more the average price paid exceeds the price limit, the lower the score will be. If the value is less than 1 then for every 10% of the price limit that is below e there is a bonus of 1 point, therefore if the average price paid is 0 then the score will be 40, the more the average price paid is below the price limit, the higher the score will be.

To compute the generator type preference score we divide the number of current generators supplying this consumer that match the type preference and divide by the total number of supplying generators. This fraction is multiplied by 30 to obtain the score. Concerning the generator ranking preference the process is exactly the same as with the type preference. The final computed score for the satisfaction level of this consumer in the current timeslot is the sum of those three scores giving a number between 0 and 100. To obtain the overall consumer satisfaction level of a particular consumer we simply average the scores from all its time slots.

- **Failure Potential Level:** The score for this performance measure is computed by simply averaging the failure probabilities of all devices and power lines that are currently in use. Since the average will be a real number between 0 and 1, we multiply by 100 to normalize the score.
- **Variability:** To normalize this value we generate a score between 0 and 1 for every electrical component that can be adjusted. This value is computed by dividing all variations that were performed in the component over the maximum number of variations that could be performed.

4.2.3. Testing and Detailed Results

When evaluating results we used two main samples: a 25 microgrid sample and a 50 microgrid sample. The way performance measures are evaluated for the sample macrogrids is by computing performance measures for each independent microgrid, then collecting and aggregating all those results and averaging them to obtain the final scores. The macrogrid samples have been evaluated using seven different healing strategies, the first analysis is to test

them using no decision model or intelligence, then we test them without a decision model set in the management agent but having User agents and DER agents working with full autonomy, this mode allows us to see how the macrogrid performs when just having a supply/demand model carried out in a completely distributed way in each macrogrid ie not having a central authority (management agent) making the decisions.

Then we can test the macrogrid using the default decision model described in 3.2. Next, we test using a Tabu Search Decision Model, there are two possible implementations for Tabu Search as described in section 3.4, both encodings are tested. Finally we also test the genetic algorithm decision model also having two different encodings as described on section 3.5. We perform each one of these seven tests to evaluate every performance measure for each of the two macrogrid samples. Moreover, the Tabu Search and GA decision models are able to prioritize certain performance measures when making decisions, to test this feature we do the previous evaluation for six different cases:

- When all performance measures are equally considered
- When minimizing outage time is highest priority
- When minimizing total cost is highest priority
- When maximizing consumer satisfaction is highest priority
- When minimizing failure probability is highest priority
- When minimizing variability is highest priority

The detailed tests performed follow; figures 29 – 40 show the detailed results for the 25 Microgrids sample and then the 50 Microgrids sample.

Performance Measures	No Intelligence (Agents Disabled)	No DM User/Der Agents full autonomy	Default Decision Model	Tabu Search Decision Model A	Tabu Search Decision Model B	Genetic Algorithms Model A	Genetic Algorithms Model B
Outage Time	87	17	23	45	37	47	53
Overall Cost	92	35	33	63	56	68	75
Consumer Satisfaction	13	42	42	57	62	53	55
Variability	0	25	37	65	65	72	70
Fault Potential	75	67	68	53	48	43	44

Figure 29: 25 Microgrids Sample when Performance Measures are Equally Considered

Performance Measures	No Intelligence (Agents Disabled)	No DM User/Der Agents full autonomy	Default Decision Model	Tabu Search Decision Model A	Tabu Search Decision Model B	Genetic Algorithms Model A	Genetic Algorithms Model B
Outage Time	87	17	23	33	31	44	52
Overall Cost	92	35	33	54	56	67	72
Consumer Satisfaction	13	42	42	36	58	42	50
Variability	0	25	37	64	65	68	68
Fault Potential	75	67	68	62	60	52	45

Figure 30: 25 Microgrids Sample when Minimizing Outage Time is Highest Priority

Performance Measures	No Intelligence (Agents Disabled)	No DM User/Der Agents full autonomy	Default Decision Model	Tabu Search Decision Model A	Tabu Search Decision Model B	Genetic Algorithms Model A	Genetic Algorithms Model B
Outage Time	87	17	23	48	37	52	55
Overall Cost	92	35	33	59	55	65	70
Consumer Satisfaction	13	42	42	46	62	43	45
Variability	0	25	37	68	66	68	68
Fault Potential	75	67	68	60	55	53	48

Figure 31: 25 Microgrids Sample when Minimizing Overall Cost is Highest Priority

Performance Measures	No Intelligence (Agents Disabled)	No DM User/Der Agents full autonomy	Default Decision Model	Tabu Search Decision Model A	Tabu Search Decision Model B	Genetic Algorithms Model A	Genetic Algorithms Model B
Outage Time	87	17	23	52	46	49	55
Overall Cost	92	35	33	74	72	75	71
Consumer Satisfaction	13	42	42	53	67	60	60
Variability	0	25	37	66	65	69	70
Fault Potential	75	67	68	55	53	57	48

Figure 32: 25 Microgrids Sample when Maximizing Consumer Satisfaction is Highest Priority

Performance Measures	No Intelligence (Agents Disabled)	No DM User/Der Agents full autonomy	Default Decision Model	Tabu Search Decision Model A	Tabu Search Decision Model B	Genetic Algorithms Model A	Genetic Algorithms Model B
Outage Time	87	17	23	47	47	53	56
Overall Cost	92	35	33	65	62	68	78
Consumer Satisfaction	13	42	42	51	49	50	50
Variability	0	25	37	67	66	68	67
Fault Potential	75	67	68	46	41	37	35

Figure 33: 25 Microgrids Sample when Minimizing Fault Potential is Highest Priority

Performance Measures	No Intelligence (Agents Disabled)	No DM User/Der Agents full autonomy	Default Decision Model	Tabu Search Decision Model A	Tabu Search Decision Model B	Genetic Algorithms Model A	Genetic Algorithms Model B
Outage Time	87	17	23	49	45	54	57
Overall Cost	92	35	33	73	67	75	79
Consumer Satisfaction	13	42	42	43	49	37	43
Variability	0	25	37	59	55	62	53
Fault Potential	75	67	68	64	57	49	51

Figure 34: 25 Microgrids Sample when Minimizing Variability is Highest Priority

Performance Measures	No Intelligence (Agents Disabled)	No DM User/Der Agents full autonomy	Default Decision Model	Tabu Search Decision Model A	Tabu Search Decision Model B	Genetic Algorithms Model A	Genetic Algorithms Model B
Outage Time	85	23	26	52	42	50	51
Overall Cost	88	39	35	73	59	72	73
Consumer Satisfaction	21	33	32	52	58	48	51
Variability	0	42	49	68	65	72	69
Fault Potential	73	75	70	52	52	41	45

Figure 35: 50 Microgrids Sample when Performance Measures are Equally Considered

Performance Measures	No Intelligence (Agents Disabled)	No DM User/Der Agents full autonomy	Default Decision Model	Tabu Search Decision Model A	Tabu Search Decision Model B	Genetic Algorithms Model A	Genetic Algorithms Model B
Outage Time	85	23	26	45	43	52	57
Overall Cost	88	39	35	53	63	77	72
Consumer Satisfaction	21	33	32	36	42	33	37
Variability	0	42	49	65	65	63	65
Fault Potential	73	75	70	45	44	47	54

Figure 36: 50 Microgrids Sample when Minimizing Outage Time is Highest Priority

Performance Measures	No Intelligence (Agents Disabled)	No DM User/Der Agents full autonomy	Default Decision Model	Tabu Search Decision Model A	Tabu Search Decision Model B	Genetic Algorithms Model A	Genetic Algorithms Model B
Outage Time	85	23	26	56	55	63	59
Overall Cost	88	39	35	53	57	62	59
Consumer Satisfaction	21	33	32	44	39	27	33
Variability	0	42	49	65	65	67	65
Fault Potential	73	75	70	46	42	49	53

Figure 37: 50 Microgrids Sample when Minimizing Total Cost is Highest Priority

Performance Measures	No Intelligence (Agents Disabled)	No DM User/Der Agents full autonomy	Default Decision Model	Tabu Search Decision Model A	Tabu Search Decision Model B	Genetic Algorithms Model A	Genetic Algorithms Model B
Outage Time	85	23	26	55	53	66	62
Overall Cost	88	39	35	58	61	69	72
Consumer Satisfaction	21	33	32	57	61	48	47
Variability	0	42	49	63	64	67	66
Fault Potential	73	75	70	46	43	52	45

Figure 38: 50 Microgrids Sample when Maximizing Consumer Satisfaction is Highest Priority

Performance Measures	No Intelligence (Agents Disabled)	No DM User/Der Agents full autonomy	Default Decision Model	Tabu Search Decision Model A	Tabu Search Decision Model B	Genetic Algorithms Model A	Genetic Algorithms Model B
Outage Time	85	23	26	53	57	59	63
Overall Cost	88	39	35	58	63	75	71
Consumer Satisfaction	21	33	32	52	46	39	41
Variability	0	42	49	65	64	65	65
Fault Potential	73	75	70	42	45	53	52

Figure 39: 50 Microgrids Sample when Minimizing Fault Potential is Highest Priority

Performance Measures	No Intelligence (Agents Disabled)	No DM User/Der Agents full autonomy	Default Decision Model	Tabu Search Decision Model A	Tabu Search Decision Model B	Genetic Algorithms Model A	Genetic Algorithms Model B
Outage Time	85	23	26	53	55	61	59
Overall Cost	88	39	35	57	64	68	66
Consumer Satisfaction	21	33	32	43	45	43	41
Variability	0	42	49	67	63	65	63
Fault Potential	73	75	70	45	45	57	54

Figure 40: 50 Microgrids Sample when Minimizing Variability is Highest Priority

From the results presented for the two different scenarios we can observe that the simulator allows the evaluation of those same scenarios using different decision models and performance measures. Moreover, it is possible to experiment by giving a higher weight to specific performance measures; this allows potential decision makers to emphasize the measurements that are more crucial to their goals. This was thoroughly assessed in the experimental testing performed as can be seen in Figures 30 and 36 which presents the results of testing when performance measures are equally considered for the 25 Microgrid and 50 Microgrid samples respectively. Figures 30- 34 and Figure 36-40 show the same detailed results when giving more weight to a specific performance measures during testing.

We can also observe from the experimental results presented that a particular decision model can be very efficient on maximizing performance for a particular performance measure but produce a lower quality solution with respect to another performance measure. That is why these simulations show considerable potential to show a decision maker what would be the best choice for a reasoning strategy to be used in a particular scenario, in order to maximize the system performance for those aspects that affect the most their particular goals.

5. CONCLUSIONS AND FUTURE WORK

The framework we presented in this research work addresses many of the shortcomings of traditional centralized and decentralized schemes by utilizing a hierarchical distributed control scheme. It also supports direct comparisons among different decisions models, by implementing them in a separate independent layer from the agents.

The framework was further proved by developing a simulator based on it. A major goal of our simulator was to support the appropriate decision models for the various self-healing scenarios that can occur. We successfully allowed the configuration of different decision models, to compare how agents perform in the same case scenario when using different reasoning processes. We achieved that by using a topology independent framework that places the physical aspects of the power grid and the actual agents in separate independent layers. We also successfully implemented an API that allowed researchers to develop their own models and easily integrate them in the simulator for testing. The only requirement for researchers was to use the Java language when writing the decision model. By providing only their required data on request to decision model developers we allowed them to focus on making decisions rather than being concerned about where the data is stored or the different sources it is obtained from. Other important features successfully provided in our implementation included innovations in supporting dynamic agent generation, scalability and control of testing environment.

The simulation system implemented in this work is a flexible and highly scalable system. We proved this to a fair degree in our testing and experimentation by creating large macrogrid samples consisting of interconnected microgrids running on separate Jade containers and having all these containers running on different machines in the cloud. Also the simulator was provided

with a convenient GUI that allowed researchers to create their own grid topologies and intelligent agent mappings through a convenient and easy to use graphical interface. The GUI also allows researchers to successfully manipulate the simulation at run time through actions such as shutting down devices or forcing component failures.

Through our testing and experimentation we found the simulator answers the questions we set as our goal to a fair degree. By testing the same scenario using different decision models and measuring the performance using the independent support within the simulator, it was possible to get scores and see how different decision models affected the different aspects of the grid and to compare them. One of our main goals was to create a simulator that could answer questions that the previous simulators were unable to answer, like:

- What decision model would be most appropriate for handling self healing in particular situation
- What decision model can guarantee that the reliability and efficiency is maximized for a power system
- How to find a configuration that avoids potential cascading failures while still deliberate enough to maintain the efficiency in the system overall

The experimentation we did with the simulator showed that it has the potential to generate answers to those questions. The results presented showing scores for each different performance measure and decision model used holds the key to find answers to those questions. Our simulator also allows researchers to emphasize a particular performance measure to evaluate how decision models affect it in particular, we tested all scenarios using different cases where each different performance measure is given a greater weight than the others at a time.

Our future works will focus on redesigning the simulator to make it an open source software. The current version has a very complex design due to all changes that had to be done that we weren't aware of in our initial design. This has been an ongoing research for many years, so through this journey we faced many new unexpected challenges as we were learning about the smart grid, and this eventually made the simulator design very complicated.

After we redesign the simulation we are planning on working on creating a whole suite of testing scenarios and also developing other decision models, some based on known techniques like Integer Linear Optimization or Neural Networks, and some based on completely original reasoning engines. Through extensive testing of many decision models on several different scenarios, we intend to determine what reasoning techniques are better to handle particular scenarios. The final long term goal is to be able to develop an expert system that is able to selectively enable the most appropriate decision model to handle the various self healing situations in the grid.

6. REFERENCES

[1] N. Jennings, M. Wooldridge, Agent-Oriented Software Engineering, in proceedings of the 9th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, 2000.

[2] M.D. Lemon, Venkatarajmanan, G., and Chapman, P. Using Microgrids as a Path towards Smart Grids. 2009

[3] J. Hossack, S.D.J. Mcanhur, J.R. Mcdonald, J. Stokoe and T. Cumming, A Multi-agent Approach to Power System Disturbance Diagnosis, In Proc. International conference on power system management and control, April 2002, Vol. 488, pp. 317-322.

[4] K. Nygard, Prakash Ranganathan et.al, Optimization Models for Energy Reallocation in a Smart Grid, IEEE INFOCOM 2011, Machine to Machine Communications and Networking Workshop, April 10-15 2011, Shanghai, China.

[5] A. Massoud, Securing the Electricity Grid," The Bridge, quarterly publication of the U.S. National Academy of Engineering, Volume 40, Number 1, pp. 13-20, Spring 2010

[6] Hennebry, Michael J., Kamel, A, and K. Nygard, An Integer Programming Model for Asssigning Unmanned Air Vehicles to Tasks, in Recent Developments in Cooperative Control and Optimization, Kluwer Academic Publishers, Dordrecht, Netherlands, 2003.

[7] M. Pipattanasomporn, et al., Multi-Agent Systems in a Distributed Smart Grid: Design and Implementation, Proc. Proc. IEEE PES 2009 Power Systems Conference and Exposition (PSCE'09), 2009.

[8] Z. Noorian, et al., "An Autonomous Agent-based Framework for Self-Healing Power Grid," Proc. IEEE Conference on Systems, Man and Cybernetics,, 2009.

[9] S. Karnouskos, T. Nass de Holanda (2009): Simulation of a smart grid city with software agents, European Modeling Symposium EMS 2009, Athens.

[10] S. Rahman, M. Pipattanasomporn and Y. Teklu, Intelligent Distributed Autonomous Power Systems (IDAPS), *In* Proc. 2007 the IEEE PES Annual General Meeting, Tampa, Florida, 8pp.

[11] T. Nagata, and H. Sasaki, "A Multi-agent Approach to Power System Restoration," IEEE Trans. on Power Systems, vol. 17, no. 2, pp. 457–462, 2002.

[12] T. Nagata, T., H. Fujita, and H. Sasaki, "Decentralized approach to normal operations for power system network", Proc. 13th International Conf. on Intelligent Systems Application on Power Systems, Cambridge, pp. 407–412, 2005.

[13] J. Momoh, and O. Diouf, "Optimal Reconfiguration of the Navy Ship Power System Using Agents," Proc. IEEE PES Transmission and Distribution Conf. and Exhibition, Dresden, pp. 562–567, 2006,.

[14] M. Nordman, and M. Lehtonen, "Distributed Agent-based State Estimation for Electrical Distribution Networks," IEEE Trans. on Power Systems, vol. 20, no. 2, pp. 52–658, 2005.

[15] J. Solanki, S. Khushalani, and N. Schulz, "A multi-agent solution to distribution systems restoration," *IEEE Trans. on Power Systems*, vol. 22, no. 3, pp. 1026–1034, 2007.

[16] D. Sutanto, Y. Ye, and M. Zhang, "Design of an Intelligent Self-Healing Smart Grid using a Hybrid Multi-Agent Framework", Journal of Electronic Science and Technology, vol. 9,no. 1,March 2011.

[17] S. McArthur, E. Davidson, J. Hossack, and R. McDonald, Automating Power System Fault Diagnosis through Multi-agent System Technology, Proc. Hawaii International Conference on System Sciences, 2004.

[18] E. Camponogara and S.N. Talukdar, Agent Cooperation: Distributed Control Applications, Proc. the International Conference on Intelligent System Application to Power Systems, April 1999.

[19] S. Pahwa1, A. Hodges, C. Scoglio, and S.Wood , Topological Analysis of the Power Grid and Mitigation Strategies Against Cascading Failures, In Proc. IEEE International Systems Conference, 2010.

[20] J. Hossack, S. Mcanhur, J. Mcdonald, J. Stokoe and T. Cumming, A Multi-agent Approach to Power System Disturbance Diagnosis, In Proc. International Conference on Power System Management and Control,, Vol. 488, pp. 317-322, 2002.

[21] W. Ketter, J. Collins, J. and C. Block, "Smart Grid Economics: Policy Guidance Through Competitive Simulation," ERIM Reprot Series, 2010

[22] F. Glover and C. McMillan (1986). "The general employee scheduling problem: an integration of MS and AI". Computers and Operations Research.

[23] F. Glover. "Future paths for integer programming and links to artificial intelligence", Computer & Operations Research 13, pp. 533-549, 1986

[24] F. Glover. "Tabu Search - Part 1". *ORSA Journal on Computing* **1** (2): 190–206, 1989

[25] F. Glover. "Tabu Search - Part 2". *ORSA Journal on Computing* **2** (1): 4–32, 1990

[26] P. Hansen, "The steepest ascent mildest descent heuristic for combinatorial programming", talk presented at the Congress of Numerical Methods in Combinatorial Optimization, Capri, 1986

[27] F. Glover, and M. Laguna, Tabu Search, Kluwer Academic Publishers, Dordrecht, 1997.

[28] M. Laguna, R. Marti, and V, Campos, Intensification and diversification with elite tabu search solutions for the linear ordering problem, Comput. Oper. *Res.*, 26, 1217, 1999.