DEVELOPMENT OF INTELLIGENT AND SECURED ALGORITHMS FOR MODERN COMPLEX SYSTEM CONTROL AND PROTECTION

by

Arangamanikkannan Manickam

B.E., Anna University, India, 2009

A thesis submitted to the Department of Computer Science
College of Arts and Sciences
The University of West Florida
In partial fulfillment of the requirements for the degree of
Master of Science

2011

The thesis of Arangamanikkannan Manickam is approved:

Sharon J. Simmons, Ph.D., Committee Member	Date
Sukumar Kamalasadan, Ph.D., Committee Co-Chair	Date
Thomas Reichherzer, Ph.D., Committee Chair	Date

Accepted for the Department/Division:

Leonard W. ter Haar, Ph.D., Chair	Date

Accepted for the University:

Richard S. Podemski, Ph.D., Dean, Graduate School	Date

ACKNOWLEDGMENTS

It is a pleasure for me to thank all who worked with me to make this research possible, especially Dr. Edwards and Dr. Simmons who introduced me to the world of research and Dr. Reichherzer who encouraged and supported me throughout the period of this study.

I also want to express my gratitude to Dr. Kamalasadan who initiated this research and guided me by spending his precious time even though he had to leave the University of West Florida in the middle of this research. I would like to acknowledge all my friends and family members who encouraged me to finish my work on time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

DEVELOPMENT OF INTELLIGENT AND SECURED ALGORITHMS FOR MODERN COMPLEX SYSTEM CONTROL AND PROTECTION

Arangamanikkannan Manickam

This research is focused on developing a Multi-Agent Methodology (MAM) for protecting and controlling the components in an electric power grid. The software agents are replicated using a concept called code mutation, which hardens the attacks against the agent network. Replication of mutated code is used with a majority-voting algorithm to prevent attack consequences from occurring when an attack eventually bypasses the prevention mechanisms. These agents are grouped to form a cluster that can either be a monitor or a supervisor. The clusters work collaboratively by communicating with each other using a developed ontology. The supervising agent cluster obtains a global view of the entire system and, hence, performs a wide-area control operation using neural network based adaptive-critic approaches. We test the developed agent clusters for control and protection of a two-area and multi-machine power system model, and we test also the wide-area controller on the same power network and demonstrate it to be working better than local controllers.

CHAPTER I

INTRODUCTION AND LITERATURE REVIEW

Modern hardware components are highly sophisticated and complex. A complete system consists of multiple modern hardware components that make the controlling extremely difficult. For example, current smart phones feature more advanced computing capability and connectivity than a low-end mobile phone, and they demand powerful processors, abundant memory, larger screens, and highly functional operating systems. A smart phone can be considered as a handheld computer integrated with a mobile telephone. Similar to smart phones, in the electric power industry, smart grids deliver electricity from suppliers to customers using two-way digital communication to control appliances at consumers' homes, thereby saving energy, reducing cost, and increasing reliability.

These modern systems are highly sophisticated in both hardware and software components. This integration of hardware and software modules requires some additional enhancements for improving their performance. The controlling of these systems should be done with high resolution, and security should be provided for both the system components and the communication between them. Security is considered in two aspects: system security and communication security. System security deals with securing the components in the system from exploits, and communication security mainly lies in securing the messages passed between the systems. To enable these two types of securities in modern systems, software agents are developed with inherent security features that make them robust and reliable.

Computer programming has progressed from machine code to assembly language, to machine independent programming languages, to sub-routines, to procedures, to abstract data types, to objects, and to agents. A software agent is expected to act on behalf of someone to carry out a particular task that has been delegated to it (Bradshaw, 1997). But since it is tedious to have to spell out every detail, we would like our agents to be able to infer what we mean when we tell them. Agents can infer this detail only if they know something about the context of the request.

Agents

Agents are high-level autonomous software abstractions (Wooldridge, 2009). A software agent can be defined as a software program running on a computer system that is capable of performing independent actions on behalf of its user or owner thereby deciding what needs to be done to satisfy the design objectives, rather than continuously being instructed by a supervisor. An agent is also defined as a software (or hardware) entity that is situated in some environment and is able to autonomously react to changes in that environment.

The separation of agent from environment means that agents are inherently distributable. Placing copies of the same agent in different environments will not affect the reasoning abilities of each agent nor the goals it was designed to achieve. However, the specific actions taken by each agent may differ because of different observations from two different environments. This mechanism means that an agent can operate usefully in any environment that supports the tasks the agent intends to perform. Software agents hold a property, named *autonomy*, which is defined as the ability to schedule actions based on environmental observations. A rational agent is an agent that has clear preferences and always chooses to perform the action that results in the optimal outcome.

Agent development fits into three different categories. *Agent theory* is concerned with the question of what an agent is and the use of methods for representing and reasoning about the properties of agents. *Agent architectures* can be thought of as software engineering models of agents. Researchers in this area are primarily concerned with the problem of designing software or hardware systems that will satisfy the properties specified by agent theorists. Finally, *agent languages* are software systems for programming and experimenting with agents. These languages may embody principles proposed by theorists.

A relay-agent is used to protect the system which moves between equipment to utilize data and functions distributed over the power system (Tomita, Fukui, Kudo, Koda, & Yabe, 1998). This technique provides a powerful means of advanced protection functions by combining microprocessor technology and information-communication technology.

Intelligent Agents

The concept of an agent has become important in both artificial intelligence and mainstream computer science (Wooldridge & Jennings, 1995). The difference between the definition of an agent and an intelligent agent is that the intelligent agent holds *flexible autonomy*, whereas the traditional agent holds *autonomy*. We define an intelligent agent to be an agent that complies with the following requirements.

1. *Reactivity.* An intelligent agent is able to react to changes in its environment in a timely fashion, and it takes some action based on those changes and the function it is designed to achieve without human intervention.
2. *Pro-activeness.* Intelligent agents exhibit goal-directed behavior. Goal-directed behavior connotes that an agent will dynamically change its behavior in order to achieve its goals. For example, in a power network when an agent finds out that its

neighbor is not responding while seeking for some information, it can report this issue to a supervisor, where some corrective action could be taken. This pro-activeness is described as an agent's ability to "take the initiative." This ability is one that distinguishes agents from traditional software and hardware systems.

3. *Social ability.* Intelligent agents are able to interact with other agents to satisfy a global system goal. Social ability is more than simple passing of data among different software and hardware entities. It connotes the ability to negotiate and interact in a cooperative manner. This ability allows agents to converse rather than simply pass data.

Multi-Agent Systems

A Multi-Agent System (MAS) consists of two or more intelligent agents that interact with each other. In the most general case, agents will be acting on behalf of users with different goals and motivations. To successfully interact, the agents will require the ability to cooperate, coordinate, and negotiate with each other as much as people perform these activities. An MAS can also be defined as distributed and coupled networks of intelligent software agents working in coordination toward a global goal.

System control. Controlling a system is more critical than controlling a device. There are numerous types of controllers in use today that help to achieve and maintain stability in a power system. Stationary controllers are used with plants whose operating dynamics are so limited that they can be designed with fixed parameter models. But as plant dynamics change, the performance of these types of controllers becomes worse, and the system becomes unstable. The model reference adaptive controller is a type of self-tuning controller that uses a reference model of the plant being controlled to develop its control signals (Narendra & Annaswamy, 1992). The plant output is compared to the reference model to generate an error signal that is used by the controller to adapt its operating parameters. This controlling method works well with linear plants and works better than proportional-integral-derivative controllers. Intelligent controllers use the concept of artificial intelligence techniques to develop controllers capable of learning and modifying their behavior while interacting with the controlled system. The tools used for developing these intelligent controllers include Neural Networks (NN), fuzzy logic, machine learning, and genetic algorithms. NNs are able to learn and represent any function, given sufficient neurons and training. These NNs are used in Dual Heuristic Programming (DHP) to perform wide-area controlling where a system is considered as a whole instead of controlling a specific component in the system. One example for modern and complex physical systems is a power system. This research focuses on developing intelligent and secured algorithms for controlling and protecting power system components.

Electrical power systems contain components that need to be monitored periodically. Researchers have developed MAS that contributes to the control and protection of components in a power system. Thus, the protection of power systems is achieved without human intervention. But manpower is still needed for managing the logs and for solving problems that computers cannot. These agents are placed in the nodes of the power network, and they communicate through the Internet or a devoted transmission line. These communication channels are vulnerable to cyber attacks. So the agent's platform and its communication need to be secured.

Multi-agent systems for power system applications. There are two beneficial aspects supported by MAS needed for power engineering applications: flexibility and extensibility. Flexibility is defined as the ability to respond correctly to dynamic situations and to support replication in varied situations (McArthur et al., 2007a). This definition is very similar to *autonomy*, and therefore, intelligent agents should automatically be flexible.

An MAS is used for restoring power in case of failure (Solanki, Khushalini, & Schulz, 2007). Different types of agents are used to monitor different components of the system. These agents communicate with each other in the power restoration phase. Modern communication techniques and the equipment provided by distribution-automation evolving technology are combined together to provide efficient high-end protection and methods for controlling the power system (Baxevanos & Labridis, 2007). This combination provides a flexible and versatile MAS that is used for fault isolation and power restoration

The secondary voltage control of power systems has been developed and applied mainly for the generator automatic voltage regulator to improve the power system voltage stability. Research on secondary voltage control involves various types of power system voltage controllers, automatic voltage regulators, static VAR compensators (VAR is referred to as volt-ampere reactive), and static synchronous compensators, for a new application: power system voltage management in system contingencies. This secondary voltage control is implemented based on the principles of MAS theory (Wang, 2001).

Protection engineering is the process of collecting data from a range of monitoring devices to perform post-fault disturbance diagnosis. In the past, heterogeneous intelligent systems have been developed to interpret the data and to provide information to engineers to assist with the disturbance diagnosis task. An MAS is developed as a flexible and scalable alternative for protection engineering diagnostic agents, which integrate a legacy Supervisory Control And Data Acquisition (SCADA) interpretation system with new methods for digital fault recorder data interpretation and for enhancing fault record retrieval from remote fault recorders (Hossack, McArthur, Menal, & McDonald, 2003; Davidson, McArthur, McDonald, Cumming, & Watt, 2006). These agents have been intelligently interpreting and managing data online at a transmission system operator in the U.K.

FIPA standards for agent development. The Foundation for Intelligent Physical Agents (FIPA) is an IEEE (IEEE is known as Institute of Electrical and Electronics Engineers) computer society standard that promotes agent-based technology and the interoperability of its standards (Figure 1) with other technologies.

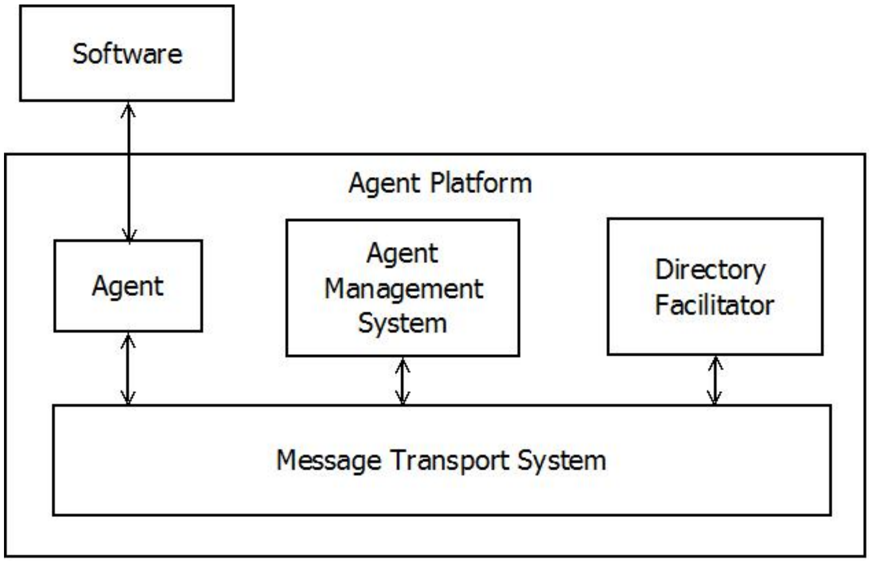


Figure 1. Agent reference model developed by FIPA. FIPA is known as Foundation for Intelligent Physical Agents.

FIPA specifications explain only about the interfaces through which agents may communicate with each other. These specifications do not describe the implementation details.

FIPA specifications are divided into five categories: applications, abstract architecture, agent communication, agent management, and agent message transport (FIPA, 1996).

Fault-Tolerant Systems

Building redundancy into systems is one of the standard engineering approaches for gaining fault tolerance. Building redundancy into MAS simply involves providing more than one agent with a given set of abilities. If an agent needs the services of a second agent in order to fulfill its goals and the second agent fails, the agent can pro-actively seek an alternative agent to provide the services it requires. The alternate agent can be found easily by querying the directory facilitator if there is one. This redundancy may be provided by simple duplication of each agent, possibly with distribution of duplicates across different computers in the network or different processes in the same computer. This redundancy would provide a tolerance to physical faults, such as the loss of a network connection or damage to a computer. Tolerance to programming-related faults would require a more design-intensive solution rather than simply running two copies of a single agent; The same functionality would be coded differently in those two agents. Various applications and operating environments will have differing requirements for levels of robustness and fault tolerance, and so the approach taken must be application-specific.

A distributed, decentralized, and fault-tolerant agent framework based on the .NET is used to solve distributed and complex problems (Ali, Shaikh, Shah, & Shaikh, 2010). The agent platform named as ACENET (Agent Collaborative Environment based on .NET) has been developed to follow FIPA specifications and to work under a decentralized architecture.

Security Vulnerabilities

The power system is vulnerable because of remotely accessible supervisory and control devices. It includes digital protective relays, telemetry devices, remote terminal units, data processing units, programmable logic controllers, intelligent electronic devices, and microprocessor-based substation controllers. These controllers interact with SCADA and energy management systems. The vulnerable points include field devices that have communication links, substation devices, network controlled devices including station computers, communication links to SCADA systems, and generating station network interfaces. The mechanisms of an attack could be within the power network (inside attack) or from external sources. Attacks by an insider can be easily detected and are very rare. On the other hand, the attacks by the outsider require a mechanism that accesses the device remotely. Those mechanisms include methods capable of circuit switching, analog or digital monitoring, calculating data values for protective functions, transmitting data to and from control power apparatus, and communication devices that are used for remote access (Leon, Foss, Krings, & Oman, 2002).

A type of cyber-attack called denial-of-service causes a traffic avalanche in short durations and can potentially bring down systems, thereby causing a disruption of services (Ten, Govindarasu, & Liu, 2007). However, all the work in MAS and agent based control illustrate agent architecture that resides on a power system bus could be attacked by an outside source as well. The possibility of an attack creates a critical issue related to the robustness and resiliency of an agent against an attack. A novel solution for this problem is provided in the next section.

Agent Security

As described earlier, agents are autonomous software programs running on a distributed network of control processors, which communicate with each other to share information. There are risks associated with distributed agent architectures, namely cyber attacks (Simmons, Edwards, Wilde, Just, & Satyanarayana, 2006; Chien & Szor, 2002). A successful attack on a single agent can have cascading effects on the entire system. These intelligent agents must be defended against such attacks.

These cyber attacks result in agents entering a type of failure mode. The failure modes have been classified into three distinct categories. *Crash failures* occur when a process functions correctly until it fails, at which time it immediately stops executing. These types of failures are clearly visible and are recoverable by simply restarting the crashed process.

The next and less easily identified failure is *response failure*. A system in a response failure mode could fail to respond, respond slowly, or give an incorrect response. It is difficult for a client process to distinguish between a failure to respond and a slow response of a server process. Recovery of a response failure is achieved by destroying the malfunctioning process and restarting a replacement process.

The most difficult failure mode is *byzantine failure*, also known as *arbitrary failure*. A process in byzantine failure will produce arbitrary results. It is assumed that the results of this type of failure will be such that the greatest possible circumstances will result. For example, in an agent-based system, arbitrary results would affect other agents and prevent the system from functioning properly. Since a malfunctioning process could mimic correct operation of processes, it can be extremely difficult to distinguish from properly functioning processes. Processes in this failure mode present the most difficult challenge to failure detection. So the security feature in any agent should be capable of detecting the failure, preventing the cascading effect of the failure, and recovering from the failure more quickly. This capability reduces the possibility that consequent attacks of same type to be successful.

Multi-agent methodology developed with the use of mutation (Edwards, Simmons, & Wilde, 2007) secures the agents from attacks of malicious intent. The mutation methodology is a part of multi-level security architecture and has been incorporated into a prototype for protecting software agents. As mutation modifies the memory footprint of an executing process without altering its runtime behavior, these processes are shown to be resistant to attacks resulting in arbitrary code execution. Replication of mutated code is used with a majority-voting algorithm to prevent attack consequences from occurring when an attack eventually bypasses the prevention mechanisms.

Wide-Area Control

Power systems are large-scale, nonlinear, nonstationary, and distributed over large geographical areas. So system wide disturbances in the power systems are a challenging problem for the power industry. Modern power systems are becoming more complex, and thus, these systems are more vulnerable to disturbances. When a major disturbance occurs, protection and control actions are required to stop the power system degradation, restore the system to a normal state, and minimize the impact of the disturbance.

The standard power system controllers, such as generator exciter, automatic voltage regulator (Kundur, Balu, & Lauby, 1994), speed governor (Kundur et al., 1994), Power System Stabilizer (PSS; Kundur et al., 1994), and power electronics-based flexible AC transmission system devices (Hingorani & Gyugyi, 2000) are local non-coordinated linear controllers. Each of these controllers controls some local power system component to achieve a local optimal performance, but they do not have any information about the rest of the system performance. Furthermore, the possible interactions among these local controllers might lead to adverse effects, thereby causing inappropriate control effort by different controllers. When severe disturbances or contingencies occur, these local controllers are not always able to guarantee stability (Okou, Dessaint, & Akhrif, 2005). To overcome the shortcomings of these local controllers, a novel optimal wide-area coordinating neurocontrol based on wide-area measurements for a power system with a PSS, a large wind farm, and multiple flexible AC transmission system devices has been developed (Qiao, Venayagamoorthy, & Harley, 2007).

Wide-Area coordinating System Centric Controller and Observer (WASCCO) is proposed in this research because it is critical for improving the system-wide stability and dynamic performance. The Wide-Area Controller (WAC) operates at a global level to coordinate the actions of local controllers. Control center and supervisor of a particular area of the power system are considered to be the global level components. The local controllers communicate with the WAC for sending reports and receiving coordination/control signals to achieve system-wide performance goals.

Research Objectives, Proposed Design, and Methodology

This research thesis is focused mainly on developing a self-secure and intelligent system capable of controlling wide-area power systems using NN-based adaptive-critic schemes to stabilize synchronous generators in the power grid.

The following are the objectives for this research study.

- Design and develop a FIPA-compliant MAS for monitors and supervisors from the basic self-secure and fault-tolerant architecture implemented based on the mutation concept.

- Develop an ontology for the MAS to converse with each other. Implement message generator and parser to generate and understand messages using this ontology.
- Develop WASCCO, the proposed controller, using adaptive-critic designs and embed this controller with the supervisor.
- Test the MAS and the embedded controller on a two-area and five-machine (multi-machine) power system.

The proposed intelligent and secured algorithm comprises of a WAC, central grid observer, and agent-based communication links (Figure 2).

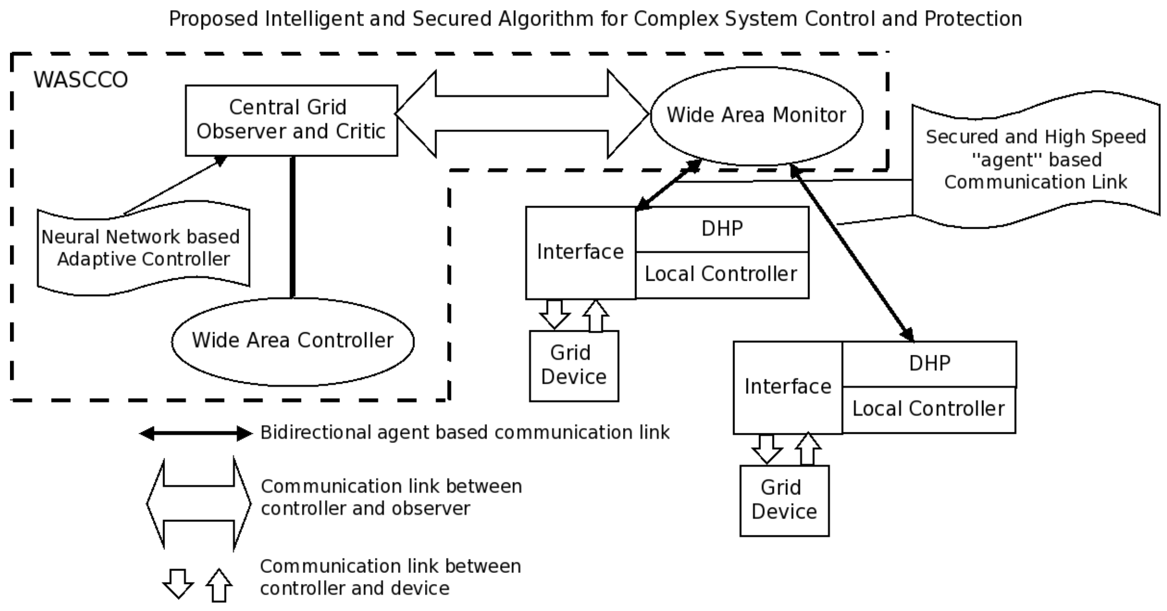

Proposed Intelligent and Secured Algorithm for Complex System Control and Protection
WASCCO
Central Grid Observer and Critic
Wide Area Monitor
Secured and High Speed "agent" based Communication Link
Neural Network based Adaptive Controller
Wide Area Controller
Interface
DHP
Local Controller
Grid Device
Interface
DHP
Local Controller
Grid Device
Bidirectional agent based communication link
Communication link between controller and observer
Communication link between controller and device

Figure 2. Proposed Intelligent and Secured Algorithm for Complex System Control and Protection. DHP is Dual Heuristic Programming. WASCCO is the Wide-Area Coordinating System Centric Controller and Observer.

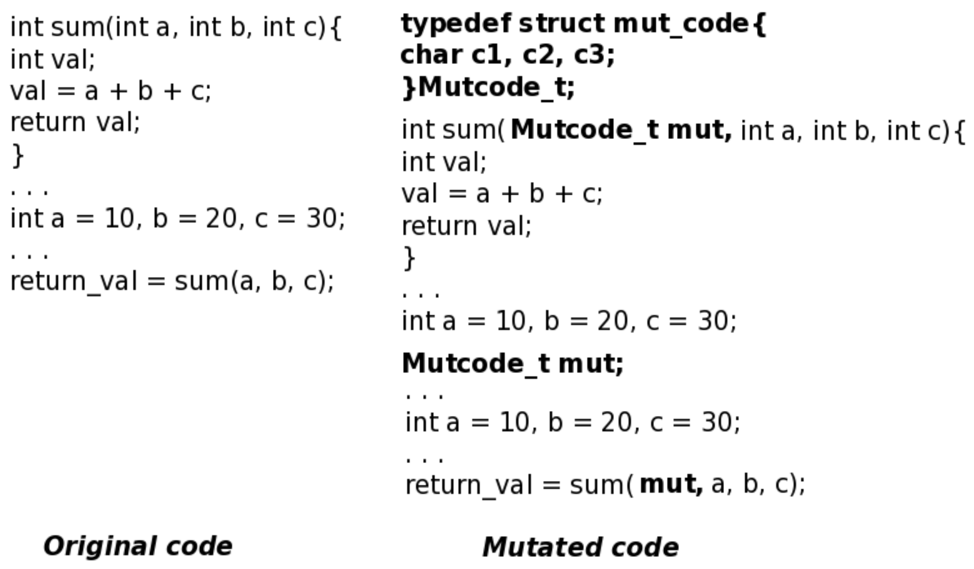
In this design, the wide-area monitor interacts with the local controllers and obtains the plant states through the agent-based, secured, and high-speed communication link. The local controllers control the plant operation with their knowledge and use the intelligence of wide-area monitor for better stability. This wide-area monitor is observed by an observer based on a critic NN, which is then used to generate control signals from the WAC. The control signal generated from this controller will be based on the entire power grid, and hence, the stability provided by this controller will be better than local controllers. The communication link between the local controller and the wide-area monitor is secured, and the communication agents responsible for this communication are also secured using the mutation methodology.

Mutation engine design. A prototype mutation engine has been designed to mutate C programs. The mutation engine uses the C to preprocess the source (Edwards et al., 2007). The following are the steps used for mutation of a source code.

- The source code is sent to the preprocessor so that the stripped code can be parsed without any problems associated with compiler derivatives.
- The preprocessed source files are then scanned for function prototypes, definition, and calls. Each encountered prototype or definition is mutated, both in the parameter list and in the variables, and the resulting mutation is recorded.
- Function calls encountered before a prototype are assumed to be system calls and not altered. The resulting source code is passed to the compiler that outputs the object code.
- When all the source files have been compiled, the individual object files are linked to create an executable.
- Mutated source files and object files are retained, thereby preventing the examination of the source code to construct an exploit directed at the mutation.

One means of combating buffer overflow attacks is to prevent the attacker from constructing an input that will result in arbitrary code execution. The mutation agent will alter the structure of the executing code to result in crash failure instead of byzantine failure (Edwards et al., 2007).

The implemented mutation engine scans the source code for function declarations, definitions, and calls (Figure 3). Each time a new function is located, a randomly defined structure is inserted into the argument list. Mutation of function declaration, definition, and function calls are done identically so that the resulting mutated code does not lead to any incompatible function calls. The structure insertion alters the offset from the parameters to the return address in the runtime stack. A similar insertion is made as a local variable declaration to ensure that local variables are not in a constant distance from the return address. Changing the offset of return address makes it hard for the buffer overflow attacks to occur. The resulting mutated source code is functionally equivalent to the original source code. The mutation modifies only the runtime stack layout compared to the original code.



```
int sum(int a, int b, int c){
int val;
val = a + b + c;
return val;
}
. . .
int a = 10, b = 20, c = 30;
. . .
return_val = sum(a, b, c);
```

Original code

```
typedef struct mut_code{
char c1, c2, c3;
}Mutcode_t;
int sum( Mutcode_t mut, int a, int b, int c){
int val;
val = a + b + c;
return val;
}
. . .
int a = 10, b = 20, c = 30;
Mutcode_t mut;
. . .
int a = 10, b = 20, c = 30;
. . .
return_val = sum( mut, a, b, c);
```

Mutated code

Figure 3. Example mutation of a source code with an arbitrary structure insertion (shown in bold font) in the parameter list.

Wide-area controller design. Our research group proposed several approaches for power system stabilization including NN-based intelligent adaptive controller (Swann & Kamalasadan, 2009), PSS based on fuzzy model reference adaptive controller (Kamalasadan & Swann, 2009), and intelligent controller based on the theory of supervisory loops (Kamalasadan, Swann, & Ghandakly, 2009b). The same group developed a hybrid controller for speed control of synchronous generators (Kamalasadan, Swann, & Ghandakly, 2009a). This research continued with developing several system-centric control approaches (Kamalasadan & Swann, 2010, 2011) for the analysis of inter-area mode oscillations.

In our proposed design, the agent clusters are placed in each node (plant) of the power system. These clusters are named based on their functionality. The clusters performing the monitoring activity of the synchronous generators are called Bus Management Agent Clusters (BMAC), and each area is equipped with a single supervisor. These supervisors are called Bus Supervisory Agent Cluster (BSAC). In a multi-machine power system, the WASCCO can obtain electrical parameters from various generators. All the BMACs will periodically send the plant information to their supervisor. So the supervisor in each area will have the information about all the generators in its own area, and hence, it can obtain a global view of its area. The WASCCO module can be plugged into the supervisor in each area. This module is designed based on adaptive-critic approaches. The input given to this NN is the parameters received from all local BMACs, and the output generated is a control signal for each generator. This control signal is sent back to the BMAC, from where it reaches the plant. Adaptive-critic designs for neurocontrol is suitable for learning in noisy, nonlinear, and nonstationary environments (Prokhorov & Wunsch, 1997). They have common roots as generalizations of dynamic programming for neural-based reinforcement learning approaches.

Implementation of the DHP model requires at least two NNs: (a) the *Action* or *Actor*, which acts as the controller; and (b) the *Critic*, which is used to train the Action (Swann, 2010). For some configurations, a third NN may be used to mimic or identify the plant when algebraic equations describing plant operation are not available. The identifier is trained to mimic the operation of the plant and may also be used to predict the future response of the plant to a control variable, given a set of state variables. Each of the NNs is similar in structure but is used in different ways to make the DHP controller successful in its task of optimal control. This supervised learning uses a known correct output for a given set of inputs. This known output is used to calculate the error and to adjust the NN's weights and biases using a backpropagation learning algorithm. In contrast, the adaptive-critic controller is used when there is no known (best) output to train on, so the optimal control output is an unknown. Instead, this reinforcement-style learning process uses a critic NN to discipline or guide the action NN in learning its task. In adaptive-critic based controller designs, the critic NN learns to approximate the utility function.

Research methodology. The research methodology focuses on the design and development of an MAS, the security mechanisms used for communication among agents in the system, and the control of wide-area power system using adaptive-critic techniques.

Proposed design. The proposed design uses the mutation-based agent platform (Edwards et al., 2007) for developing FIPA-compliant intelligent agents capable of communicating with each other. Agent Management Service (AMS) and Directory Facilitator (DF), members of FIPA-compliant agent clusters, play a significant role in communicating with other agents. These members provide querying capabilities for remote agents to know the properties of local agents and also their services. Both AMS and DF are designed and developed for each agent cluster and are placed in the nodes of the power network. The communication not only makes the agents pass messages but also facilitates conversation among them by enabling questioning capability. For this communication, both a content language and an ontology are required. Agents are able to generate messages following the ontology and the content language and are able to decode messages generated by other agents. Monitoring agents and supervising agents are developed with the capabilities mentioned above. The supervisors are equipped with a WAC developed using adaptive-critic approaches. Adaptive-critic designs developed by Swann (2010), used for achieving local optimal control, is further enhanced to perform wide-area control operation. Signals from the components of adaptive-critic designs are defined in Appendix A. The detailed design of the proposed WAC is described in Chapter 4.

The next chapter describes the MAS design, the communication mechanisms used by the agents, and the ontology developed for the communication. Chapter 3 delivers information about the design of agent clusters and the intelligent methods used in the communication between these clusters. This chapter also describes the functions these agent clusters using flowcharts and state transition diagrams. Chapter 4 explains the design of WASCCO. The development of the designed controller is explained in Chapter 5, and some results of the simulation conducted to test the proposed design are presented. This chapter also provides some information about the security features that could be applied to the developed model. Chapter 6 concludes the research and highlights the future scope of this work. A term *plant* is used to refer the power system model used for testing the developed architecture.

CHAPTER II

MULTI-AGENT SYSTEM DESIGN

This chapter describes the architecture of a general MAS and its implementation. This MAS uses *mutation* as a mechanism for agent's security and *replication* as a mechanism for providing fault tolerance. Mutation modifies the memory footprint of an executing process without altering its runtime behavior. Mutated agent code is replicated and is used with a majority-voting algorithm to prevent attack consequences from occurring when an attack eventually bypasses the prevention mechanisms. There is a group of agents working together to carry out the task for a single node and also for providing the security features. This group of agents is classified as monitors and supervisors based on the activities. The functionality and development of these specialized agent cluster are explained in the next chapter.

Multi-Agent Methodology

Our Multi-Agent Methodology (MAM; Figure 4) includes two Special Purpose Agents (SPA) and five General Purpose Agents (GPA). The SPAs are used for obtaining the information about agents in a platform through querying. The GPAs design a multi-layered security model, in which each layer provides facilities to prevent attacks, detect successful attacks, and recover from the consequences of successful attacks. Facilities of the original agent model are redistributed in the security model to provide a secure framework for implementing agents and for the upgrade of agents for future maintenance (Edwards et al., 2007). The GPAs and SPAs are combined together to form monitors and supervisors.

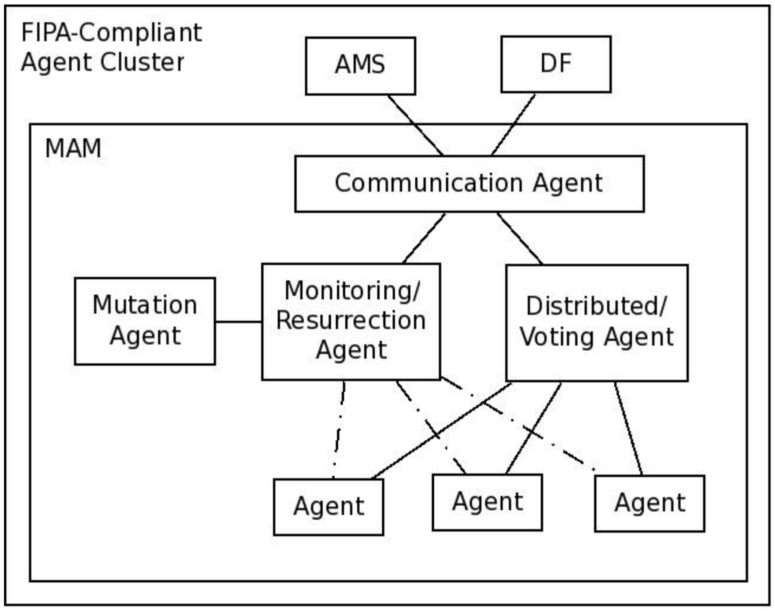


Figure 4. Multi-Agent Methodology with all the individual agents in the agent cluster and the control flow and data flow between them. Data flow is shown as solid lines, and control flow is shown as dotted lines. DF is Directory Facilitator, and AMS is Agent Management Service.

Special purpose agents. There are two SPAs, which involve in inter-agent collaboration and help other agents to know the details of local agents and the services they provide. The SPAs are AMS and DF. The specific duties of these SPAs are described separately.

Agent management service. The AMS acts like the white pages. It knows the information about all the agents and their communication addresses. It also knows the MAM configuration, such as the number of replicated agents present in the system. The process identifiers of all the replicated agents are stored in the AMS that can be used by the supervisor to remotely restart any agent if it is not responding. This AMS can be contacted by any remote agent in the entire system. The Monitoring/Resurrection Agent (MRA) loads all the agents to memory after mutation, and it sends the information about the agents to the AMS. The AMS registers these details in its internal database immediately after loading them. These details can be easily obtained by sending query messages. While processing the query, the AMS looks for the format of the query and validates the query by inspecting the header. Then the AMS collects all the requested information and sends a response to the query seeker.

Directory facilitator. The DF acts like the yellow pages by holding a service list. It knows about the services provided by the computational agents present in the platform. The MRA initially starts all the agents and registers them with both the AMS and the DF. If there are multiple replicated agent groups, this service list will grow bigger. The DF is used by the agents present in the remote nodes to obtain the service information about local agents. So any outside agent can query the DF service database. If an agent cluster in a remote node wants to measure the remaining power in a particular node, it can query the DF database of that node to find out whether or not any agent is providing power monitoring services. It is not necessary that each monitoring activity should be done by a separate agent. A single agent can monitor all the parameters in a particular power system node, thereby reducing the overhead in intra-agent communication. As all the services provided by the agents are listed in the service list along with the communication information, the remote agent can contact the intended agent immediately after querying. This DF can also be combined with the AMS. However, it is separated from the AMS in delegating duties so that a single agent is not overloaded with an extremely high number of queries.

General purpose agents. There are five GPAs that provide security services and perform monitoring or supervision services.

Communication agent. Communication Agent (CA) is one of the general purpose agents in the MAM, and it provides the first layer of security. It is the communication end point and acts as a gateway for messages sent among other agent clusters in the system (Figure 5).

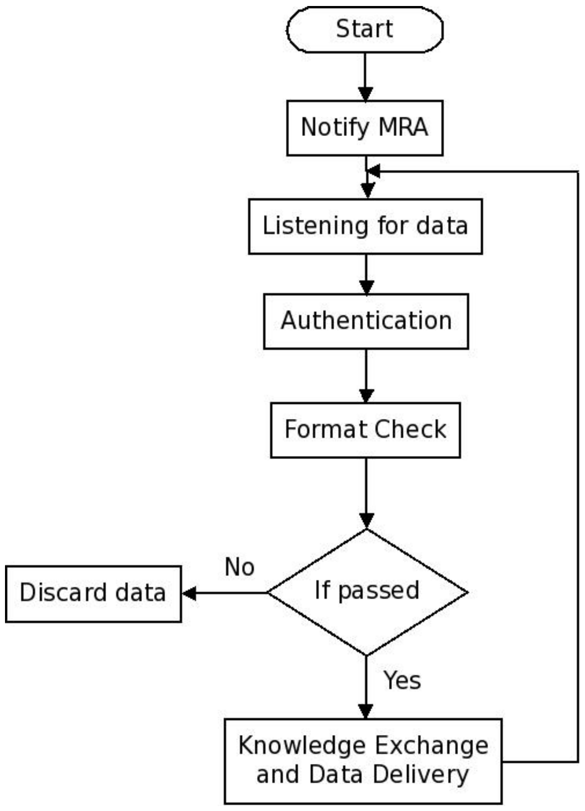


Figure 5. Functional flowchart of CA. CA is Communication Agent, MRA is Monitoring/Resurrection Agent.

Data that is received from external sources is authenticated using modular methods to identify and discard messages from spoofed origins. Additionally, incoming data is validated for compliance with the message structure, type, and size requirements. Messages that are not within expected parameters will be identified, thereby preventing any adverse effect to any other agent in the system. Messages following the specified format will be delivered to the destination agents. All the local messages sent within the agent cluster are delivered directly to the individual agents without reaching this CA. The messages from a remote agent to any cluster reaches the CA of that cluster, and the CA delivers the message to the intended agent in the cluster because this CA knows about all the agents in the cluster and their communication addresses. Similarly, the messages sent by the members of the cluster to the outside world (either to the node, other monitor, or the supervisor) are sent through the CA. Outgoing messages follow the same steps mentioned in the flowchart except the authentication phase. However, these messages will be checked for compliance with format, type, and size expectations, and hence, a faulty agent will not be allowed to transmit out-of-bounds messages. Therefore, all attempts to disrupt the execution of other agent systems are prevented.

The CA has some additional functionality that makes the fault restoration quicker.

- *Knowledge exchange:* The knowledge exchange is done by the CA. This agent periodically exchanges the information with all of its adjacent neighbors.
- *Fault reporting:* The CA also involves fault reporting. Whenever the replicated agents sense a problem in the system, the information is sent to the CA, and the CA forms a report message following the content language and the ontology designed for communication and sends it to the supervisor.

The CA acts as a router for routing messages coming from the outside agent cluster (Figure 6). It delivers the messages by inspecting the message type, source, and destination. If these messages are query messages, the response messages follow the same path to be delivered to the source, and an acknowledgement is received from the sender to ensure proper delivery.

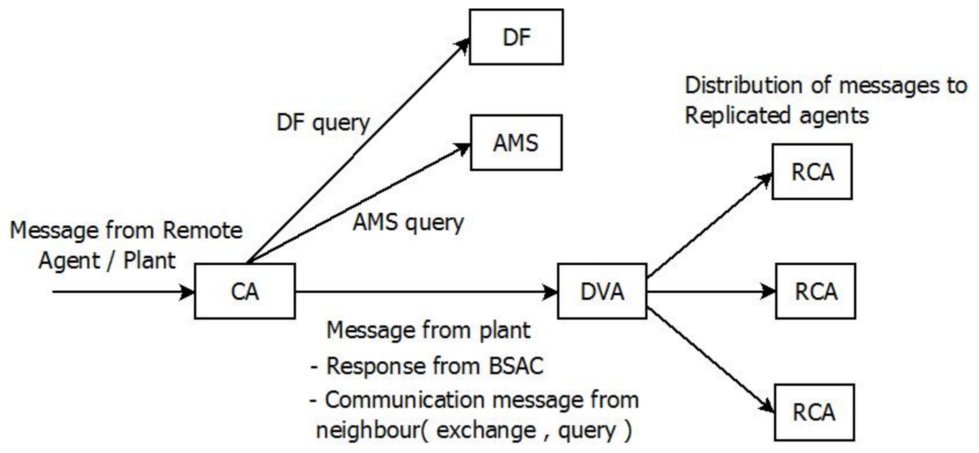

DF
AMS
DF query
AMS query
Distribution of messages to Replicated agents
RCA
RCA
RCA
Message from Remote Agent / Plant
CA
DVA
Message from plant
- Response from BSAC
- Communication message from neighbour(exchange , query)

Figure 6. Routing Diagram of CA. CA is Communication Agent, MRA is Monitoring/Resurrection Agent, DF is Directory Facilitator, AMS is Agent Management Service, DVA is Distribution/Voting Agent, and RCAs are Replicated Computational Agents.

Distribution/voting agent. The Distribution/Voting Agent (DVA) acts as the leader for all replicated agents (Figure 7). It serves three main purposes:

- data distribution to replicated agents;
- voting of agent responses; and
- reporting about non-responsive agents to the MRA.

The DVA receives data from the CA and validates the received data. This agent is considered to be more knowledgeable than the CA. It checks the headers to verify whether or not the data is from the power system node such as a power plant or from other monitor/supervisor agents. If the data has been verified to be valid, then this agent distributes the data to all replicated agents and waits for their response. It also starts a timer to find out whether or not the agents respond within the allowable time. When the replicated agents finish processing the message, they will send a response back to the DVA. The DVA collects these responses, and each response is considered as a *cast-vote* from the replicated agents. Then this DVA continues with the voting process to find out the majority of responses and sends that to the CA as the final response from the agent. During the voting process, if one of the agents does not respond, responds slowly, or sends an incorrect response, the DVA will report this deviation immediately to the MRA. The MRA will kill that particular replication and resurrect a new agent from the source code by mutation process.

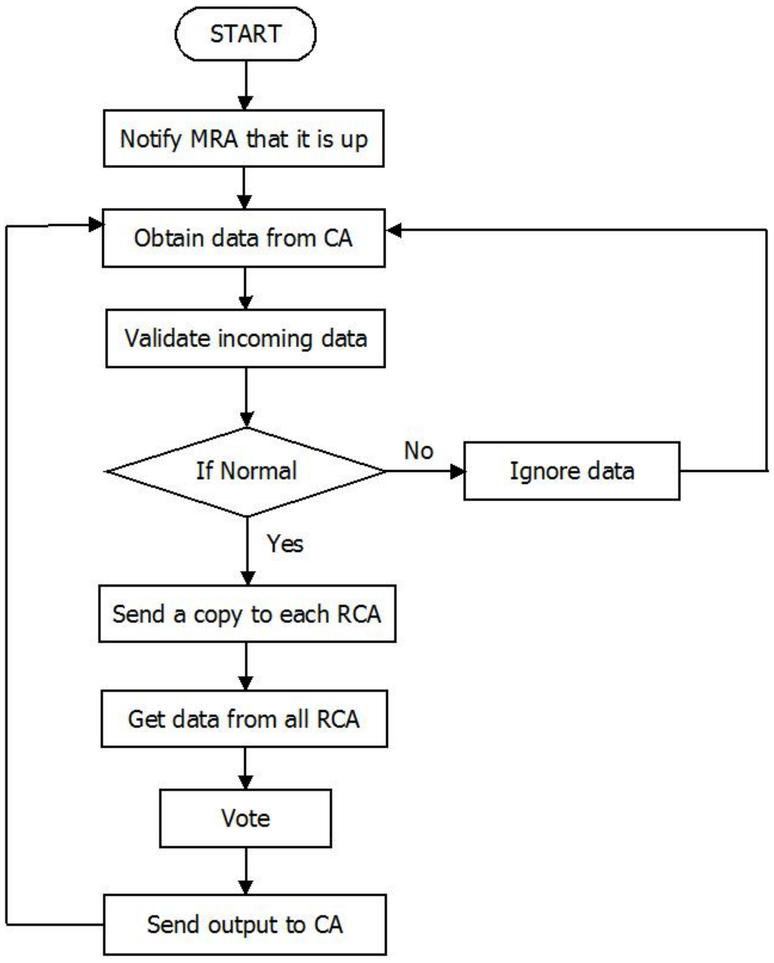


Figure 7. Functional flowchart of DVA. DVA is Distribution/Voting Agent, CA is Communication Agent, and RCAs are Replicated Computational Agents.

Replicated computational agents. Replicated Computational Agents (RCA) are the actual members in the MAM performing the task for monitoring/controlling activity of the power system components (Figure 8). RCAs accept authenticated input from the DVA, process them, and respond back to the DVA. If the parameters exceed a threshold limit, these replicated agents send a signal to shut down the generator. If they find out any abnormal condition that cannot be solved, they communicate with the supervisor to find out a solution. They will not accept further inputs for some period of time until they receive a response from the supervisor. Thereafter, they follow the order given by their supervisor and send a command to activate the order. To facilitate voting and violation, computational agents are replicated. Each RCA is computationally equivalent and initialized at the same state. An RCA is considered to be compromised if it does not respond, responds slowly, or sends an incorrect response.

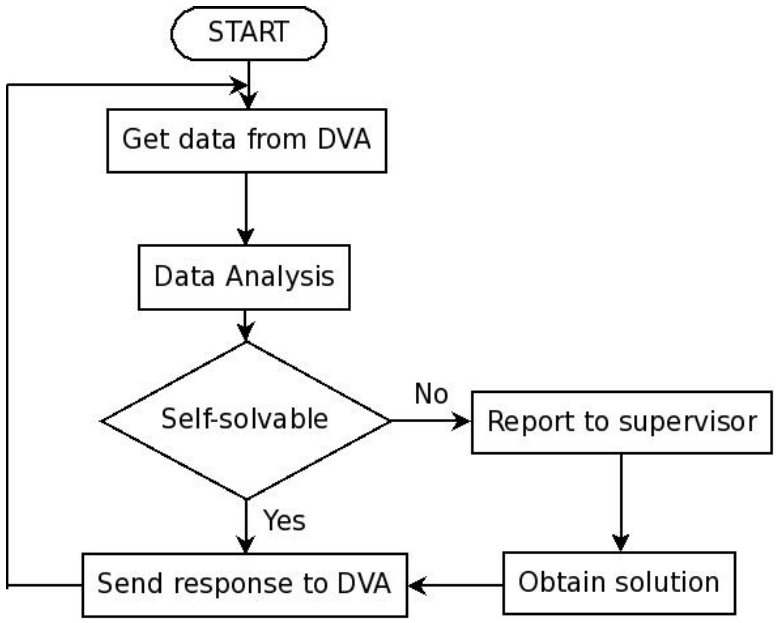


Figure 8. Functional flowchart of RCA. RCA is Replicated Computational Agent, and DVA is Distribution/Voting Agent.

Monitoring/resurrection agent. The MRA is the heart of our MAM. It is the initiator of all agents. This agent is the first to be loaded to memory, and it starts all other agents by mutating them using the mutation engine. If the MRA is notified about any abnormal behavior of one of the replicated agents, it will follow the resurrection process (Figure 9).

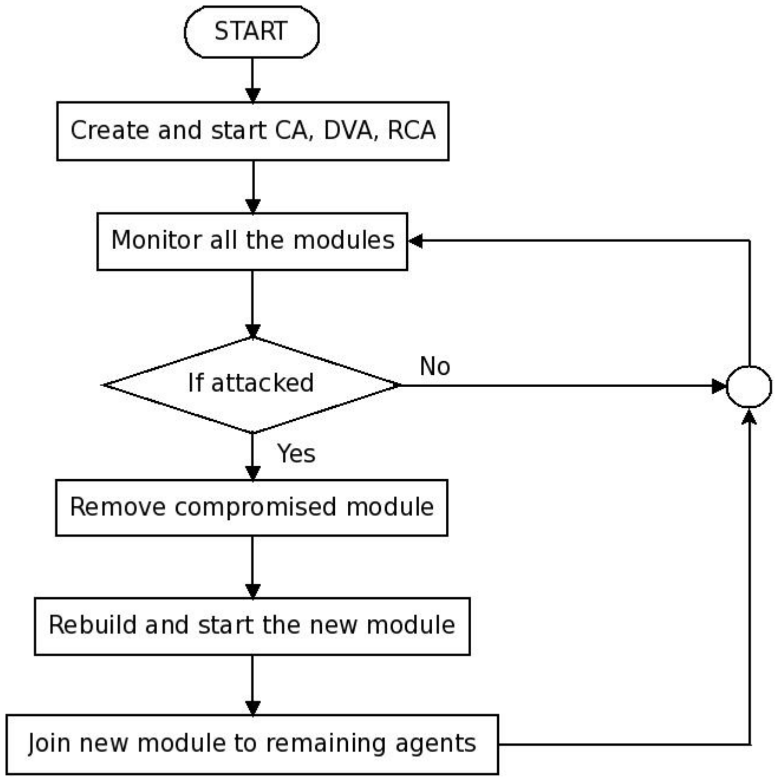

START
Create and start CA, DVA, RCA
Monitor all the modules
If attacked
No
Yes
Remove compromised module
Rebuild and start the new module
Join new module to remaining agents

Figure 9. Functional flowchart of MRA. MRA is Monitoring/Resurrection Agent, CA is Communication Agent, DVA is Distribution/Voting Agent, and RCAs are Replicated Computational Agents.

The steps taken for the resurrection process are listed below (Edwards et al., 2007).

1. *Remove the compromised RCA:* The corrupted RCA is killed if it still exists. If the RCA has crashed because of a failed intrusion attempt, then no further activity is needed at this step.
2. *Rebuild a new RCA:* A replacement RCA is compiled from the original source code. As explained below, this mechanism allows a mutation framework to be placed in the overall structure to increase security. Neither the object code nor the executable code will be retained after the new RCA has begun execution.

3. *Start the new RCA:* Once the executable code has been created, it is used to create a new executing agent to replace the compromised agent. After the new RCA has been successfully started, the executable code is removed from secondary storage to prevent the code from readily being examined.
4. *Inform the DVA:* The DVA is informed that the resurrected process is executing in a pristine state. It is the responsibility of the DVA to restore a valid computational state to the new RCA and to incorporate it into the voting procedure.
5. *Begin buffering input:* Input values from the Internet and serial devices will continue to arrive while the process is completed. These values are buffered to be acted upon later.
6. *Request saving of the current state:* Each loyal agent is asked to save its current state. We expect the agent state to be small and the save to occur quickly. Since all states are saved from loyal agents, they will be identical. A comparison of saved states can provide an additional validation of loyalty.
7. *Request loading of saved state:* The newly created agent, currently in a pristine state, will be asked to load its state from the one saved in the previous step. When this step is completed, all agents should have identical states, and the resurrection is considered to be complete.
8. *Resume normal operation:* Buffered input is sent to all agents, including the newly resurrected agent so that no input data is lost. As the buffer is cleared, the entire system returns to processing input in real time.

While faulty agents are being resurrected, the remaining loyal agents continue to process input and provide output to the DVA. Once the DVA has been informed of the completed resurrection, processing must be temporarily halted. Input values are buffered while the newly created agents are brought into a state consistent with other agents. Steps 5 through 8 are followed by the DVA to complete the resurrection process.

The MRA also monitors the high-level agents like CA, DVA, AMS, and DF. This monitoring process is done by periodically sending them a ping message. If one of them does not respond to the ping properly, the resurrection process is followed. As the MRA periodically communicates with its replicated agents, it is able to find out abnormalities in the replicated agents and to carry out the resurrection process, a capability which is known as *Intra-cluster communication intelligence*. The MRA is currently implemented in software, but we assume that a hardware implementation for MRA is forthcoming. This implementation will harden attacks against the MRA because an agent in hardware eliminates the possibility of exploitation/modification.

Consider an example in which a particular agent replication is identified to be malfunctioning by the MRA (Figure 10).

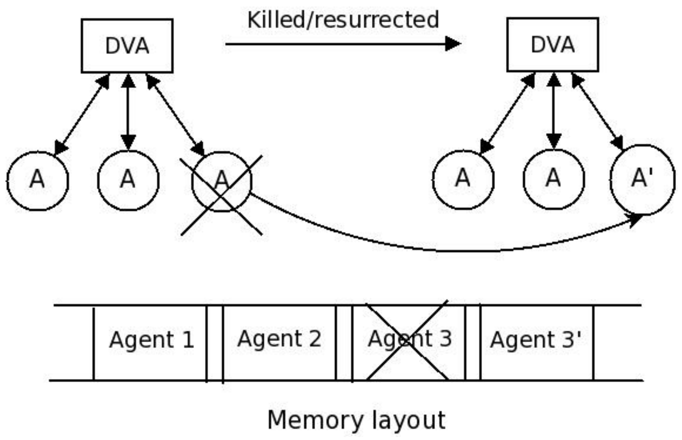
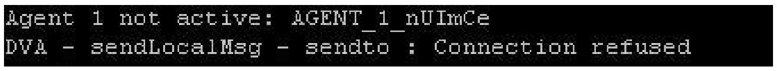


Figure 10. Example on mutation and agent resurrection. DVA is Distribution/Voting Agent, A refers to replicated agents, and A' refers to the resurrected agent.

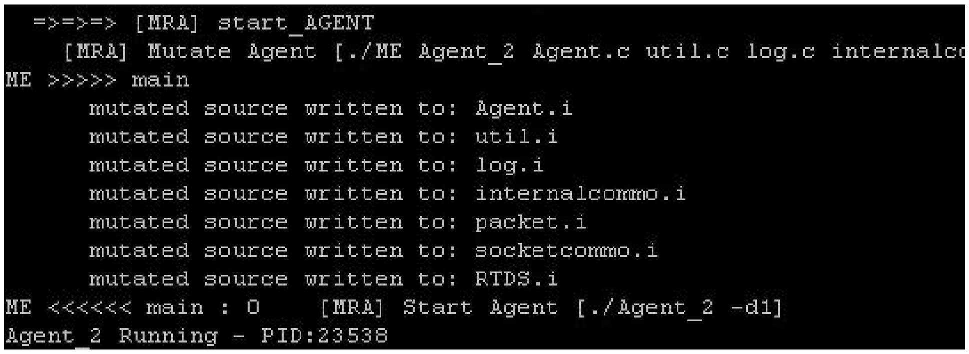
After the identification, the MRA terminates the malfunctioning agent, and a new replacement is resurrected by mutation. Mutation is done in such a way as to ensure that the memory layout is altered. If an agent is attacked by modifying a particular memory location, the mutation process kills and redesigns the agent. It will be difficult for the attacker to disrupt the agents from the bus. Since the new agent is mutated and loaded separately, it will be difficult for the attacker to identify the new agent as well.

In order to test the working of resurrection, we killed one of the replicated agents using the *kill* command in the Linux platform. After killing the agent, the DVA tries to send a message to the killed agent, and it gets refused because the agent is not active (Figure 11, Section a).



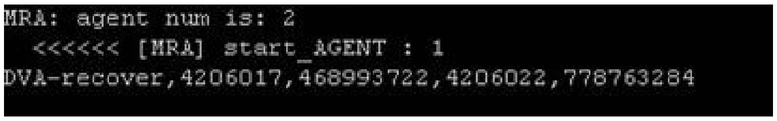
```
Agent 1 not active: AGENT_1_nUImCe
DVA - sendLocalMsg - sendto : Connection refused
```

(a) Agent killed



```
  =>=>=> [MRA] start_AGENT
    [MRA] Mutate Agent [./ME Agent_2 Agent.c util.c log.c internalc
ME >>>>> main
      mutated source written to: Agent.i
      mutated source written to: util.i
      mutated source written to: log.i
      mutated source written to: internalcommo.i
      mutated source written to: packet.i
      mutated source written to: socketcommo.i
      mutated source written to: RTDS.i
ME <<<<<< main : 0    [MRA] Start Agent [./Agent_2 -d1]
Agent_2 Running - PID:23538
```

(b) Agent resurrected



```
MRA: agent num is: 2
  <<<<<< [MRA] start_AGENT : 1
DVA-recover,4206017,468993722,4206022,778763284
```

(c) DVA recovered

Figure 11. Test on agent resurrection. MRA is Monitoring/Resurrection Agent. DVA is Distribution/Voting Agent. ME is the implemented Mutation Engine. PID is process identifier. AGENT is the replicated mutation of agent.

The DVA reports this activity to the MRA, and then the MRA starts the resurrection process. During the resurrection process, the MRA calls the mutation engine to replace the killed agent by a new agent (Figure 11, Section b). After the resurrection, the DVA is informed so that it can start sending the buffered input to all agents including the newly resurrected agent. After the DVA is recovered (Figure 11, Section c), the entire system continues to operate normally.

Mutation agent. The mutation agent runs on a command from the MRA. This module mutates the agent source code and returns the compiled mutated source code to the MRA for execution. The mutated source is not stored on disk, and the resulting executable is deleted after execution has begun. This mechanism prevents direct examination to design an attack targeted for the mutant. In this mutation method, the runtime stack is modified, thereby preventing buffer overflow attacks in a better way. By altering the mutation for each agent, we can eliminate a single attack reflecting in multiple agents.

As mutation alters the code layout, successful attacks are mitigated. Mutation ensures that the consequences of successful attacks differ, thereby allowing the DVA to identify and report faulty agents. If an agent fails to respond or responds in a manner not consistent with other agents, then that agent is killed, and a new mutant is resurrected and replaced.

Agent Communication

Mechanisms for communication between agents enable their social abilities (McArthur et al., 2007b). This communication also helps the agents for distributed problem-solving. The communication can be from a simple knowledge exchange message to a sophisticated fault report. As agent technology has matured, numerous methods for their communication capabilities have been developed. The language used for communication among agents is called Agent Communication Language (ACL). A similar communication mechanism named common instrument middleware architecture is developed (McMullen & Reichherzer, 2006) for integrating with scientific instruments, individual sensors or actuators, and sensor networks into computing and storage grids. This architecture uses the *parcel* protocol for sending messages when the parcels are documents written in extensible markup language, and these parcels identify what operations a client wants to perform on a channel. When these messages are sent to a client as a response, they contain data or result codes from sensors or actuators subsumed by the channel. The ontology designed for this architecture contains classes to model the physical components such as observatories, instruments, sensors, and actuators. This ontology makes the description amenable to machine reasoning tasks.

One of the ACL to be used by different researchers across different fields was the knowledge query manipulation language. In recent years this language has been known as FIPA-ACL (Finin, Labrou, Mayfield, & Bradshaw, 1997). A FIPA-ACL message contains 13 fields (Table 1). Among these, the first and only mandatory field in the message is the *performative* field that defines the type of communicative act. By classifying the message using a performative, FIPA-ACL ensures that recipients will understand the meaning of a message in the same way as the sender, removing any ambiguity about the message's content. FIPA specifies 22 performatives or communicative acts that define the type of message content and the flow of messages expected by each agent during specific classes of communicative acts (FIPA, 1996). The agents that follow a similar type of communication mechanism are called FIPA-compliant agents. The content of a message comprises two parts: content language and ontology. The content language defines syntax or grammar of the content. The semantics or lexicon is drawn from the ontology. The content language and ontology employed are declared in the *content* and *ontology* fields of the message.

Table 1

FIPA-ACL Message Structure

Message Field	Description
performative	Type of communicative act
sender	participant in communication
receiver	participant in communication
reply-to	participant in communication
content	Content of message
language	Content language
encoding	Encoding of content
ontology	Ontology used
protocol	Protocol for conversation
conversation-id	ID for conversation control
reply-with	Conversation control parameter
in-reply-to	Conversation control parameter
reply-by	Conversation control parameter

FIPA has proposed standards for four different content languages: semantic language, knowledge interchange format, resource definition framework, and constraint choice language. In addition to the content languages above, there are other content languages such as description logic and DARPA agent markup language that are closely related to the web ontology language standards developed by the semantic web community. The choice of content language is important because the chosen language will shape how a given ontology is expressed. The most widely used semantic language is only one of the four FIPA content language specifications to reach a stable standard. The ontology describes the concepts of a domain and the relationship among those concepts in a structured manner. Ontology design includes the model domain *concepts*, *predicates*, and *agent actions*. Concepts can be physical concepts such as substations, transformers, and circuit-breakers. Predicates are used to specify concept relationships and are usually evaluated as true or false, and sometimes these predicates can contain some useful information. An action is a special type of concept specifically for communicative acts such as *request* and *call-for-proposal*, in which agents discuss an event happening. Agents use the ontology for passing of information, formulating questions, and requesting the execution of actions related to their specific domain.

Design of content language and ontology. The content language used here is a customized language that is capable of expressing the electrical parameters among agents, and the agents can exchange their knowledge and query other agents. This content language is named as Multi-Agent System for Power Systems (MASPS). It is formed using a string of characters and symbols. Symbols are used as predicates and delimiters. Any information expressed by this language will contain the type of the parameter, a colon (:), the actual value of the parameter, and a semicolon (;) at the end. The message can contain any number of parameters.

```
parameter-type :  parameter-value ;
```

The performative of the MASPS content language clearly describes the reason for an agent's communication (Table 2).

Table 2

Performatives of MASPS Content Language

Performative	Description
BMAC-COMMO	Communication message sent from one BMAC to another BMAC
BMAC-QUERY	Query message sent by one BMAC to another BMAC seeking some information
BSAC-REPORT	Report sent from BMAC to its BSAC in case problems that are not self-solvable
BSAC-RESPONSE	Response sent by BSAC for reports received
AMS-QUERY	Sending a query message to the AMS in any BMAC
AMS-RESPONSE	Sending a response message from the AMS in any BMAC
DF-QUERY	Sending a query message to the DF in a BMAC
DF-RESPONSE	Sending a response message from the DF in a BMAC
BMAC-VOTE	When the BSAC asks for information, BMACs respond by sending votes
BMAC-INFORM	BSAC informs BMACs about any activity
BSAC-POLL	BSAC polls the BMACs for obtaining information
BMAC-ACK	Acknowledgement sent by BMAC
BSAC-ACK	Acknowledgement sent by BSAC

Note. COMMO is communication, BMAC is Bus Management Agent Cluster, BSAC is Bus Supervisory Agent Cluster, AMS is Agent Management Service, and DF is Directory Facilitator.

The ontology is designed for the MAS with concepts, predicates, and actions (Table 3). The concepts in the ontology include the physical objects and the attributes pertaining to those objects. The concepts are presented in a hierarchical form (Figure 12).

The concept may be a physical entity such as a power consumer, a power producer, a controller, or an electrical tool. The concept can also be an attribute for these physical entities. The attribute can be a value-attribute or a state-attribute. The value-attribute is used to measure the parameters of the physical entities. The state-attribute is used to mention the state (*on* or *off*) of physical concepts.

Table 3

Ontology Design

Ontology	Fields
Concepts	generator, transformer, circuit-breaker, voltage, power, load, speed, control-signal, simulation-time, current, PSS, WAC, object-state, governor, exciter
Predicates	”?” - used to ask questions
Agent actions	isolate/shutdown the generator, breaker activation, control-signal

Note. PSS is Power System Stabilizer. WAC is Wide-Area Controller.

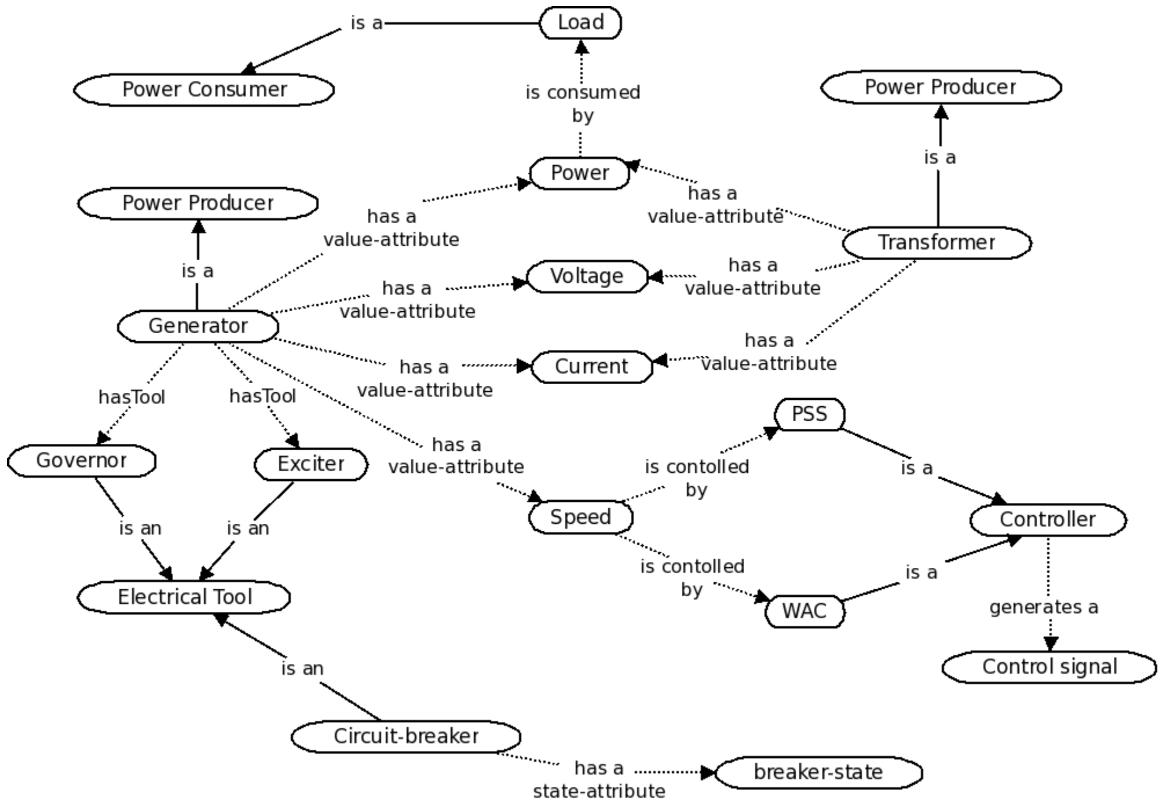


Figure 12. Hierarchical structure for concepts in the developed ontology. PSS is Power System Stabilizer. WAC stands for Wide-Area Controller.

The predicates are used to ask questions among the agent clusters. Agent actions are used to trigger actions in the node based on the received message toward the protection of the system components. The shutdown/isolate action sends a signal to the plant and commands to shutdown the generator. Breaker-activation action includes the breaker number. The control signal is generated from the BSAC using the embedded WAC. This control signal is a number, which is fed in to the exciter of the generator to stabilize the speed.

Message structure. The communication among the agents is done by passing strings of messages through the Internet using user datagram protocol. These strings are formed in a specific format. The format is different for each message type. The message structure has the following fields: source identification, performative (type of the message), content, ontology, and the actual message. The description for the fields are given below.

- *Identification* is used to identify the sender.
- *Performative* defines the type of the message.
- *Content* is the content language used for the message.
- *Ontology* is the ontology used for the message.
- *Actual message* contains a string of message that follows the ontology and formed using the content language described above.

All the agent clusters know the developed ontology and the message structure of MASPS content language (Figure 13). The replicated agents contain the message generator module that generates messages following the message structure and ontology based on the type of message. Both RCA and CA have the parser module. The CA uses this module to parse the incoming message and to do the format check. The RCA uses this module to parse the message and to perform the monitoring/supervising activity.

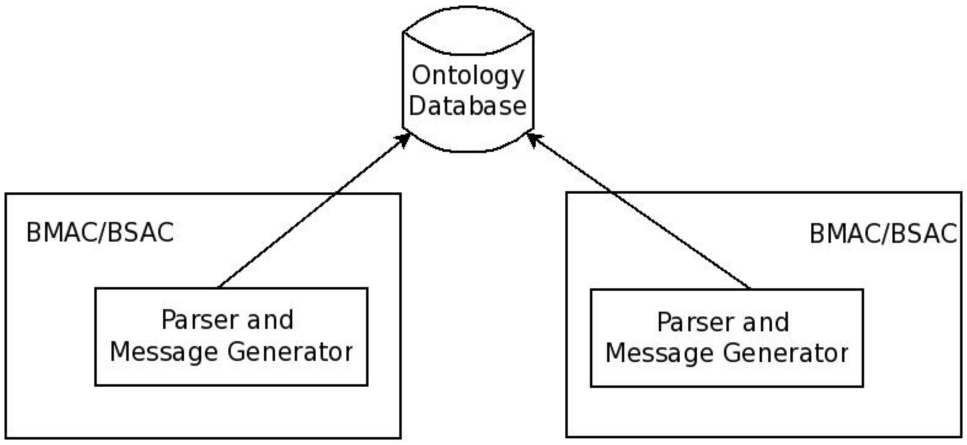


Figure 13. Parser and message generator. BMAC is Bus Management Agent Cluster, and BSAC is Bus Supervisory Agent Cluster.

Example query messages sent to the AMS and the responses are shown below. These messages are based on the MASPS content language and ontology.

Query: *AMS-QUERY;*

Response: *AMS-RESPONSE:1;Replications-3;pid1-11422;pid2-11562;pid3-11674;*

The above query simply asks the AMS to provide information about all the agents.

Query: *AMS-QUERY:agent?1;*

Response: *AMS-RESPONSE:1;Replications-3;pid1-11422;pid2-11562;pid3-11674;*

The above query asks for the information about a particular agent by giving the agent identification number.

Example query messages sent to the DF and the responses are shown below.

Query: *DF-QUERY;*

Response: *DF-RESPONSE:1;Replications-3;pid1-11422;pid2-11562;pid3-11674;*

Service-Voltage:current:load:power:breaker status:speed;

The above query is used to obtain the service information of all the agents.

Query: *DF-QUERY:agent?1*

Response: *DF-RESPONSE:1;Replications-3;pid1-11422;pid2-11562;pid3-11674;*

Service-Voltage:current:load:power:breaker status:speed;

The above query is used to obtain service information about a particular agent by giving the agent identification number.

Query: *DF-QUERY:service?current;*

Response: *DF-RESPONSE:1;Replications-3;pid1-11422;pid2-11562;pid3-11674;*

The above query asks for the information about an agent that provides a particular service.

Agent Configuration

A time experiment is done to measure the execution time for the activity of a single agent cluster. The experiment results (Table 4) are analyzed to understand the agent's life-cycle.

Table 4

Time Experiment

Agent	Time (ms)
Bootstrapping	340000.00
RCA	0.37
DVA	0.97
AMS	0.43
DF	0.40
CA	1.83
Turnaround Time	399.00
BSAC Response time	649.00

Note. CA is Communication Agent. DVA is Distribution/Voting Agent. AMS is Agent Management. DF is Directory Facilitator. BSAC is Bus Supervisory Agent Cluster. RCA is Replicated Computational Agent.

In this test, the MRA runs throughout the test period. This agent initiates other agents by mutating them using the mutation agent. Once all the individual agents in the cluster are started, they are ready to process the data. This process is known as bootstrapping. The agent mutation is done once during the initialization of the system. The same sequence occurs if one of the agents is non-responsive and resurrected. The activity sequence of a single agent cluster is drawn as a sequence diagram (Figure 14).

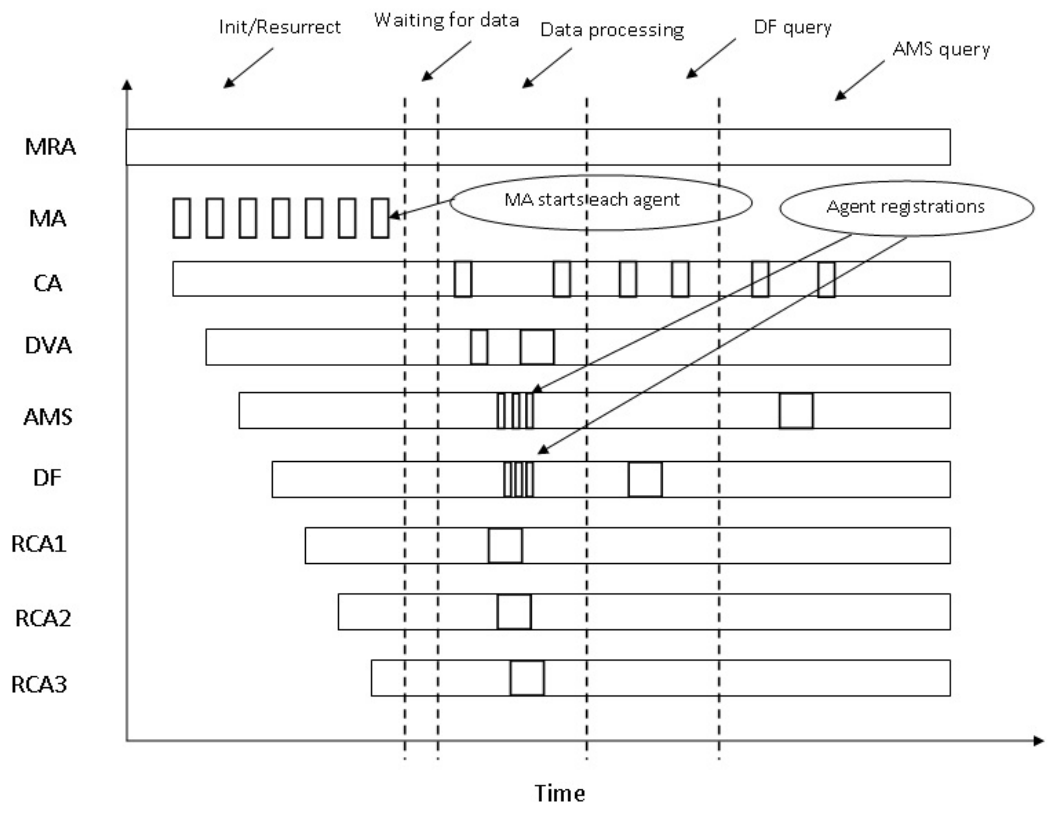


Figure 14. Timing diagram for a single agent cluster. CA is Communication Agent, MRA is Monitoring/Resurrection Agent, DF is Directory Facilitator, AMS is Agent Management Service, DVA is Distribution/Voting Agent, and RCAs are Replicated Computational Agents.

During normal data processing, the CA receives the data, exchanges it with its neighbors, and sends it to the DVA. The DVA distributes this data to the replicated agents and performs voting. Then the response is sent back to the CA. If a query message is received for AMS or DF, it is sent to the corresponding agent, and the response is routed accordingly. All other messages are sent to replicated agents through the DVA.

Multi-Agent Methodology Testing

The developed model is tested on a two-area and five-machine power system. More details about the model can be found in Appendix B and its parameters are presented in Appendix C. In this model, generators G_2, G_3, and G_5 are considered to form the first area, and generators G_1 and G_5 form the second area. These two areas are connected using a tie-line interconnecting buses 6 and 7. In this model, the developed agent cluster is placed on a single bus locally, and the test is conducted (Manickam, Swann, Kamalasadan, Edwards, & Simmons, 2010).

Under normal conditions, each area serves its own load and is almost fully loaded with a small load flowing over the tie-line. For this test, the G_3 is considered to be the target for an outside attack. In the simulation, the system is subjected to a short-circuit disturbance applied to the center of the transmission line connecting Bus 3 and Bus 6. Later in the simulation, an attacker who has achieved external control of portions in the power network is able to sever the generator connections on the grid. The agent placed in Bus 3 is able to detect that the system is no longer in control and takes action to reduce the potential for equipment damage.

The G_3 model is captured (Figure 15) with the interconnection of the generator to Bus 3 and a transmission line (modeled as pi sections) connecting Bus 3 to Bus 6. The initial disturbance is applied at $t = 5.0$ seconds and lasts for 0.1 seconds. The breakers labeled *BRK* (Figure 15) open immediately after the application of the fault and remain open until $t = 15$ seconds.

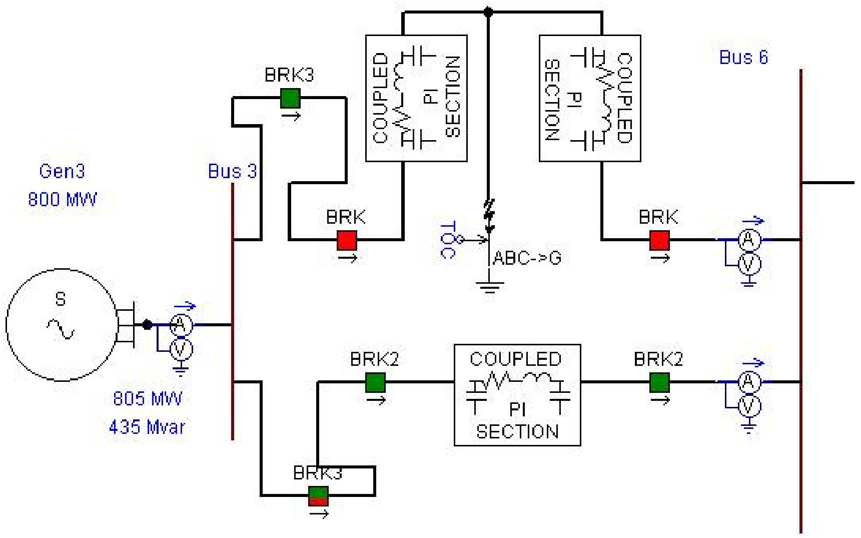


Figure 15. Generator 3 model. Bus-i denotes the bus number. BRK-i represents the breaker number. TOC is Time of Control.

There are some components (Figure 16) in the generator side for the agent to control. These components are activated based on the response from the agent. The components either isolate the generator from the rest of the system or include the generator to the system after the fault restoration.

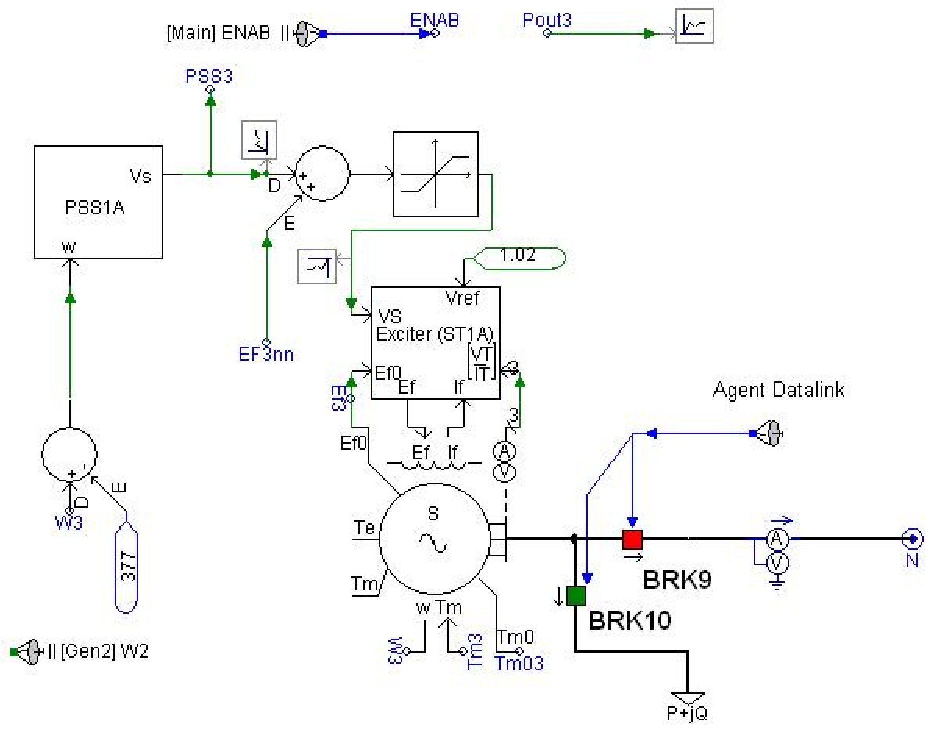


Figure 16. Generator 3 controller with multi-agent methodology. PSS1A is a Type 1 Power System Stabilizer. PSS is the output of Power System Stabilizer. W3 is rotor speed for Generator 3. T_m refers to mechanical torque. T_e is electrical torque. T_{qRef} is the torque reference input. E_f is field excitation voltage. I_f is field excitation current. ph is phase. V_{ref} is voltage reference. Pout is power output. ENAB is the signal for enabling the controller. EF3nn is controller output. BRK-i is breaker. The suffix “3” denotes generator number.

The speed of the generator (Figure 17) is greatly affected by the fault. The control signal from the agent isolates the affected transmission line and maintains control of the speed of the generator. When the attack occurs at $t = 10$ seconds, the speed of G_3 goes to a high value. An integrator device used in this test integrates the value of the speed error (Figure 18) over time. During the fault event, the integrated error value remains within 5 units. With the onset of the attack event at $t = 10$ seconds, with the severing of the remaining transmission line, the speed error quickly rises as the unloaded generator accelerates.

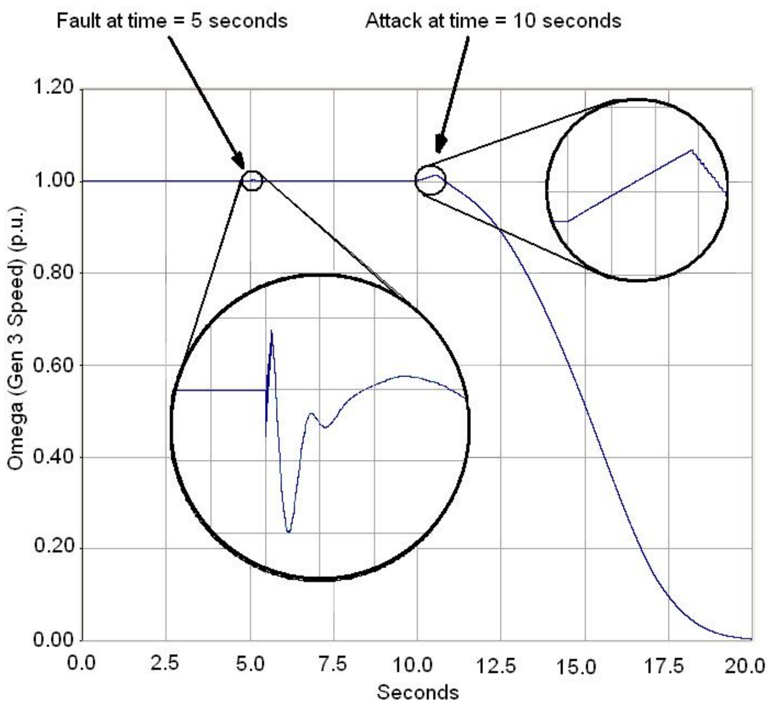


Figure 17. Generator 3 speed zoomed to show the deviations during the application of fault at $t = 5$ seconds.

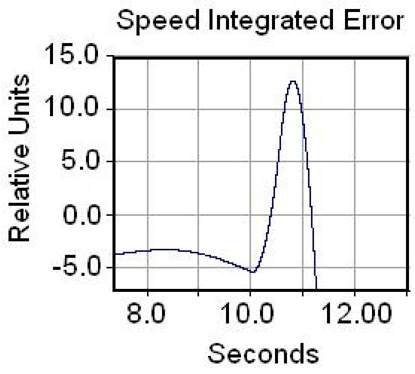


Figure 18. Generator 3 speed integral error showing the integral value that exceeds the threshold during the attack.

The fault in a single generator affects the speed of all generators in the system. The agent cluster situated on Bus 3 monitors this activity and sends a control signal that is used to stabilize the system. The first command (Figure 19) is issued by the agent cluster just after $t = 10.5$ seconds. The generator state monitor switches to the *off* state immediately afterwards, which initiates the generator shutdown sequence. As part of this sequence, the G_3 is disconnected from Bus 3 by the breaker labeled *BRK9* (Figure 16). Another breaker labeled *BRK10* uses the same control signal to connect an onsite load at the generator station to act as an electrical brake and, hence, slows down the unpowered generator to a complete stop.

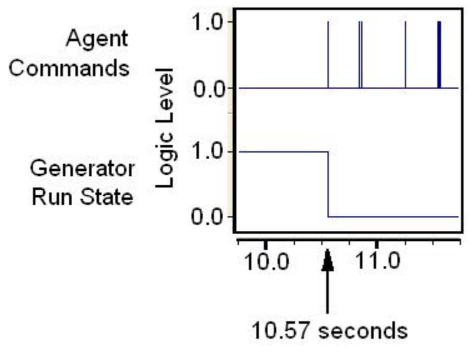


Figure 19. Command issued by the agent to isolate the generator and the corresponding generator run state.

CHAPTER III

AGENT CLUSTER DEVELOPMENT

The MAM that was explained earlier can be used for performing any activity that is reliable, fault-tolerant, and secure. This methodology is applied in developing agent clusters that are used for monitoring and protecting the power system components. This chapter explains the development of agent clusters, the functional activities of the developed agent clusters, the mechanisms used for communication among these clusters using the developed ontology, and the intelligence of the clusters in solving problems. The developed cluster model is then tested on a two-area power system, and the results are presented.

The test results presented in the previous chapter show that the developed MAM works well on a single power node. But we still have the impact of the fault applied in a single generator on all the other generators. If we place the agent clusters on all the nodes of the two-area power system, we need some coordination and supervision among the agent clusters, which necessitate having two types of agent clusters. Agent clusters that are situated near generating stations and end users are named as *monitors*, as they monitor the activity of power flowing through the transmission lines. We need to have an agent cluster situated in each area to control these monitoring agent clusters. These controlling clusters are named as *supervisors*. So each supervisor will have its own monitors, and the monitors will report any abnormal activity in the power node when they are not able to find a solution. In the following section, the design of these two agent clusters is described. They both follow the same multi-level security architecture and have the same functionality except for some core activity done by the replicated computational agents. This activity could be either monitoring or supervising based on the type of the cluster.

Design of Bus Management Agent Cluster

The BMACs are located only at buses near generating stations, and they keep monitoring these buses and protect the generators from abnormal conditions (Figure 20). The abnormality is considered as some control components in the generator being compromised.

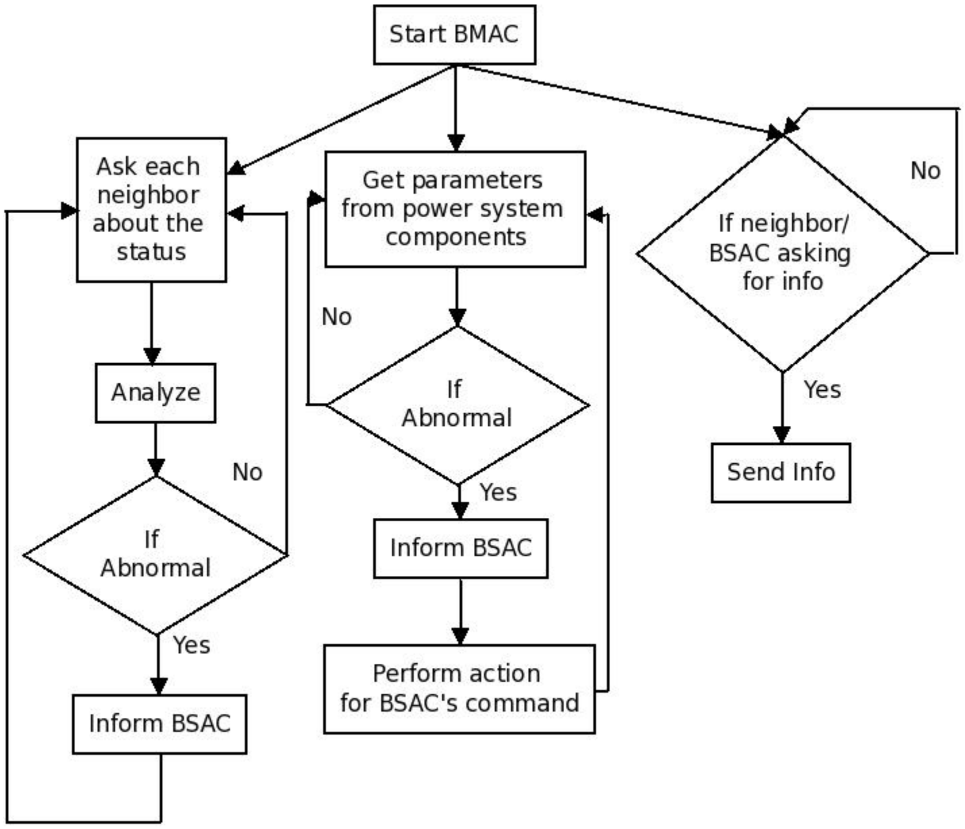

Start BMAC
Ask each neighbor about the status
Analyze
If Abnormal
No
Yes
Inform BSAC
Get parameters from power system components
No
If Abnormal
Yes
Inform BSAC
Perform action for BSAC's command
If neighbor/ BSAC asking for info
No
Yes
Send Info

Figure 20. Activity diagram of Bus Management Agent Cluster. BSAC is Bus Supervisory Agent Cluster.

The communication ability of the BMAC makes it capable of monitoring the activity of other agent clusters in the system. If one of these agent clusters does not function properly, the

BMAC will report to the BSAC. The BMAC is not authorized to provide power or to take control of other nodes in the system. It can only control the components in the power node where it is located. All BMACs can communicate with other BMACs in the same area to exchange information.

The activity of a BMAC can be summarized as the following steps.

- For messages from a node, do the following steps:
  - If the message parameters are within the threshold, send a normal response.
  - If any abnormal condition is detected in the generator, send a command to shut down the generator.
  - If any other fault message is received, inform the BSAC about this issue.
- For messages from a neighbor BMAC, do the following steps:
  - Update the neighbor information database, and send an acknowledgement.
  - If an AMS or a DF query is received, redirect the query to the corresponding agent, and send the response back to the sender.
- For messages from the BSAC, do the following steps:
  - If the BSAC asks for information about node parameters, send the information immediately.
  - If the BSAC sends information about other BMACs, update the neighbor information database.
  - If the BSAC asks for power to be delivered to neighbors, send a positive command if power is available, and respond with a negative command if the remaining power is not sufficient.
  - If the BSAC sends command to activate a component in the node, follow the order by sending a signal to the node.

Design of Bus Supervisory Agent Cluster

The protection of the power system is done by the BSAC (Figure 21). This cluster is similar to the BMAC in architecture and implementation. However, the activity of replicated agents will differ from BMACs.

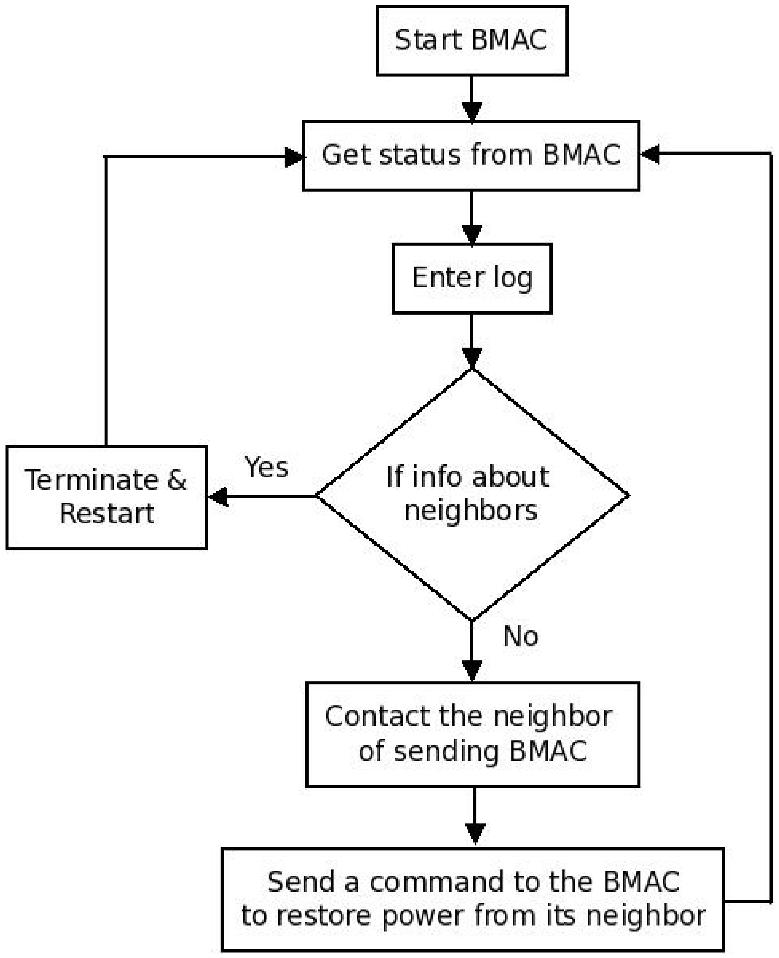

Start BMAC
Get status from BMAC
Enter log
If info about neighbors
Yes
Terminate & Restart
No
Contact the neighbor of sending BMAC
Send a command to the BMAC to restore power from its neighbor

Figure 21. Activity diagram of Bus Supervisory Agent Cluster. BMAC is Bus Management Agent Cluster.

The replicated agents in this cluster are responsible for supervising the agent clusters present in the lower-level buses; In BMACs, the replicated agents are responsible for monitoring the generator buses. Even though these BMACs and BSACs do not have any architectural differences, the functions performed by the replicated agents differ. The supervisory level agent periodically obtains the power system parameters from the monitoring agents and analyzes the data to identify any abnormality in the system. If the supervisor suspects any abnormal behavior, then it sends a command to react the abnormal behavior. The presence of the AMS and the DF is more beneficial for obtaining remote information.

The architecture and implementation procedure for the supervisory level agents are similar to monitoring agents. However, following additional qualities are required for these supervisory level agents to guarantee fault tolerance and security.

1. The number of replicated agents should be increased for enhancing security and fault tolerance.
2. The communication among supervisors and monitors should be strictly secure.

The activity of a BSAC can be summarized as the following steps.

- For messages received from a node, do the following steps:
  - If message parameters are within the threshold, send a normal response.
  - If any abnormal condition is detected in the generator, send a command to shut down the generator.
- For messages received from a BMAC, do the following steps:
  - If power restoration is needed because of a fault,
    - ∗ Compute the remaining power available in the neighbor nodes of the affected area, and send a command to restore power if available.
    - ∗ If enough power for restoring the affected area is not available, ignore the power shortage, continue monitoring the availability of power, and restore power in the affected area once it becomes available.

- If the report is about a neighbor BMAC,
  - If the neighbor is not responding properly, restart the neighbor agent.
  - If power is needed for the neighbor, follow the power restoration sequence.
- If the report is about fault restoration,
  - Inform the neighbor nodes of the affected area about the fault restoration.
  - Send a command to the faulted area to activate the generator.
- If a query message is received from other BSACs,
  - Respond to the query using the knowledge stored in the database

Unified Modeling Language (UML) is a general-purpose modeling language used in the field of object-oriented software engineering. The UML is used to specify, visualize, modify, construct, and document the artifacts of an object-oriented software system under development. The UML representation of the entire agent model is shown in Appendix D. In this UML representation, the activity and behavior of all the individual agent modules in the cluster are explained.

Inter-Cluster Communication

The communication end point is a communication agent. So all the communication to and from the outside world is routed through this agent. If the replicated agents in one of the monitors identify a fault in the node, they notify the communication agent. Now, the communication agent will send a report to the supervisor, and it will wait for a response. While waiting for the response from the supervisor, it will neither send any messages to the node nor accept any messages from the node. All the monitors in the same area can communicate with each other and with the supervisor of their corresponding area. The monitors in a particular area cannot communicate with the monitors in the other area. This communication should take place through the supervisor of that area. For example, if a monitor from the first area wants to communicate with a monitor in the second area, this communication request should be sent to the supervisor of the first area. Then this supervisor communicates with the supervisor of the second area. The second area supervisor obtains the information and sends it back to the first area supervisor, from where it reaches the local monitor that requested the information.

Knowledge sharing. All the agents are capable of communicating with each other. The knowledge sharing is done at some specific time intervals, which ensures the network is not overloaded because of knowledge sharing messages. Knowledge sharing can be done using either *push* or *pull* protocols. The push method is known as knowledge exchange, where a monitor pushes the information that it knows to its neighbors. The pull method is known as knowledge query, where a monitor pulls the information from a remote node through querying.

Knowledge exchange. In knowledge exchange, monitors periodically share their knowledge with their neighbors through the Internet (Figure 22). The interval between each shared message is optimized so that it does not affect the critical communication between the power node and the monitor associated with it. Because of this knowledge exchange, the agent clusters are not only aware of their own node, but also they are able to monitor their neighbor nodes' activities.

Knowledge query. If an agent cluster needs to know anything specific about its neighbor node, it can request that information (Figure 22). This query may not be needed if the neighbors periodically send their information. If the information is not sent or it is lost, then the BMAC can send a knowledge query message to the intended neighbor. If the neighbor is alive, it will immediately respond to the query by sending the requested information. If the neighbor is not alive or not responding within a certain period of time, the requesting BMAC will declare that agent cluster as non-responsive and report this activity to its BSAC. So the knowledge sharing not only shares the agent cluster's knowledge but also helps for an agent cluster to know the status of other agent clusters.

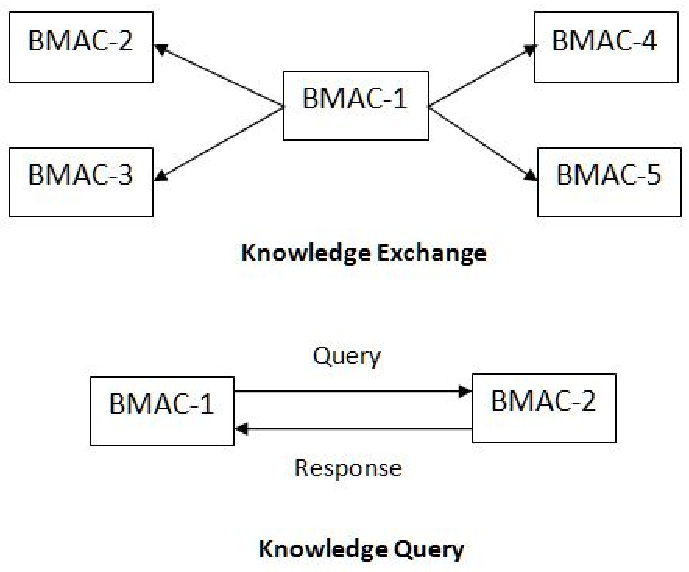


Figure 22. Knowledge sharing between agent clusters: showing both knowledge exchange and knowledge query. BMAC-i is Bus Management Agent Cluster.

Inter-cluster communication intelligence. Communication among the monitoring agent clusters improves the system performance by making the agent clusters more reliable and fault-tolerant. This communication also speeds up the fault restoration. As the BSAC can communicate with all other BMACs, it is easier to solve a problem in the particular BMAC.

Consider an example where the power nodes, BMACs, and BSACs are communicating among themselves to solve a problem (Figure 23). The BMAC in Node 3 senses a problem and reports to its BSAC that is located in Node 6 (Step 1; shown in solid lines). The BSAC then communicates with the neighbors of the faulty area that are $BMAC - 2$ and $BMAC - 5$ (Step 2; shown in dashed lines). The neighbors respond for BSAC's communication message (Step 3; shown in dotted lines), and the BSAC sends a response back to $BMAC - 3$ (Step 4; shown in solid lines). Finally, this $BMAC - 3$ then sends a control signal to the power node to draw power for re-powering the affected area.

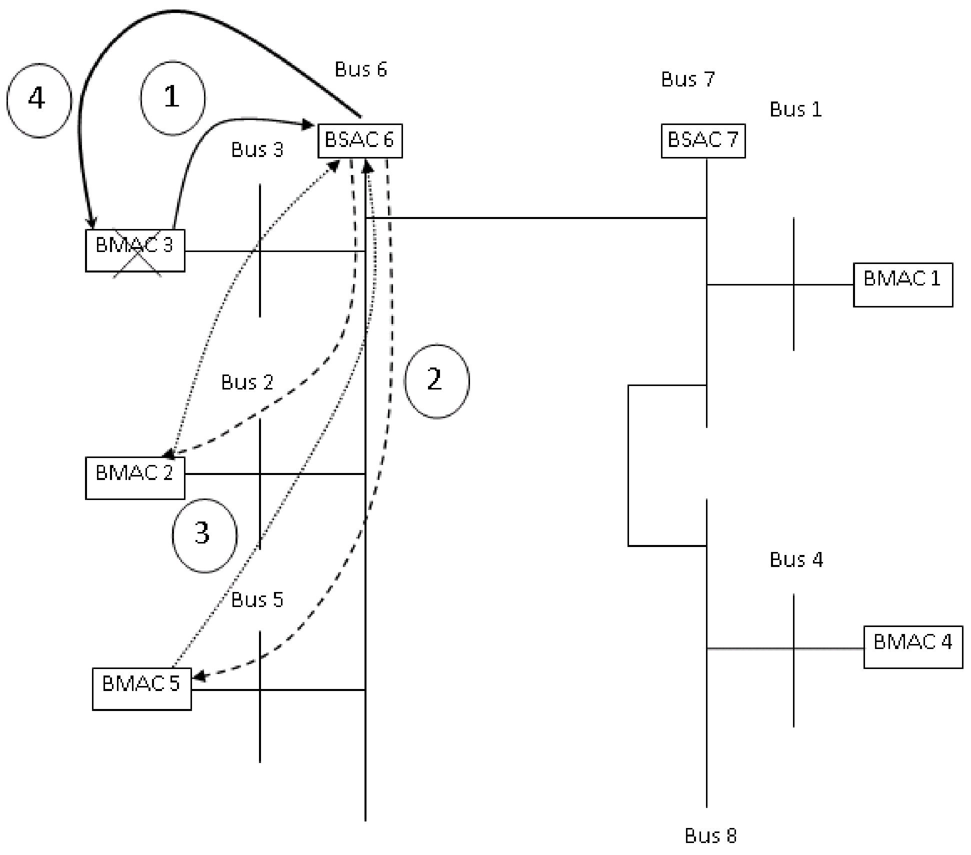


Figure 23. Example for communication intelligence. BMAC-i is Bus Management Agent Cluster, BSAC-i is Bus Supervisory Agent Cluster, and $Bus - i$ denotes the bus number. Circled numbers show the steps involved in fault restoration.

Agent communication allows the problem to be solved in three phases.

1. *Problem identification phase:* In this phase, the replicated agents in the source monitoring agent cluster identify any abnormal condition in the generator parameters either by themselves or by receiving some information from their neighbors. Once the problem is identified, the monitor tries to solve the problem and sends a corrective control signal back to the generator, and hence, it continues with the third phase. If the problem is not self-solvable, then the monitor continues with the next phase.
2. *Reporting phase:* In this phase, the monitor is not able to solve the problem and needs supervision. So the monitor generates a report containing the problem description and the source of the problem, sends this report to its immediate supervisor, and waits for a response. During the waiting time, the source monitor does not process messages received from the plant to avoid sending multiple fault report messages.
3. *Command generating phase:* In this phase, the source monitor receives a response from its supervisor and generates appropriate commands to activate the solution found by the supervisor. This control signal is sent to the generator node, and an acknowledgement is sent back to the supervisor.

The supervisor is the higher level agent cluster that has a global view of the entire system. It also has the ability to communicate with all the monitors in its area and the supervisors in other areas, which facilitates the supervisor in obtaining information from any monitor in the entire agent network. As the supervisor knows the design of the entire system, it is considered to be more knowledgeable than the monitors. Because of the communication capability, the supervisor can obtain the present measure of electrical parameters from any part of the power network. So the supervisor is able to find out problems and solves them. The solution is sent to the monitor of the affected area, from where it reaches the actual power node. After making the corrective action, all agent clusters continue to work normally.

States of Agent Clusters

The behavior of agent clusters is described using state transition diagrams that are built on the system's finite number of states. The behavior is analyzed and represented in a series of events that could occur in one or more possible states.

States of monitor. The execution states of a monitor explain the series of steps carried out by a monitoring agent cluster in any node of the power network. There are five states for a monitor (Figure 24).

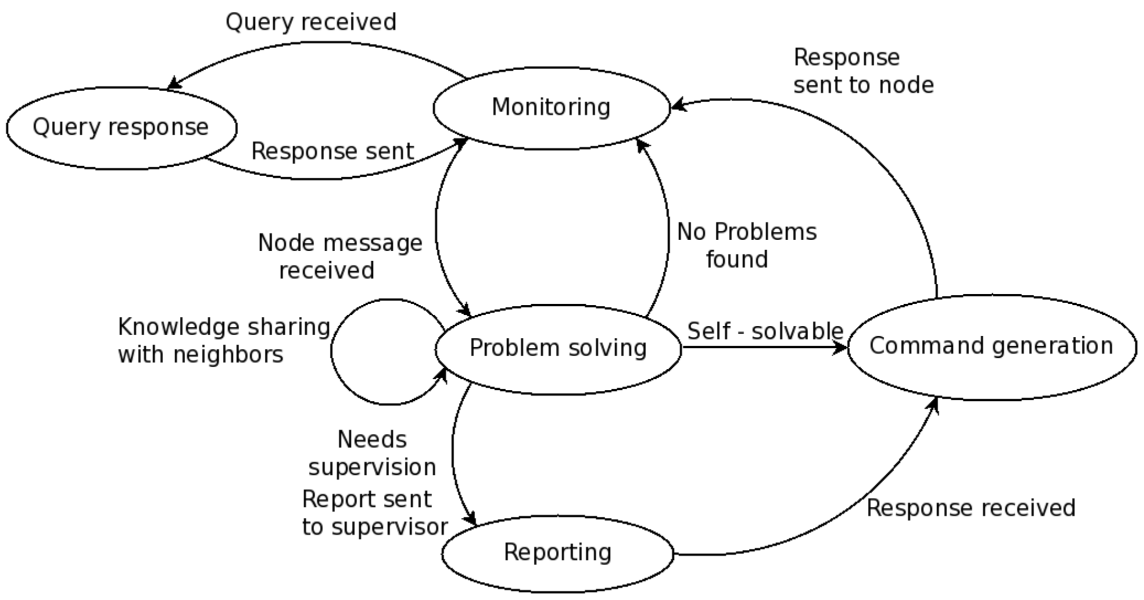


Figure 24. State transition diagram for a single BMAC. BMAC is the Bus Management Agent Cluster.

The following description shows when these states are switched and explains the activities performed in each state.

1. *Monitoring state:* This state is the initial state for all monitors. The monitoring agent cluster waits for a message either from the power node or from other agent clusters. If the received message is a query from other monitors, the state is changed to *query response state.* If the monitor receives a message from the power node, it switches its state to the *problem-solving state*, where it analyzes the power system with the electrical parameters received from the node.
2. *Query response state:* In this state, the queries from remote agent clusters are handled. The query can be a knowledge query from another monitor, a query message from its supervisor, an AMS query, or a DF query. The response for the query is generated by the appropriate agent in the cluster and is delivered to the query seeking agent.
3. *Problem-solving state:* This state is the main state of a monitor. In this state, the power node parameters are analyzed by the replicated computational agents. If the monitor is able to handle the problem, it can switch its state to the *command generation state* and send a command back to the plant to solve the identified problem. If the monitor is not able to find a solution, it switches its state to the *reporting state*, where a report is sent to its immediate supervisor.
4. *Reporting state:* In this state, a report about the problem is generated by the monitor that includes the description of the problem and its source. This report is sent to the supervisor. The monitor will not process further messages from the node until it receives a response from its supervisor, and hence, the monitor avoids

making the problem worse by sending a wrong control signal. Once a response is received from the supervisor, the state is switched to the *command generation state.*

5. *Command generation state:* This state is the final state in the problem-solving cycle, where a command/control signal is generated based on the solution found for the problem. This signal is sent to the power node, and the state is again changed back to the *monitoring state* for receiving further messages.

States of supervisor. The execution states of supervisor describe the steps needed for problem-solving and WAC operation (Figure 25).

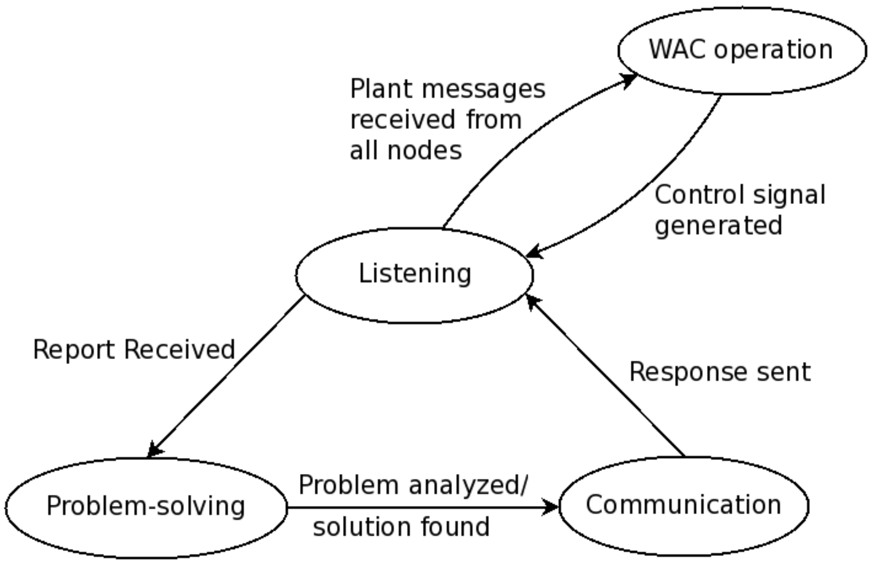


Figure 25. State transition diagram for a single BSAC. BSAC is Bus Supervisory Agent Cluster, and WAC stands for Wide-Area Controller.

The following are the steps involved in a supervisor's execution.

1. *Listening state:* This state is the initial state for a supervisor. In this state, the supervisor is waiting for messages from the monitors and the power node, where the supervisor is located. All the monitors situated in a particular area exchange

their node messages with their supervisor. Once the supervisor receives messages from all the monitors, it switches its state to the *WAC operation state*. If the supervisor receives a fault report from one of the monitoring agents, the state is changed to the *problem-solving state.*

2. *WAC operation state:* In this state, the wide-area controlling is performed. The inputs from all nodes are given to the WAC, and a control signal is generated for each node. This control signal is sent back to the individual monitors from where it reaches the power node to which the monitor belongs. After sending the control signal, the state is switched back to the *listening state.*
3. *Problem-solving state:* This state is reached if a problem is found in one of the monitors and a report is received. The supervisor analyzes the problem and computes a solution to resolve the problem. The state is then switched to the *communication state*, where it communicates with other monitors in its area to resolve the problem.
4. *Communication state:* Once a solution is found, the supervisor switches its state to this state. In this state, the supervisor contacts the neighbors of the affected node monitor to find a corrective action. For power restoration problems, the supervisor polls the neighbor monitors for available power, and those monitors send a vote message containing the present available power in their node. After a corrective action is found and a response is sent back to the affected monitor, the supervisor switches its state to the *listening state* to be able to receive further messages.

Activity Sequence of the Entire System

The activity of the entire agent network is drawn as a sequence diagram (Figure 26). During normal conditions, the agent clusters exchange data, process the message received from the power node, and send a response back to the nodes. A faulty report case is clearly depicted, in which a BMAC senses a fault in its node and reports it to the BSAC, and the BSAC communicates with the neighbors of the affected area BMAC and solves the problem by sending a command back to the reporting BMAC.

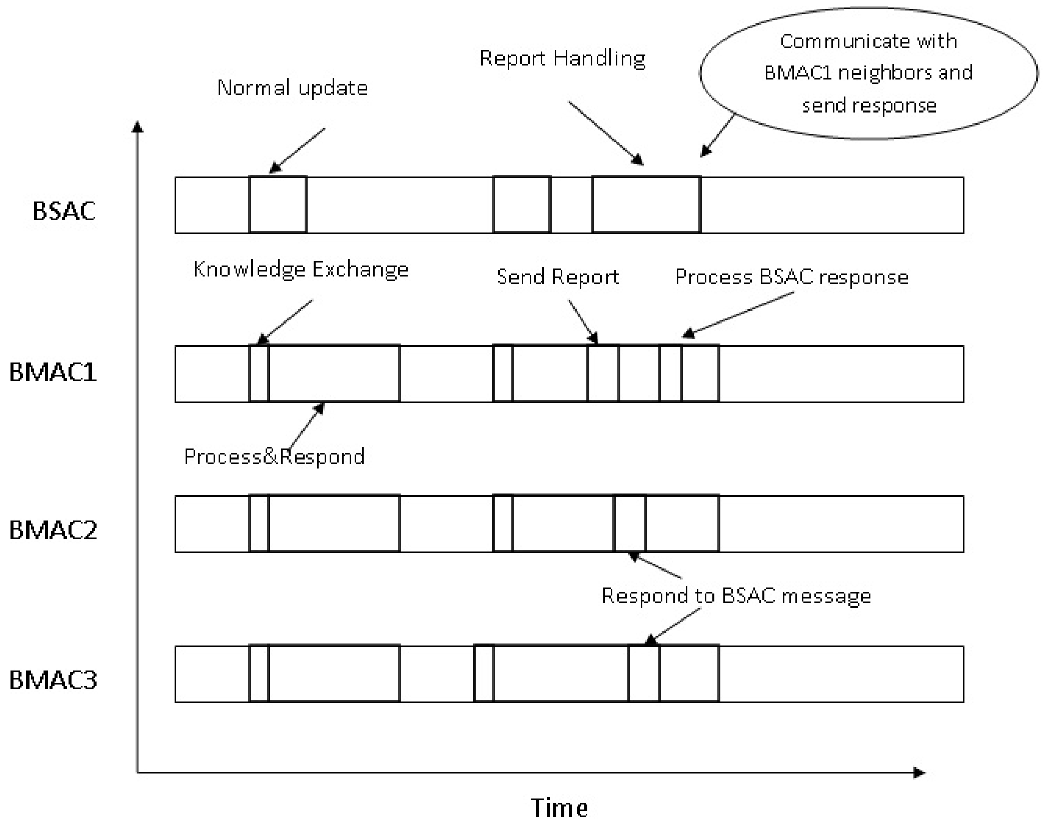


Figure 26. Timing diagram for the entire system. BMAC is Bus Management Agent Cluster. BSAC is Bus Supervisory Agent Cluster.

Agent Cluster Testing

To overcome the issues related to local control, the developed monitoring agent clusters and supervising agent clusters are placed on the same two-area and five-machine power system (Figure 27). In this power system model, monitors are placed on buses 1, 2, 3, 4, and 5 and supervisors are placed on buses 6 and 7. A fault is simulated in G_3 that causes a part of the area to be unpowered. The monitor located on that node identifies this problem and reports to the supervisor of its area, and the supervisor identifies the available power in the neighbor nodes of the affected area to re-power the affected area (Manickam, Kamalasadan, Edwards, & Simmons, in press).

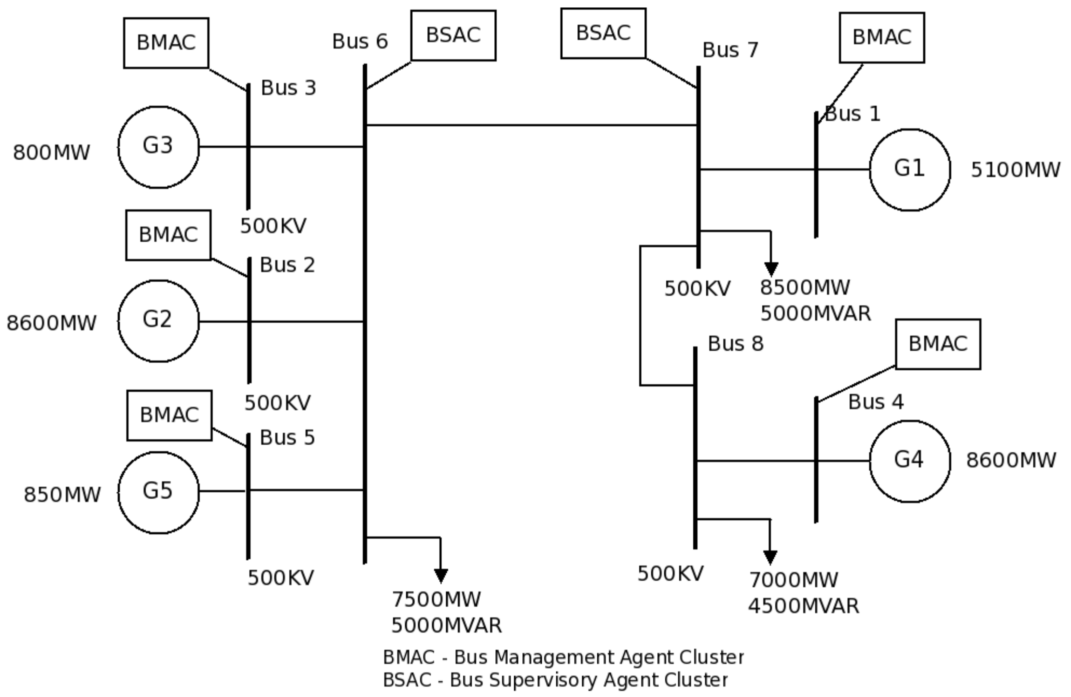


Figure 27. One-line diagram of a two-area model with agent clusters located in buses. Gi is the generator. Bus-i is the bus number.

In the node of G_3 (Figure 28), the breakers labeled *BRK1* separate the upper area (first sub-area) from the system. Similarly, the breakers labeled *BRK2* separate the lower area (second sub-area) from the system. Both of these breakers are used to simulate a fault in that area. The breakers labeled *BRK3* separate the generator from the rest of the power system. These breakers are used to isolate the generator from the system to protect it from intruders when the agent senses the occurrence of an intrusion. The breaker labeled *BRK11* connects the two loads while being closed. This breaker is used to power the load in the first sub-area when there is a fault in that area. There are two loads in Node 3. One load is connected to the first sub-area, and the second load is connected to the second sub-area. These two loads are critical and need to be given power all the time even when there is a problem in that node. The steps for fault simulation and fault restoration are explained separately.

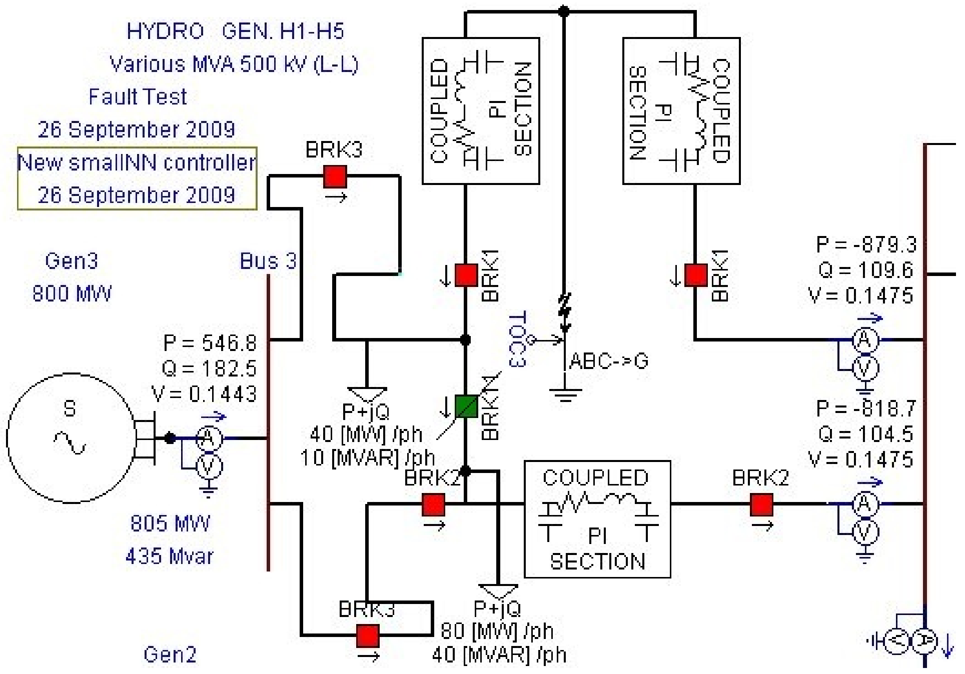


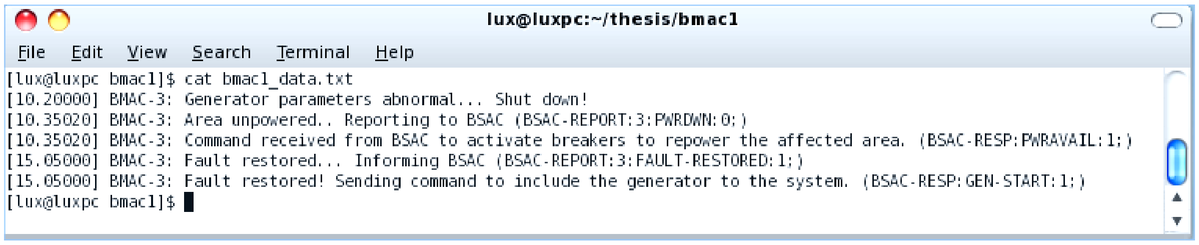
Figure 28. Generator 3 model for fault construction. BRK-i refers to the breaker number. Gen-i refers to the generator number. Bus-i refers to the bus number.

Fault simulation. There is a fault simulated in the first sub-area at $t = 5$ seconds. *BRK1* breakers are opened, and the power to the load connected to the first sub-area is stopped.

At $t = 10$ seconds, *BRK2* are also opened, and this operation stops the power supplied to the second sub-area. This activity also causes the generator power to decrease because all the power drawing components are removed from the system. The monitor situated in that node senses all the breaker changes and the decrease in power. This agent cluster also monitors the speed of that generator. As the speed decreases, the monitor identifies the abnormality of the generator, and it sends a command to take the generator out of the system for protection. At this time, there is no source of power to supply the loads. The monitor immediately notifies the supervisor about this problem through the communication agent. The supervisor needs to know whether or not the power available in all other nodes of the same area is sufficient to restore power in the affected area. The supervisor knows all the information of the neighbor monitors, as the information is sent periodically. Thus, the supervisor looks into its database to analyze the system, and hence, it validates the fault report. If the power is available in a neighbor node, the supervisor sends a command to the monitor to get power from that neighbor. The monitor obtains the command from the supervisor and re-powers the affected area. It closes the *BRK2* to obtain power from the neighbor node and also closes the *BRK11*, which supplies power to the load connected to the first sub-area.

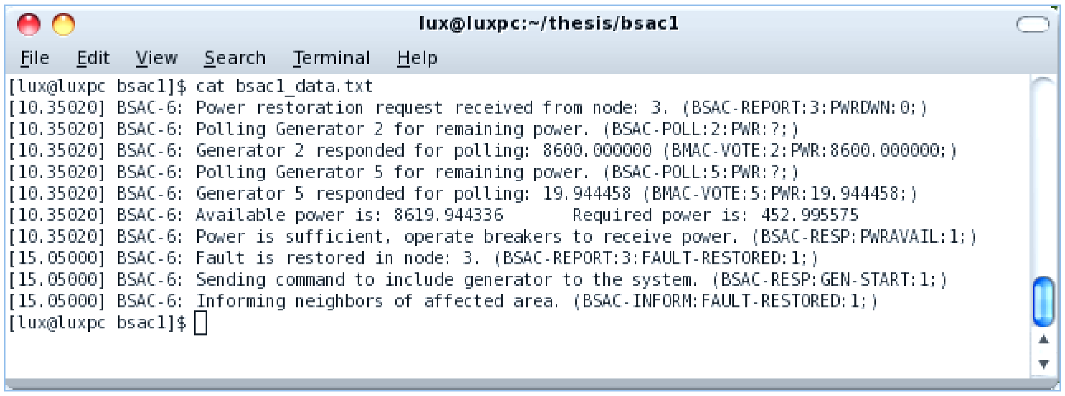
Fault restoration. After some time, the fault in both the sub areas is restored. At $t = 15$ seconds, both *BRK1* and *BRK2* are closed to simulate the fault restoration. As the monitor remembers the previous state of the parameters sent by the node, it senses the breaker change, and hence, it understands that the fault is restored and notifies the supervisor about this issue. The supervisor sends this information to the neighbors, who were supplying power to the affected area. An acknowledgement is sent to the supervisor by each neighbor. Then the supervisor sends a command to the monitor to include the generator into the system. The monitor includes the generator into the system before it opens the *BRK11*, which was used to supply power to the first sub-area from the second sub-area. After restoring the fault by including the generator that was offline earlier, a simple notification message is sent to the supervisor, and the supervisor understands that the problem is resolved.

The log messages stored in the monitoring agent cluster of the affected node and the supervising agent cluster are analyzed (Figure 29) to understand the communication between these agent clusters.



```
[lux@luxpc bmac1]$ cat bmac1_data.txt
[10.20000] BMAC-3: Generator parameters abnormal... Shut down!
[10.35020] BMAC-3: Area unpowered.. Reporting to BSAC (BSAC-REPORT:3:PWRDWN:0;)
[10.35020] BMAC-3: Command received from BSAC to activate breakers to repower the affected area. (BSAC-RESP:PWRAVAIL:1;)
[15.05000] BMAC-3: Fault restored... Informing BSAC (BSAC-REPORT:3:FAULT-RESTORED:1;)
[15.05000] BMAC-3: Fault restored! Sending command to include the generator to the system. (BSAC-RESP:GEN-START:1;)
[lux@luxpc bmac1]$
```

(a) Log messages for BMAC on Node 3



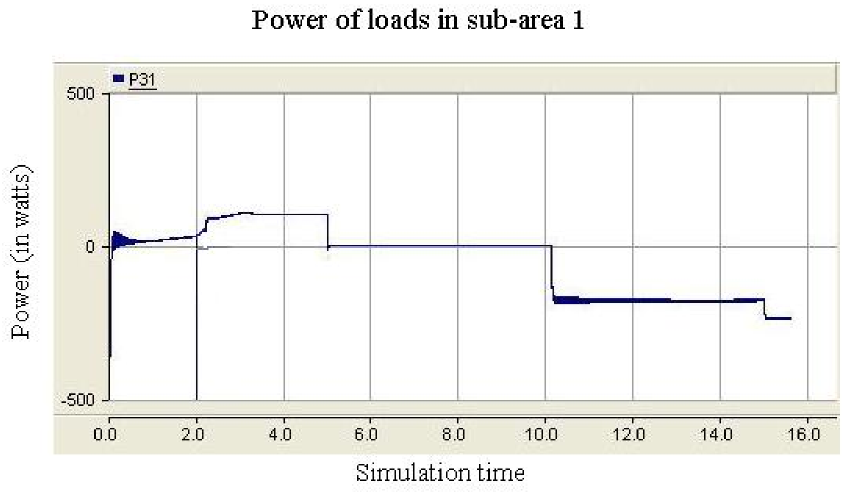
```
[lux@luxpc bsac1]$ cat bsac1_data.txt
[10.35020] BSAC-6: Power restoration request received from node: 3. (BSAC-REPORT:3:PWRDWN:0;)
[10.35020] BSAC-6: Polling Generator 2 for remaining power. (BSAC-POLL:2:PWR:?;)
[10.35020] BSAC-6: Generator 2 responded for polling: 8600.000000 (BMAC-VOTE:2:PWR:8600.000000;)
[10.35020] BSAC-6: Polling Generator 5 for remaining power. (BSAC-POLL:5:PWR:?;)
[10.35020] BSAC-6: Generator 5 responded for polling: 19.944458 (BMAC-VOTE:5:PWR:19.944458;)
[10.35020] BSAC-6: Available power is: 8619.944336       Required power is: 452.995575
[10.35020] BSAC-6: Power is sufficient, operate breakers to receive power. (BSAC-RESP:PWRAVAIL:1;)
[15.05000] BSAC-6: Fault is restored in node: 3. (BSAC-REPORT:3:FAULT-RESTORED:1;)
[15.05000] BSAC-6: Sending command to include generator to the system. (BSAC-RESP:GEN-START:1;)
[15.05000] BSAC-6: Informing neighbors of affected area. (BSAC-INFORM:FAULT-RESTORED:1;)
[lux@luxpc bsac1]$
```

(b) Log messages for BSAC on Bus 6

Figure 29. Log messages on BMAC of affected area and the BSAC with the communication messages between agent clusters. BMAC is Bus Management Agent Cluster. BSAC is Bus Supervisory Agent Cluster.

The format of the log message is as follows: the simulation time within square brackets (in seconds), the identity of the agent cluster, the description of the activity at that moment, and the communication message within brackets. When the monitoring agent cluster in the affected node identifies the fault, it sends a command at $t = 10.2$ seconds to shut down the generator. As the generator in that area is removed, that area becomes unpowered, and this incident is reported to the supervising agent cluster located in Bus 6 at $t = 10.35020$ seconds. Upon receiving the power restoration request, the supervisor polls the neighbors of the affected area agent cluster, G_2 and G_5, for the available power in their power nodes. Once the supervisor receives a response from the neighbors, the total available power is calculated, and a command is sent to the affected area monitor to get power from its neighbors. The monitor receives this response and activates corresponding breaker to receive power from its neighbors. The fault in Node 3 is restored at $t = 15.05$ seconds, and the monitor in that area alerts the supervisor about this restoration. The BSAC sends a command to include the generator into the system and informs the neighbors, who supplied power during the fault.

The power levels of the loads connected in Node 3 are plotted (Figure 30) to study the problem and its restoration. In the first sub-area, the power reaches zero at $t = 5$ seconds and in the second sub-area, power decreases to zero at $t = 10$ seconds. The monitor alerts the supervisor about this issue. The supervisor performs the power restoration steps, and the power is restored at $t = 10.1$ seconds. Within a time frame of 5 to 10 seconds, the power delivered from the second sub-area is high because the first sub-area is removed from the system. The negative power in the plot ensures the power is coming into the node from remote nodes.



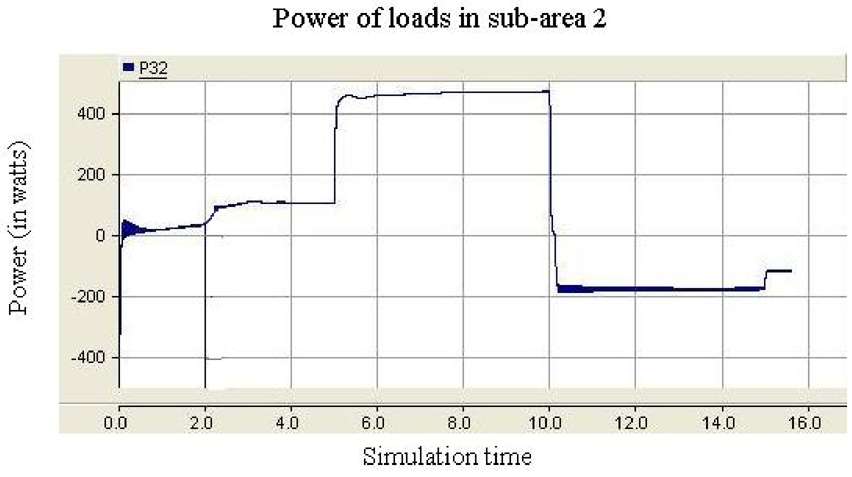


Figure 30. Power of the loads connected in Node 3. P31 refers to the power in the first area in Node 3 and P32 refers to the power in the second area.

The commands sent (Figure 31) by the agent cluster perform three different activities: generator shut down, re-powering the affected area, and including the generator after fault restoration.

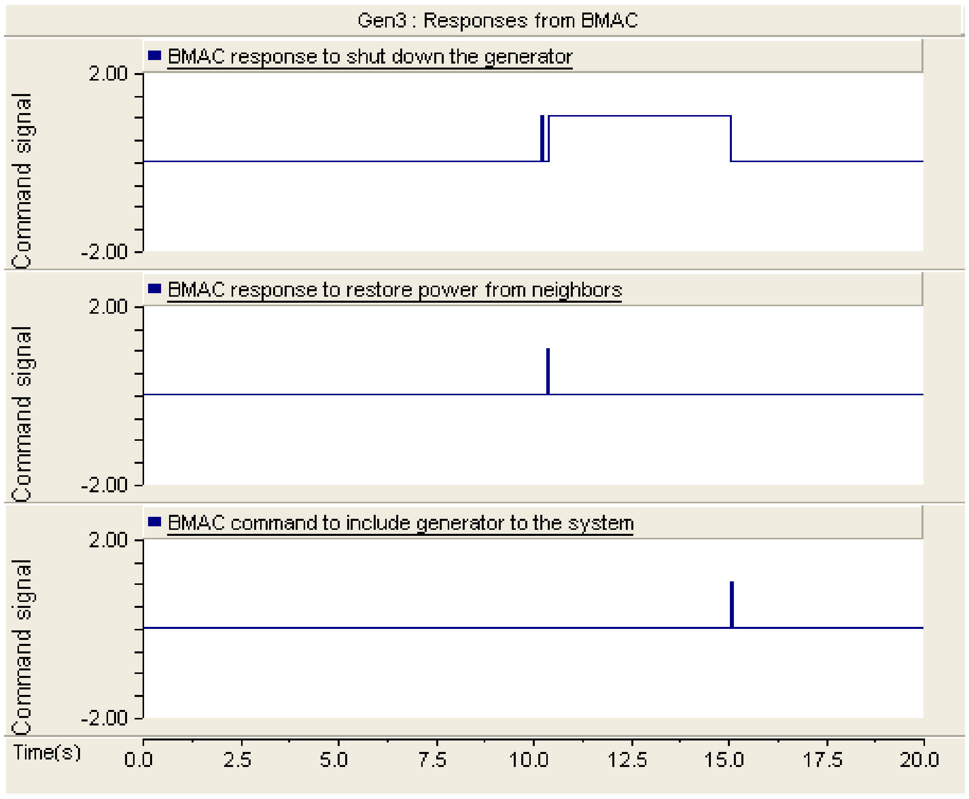


Figure 31. Agent responses for shutting down the generator, re-powering the faulted area, and including the generator after the fault restoration.

CHAPTER IV

WIDE AREA CONTROLLER DESIGN

The supervisors in our design are capable of obtaining a global view of the entire power system. This global knowledge is used for performing wide-area control. This chapter describes the design of the proposed controller that is based on adaptive-critic approaches. The training of the NNs is performed in two phases: offline and online. The mechanisms used for performing the offline training are explained in this chapter, and the results of offline training are presented. The online training techniques and results are explained in the next chapter.

Wide-Area Controller Design

The grid devices are usually controlled by local controllers. As we studied earlier, the local controllers do not guarantee better stability of the system performance. So these local controllers are coordinated by a supervisor (Figure 32). This supervisor receives plant present conditions and control signals from all the local controllers, thereby obtaining a global view of the entire power system. This global knowledge makes the WAC capable of providing better system stability during severe disturbances. The design of this controller is based on reinforcement learning (Figure 33). The control signal generated by this WAC is used to coordinate the activities of local controllers.

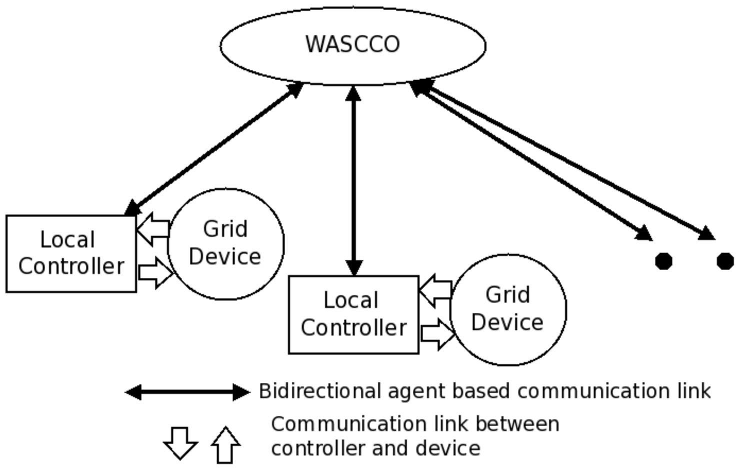

WASCCO
Local Controller
Grid Device
Local Controller
Grid Device
Bidirectional agent based communication link
Communication link between controller and device

Figure 32. Wide-area controller design. WASCCO is the Wide-Area Coordinating System Centric Controller and Observer.

Whether or not the training uses the reinforcement or the supervisory method, the process of training a NN requires computing an error value that describes how the NN's output varies from the desired value. Then a method that uses this error value to improve the performance of the NN must be found. The algorithm that defines that calculation is called backpropagation. Originally described by Werbos (1994), the method has since been adapted to wide use in training NNs (Lendaris & Shannon, 1999). In its essence, backpropagation is an implementation for calculating derivatives using the chain rule. Using the derivatives allows us to calculate the sensitivity of each component of the NN to the error and adjust those components so that the error can be reduced to zero.

Since we have a plant that we are trying to model, that plant can act as a teacher (in supervisory mode) to guide the learning of NN by showing it the correct output values for any set of inputs. We will first describe the forward calculations used to calculate the NN output. Then we will describe how backpropagation is performed.

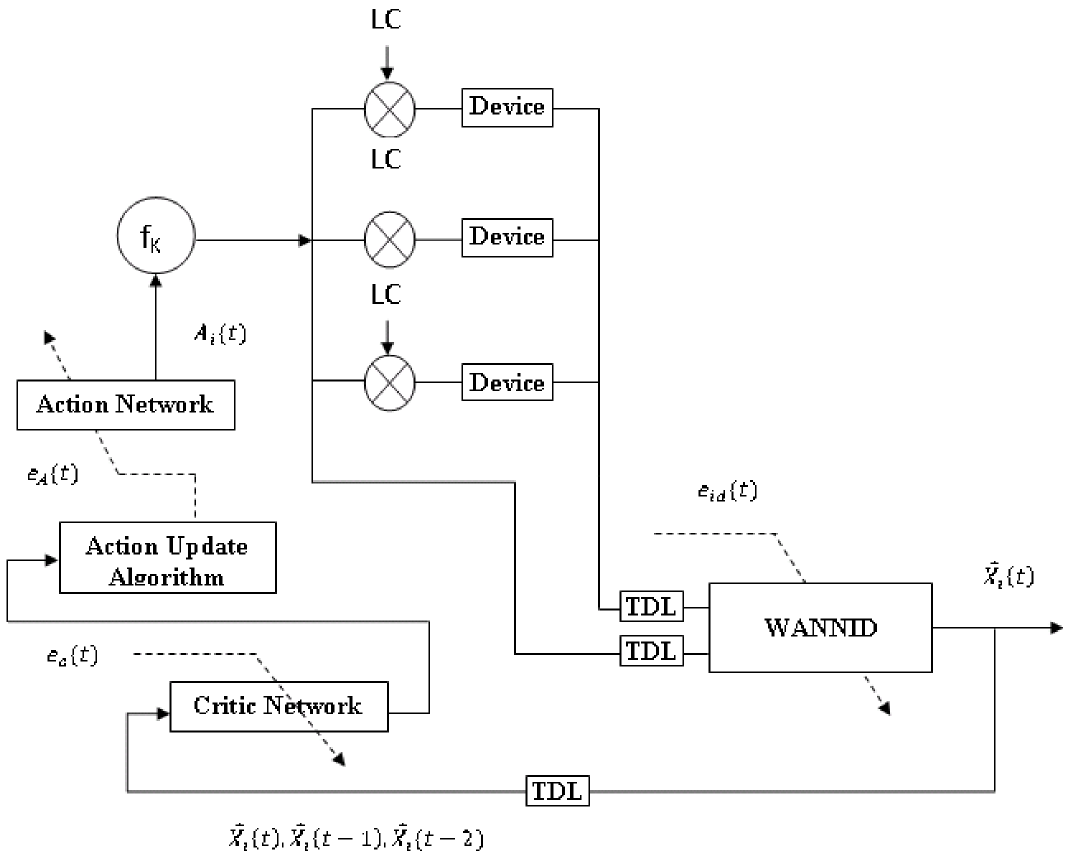


Figure 33. Wide-area controller block diagram. WASCCO is the Wide-Area Coordinating System Centric Controller and Observer. U_a = controller output. Error function adjustment is e_{ad}. X_i is plant output. X_m = reference model states. U = combination function output. U_a is controller output. Combination function is f_k. A_i is the output from the action network. Output of action update algorithm is e_A. Critic error is e_c. Identifier error is e_{id}. λ = critic output. $\hat{X}_i(t)$ = identifier network output. Time is t.

At the beginning, the weights are initialized to small random values. Then the steps involved in training an adaptive-critic controller using backpropagation are given below.

1. Apply an input to the NN.
2. Calculate the NN output.
3. Compare the output of the NN with the plant output to generate an error signal.
4. Adjust the weights and biases using the backpropagation algorithm.

5. Repeat these steps until the error reaches an acceptable value.

The following paragraphs examine these steps in more detail and also describe the wide-area NN identifier (WANNID) components and interconnections (Figure 34).

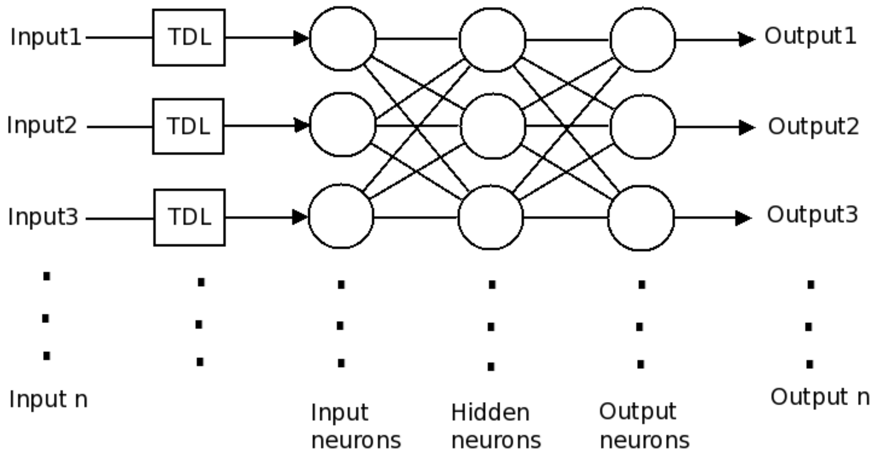


Figure 34. Interconnections of a neural network. TDL is time delay logic. Number of inputs and outputs vary for different types of networks.

As discussed earlier, the input to the NN (Step 1) is the array a0 and consists of the plant and control values, along with their delayed values. Layer 1 output (a1) is

$$a1 = f1(W1.a0 + b1), \tag{1}$$

where the array $W1$ contains the weights at the first layer, $a0$ is the input, and elements of $b1$ are the input biases. In effect, each input is scaled by the weights and added to the $b1$ array before being fed to the input of the logistic function. The inputs are assumed to be coming from three generators, and each generator has three values; therefore, we have nine values in total. Using a delay of three, the number of inputs to the NN become 27. So the product of the 27 element input array and the 20 by 27 element $W1$ matrix yields a 20 element array at the $b1$ position. Each element of this product is summed with the 20 elements of array b1 and fed to the inputs of the sigmoid function. Thus, layer 1 output (a1) is

$$a1 = f1(W1.a0 + b1), \tag{2}$$

where $W1$ elements are the weights at the first layer, $a0$ is the input, and elements of $b1$ are the input biases. The function $f1$ is the sigmoid that was previously discussed.

Similarly, layer 2 output (Step 2) value is

$$a2 = f2(W2.a1 + b2), \tag{3}$$

where the array $W2$ contains the weight values at the second layer, elements of $b2$ are the biases for layer 2, and $f2$ is a linear function for the output layer.

The error (Step 3) is calculated as

$$e = t - a2, \tag{4}$$

where t is the desired plant output, and $a2$ is the NN output.

To use the backpropagation algorithm (Step 4), we must compute the sensitivities of each layer's output to the calculated error (Step 3), starting with the output layer and working backward to the input layer. Since the output layer is a linear function, its derivative is unity. The logistic function used in the NN is

$$F(x) = \frac{1}{1+e^{-x}}. \tag{5}$$

The derivative of Equation 5 is $(1 - a1)(a1)$, where $a1$ is the output of the logistic function (Figure 34). Thus, the sensitivity of the output layer is computed as

$$S2 = -2.[1].e. \tag{6}$$

The first layer sensitivity is then calculated by backpropagating the sensitivity from the output layer to the first layer as

$$S1 = (1 - a1)(a1).W2.S2, \tag{7}$$

where $W2$ is the weight matrix of the output layer, and $S2$ is the sensitivity that was previously calculated. Now that sensitivities $S1$ and $S2$ are known, the weights and biases may be adjusted. The weight adjustments and new values of $W2(t+1)$ for the output layer ($W2$) are calculated as

$$W2(t+1) = W2(t) - \alpha.(S2).a1, \tag{8}$$

where α is the learning rate. Biases for the output layer are calculated as

$$b2(t+1) = b2(t) - \alpha.(S2). \tag{9}$$

New layer 1 weights (W1(t+1)) are calculated as

$$W1(t+1) = W1(t) - \alpha.(S1).a0. \tag{10}$$

Finally, new bias values for the first layer $(b1(t+1))$ are calculated as

$$b1(t+1) = b1(t) - \alpha.(S1). \tag{11}$$

To determine whether or not to continue training (Step 5), the Root Mean Square (RMS) error is calculated by squaring and then summing the error values of each training sample. The square root of the sum is then taken at the completion of each training iteration. The error so calculated is compared to previous values to see if effective training is continuing or if training may be terminated because the error has either reached an acceptable value or ceased to decrease at a reasonable rate.

All the NNs are configured with optimum number of hidden neurons and delays (Table 5). The inputs and outputs may vary depending on the type of NN. The inputs given to the identifier NN are rotor speed, terminal voltage, and the control signal used for controlling the generators. The inputs given to the action and the critic NNs are rotor speed and terminal voltage of the generators.

Table 5

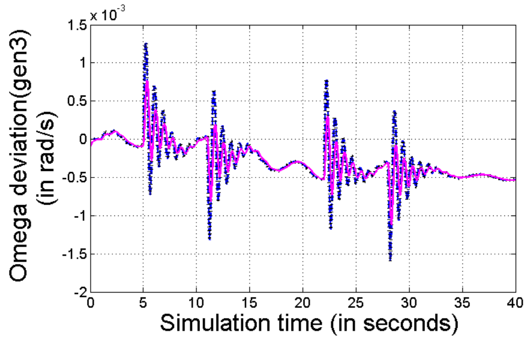
Configuration of Neural Networks

Neural network	Inputs	Delays	Hidden layers	Outputs	Input signal	Output signal
Identifier	9	3	15	6	omega, Vt, ctrl	omega, Vt
Action	6	3	30	3	omega, Vt	ctrl signal
Critic	6	3	13	6	omega, Vt	lambda

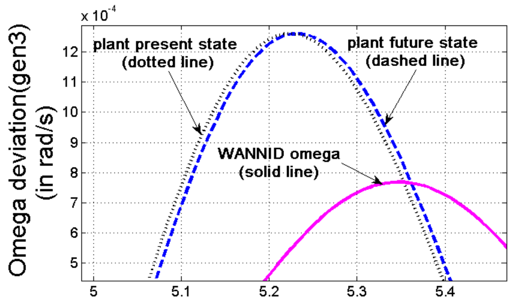
Note. Omega is rotor speed, Vt is terminal voltage, ctrl is control signal, and lambda is the output of critic network.

Identifier Offline Training

The NN used to identify the plant is initially trained offline using a wide-area power system model that is controlled by a PSS. The inputs and outputs of the power system model are captured and fed into the NN in a training procedure. The rotor speed output (omega or ω) of the identifier is graphed (Figure 35) to compare it with the plant output after a single pass of offline training. A similar comparison is made (Figure 36) for the terminal voltage (Vt) output.

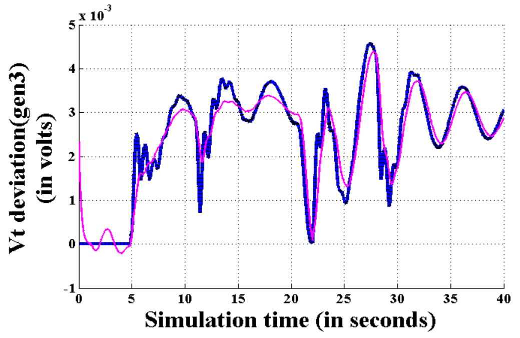


(a) Original

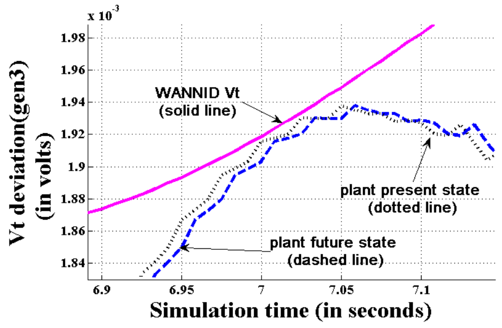


(b) Zoomed

Figure 35. Offline training of WANNID on Generator 3 for omega (first pass). WANNID refers to the wide-area neural network identifier. Omega is rotor speed.



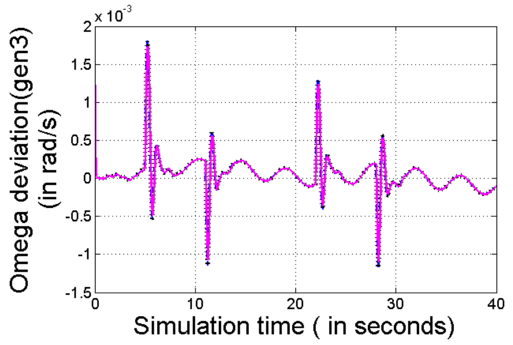
(a) Original



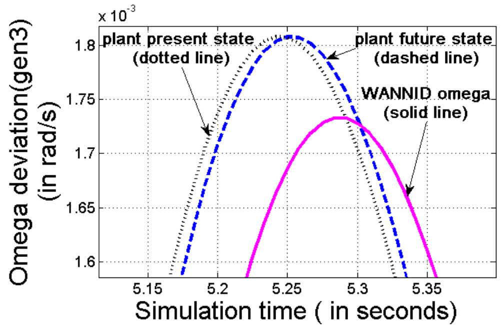
(b) Zoomed

Figure 36. Offline training of WANNID on Generator 3 for Vt (first pass). WANNID refers to the wide-area neural network identifier. Vt is terminal voltage.

Even with a single pass, the identifier tries to mimic the plant. The presented figures show the activity for only a single generator in the model. But the NN is fed with the outputs from all the synchronous generators in the system. After 10 passes of offline training, the learning of the identifier is complete (Figure 37 and 38). Here we can clearly see the accuracy of the identifier output. In the original graph, the plot for the identifier output exactly matches the output of the plant. An error value is calculated based on the output of the WANNID and the desired value. This value is used as an error signal for changing the weights and biases.

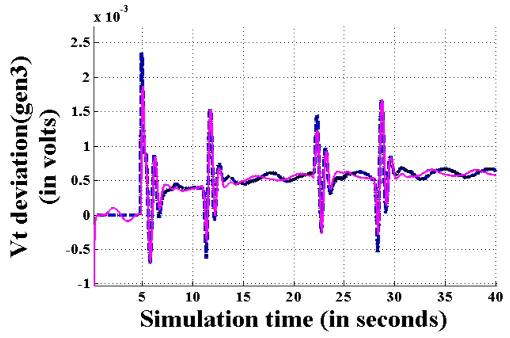


(a) Original

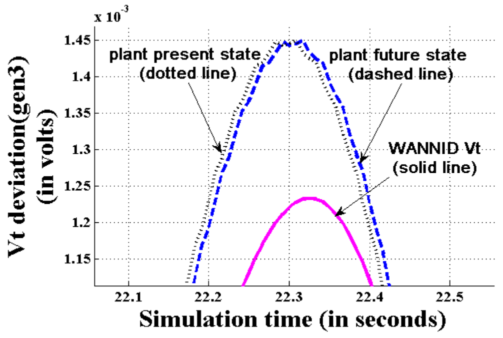


(b) Zoomed

Figure 37. Offline training of WANNID on Generator 3 for omega (after 10 passes). WANNID refers to the wide-area neural network identifier. Omega is rotor speed.



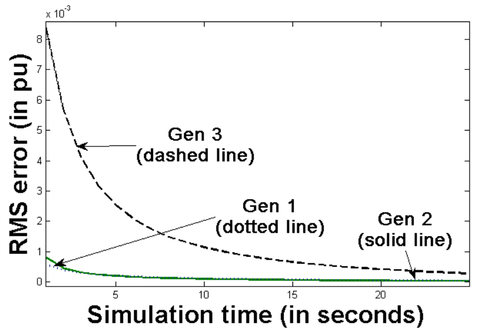
(a) Original



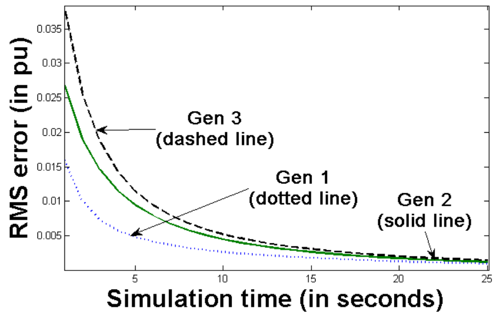
(b) Zoomed

Figure 38. Offline training of WANNID on Generator 3 for Vt (after 10 passes). WANNID refers to the wide-area neural network identifier. Vt is terminal voltage.

The RMS error is calculated using the error value captured at the end of each simulation step. This single error value is stored at the end of each pass and plotted after the completion of all passes (Figure 39). We can see from the plot that the error is getting minimized as we continue running several passes. At the end of the last pass, the error starts stabilizing and ensures the learning of WANNID is performed well.



(a) Omega



(b) Vt

Figure 39. RMS error of WANNID for both omega and Vt. WANNID is wide-area neural network identifier. RMS is Root Mean Square. Omega is rotor speed. Vt is terminal voltage.

Dual Heuristic Dynamic Programming Algorithm

In an adaptive-critic based controller, there is no need to have a known (best) value of the control function. Instead, only a desired cost function J is needed:

$$J(t) = \sum_{k=0}^{\infty} \gamma^k U_i(t+k). \tag{12}$$

$U_i(t+k)$ is the utility or local cost function, while γ is the discount factor needed to maintain the solution as a finite horizon problem with a limit on the upper bound of the solution. By selecting an appropriate value of $(0 < \gamma < 1)$, we can weight the future values of the utility function and effect the DHP convergence process (Swann, 2010). A value of zero uses only the present value of the utility function as the desired optimization, and a factor of unity considers all future values of the utility function. The critic will learn to approximate the derivative of the J function with respect to the plant states. Plant states considered here are rotor speed (omega or ω) and terminal voltage (Vt). The critic learns this approximation by minimizing the error expressed by the summation

$$||E|| = \sum_t e_c^T(t)\, e_c(t) \tag{13}$$

over time period t, where the value of e_c at each time period is given by

$$e_c(t) = \frac{\partial J(Y(t))}{\partial Y(t)} - \gamma \frac{\partial J(Y(t+1))}{\partial Y(t)} - \frac{\partial U(t)}{\partial Y(t)}. \tag{14}$$

Each component part $\partial(.)/\partial Y(t)$ of Equation 14 is a vector that contains the partial derivative of the scalar $(.)$ with respect to the components of the vector Y. For the DHP, applying the chain rule for derivatives results in

$$\frac{\partial J(t+1)}{\partial Rj(t)} = \sum_{i=1}^{n} \lambda i(t+1) \frac{\partial Ri(t+1)}{\partial Rj(t)} + \sum_{k=1}^{m} \sum_{i=1}^{n} \lambda i(t+1) \frac{\partial Ri(t+1)}{\partial Ak(t)} \frac{\partial Ak(t)}{\partial Rj(t)}, \tag{15}$$

where $\lambda_i(t+1) = \partial J(t+1)/\partial R_i(t+1)$, and n and m are numbers of the outputs of the model and action NNs respectively. Using Equation 15 allows us to express each of the n components of $e_c(t)$ in Equation 14 as

$$e_{cj}(t) = \frac{\partial J(t))}{\partial Rj(t)} - \gamma \frac{\partial J(t+1)}{\partial Rj(t)} - \frac{\partial U(t)}{\partial Rj(t)} - \sum_{k=1}^{m} \frac{\partial U(t)}{\partial Ak(t)} \frac{\partial Ak(t)}{\partial Rj(t)}. \tag{16}$$

Training the DHP controller

Training of the complete DHP controller follows a sequence of steps as detailed by Prokhorov and Wunsch (1997). As the identifier NN and action NN were previously trained offline, it is not necessary to adjust them during this process. An iterative procedure is used (Table 6), where the critic's weights are adjusted while holding the action's weights fixed, and then the action's weights are adjusted while the critic's weights are fixed (Figure 40).

Table 6

Critic NN Training Steps

Step	Action	Comment
1	Initialize $t = 0 and R(0)$	Set up for loop
2	$\lambda(t) = f_C(X(t), W_C)$	compute critic output λ (lambda)
3	$A(t) = f_A(X(t), W_A)$	compute action output (A)
4	$X(t+1) = f_M(X(t), A(t), W_M)$	compute identifier output(X)
5	$\lambda(t+1) = f_C(X(t+1), W_C)$	compute future lambda
6	Compute $e_c(t) and \partial\lambda(t)/\partial W_C$	compute error and derivatives
7	Update critic's weights W_C	adjust critic network weights
8	$t = t+1$	next time step
9	Repeat starting at step 2	loop

Note. NN is neural network. R and X = states. λ = critic output. f_C = critic function. W_C = critic weights. A = action output. f_A = action function. W_A = action weights. f_M = identifier function. W_M = identifier weights. Critic error is e_C. Time is t.

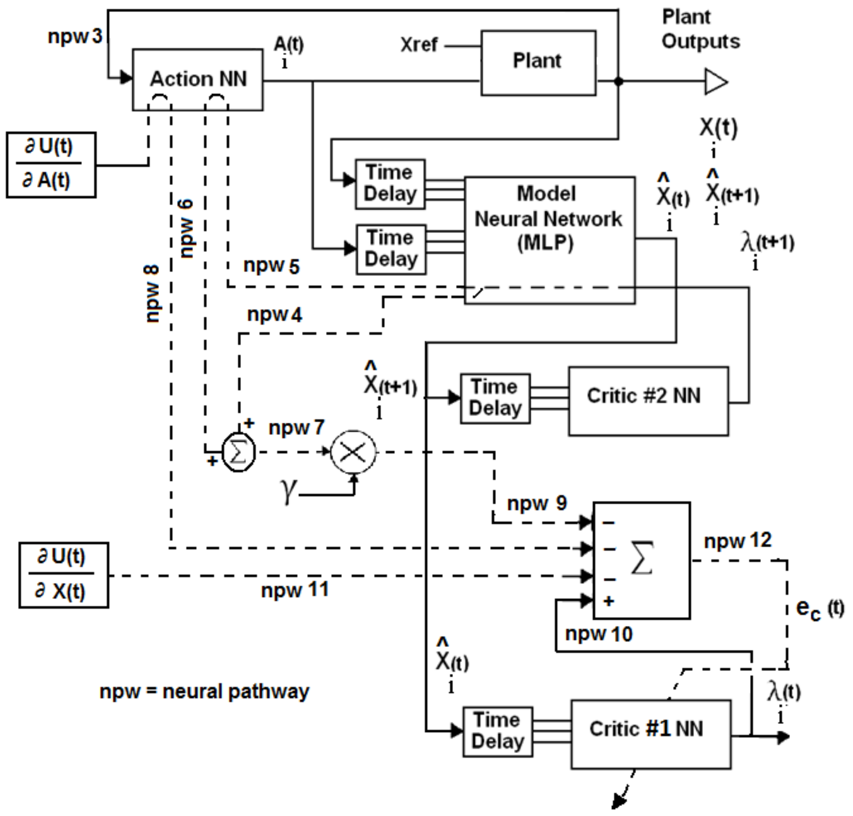


Figure 40. Critic network adaptation. NN = neural network. MLP is multi-layer perceptron. Neural pathway is npw. X_{ref} is the plant reference. X_i is the plant output. $\hat{X}_i$ is the identifier output. A_i is the action output. U is controller output. λ is critic output. Critic error is e_c. $\sum$ is the summation function. γ is the discount factor. Time is t.

The steps of the action's adaptation and training cycle (Table 7) are performed in the same manner as the steps previously describing the critic training (Table 6). During the action portion (Figure 41) of the DHP training cycle, we repeat training and weight adjustments as many times as necessary to minimize the error. To stay on the correct gradient descent path, the learning rate is kept small and adjustments are made until there is a little change in the weight and bias values within the action NN.

Table 7

Action NN Training Steps

Step	Action	Comment
1	Initialize $t = 0 and R(0)$	Set up for loop
2	$\lambda(t) = f_C(X(t), W_C)$	compute critic output λ (lambda)
3	$A(t) = f_A(X(t), W_A)$	compute action output (A)
4	$X(t+1) = f_M(X(t), A(t), W_M)$	compute identifier output(X)
5	$\lambda(t+1) = f_C(X(t+1), W_C)$	compute future lambda
6	Compute $e_A(t) and \partial A(t)/\partial W_A$	compute error and derivatives
7	Update the action's weights W_A	adjust action network weights
8	$t = t+1$	next time step
9	Repeat starting at step 2	loop

Note. NN is neural network. R and X = states. λ = critic output. f_C = critic function. W_C = critic weights. A = action output. f_A = action function. W_A = action weights. f_M = identifier function. W_M = identifier weights. Action error is e_A. Time is t.

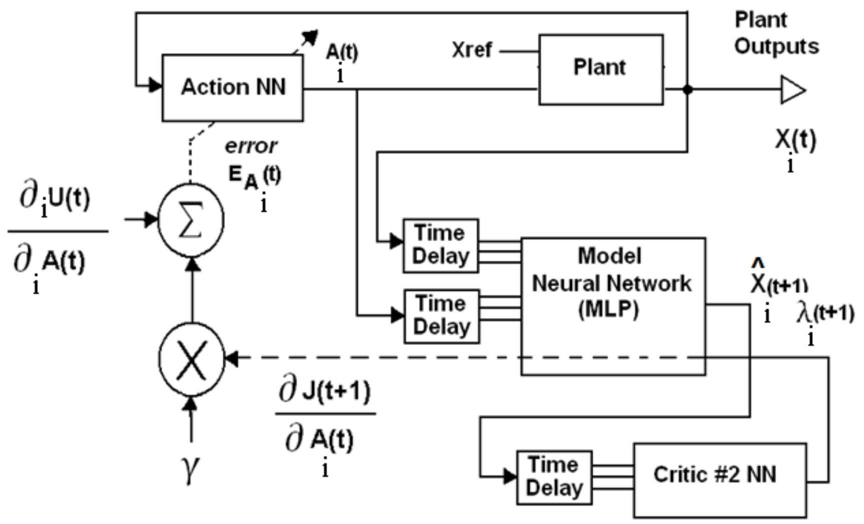


Figure 41. Action network adaption. NN = neural network. A_i is action output. MLP is multi-layer perceptron. X is plant output. $\hat{X}_i$ is the identifier output. U_i is the controller output. Action error is e_A. J is the Jacobian. λ is the critic output. $\sum$ is the summation function. The discount factor is γ. Time is t.

Action Offline Training

Before proceeding with the online training of the DHP controller, the action NN should be trained to act as a controller of some sort, a step that will ensure the system stability during the training of the other part of the DHP controller. For this purpose, the action network is trained to act similar to the PSS. During this training, the PSS is used to control the response of the plant. The plant state value along with the control value generated by the PSS are used to train the action network in the same manner as the identifier was previously trained.

The weights and biases of the action network are stored at the end of each pass and loaded during the next pass. These saved values are plotted (Figure 42) to see the learning of action. After 10 passes, the weights and biases are reaching a stable value from the initially assigned random value. After completing all passes of offline training, an experiment is conducted to test the controller learning. For this, the control signal generated by the action network and the PSS output are plotted (Figure 43) to measure the performance of action during this offline training.

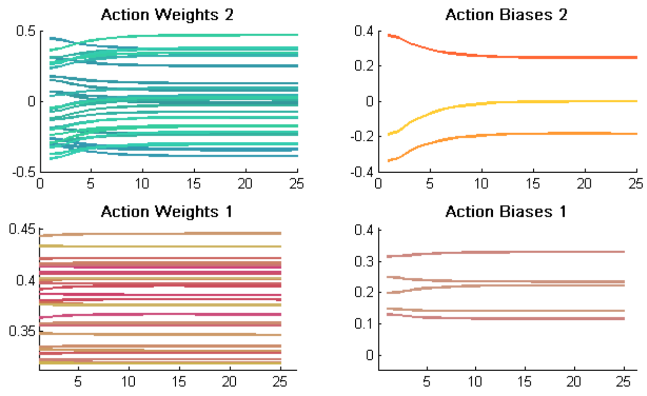


Figure 42. Action's weights and biases during the offline training. Weights1 and biases1 are the weights and biases used between the input layer and the hidden layer. Weights2 and biases2 are the weights and biases used between the hidden layer and the output layer.

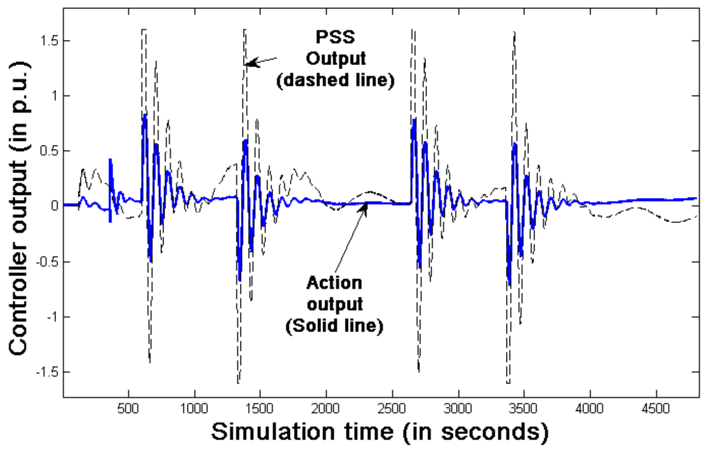


Figure 43. Comparison of the offline trained action output and the PSS output. PSS is Power System Stabilizer.

The graph shows that the action is performing similar to the PSS, and the use of some gain in the action output will make it perform much better. To evaluate the learning of action network, the RMS error is captured at the end of each pass. The stored error value is plotted after completing all passes (Figure 44).

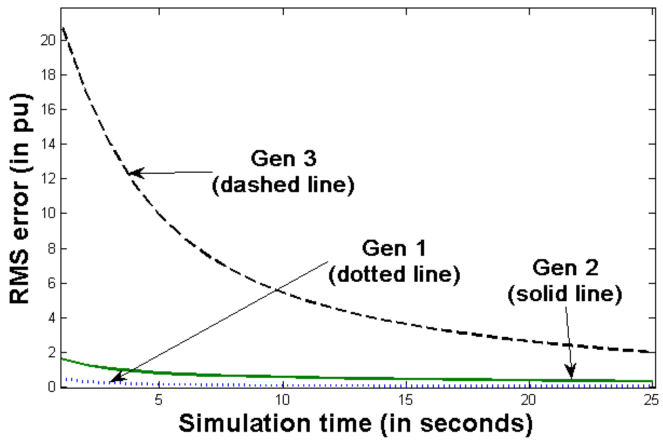


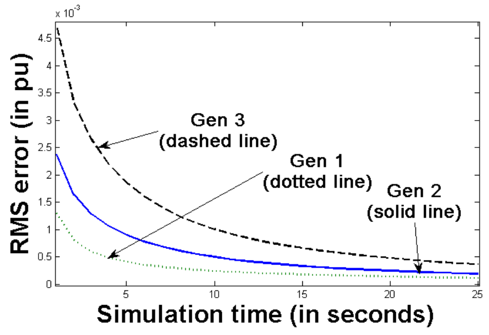
Figure 44. Action RMS error for control signals generated for three generators. RMS is Root Mean Square.

It can be seen from the plot that this value reduces at the end of each pass and starts stabilizing while reaching the last pass. It confirms that the action has learned based on the present plant conditions.

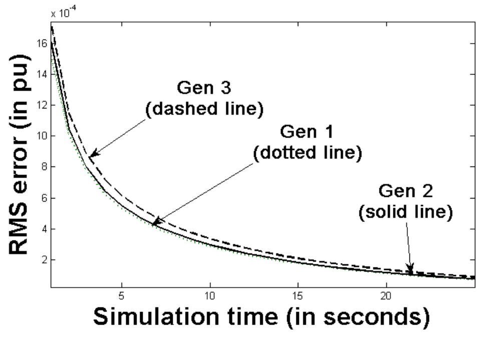
Critic Offline Training

The critic NN is also trained offline similar to the other two neural networks. The offline training is done with random values of weights and biases, and 25 passes of offline training are conducted. The weights and biases are stored at the end of each pass of offline and loaded during the next pass similar to how we performed for the action network.

The RMS error for the critic network is plotted (Figure 45), and it can be seen that the error gets minimized after each pass of offline training. The critic output is known as lambda and is separated into two different plots (lambda1 and lambda2) for better visibility. The saved weights and biases are plotted (Figure 46) to see the learning of the critic network. After 10 passes, the values are reaching a stable state that demonstrates the offline training is completed.



(a) lambda1



(b) lambda2

Figure 45. Critic RMS error for lambda1 and lambda2. RMS is Root Mean Square. Lambda is the output of critic network.

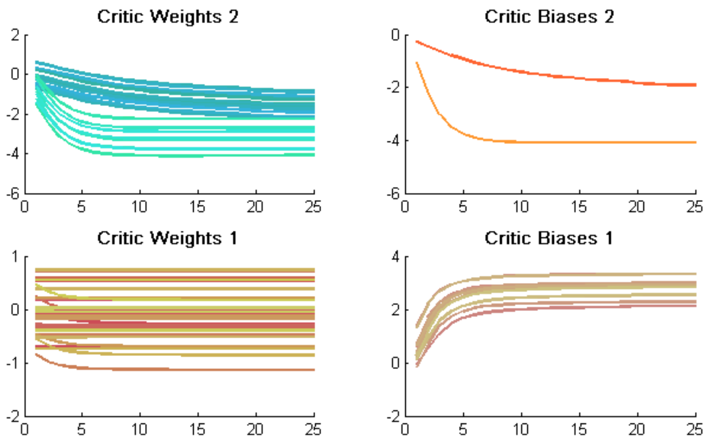


Figure 46. Critic’s weights and biases during the offline training. Weights1 and biases1 are the weights and biases used between the input layer and the hidden layer. Weights2 and biases2 are the weights and biases used between the hidden layer and the output layer.

CHAPTER V

WIDE AREA CONTROLLER DEVELOPMENT

This chapter describes the development and implementation of the designed controller. After the completion of offline training for all three NNs, an online training is conducted that helps the NNs to learn depending on present and dynamic conditions of the plant parameters. The online training is performed using the knowledge learned during the offline training.

Identifier Online Training

An identifier that is well trained with the offline training performs better during the online training. The output of the identifier matches close to the plant state for both the rotor speed (omega or ω) and the terminal voltage (Vt). The speed and voltage are plotted for a single generator (Figures 47 and 48) of the power system.

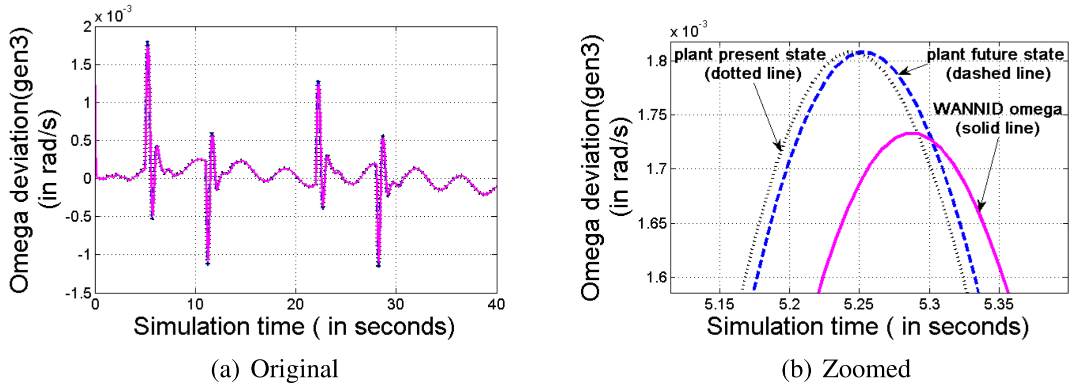
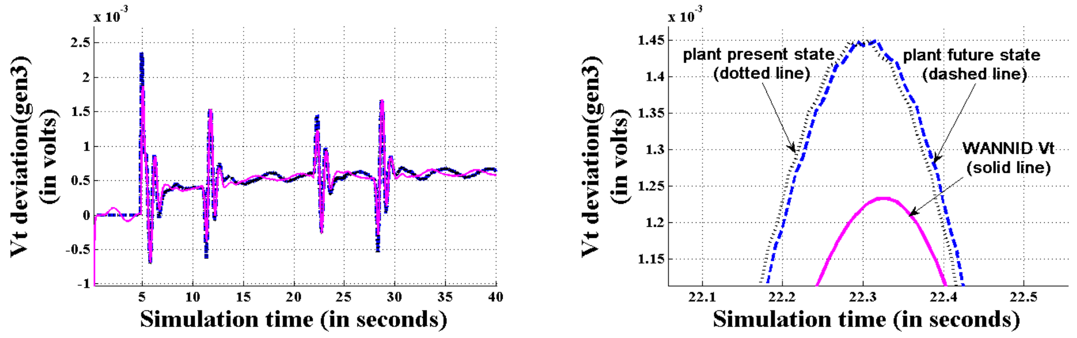


Figure 47. Online training of WANNID on Generator 3 for omega. WANNID refers to the wide-area neural network identifier. Omega is rotor speed.



(a) Original (b) Zoomed

Figure 48. Online training of WANNID on Generator 3 for Vt. WANNID refers to the wide-area neural network identifier. Vt is terminal voltage.

Two versions of the plot are presented in the figures, in which the zoomed version allows easier examination of the graphs, and the original version helps in identifying the shape of the plot. During the online training, the normalized data sent to the identifier plays an important role for the identifier to learn better.

The scaling used for normalizing the incoming data is set to [7 6 3 7 6 3 7 6 3], where the values of this array represent the factors used for scaling the rotor speed, the terminal voltage, and the control signal used to control the plant. Three sets of these values are for three generators used for the training. This data set is selected after careful examination on the outputs using some trial values. It can be seen from the plot (Figure 47) that the speed is trained well, and the identifier output is indistinguishable from the plant output. In the plot for terminal voltage (Figure 48), the identifier output tries to follow the plant output, but not as well as the output for rotor speed. The weight-adjustment routine uses the error to calculate the weight and bias adjustments (Figure 49). An error signal (Figure 50) is computed by calculating the difference between the output of the identifier and the future state of the plant. This error signal is multiplied by a gain and fed back to the NN using the backpropagation learning algorithm. The change in the weights and biases shows that the identifier is adjusting its weights and biases dynamically based on the present plant conditions.

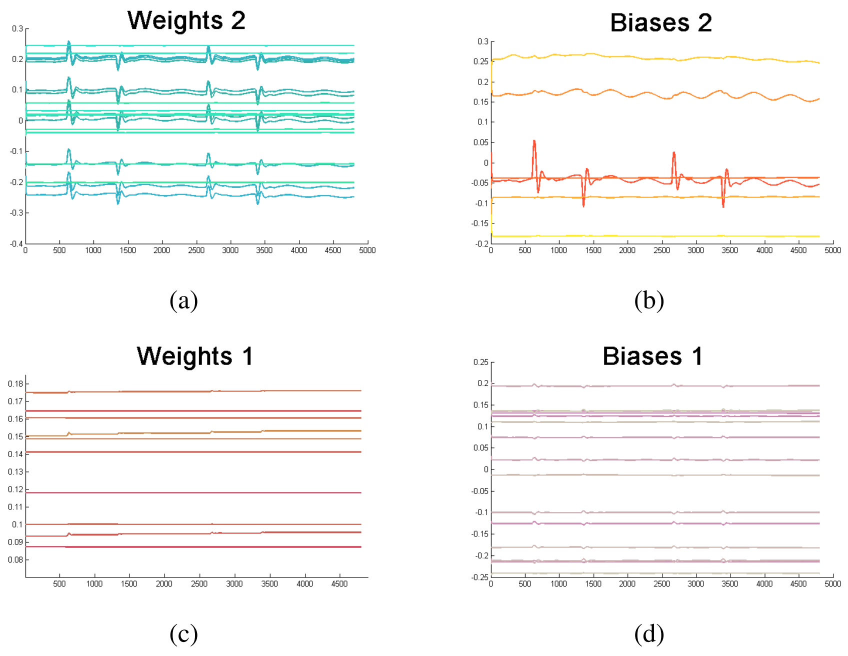
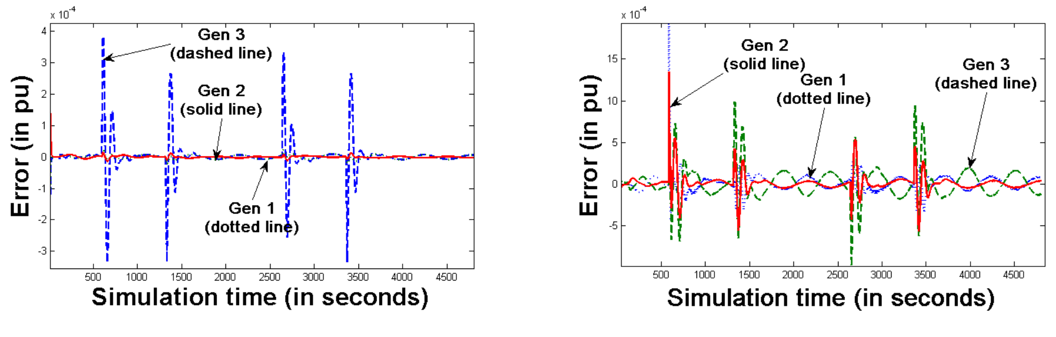


Figure 49. Weights and biases for WANNID during the online training. WANNID is wide-area neural network identifier. Weights1 and biases1 are the weights and biases used between the input layer and the hidden layer. Weights2 and biases2 are the weights and biases used between the hidden layer and the output layer.



(a) Original (b) Zoomed

Figure 50. WANNID error for omega and Vt in all three generators. WANNID is wide-area neural network identifier. Omega is rotor speed. Vt is terminal voltage.

Action Online Training

One of the concepts of the designed controller is that the performance can be improved with the use of online training. Previously, the action network was trained to mimic the PSS and trained very well. During the online training, we want the controller to look at the near-term operating conditions and make proper adjustments to improve the plant operation.

To do this training, we rely on backpropagation of an error signal. In this case, the error signal is constructed from a combination of the plant deviation from an optimum condition, the value of the PSS output, and the existing action output. As the action's weights and biases are adjusted, the error decreases at each iteration. Before starting the online training, the weights and biases are initialized to those obtained during the offline training. Adjustments to the weights and biases are done during the online training (Figure 51) similar to how we performed for the offline training.

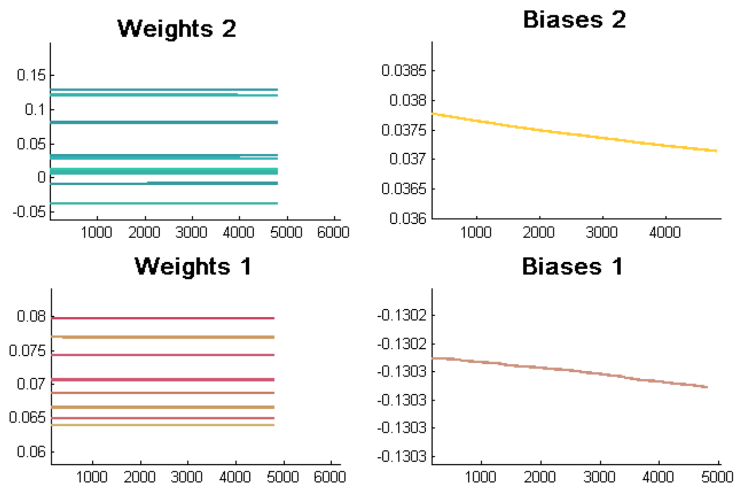
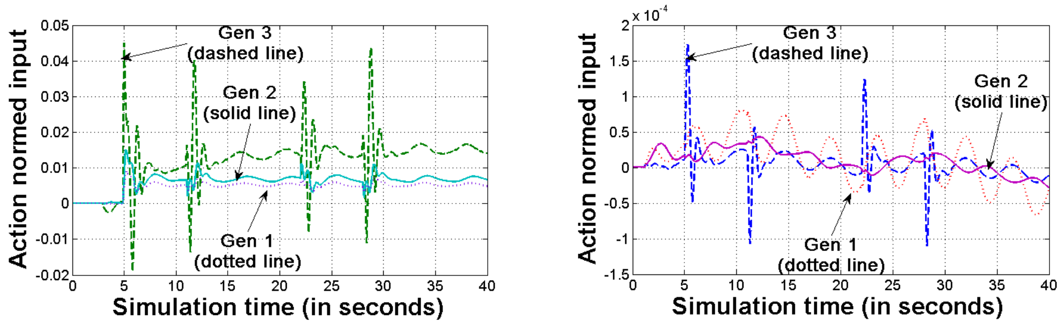


Figure 51. Weights and biases for the action network during the online training (zoomed). Weights1 and biases1 are the weights and biases between the input layer and the hidden layer. Weights2 and biases2 are the weights and biases between the hidden layer and the output layer.

The plant values sent to the action network are normalized well (Figure 52) so that the NN generates an output signal to control the plant operation and to improve the system performance. The calculated action error (Figure 53) drives the action training process.



(a) omega (b) Vt

Figure 52. Normalized input to action network with omega and Vt for all three generators. Omega is rotor speed. Vt is terminal voltage.

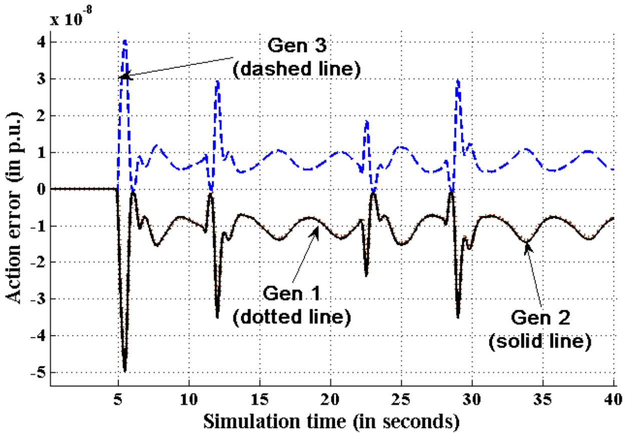


Figure 53. Action network error for the control signals captured for all three generators during the online training.

Critic Online Training

The critic online training is proceeded in a similar way that we used for the action network. The weights and biases are initialized to those obtained during the offline training.

While performing the online training, the weights and biases (Figure 54) change dynamically using the utility function for improved plant operation. Similar to other NNs, an error signal is calculated for the critic network (Figure 55) that helps the network to adapt to the dynamic plant conditions.

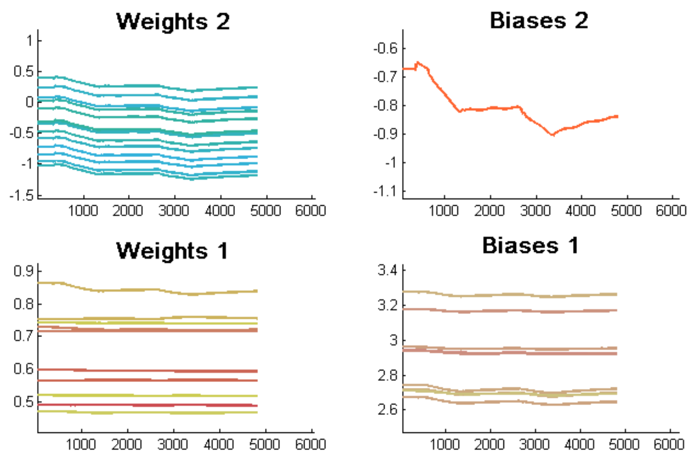
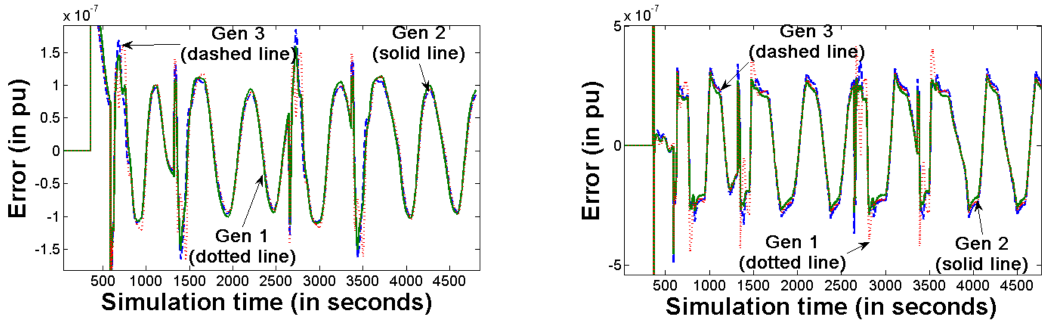


Figure 54. Weights and biases for critic network during the online training. Weights1 and biases1 are the weights and biases used between the input layer and the hidden layer. Weights2 and biases2 are the weights and biases used between the hidden layer and the output layer.



(a) omega

(b) Vt

Figure 55. Critic network error for omega and Vt for all three generators. Omega is rotor speed. Vt is terminal voltage.

Testing and Results

To test the designed WAC, the two-area and five-machine power system is considered (Figure 56). In this model, generators 2, 3, and 5 form the first area, and generators 1 and 4 form the second area. The controller receives the plant outputs from three synchronous generators $G1$, $G2$, and $G3$. These three generators receive a separate control signal from the controller that is fed into their exciter. This model is implemented in the PSCAD® software. The main page of PSCAD® implementation can be seen in Appendix E.

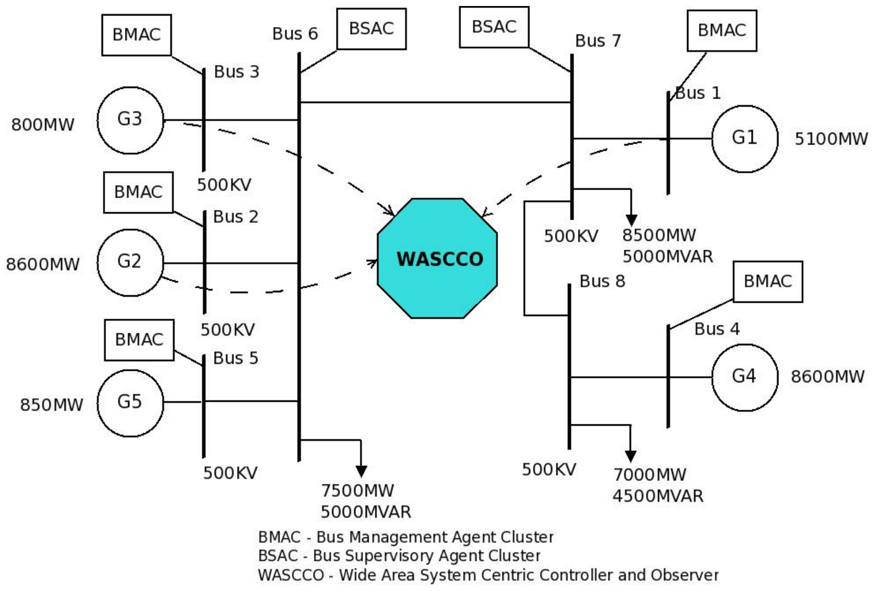


Figure 56. Wide-Area Coordinating System Centric Controller and Observer placed in a two-area and five-machine power system.

Implementation and testing. In the PSCAD implementation, a separate sub-page (Figure 57) is created that reflects the developed controller. This module receives electrical parameters from all three generators. The left-hand portion of the figure (Figure 57) shows the inputs to the WAC module (WA_ActionFE.m). These inputs are captured from three generators that are used in this experiment.

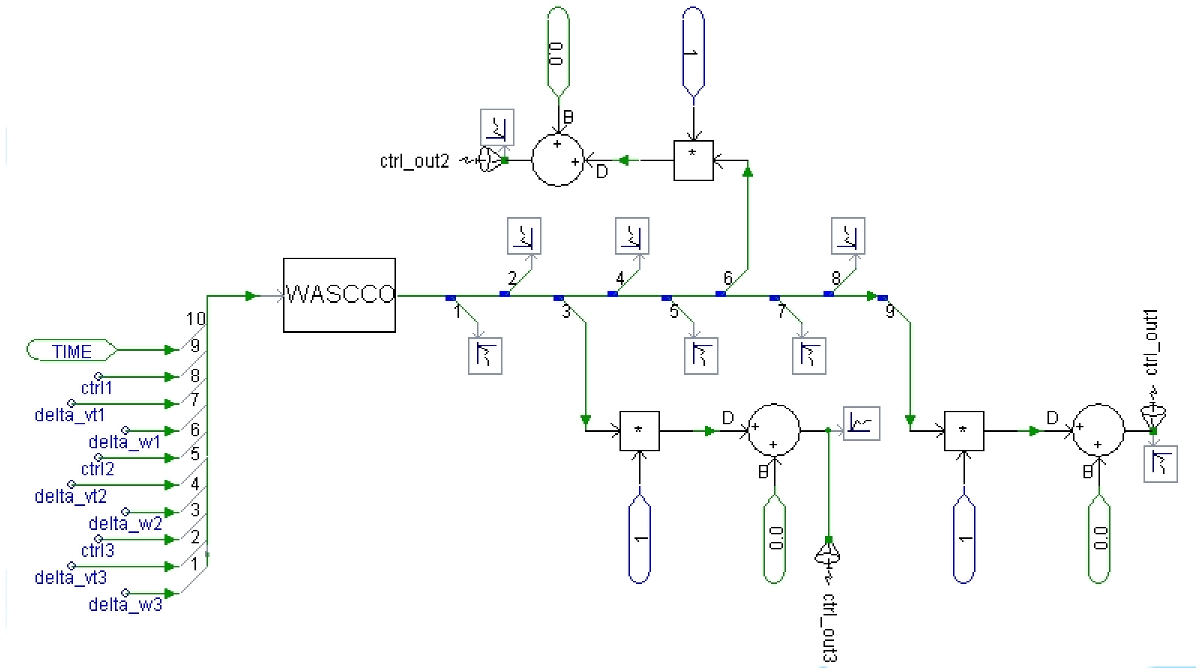


Figure 57. PSCAD implementation sub-page showing the wide-area controller. WASCCO is Wide-Area Coordinating System Centric Controller and Observer. Generator speed is *wi*. *Vt* is terminal voltage.

From each generator, the speed deviation (delta_w-i), the voltage deviation (delta_vt-i), and the present control signal (ctrl-i) used to control the plant are sent to the controller module. Here *i* in each signal refers to the generator number. The signal named *TIME* is the present simulation time. These signals are connected to a multiplexer and then sent to the module labeled *WASCCO* that represents the controller. Similarly, the outputs are demultiplexed into nine separate signals. These outputs are formed as three sets of values, in which each set has three values corresponding to a single generator. In this set, the first two values are the normalized generator speed and the normalized terminal voltage. The third value is the control signal (ctrl_out-i) used for stabilizing that generator. The outputs can be seen in the right-hand portion of the figure (Figure 57). The actual controller is implemented using MATLAB®

software. At each simulation step, the controller module is invoked from the PSCAD® model. The received controller output is multiplied by a gain to increase the magnitude, and an offset is used to set the shape.

The measure of generator stability cannot be seen with the rotor speed or its deviation. To measure the inter-area stability, we should use the difference in rotor speed variations between generators in different areas. The absolute rotor speed deviation of an individual generator is not as important as whether or not all the generators are spinning in synchronization with each other. To present the results, we measure the speed difference between the $G3$ from the first area and the $G1$ from the second area. This measurement is referred to *omega 1 minus omega 3* in the following figures.

The test case considered here is a torque disturbance on $G3$. This torque disturbance is applied at four different time periods. During this disturbance, the torque of the $G3$ increases to a value of 0.2 pu at t=5 seconds, decreases to a value of 0.2 pu at t=11 seconds, then again increases to a value of 0.2 pu at t=22 seconds, and finally decreases to a value of 0.2 pu at t=28 seconds. The speed of the generators oscillate during the application of these disturbances. Our WASCCO tries to minimize the speed deviation as much as possible. The first test is done without applying any control signal in any of the generators. This test is used as a benchmark for measuring the performance of other controllers.

To demonstrate the stability of the system, we overlay the rotor speed differences without using any controller, using PSS as the controller, and using WASCCO as the controller (Figure 58). The speed deviation for the no-control plot shows the largest deviation of rotor speed. Then the speed deviation for the PSS shows some improvement on the stability compared to the no-control plot. The WASCCO demonstrates that the stability of the system is improved greatly because of its global knowledge. During the first peek in the graph (Figure 58), the difference between omega deviation for the no-control plot and the WASCCO plot is nearly 0.2 rad/s, and the difference between omega deviation for the PSS plot and the WASCCO plot is about 0.1 rad/s. From this test, we can infer that the WASCCO works more efficiently than the local controller because it has the knowledge of the entire system.

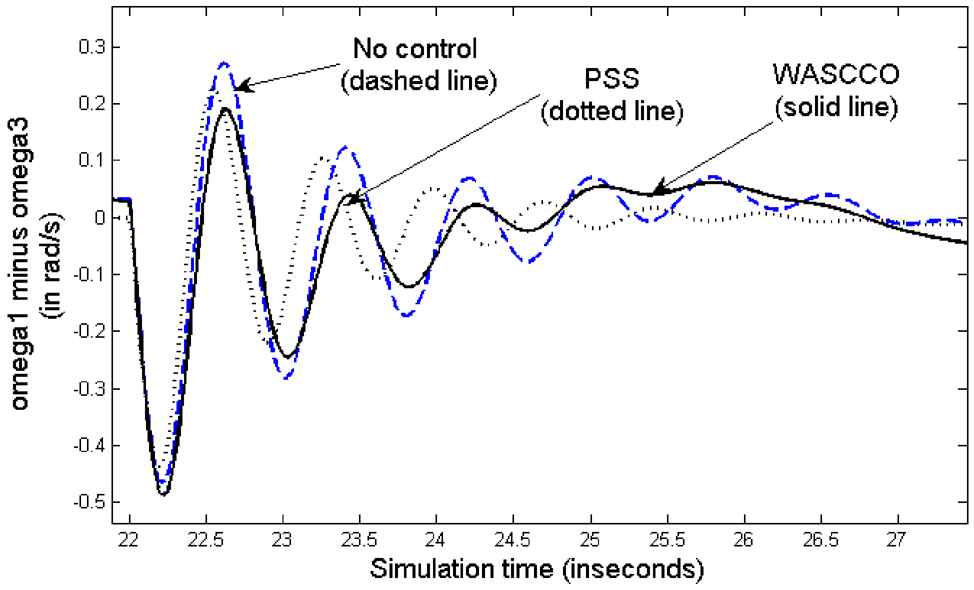


Figure 58. Performance comparison of controllers with initial configuration of WASCCO. PSS is Power System Stabilizer. WASCCO is Wide-Area Coordinating System Centric Controller and Observer. Omega is rotor speed.

Controller enhancement. After the initial test, we identified that the WASCCO can perform better than any other local controller. To improve the performance of our controller, some fine-tuning is done in the action network because this network is the core module that generates the control signal. The activities used to enhance the operation of the controller are the following:

- normalization of action input;

- change in the number of hidden neurons for the action network; and
- feeding the plant future state identified by the identifier to the action network.

The normalization of action input plays a crucial role in the learning of action network. The inputs are rotor speed and terminal voltage. Between these inputs, giving more priority to the speed results in larger oscillation of the controller output, which in turn oscillates the rotor speed. So an experiment is done with giving more priority to the terminal voltage instead of the rotor speed, and this change proved to be working well. The next experiment is done with changing the number of hidden neurons. In any NN, providing sufficient number of neurons will improve the learning. Sequence of experiments are conducted to identify the number of hidden neurons that improve the action network learning. Finally, the action input is changed from the plant output to the identifier output. The output of a well-trained identifier mimics the future state of the plant and improves the action learning.

After implementing all the enhancements described above, we conduct a final test to measure the performance of our enhanced controller. A graph is plotted (Figure 59) again for the speed deviation of generators without any controller, with the PSS, and with the enhanced configuration of WASCCO. In this comparison, we can clearly see that the speed of the generator comes to a stable state immediately after the first peek, and for other controllers, the speed stabilizes after a few oscillations. A full graph is plotted (Figure 60) to see the entire simulation with all three test cases.

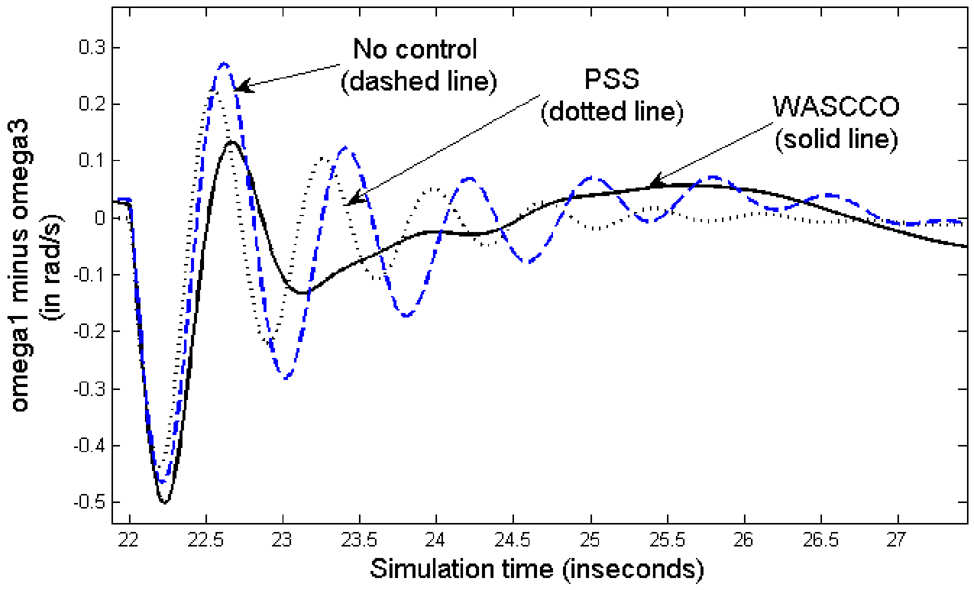


Figure 59. Performance comparison of controllers with enhanced configuration of WASCCO (zoomed). PSS is Power System Stabilizer. WASCCO is Wide-Area Coordinating System Centric Controller and Observer. Omega is rotor speed.

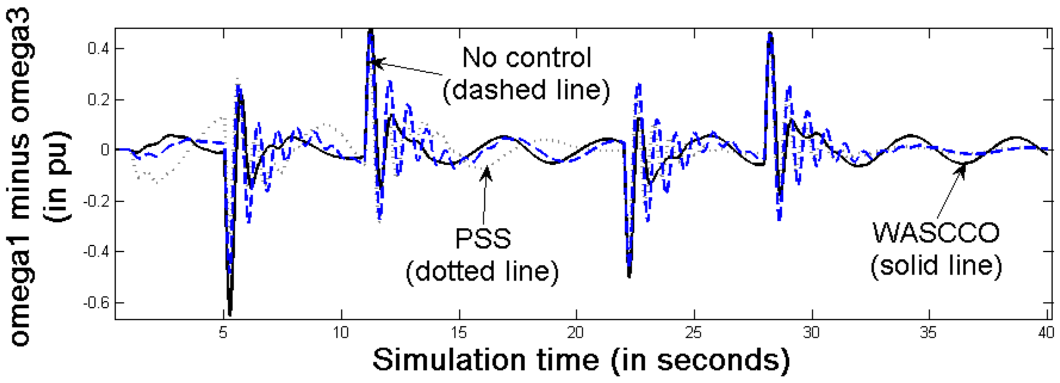


Figure 60. Performance comparison of controllers with the enhanced configuration of WASCCO. PSS is Power System Stabilizer. WASCCO is Wide-Area Coordinating System Centric Controller and Observer. Omega is rotor speed.

Testing controller with fault simulation. The performance of the controller was analyzed with the torque disturbance test earlier, and it performed the job very well. To ensure the working of the controller, we consider another test case based on fault simulation that we used to test the agent framework. The same two-area and five-machine power system is considered for this simulation. The total simulation time considered for this case is 40 seconds. Initially, there is a short circuit fault simulated in the center of top transmission lines (modeled as coupled pi sections) of the $G3$ (Figure 61) at $t = 5$ seconds, and this fault lasts for a duration of 0.1 seconds. Immediately after the fault, the breakers labeled BRK are opened to remove the top transmission lines from the system. The speed of the generator oscillates because of a change in the connected load. The WASCCO sends a control signal that stabilizes the speed of the generator. At $t = 15$ seconds, the previously opened BRK are now closed.

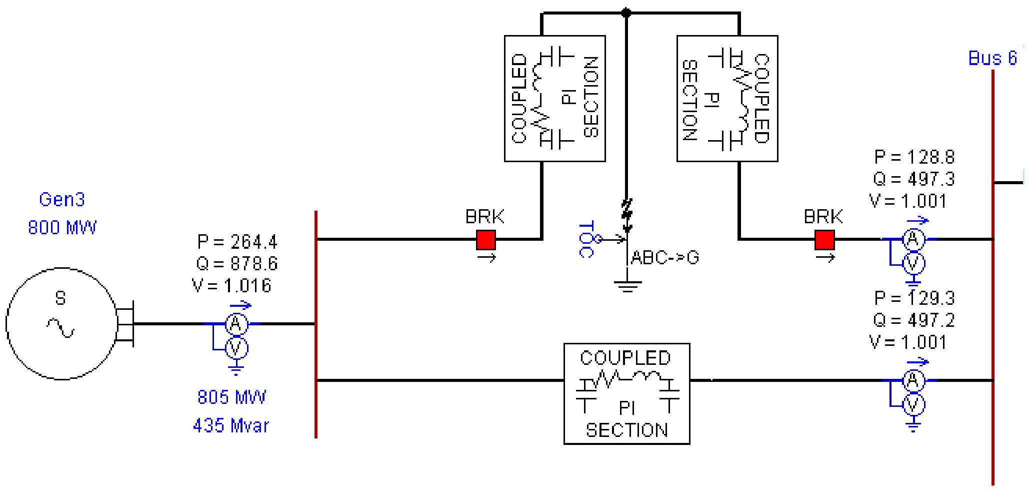


Figure 61. Fault simulation in Generator 3. BRK denotes breaker. TOC is Time of Control. ABC represents phase of current. G is Ground.

The same test is conducted under three different circumstances. First, no control signal is given to any of the generators in the system. Second, the PSS is used to control the $G1$, $G2$, and $G3$. Finally, the control signal from the WASCCO is given to those three generators. The speed deviation between the $G1$ and the $G3$ are plotted (Figure 62) to measure the performance of the developed controller. The same plot is zoomed (Figure 63) to show the portion of initial short circuit fault at $t = 5$ seconds. It can be clearly seen that the plot for the WASCCO is better than the no-control and the PSS plots.

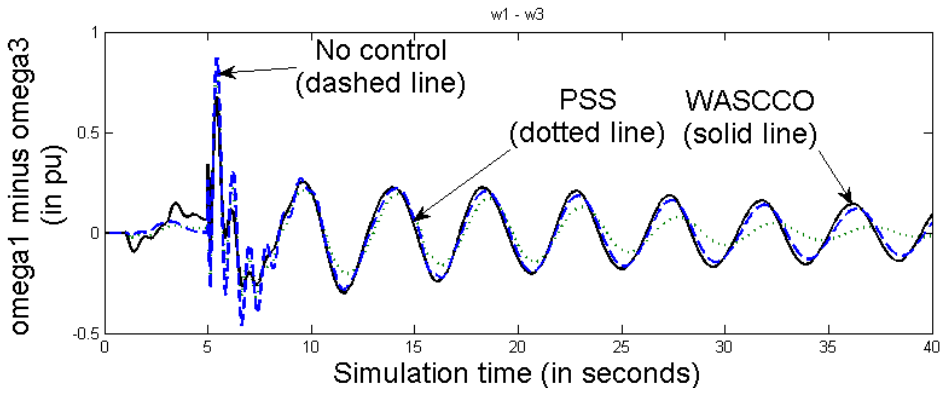


Figure 62. Performance comparison of controllers for the fault simulation test case. PSS is Power System Stabilizer. WASCCO is Wide-Area Coordinating System Centric Controller and Observer. Omega is rotor speed.

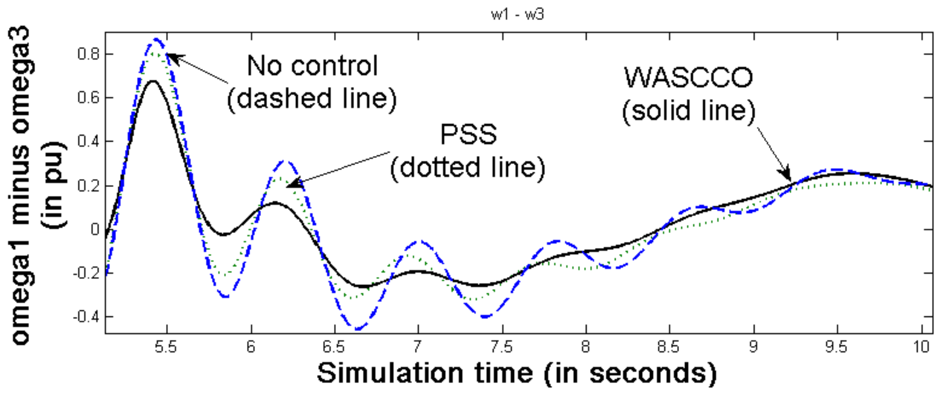


Figure 63. Performance comparison of controllers for the fault simulation test case with fault portion zoomed. PSS is Power System Stabilizer. WASCCO is Wide-Area Coordinating System Centric Controller and Observer. Omega is rotor speed.

Communication Security

The controller described in this chapter is combined with the supervising agent cluster. This agent cluster is demonstrated to be secure in the face of malicious attacks. However, the occurrence of these attacks could be minimized with a secure communication channel. This security feature is provided by the SCADA. Thus, the communication among our agent clusters could be routed through the SCADA system, a mechanism that will enhance the security of the existing architecture. The communication of power systems is time sensitive, and the security mechanism should not lead to any delay during the communication. The SCADA shows that the security mechanisms can be handled with high resolution (Hadley & Huston, 2007). Various implementations for securing the SCADA communications have been developed in the past few years. One such implementation is Secure SCADA Communications Protocol (SSCP). The Pacific Northwest National Laboratory evaluated the cryptographic implementation of SSCP and its performance. The SSCP uses the Distributed Network Protocol version 3 (DNP3) for communication and the Hashed Message Authentication Code (HMAC) for verifying the sender and securing the communication.

Hashed message authentication code. HMAC is a mathematical function approved by the National Institute of Standards and Technology and is used in SHA-1 and SHA-256 (SHA stands for Secured Hash Algorithm) algorithms. The message sender produces a HMAC value that is formed using a unique identifier, the message input, and a symmetric key (Figure 64). This encrypted message is received by the receiver. The receiver computes its own HMAC by using the received message, the unique identifier, and the symmetric key. It compares the newly calculated HMAC with the one received from the sender. If both of the keys match, the receiver ensures that the message has been received correctly. The unique identifier is comprised of two unique components: a 4-byte time field and a 2-byte sequence number. Time is used in the unit of seconds, and the 2-byte sequence number allows multiple communications to take place within a 1 second window. So this scheme provides a unique value for repeatable messages.

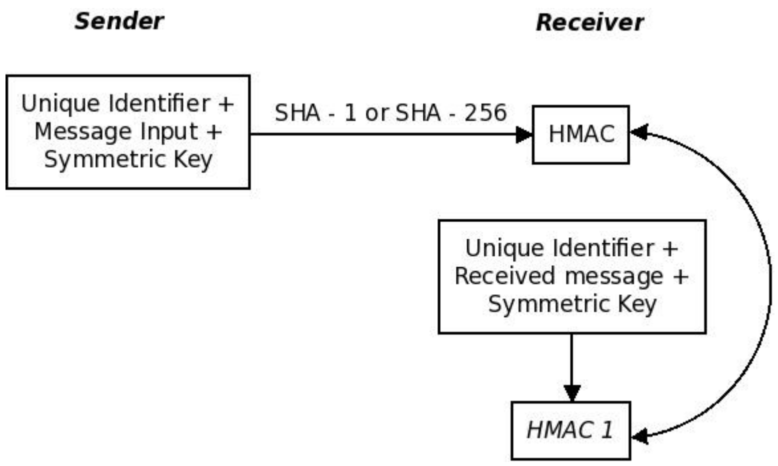

Sender
Receiver
Unique Identifier + Message Input + Symmetric Key
SHA - 1 or SHA - 256
HMAC
Unique Identifier + Received message + Symmetric Key
HMAC 1

Figure 64. HMAC sender and receiver. HMAC is Hashed Message Authentication Code. SHA is Secured Hash Algorithm.

The symmetric key that is used to compute the HMAC should be exchanged between the sender and the receiver. This exchange is done using the Diffie-Hellman key exchange algorithm. However, any key update algorithm will require more processing power than the authentication algorithms. For this purpose, the SSCP supports pre-shared keys for remote devices with limited processing capabilities. This approach builds upon the cryptographic one-time pad algorithm, where a message is encrypted with a random key that is known by both the sender and the receiver and is used only once. The pre-shared keys implemented in the SSCP are unique for a session and not for each individual message.

Distributed network protocol. The DNP3 was designed to optimize the communication of data acquisition and control commands from one computer to another (DNP Users Group, 2007). According to the DNP3, computers are classified into two groups: outstation and master. The outstation refers to computers that are in the fields, and the master refers to computers that are in the control centers. This protocol provides rules for both outstations and masters to communicate data and control commands. The DNP3 is not a general purpose protocol. It is designed to optimize the transmission of data acquisition information and control commands from one computer to another. The system architecture (Figure 65) of the DNP3 can be represented as three different models.

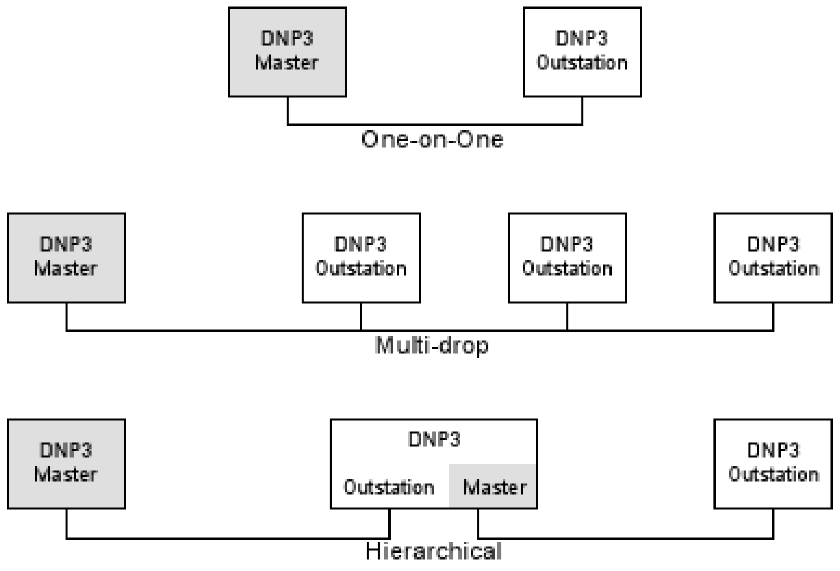


Figure 65. DNP3 system architecture with master and outstation nodes. DNP3 is Distributed Network Protocol version 3.0

The architecture presented in the top is a simple one-on-one system having a single master and a single outstation. The physical connection between these two devices is typically a dedicated dial-up telephone line. The second type of system is known as a multi-drop design, in which a single master station communicates with multiple outstation devices. Conversations are typically between the master and a single outstation at a time. So the master continually interrogates each outstation in a round-robin fashion. The communication media for this type is a multi-dropped telephone line, a fiber-optic cable, or a radio. In hierarchical forms, a station may operate as a master for gathering information or sending commands to the outstation in another station. Afterward, the station may change its role to become an outstation. Both the master and the outstation devices have a multi-layer software architecture (Figure 66).

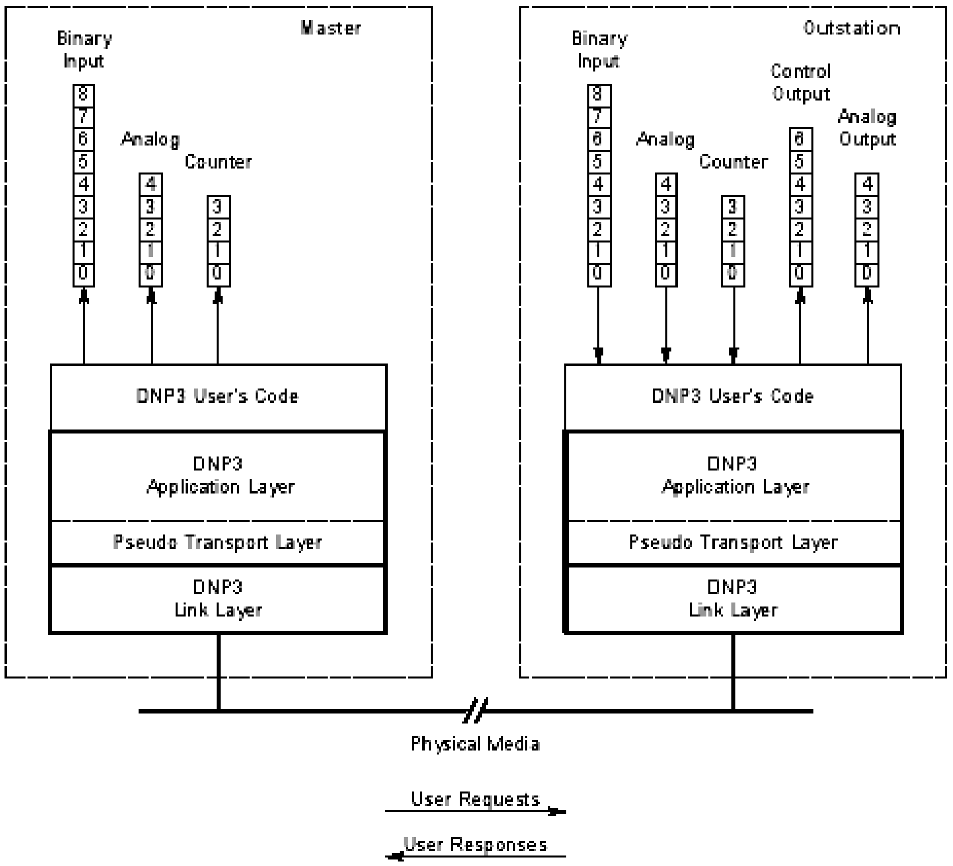


Figure 66. DNP3 protocol stack will all layers. DNP3 is Distributed Network Protocol version 3.0

The top layer is the DNP3 user layer. In the master, this layer interacts with the database and initiates requests for data from the outstation. In the outstation, the same layer is responsible for querying the database and responding to master requests. The DNP3 software layers provide reliable data transmission to effect an organized approach to the transmission of data and commands. The link layer has the responsibility of keeping the physical link reliable. This reliability is achieved by providing error detection and duplicate frame detection. The link layer sends and receives packets.

A DNP3 frame (Figure 67) consists of a header and a data section. The header specifies the frame size, data link control information, and the DNP3 source and destination device address. The data section is called a payload and contains the data passed down from the top layers. Every frame begins with two sync bytes that help the receiver to determine the beginning of the frame. The length specifies the number of octets in the remainder of the frame, and it does not include the octets used for cyclic redundancy check. The link control octet is used by the sending and receiving link layers to coordinate their activities.

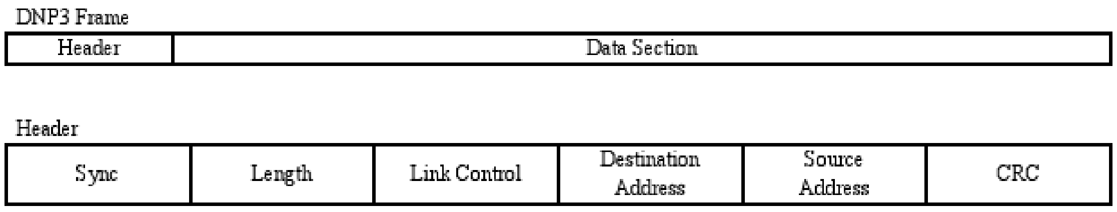


Figure 67. DNP3 frame format. DNP3 is Distributed Network Protocol version 3.0. Sync is synchronization bit. CRC is Cyclic Redundancy Check.

The features described above can prevent the possibility of malicious attacks. Our developed architecture can be piggybacked with these features and provide a highly secured method for power system control and protection.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

Conclusions

The software agents based on a multi-level security architecture are capable of preventing several common types of attacks. However, an assumption is made that eventually some attack will succeed in bypassing the prevention layers and will reach the underlying computational agent. During the occurrence of an attack, the replicated mutations will provide for the detection of such an attack, and hence, the consequences of the attack are eliminated. The clusters for monitoring and supervising the components of electric power network are formed using the above mentioned multi-level security architecture. These agent clusters are developed based on the FIPA specifications and are known as FIPA-compliant agents. These FIPA-compliant agent clusters are placed on power network nodes and communicate with each other, thereby forming another network called an agent network. This communication uses the developed ontology as a vehicle to converse about the activity in the power grid. The clusters are divided into monitors and supervisors. The supervisors are informed by the monitors regarding the activity of the electric power network. So the supervisors are capable of analyzing the entire power network and are able to solve problems identified by the monitors. As the supervisors know the details about all the components in the power system, they have a global view of the entire power network. This global knowledge gives the supervisors the capability of controlling the wide-area power system. This operation is performed by the

NN-based adaptive controller. Having the parameters of all the components in the power network, the controllers are capable of generating a control signal that improves the system stability.

The developed agent clusters are demonstrated to be secure against attacks of malicious intent and to be able to protect the power system components. The supervisors equipped with the developed wide-area controller are shown to be performing better than existing local controllers in a two-area power system model that was implemented using the PSCAD® software.

Future Work

The MRA, which is the heart of the multi-agent methodology, is now implemented in software. This agent is responsible for providing the core layer of security. Attacks against the MRA can be eliminated by implementing this module in hardware. During this experiment, all agent modules were running on a single machine, and the communication took place without any delay. Distributing these agent clusters among multiple machines will provide a more realistic study environment of the developed architecture. Applying the developed wide-area controller in a power system model including components that exhibit more dynamic behavior will help for better evaluation. The above mentioned enhancements will guide for the future scope of this research.

REFERENCES

Ali, G., Shaikh, N. A., Shah, M. A., & Shaikh, Z. A. (2010). Decentralized and fault-tolerant FIPA-compliant agent framework based on .Net. *Australian Journal of Basic and Applied Sciences*, *4*(5), 844–850. Retrieved from www.insipub.com/ajbas/2010/844-850.pdf

Baxevanos, I., & Labridis, D. (2007). Implementing multiagent systems technology for power distribution network control and protection management. *IEEE Transactions on Power Delivery*, *22*(1), 433–443. doi:10.1109/TPWRD.2006.877085

Bradshaw, J. M. (1997). *An introduction to software agents*. Cambridge, MA: MIT Press.

Chien, E., & Szor, P. (2002). Blended attacks exploits, vulnerabilities and buffer-overflow techniques in computer viruses. *Virus Bulletin Conference*, 1–36. Retrieved from www.peterszor.com/blended.pdf

Davidson, E. M., McArthur, S. D. J., McDonald, J. R., Cumming, T., & Watt, I. (2006). Applying multi-agent system technology in practice: Automated management and analysis of scada and digital fault recorder data. *IEEE Transactions on Power Systems*, *21*(2), 559–567. doi:10.1109/TPWRS.2006.873109

DNP Users Group. (2007). *A DNP3 protocol primer.* Retrieved from www.dnp.org/About/DNP3%20Primer%20Rev%20A.pdf

Edwards, D., Simmons, S., & Wilde, N. (2007). Prevention, detection and recovery from cyber-attacks using multilevel agent architecture. *IEEE International Conference on Systems of Systems Engineering*, 1–6. doi:10.1109/SYSOSE.2007.4304228

Finin, T., Labrou, Y., Mayfield, J., & Bradshaw, J. (1997). KQML as an agent communication language. In J. Bradshaw (Ed.), *Software Agents* (pp. 291–316). Cambridge, MA: The MIT Press.

Foundation for Intelligent Physical Agents (FIPA). (1996). *FIPA specifications.* Retrieved from http://www.fipa.org

Hadley, M. D., & Huston, K. A. (2007). *Secure SCADA communication protocol performance test results.* (Contract No. DE-AC05-76RL01830). Retrieved from http://www.oe.energy.gov/DocumentsandMedia/10-SSCP_Test_Results.pdf

Hingorani, N. G., & Gyugyi, L. (2000). *Understanding FACTS: Concepts and technology of flexible AC transmission systems*. New York, NY: IEEE Press.

Hossack, J. A., McArthur, S. D. J., Menal, J., & McDonald, J. R. (2003). A multi-agent architecture for protection engineering diagnostic assistance. *IEEE Transactions on Power Systems*, *18*(2), 639–647. doi:10.1109/TPWRS.2003.810910

Kamalasadan, S., & Swann, G. D. (2009). A novel power system stabilizer based on fuzzy model reference adaptive controller. *IEEE Power and Energy Society General Meeting*, 1–8. doi:10.1109/PES.2009.5275897

Kamalasadan, S., & Swann, G. D. (2010). Analysis of inter-area mode oscillations using intelligent system-centric controllers. *IEEE Power and Energy Society General Meeting*, 1–7. doi:10.1109/PES.2010.5590192

Kamalasadan, S., & Swann, G. D. (2011). A novel system-centric intelligent adaptive control architecture for damping inter-area mode oscillations in power system. *IEEE Transactions on Industry Applications*, *47*(3), 1487–1497. doi:10.1109/TIA.2011.2126037

Kamalasadan, S., Swann, G. D., & Ghandakly, A. A. (2009a). An intelligent hybrid controller for speed control and stabilization of synchronous generator. *Proceedings of the 2009 International Joint Conference on Neural Networks*, 1481–1488. doi:10.1109/IJCNN.2009.5179053

Kamalasadan, S., Swann, G. D., & Ghandakly, A. A. (2009b). An intelligent paradigm for electric generator control based on supervisory loops. *Proceedings of the 2009 International Joint Conference on Neural Networks*, 1489–1496. doi:10.1109/IJCNN.2009.5178982

Kundur, P., Balu, N., & Lauby, M. G. (1994). *Power system stability and control*. New York, NY: McGraw-Hill.

Lendaris, G. G., & Shannon, T. T. (1999). Designing (approximate) optimal controllers via DHP adaptive critics and neural networks. Manuscript in preparation. Retrieved from http://www.nwcil.pdx.edu/pubs/papers/DesigningOptimalControllers.pdf

Leon, D. C. de, Foss, J. A., Krings, A., & Oman, P. (2002). Modeling complex control systems to identify remotely accessible devices vulnerable to cyber attacks. *ACM Workshop on the Scientific Aspects of Cyber Terrorism*. Retrieved from www2.cs.uidaho.edu/~krings/publications/SACT-2002-D.pdf

Manickam, A., Kamalasadan, S., Edwards, D., & Simmons, S. (in press). Intelligent multi-agent framework for power system control and protection.

Manickam, A., Swann, G., Kamalasadan, S., Edwards, D., & Simmons, S. (2010). A novel self-evolving multi-agent methodology for power system monitoring and protection against attacks of malicious intent. *IEEE Power and Energy Society General Meeting*, 1–8. doi:10.1109/PES.2010.5589750

McArthur, S. D. J., Davidson, E. M., Catterson, V. M., Dimeas, A. L., Hatziargyriou, N. D., Ponci, F., Funabashi, T. (2007a). Multi-agent systems for power engineering applications—part I: Concepts, approaches, and technical challenges. *IEEE Transactions on Power Systems*, *22*(4), 1743–1752. doi:10.1109/TPWRS.2007.908471

McArthur, S. D. J., Davidson, E. M., Catterson, V. M., Dimeas, A. L., Hatziargyriou, N. D., Ponci, F., Funabashi, T. (2007b). Multi-agent systems for power engineering applications—part II: Technologies, standards, and tools for building multi-agent systems. *IEEE Transactions on Power Systems*, *22*(4), 1753–1759. doi:10.1109/TPWRS.2007.908472

McMullen, D., & Reichherzer, T. (2006). Identity and functionality in the common instrument middleware architecture. *Proceedings of the Second Workshop on Formal Ontologies Meet Industry*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.6907&rep=rep1&type=pdf

Narendra, K. S., & Annaswamy, A. M. (1992). *Stable adaptive systems*. Englewood Cliffs, NJ: Prentice-Hall.

Okou, F., Dessaint, L. A., & Akhrif, O. (2005). Power system stability enhancement using a wide-are signals based hierarchical controller. *IEEE Transactions on Power Systems*, *20*(3), 1465–1477. doi:10.1109/TPWRS.2005.852056

Prokhorov, D., & Wunsch, D. (1997). Adaptive critic designs. *IEEE Transactions on Neural Networks*, *8*(5), 997–1007. doi:10.1109/72.623201

Qiao, W., Venayagamoorthy, G. K., & Harley, R. G. (2007). Optimal wide-area monitoring and nonlinear adaptive coordinating neurocontrol of a power system with wind power integration and multiple FATCS devices [Special Issue]. *Proceedings of the 2007 International Joint Conference on Neural Networks*, 466–475. doi:10.1016/j.neunet.2007.12.008

Simmons, S., Edwards, D., Wilde, N., Just, J., & Satyanarayana, M. (2006). Preventing unauthorized islanding: Cyber-threat analysis. *Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering*, 184–188. doi:10.1109/SYSOSE.2006.1652294

Solanki, J. M., Khushalini, S., & Schulz, N. N. (2007). A multiagent solution to distribution system restoration. *IEEE Transactions on Power Systems*, *22*(3), 1026–2034. doi:10.1109/TPWRS.2007.901280

Swann, G. D. (2010). *Development of system-centric control approaches using neural networks for power system stabilization* (Unpublished master's thesis). University of West Florida, Pensacola, Florida.

Swann, G. D., & Kamalasadan, S. (2009). A novel radial basis function neural network based intelligent adaptive architecture for power system stabilizer. *North American Power Symposium*, 1–7. doi:10.1109/NAPS.2009.5483983

Ten, C. W., Govindarasu, M., & Liu, C. C. (2007). Cybersecurity for electric power control and automation systems. *IEEE International Conference on Systems, Man, and Cybernetics*, 29–34. doi:10.1109/ICSMC.2007.4414239

Tomita, Y., Fukui, C., Kudo, H., Koda, J., & Yabe, K. (1998). A cooperative protection system with an agent model. *IEEE Transactions on Power Delivery*, *13*(4), 1060–1066. doi:10.1109/61.714454

Wang, H. (2001). Multiagent co-ordination for the secondary voltage control in power system contingencies. *IEE Proceedings in Instrumental Electrical Engineering, Generation, Transmission and Distribution*, 61–66. doi:10.1049/ip-gtd:20010025

Werbos, P. J. (1994). Backpropagation through time: What it is and how to do it. *Proceedings of the IEEE*, *78*(10), 997–1007. doi:10.1109/5.58337

Wooldridge, M. (2009). *An introduction to multiagent systems*. Chichester, England: John Wiley & Sons.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, *10*(2), 115–152. Retrieved from http://www.csc.liv.ac.uk/~mjw/pubs/ker95/ker95-html.html

APPENDIXES

Appendix A

Glossary of Variables

$\hat{V}_t$	wide-area identifier terminal voltage, part of $\hat{X}$
$\hat{X}$	wide-area identifier output
$\hat{\omega}$	wide-area identifier rotor speed, part of $\hat{X}$
$A(t)$	action output
e_a, e_A	action error
e_{ad}	adjustment error
e_c	critic error
e_{id}	identifier error
e_{ref}	difference in error between plant and model
f_A	action function
f_C	critic function
f_k	combining function for wide-area controller
f_M	wide-area identifier function
J	cost function
r	reference signal
u	‘k’ function output (combined adaptive and DHP controller)
U	utility function value
u_a	adaptive controller output
u_{nn}	neural network control signal
u_p	proportional controller output
V_f	excitation generated voltage
V_{inf}	infinite bus voltage
V_{ref}	reference voltage
V_T	generator model terminal value
W_A	action weights and biases
W_C	critic weights and biases
W_M	wide-area identifier weights and biases
X	a vector of plant output
X_L	transmission line reactance
X_m	a vector of model output
X_p	a vector of plant output
x_{ref}	plant reference

x_{rm}	reference model output
X_T	transformer reactance
α	learning rate
γ	discount factor value
λ	critic output
Σ	linear combining function
ω	generator model rotor speed
ω_0, ω_B	generator model rotor reference (or base) speed

Appendix B

Power System Model

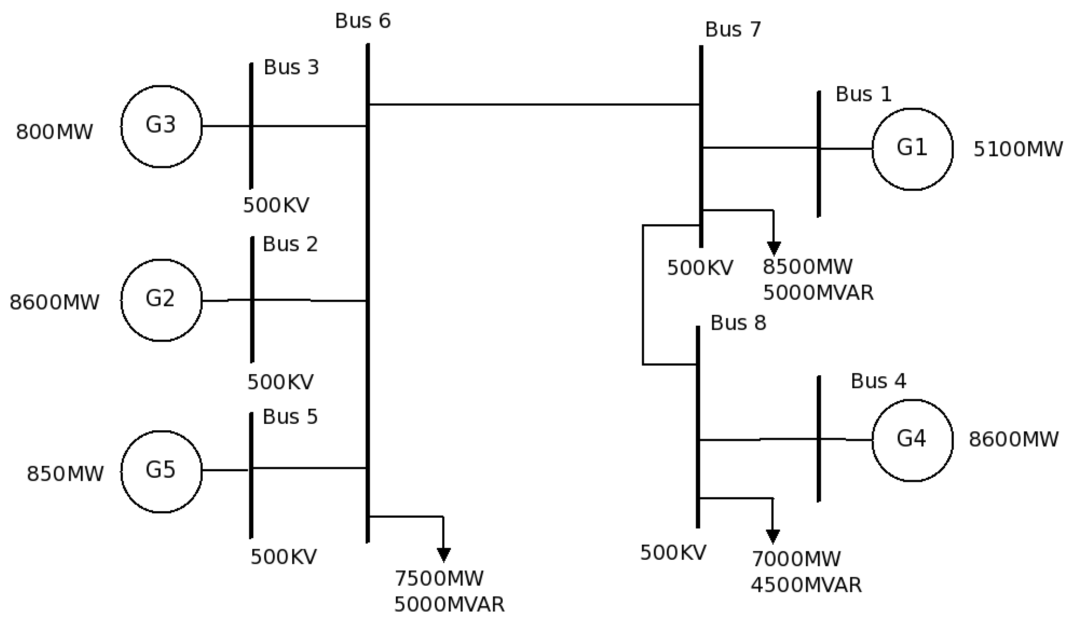


Figure B1. One-line diagram of the two-area model. Gi is the generator number. Bus-i is the bus number.

In this section, the power system model earlier referred to as “plant” is described. This power system model is a multi-machine model with five-generator, eight-bus, and two-area power system represented in PSCAD®. The parameters used for building this model are presented in the next Appendix.

For a multi-machine power system, equations can be summarized as follows:

$$\delta(t) = \omega(t) - \omega_0. \tag{B.1}$$

$$\omega(t) = \frac{1}{t_j}(t_m + g - t_\varepsilon). \tag{B.2}$$

$$0 = g(\delta, e_q^{'}, e_q^{''}, e_d^{'}, \theta, V). \tag{B.3}$$

$$0 = h(\delta, e_q^{'}, e_q^{''}, e_d^{'}, \theta, V). \tag{B.4}$$

$$0 = g_i = \sum_j V_i V_j B_{ij} \sin(\theta_i - \theta_j) + P_i^d. \tag{B.5}$$

$$0 = h_i = V_i^{-1}\left[-B_{ii}V_i^2 - \sum_{j\neq i}^{n+m+1} V_iV_jB_{ij}\cos(\theta_i-\theta_j) + Q_i^d(V_i)\right]. \tag{B.6}$$

In the multi-machine power system, each generator will be represented as in Equations B.1 - B.4. A sixth-order model for each generator is developed using a graphical user package known as PSCAD®. Also, a hydro-governor model, an excitation system model, and an AVR model is developed using the same graphical package. The transmission lines and the bus connections are designed and developed as algebraic equations as represented in Equations B.5 and B.6. A one-line diagram is used to illustrate the multi-machine system (Figure B1). Generators 2, 3, and 5 form the first area, and generators 1 and 4 form the second area.

Appendix C

Multi-Machine System Parameters

Table C1

Generator Parameters

Parameter	G1	G2	G3	G4	G5
x_d	0.1026	0.1026	1.0260	0.1026	1.0260
x_q	0.0658	0.0658	0.6580	0.0658	0.6580
$x_d^{'}$	0.0339	0.0339	0.3390	0.0339	0.3390
$x_d^{''}$	0.0269	0.0269	0.2690	0.0269	0.2690
$x_q^{''}$	0.0335	0.0335	0.3350	0.0335	0.3350
$T_{do}^{'}$	5.6700	5.6700	5.6700	5.6700	5.6700
$T_{do}^{''}$	0.6140	0.6140	0.6140	0.6140	0.6140
$T_{qo}^{''}$	0.7320	0.7320	0.7320	0.7320	0.7320

Table C2

Governor Parameters

Parameter	G1	G2	G3	G4	G5
T_s	0.25000	0.25000	0.25000	0.25000	0.25000
a	-0.00015	-0.00015	-0.00133	-0.00015	-0.00133
b	-0.01500	-0.01500	-0.17000	-0.01500	-0.17000

Note. Gi in column headings refers to the generator number.

Table C3

Load Admittances in pu

Admittances	Values
L_1	7.5 - j5.0
L_2	8.5 - j5.0
L_3	7.0 - j4.5

Table C4

Exciter and AVR Parameters

Parameter	G1	G2	G3	G4	G5
k_A	190.00	190.00	190.00	190.00	190.00
k_C	0.08	0.08	0.08	0.08	0.08
T_B	10.00	10.00	10.00	10.00	10.00
T_C	1.00	1.00	1.00	1.00	1.00
T_R	0.04	0.04	0.04	0.04	0.04

Note. Gi in column headings refers to the generator number.

Table C5

Transmission Line Parameters

Line	R_t	X_t	$B_t/2$
1-7	0.00435	0.01067	0.01536
2-6	0.00213	0.00468	0.00404
3-6	0.01002	0.03122	0.03204
4-8	0.00524	0.01184	0.01756
5-6	0.00711	0.02331	0.02732
6-7	0.04032	0.12785	0.15858
7-8	0.01724	0.04153	0.06014

Note. R_t is resistance, X_t is reactance, and B_t is susceptance.

Appendix D

UML Diagram of Agent Network Implementation

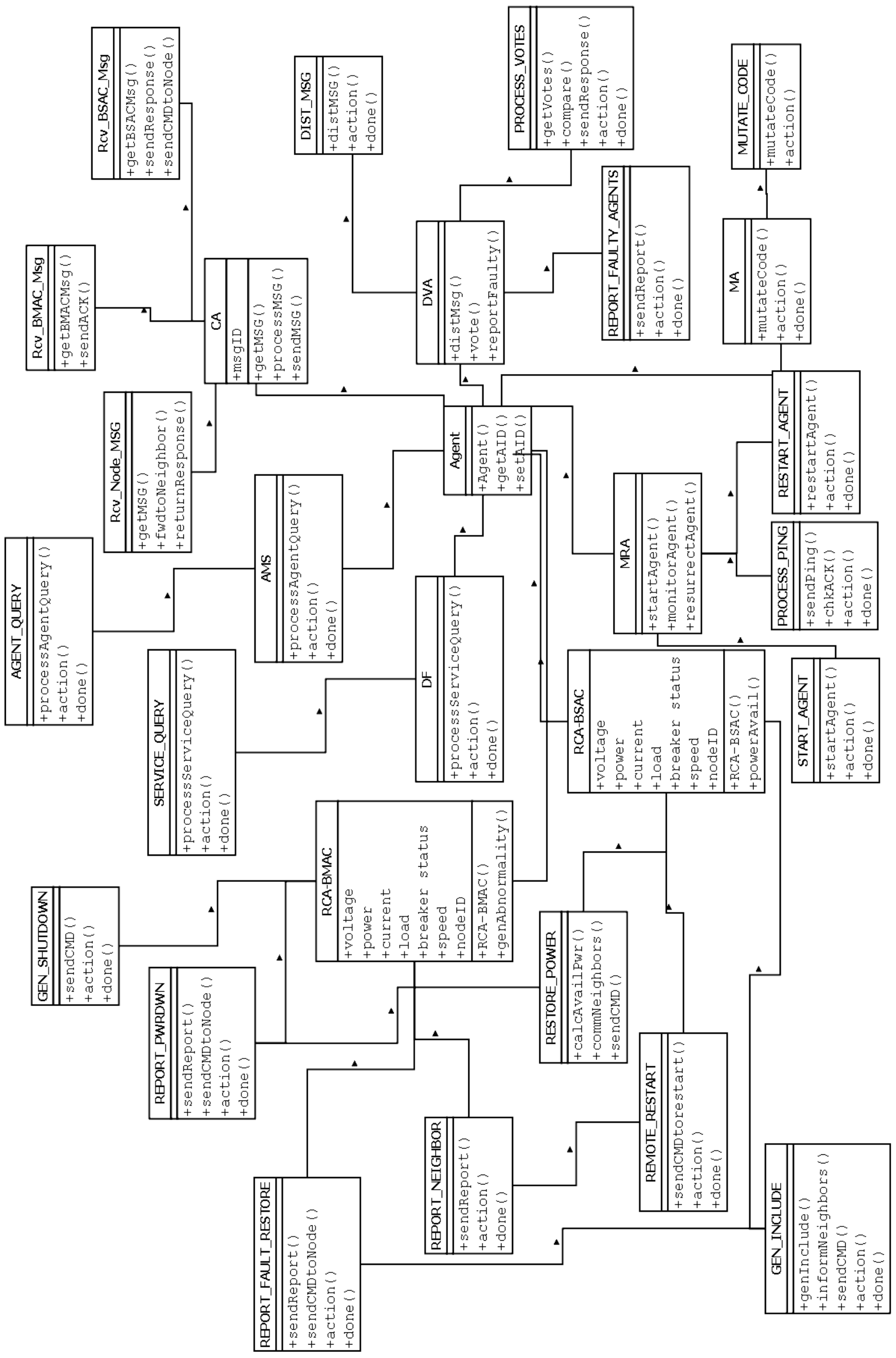
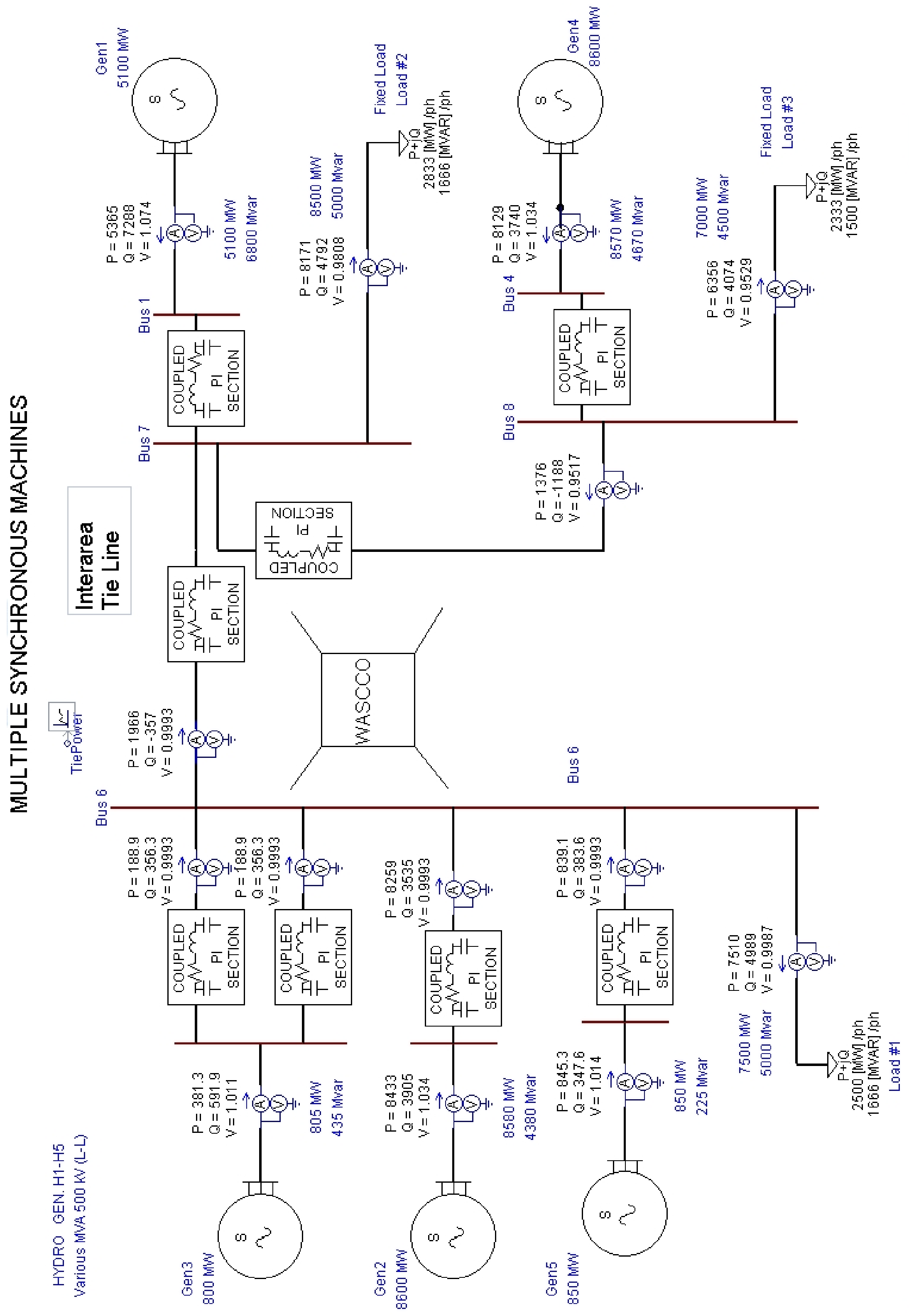


Figure D1. UML Representation of the entire model. UML is Unified Markup Language. CA is Communication Agent, DVA is Distribution/Voting Agent, MRA is Monitoring/Resurrection Agent, RCAs are Replicated Computational Agents, MA is Mutation Agent, BMAC is Bus Management Agent Cluster, BSAC is Bus Supervisory Agent Cluster, AMS is Agent Management Service, and DF is Directory Facilitator.

Appendix E

PSCAD® Main Page



MULTIPLE SYNCHRONOUS MACHINES
Interarea Tie Line
TiePower
WASCCO
Gen1 5100 MW
Gen2 8600 MW
Gen3 800 MW
Gen4 8600 MW
Gen5 850 MW
HYDRO GEN. H1-H5
Various MVA 500 kV (L-L)
Bus 1
Bus 4
Bus 6
Bus 7
Bus 8
COUPLED PI SECTION
P = 5365 Q = 7288 V = 1.074
5100 MW 6800 Mvar
P = 8171 Q = 4792 V = 0.9808
8500 MW 5000 Mvar
Fixed Load Load #2
P+jQ 2833 [MW] /ph 1666 [MVAR] /ph
P = 8129 Q = 3740 V = 1.034
8570 MW 4670 Mvar
P = 6356 Q = 4074 V = 0.9529
7000 MW 4500 Mvar
Fixed Load Load #3
P+jQ 2333 [MW] /ph 1500 [MVAR] /ph
P = 1376 Q = -1188 V = 0.9517
P = 1966 Q = -357 V = 0.9993
P = 188.9 Q = 356.3 V = 0.9993
P = 188.9 Q = 356.3 V = 0.9993
P = 8259 Q = 3535 V = 0.9993
P = 839.1 Q = 383.6 V = 0.9993
P = 7510 Q = 4989 V = 0.9987
7500 MW 5000 Mvar
P+jQ 2500 [MW] /ph 1666 [MVAR] /ph
Load #1
P = 381.3 Q = 591.9 V = 1.011
805 MW 435 Mvar
P = 8433 Q = 3905 V = 1.034
8580 MW 4380 Mvar
P = 845.3 Q = 347.6 V = 1.014
850 MW 225 Mvar

Figure E1. PSCAD® main page implementation. WASCCO is Wide Area Coordinating System Centric Controller and Observer. *P* is real power. *Q* is reactive power. *V* is voltage. *A* is current. Gen-i refers to the generator.