2005

Multiagent autonomous energy management

Shilpa B. Ganesh
West Virginia University

Multiagent Autonomous Energy Management

by

Shilpa B Ganesh

Thesis submitted to the
College of Engineering and Mineral Resources

at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in

Software Engineering

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2005

Keywords: Agents, Multiagent Systems, Maximum Flow Algorithms, Decision Networks,
Electric Shipboard Power Systems, Energy Management

Abstract

Multiagent Autonomous Energy Management

by

Shilpa B Ganesh

Master of Science in Software Engineering

West Virginia University

Ali Feliachi, Ph.D., Chair

The objective of this thesis is to design distributed software agents for reliable operation of integrated electric power systems of modern electric warships. The automatic reconfiguration of electric shipboard power systems is an important step toward improved fight-through and self-healing capabilities of naval warships. The improvements are conceptualized by redesigning the electric power system and its controls. This research focuses on a new scheme for an energy management system in the form of distributed control/software agents. Multiagent systems provide an ideal level of abstraction for modeling complex applications where distributed and heterogeneous entities need to cooperate to achieve a common goal. The agents' task is to ensure supply of the various load demands while taking into consideration system constraints and load and supply path priorities. A self-stabilizing maximum flow algorithm is investigated to allow implementation of the agents' strategies and find a global solution by only considering local information and a minimum amount of communication. This decentralized reconfiguration scheme is complemented with a layer of hierarchically organized agents to provide situational awareness capabilities. The two layers combined allow for distributed decision making while operating the system with global and optimal goals. While the reconfiguration layer performs negotiation and energy management, the situational awareness layer adds autonomous and intelligent agents that help determine component failures, tracking performance, and analyzing system events that have the potential to degrade system performance and reliability. A list of possible events includes short circuits, open circuits, loss of communication networks, and failures of agents themselves. The agents in the situational awareness layer are added using influence diagrams to assist human operators in determining the "silent death" of components or agents and to further improve the autonomous operation capabilities of the shipboard power system. Tests of different critical scenarios are investigated to demonstrate the feasibility and flexibility of the maximum flow algorithm and to evaluate the agents' decision making performance. The results are very promising.

Acknowledgement

The following is a simple representation in words of the immense gratitude I hold to all those wonderful people who have helped me make this day a reality.

First of, I duly thank my advisor Professor Ali Feliachi for giving me the opportunity to work for him. The completion of this thesis is definitely the result of his constant advice and encouragement. I also like to extend my sincere gratitude to Dr. Karl Schoder who has been instrumental in giving the right directions to my research and without his guidance and supervision, this thesis would have been an impossible feat. His supervision affected my research tremendously.

I am also thankful to Dr. Hong-Jian Lai who helped me get my hands together in the basic concepts of Graph theory, which formed the foundation of my work. I would like to thank my other committee members Ms. Cynthia D. Tanner and Dr. Katerina Goseva-Popstojanova for their review and recommendations on my thesis. I am thankful to the whole APERC team for their valuable feedback to my presentations during the Wednesday meetings. I am also grateful to Dr. Srinivas Kankanahalli who intrusted in my research potential during my early days in WVU.

My family, though not directly present here, have been my strength and motivation perpetually. I am indebted to my father, mother and sister for their continuous encouragement and support without which I would not have been able to successfully finish this endeavor of mine. The person whom I cannot thank, but just acknowledge, is my friend and fiancé, Shrawan, who has been my motivation to come all the way to the United States and who has been by my side all through my Masters.

I would also like to thank my roommates in Morgantown and my friends in Bangalore for all their support. Last, but in all ways the highest, I would like to thank "The Transcendental Almighty" for blessing me with all these wonderful people in my life.

Funding for this work was provided by the US DEPSCoR and Office of Naval Research (ONR) under grant (DOD/ONR N000 14-031-0660).

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
ARCHON	ARchitecture for Cooperative Heterogeneous ON-line systems
BDI	Belief-Desire-Intention
CCC	Command & Control Center
DAI	Distributed Artificial Intelligence
EMS	Energy Management Systems
FIPA	Foundation of Intelligent Physical Agents
IT	Information Technology
JDBC	Java Database Connectivity
ODBC	Open Database Connectivity
ONR	Office of Naval Research
KQML	Knowledge Query and Manipulation Language
KS	Knowledge Source
MAS	Multiagent Systems
RMI	Remote Method Invocation
RSAD	Reduced Scale Advanced Demonstrator
SPID	Strategic Power Infrastructure Defense
SPS	Shipboard Power System
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
UML	Unified Modeling Language

Chapter 1

Introduction

1.1 Motivation

Since its dawn in the mid to late 1970s distributed artificial intelligence (DAI) evolved and diversified rapidly, and today, is an established and promising research and application field bringing together and drawing on results, concepts, and ideas from many disciplines, including artificial intelligence (AI), computer science, sociology, economics, organization and management science, and philosophy [98]. Reflecting the focus of interest of the researchers in this field the field has adopted the name of Multiagent systems [84].

Two primary reasons can be identified as the driving forces behind the growth of Multiagent systems in recent years. First, multiagent systems have the capability of playing a key role in computer science and its applications, as modern computing platforms and information environments are very often open, large, distributed, and complex. To cope with applications that require processing of huge amounts of spatially distributed data, computers need to operate more as "individuals" or agents rather than mere "parts" [98]. Secondly, multiagent systems have the capacity to play important roles in analyzing and developing models and theories of interactivity in human societies [98]. Humans interact in various ways and at different levels. For instance, humans observe and model one another, they request and provide information, they negotiate and discuss, they develop shared views of their environment, they detect and resolve conflicts, and they form and dissolve organizational structures such as teams, committees, and economies. Many interactive processes among humans are still not very well understood, although they are an immanent part of our everyday life. DAI technologies allows us to investigate their sociological and psychological foundations [98].

Autonomous agents and multiagent systems embody a novel paradigm for analyzing, designing, and implementing solutions to complex problems. The agent-based view has the potential to improve the way in which people conceptualize and implement different types of software systems by offering a powerful repository of tools and techniques. Agents are used in a wide range of applications, ranging from small systems such as personalized email filters to large, complex, mission critical systems such as air-traffic control [49].

The distributed control of electric power systems on modern warships is one such mission critical application where the application of software agents is applied. The success of increasingly all electric commercial ships has sparked interest to incorporate the advantages of improved electric power systems in naval warships [21]. The design of the shipboard power system is dictated by the demand to reduce manning and costs and is step-by-step replacing mechanical-hydraulic systems by electric solutions. This process will ultimately also improve the survivability [38] of the envisioned all electric warship.

The use of software agents gained popularity by recent advances in software engineering to construct distributed systems that cooperate to reach a common goal [98]. This new philosophy of implementing decentralized control algorithms is discussed in this thesis and envisioned improvements and advantages of a multiagent-based control framework are applied to build an energy management system with situational awareness capabilities incorporated within it.

1.2 Problem Statement

The necessity of improving warships is the bottom line of several incidents in which the electric power was lost through only a single incident [94]. The challenge is to provide a new and distributed answer for a decentralized energy management system for the electric shipboard power system that allows automatic reconfiguration based on situational awareness. The energy management system to be developed here should help solve the problem of providing the various loads with electric power while incorporating mission and load priorities.

1.3 Objective

The objective of this research is to develop an agent architecture for the shipboard power system that incorporates the following general attributes:

- Defines a reusable and scalable architecture for the construction of intelligent software components and agents;
- Allows developing and deploying hybrid intelligent agent applications; and
- Is an event-driven, interactive development, and test environment including a means of modeling physical systems.

More specifically, this research is to design distributed intelligent control agents for reliable operation of integrated electric power systems of modern warships. In the event of scheduled load changes or unforeseen disturbances, the power system is expected to operate at a minimum level of performance in areas that could be mission critical and thus result in saving lives. The idea is to integrate three layers that will allow to achieve the desired goals: (i) the electrical network, (ii) a computer, control, and communication network, and (iii) the human operator(s). This multidisciplinary research and development will guarantee the best achievable overall system performance and reduce costs and time required for reengineering. These goals can be summarized as follows:

- Incorporate a mathematical model of the shipboard power system that is linked to and exchanges information with the controlling agent architecture;
- Agent architecture for automatic reconfiguration of power distribution on shipboard power systems;
- Human-machine interface to present and exchange information between the autonomous system and the human operator(s).

1.4 Approach

The approach taken to solve the power flow problem for the energy management system is based on spatially distributed software agents. These agents make local decisions to reach a globally acceptable solution with limited amount of communication. The distributed agent concept needs to build on an appropriate framework for implementing automatic reconfiguration based on situational conditions. Graph theory provides a formal basis to represent the distributed control system and to develop algorithms for a decentralized energy management solution. Maximum flow algorithms have been investigated to find an answer to this challenge. The energy management system is complemented by decision networks to aid in determining the current state of the system. The approach is hence summarized as follows:

- Maximum flow algorithms from graph theory for reconfiguration
- Decision networks for incorporating situational awareness capabilities

These approaches are going to be implemented using the MATLAB/Simulink [62] environment for modeling and simulating the shipboard power system in combination with JAVA technology [45].

1.5 Outline of the Thesis

An outline of the remaining chapters of this thesis is as follows:

- **Chapter 2**: Literature Survey

  This chapter gives an overview of the research background (Section 2.1) and the related work concerning electric shipboard power systems and multiagents (Section 2.2), graph theory (Section 2.3), and decision networks (Section 2.4).
- **Chapter 3**: Software Agents and Agent-Based Systems

  This chapter discusses the concepts and terminologies associated with software agents (Section 3.1) and multiagent systems (Section 3.2). In (Section 3.3) agent-based software engineering is discussed.
- **Chapter 4**: Modeling the Electric Shipboard Power System

  The challenges of designing the energy management system are discussed in this chapter. It describes the model of the shipboard power system (Section 4.1) and the interfacing of the physical model (Section 4.2).
- **Chapter 5**: Energy Management System

  Details regarding the energy management system is given in this chapter. The chapter introduces some fundamental definitions used in graph theory (Section 5.1) including maximum flow and self-stabilizing maximum flow algorithms. The shipboard's power flow as implemented in form of a maximum flow problem is discussed in (Section 5.2) followed by a discussion (Section 5.3) and conclusion (Section 5.4).

- **Chapter 6**: Situational Awareness Component

  This chapter addresses issues of identifying problems with the physical system layer and reconfiguration layer. It begins with a brief overview of decision problems and decision networks (Section 6.1). The shipboard's situational awareness component is discussed in (Section 6.2), along with a few concluding remarks (Section 6.3).

- **Chapter 7**: Multiagent Architecture

  The technical approach (Section 7.1), the architecture of the multiagent system (Section 7.2) and the agent modeling approach adopted (Section 7.3) is explained along with the implementation setup in (Section 7.4) of this chapter.

- **Chapter 8**: Case Studies and Discussion of Results

  This chapter presents case studies of the autonomous energy management system and discusses important performance and reliability aspects.

- **Chapter 9**: Summary and Future Work

  The final chapter summarizes the accomplishments in developing the autonomous energy management system (Section 9.1) and gives suggestions for future work (Section 9.2).

A list of publications and details of an award as a result of this research is given in Appendix A.1 and A.2 respectively.

Chapter 2

Literature Survey

This chapter gives an overview of the related work associated with modeling, operation and control, and reconfiguration of electric shipboard power systems. It also gives a brief glimpse into the related work in the field of multiagent systems, graph theory, and decision networks.

2.1 Modeling and Simulation

Today's commercial ships and warships are changing toward a completely electric system. To this end, Naval Combat Survivability Generation & Propulsion and DC Distribution Testbeds have been [74] developed as a common base for modeling the electric warship. With the aim towards improving the survivability of warships, the remaining control challenge requires highly efficient simulation environments for the hybrid shipboard systems to allow the development of advanced control strategies.

Modeling electrical power generation and propulsion systems is challenging but has the potential to significantly reduce the cost and risk associated with the design and acquisition process. The British Ministery of Defense and The Mathworks Ldt. [22] have developed models of marine power systems and a supporting library of generic components.

Previously, the shipboard AC power systems were based on AC radial distribution topology

[106]. Maintaining the security or integrity of such systems is directly influenced by the power system dynamics. The following gives a brief summary of work related to these systems.

In [102] an approach to conduct computer simulation studies of fault scenarios, which may occur in the electric power system on the ship as a result of battle damage, is presented. A test system for the U.S. navy shipboard power system was modeled and simulated using the Alternative Transients Program (ATP) [4] and a geographical information system. The test system included models for generators, cables, transformers, induction motors, and constant impedance loads combined with their location onboard. The geographical information system was used to both select and display affected (damaged) areas and components. This information was then conveyed in an ATP test system to simulate the fault scenario and observe the system's behavior.

Navy shipboard power systems have different characteristics when compared with utility power systems [12]. For example, there is very little rotational inertia in shipboard power systems relative to their load and prime movers on the shipboard are faster then that of utility systems. The authors in [105] and [106] present results of analyzing shipboard power systems using three popular software tools in transient simulation studies: ATP [4], PSpice [76], and Saber [93, 80]. The conclusions suggest that ATP is a good tool for generating detailed component models and calculating model parameters directly from standard specifications and ratings of the equipment. PSpice and Saber have an excellent graphical interface for building complex systems and flexibility in establishing monitoring points and processing output data.

The power flow problem can be solved numerically at a central place using various techniques, for example Newton-Raphson algorithms to solve the non-linear set of equations and it can also be formulated as linear programming problem to allow incorporation of limits and priorities of loads and supply paths. These approaches have been investigated by several authors and can be found in [13] and [14].

An expert system approach has been presented in [86] to extend the analysis of fault conditions and help in network reconfiguration for restoring the power system after occurrence of disturbances. This approach helps greatly in operating the shipboard power system and in making the correct decisions in emergency situations. Nevertheless, the disadvantages of being a centralized approach and the required time for the expert system to infer suggested switching actions have yet to be overcome.

In [89] a multiagent system approach is adopted to reconfigure the shipboard power system without human intervention and without centralized scheduling and planning. The solution is to use an agent to represent and exert high-level control on each critical system component. By summarizing the power budget and cost, the resource can be reserved, de-allocated and redistributed without a central scheduling entity. The agent development framework used is the Lightweight Extensible Agent Platform (LEAP) [56] library combined with Java Agent DEvelopment framework (JADE) [44], a Foundation of Intelligent Physical Agents (FIPA) [27] compliant agent platform.

The AC generation and distribution based systems suffered in several incidents from severe damage and inoperability due to a single hit [94]. The USS Stark, Roberts, Princeton, and Cole incidents all left modern U.S. warships without electric power in the water [94]. The identified major weakness is the centralized controls that provide remote control with manual backup. This problem necessiated the redesign of the whole system. The transition process from traditionally radial AC power systems to zonal DC distribution systems was initiated and envisioned to lead to a fully Integrated Power System (IPS). The IPS takes advantage of a reconfigurable DC distribution system feeding load centers from two DC buses. The location of the two buses is chosen to minimize the likelihood of both buses being disabled due to a single disturbance. Besides the flexibility in system configuration, the IPS reduces weight, costs due to labor and improves the overall survivability of the combatant with reduced manning. The envisioned shipboard power system in the form of a block diagram is shown in the Fig. 2.1. The block diagram corresponds to the Naval Combat Survivability Generation & Propulsion and DC distribution testbeds [74, 88].

The ability to fight through combat damage requires systems with the ability to sense, isolate, and quickly compensate for major disruptions. Modern military systems share a fundamental challenge: to ensure continuity of service for distributed mission and life critical services despite both natural and hostile disruptions. The shipboard power system includes a spatially distributed, variable structure physical plant and associated hybrid sensing, communication, control, and actuation facilities. Damage is assumed to be clustered spatially and temporally resulting in concurrent disruption of both the physical plant and the control system [95]. The primary objective is to provide continuity of service despite faults and failures.

The Naval Combat Survivability Generation & Propulsion and DC distribution testbeds

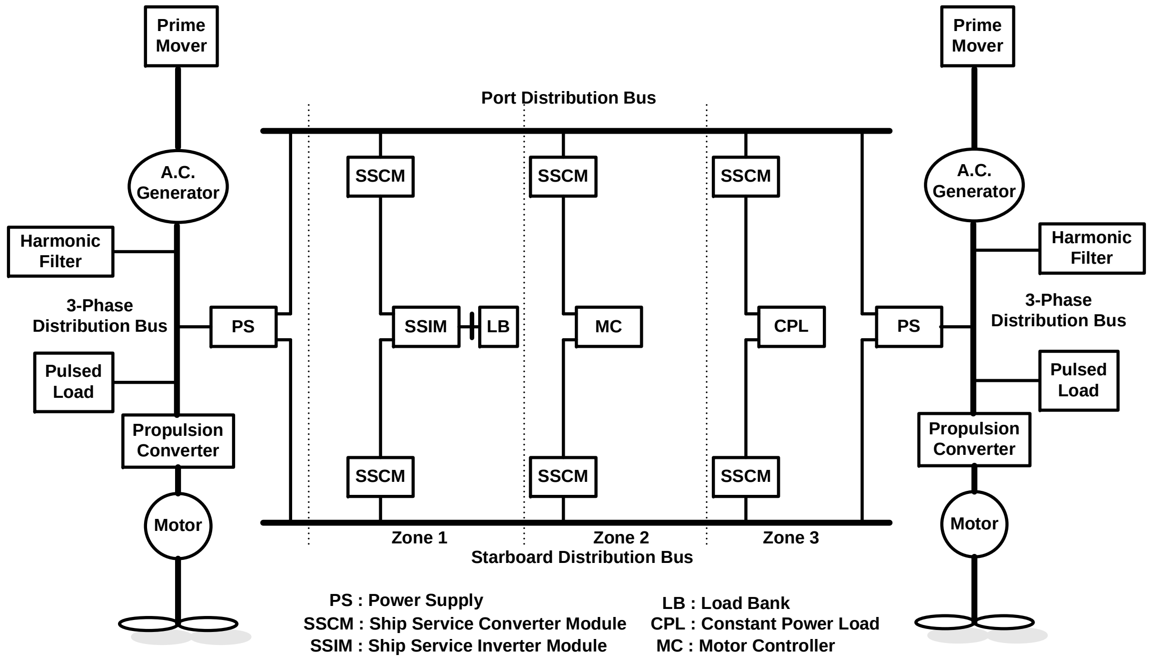


Figure 2.1: *NCS Generation & Distribution Test Bed*

have been developed as a common base for modeling the electric warship. The testbed is fed by two AC sources represented by induction motors as prime movers, synchronous generators, and exciters. The propulsion systems are connected to the AC system using power electronics for a flexible drive system. The AC is converted to DC for distribution of the electric power by two DC buses, the port bus and starboard bus. To improve survivability, the loads on board are grouped into zones and each zone is supplied from both of the DC buses by additional DC/DC converters. This arrangement allows a certain redundancy as well as graceful performance degradation as the converters are used to limit currents and isolate faulty loads or entire parts of the system. The loads represent typically encountered power demands including induction motors, three-phase AC-loads, pulsed loads, and constant power loads.

The remaining control challenge requires advanced control strategies to ensure system stability and to take advantage of the system's capabilities [104]. The controls to be developed include local controls for the various power electronic devices and the automation of processes to help system operators in coping with routine operations, maintenance, and emergency situations to optimize life-time and cost performance. The answer to this problem is envisioned in dependable automation strategies [95] that for example extend nonlinear control theory, apply analytic redundancy, utilize distributed intelligence to form robust networks, and allow reconfiguration based on situational awareness.

2.2 Agent Based Systems

The Open Autonomy Kernel (OAK) in [81] is a control agent architecture that is based on a unique combination of model-based reasoning and software agents. OAK has been implemented on Navy auxiliary systems, such as the chilled water Reduced Scale Advanced Demonstrator (RSAD) at the Naval Surface Warfare Center, Philadelphia. It shows how control agents based on Markov blankets may be used to develop effective control of large chemical and biological surveillance sensor grids.

Rockwell Automation [78, 63] has developed a set of tools and methodologies to help design, build, test, and verify distributed control systems that overcome the survivability problem of single controller approaches. Using these tools and methodologies, they have devised an agent architecture that has been applied to a US Navy shipboard chilled–water system. Simulation results indicate that the approach can lead to systems that operate in a real-life application with critical constraints such as survivability achieved through dynamic reconfiguration with reduced human supervision.

In [77] a multiagent approach is proposed for power system restoration. The proposed system consists of a number of *bus agents BAGs* and a single *facilitator agent (FAG)*. The BAG is developed to decide a suboptimal target configuration after a fault occurence by interacting with other BAGs based on only locally available information, while the FAG is to act as a manager in the decision process. Simulation results show that the method is able to reach sub-optimal target configurations, which are compared with those obtained by a mathematical programming approach.

ARCHON (ARchitecture for Cooperative Heterogeneous ON-line systems), Europe's largest project in the area of DAI [20], has devised a general-purpose architecture, software framework, and methodology that supports the development of DAI systems. A number of industrial domains such as electricity distribution and supply, control of a cement kiln complex, control of a particle accelerator, and control of a robotics application have used this concept for redesigning controls.

A real-life, wide-area, adaptive protection and control system, the Strategic Power Infrastructure Defense (SPID) system [57], has been developed by the Advanced Power Technologies (APT) Consortium [2]. By incorporating multiagent system technologies, the SPID system is able to assess power system vulnerability, monitor hidden failures of protective devices, and provide adaptive control actions to prevent catastrophic failures. In [57], a new concept is included for bargaining by multiagents to identify the decision options to reduce the system vulnerability.

A wide range of application domains makes use of agent-based systems. Agent applications are being developed for fields as varied as manufacturing, entertainment and electronic commerce. The following gives a brief overview of the different areas of application, for a more extensive list see [49].

Industrial applications of agent technology were among the first to be developed, and today, agents are being applied in a wide range of industrial systems such as manufacturing [73] [72], process control [20], telecommunications [82], air traffic control [15] [58], and transportation systems[11].

Entertainment applications such as computer games are extremely challenging and remunerative. Agents have a prominent role in computer games [35] [97], interactive theater [92] [37], and virtual reality applications. Such systems tend to be full of semi-autonomous animated characters, which can be implemented as agents.

Commercial applications tend to be oriented much more towards the mass market unlike industrial applications and involve highly complex systems that need to be operated in comparatively smaller areas. Some of the examples are information management applications like directory services, database inquiry, media indexing [60] [90]; service management applications like multimedia services, intelligent network management services, trip planning and guidance services [53] [18]; electronic commerce [17] [55], business management applications like financial services, workflow management, and office automation [48].

Medical informatics is a major growth area in computer science: new applications are being found for computers every day in the health industry and is a definite domain for agents to be applied. Two of the earliest applications are patient monitoring systems [36] and health care [42].

Agent-based systems are being used to further research other IT areas such as vision processing, learning and adaptive systems, speech processing, distributed knowledge-based systems, and human-computer interface.

2.3 Graph Algorithms

Ahuja in [1] introduces many applications in context with core network flow models — shortest path problems, maximum flow problems, minimum cost flows, and network optimization models like minimal spanning trees. Graph algorithms have diverse fields of applications including engineering, management science, computer science and communications, and manufacturing.

The maximum flow problem has been extensively studied for a long period of time and many algorithms have been developed. The author in [33] surveys the recent improvements that have been made in theoretical performance of maximum flow algorithms.

Graph theory has been applied to many electric power systems related applications. In [87] the problems of the steady-state security enhancement of radial distribution networks after structural disturbances causing violations of imposed operating limits are considered. Corrective actions used for relief of violations are of switching operations type. The paper describes the development of a graph-oriented control algorithm based on the linearized system model, where the synthesis of corrective controls is formulated as a combinatorial problem of mixed-integer programming. A heuristic algorithm is suggested as a solution of the problem. The fact that all disturbances as well as corrective actions are of topological nature makes the application of graph representation the natural way of describing the system, having both algorithmic and computer implementation consequences. The feasibility of the algorithm is tested on two examples of real medium-voltage distribution networks. Results show good properties of the proposed approach and its suitability for the solution of practical corrective control problems.

In [96] a novel approach using the minimal cutsets with minimum net flow to island the system following large disturbances has been presented. Slow coherency [103] has been proved effective in determining sets of generator groups among weak connections in any given power system. In [96] two comprehensive approaches to deal with islanding the actual system based on the grouping information by using the minimal cutsets technique in graph theory are provided. The issue of minimal cutsets has been widely discussed in areas related to network topology determination, reliability analysis, etc. The results show potential in application to power system islanding. The verification of the islanding scheme is provided based on a WECC 179-Bus, 29-Generator test system.

2.4 Decision Networks

Decision networks are a tool for representing decision scenarios for decision makers. A solution to a decision network is a set of strategies which ensures maximal expected utility. Many algorithms have been proposed for determining the optimal decision policies in decision networks. The following discusses those relevant to this research.

An algorithm has been developed by Shachter [85], that can evaluate any well formed influence diagram and determine the optimal policy for its decisions. The influence diagram can be analyzed directly, hence there is no need to construct other representations such as decision trees [54]. The result is that the analysis can be performed using the decision maker's perspective on the problem. In [71] clinical examples are used to illustrate the mathematical operations of the influence diagram evaluation algorithm.

In [41] methods for managing the complexity of information displayed to people responsible for making high-stakes, time-critical decisions are described. The techniques provide tools for real-time control of the configuration and quantity of information displayed to a user, and a methodology for designing flexible human-computer interfaces for monitoring applications. After defining a prototypical set of display decision problems, details regarding how to enhance computer displays used for monitoring complex systems by measuring the expected value of revealed information is described. The presentation is motivated by discussing the efforts to employ decision-theoretic control of displays for a time-critical monitoring application at the NASA Mission Control Center in Houston.

The use of decision networks to build mixed-initiative [65] decision-support agent systems for real-world applications is demonstrated in [24]. Agents should represent domain information that resembles knowledge of a human expert and objectively analyze alternatives that provide human decision makers with choices and corresponding justifications. This approach is demonstrated in [24] by examples in the traffic management and the academic advising domain to highlight application possibilities.

2.5 Conclusion

This chapter listed and summarized the research work done with respect to the shipboard power systems, multiagent systems, graph theory, and decision networks. It gave an insight on the various approaches adopted for simulating and modeling the shipboard power system, and the different theoretical approaches to incorporate reconfiguration. Also, various application areas of multiagent systems and application of the same to the domain of shipboard power system were briefly listed. Graph theory algorithms as applied to power systems was discussed along with the application of decision networks.

Chapter 3

Software Agents and Agent-Based Systems

This chapter gives an overview on the various concepts and terminologies associated with software agents and agent-based systems. The term "software agent" and its associated attributes along with the various agent architectures will be discussed. Furthermore, multiagent systems and the advantages of solving a problem using this approach are addressed. A review of various agent communication strategies and interaction protocols is given. The Java language is reviewed as suitable programming language for agent-based systems. Over the past few years computer scientists have introduced a new paradigm – agent-based software engineering which is also explained.

3.1 Software Agents

In this section the definition of an *agent* is given and various agent attributes are discussed.

3.1.1 Software Agents - Definition

Various definitions from different disciplines for the term agent exist today. In [29], the authors discuss several definitions of agents of well-known authors in the domain.

According to [99]: "An agent is a computer system that is situated in some environment, and that is capable of flexible autonomous actions in this environment in order to meet its design objectives."

In [61] it is defined as: "Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realise a set of goals or tasks for which they are designed."

Ferber in [26] arrived at the following definition:
An agent is a physical or virtual entity

(a) which is capable of acting in an environment,
(b) which can communicate directly with other agents,
(c) which is driven by a set of tendencies or goals (in the form of individual objectives or of a satisfaction/survival function which it tries to optimise),
(d) which possesses resources of its own,
(e) which is capable of perceiving its environment (but to a limited extent),
(f) which has only a partial representation of this environment (and perhaps none at all),
(g) which possesses skills and can offer services,
(h) which may be able to reproduce itself,
(i) whose behaviour tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representation, and the communication it receives.

Weiss [98] describes: "An agent is a computational entity such as a software program or a robot that can be viewed as perceiving and acting upon its environment and that is autonomous in that its behaviour at least partially depends on its own experience."

IBM's White Paper [31] defines agents as:
"... software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires."

In this thesis, the following definition is applied:

> A software agent is a thread of execution that is situated in some *environment* and capable of *flexible autonomous* interaction with this environment in order to meet its goals.

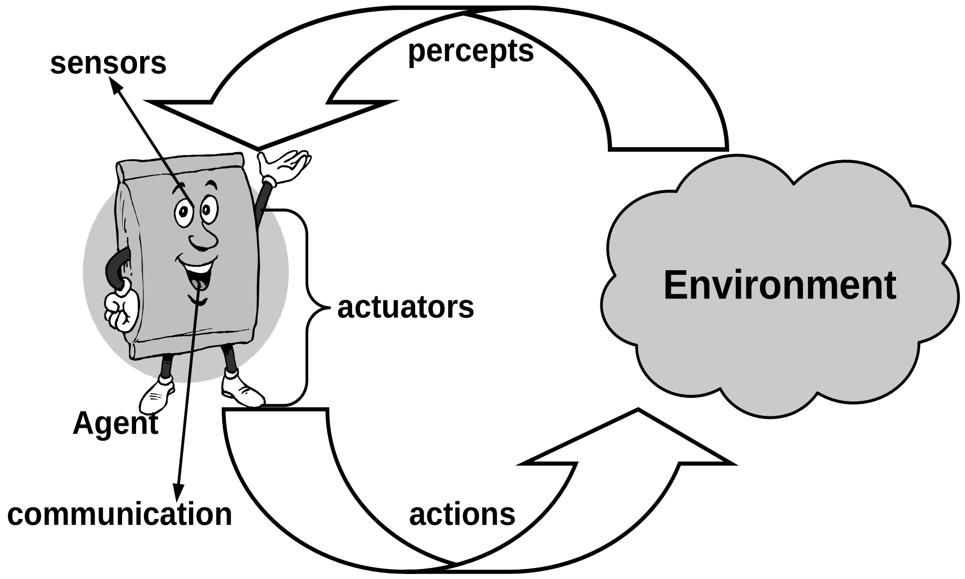


Figure 3.1: *Software Agents*

An agent and its interaction with an environment is shown in Fig. 3.1. The agent posseses means of sensing, communicating, processing, and acting. According to the given definition, three vital concepts can be identified: environment, autonomy, and flexibility.

- *Environment:* The agent's sensors perceive inputs from its environment and perform actions through actuators that affect the environment in some way.
- *Autonomy:* The system should be capable of taking its own actions without being directed or supervised by humans (or other agents), and that it should have control over its internal state and actions.
- *Flexibility:* Means the following:
  - *Reactive:* Perceive the environment and respond in a timely fashion to changes that occur in that environment;
  - *Pro-active:* Exhibit goal-directed behavior and take the initiative when appropriate;
  - *Social:* Interact with other agents and humans in order to complete and solve a problem while simultaneously help others in their activities.

Other Agent Attributes

Some of the other commonly identified agent attributes are as follows [30]:

- *Mobility:* Agents can have the ability to move to different environments and carry data as well as a set of intelligent instructions which can be executed remotely.

- *Temporal Continuity:* Agents are continuously running processes, not “one shot” computations that terminate.
- *Adaptivity:* Agents continuously adapt to changes in the environment.

The ongoing interest in autonomous agents did not emanate from a vacuum but researchers and developers from various disciplines have been making progress on related issues in the last decade. Contributions have been mainly from the fields of: artificial intelligence, object-oriented programming and concurrent object-based systems, and human-computer interface design. For more details regarding the same refer to [49].

3.1.2 Software Agent Architectures

The following describes possible agent architectures [98]:

Logic Based Agents

Decision making in logic based approaches to building agents, is considered as deduction process. An agent’s “program,” i.e., its decision making strategy, is encoded as a logical theory, and the process of selecting an action is reduced to a problem of proof. Logic-based approaches are elegant and have clean (logical) semantics—wherein lies much of their long-lived appeal. But logic-based approaches have many disadvantages. The inherent computational complexity of theorem proving makes it questionable whether agents as theorem provers are capable of operating effectively in time-constrained environments. Decision making in logic-based agents is based on the assumption of calculative rationality—the assumption that the state of the world is not going not change in any substantial way while the agent is deciding what to do, and that an action which is rational when decision making begins will be rational when it concludes.

Subsumption Architecture

Subsumption architecture is the best-known reactive agent architecture, in which decision making is implemented in some form of direct mapping from situation to action. In other words, an agent makes a decision through a set of task accomplishing behaviors. Each behavior is regarded as an individual action function and designed to achieve a particular task. These behaviors are implemented as rules in the form:

$$situation \rightarrow action$$

which maps perceptual input directly to actions. As multiple rules can be fired simultaneously, a subsumption hierarchy is utilized to arrange behaviors into layers. Lower layers in the hierarchy can inhibit higher layers. The higher layers abstract higher-level behaviors while the lower layers implement more basic and time constrained tasks. The overall decision is made through a set of behaviors with the inhibit relations linking them.

The major advantage of this architecture is that it has a high computational efficiency and is robust to single point failures. The disadvantages are decision making is based on information about the agent's current state and it must inherently take a 'short-term' view, difficulty in building up agents of multiple layers, and the difficulty in applying iterative learning to improve agents' behaviors. In addition, no methodology other than trial and error is available to build such agents.

Belief-Desire-Intention (BDI) Agents

BDI architectures are practical reasoning architectures, in which the process of deciding what to do resembles the kind of practical reasoning that we appear to use in our everyday lives [40]. The basic components of a BDI architecture are data structures representing the beliefs, desires, and intentions of the agent, and functions that represent its deliberation (deciding what to do) and means-ends reasoning (deciding how to do it). Intentions play a central role in the BDI model: they provide stability for decision making, and act to focus the agent's practical reasoning.

The procedure of BDI decision-making is shown in Fig. 3.2. The belief revision function (BRF) takes the perceptions and the agent's current beliefs as input to determine a set of new beliefs. The option generation function (generate options) decides on the basis of the current beliefs and intentions a series of options. These generated options serve as new desires. The filter function, which models the agent's deliberation process, will then be utilized to actually select some of the desires. The chosen desires become intentions and will guide the action selection function (execute), which determines an action to be performed by the agent.

A major issue in BDI architectures is the problem of striking a balance between being

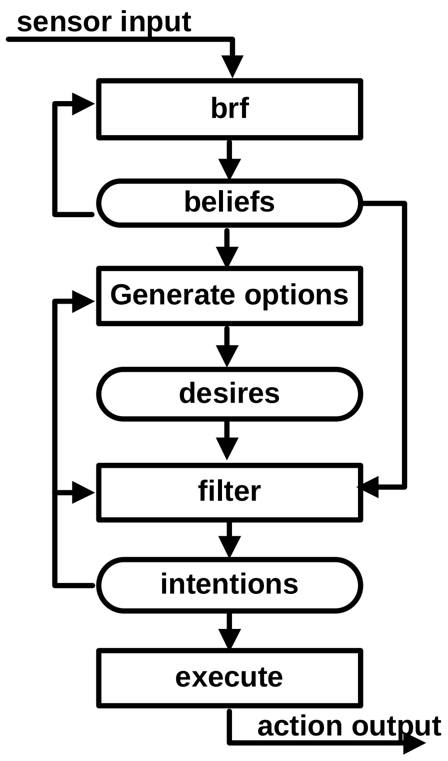


Figure 3.2: *BDI Architecture*

committed to and overcommitted to one's intentions. The BDI model is attractive for several reasons. First, it is intuitive—we all recognize the processes of deciding what to do and then how to do it, and we all have an informal understanding of the notions of belief, desire and intention. Secondly, it gives a clear functional decomposition, which indicates what sort of subsystems might be required to build an agent.

Layered Architectures

Layered architectures as shown in Fig. 3.3 are currently the most popular general class of agent architectures. Layering represents a natural decomposition of functionality: it is easy to see how reactive, pro-active, and social behavior can be generated by the corresponding layers in an architecture. Information and control flow within the layers are identified in two ways: Horizontal layering and Vertical layering. In horizontal layered architectures the software layers are each directly connected to the sensor input and action output. In vertically layered architectures the sensor input and action output are connected to a fixed layer. The advantage of horizontally layered architectures is their conceptual simplicity: an agent can

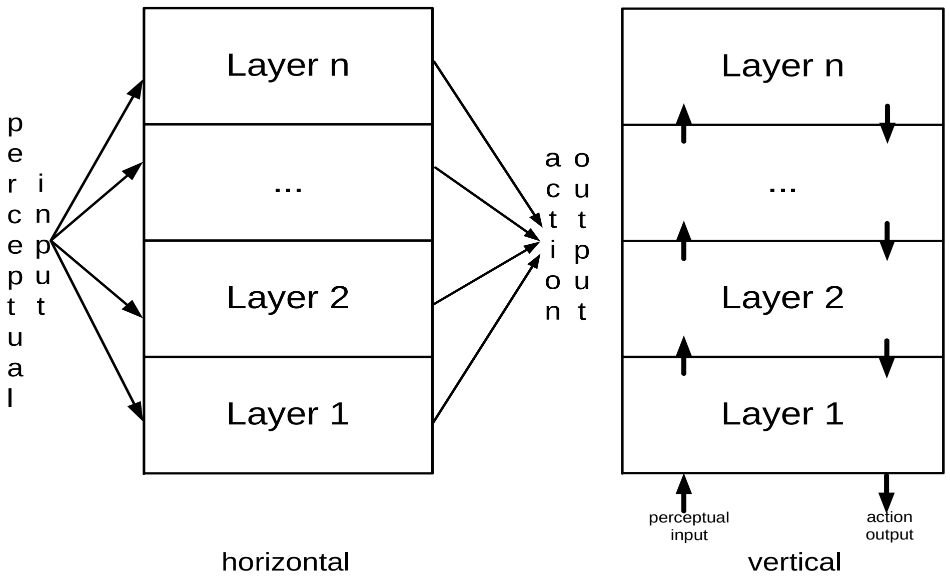


Figure 3.3: *Layered Architecture*

exhibit n different types of behaviors by simply employing n layers. However, inconsistent actions might be suggested and unstable overall agent behavior may result. The problem can be alleviated in the vertically layered architectures. The vertically layered architectures can be divided into one-pass architectures and two pass architectures. In onepass architectures, control flows sequentially through each layer until the final layer generates action output. In two-pass architectures, information flows up the architecture and control flows back down.

With the basic notion of an agent, its attributes defined, and various agent architectures discussed, the next section takes the step towards a population of co-operating agents.

3.2 Multiagent Systems

A multiagent system (see Fig. 3.4) is composed of a population of agents. The agents interact with each other to reach common objectives while simultaneously pursue individual objectives [26]. In such a system, the agents form a loosely-coupled network of problem solvers and work together to solve problems that are beyond their individual capabilities [101]. These problem solvers, characterized by various degrees of problem solving capabilities, may be spatially distributed and heterogeneous in nature. Research in multiagent systems is mainly connected with coordinating intelligent behavior among these agents so that they collectively benefit from individual knowledge, goals, skills, and plans and take appropriate actions to solve problems.

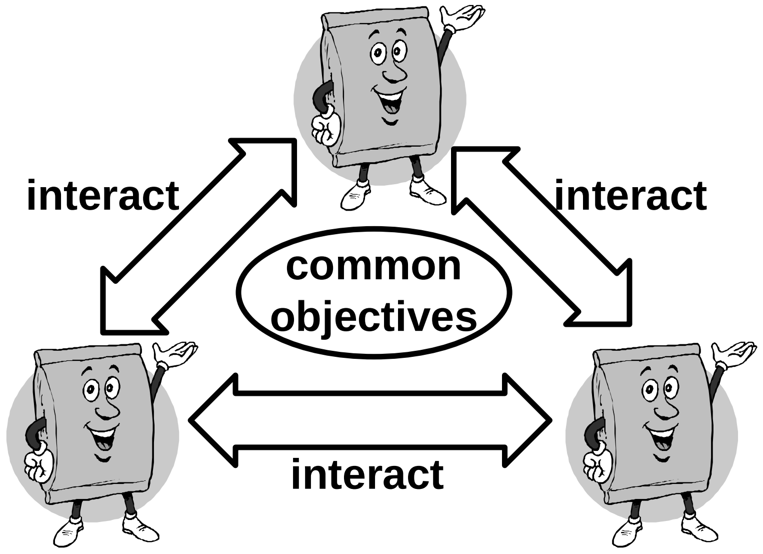


Figure 3.4: *Multiagent Systems*

Each agent can be identified as an entity (e.g., a machine, a plant or a part) and thus help in incremental growth and flexible expansion. The advantage of scalability is provided as each agent is allowed to join a system, begin coordinating with other agents, and leave a system once it has achieved its goal without affecting the operation of the system.

3.2.1 Characteristics of Multiagent Systems

The following is a summary of typical multiagent systems [49]:

- Every agent has incomplete knowledge or capabilities in solving the problem, hence a limited viewpoint.
- Multiagent systems are typically open [98] and have no centralized control.
- Data is decentralized and computations are performed asynchronously.

3.2.2 Communication Protocols

Agents' communication is vital in order to achieve better goals for themselves or of the society/system in which they exist. Communication protocols enable agents to exchange and understand messages. As an example, a communication protocol might specify that the following types of messages can be exchanged between two agents:

- Propose a course of action

- Accept/Reject a course of action
- Disagree with a proposed course of action
- Counterpropose a course of action.

Communication allows coordination of actions and behavior among the agents, resulting in systems that are more coherent. According to [98] the following are the possible communication protocols:

Coordination

Coordination is a property of a system of agents performing some activity in a shared environment. The degree of coordination is the extent to which the agents are able to avoid extraneous activity by reducing resource contention, avoiding deadlock, and maintaining applicable safety conditions. Cooperation is coordination among harmonious agents, while negotiation is coordination among competitive or self-interested agents. Typically, to cooperate successfully, each agent must have a model of other agents and also develop a model of future interactions.

Speech Acts

Speech Act theory [83] is a popular basis for analyzing communication where in the spoken human communication is used as the model for communication among computational agents. Speech act theory views human natural language as actions, such as requests, suggestions, commitments, and replies. For example, when you request something, you are not simply making a statement, but creating the request itself. It helps define the type of message by using the concept of the illocutionary force. The sender's intended communication act is clearly defined, and the receiver has no doubt as to the type of message sent.

KQML

The knowledge query and manipulation language (KQML) is a protocol for exchanging information and knowledge. The elegance of KQML is that all information for understanding the content of the message is contained in the communication itself. KQML-speaking agents appear to each other as clients and servers. The communication among such agents can be either synchronous or asynchronous. KQML messages can be "nested," i.e., the contents of a KQML message may be another KQML message, which is self contained.

Ontologies

An ontology is a specification of the objects, concepts and relationships in an area of interest. Concepts can be represented in first-order logic as unary predicates and higher-arity predicates represent relationships. An ontology is more than a taxonomy of classes; the ontology must describe the relationships. It is analogous to a database schema, not the contents of a database itself.

3.2.3 Interaction Protocols

Interaction protocols govern the exchange of a series of messages among agents, i.e., a conversation. For instance, based on the message types given in the previous section, the following conversation corresponds to an interaction protocol for negotiation between Agent1 and Agent2:
Agents start negotiation by

- Agent1 proposes some action to Agent2
  Agent2 evaluates the proposal and
- sends acceptance/rejection to Agent1 OR
- sends counterproposal to Agent1 OR
- sends disagreement to Agent1

As per [98], several interaction protocols have been devised for systems of agents:

Coordination Protocols

In an environment with limited resources, agents must coordinate their activities with each other to further their own interests or satisfy group goals. The actions of multiple agents need to be coordinated because there are dependencies between the agents' actions, there is a need to meet global constraints, and no one agent has sufficient competence, resources or information to achieve system goals. While the distributed goal formalism has been used frequently to characterize both global and local problems, the key agent components are *commitment* and *convention*. Commitments are viewed as pledges to undertake a specific course of action, while conventions provide a means of managing commitments in changing circumstances. Commitments and conventions are the cornerstones of coordination: commitments provide the necessary structure for predictable interactions, and social conventions provide the necessary degree of mutual support.

Contract Net

The contract net protocol is an interaction protocol for cooperative problem solving among agents and is most widely used. It is modeled on the contracting mechanism used by businesses to govern the exchange of good and services. The contract net provides a solution for the so-called connection problem: finding an appropriate agent to work on a given task.

An agent wanting a task solved is called the manager; agents that might be able to solve the task are called potential contractors. From a manager's perspective, the process is to announce a task, receive and evaluate bids from potential contractors, award a contract to a suitable contractor, and receive and synthesis results. From a contractor's perspective, the process is to receive task announcements, evaluate one's capability to respond, respond or decline, perform the task, and report results. The contract net offers the advantages of graceful performance degradation. If a contractor is unable to provide a satisfactory solution, the manager can seek other potential contractors for the task.

Blackboard System

The blackboard system as shown in Fig. 3.5 consists of a set of specialized knowledge sources, a centralized blackboard data structure, and a control strategy. The blackboard data structure acts as a global data repository, which facilitates indirect communication between knowledge sources. It contains input data, partial solutions and other data that are in various problem-solving states. The blackboard can be thought of as a dynamic "library" of contributions to the current problem that have been recently "published" by other knowledge sources.

The blackboard architecture was first introduced in the Hearsay II speech recognition project [25]. It featured a system with multiple knowledge sources or independent agents, each with a specific domain of expertise related to speech analysis. Each agent works on its own pace on its part of the problem, it refers to the blackboard to pick up new information posted by other agents, and in turn posts its results to the blackboard. Thus, the blackboard architecture allows multiple agents to work independently and cooperate with each other towards solving the problem.

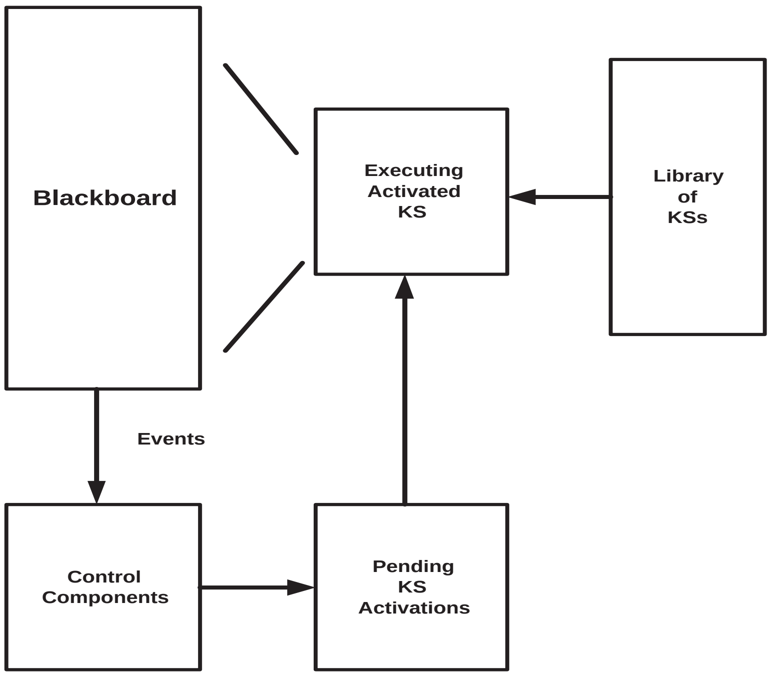


Figure 3.5: *Architecture of basic blackboard system showing the blackboard, knowledge sources (KS) and control components.*

Market Mechanisms

Market mechanisms are effective for coordinating the activities of many agents with minimal direct communication among the agents. Everything of interest to an agent is described by current prices - the preferences or abilities of others are irrelevant except insofar as they affect the prices. There are two types of agents *consumers*, who exchange goods, and *producers*, who transform some goods into other goods. Agents bid for goods at various prices, but all exchanges occur at current market prices. All agents bid so as to maximize either their profits or their utility.

One of the oldest applications of market mechanisms is in decision-theoretic planning, which models the costs and effects of actions quantitatively and probabilistically. For many applications, where the probabilities can be estimated reliably, this leads to highly effective plans of actions.

Negotiations

A frequent form of interaction that occurs among agents with different goals is termed negotiation. It is a process by which a joint decision is reached by two or more agents, each trying to reach an individual goal or objective. The agents first communicate their positions, which might conflict, and then try to move towards agreement by making concessions or searching for alternatives.

The major features of negotiation are (1) the language used by the participating agents, (2) the protocol followed by the agents as they negotiate, and (3) the decision process that each agent uses to determine its positions, concessions, and criteria for agreement.

3.2.4 Advantages of Multiagent Systems

Multiagent systems offer a way to mitigate the constraints of centralized, planned, sequential control and provide systems that are decentralized, emergent, and concurrent. Applications are inherently distributed either functionally or spatially. For instance, a group of experts with different specialities collaborating to solve a complex problem corresponds to an application being functionally distributed. The advantages offered by using autonomous agent-based systems can be identified as follows:

- Fault Tolerance – Agents are a distributed mechanism towards problem solving and thus a system made of autonomous agents will not collapse when one or more of its components fail as there will not be any single point of failure.
- Self-Configuring Systems – A collection of agents are capable of reconfiguring themselves as they run. This is crucial for systems that need to respond to a wide range of operating conditions. As the overall system behavior emerges from local decisions, the system readjusts itself automatically to environment noise or the removal of other agents. Thus a fully functional self-configuring system can be smoothly implemented by merely networking agent resources.
- Modular Software/Scaleable Architecture – The reason why agents are considered as dynamic and powerful entities is because of the factorization of the problem they provide. Every agent can be considered as an entity and thus allows incremental growth and flexible expansion. The advantage of scalability is provided as each agent is allowed to join or leave a system.

- Decreased communication – By exchanging only high level solutions with other agents.
- Flexible systems – By having agents with distinctive expertise dynamically team up to solve problems.
- Faster Problem Solving – By employing parallelism.

This section has discussed the basic issues that must be dealt with while building a MAS. These issues are concerned with providing communication and interaction in the MAS. Communication enables the agents in the MAS to exchange information on the basis of which agents coordinate their actions. Interaction is very important as it is the process of interaction that allows several intelligent agents combine their efforts. The next section discusses the motivation for the use of Java in multiagent systems development.

3.2.5 JAVA

To correctly develop autonomous, intelligent, reactive pieces of software, we must have good ways of implementing, debugging, and evaluating them. This section describes in detail various reasons why Java [45] renders itself as a suitable choice to construct software agents and also build an agent framework [6].

Object-Oriented

Object-oriented design is a very powerful concept as it facilitates the clean definition of interfaces and makes it possible to provide reusable software components. It is an approach that focuses design on the data and on the interfaces to it. It is also the mechanism for defining how modules "plug and play." Though there are many languages which are object oriented in nature, Java is one of the few languages which enforces it. The user has no choice but to encapsulate all data in objects. Since agents are essentially built around a group of different object components, Java is an ideal language for developing them.

Network-Savvy

Communication is a very crucial aspect in the development of multiagent systems. Agents located on different machines and in different environments have to communicate information and knowledge about their goals, beliefs and intentions to each other to coordinate and cooperate so as to bring about a coherent solution. Thus communication is a very important aspect in the development of any MAS. Java lends itself as an extremely suitable choice in this regard. It offers an comprehensive library of classes and routines which cope easily with both UDP and TCP/IP protocols and thus supports sending both broadcast and directed messages across the network. Moreover it provides the feature of RMI (Remote Method Invocation) which is extremely helpful while creating a family of interacting agents. The process of creating network connections is extremely easy when compared to other languages.

Secure

As Java is intended for use in networked/distributed environments, a great deal of importance has been placed on security. Java allows for the construction of virus-free and tamper-free systems. The authentication techniques are based on public-key encryption.

Architecture Neutral and Portable

As agents are characterized by being spatially distributed, applications must capable of being executed anywhere on the network without prior knowledge of the target hardware and software platform. Using Java has definitely a cutting edge, as it is architecture neutral and portable. Java adopts a “binary code format” that is nondependent of the hardware architectures and operating system interfaces as a measure to solve the binary-distribution problem. The format of this system-independent binary code is architecture neutral. No special porting activity is needed for an application written in Java to execute in a given hardware and software environment if the Java run-time platform is made available. The Java compiler generates bytecodes, which is a high-level machine independent code for an abstract machine that is implemented by the Java interpreter and run-time system. The most important benefit of the interpreted bytecode approach is that compiled Java programs are portable to any system on which the Java interpreter and run-time system have been implemented. The architecture-neutral feature is one huge step towards achieving portability. And also programming languages such as C and C++ suffer from the defect of designating many fundamental data types as “implementation dependent.” Java eliminates this issue by defining a standard behavior that will apply to the data types across all platforms.

Database Connectivity - JDBC

The Java Database Connectivity kit lets Java programmers connect to any relational database, query it, or update it using the industry standard structured query language (SQL). This is a very useful feature as databases are among the most common uses of software and hardware today.

Native Methods

While developing a multiagent application, there are often situations when it is essential to use the code written in a different language(native code) and call it from Java (either the system needs some pre-existing legacy code or it is very essential in order to maximize the speed of the code). To facilitate this, Java provides ways to call already compiled code (native methods). The features of JDBC along with native methods help in enabling legacy software integration.

3.3 Software Engineering of Agent-Based Systems

Agents are generally regarded as autonomous decision making systems, which sense and act in some environment. Agents appear to be a promising approach to developing complex applications, ranging from internet-based electronic commerce and information gathering to industrial process control [46].

3.3.1 Agents and AI

The research in the field of AI had been largely responsible for the emergence of the discipline of intelligent agents [100]. One way of defining AI is as the problem of building an intelligent agent [79]. But it is important to draw a thin line of difference between the broad intelligence that is the clear-cut goal of the AI community, and the intelligence which is sought for in agent-based systems. The only intelligence required of agents is that they must make an acceptable decision about what action to perform next in their environment in time for this decision to be useful. Other requirements for intelligence will be determined by the domain in which the agent is going to be applicable: for instance, not all agents need to exhibit a learning capability. Agents are nothing but software components that must be designed and implemented in the same way as any other software component. AI techniques are more often the most appropriate way of building agents.

3.3.2 Agents and Expert Systems

Expert systems were considered the AI technology of the 1980s [100]. For knowledge-rich domains, expert systems are the most appropriate as they are capable of solving problems or giving advice. MYCIN is a classic example of an expert system. It was intended to assist physicians in the treatment of blood infections in humans. The most important distinction between agents and expert systems is that expert systems like MYCIN are inherently disembodied. They are not capable of interacting directly with any environment, i.e., they get information not via sensors, but through a user acting as middle man. Additionally, expert systems are not required to operate in real-time. Finally, in expert systems it is not required that they are capable of cooperating with other agents. But inspite of these differences between expert systems and agents, some expert systems, look very much like agents. An apt example for this is ARCHON [47], which started as a collection of expert systems, and ended up being viewed as a multiagent system. ARCHON operates in the domain of industrial process control.

3.3.3 Agents and Objects

The growing interest in programming languages such as JAVA has aided the object oriented approaches in being considered as the "best practice" not only by the academic computer science community but also by the software industry. Agents are identical to objects in most important respects: they encapsulate state and behavior, and communicate via message passing. Although there are certain similarities between object and agent oriented approaches, there are also a number of important differences [51]. Firstly, objects are generally passive in nature hence they come alive when they are sent a message. Secondly, although objects encapsulate state and behavior, they do not encapsulate behavior activation i.e., action choice. Thus, any object can invoke any publicly accessible method on any other object. Once the method is invoked, the corresponding actions are performed. Thirdly, object-oriented technology does not succeed to provide adequate set of concepts and mechanisms for modeling complex systems. According to [8] for modeling complex systems objects, classes, and modules provide an essential yet insufficient means of abstraction. Finally, object-oriented approaches provide only minimal support for specifying and managing organizational relationships.

3.3.4 Agent-Based Systems

The technology of multiagent systems seems to radically alter the way in which complex, distributed, open systems are conceptualized and implemented. Adopting a multiagent approach to system development affords software engineers a number of significant advantages over contemporary methods. This does not imply that multiagent systems are a silver bullet [9]—there is no evidence to suggest they will represent an order of magnitude improvement in software engineering productivity. However, for certain classes of application, an agent-based approach can significantly improve the software development process. This approach follows the phases of Specification, Implementation, and Verification [100] [50].

Specification

The software development process begins by establishing the client's requirements. When this process is completed, a specification is developed which sets out the functionality of the new system to be developed. The purpose of this section is to consider what a specification for an agent-based system might look like. What are the requirements for an agent specification framework? What sort of properties must it be capable of representing? Taking the view of agents as practical reasoning systems, the predominant approach to specifying agents has involved treating them as intentional systems that may be understood by attributing to them mental states such as beliefs, desires, and intentions [99]. An agent specification framework must be capable of capturing at least the following aspects of an agent-based system:

- the *beliefs* agents have;
- the *ongoing interaction* agents have with their environment;
- the *goals* that agent will try to achieve;
- the *actions* that agents perform and the effects of these actions.

The most successful approach appears to be the use of a temporal modal logic [99]. The fundamental problem faced by agent researchers is developing a formal model that gives a good account of the interrelationships between the various attitudes that together comprise an agents internal state [99]. Comparatively few serious attempts have been made to specify real agent systems using such logics - see, [28] for one such attempt.

Implementation

Once given a specification, we must implement a system that is correct with respect to this specification. The next issue to be considered is converting this abstract specification to concrete computational model. There are at least three possibilities for achieving this transformation as specified in [100]:

1. manually refine the specification into an executable form by following some principled but informal refinement process;
2. directly execute or animate the abstract specification; or
3. translate or compile the specification into a concrete computational form using an automatic translation technique.

Refinement
At the time of writing, most software developers use structured but informal techniques for transforming specifications into concrete implementations. One of the most common techniques in widespread use are based on the idea of top-down refinement. In this approach, an abstract system specification is refined into a number of smaller, less abstract sub-system specifications, until the derived sub-systems are simple enough to be directly implemented and which together satisfy the original specification. It is required that each step represents a true refinement of the more abstract specification that preceded it. For functional systems, the refinement process is well understood, and comparatively straightforward [66]. For reactive systems, refinement is not so straightforward, as reactive systems must be specified in terms of their ongoing behavior. The refinement problem for agent-based systems, where specifications may be regarded as even more abstract than those for reactive systems, is quite challenging.

Structured but informal refinement techniques are the backbone of real-world software engineering. For agent-based techniques to become widely used outside the academic community, the approach of informal, structured methods for agent-based development will be essential. An example, one such technique is followed in [59] using a standard specification technique (in their case, Z), and then using traditional refinement methods (in their case, object-oriented development) for transforming the specification into an implementation. This approach has the advantage of being familiar to a much larger user-base than entirely new techniques, but suffers from the disadvantage of presenting the user with no features that make it particularly well-suited to agent specification. A similar approach is taken in this research by using Unified Modeling Language (UML) statecharts [70] for specification and the object oriented development to transform the specification into an implementation using the Java programming language.

Directly Executing Agent Specifications

One major disadvantage with manual refinement methods is that they introduce the possibility of errors. If no proofs are provided, to demonstrate that each refinement step is indeed a true refinement, the correctness of the implementation process depends upon little more than the intuitions of the developer. This is definitely undesirable for applications in which correctness is a major issue. One possible way of overcoming this problem, is to get rid of the refinement process altogether, and perform direct execution of the specification.

Suppose a system specification is given, Φ, which is putforth in some logical language L. One way of obtaining a concrete system from Φ is to treat it as an *executable specification*, and *interpret* the specification directly in order to generate the agent's behavior. Interpreting an agent specification can be viewed as a kind of constructive proof of satisfiability. By this it can be shown that the specification Φ is satisfiable by building a model (in the logical sense) for it. If models for the specification language L can be computationally interpreted, then model building can be viewed as executing the specification.

Compiling Agent Specifications

An alternative to direct execution is compilation. In this approach, the abstract specification is transformed into a concrete computational model using some automatic synthesis process. The main advantage of compilation over direct execution is the run-time efficiency. Direct execution of an agent specification, involves the manipulation of a symbolic representation of the specification at run time. This manipulation corresponds to reasoning of some form, which is computationally costly. Compilation approaches aim to reduce abstract symbolic specifications to a much simpler computational model, which requires no symbolic representation. The "reasoning" work is performed off-line, at the time of compilation; execution of the compiled system can then be done with little or no run-time symbolic reasoning.

The general approach of automatic synthesis, although theoretically appealing, is limited in a number of important aspects. As the agent specification language becomes more expressive, even off-line reasoning becomes too expensive to carry out. The systems generated in this way are not capable of learning, (i.e., they are not capable of adapting their "program" at runtime).

Finally, as the case with direct execution approaches, agent specification frameworks donot have concrete computational interpretation, making such a synthesis practically not possible.

Verification

Once a concrete system is developed, it should be shown that this system is correct with respect to the original specification. This process is known as verification, and it is especially important if any informality is introduced into the development process. The approaches to the verification are divided into two broad classes: (1) axiomatic; and (2) semantic (model checking).

Axiomatic Approaches
Axiomatic approaches to program verification were the first to enter the mainstream of computer science, with Hoare's work [39] in the late 1960s. Axiomatic verification necessitates that a logical theory should be derived systematically from the concrete program that corresponds to the behavior of the program. This can be termed as the program theory. If the program theory is expressed in the same logical language as the original specification, then verification reduces to a proof problem: showing that the specification is a theorem of (equivalently, is a logical consequence of) the program theory. The development of a program theory is made feasible by axiomatizing the programming language in which the system is implemented. For instance, Hoare logic gives more or less an axiom for every statement type in a PASCAL-like language. Given the axiomatization, the program theory can be derived from the program text in a systematic way.

Semantic Approaches: Model Checking
Axiomatic verification reduces to a proof problem and hence they are inherently limited by the difficulty of this proof problem. Proofs are hard enough, even in classical logic; the addition of temporal and modal connectives to a logic makes the problem considerably harder. For this reason, more efficient approaches to verification have been sought. One successful approach is that of *model checking*. As the name suggests, model checking approaches are based on the semantics of the specification language unlike axiomatic approaches which generally rely on syntactic proof.

The model checking problem is simple: for given a formula Φ of language L and a model M for the language L, the validity of Φ in M has to be determined. Model checking-based verification has been studied in connection with temporal logic. The technique relies upon the close relationship between models for temporal logic and finite-state machines. Suppose that Φ is the specification for some system, and Π is a program that is supposed to be implementing Φ. Then, in order to determine whether or not Π truly implements Φ, a model M_Π is generated that corresponds to Π, in the sense that M_Π encodes all the possible computations of Π. The main advantage of model checking over axiomatic verification is in complexity. Model checking using the branching time temporal logic CTL [19] can be done in polynomial time, whereas the proof problem for most modal logics more complex.

3.4 Conclusion

The chapter gave an overview on agents, agent-based systems and agent based software engineering. The various components and approaches both necessary and possible to construct agent-based systems have been discussed. The following chapters will consequently address the modeling of shipboard power system as the domain of interest and provide the tools to construct and program the "inside" of agents necessary to develop the autonomous energy management system.

Chapter 4

Modeling the Electric Shipboard Power System

The challenge of designing an agent-based energy management system requires the possibility to interact with a model of the electric shipboard power system. Only with the availability of either a mathematical or hardware model of the physical system can the higher levels of the design goals be evaluated. This work will build on a mathematical model and the following will introduce the assumptions made in defining an appropriate model and its interface to the other parts of the autonomous energy management system.

4.1 Model Description

The modified version of the shipboard power system is shown in Fig. 4.1. The model builds on the integrated electric power system as introduced earlier but ignores the pulsed load components and adds an additional load in each of the zones. For this work, all components are modeled by discrete transfer functions of first and second-order to represent input-output relationships concerning active power and voltage values. This setup allows modeling enough details for the energy management system while decreasing the required simulation time. The components' transfer functions and parameters as used for implementation in the MATLAB/Simulink environment are given in Appendix B.

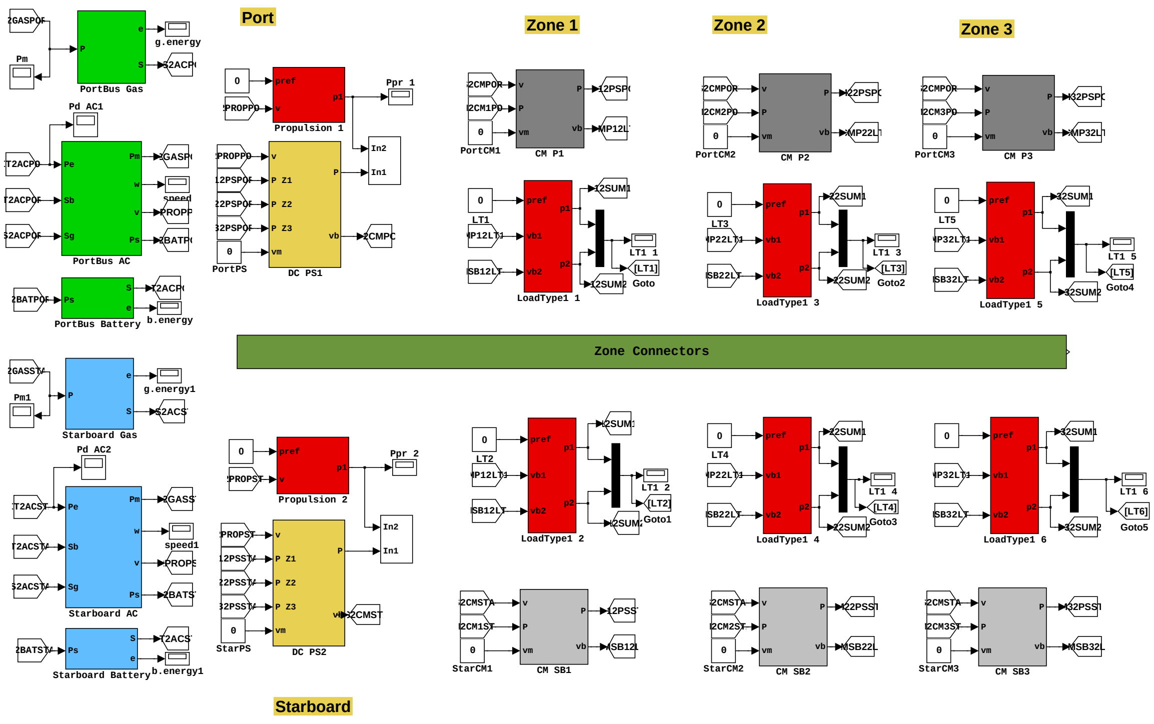


Figure 4.1: *Shipboard Power System*

4.2 Interfacing the Physical Model

The above shown physical system model is based on only local controls for individual components. If any of the components fail, the rest of the system gets affected resulting in either bringing down of the whole system or parts of the load not receiving power. What is needed is an autonomous energy management system that has the ability to automatically reroute the power flow based on the current state of the system. For example, if a line is lost due to some catastrophic event, then an alternate route should be chosen on the fly with minimal or no involvement of human operators. The energy management system should also incorporate the criteria of restoring power to loads based on their priorities.

The platform for implementing such an energy management system for the shipboard power system should facilitate the execution of multiple threads of actions working towards their goals asynchronously. At the same time the platform has to be portable and architectural neutral. Java, which supports multithreading and which is considered as a defacto standard programming language, is chosen as the ideal platform. Another important reason for choosing the Java platform is its compatibility and interoperability with the MATLAB/Simulink environment.

4.3 Conclusion

This chapter introduced the mathematical model of the shipboard power system as appropriate for the energy management system. The important aspects of interfacing the model with the autonomous energy management components to be developed has been addressed and lead to the choice of combining MATLAB/Simulink with agents and tools to be written in Java.

Chapter 5

Energy Management System

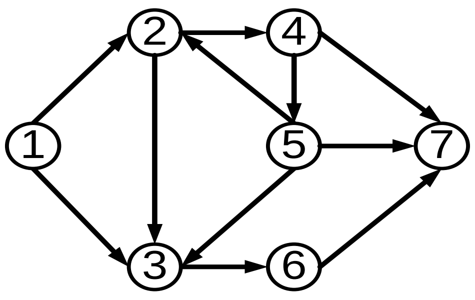
The chapter gives details regarding the shipboard energy management system. As the design builds on a graph theoretic approach using a specific form of maximum flow algorithm, a brief introduction to graph theory followed by the maximum flow problem and self-stabilizing maximum flow algorithms is given. Consequently, the issues of transforming the general algorithm into an application considering constraints given by a naval system are addressed and an agent-based maximum flow algorithm for the energy management system constructed.

5.1 Graph Theory

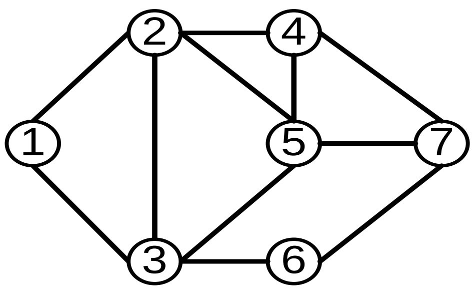
Graph theory is a domain that lies on the cusp of several fields of interest, including applied mathematics, computer science, engineering, management and operation research. The maximum flow problem is an elementary problem in graph theory and combinatorial optimization. The maximum flow problem seeks a feasible solution that sends the maximum amount of flow from a source to a sink node and will be useful in the design of the energy management system.

5.1.1 Definitions and Notation

In this section the basic definitions and notation from graph theory are presented. Also, an overview of some fundamental properties of graphs is given.



(a) ***Directed graph***



(b) ***Undirected graph***

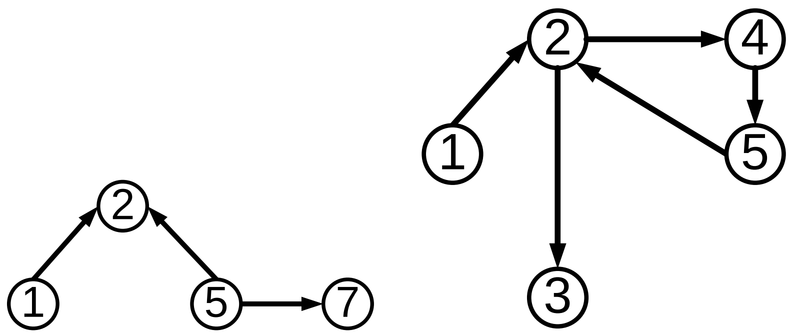
Figure 5.1: *Directed and Undirected Graphs*

Directed Graphs and Networks: A *directed graph* $G = (N, A)$ consists of a set N of nodes i_i and a set A of arcs a_i whose elements are ordered pairs of distinct nodes. Fig. 5.1(a) gives an example of a directed graph. For this graph, $N = \{1, 2, 3, 4, 5, 6, 7\}$ and $A = \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 6), (4, 5), (4, 7), (5, 2), (5, 3), (5, 7), (6, 7)\}$. A *directed network* is a directed graph whose nodes and/or arcs have associated numerical values to represent, e.g., costs or capacities. They are also termed as weighted directed graphs.

Undirected Graphs and Networks: *Undirected graphs* and *undirected networks* are defined in the same manner as their respective directed counterparts except that arcs are unordered pairs of distinct nodes. Fig. 5.1(b) gives an example of an undirected graph. An undirected $arc(i, j)$ can be regarded as a two-way street with flow permitted in both directions: either from node i to node j or from node j to node i.

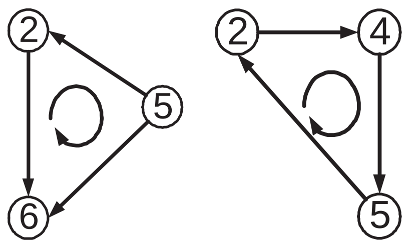
Walk and Path: A *walk* in a directed graph $G = (N, A)$ is a subgraph of G consisting of a sequence of nodes and arcs $i_1 - a_1 - i_2 - a_2 - ... - i_{r-1} - a_{r-1} - i_r$ satisfying the property that for all $1 \leq k \leq r - 1$, either $a_k = (i_k, i_{k+1}) \in A$ or $a_k = (i_{k+1}, i_k) \in A$. A *path* is a walk without any repetition of nodes. Fig. 5.2 illustrates two walks: 1-2-3-4 and 1-2-4-3-2-6. Fig. 5.2(a) shows a path but Fig. 5.2(b) is not because node 2 is visited twice.

Directed Walk and Directed Path: A *directed walk* is an "oriented" version of a walk in the sense that for any two consecutive nodes i_k and i_{k+1} on the walk, $(i_k, i_{k+1}) \in$ A. Fig. 5.2*(b)*



(a) **(b)**

Figure 5.2: *Examples of paths and walks in graphs*



(a) **(b)**

Figure 5.3: *Examples of cycles*

shows a directed walk. A *directed path* is a directed walk without repetition of nodes, i.e., it has no backward arcs.

Cycle: A *cycle* is a path $i_1 - i_2 - ... - i_r$ together with the arc (i_r, i_1) or (i_1, i_r). Fig. 5.3(a) gives an example of a cycle.

Directed Cycle: A *directed cycle* is a directed path $i_1 - i_2 - ... - i_r$ together with the arc (i_r, i_1). Fig. 5.3(b) shows a directed cycle.

Acyclic Graph: A graph is *acyclic* if it contains no directed cycle.

5.1.2 The Maximum Flow Problem

A fundamental problem in graph theory is the maximum flow problem. Many different sequential and parallel algorithms have been developed [1]. All the parallel algorithms but the one by Gosh, Gupta, and Pemmaraju in [34] present implementations that do not run in polylogarithmic time on a polynomial number of processors. Other advantages of this algorithm are: simple, passive, self-stabilizing, and local checking and corrections only. The algorithm has been investigated here to find a solution to the power flow problem of the electric shipboard system. The following briefly defines the maximum flow problem and notation used. More details can be found in [34].

A directed graph (digraph) $G = (V, E)$ with $n = |V|$ number of nodes and $e = |E|$ number of edges as shown in Fig. 5.4. The graph should not contain any directed cycles of length 2: if $(i, j) \in E$ then $(j, i) \notin E$. The nodes s and t are the distinct source and sink nodes, respectively. The source node s has no incoming edges, the sink node t has no leaving edges. Every edge (i, j) is associated with a real-valued flow $f(i, j)$ and a non-negative real-valued capacity $C(i, j)$. Any feasible flow f in G obeys the flow conservation constraint, i.e., the *incoming flow* $I_f(i) = \sum_{(j,i)\in E} f(j, i)$ equals the *outflow* $O_f(i) = \sum_{(i,k)\in E} f(i, k)$:

$$\text{for all } i \in V - \{s, t\} : \ I_f(i) = O_f(i)$$

Flows are also satisfying the skew symmetry property of $f(i, j) = -f(j, i)$. The maximum flow problem is to maximize the outflow of the source node $O_f(s)$.

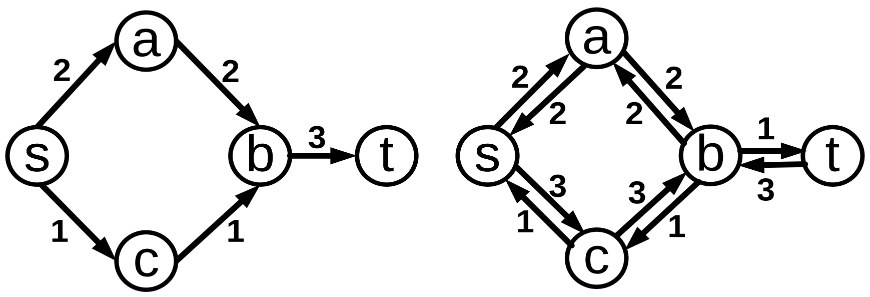
Definition 1: For any flow f and for each pair of nodes $(i, j) \in V \times V$, the *residual capacity* $r(i, j)$ is equal to $C(i, j) - f(i, j)$.

Definition 2: For any flow f in G, the *residual graph* G_f of G with respect to f is defined as the weighted digraph $G_f = (V_f, E_f)$, where $V_f = V$ and $(i, j) \in E_f$ if and only if $r(i, j) > 0$.

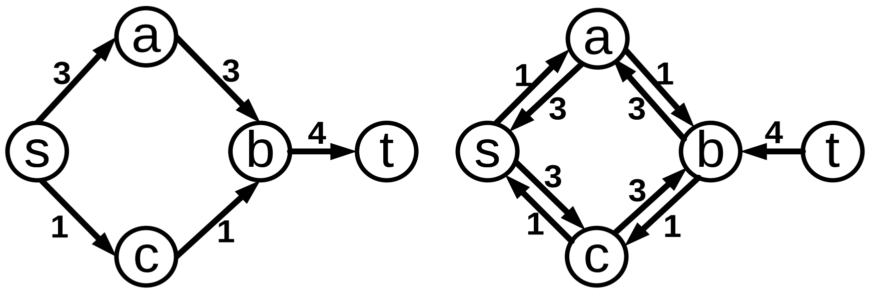
Definition 3: A directed path in the residual graph G_f from s to t is called an *augmenting path.*

Example: A feasible flow of a graph and its residual graph are given in Fig. 5.4(a). All edges have a capacity of 4 and $C(s, a) = C(a, b) = C(s, c) = C(c, b) = C(b, t) = 4$. The number on an edge indicates the flow along an edge, e.g., the flow from node b to the target node t $f(b, t) = 3$. The residual graph for the flow gives the remaining capacity, for example the residual flows $r(b, t) = C(b, t) - f(b, t) = 4$ - $3 = 1$ and $r(t, b) = C(t, b) - f(t, b) = 0 - (-3) = 3$.

A maximum flow and its corresponding residual graph are shown in Fig. 5.4(b), respectively. It has been achieved by increasing the flow on the augmenting path $s - a - b - t$ by its minimum residual capacity of $r(b, t) = 1$. Note that after increasing the flow, the updated



(a) ***Feasible flow and its residual graph***



(b) ***Maximum flow and its residual graph***

Figure 5.4: *Example graph with edge capacities of 4*

residual graph contains no augmenting path and, therefore, the flow from the source to the target cannot be increased further.

Applications

The maximum flow problem arises in a wide variety of situations and in several forms. A list includes the Feasible Flow Problem, Problem of Representatives, Matrix Rounding Problem, Scheduling on Uniform Parallel Machines, Distributed Computing on a Two-Processor Computer, and Tanker Scheduling Problem. For an overview of applications see [1].

5.1.3 Self-Stabilizing Algorithms

Two radically different approaches toward fault tolerance are followed in literature. In *robust algorithms* each step of each process is taken with sufficient care to ensure that, inspite of failures, correct processes only take correct steps. In *self-stabilizing algorithms* correct processes can be affected by failures, but the algorithm is guaranted to recover from any arbitrary configuration when the processes resume correct behavior [91]. Robust algorithms continuously show correct coordinated behavior even when failures occur, but the number of failures is limited and the failure model must usually be known precisely. In self-stabilizing algorithms any number of failures of arbitrary types is allowable, but correct behavior of the algorithm is suspended until some time after the repair of the failures [91]. A self-stabilizing algorithm can be started in any system configuration and eventually reaches an allowed state, and behaves according to its specifications from then on. Consequently, the effects of temporal failures die out, and also, there is no need to initialize the system consistently. Self-stabilizing (also referred to as stabilizing) algorithms were introduced as early as in 1974 [23].

Robust algorithms follow a *pessimistic* approach, suspecting all information received, and precede all steps by sufficient checks to guarantee the validity of all steps of correct processes. Validity must be guaranteed in the presence of faulty processes, which necessitates restriction of the number of faults and of the fault model.

Stabilizing algorithms are *optimistic*, which may cause correct processes to behave inconsistently, but guarantee a return to correct behavior within finite time after all faulty behavior ceases. That is, self-stabilizing algorithms protect against *transient failures* and eventual repair is assumed, and this assumption allows to abandon failure models. Rather than considering processes to be faulty, it is assumed that all processes operate correctly, but the configuration can be corrupted arbitrarily during a transient failure. The configuration at which the analysis of the algorithm starts is considered the initial one of the (correctly operating) algorithm. An algorithm is therefore called stabilizing if it eventually starts to behave correctly (according to the specification of the algorithm), regardless of the initial configuration.

Properties of Self-Stabilizing Algorithms

Self-stabilizing algorithms offer the following three fundamental advantages over classical algorithms.

- *Fault tolerance.* Self-stabilizing algorithms provide full and automatic protection against all transient process failures, because the algorithm recovers from any configuration, no matter how much the data has been corrupted by failures.
- *Initialization.* The need of proper and consistent initialization of the algorithm is eliminated, because the processes can be started in arbitrary states and yet eventually coordinated behavior is guaranted.
- *Dynamic topology.* A self-stabilizing algorithm computing a topology dependent function

converges to a new solution after the occurence of a topological change.

5.1.4 Self-Stabilizing Maximum Flow Algorithm

By observing the drawbacks of the technique presented by Awerbuch and Varghese [5], Ghosh et al. [34] were motivated to find a simple, passive, distributed self-stabilizing algorithm for the maximum flow problem that uses local checking and local correction only. Ghosh et al. have taken a step in this direction in [34] by presenting an algorithm that computes the maximum flow in an acyclic digraph starting in an *arbitrary* initial state. The algorithm is simple to understand and implement. In addition to being tolerant to transient faults, the algorithm can also automatically adjust to dynamic changes in network topology or in edge capacities. In particular, the self stabilizing algorithm can adapt (dynamically) to *(a)* arbitrary addition or deletion of edges in the graph, *(b)* addition and deletion of nodes provided that the number of nodes in the network is within a constant bound known a priori to all processes, and *(c)* arbitrary changes in the capacities of the edges.

The algorithm uses an approach similar to that of Goldberg and Tarjan [32]. It is assumed that each node in G contains a process that asynchronously makes moves based on local information only. The moves by a process update the local state of that process alone.

It is assumed that G is an acyclic digraph, and associated with each edge (i, j) in G there is a variable $f(i, j)$ that stores the current flow from node i to node j. Both process i and process j can read from and write into the variable $f(i, j)$. Each node can then change $f(i, j)$ by changing the value of its local variable. Each node i contains a single variable $d(i)$ that denotes what i "believes" to be the length of the shortest directed path from the source s to itself in the residual graph G_f. The value of $d(i)$ is restricted to be an integer in the range $[0..n]$. The flow $f(i, j)$ is referred to as the *f-value* of edge (i, j) and $d(i)$ as the *distance-value* or *d-value* of node i. Each node can make moves that update its d-value or the f-value(s) of incident edges. A move that updates a node's d-value is called a *d-move* and a move that updates an edge's f-value is called an *f-move*. The algorithm allows one type of d-move and three types of f-moves. The algorithm contains four guarded actions, labeled $GS1$ through $GS4$ for convenience. For each i, $1 \leq i \leq 4$, the guard and action in GSi are referred to as G_i and A_i respectively.

The main idea behind the algorithm is as follows. For any node $i \neq \{s, t\}$, let $demand(i) =$

$O_f(i) - I_f(i)$. In addition, let $demand(t) = \infty$. Each node i, $i \neq s$, tries to restore the flow conservation constraint $demand(i) = 0$ either by reducing its inflow if $demand(i) < 0$, or by increasing its inflow or reducing its outflow if $demand(i) > 0$. In particular, each node with positive demand attempts to "pull" flow via a shortest path from s to itself in the residual graph. If the node believes that a path from s to itself does not exist in the residual graph, then it rejects the demand by "pushing" it back along an outgoing edge. Breadth-first search is used to keep track of the shortest paths from s to all nodes in G_f. The following explains how the four guarded statements, $GS1$-$GS4$ implement the above idea.

- **Guarded statement** $GS1$. Each node i, $i \neq s$, computes its d-value by examining the values $d(j)$ for all $(j,i) \in E_f$. Note that it is referring to edges in the residual graph G_f, not in the original graph G. Let k be a node such that $(k,i) \in E_f$ and $d(k) = \min\{d(j) \mid (j,i) \in E_f\}$. If $d(k) < n$, then node i views node k as its predecessor on the shortest path from node s to node i in G_f. Therefore, in this case, node i sets its d-value to $d(k)+1$. If $d(k) = n$, then i believes that there is no path from s to itself in G_f and therefore $d(i)$ is also set to n. It is assumed that $d(s)$ is fixed at 0.
- **Guarded statement** $GS2$. For any node i, $i \neq s$, if $demand(i) < 0$ then the total flow along incoming edges in G is reduced irrespective of the d-value of i and the d-values of the neighbors of i.
- **Guarded statement** $GS3$. For any node i, $i \neq s$, if $demand(i) > 0$ and $d(i) < n$ then, i tries to meet this demand by "pulling" flow along an incoming edge $(j,i) \in E_f$ that i believes to be on a shortest path from s to itself in the residual graph G_f. For this, node i simply picks a node j such that $(j,i) \in E_f$ and $d(j) = d(i) - 1$. The flow is then increased along the edge (j,i) by the minimum of $demand(i)$ and $r(j,i)$.
- **Guarded statement** $GS4$. For any node i, $i \neq s$, if $demand(i) > 0$ and $d(i) = n$, then i believes that there is no path in G_f from s to itself, and hence the excess demand cannot be met. In this case node i "pushes" the excess demand for flow back towards the sink t.

A remark on the distinguished nodes s and t: Node s remains idle with its d-value fixed at 0 and node t executes the same algorithm as the rest of the nodes. However, the value of $O_f(t)$ is assumed to be fixed at ∞ implying that $demand(t)$ is always greater than zero. Thus, node t "drives" the algorithm by pulling flow along incoming edges until $d(t) = n$.

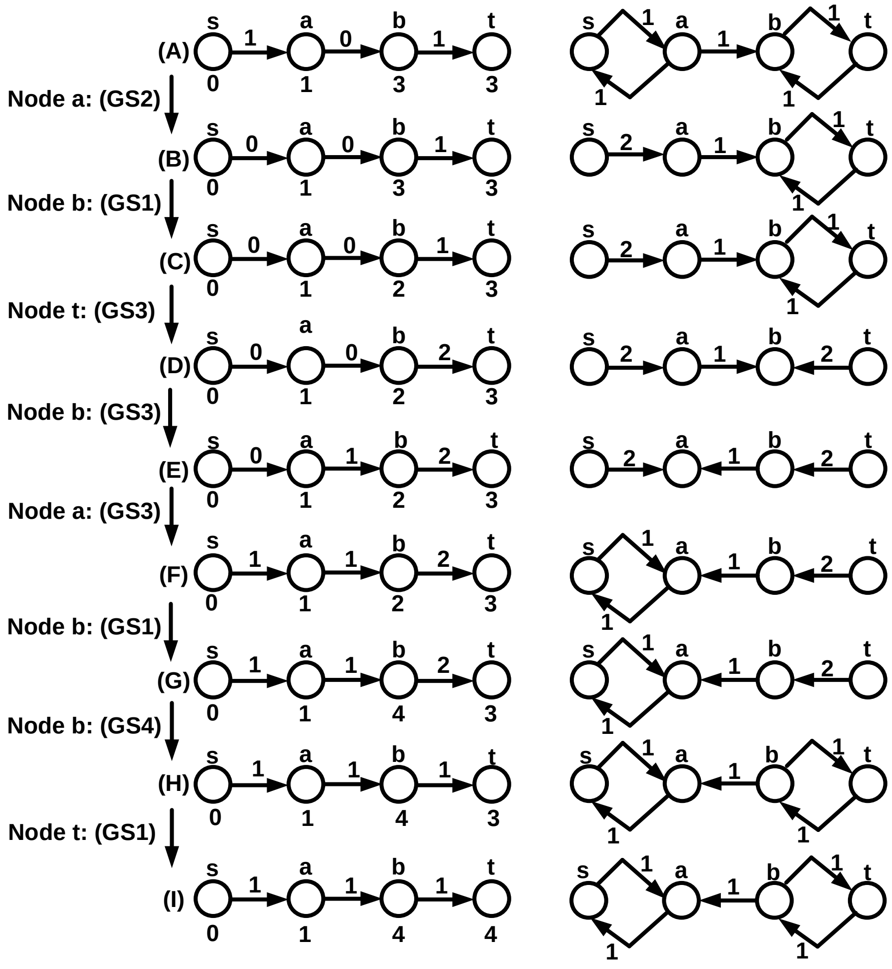


Figure 5.5: *Digraph and its Residual Graph for an Example*

The algorithm provides an upper limit on the number of moves necessary before a global solution is found based on the number of nodes n in the graph: n^2.

An example taken from [34] is shown in Fig. 5.5 and demonstrates the algorithm. The digraph G on which the algorithm is run is simply a directed path of length 3 whose nodes are labeled s, a, b, and t. The graphs on the left show how the d-values and the f-values change in G as nodes make moves. The d-value of each node is shown under the corresponding node and the f-value of each edge is shown above the corresponding edge. The graphs on the right shows the corresponding residual graphs along with weights (residual capacities) of the edges.

The shortcomings associated with this algorithm is that it does not consider the priority of the loads, i.e., loads are not restored based on their priorities. And also the algorithm does not take care of capacity violations of the edges. These shortcomings are addressed by extending the algorithm by three additional Guarded Statements namely, $GS5 - GS7$, which are explained in detail in the following section.

5.2 The Shipboard's Power Flow as Maximum Flow Problem

5.2.1 Requirements

The electric shipboard power system is modeled as an acyclic directed graph and its residual graph. The transformation of the block diagram into the acyclic graph is achieved by representing physically linked components by neighboring nodes in a graph. Each of the edges between nodes can be associated with a direction because the electric power flows only from the AC generators toward the loads. The multiple generators (sources) and loads (sinks) can be accommodated by introducing supersource and supersink nodes with additional edges. This is a common step in maximum flow algorithms and does not introduce any limitations (see for example [1]). The edges' capacities from the supersource and to the supersink nodes represent the upper bounds on generation capabilities and load demands, respectively. In order to avoid bi-directional flows (loops of length 2) in the DC distribution bus, the bus has been represented as a single node rather then a bi-directed edge. Every load is associated with a priority value which can be *high* or *low* under different system conditions. Each agent should perform locally by observing its environment and taking corrective actions to satisfy its goals, i.e., to request power for a load, to route the power flow according to load priorities, etc. The agents' actions are taken without synchronization. By making local and asynchronous moves that follow a self-stabilizing concept, convergence to the desired solution is achieved. Also, as disturbances can be seen as arbitrary initial conditions, tolerance to transient faults that upset system conditions is given. The energy management system for the SPS is based on steady state analysis and short term transients are not considered. The algorithm published by [34] provides such a framework for the desired agents' behavior and allows the design of agents that adjust to changes in network topology and edge capacity. The computational model and changes made to the algorithm to accommodate requirements of the energy management system are presented next.

5.2.2 Agent's Maximum Flow Algorithm

Every node in the graph is allowed to make moves by changing its distance value or flows. The algorithm executed by the agents at their nodes i $(i \neq s)$ is given in Listing 5.1. The algorithm is based on seven guarded statements $GS1 - GS7$ that consist of their respective guard G_i and action A_i. The first six actions represent local steps taken to find a solution to the power flow problem. Guarded statement $GS7$ notifies other agents by publishing to the blackboard its status of making local changes or convergence to a local solution.

The algorithm is continuously executed by every agent $i \neq s$ in order to restore the $demand(i) = O_f(i) - I_f(i) = 0$. The two distinguished agents s (source) and t (sink) differ in their behavior from other nodes: Agent s is idle; agent t performs the same program as the other nodes but with a $demand(t) = \infty$. The agents' actions perform changes to increase or decrease their inflows or to reduce their outflows. Agents with the belief that no direct path from the source to itself exists push back the demand on their outgoing edges. The following describes the guarded statements in more detail:

Guarded statement GS1: Every node i, $i \neq s$, updates its $d(i)$-value by determining $d(k) = \min\{d(j) \mid (j,i) \in E_f\}$. If $d(k) < n$ then node k is a predecessor of node i and $d(i) = d(k) + 1$, otherwise $d(i) = n$ and no direct path from the source to node i exists. The distance value of the source node is set to $d(s) = 0$.

Guarded statement GS2: Every node i, $i \neq s$, reduces the total flow along incoming edges in case $demand(i) < 0$.

Guarded statement GS3: Every node i, $i \neq s$, with $demand(i) > 0$ and $d(i) < n$ increases the flow on available incoming path $(j,i) \in E_f$ with $d(j) = d(i) - 1$, by the minimum of $demand(i)$ and residual flow $r(j,i)$.

Guarded statement GS4: Every node i, $i \neq (s,t)$, with $demand(i) > 0$ and $d(i) = n$ reduces the outflow on the path $(i,j) \in E$ where $f(i,j) > 0$.

Guarded statement GS5: Every node i, $i \neq s$, checks the flows on incoming edges $(j,i) \in E$ and reduces flows in case of capacity violations. Note that the above guarded statements never lead to this condition in case a valid initial flow existed. This statement rather allows changing system conditions including the loss of a link.

Listing 5.1: *Maximum Flow Algorithm*

```
Node $s$: idle with d($s$) = 0, Node $t$: demand($t$) = $\infty$,
initially history($i$) = 1.

Nodes' algorithm: for agent $i$, $i \neq s$
begin
    $\{GS1\}$ : $d(i) \neq \min(D(i) \cup \{n\})$     ? $d(i) := min(D(i) \cup \{n\})$
    $\{GS2\}$ : $demand(i) < 0$                      ? Reduce_Inflow($i$)
    $\{GS3\}$ : $\exists\, j \in IN(i) : pull(j,i)$  ? $f(j,i) := f(j,i) + \min(demand(i), r(i,j))$
    $\{GS4\}$ : push($i$)                            ? Reduce_Outflow($i$)
    $\{GS5\}$ : $\exists\, j \in LIMIT(i)$           ? $f(j,i) := f(j,i) - (f(j,i) - C(j,i))$
    $\{GS6\}$ : validate($i$)                        ? Update_Flows($i$)
    $\{GS7\}$ : change($i$)                          ? Notify($i$)
end

Procedure Reduce_Inflow($i$)
begin
   Find $(k,i) \in E$ such that $f(k,i) > 0$
   $f(k,i) = f(k,i) - \min(-demand(i), f(k,i))$
end

Procedure Reduce_Outflow($i$)
begin
   Find $(i,k) \in E$ such that $f(i,k) > 0$
   $f(i,k) = f(i,k) -$ min($demand(i), f(i,k)$)
end

Procedure Update_Flows(i)
begin
   1a. $f(i,j) = f(i,j) - \min(demand(i), f(j,k))$
   1b. $f(i,j) = f(i,j) - \min(demand(i), f(j,k))$
   2. Find $m \in$ LLN such that $f(m,t) > 0$
      $f(k,t) = f(k,t) + r(k,t) \wedge f(m,t) = f(m,t) - r(k,t)$
end

Procedure Notify(i)
begin
   Blackboard_Send(status($i$))
   history($i$) = status($i$)
end

$Notation$:
demand($i$)  = $Of(i) - If(i)$                             D($i$)        = {d($p$) + 1 | $p \in$ IN($i$)}
IN($i$)      = $\{j \mid (j,i) \in Ef \wedge (j,i) \in E\}$  pull($j,i$)   = (demand($i$) > 0) $\wedge$ (d($i$) < n) $\wedge$
                                                                           (d($j$) = d($i$) - 1)
OUT($i$)     = $\{j \mid (i,j) \in E\}$                     push($i$)     = (demand($i$) > 0)$\wedge$(d($i$) = n)$\wedge$($i \neq t$)
LN           = $\{j \mid (j,t) \in E\}$                     LIMIT($i$)    = $\{j \mid$ f$(j,i) >$C$(j,i) \wedge (j,i) \in E\}$
HLN          = $\{i \mid j \in LN \wedge priority = high\}$  priority(i)   = { '1' if $(j,k) \in E \wedge k \in LLN \wedge$
                                                                           f$(j,k) > 0$ else '0'}
validate(i)  = 1a. (demand($i$) $> 0$ $\wedge$ d($i$) $= n \wedge i \notin$TN $\wedge$ $j \in$TN $\wedge$ f$(i,j) > 0$ $\wedge$ priority$(i) = 1)\vee$
               1b. (demand($i$) $> 0$ $\wedge$ d($i$) $= n \wedge i \in$TN $\wedge$ $j \in$LLN $\wedge$ f$(i,j) > 0)\vee$
                2. (demand($i$) $= 0$ $\wedge$ d($i$) $= n \wedge i \notin$TN $\wedge$ $k \in$HLN $\wedge$ $(j,k) \in E$ $\wedge$ f$(j,k) <$C$(j,k)$)
LLN          = $\{i \mid j \in LN \wedge priority = low\}$   status($i$)   = G1 $\vee$ G2 $\vee$ G3 $\vee$ G4 $\vee$ G5 $\vee$ G6
TN           = $\{j \mid (j, LN) \in E\}$                   change(i)     = status(i) $\neq$ history(i)
```

Guarded statement GS6:

- Every node i, $i \neq s$, with $demand(i) > 0$, $d(i) = n$:
  - $i \notin TN$ (TN corresponds to special nodes connected directly to the load nodes), and i connected to $j \in TN \wedge (i, j) \in E$ with $flow(i, j) > 0$, if $priority(i) = 1$ (it is 1 if it is connected to a low priority load 'k' which is supplied, otherwise 0) then reduce flow along (i, j) by the minimum of $demand(i)$ and $f(j, k)$
  - $i \in TN$ and i connected to a low priority load j with $(i, j) \in E \wedge flow(i, j) > 0$ reduce the flow on low priority load j by the minimum of $demand(i)$ and $f(i, j)$
- For every node $i, i \neq (s, TN)$, $demand(i) = 0$, $d(i) = n$ and i is connected to $j \in TN$ and if j is connected to unsupplied high priority load 'k' then reduce supply to low priority loads, by minimum of $demand(j)$ and flow to low priority loads $f(i, j)$ until the high priority load is supplied. Increase the flow to high priority load proportionately.

Guarded statement GS7: Every node i, $i \neq s$, publishes its status to the blackboard in case it has changed.

A mapping of the physical, electrical components/subsystems to the graph theory concept is shown in Fig. 5.6. By depicting the shipboard power system as directed acyclic graph, considering every component as a node, the reconfiguration layer is comprised of 22 agents. The list of agents includes 2 AC sources, 2 power supply modules, 2 DC distribution buses, 6 converter modules, 6 loads in the zones, 2 propulsion loads, and the supersource and supersink nodes.

5.3 Discussion

Designing agents that work together effectively involves several different kinds of activities like sharing the tasks, information, and the dynamic co-ordination of multiagent activities. Agents are forced to cooperate due to their limited knowledge and capabilities and need to work together towards the common goal. The agents in the graph-theoretical approach taken here follow a "Partial Global Planning" concept [98]. This distributed planning concept is well suited for applications where some uncoordinated activity is part of the problem solving. As such, the graph theory introduced here provides the means of establishing a design concept that allows programming the agents' body and therefore taking the step from a formal concept towards a specific implementation.

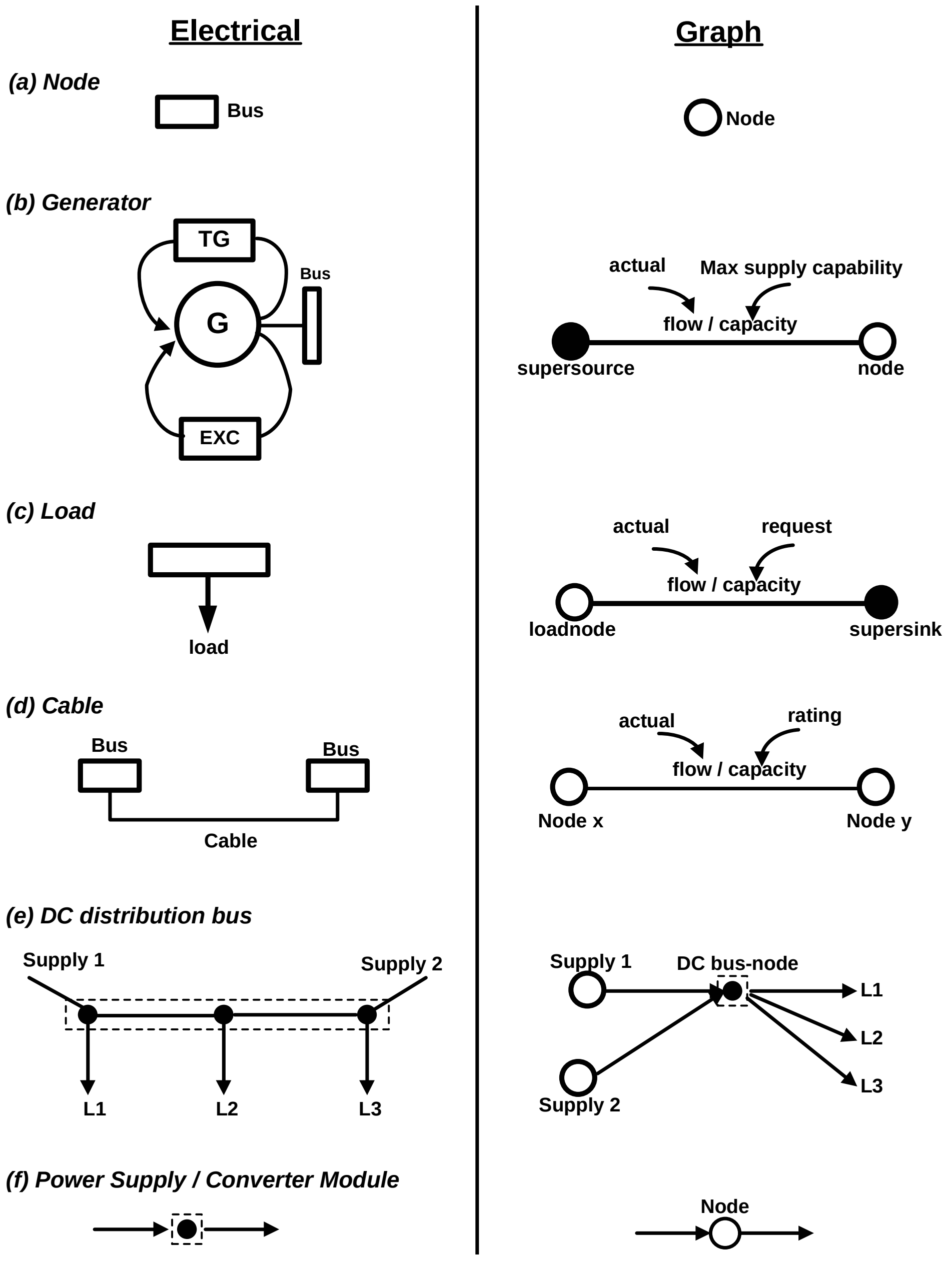


Figure 5.6: *Mapping of electrical components to graph theoretic representation*

5.4 Conclusion

This chapter discussed the challenges of providing an autonomous energy management component for the operation of electric shipboard power systems. The designed multiagent system allows a decentralized approach in solving the power flow problem while incorporating specific needs of a naval system: maximum supply according to component priorities. Different levels of priority within in the same level for the various components can also be easily incorporated into the existing system. The apparent problems of this decentralized design include communication issues and the detection of failures. These issues will be addressed in the following chapter.

Chapter 6

Situational Awareness Component

This chapter builds on the multiagent approach introduced in the previous chapter and addresses issues of identifying problems in the physical system layer as well as the operation of the reconfiguration agent layer. The challenges to be addressed are: What problems may occur during the operation and how can a component that supports the human operator in identifying and solving such problems be constructed? The most important step towards this goal is the recognition of an "uncertainty element" as the decentralized and autonomous approach relies on functionally correct agents and the availability of communication links. Therefore, the following will introduce a tool for making decisions under uncertainty and tailor the tool to the shipboard power system application.

6.1 Decision Problems and Decision Networks

Evaluating and choosing from a set of available actions is termed as decision making. The process of decision making becomes complex when problems are uncertain, dynamic, distributed, and heterogeneous in nature. This section is concerned with an approach for decision making under uncertainty: decision networks as an extension of probabilistic reasoning of Bayesian networks. Decision networks incorporate and extend Bayesian networks by both actions under consideration and values of desirability of the resulting outcomes.

Decision networks are useful in solving decision problems where the goal is to find a set of decisions that maximize the value of the expected utility. They are used for visualizing the probabilistic dependencies among the variables in the decision model.

The decision network as a whole depicts a decision or planning problem that a "decision maker" (a person or an agent) faces. Decision networks combine Bayesian networks [16] with additional nodes for action and utilities. It is a compact representation emphasizing features of decision problems. The decision network representation combines the two components of knowledge about beliefs and actions.

Once the decision network is constructed, it may be solved, after which a model-iteration and what-if analysis can be performed [68]. In its more general form, a decision network represents information about the current state of an agent, the possible actions associated with it, what state will be reached as a result of that action and the utility value of that state [79].

To complement the previously discussed reconfiguration layer, a layer has to be provided which adds autonomous and intelligent agents that help determine component failures, tracking performance and analyzing system events that hold the potential to degrade system performance and reliability. This added functionality using decision networks will assist human operators in determining the "silent death" of components or agents, detecting the physical system conditions like open circuit, short ciruit, and overload, and further improves the autonomous operation capabilities of the shipboard power system.

The following discusses important components of decision networks and gives a simple example before designing a situational awareness component for the energy management system.

6.1.1 Utilities

When deciding upon an action, we need to consider our preferences between the possible outcomes for the available actions. Utility theory [54] provides a way to represent and reason with preferences. A utility function determines preferences, reflecting the "desirability" of the outcomes, by mapping each preference to real numbers. Such a mapping allows combining utility theory with probability theory. It allows calculating which action A is expected to deliver the most value (or "expected utility" EU) given any available evidence E [54]:

$$EU(A \mid E) = \sum_i P(O_i \mid E, A) \quad U(O_i \mid A) \tag{6.1}$$

where

- E is the evidence available,
- A is an action which is non-deterministic and is associated with possible outcome states O_i,
- $U(O_i \mid A)$ is the utility value of each of the outcome states for the given action A,
- $P(O_i \mid E, A)$ is the conditional probability distribution over the possible outcome states, for the given evidence E and action A.

The ability of computing the expected utility combined with the *principle of maximum expected utility* [79] allows designing rational decision makers (or agents): a rational decision maker acts to maximize the expected utility value.

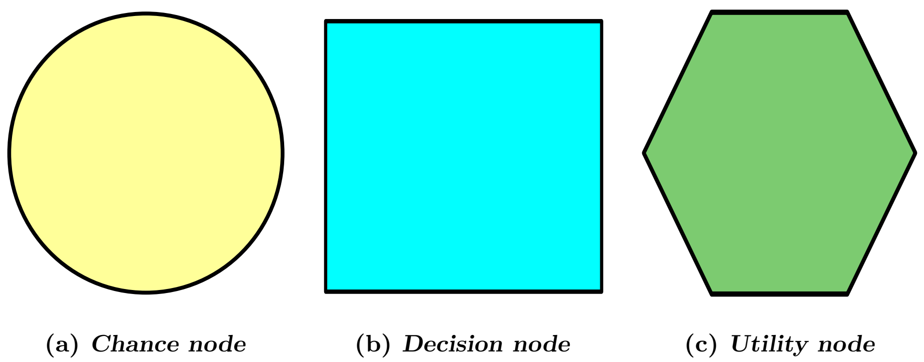
6.1.2 Network Structure

As stated earlier, decision networks extend Bayesian networks by two additional node types. The three types of nodes as shown in Fig. 6.1 are:

- **Chance nodes**, shown as circles, represent random variables as known from Bayesian networks. Each chance node has associated with it a conditional distribution that is indexed by the states of the parent nodes or prior probabilities.
- **Decision nodes**, shown as rectangles, represent the decision being made at a particular point in time. They contain a set of possible choices for a decision.
- **Utility nodes**, shown as hexagon, represent the agent's utility function. They are also called value nodes. They contain a function to calculate the expected utility on the basis of parent chance and decisions nodes and measure how desirable the result can be. Each utility node has an associated utility table.

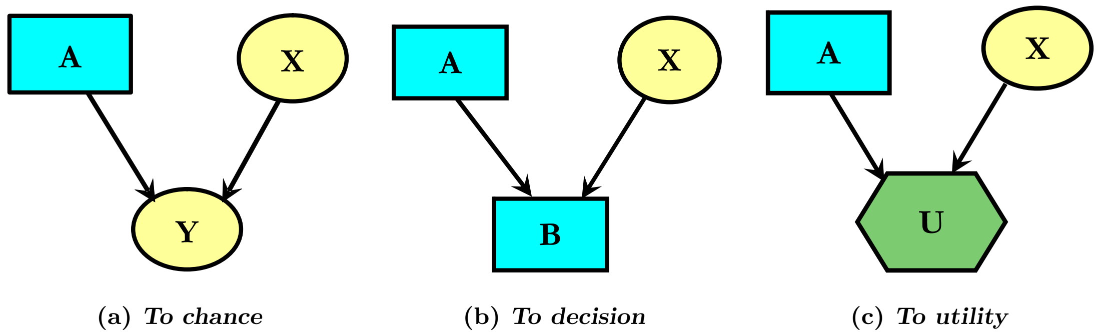
Decision networks form a directed acyclic graph with the following structural properties:

- There is a directed path comprising all decision nodes; and
- The utility nodes have no children.



(a) ***Chance node*** (b) ***Decision node*** (c) ***Utility node***

Figure 6.1: *Decision Network Nodes*



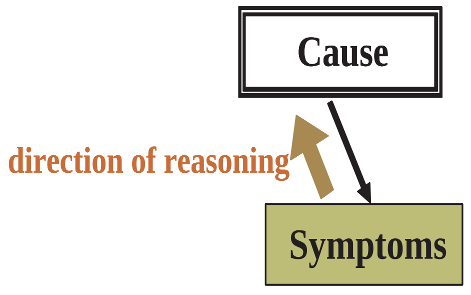
(a) ***To chance*** (b) ***To decision*** (c) ***To utility***

Figure 6.2: *Decision Network Links*

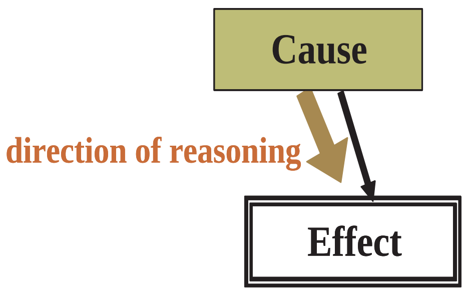
6.1.3 Network Links

The meaning of links in a decision network depends on the link's destination. A link pointing into a chance node signifies *relevance*. For example, nodes A and X of Fig. 6.2(a) point to node Y and therefore imply that the probability of Y depends on that of A and X. Links towards a decision node have a special meaning and are called *informational* links. They indicate what will be known at the time of decision making. In other words, the decision maker will know the values of all the nodes which have links into that decision node but will not know the values of any other nodes. Fig. 6.2(b) depicts such a situation: A and X are known at the time decision B is made. Finally, links into utility nodes signify *functional dependence*. As shown in Fig. 6.2(c), the utility is calculated by some function f on A and X: $U = f(A, X)$.

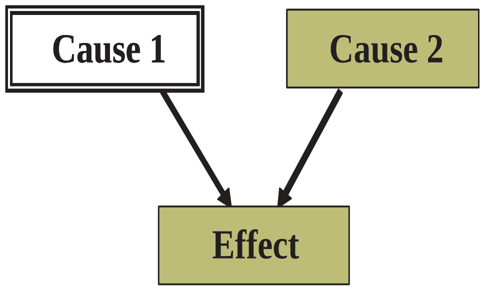
If there are a number of decision nodes, possibly corresponding to decisions made at different times, then solving the network will find a decision function for each of them to form a set known as "policy." This policy corresponds to a full conditional plan and specifies what



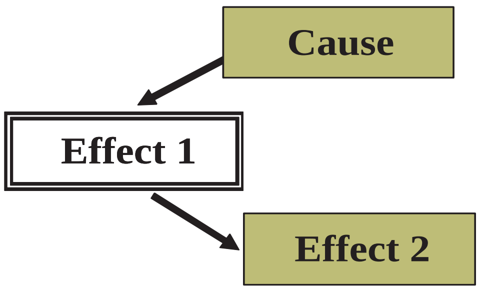
(a) *Diagnostic*



(b) *Predictive*



(c) *Intercausal*



(d) *Combined*

Figure 6.3: *Decision Network - Reasoning*

to do in each possible event based on the available information.

6.1.4 Reasoning

In Bayesian networks, when observing a value of a variable (the evidence), the belief in other variables' values can be updated. This process is known as probability propagation or belief updating and is performed via a flow of information through the network. An advantage of Bayesian networks is based on their full representation of probability distributions over variables. Therefore, information evaluation (or reasoning) can be performed not only in the direction of arcs but also in any other direction. As a result of this, four types of reasoning are possible: diagnostic – from symptoms to cause (Fig. 6.3(a)), predictive – from evidence about causes to effects (Fig. 6.3(b)), intercausal – reasoning about mutual causes of a common effect (Fig. 6.3(c)), and combined – using the other three types combined to query the network based on evidence found (Fig. 6.3(d)). For more details regarding the same refer to [54].

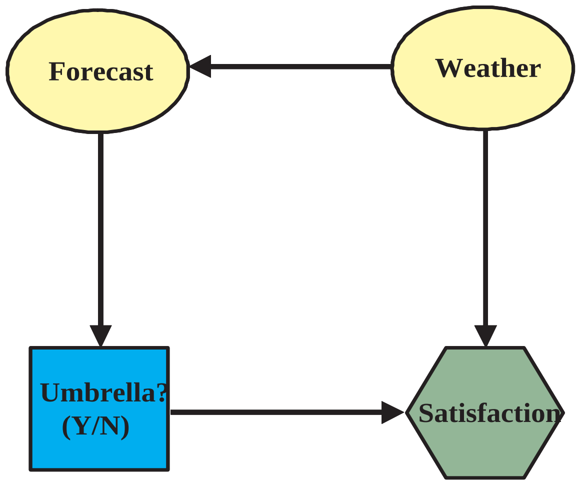


Figure 6.4: *Umbrella Example*

Table 6.1: *Conditional probability table for Weather Forecast*

Weather	**Forecast**		
	Sunny	*Cloudy*	*Rainy*
No_Rain	0.70	0.20	0.10
Rain	0.15	0.25	0.60

6.1.5 "Umbrella" Example Decision Network

An decision problem from [69] known as "Umbrella" as shown in Fig. 6.4 is discussed in the following. The network has 2 chance nodes representing the weather forecast in the morning (sunny, cloudy or rainy), and whether or not it actually rains during the day (rain or norain). The probability values of weather forecast and weather are given in the Tables 6.1 and 6.2 respectively. Further, decision node of whether or not to take an umbrella, and a utility node that measures the decision maker's level of satisfaction complete the network. There is a link from Weather to Forecast capturing the believed correlation between the two (perhaps based on previous observations).

A link exists from Forecast to Umbrella indicating that the forecast is known to the decision maker when he makes the decision. But there is no link from Weather to Umbrella because if he knew for certain about the nature of the weather then it was an easy decision to make whether or not to take the umbrella.

There are links from Weather and Umbrella to Satisfaction, capturing the idea that he is the happiest when it is sunny and he does not take an umbrella (utility = 100), happy when

Table 6.2: *Prior probability table for Weather*

Weather	
No_Rain	*Rain*
0.70	0.30

Table 6.3: *Utility values for Umbrella Example*

Weather	**Umbrella?**	**Utility**
No_Rain	*No*	100
No_Rain	*Yes*	20
Rain	*No*	0
Rain	*Yes*	70

it is raining and he does take an umbrella (utility = 70) as given in Table 6.3. He dislikes to carry an umbrella when it is sunny (utility = 20), but is most unhappy when it rains and he is without one (utility = 0).

The values of the expected outcomes are given in the Table 6.4 and is explained as follows: Prior to any information being available, deciding to take the umbrella results in an expected value of 35, while leaving it at home evaluates to 70. Obviously, the best choice given the available information is to leave the umbrella at home. If the decision maker gets to know that the weather forecast is sunny, then the expected values of utility corresponding to each decision choice changes. The best decision then is to still leave the umbrella at home, but the expected utility has increased to 91.59, because the extra known information indicates it is now more likely that the umbrella will not be required. Say for instance the forecast was cloudy. Still the best decision is to leave the umbrella at home, but the expected utility has decreased to 65.12, because of the increased chance of rain. For "rainy," the best decision changes to "take the umbrella," and the expected utility of that decision is 56.

Table 6.4: *Table of Outcomes (Utility values)*

	Decision	
Forecast	*Take*	*Leave*
Unknown	35.00	70.00
Sunny	24.21	91.59
Cloudy	37.44	65.12
Rainy	56.00	28.00

6.1.6 Advantages

Decision networks are mathematically precise and have been used for more than twenty years as an aid in formulating decision analysis problems. The major advantage of the decision network is an unambiguous and compact representation of probabilistic and informational dependencies. Also, introducing new factors does not contribute to exponential growth of information as each additional factor to be considered requires only a node and an arc. They provide a means to compare alternatives and also serve as a valuable communication tool between the decision maker and the analyst.

6.1.7 Implementation Tool: Netica

The Netica APIs [69] are family of comprehensive toolkits for working with Bayesian belief networks and decision networks. They are used to build, modify, transform, perform probabilistic inference, and store networks. The inference engine is used to answer queries or find optimal solutions. Netica-J offers the complete Netica API in Java. Its object-oriented design and the possibility to access all of Netica's capabilities were the motivation for its incorporation in the envisioned energy management system using agents written in Java.

6.2 Situational Awareness Component for the Energy Management System

While the reconfiguration layer based on the maximum flow algorithm was specifically chosen to allow online modifications of its graph structure, e.g., addition or deletion of nodes and edges, it still requires a component to identify the changes due to failures occuring in the system. Once these changes have been identified, the reconfiguration of the agent-based network of problem solvers can occur. Otherwise, the maximum flow agents may represent an outdated solution or not converge to a new globally consistent solution for the energy flow.

The situational awareness agents supervise the operation of the reconfiguration and implementation agents by observing the following:

1. The guarded statements, which serve as rules for the reconfiguration agents, are used to derive input information concerning the logical operation of the energy management

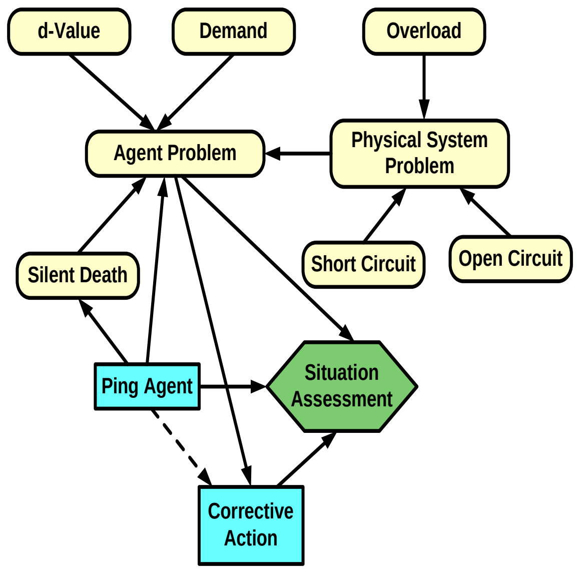


Figure 6.5: *Decision Network for agents in the situational awareness layer*

system (see Fig. 6.5 for a the graphical representation of the decision network and the corresponding nodes):

- **Chance node 1 (d-Value):** The flow of energy allows determine the range of possible distance values of neighboring agents. For example, the d-value of agent i when linked to agent j by an incoming and supplying edge, has to obey $d(i) = d(j) + 1$ or $d(i) = n$.
- **Chance node 2 (Demand):** Any agent has to drive its demand to zero (inflow has to equal outflow).
- **Chance node 3 (Silent Death):** Indicates loss of communication between reconfiguration agent and the blackboard.
- **Chance node 4 (Agent Problem):** Combines the causal chance nodes 1-3 and 8 (discussed below) to calculate the conditional probability of an apparent agent problem.

2. Measurements within the physical layer are communicated by the implementation agent to the situational awareness layer to identify physical system problems:

   - **Chance node 5 (Overload):** Indicates that the corresponding (physical) device is currently providing/demanding more power when anticipated.
   - **Chance node 6 (Short Circuit):** Indicates that the corresponding (physical)

device is actively limiting the power flow due to a short circuit.

- **Chance node 7 (Open Circuit):** Indicates that the corresponding (physical) device lost its power supply.
- **Chance node 8 (Physical System Problem):** Summarizes the state of the physical system.

The decision network is envisioned to help determine the necessity and value of monitoring the individual agent's communication capabilities more closely and to provide suggestions for corrective actions:

- **Decision node 1 (Ping Agent):** Decision (True/False) to trigger a communication test that provides evidence for either an operational or failed agent communication system.
- **Decision node 2 (Corrective Action):** Suggests taking corrective action. The possible actions are: no action and deactivate agent.

The remaining node to be defined is the utility node:

- **Utility node (Situation Assessment):** Allows computation of the respective values of making decisions.

The specific values of apriory probabilities for chance nodes *d-value*, and *Demand* are through experience, while that of chance nodes *Overload*, *Short Circuit*, *Open Circuit*, and *Silent Death* are based on assumptions. The aprior probability values along with conditional probabilities of chance nodes, and the utility values can be found in Tables 6.5, 6.6, 6.7, 6.8, and 6.9 respectively.

Table 6.5: *Prior probabilities*

d-value	$d(i) = n$	$d(i) = d(j) + 1$	*wrong*
%	45	45	10

demand	*zero*	*notzero*
%	90	10

	FALSE	*TRUE*
Overload	90	10
Short Circuit	80	20
Open Circuit	80	20

Table 6.6: *Conditional probabilities for Physical System Problem*

Overload	**Short Circuit**	**Open Circuit**	*FALSE*	*TRUE*
FALSE	*FALSE*	*FALSE*	98	2
FALSE	*FALSE*	*TRUE*	30	70
FALSE	*TRUE*	*FALSE*	20	80
FALSE	*TRUE*	*TRUE*	50	50
TRUE	*FALSE*	*FALSE*	20	80
TRUE	*FALSE*	*TRUE*	50	50
TRUE	*TRUE*	*FALSE*	20	80
TRUE	*TRUE*	*TRUE*	100	0

Table 6.7: *Conditional probabilities for Silent Death*

Ping Agent	*FALSE*	*TRUE*
FALSE	80	20
TRUE	98	2

Sample data of observed values of demand and d-value during reconfiguration is shown graphically in Fig. 6.6, and 6.7 respectively.

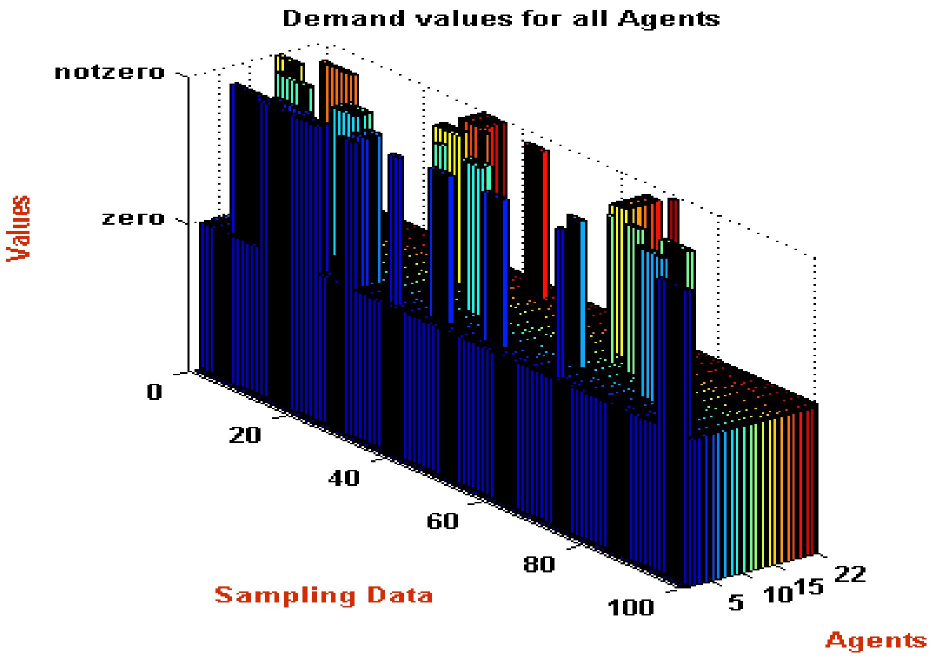


Figure 6.6: *Observation of demand for all the agents*

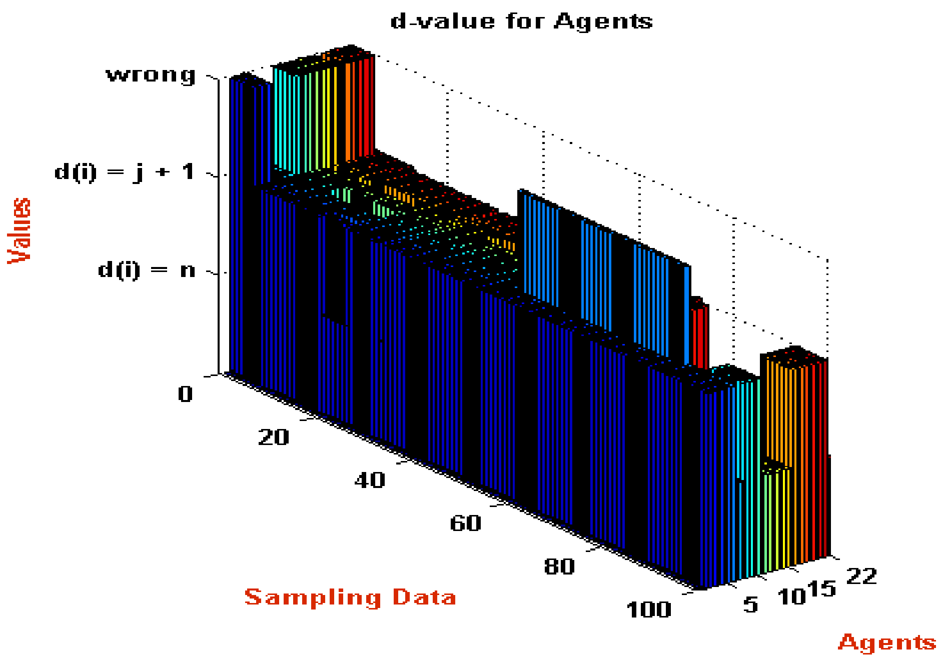


Figure 6.7: *Observation of d-value for all the agents*

6.3 Conclusion

To summarize, this chapter presented the situational awareness component which is an addition and enhancement to the multi-layered agent architecture of the energy management system. Improvements are made possible by providing this awareness component that identifies the operational status and performance of the shipboard system. It also suggests corrective measures to deal with critical situations. The benefit of decision networks allows operators to first gain trust in the correctly functioning situational awareness agents before finally delegating the final decision making to the agents as well. The component is implemented by combining agent-based techniques and decision networks and consequently enhances the envisioned modernized control architecture for shipboard power systems.

Table 6.8: *Conditional probabilities for Agent Problem*

d-value	**demand**	**PA**	**SD**	**PP**	*FALSE*	*TRUE*
a	zero	*FALSE*	*FALSE*	*FALSE*	99	1
a	zero	*FALSE*	*FALSE*	*TRUE*	30	70
a	zero	*FALSE*	*TRUE*	*FALSE*	30	70
a	zero	*FALSE*	*TRUE*	*TRUE*	15	85
a	zero	*TRUE*	*FALSE*	*FALSE*	100	0
a	zero	*TRUE*	*FALSE*	*TRUE*	8	92
a	zero	*TRUE*	*TRUE*	*FALSE*	8	92
a	zero	*TRUE*	*TRUE*	*TRUE*	1	99
a	notzero	*FALSE*	*FALSE*	*FALSE*	30	70
a	notzero	*FALSE*	*FALSE*	*TRUE*	10	90
a	notzero	*FALSE*	*TRUE*	*FALSE*	10	90
a	notzero	*FALSE*	*TRUE*	*TRUE*	10	90
a	notzero	*TRUE*	*FALSE*	*FALSE*	70	30
a	notzero	*TRUE*	*FALSE*	*TRUE*	10	90
a	notzero	*TRUE*	*TRUE*	*FALSE*	10	90
a	notzero	*TRUE*	*TRUE*	*TRUE*	4	96
b	zero	*FALSE*	*FALSE*	*FALSE*	98	2
b	zero	*FALSE*	*FALSE*	*TRUE*	10	90
b	zero	*FALSE*	*TRUE*	*FALSE*	10	90
b	zero	*FALSE*	*TRUE*	*TRUE*	10	90
b	zero	*TRUE*	*FALSE*	*FALSE*	96	4
b	zero	*TRUE*	*FALSE*	*TRUE*	5	95
b	zero	*TRUE*	*TRUE*	*FALSE*	5	95
b	zero	*TRUE*	*TRUE*	*TRUE*	5	95
b	notzero	*FALSE*	*FALSE*	*FALSE*	95	5
b	notzero	*FALSE*	*FALSE*	*TRUE*	10	90
b	notzero	*FALSE*	*TRUE*	*FALSE*	10	90
b	notzero	*FALSE*	*TRUE*	*TRUE*	10	90
b	notzero	*TRUE*	*FALSE*	*FALSE*	96	4
b	notzero	*TRUE*	*FALSE*	*TRUE*	5	95
b	notzero	*TRUE*	*TRUE*	*FALSE*	5	95
b	notzero	*TRUE*	*TRUE*	*TRUE*	5	95
c	zero	*FALSE*	*FALSE*	*FALSE*	30	70
c	zero	*FALSE*	*FALSE*	*TRUE*	5	95
c	zero	*FALSE*	*TRUE*	*FALSE*	5	95
c	zero	*FALSE*	*TRUE*	*TRUE*	5	95
c	zero	*TRUE*	*FALSE*	*FALSE*	5	95
c	zero	*TRUE*	*FALSE*	*TRUE*	5	95
c	zero	*TRUE*	*TRUE*	*FALSE*	5	95
c	zero	*TRUE*	*TRUE*	*TRUE*	5	95
c	notzero	*FALSE*	*FALSE*	*FALSE*	10	90
c	notzero	*FALSE*	*FALSE*	*TRUE*	5	95
c	notzero	*FALSE*	*TRUE*	*FALSE*	5	95
c	notzero	*FALSE*	*TRUE*	*TRUE*	5	95
c	notzero	*TRUE*	*FALSE*	*FALSE*	10	90
c	notzero	*TRUE*	*FALSE*	*TRUE*	5	95
c	notzero	*TRUE*	*TRUE*	*FALSE*	5	95
c	notzero	*TRUE*	*TRUE*	*TRUE*	0	100

d-value: a — $d(i) = n$, b — $d(i) = d(j) + 1$, c — wrong

Table 6.9: *Utility values*

Ping Agent	**Agent Problem**	**Corrective Action**	**Utility**
$FALSE$	$FALSE$	$FALSE$	1
$FALSE$	$FALSE$	$TRUE$	0.01
$FALSE$	$TRUE$	$FALSE$	0.1
$FALSE$	$TRUE$	$TRUE$	0.4
$TRUE$	$FALSE$	$FALSE$	0.7
$TRUE$	$FALSE$	$TRUE$	0.02
$TRUE$	$TRUE$	$FALSE$	0.1
$TRUE$	$TRUE$	$TRUE$	0.6

Chapter

7

Multiagent Architecture

Through the many interesting features, agents provide autonomy and heterogeneity, restrict access to resources and guarantee specialized integrity requirements, and model organizations and nonterminating tasks in them. Agents are best applied to achieving flexibility and agility, improving efficiency of processes, and helping manage complexity. Just by using the terms “agents” or “multiagents” to describe a system does not mitigate the issues concerning agent-based systems. Specific solutions are needed to address the challenges of complex systems. The following will combine the above discussed components to form a consistent and cooperating multiagent based energy management system. The graphical Unified Modeling Language will be the tool of choice to specify the system components and allow taking advantage of UML’s graphical expressiveness while avoiding the complexities of a text based specification/programming language.

7.1 Technical Approach - Prototyping Model

As an approach towards designing a multiagent architecture for the shipboard power system, a combination of traditional software engineering approaches [75] with considerations from the agent-oriented field is followed here. The research adopts the “prototyping model” to help build and evaluate the multiagent system. The multiagent architecture follows a layered

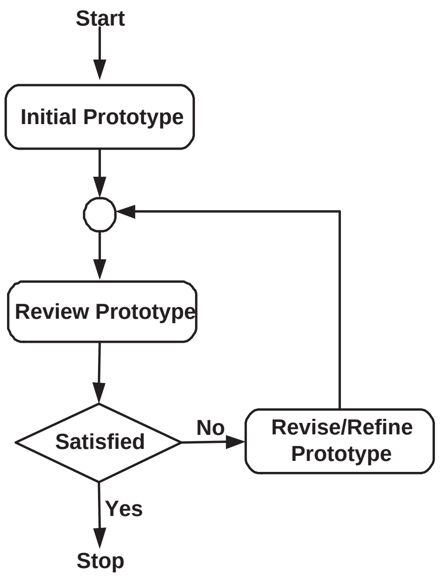


Figure 7.1: *Prototyping Process*

approach in solving the energy management problem using the blackboard system as an interaction protocol. UML statecharts are used to specify agents and the object oriented language Java is used for implementing agents.

Prototyping [10] is a valuable technique to help software engineers explore the design space while gaining insight and a feel for the dynamics of the system. It also allows engineers learn more about the relationships among design features and the desired computational behavior. It promotes early experimentation with alternative design choices allowing engineers to pursue different solutions without efficiency concerns.

The research follows a generic prototyping process as shown in Fig. 7.1. A series of steps are performed iteratively until the objectives are attained.

7.2 Architecture Overview

7.2.1 Layered Architecture

Layered architecture is opted due to two major reasons. Firstly, it provides a functional separation which makes it organized and easily modifiable. Secondly, it supports independence between the layers which allows us to replace, add or remove layers without affecting the others. To maximize the level of independence, the layered approach uses a blackboard as an interaction protocol as discussed next.

7.2.2 Blackboard for Communication Infrastructure

Blackboard systems are an excellent paradigm for processing and fusing information and developing situation assessments. This paradigm is particularly appropriate for integrating diverse knowledge and performing multi-level reasoning. The approach for this research adopts the blackboard system as a shared memory structure, facilitating inter-layer interaction. The blackboard system is a convenient method for the various layers to post their solutions and status, and fetch the required results.

7.2.3 Architecture for the Energy Management System

The overall layout of the shipboard energy management system is based on grouping the different system parts into layers. The layers form a logical architecture according to their functionality (see Fig. 7.2).

The lowest layer represents the mathematical model for the physical system. The shipboard model simulates the various components of the ship and their interactions. It is modeled using MATLAB/Simulink. The components have local controls, e.g., proportional-integral controllers and their reference and parameter values, which are modified and adjusted by the decision implementation agents.

The implementation layer represents the layer closest to the physical power system. This layer contains the decision implementation agents that are the point of interface between the individual devices. These agents, which are M-S functions, continuously monitor the blackboard to implement the power flow solution agreed upon by agents in the reconfiguration layer. These agents access locally their respective maximum flow agents to inquire the solution to be implemented.

The decentralized power flow solution using the maximum flow algorithm is part of the reconfiguration layer. The agents in this layer reach a globally feasible solution by executing their programs asynchronously and locally but can communicate status using the blackboard. Here each agent follows a set of seven rules as was explained in detail in Chapter 5. These rules are based on the self-stabilizing maximum flow algorithm from Graph theory. The agents in the reconfiguration layer are implemented using the Java programming language [45] where

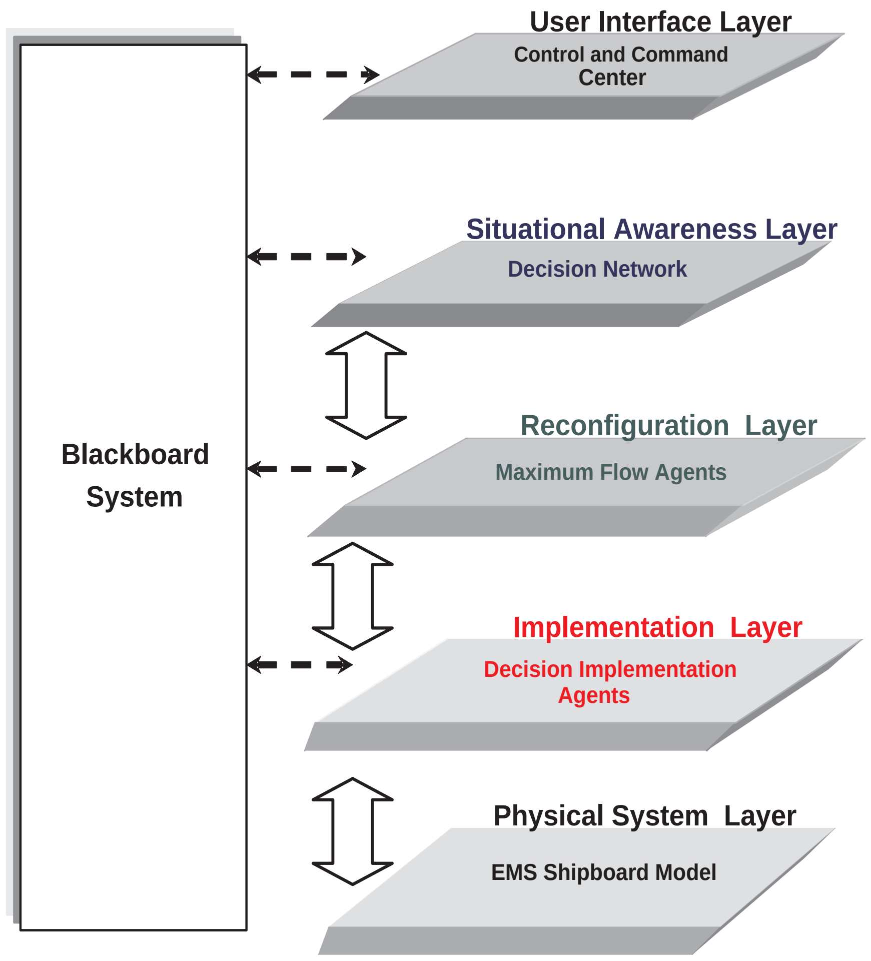


Figure 7.2: *Layered Architecture for the Energy Management System*

every agent is implemented by an asynchronously executing thread.

The agents of the situational awareness layer are built on top of the reconfiguration and implementation layers to provide system operators on board with updated information concerning the operational status of the system. These agents help the operators assess the situation on hand and make appropriate decisions based on the possible course of action suggested by the agents in crisis situations. They supervise the actions by the agents in the reconfiguration layer and implementation layer by observing certain variables, the details have been given in the previous chapter.

The topmost layer represents the human-machine interface for the command and control center. This layer displays important system information and also allows the system operator to communicate with system components, e.g., request for desired changes in propulsion power. It can also be used to simulate disturbances, e.g., loss of generation.

The next sections discuss the specification process adopted in designing the agents for the energy management system.

7.3 Agent Specification

7.3.1 UML Statecharts

The specification of multiagent systems needs to be meaningful and at the same time explanatory so that its possible to model not only the dynamic behavior of a single agent, but also the collaboration among several agents and the changes caused by external events from the environment. For this, UML statecharts [70] seem to provide an adequate means. They provide a way to define a mapping between the internal state of an agent and its operations in the world. The use of UML as a specification and modeling language is already widely accepted. It is possible to get a synoptic overview of the functionality of the complete multiagent system by means of only one type of diagram, namely statecharts [3] [67].

The following section briefly summarizes those parts of the UML statechart formalism [67] that are employed for the design of agents for the energy management system.

States and Transitions

In the UML formalism, a *state* is considered an interval in the life of a system or an agent during which a certain condition holds or an activity is performed. For example, an agent may remain in a state while it waits for some external event to occur. In a statechart a state is represented as a box with corners rounded.

External events cause the state of a system to change. Such a change of state is called a *transition*. A *transition string* is used to specify the behavior of a transition. A transition string is a tuple $T = (e, c, a) \in (E \times C \times A)$, where E is the set of (external) events, C is the set of boolean expressions over a domain and A is the set of possible actions that can be taken. A transition then can be defined as a tuple $t = (s_1, T, s_2)$, where s_1, s_2 denote arbitrary states and T is a transition string of the form (e, c, a).The (informal) semantics of t is "if the system is in state s_1 and event e occurs and the condition c holds, then the system executes

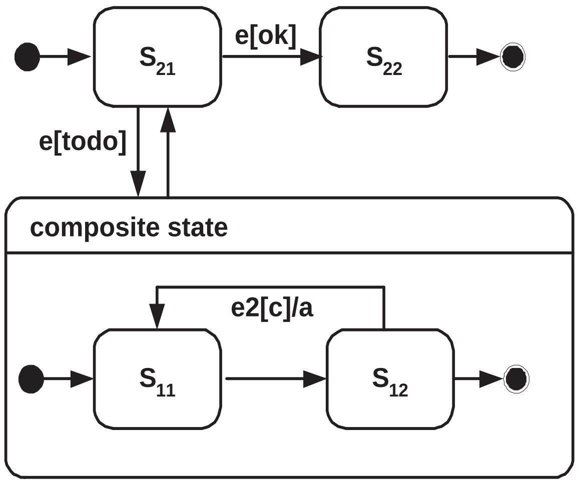


Figure 7.3: *Illustration of a Statechart*

action a and changes to state s_2". In a statechart diagram a transition is shown as a directed edge from s_1 to s_2, which is labeled with T in the form $e[c] \setminus a$.

There are different types of states in the UML statechart formalism. As a statechart is hierarchical in nature, the UML distinguishes between three major classes of states two of which are shown in Fig. 7.3.

Simple states are atomic in the sense that they do not possess any internal structure. Nevertheless it is possible to assign some kind of behavior to a simple state by defining internal transitions.

Composite states are states that can be further decomposed. They contain internal submachines which describe the activity associated with the composite state.

Concurrent states are special types of composite states. A concurrent state contains two or more composite substates, which are called regions. If an agent is in a concurrent state, it is in all regions simultaneously. Thus concurrent states are used to model concurrent activities in a system or an agent. An example is shown in Fig. 7.4.

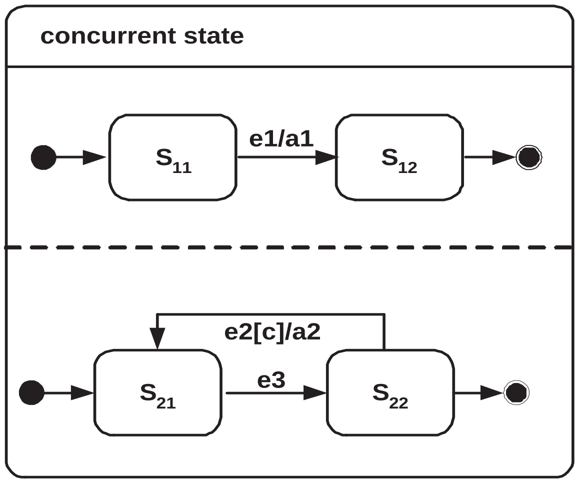


Figure 7.4: *Illustration of a Concurrent State with Two Regions*

7.3.2 Agent Specification using UML statecharts

This section gives the details regarding the specification for the agents in energy management system for the shipboard power system. Every layer of the energy management system is modeled in individual statecharts. The collaboration between the Command and Control Center, the Reconfiguration Layer and Situational Awareness Layer is also modelled. Some details are not shown, which are indicated by the *hidden decomposition icon* o—o.

Specification for the Command and Control Center

The User interface design : The specification for the Command and Control Center is as shown in Fig. 7.5. It is initially in the *init* state and then comes to the *ready* state. Once in the ready state, it remains in this state until it is triggered by an external event (the click of a button) after which it is seen that it can follow any of the three states, *execute*, *reset* and *save*. The execute state is a composite state with two concurrent states, *GUI* state and *maxflow* state. The ready state has two more concurrent states *display* and *quit*. When it enters the Display state it displays the directed graph for the Shipboard Power System, which is explained next. Once it enters the quit state it stops.

Displaying the directed graph : The Display state which corresponds to displaying of the directed graph for the Shipboard Power System is shown in Fig. 7.6. It changes from *init* state to *plotting* state. From the plotting state it evaluates to see if the quit signal is true. If

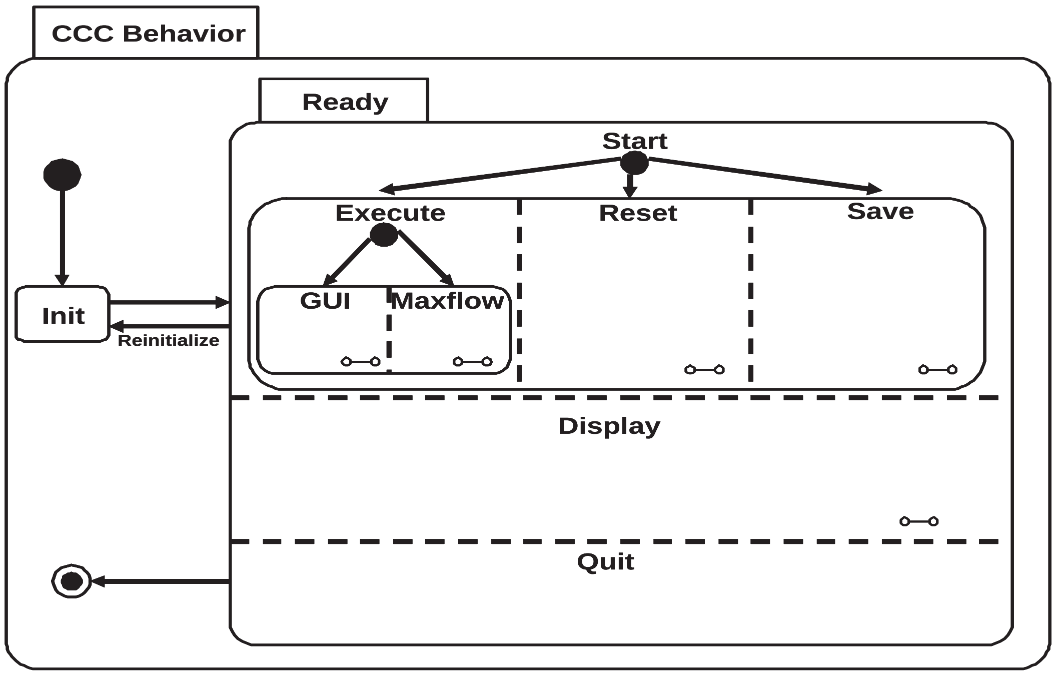


Figure 7.5: *Specification for the Command and Control Center*

the quit signal is true then it stops else it goes into the *repaint* state and again to the plotting state.

Specification for the Reconfiguration Layer

Reconfiguration of agents : The reconfiguration agents' specification is shown in Fig. 7.7. These agents are initially in the *init* state and then move to the state *execute guards* where in the guarded statements corresponding to the self stabilizing maximum flow algorithm is executed. All the agents in the reconfiguration layer follow the same specification (design). After the state "execute guards" if all the guards evaluate to a *false* then the new solution is published to the blackboard. After which it is checked if *end* evaluates to a *true*, if yes it stops, else if it is *false*, it goes to the "execute guards" state again and continues.

Specification for the Situational Awareness Layer

Situational Awareness agents : The Fig. 7.8 corresponds to the specification for the agents in the Situational Awareness layer. The agents in this layer begin execution from their *init* state to *evaluate* state. In the evaluate state the agents are in the process of supervising the

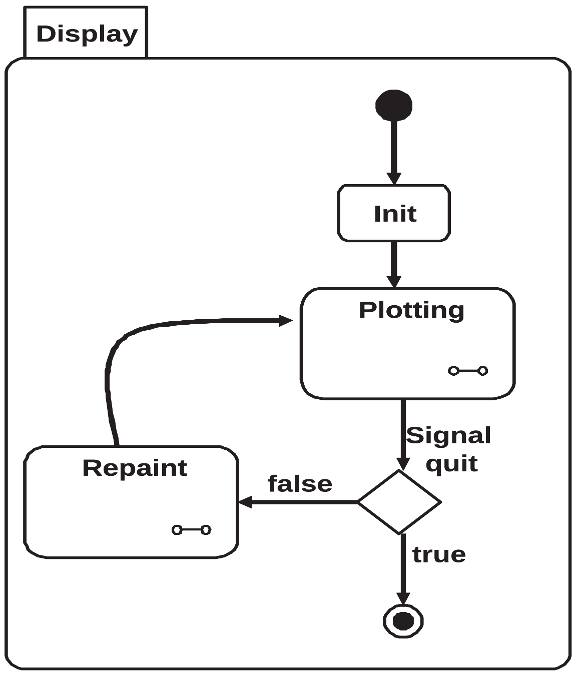


Figure 7.6: *Specification for plotting the digraph of the EMS*

agents in the reconfiguration layer and the implementation layer. From the evaluate state they determine if something is wrong. If it returns a *true* then it displays the appropriate message to the system operator, who would need to take the necessary action as a corrective measure against the problem. And then it evaluates *end* and if end is true then it exits otherwise it again goes back to the evaluate state. From the Fig. 7.8 we see that if "nothing wrong" is detected by these agents, again end is evaluated and if end is true it exits otherwise goes back to the evaluate state.

Multilayer Collaboration

Collaboration between different layers : The Fig. 7.9 shows the overall behavior of the multiagent system (excluding the implementation layer and physical system layer). The agents in the reconfiguration are in the left most concurrent state. The right most concurrent state corresponds to that of the Command and Control center and the center concurrent state corresponds to that of the agents in the situational awareness layer. From the Fig. 7.9 it is seen that the agents in all the three concurrent states are initially in the *init* state. After the init state the agents in the respective layers move into their corresponding states i.e, reconfiguration layer agents — execute guards, situational awareness — evaluate agents

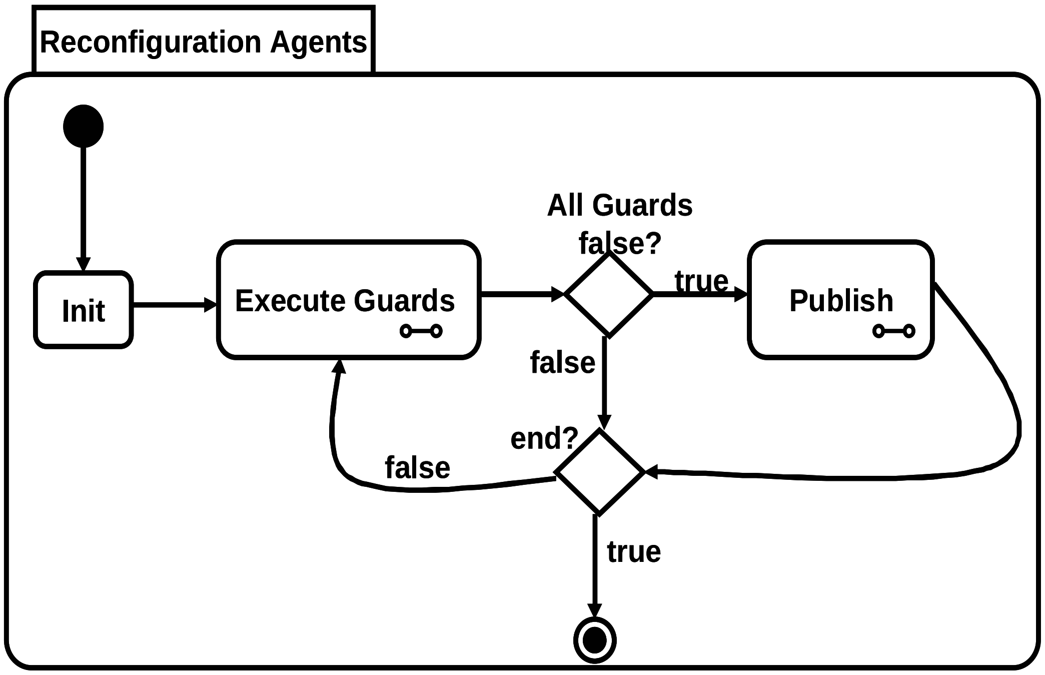


Figure 7.7: *Specification for agents in the Reconfiguration Layer*

and command and control layer — wait for input state and execute state. As and when appropriate the agents in all the layers asynchronously update the blackboard system and continue to operate in this way until end is evaluated to a *true*. When end is evaluated to true, the agents in all the layers stop execution.

The section described the high level specification for the multiagent architecture of the autonomous energy management system. The next section gives implementation details of the respective parts of the multiagent architecture.

7.4 Implementation

7.4.1 Physical System Layer

The energy management system for the electric shipboard power system has been accomplished following the agent-based framework explained in the previous chapter. The Physical system layer implemented in SIMULINK is shown in Fig. 4.1 of Chapter. 4. The average model of the shipboard power system is used to make the simulations run faster than or in real-time. This layer models the various parts of the electric shipboard power system. It consists of the

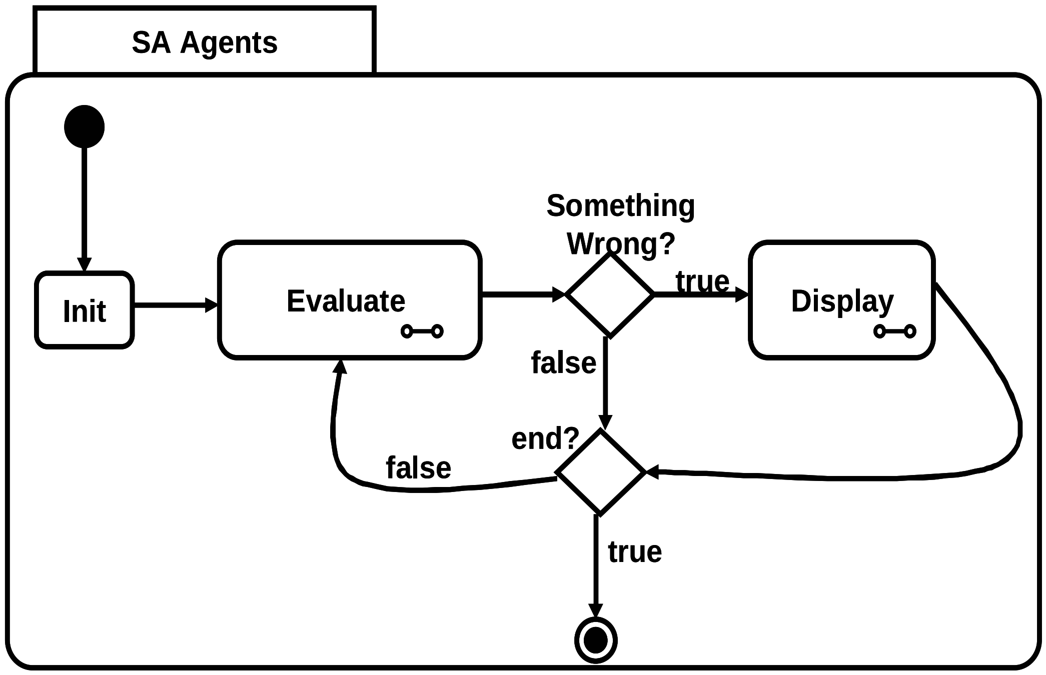


Figure 7.8: *Specification for agents in the Situational Awareness Layer*

generation and propulsion test bed (blocks in green and blue) and the DC distribution test bed. The DC distribution part of the system is divided into three zones as indicated. Each zone is equipped with two loads, one high priority and one low priority load.

7.4.2 Command and Control Center

The User Interface layer is implemented using the Java Swing components (Fig. 7.10). It is designed to help the system operator communicate with the physical system. For example, the operator can change the mode of operation of the ship by switching between modes of 'cruising,' 'traveling,' 'harbor,' and 'fighting.' Depending on each mode, the priorities of the various loads in every zone will change. The operator can also change the supply path of the loads by performing operations on the agents and edges like addition, deletion, and modification (only in the case of edges). The interface is also a means to simulate disturbances inflicted onto the system and allows evaluate the agent-based energy management system using arbitrary system conditions. The Command and Control Center (CCC) is also provided with a message board where important information regarding the system's operational status is displayed.

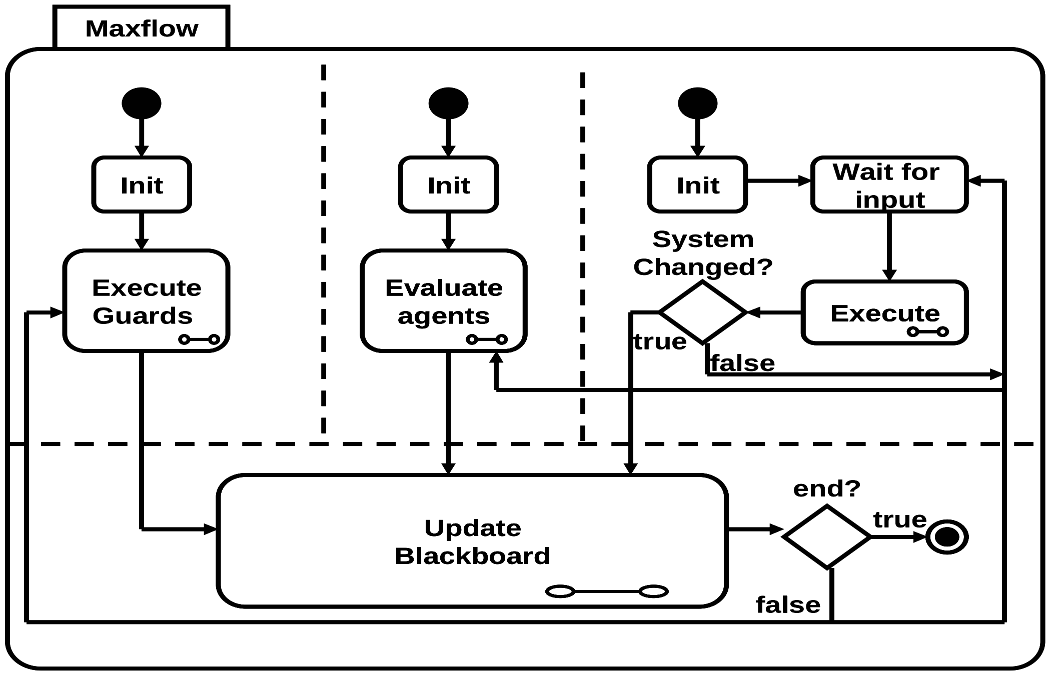


Figure 7.9: *Specification for agents' collaboration in different layers (with blackboard)*

7.4.3 Blackboard

The blackboard was implemented using standard relational database management systems (RDBMS). The blackboard system is accessible to the agents in the energy management system via the JDBC-ODBC connection, using the standard Structured Query Language (SQL). While every agent updates its own section of the blackboard, the other agents are just able to read the values.

7.4.4 Running the Simulation

The simulation is started from the MATLAB environment by starting the simulation of the physical system model. Once the simulation of the physical system has begun, the CCC is displayed. Upon hitting the 'Start' button in the CCC, the agents in the reconfiguration layer begin operating at the specified initial conditions. To display the directed graph of the shipboard power system, the 'Display' button needs to be clicked. This starts a separate thread executing asynchronously. Fig. 7.11 is a snapshot of the digraph of the SPS. The digraph is updated continuously based on the current system status. Once the reconfiguration agents agree on a solution, the implementation agents are notified and adjust the parameters in the physical system layer. These changes are applied to the physical system be either turning off (red) or turning on the loads (light blue).

Simulating external faults

Certain physical system disturbances like open circuit, short circuit, and overload conditions can also be simulated from within the physical system and reported to the system operator (through the CCC) via the implementation layer, blackboard, and situational awareness layer. The interface for disturbance simulations is shown in Fig. 7.12.

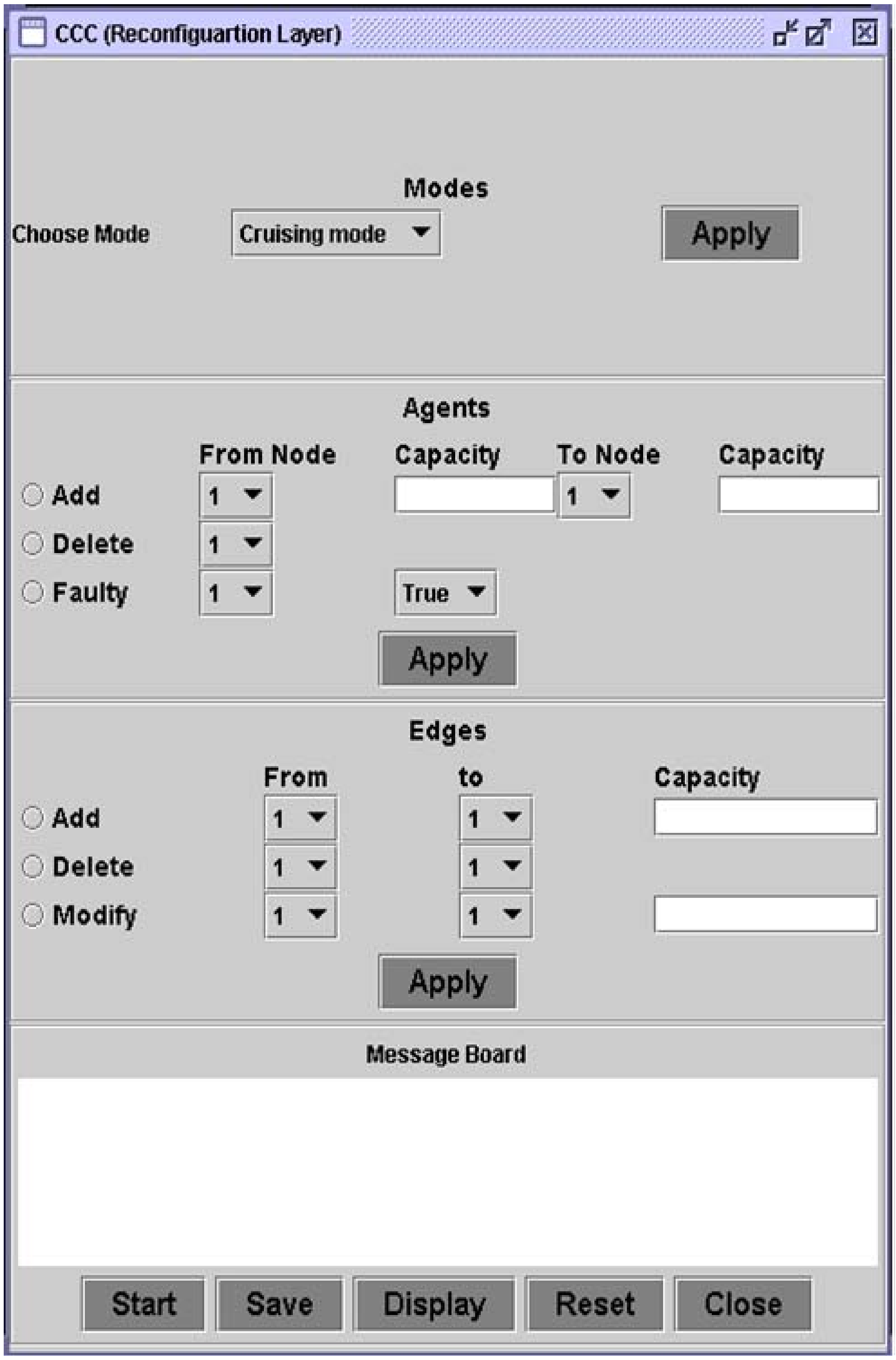
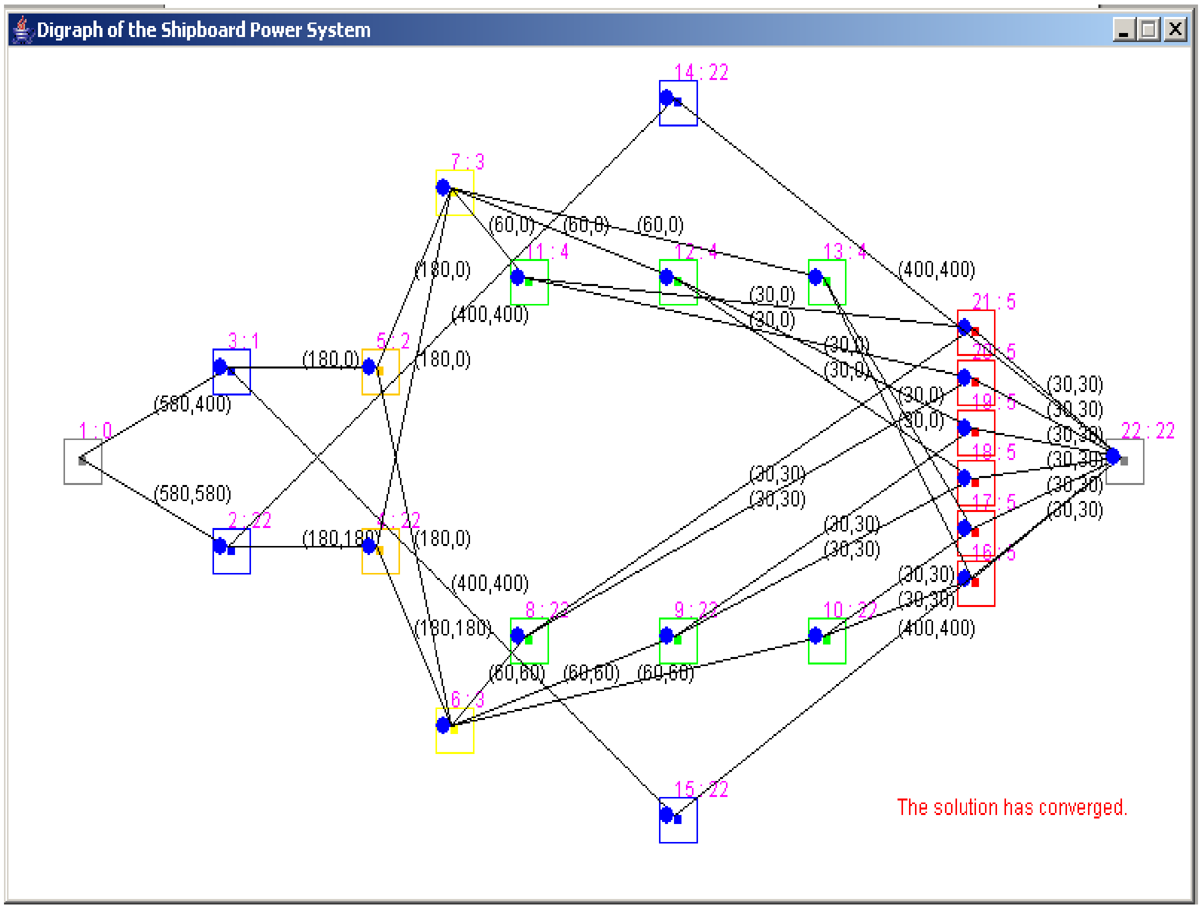


Figure 7.10: *Command and Control Center*



The solution has converged.

Figure 7.11: *Directed Acyclic Graph of the Electric Shipboard Power System*

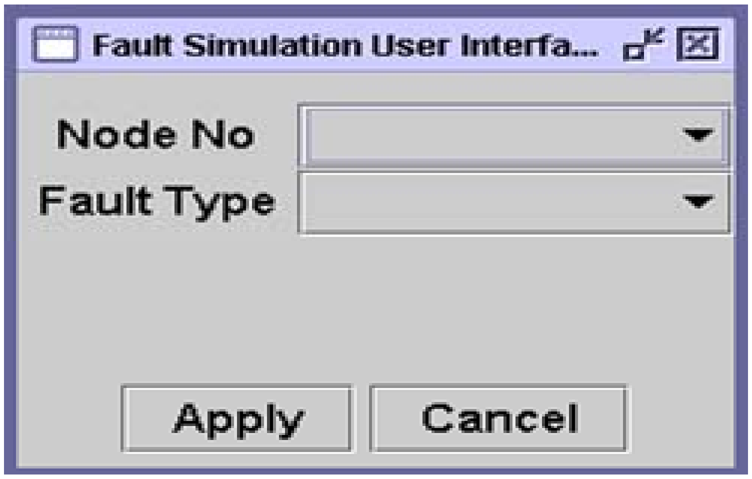


Figure 7.12: *User Interface for Simulating Faults*

Chapter 8

Case Studies and Discussion of Results

The energy management system has been tested for different system conditions, e.g., initial startup, changes in load demand, changes in edge capacity, loss of agents (nodes), and increases in demand beyond generation and power transfer capabilities. Some of the simulated case studies are discussed in the following sections.

8.1 Startup Scenario

The 'startup' scenario tests the system to find configuration solutions for various operating modes when starting from an initially deactivated ship where no power is provided to any load. The chosen modes — full-power, traveling, harbor, cruising, and fighting — and their respective minimum and maximum ranges of the loads to be supplied along with priorities are given in Table 8.4 at the end of this chapter. The following case studies and scenarios will build on these base settings.

Fig. 8.1 shows the number of moves made by the 22 maximum flow agents when configuring for the different operating modes of the shipboard power system: full-power, traveling, harbor, cruising, and fighting. To be able to test convergence characteristics, each agent was started randomly after a small time interval. The plot shows the sum of the agents' moves until the process converges to a global solution. The number of agents actively performing during

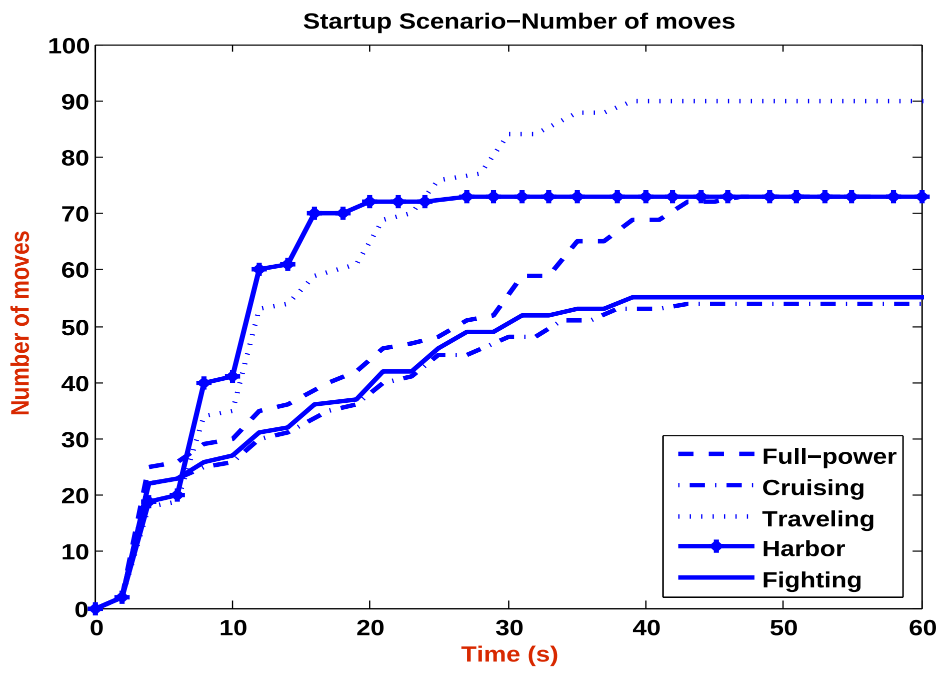


Figure 8.1: *Agents' Moves for the Startup Scenario*

reconfiguration until the solution is reached is depicted in Fig. 8.2. Statistics concerning the number of agents' moves necessary for the 'startup' scenario are given Table 8.1.

Table 8.1: *Maximum, minimum and average moves for the 'startup' scenario*

		Full-power	Cruising	Traveling	Harbor	Fighting
Startup	**Max**	84	64	90	73	80
	Avg	73	57	69	58	58
	Min	66	52	56	43	51

8.2 Rerouting Power

For a chosen operating condition (traveling mode) as shown in Fig. 8.3, a global solution was reached by the agents in the reconfiguration layer and edge $(4, 6)$, which represents the link between the Port PS and Port DC bus, is supplying power to the loads. The loss of this edge, which is analogous to either loss of communication between the agents 4 and 6 or a physical cable damage, was triggered through the CCC. For this condition, the agents in the reconfiguration layer pick up the disturbance and start to negotiate a new globally acceptable solution. Once the agents agree upon a solution a new supply path is found and the loads can be restored. For this scenario, edge $(5, 6)$ as shown in Fig. 8.4 is now supplying power.

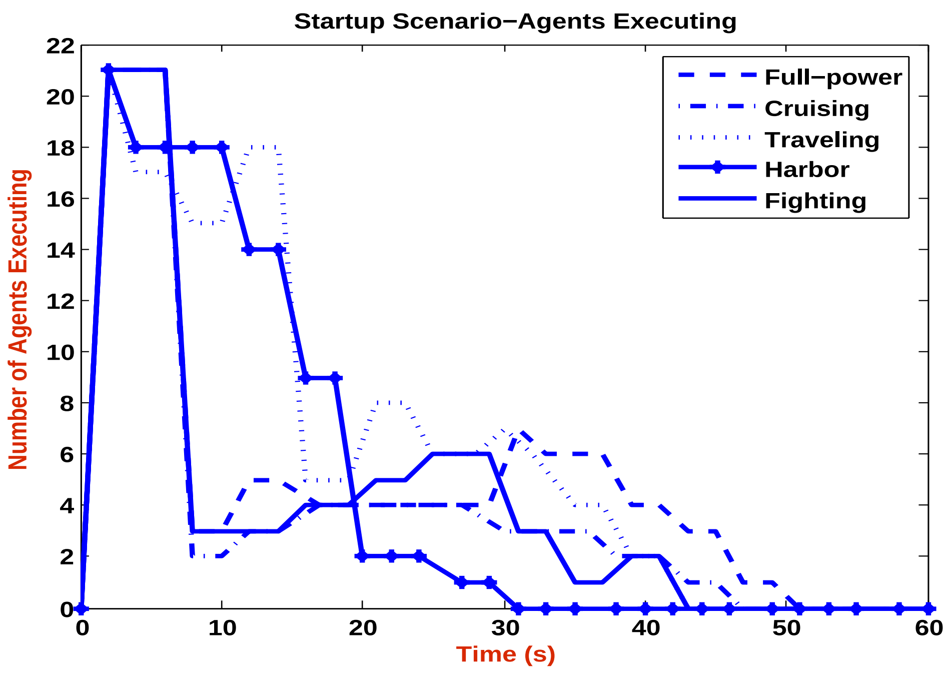


Figure 8.2: *Agents Executing for the Startup Scenario*

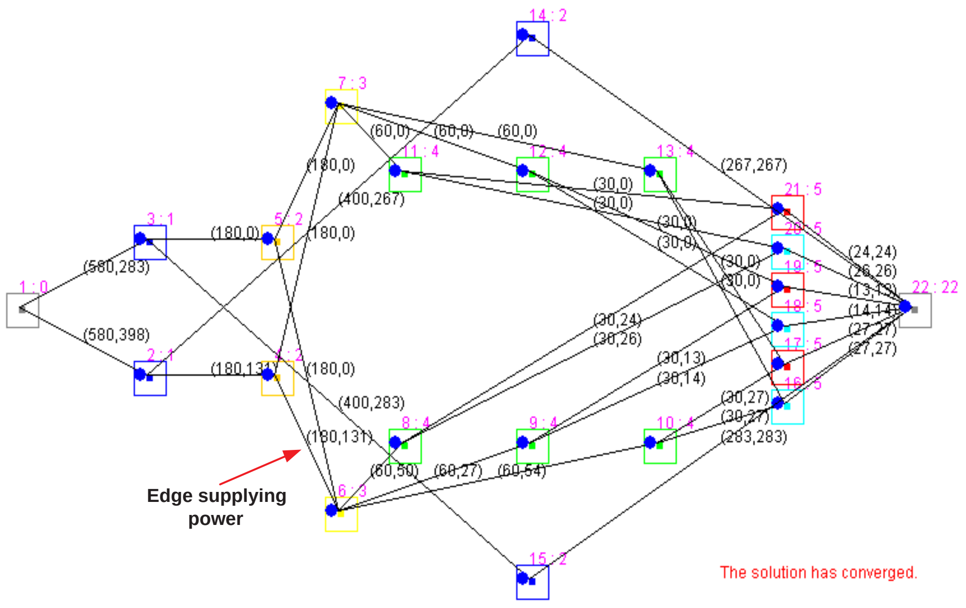


Figure 8.3: *Loads supplied through Edge (4,6)*

8.3 Restoring High Priority Loads

The following simulation scenario starts in cruising mode. In this mode, the loads are initially supplied via the link between the Port PS and Port DC bus, i.e., edge $(4, 6)$. A damage to

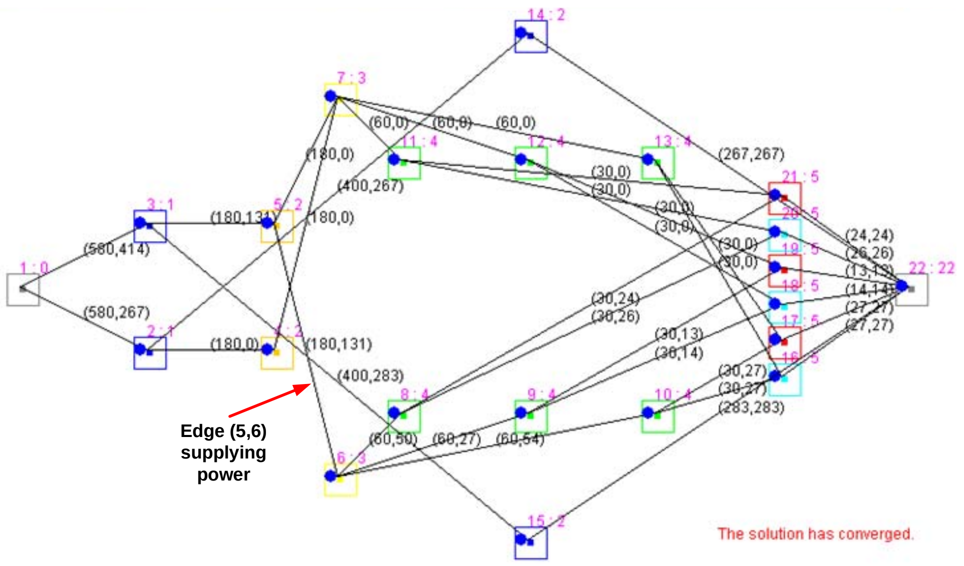


Figure 8.4: *After Disturbance: Loads supplied through Edge (5,6)*

Table 8.2: *Maximum, minimum and average moves for the 'rerouting' scenario*

		Moves
Rerouting	**Max**	55
	Avg	29
	Min	5

agent 6, which represents the Port DC bus, leads to its elimination. The second enforced disturbance is a loss of edge $(4, 7)$, the link between the Port PS and Starboard DC bus. Consequently, power is rerouted through the port-side edge $(5, 7)$ to supply the loads (see Fig. 8.5). Further loss of system capabilities was simulated by reducing the capacity of edge $(5, 7)$ representing the link between the Starboard PS and Starboard DC bus from 18.0 kW to 5.7 kW (180 to 57 units in the digraph). When a solution was agreed upon by the agents in the reconfiguration layer, only high priority loads were restored due to the reduction in distribution capacity. The low priority loads have been disconnected as shown in Fig. 8.6. This change is reflected in the physical system via the implementation layer agents with the loads being turned off.

8.4 Random Operating Conditions

The system was subjected to randomly changing operating conditions mimicing the changes happening onboard a naval ship. The modes were changed after a solution had been found.

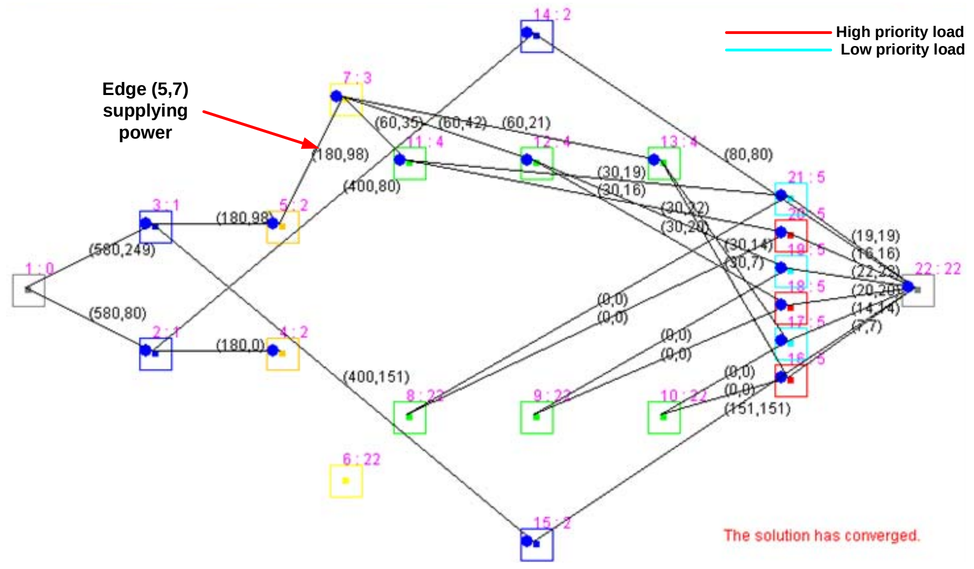


Figure 8.5: *Loads supplied Through Edge (5,7)*

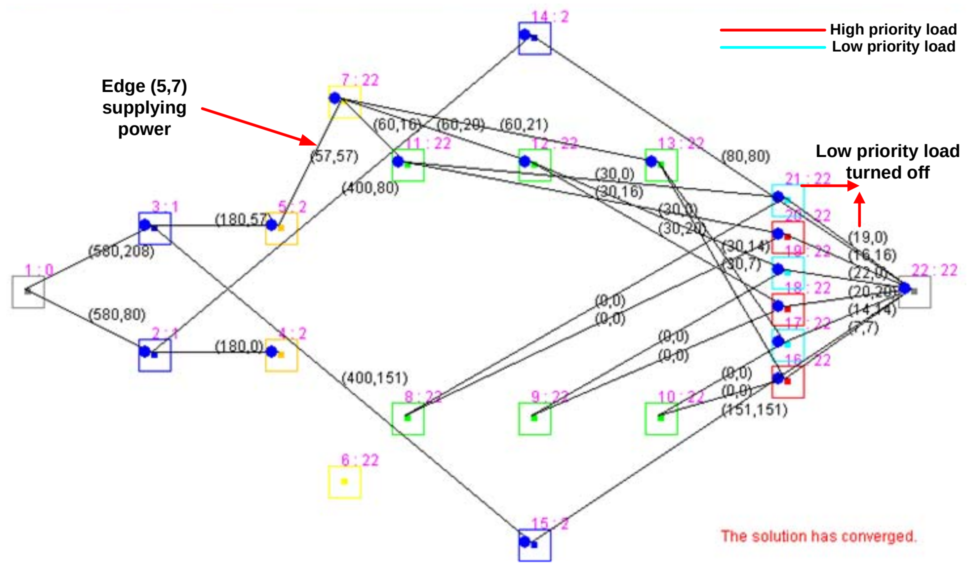

High priority load
Low priority load
Edge (5,7) supplying power
Low priority load turned off
The solution has converged.

Figure 8.6: *Loads supplied Through Edge (5,7) to Restore High Priority Loads*

Fig. 8.7 shows the plot for the following randomly chosen sequence of modes and conditions: full-power, traveling, harbor, fighting, cruising mode, loss of port DC bus (agent 6), loss of supply path between Port PS and Starboard DC bus (edge $(4, 7)$), traveling mode, supply path capacity reduction between Starboard PS and Starboard DC bus from 18 kW to 9 kW (180 to 90 units along edge $(5, 7)$).

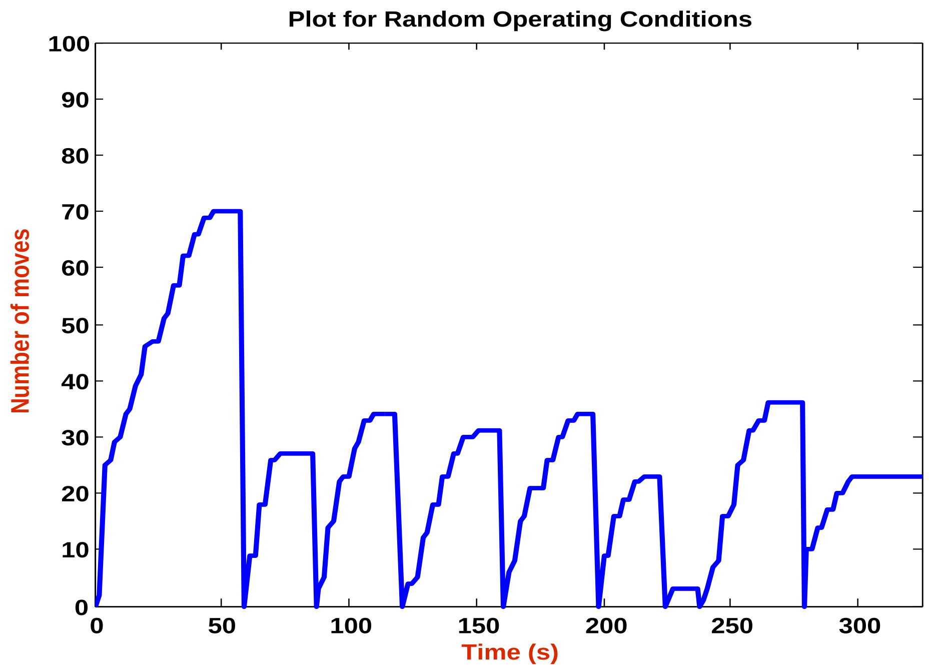


Figure 8.7: *Moves for Randomly Changing Operating Conditions*

The plot demonstrates that once the globally acceptable solution is arrived upon by the maximum flow agents, there are no more moves performed by the agents, this is indicated by the regions in the graph where the number of moves remains constant. Whenever there is any change in the system configuration, for example the ship changes its mode from harbor to fighting, the number of moves gets re-initialized to zero and the agents start working together towards forming a new global solution.

Table 8.3 gives the maximum, minimum, and average number of moves made by the agents while reconfiguring. The results show that the algorithm requires fewer than n^2 moves to compute the correct maximum flow based on the priorities of the loads starting from an arbitrary initial state: $52 \ll 22^2 = 484$.

8.5 Physical System Faults and Silent Death

To demonstrate the operation of the situational awareness layer, three cases of interest are chosen from the possible set of either normal operation or crisis scenarios and the respective decision making process presented. In the cases presented, the belief values given are in the range from zero to one indicating total rejection and agreement, respectively. Evidence

Table 8.3: *Maximum, minimum and average moves for switching between various modes*

		Full-power	**Cruising**	**Traveling**	**Harbor**	**Fighting**
Full-power	**Max**		35	29	52	33
	Avg		27	27	46	28
	Min		23	24	44	26
Cruising	**Max**	49		40	46	32
	Avg	44		31	39	27
	Min	38		24	31	22
Traveling	**Max**	41	33		45	29
	Avg	37	26		39	25
	Min	34	22		33	22
Harbor	**Max**	34	37	38		37
	Avg	28	32	32		32
	Min	25	29	28		29
Fighting	**Max**	51	38	41	49	
	Avg	44	28	35	40	
	Min	40	21	29	35	

marked as 'Unknown' means that actual evidence was not present and the decision was based on prior probabilities. This feature is important as it ensures that missing evidence, due to for example lost communication, does not invalidate the decision making process. Also, the relatively small size of the situational awareness component and the efficiency of available data mining algorithms allow evaluate all the possible cases within a fraction of a second and ensures a timely response.

8.5.1 Normal Operation

During hours of normal operation, the situational awareness layer has to report that the system is fully functioning and no corrective actions need to be taken. Entering the actual information and measurements from the lower layers into the situational awareness agent as evidence (d-Value = $d(i)+1$, Demand = 0, Silent Death = $False$, Physical System Problem = $False$) and evaluating the decision network results in utility values for the two possible actions. The network yields the actions' values for "Ping Agent" ($False = 0.98800$, $True = 0.69600$) and 'Corrective Action' ($False = 0.98200$, $True = 0.01780$.) Therefore, the network identifies correctly that no corrective action needs to be taken.

8.5.2 Silent Death

Though only a limited amount of communication is necessary to operate the distributed energy management system, it is a critical part of the agent-based approach. To examine an agent's silent death problem, where it loses it's capability to communicate with the blackboard, the scenario with (d-Value = $d(i)+1$, Demand = 0, Silent Death = $True$, Physical System Problem = $False$) has been evaluated. Reasoning yields the utility values of (False = 0.46000, True= 0.60500) for the 'Ping Agent' and (False = 0.13000, True = 0.57100) for the 'Corrective Action.' This allows notify the human operator and reconfiguration layer that the respective agent should be deactivated and can be either used to manually or automatically trigger adjustments necessary.

8.5.3 Overload

This case is simulated by providing evidence for an 'Overload' failure. The evidence of (d-Value=$Unknown$, Demand = $NotZero$, Silent Death = $Unknown$, Overload = $True$) yields (False = 0.52117, True = 0.62598) for the 'Ping Agent' and (False = 0.25591, True = 0.44927) for the 'Corrective Action.' Therefore, the situational awareness layer suggests to the command and control center that the agent has to be deactivated and reconfiguration of the system should take place.

Table 8.4: *Maximum and minimum ranges for the loads in various modes*

Mode	Load	Priority	Min demand in %	Max demand in %
$Full - power$	$StarPropulsionLoad$	$High$	100	100
	$PortbusPropulsionLoad$	$High$	100	100
	$Zone1Load1$	$High$	100	100
	$Zone1Load2$	$High$	100	100
	$Zone2Load1$	$High$	100	100
	$Zone2Load2$	$High$	100	100
	$Zone3Load1$	$High$	100	100
	$Zone3Load2$	$High$	100	100
$Cruising$	$StarPropulsionLoad$	$High$	20	50
	$PortbusPropulsionLoad$	$High$	20	50
	$Zone1Load1$	$High$	20	50
	$Zone1Load2$	Low	20	50
	$Zone2Load1$	$High$	50	90
	$Zone2Load2$	Low	50	90
	$Zone3Load1$	$High$	50	90
	$Zone3Load2$	Low	50	90
$Traveling$	$StarPropulsionLoad$	$High$	50	90
	$PortbusPropulsionLoad$	$High$	50	90
	$Zone1Load1$	Low	50	90
	$Zone1Load2$	$High$	50	90
	$Zone2Load1$	Low	20	50
	$Zone2Load2$	$High$	20	50
	$Zone3Load1$	Low	50	90
	$Zone3Load2$	$High$	50	90
$Harbor$	$StarPropulsionLoad$	Low	0	0
	$PortbusPropulsionLoad$	Low	0	0
	$Zone1Load1$	$High$	20	50
	$Zone1Load2$	Low	20	50
	$Zone2Load1$	$High$	20	50
	$Zone2Load2$	Low	20	50
	$Zone3Load1$	Low	20	50
	$Zone1Load2$	$High$	20	50
$Fighting$	$StarPropulsionLoad$	$High$	20	50
	$PortbusPropulsionLoad$	$High$	20	50
	$Zone1Load1$	$High$	50	90
	$Zone1Load2$	$High$	50	90
	$Zone2Load1$	$High$	50	90
	$Zone2Load2$	$High$	50	90
	$Zone3Load1$	$High$	20	50
	$Zone3Load2$	$High$	20	50

Chapter 9

Summary and Future work

9.1 Summary

A new agent-based concept for the energy management system of shipboard power systems has been presented. A multi-layer architecture has been used to accomplish the functionality of a human-machine interface, automatic reconfiguration using a maximum flow algorithm, agents to implement reconfiguration decisions, agents to provide situational awareness capabilities, and the mathematical model of a shipboard power system. Most of the actions necessary for the operation of the energy management system can be performed locally following a distributed concept. Complementing the energy management system is the situational awareness component whose objective is to not only support human operators by providing analysis of the operational status of the system but also to suggest corrective measures to deal with critical situations.

Assumptions made for this work concern the availability of an appropriate communication infrastructure and the simplified representation of DC distribution buses by a single node as bi-directional power flows would violate conditions necessary for the graph theoretic approach. Also, the possibility of distributed resources for the purpose of power generation is not considered.

The case studies demonstrate the feasibility and flexibility of the concept and results are promising. The agents are able to converge to a globally acceptable solution for arbitrary initial conditions and restore the loads based on priorities. The agents are able to identify changes in the system and provide an efficient means of decision support. The benefits of decision networks allow operators to first gain trust in the correctly functioning situational awareness agents before finally delegating the final decision making to the agents as well. This step will further broaden the application of agents to automate the shipboard operation and reduce the stress and reliability on human operators in times of crisis.

9.2 Future work

One of the major assumptions in the multiagent system developed was that the agents talk to each other in common vocabulary or ontology. KQML, which is a protocol for exchanging information and knowledge, can be implemented where agents have completely different internal structures but need to exchange information.

No formal agent framework is used in designing the multiagent framework. Nevertheless, the prototype presented here can be implemented with the help of tools available. For example, the Agent Building and Learning Environment (ABLE, [7]) is a good candidate.

All the simulations are presently working on a single PC. A very simple extension to this by running the simulations on two PCs, one with the MATLAB simulations and the other with the java simulations is running successfully in real-time. Nevertheless, running parts of the simulation on different machines and testing the collaborating agents in this way would allow a more elaborate investigation of the distributed energy management concept.

The multiagent system developed has not been enriched by any learning technique that would automatically improve the multiagent system. Nevertheless, AI concepts based on learning are a worthwhile next step.

Lastly, an important step would be from a purely software architecture to a mixed hardware/software approach. This step would allow investigation of appropriate means of embedded software development while dramatically boosting the response time. Consequently, successful operation of the autonomous energy management system could be performed with more stringent real-time requirements.

Appendix

A

Publications and Award

A.1 Publications

The work done so far has been presented at two conferences and published in the corresponding proceedings.

- **Energy Management System with Automatic Reconfiguration for Electric Shipboard Power Systems:**
  Abstract: The automatic reconfiguration of electric shipboard power systems is an important step toward improved fightthrough and self-healing capabilities of naval warships. The improvements are envisioned by redesigning the electric power system and its controls. This research focuses on a new scheme for an energy management system in form of distributed control agents. The control agents' task is to ensure supply of the various load demands while taking into consideration system constraints and load and supply path priorities. A self-stabilizing maximum flow algorithm is investigated to allow implementation of the agents' strategies and find a global solution by only considering local information and a minimum amount of communication. A case study using the distributed agents within a multilayer system architecture to function as energy management system is presented.
  Reference: Ganesh, S., Schoder, K., Lai, H. J., Al-Hinai, A., Feliachi, A., "Energy Management System with Automatic Reconfiguration for Electric Shipboard Power Systems" *Proc. of ASNE Reconfiguration and Survivability Symposium, RSS 2005*, Jacksonville, FL, February 2005.
- **Improving Automatic Reconfiguration of Electric Shipboard Power Systems by Adding Situational Awareness Capabilities:**

Abstract: A new layer of hierarchically organized agents is developed to complement efforts in designing energy management systems with automatic reconfiguration for electric shipboard power systems. This layer provides situational awareness capabilities to a previously implemented reconfiguration layer that was also based on agent technology. Both of these layers allow for distributed decision making while operating the system with global and optimal goals. While the reconfiguration layer performs negotiation and energy management, the situational awareness layer adds autonomous and intelligent agents that help determine component failures, tracking performance, and analyzing system events that have the potential to degrade system performance and reliability. A list of possible events includes short circuits, open circuits, loss of communication networks, and failures of agents themselves. The added functionality is using influence diagrams to assist human operators in determining the silent death of components or agents and further improves the autonomous operation capabilities of the shipboard power system. Tests of different critical scenarios are investigated in a case study to evaluate the layer's decision making performance.
Reference: Ganesh, S., Schoder, K., A., Feliachi, A., "Improving Automatic Reconfiguration of Electric Shipboard Power Systems by Adding Situational Awareness Capabilities," *Proc. of ASNE Intelligent Ship Symposium*, Villanova, PA, June 2005.

A.2 Award

- **Sigma Xi Poster Award**: This research work was awarded 3rd place in the Engineering Research Area of the Sigma Xi National Honor Society's Graduate Research Competition 2005. The competition was held on April 25, 2005, and attracted more than 74 entrees in the three different categories of engineering, life and agricultural science, and general science.

Appendix

B

Transfer Functions for Physical System Layer

The transfer functions and their respective parameters of the mathematical model of the shipboard power system are given in the following. The various transfer functions represent input-output models for power generation, converter modules, and loads. The blocks' transfer functions and parameters have been determined through simulation and identification of the Naval Combat Survivability systems as presented [74] [88]. The only modifications made to the system concern the loads: Only the constant power demand type is used here as it adequately represents all load types for the power flow solution. Pulse loads have been ignored but can be incorporated through the constant power load type as well. Also, two loads instead of only one are modeled within each of the three zones. For simulation purposes the equivalent discrete system model with a sampling time of 50 ms has been used.

Table B.1: *Parameters for Generator*

Parameter	**Value**	**Unit**
V_{droop}	0.1	p.u.
$H_{inertia}$	1	p.u.
tg_1	0.2	sec.
kp	2	p.u.

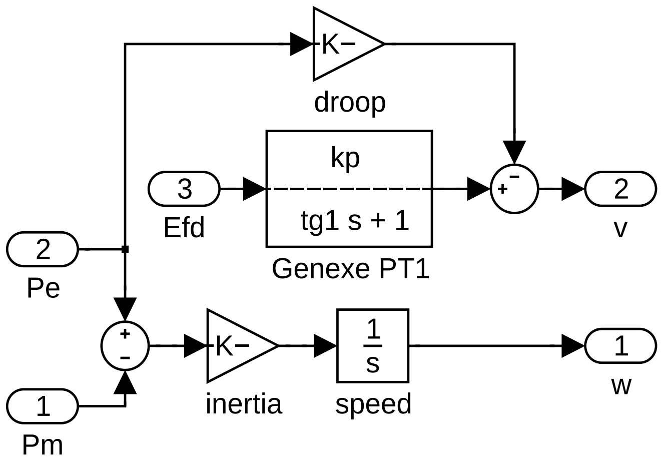


Figure B.1: *Synchronous generator model*

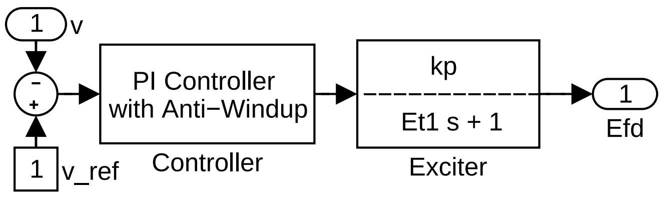


Figure B.2: *Exciter model and its control*

Table B.2: *Parameters for Exciter*

Parameter	Value	Unit
Et_1	0.1	sec
kp	1	p.u.
$PI : ki$	2	p.u.
$PI : kp$	4	p.u.
$PI : max.limit$	1.02	p.u.
$PI : min.limit$	0	p.u.

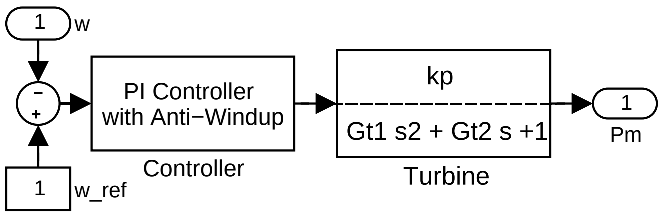


Figure B.3: *Turbine model and its control*

Table B.3: *Parameters for Turbine*

Parameter	Value	Unit
Gt_1	0.0625	sec
Gt_2	0.45	sec
kp	0.98	p.u.
$PI : ki$	1	p.u.
$PI : kp$	2	p.u.
$PI : max.limit$	1.02	p.u.
$PI : min.limit$	0	p.u.

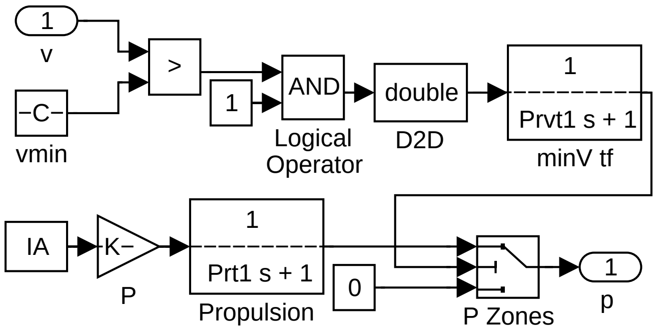


Figure B.4: *Shipboard propulsion system model*

Table B.4: *Parameters for Propulsion Load*

Parameter	Value	Unit
$Prvt_1$	0.01	sec
Prt_1	0.1	sec
P	0.55	p.u.
$vmin$	0.8	p.u.

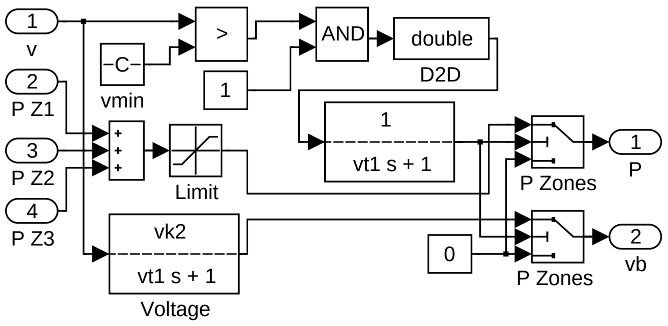
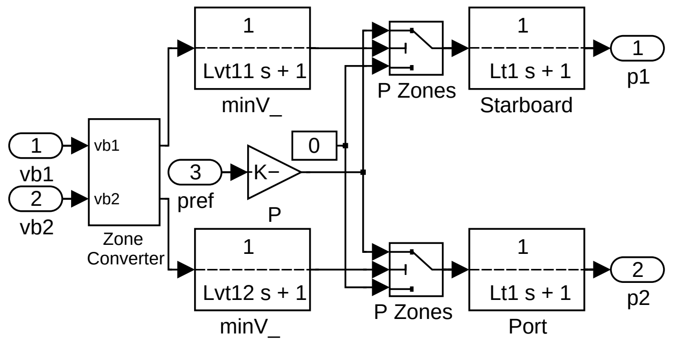


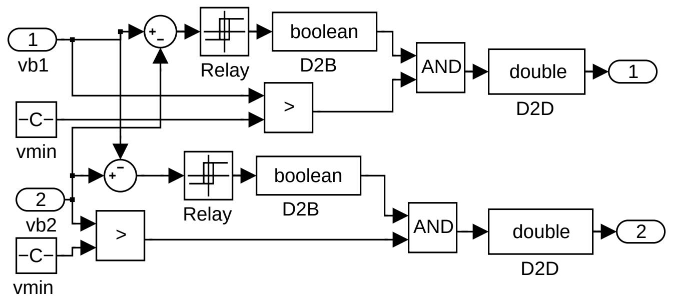
Figure B.5: *AC/DC converter model and its control*

Table B.5: *Parameters for PS (AC/DC) Converter*

Parameter	Value	Unit
$vt1$	0.2	sec
vt_1 Voltage	0.2	sec
vk_2Port	0.95	p.u.
$vk_2Starboard$	0.9	p.u.
$vmin$	0.8	p.u.



(a) ***Zone model and its control***



(b) ***Zone converter model (part of Zone model)***

Figure B.6: *Zone load and its converter model*

Table B.6: *Parameters for Zone Converter and Load*

Parameter	Value	Unit
$Lvt11$	0.01	sec
$Lvt12$	0.01	sec
$Lt1Starboard$	0.1	sec
$Lt1Port$	0.1	sec
$vmin$	0.7	p.u.

References

[1] Ahuja, R. K., Magnanti, T. L., Orlin, J. B., *Network Flows - Theory, Algorithms, and Applications*, Prentice Hall, 1993.

[2] Advanced Power Technologies (APT) Consortium, http://www.ee.washington.edu/ener-gy/apt.

[3] Arai, T., Stolzeburg, F., "Multiagent Systems Specification by UML Statecharts Aiming at Intelligent Manufacturing," *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, Bologna, Italy, July 2002.

[4] Alternative Transients Program (ATP), http://www.emtp.org.

[5] Awerbuch, B., Varghese, G., "Distributed program checking: a paradigm for building self-stabilizing distributed protocols," *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, 258–267, 1991.

[6] Bigus, J. P., Bigus, J., *Constructing Intelligent Agents Using Java*, John Wiley & Sons, 2001.

[7] Bigus, J. P., Schlosnagle, D. A., Pilgrim, J. R., Mills III, W. N., Diao, Y., "ABLE: A toolkit for building multiagent autonomic systems," *IBM Systems Journal*, 41(3), 2002.

[8] Booch, G., *Object-Oriented Analysis and Design with Applications*, Addison Wesley, Reading, MA, 1994.

[9] Brooks, F. P., *The mythical man-month*, Addison Wesley, 1995.

[10] Budde, R., Kuhlenkamp, K., Mathiassen, L., Zullighoven, H, *Approaches to Prototyping*, Springer-Verlag, New York, NY, 1984.

[11] Burmeister, B., Haddadi, A., Matylis, G., "Applications of multi-agent systems in traffic and transportation," *IEE Transactions on Software Engineering*, 144(1), 51–60, February 1997.

[12] Butler, K. L., Sarma, N. D. R., Whitcomb, C., Do Carmo, H., Zhang, H., "Shipboard Systems Deploy Automated Protection," *IEEE Computer Applications in Power*, 11(2), 31–36, April 1998.

[13] Butler, K. L., Sarma, N. D. R., Prasad, V. R., "Network reconfiguration for service restoration in shipboard power distribution systems," *IEEE Transactions on Power Systems*, 16, 653–661, November 2001.

[14] Butler-Purry, K. L., Sarma, N. D. R. , "Self-Healing Reconfiguration for Restoration of Naval Shipboard Power Systems," *IEEE Transactions on Power Systems*, 19(2), 754–762, May 2004.

[15] Cammarata, S., McArthur, D., Steeb, R., "Strategies of cooperation in distributed problem solving," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, (IJCAI-83), 767–770, Karlsruhe, Federal Republic of Germany, 1983.

[16] Charniak, E., "Bayesian Networks without Tears," *AI Magazine*, 12(4), Winter Issue, 1991.

[17] Chavez, A., Maes, P., "Kasbah: An agent marketplace for buying and selling goods," *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, 75–90, London, UK, 1996.

[18] Chen, L., Sycara, K., "Webmate : A personal agent for browsing and searching," *Proceedings of the Second International Conference on Autonomous Agents*, Minneapolis/St Paul, MN, May 1998.

[19] Clarke, E. M., Emerson, E. A., "Design and synthesis of synchronization skeletons using branching time temporal logic" *In D. Kozen, editor, Logics of Programs*, 131, 52–71, Springer-Verlag, 1981.

[20] Cockburn, D., Jennings, N. R., "ARCHON: A Distributed Artificial Intelligence System for Industrial Applications," *In: Foundations of Distributed Artificial Intelligence*, 319–344, Wiley, 1996.

[21] Distributed Intelligence for Automated Survivability Thrust, *ONR Science & Technology*, http://www.onr.navy.mil, 2002.

[22] Deverill, P., Newell, C. J., Rottier, P., Bennett, A., Bryce, S., Healey, N., "Modelling of a warships electrical power generation and propulsion system, and practical implications for supporting the acquisition process," *Proceedings of The Institute of Marine Engineering, Science and Technology AES 2003*, (Edinburgh International Conference Centre, Edinburgh, UK), February 2003.

[23] Dijkstra, E. W., "Self-stabilizing systems in spite of distributed control," *Communications of the ACM*, 17, 643–644, 1974.

[24] Dutta, P. S., Sen, S., Peng, J., "Applying Bayesian Mixed-Initiative Agents in Real-World Reasoning,", in notes of the *IJCAI Workshop on Autonomy, Delegation and Control: Interacting with Autonomous Agents*, Seattle, WA, August 4–10 2001.

[25] Erman, L. D., Hayes-Roth, F., Lesser, V. R., Reddy, D. R., "The HEARSAY-II speech-understanding system: Integrating knowledge to resolve uncertainty," *Computing Surveys* 12(2), 213–253, 1980.

[26] Ferber, J., *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, Addison–Wesley, Harlow, UK, 1999.

[27] Foundation of Intelligent Physical Agents, http://www.fipa.org, 2005.

[28] Fisher, M., Wooldridge, M., "On the formal specification and verification of multi-agent systems," *International Journal of Cooperative Information Systems*, 6(1), 37–65, 1997.

[29] Franklin, S., Gasser, A., "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents," *Third International Workshop on Agent Theories, Architectures and Languages*, Springer-Verlag, 1996.

[30] Bradshaw, J., *Software Agents*, The AAAI Press/The MIT Pres, Cambridge, MA, 1997.

[31] Gilbert, D., Aparicio, M., Atkinson, B., Brady, S., Ciccarino, J., Grosof, B., O'Connor, P., Osisek, D., Pritko, S., Spagna, R., Wilson L., "The role of intelligent agents in the information infrastructure," *IBM Report*, 1995.

[32] Goldberg, A. V., Tarjan R. E., "A new approach to the maximum flow problem," *J ACM*, 35, 921–940, 1988.

[33] Goldberg, A. V., "Recent Developments in Maximum Flow Algorithms," *Technical Report No. 98-045, NEC Research Institute, Inc.*, April 1998.

[34] Gosh, S., Gupta, A., Pemmaraju, S. V., "A Self-stabilizing Algorithm for the Maximum Flow Problem," *Distributed Computing*, 10, 167–180, 1997.

[35] Grand, S., Cliff, D., " Creatures: Entertainment software agents with artificial life," *Autonomous Agents and Multi-Agent Systems*, 1(1), 1998.

[36] Hayes-Roth, B., Hewett, M., Washington, R., Hewett, R., Seiver, A., "Distributing intelligence within an individual," *Distributed Artificial Intelligence*, 2, 385-412, Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1989.

[37] Hayes-Roth, B., "Agents on stage: Advancing the state of the art in AI," *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 967-971, Montréal, Quéebec, Canada, August 1995.

[38] Hegner, H., Tavener, K., Robey, H., "Integrated fight through power ship design considerations,", *Proceedings of The Institute of Marine Engineering, Science and Technology AES 2003*, (Edinburgh International Conference Centre, Edinburgh, UK), February 13–14 2003.

[39] Hoare, C. A. R., "An axiomatic basis for computer programming," *Communications of the ACM*, 12(10), 576–583, 1969.

[40] Hoek, W. van der., Wooldridge, M., "Towards a Logic of Rational Agency," *Logic Journal of IGPL*, 11(2), 133–157, 2003.

[41] Horvitz, E., Barry, M., "Display of Information for Time-Critical Decision Making," *11th Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, San Francisco, CA, 1995.

[42] Huang, J., Jennings, N. R., Fox, J., "An agent-based approach to health care management," *Applied Artificial Intelligence*, 9(4), 401-420, 1995.

[43] Gilbert D., Aparicio M., Atkinson B., Brady S., Ciccarino J., Grosof B., O'Connor P., Osisek D., Pritko S., Spagna R., Wilson L., "The Role of Intelligent Agents in the Information Infrastructure," *IBM Report*, 1995.

[44] Java Agent DEvelopment Framework, http://jade.tilab.com, 2005.

[45] Java Technology, http://java.sun.com, 2005.

[46] Jennings, N. R., Wooldridge, M., "Applying agent technology," *Applied Artificial Intelligence*, 9(6), 357-370, 1995.

[47] Jennings, N. R., Corera, J., Laresgoiti, I., Mamdani, E. H., Perriolat, F., Skarek P., Varga, L. Z., "Using ARCHON to develop real-world DAI applications for electricity transportation management and particle accelerator control," *IEEE Expert*, December 1996.

[48] Jennings, N. R., Faratin, P., Johnson, M. J., Norman, T. J., O'Brien, P., Wiegand, M. E., "Agent-based business process management," *International Journal of Cooperative Information Systems*, 5(2-3),105-130, 1996.

[49] Jennings, N. R., Sycara, K., Wooldridge, M., "A Roadmap to Agent Research and Development," *Autonomous Agents and Multi-Agent Systems*, 1998.

[50] Jennings, N. R., Wooldridge, M., "Agent-Oriented Software Engineering," *Handbook of Agent Technology*, AAAI/MIT Press, 2000.

[51] Jennings, N. R., Bussmann, S., "Agent-Based Control Systems," *IEEE Control Systems Magazine*, June 2003.

[52] Jensen, F. V., *An Introduction to Bayesian Networks*, Springer-Verlag, New York, 1996.

[53] Kautz, H., Selman, B., Shah, M., "The hidden web," *AI Magazine*, 18(2), 27-35, 1997.

[54] Korb, K. B., Nicholson, A. E., *Bayesian Artificial Intelligence*, Chapman & Hall/CRC, 2004.

[55] Krulwich, B., "The BargainFinder agent: Comparison price shopping on the internet," *Bots, and other Internet Beasties*, 257–263, 1996.

[56] Lightweight Extensible Agent Platform, http://leap.crm–paris.com.

[57] Li, H., Rosenwald, G., Jung, J., Liu, C. C., "Strategic Power Infrastructure Defense," *Proceedings of the IEEE*, 93(5), 918–933, 2005.

[58] Ljunberg, M., Lucas, A., "The OASIS air traffic management system," *Proceedings of the Second Pacific Rim International Conference on AI*, Seoul, Korea, 1992.

[59] Luck, M., D'Inverno, M., "A formal framework for agency and autonomy," *In Proceedings of the First International Conference on Multi-Agent Systems*, 254–260, San Francisco, CA, June 1995.

[60] Maes, P., "Agents that reduce work and information overload," *Communications of the ACM*, 37(7), 31-40, July 1994.

[61] Maes, P., "Artificial life meets entertainment: life like autonomous agents," *Communication of the ACM*, 38(11), 108–114, 1995.

[62] The Mathworks, http://www.mathworks.com, 2005.

[63] Maturana, F. P., Staron, R. J., Discenzo, F. M., Hall, K., "Intelligent Autonomous Control Architecture for Automated Ship Control, Damaged and Optimized Operations," *ASNE Intelligent Ship Symposium*, Villanova, PA, June 2005.

[64] McGeary, F., Decker, K., "DECAF Programming: Agents for Undergraduates," *Workshop on Infrastructure for Scalable Multi-Agent Systems, Autonomous Agents 2001*, 53–60, May 2001.

[65] *Notes of the AAAI-97 Spring Symposium on Computational Models for Mixed Initiative Interaction*, 1997.

[66] Morgan, C., *Programming from Specifications*, Prentice Hall International: Hemel Hempstead, England, Second edition, 1994.

[67] Murray, J., "Specifying Agents with UML in Robotic Soccer," *Proceedings of the 1st International Joint Conference on Autonomous Agents & Multi-Agent Systems*, Bologna, Italy, 2002.

[68] "Netica , Application for Belief Networks and Influence Diagrams", *User's Guide*, Version 1.05 for Windows, 1997.

[69] Norsys Software Corp., *Netica Application and API*, http://www.norsys.com, 2002.

[70] Object Management Group, Inc. *OMG Unified Modeling Language Specification*, Version 1.5, March 2003.

[71] Owens, D. K., Shahcter, R., Nease, R. F., "Representation and Analysis of Medical Decision Problems with Influence Diagrams", *Medical Decision Making*, 17(3), 241–262, 1997.

[72] Overgaard, L., Petersen, H. G., Perram, J. W., "Reactive motion planning: a multi-agent approach," *Applied Artificial Intelligence*, 10(1), 35-52, 1996.

[73] Parunak, H. V. D., "Applications of distributed artificial intelligence in industry," *In Foundations of Distributed Artificial Intelligence*, eds. O'Hare, G. M. P., Jennings, N. R., Wiley, 1995.

[74] Pekarek, S. D., Tichenor, J., Sudhoff, S. D., Sauer, J. D., Delisle, D. E., Zivi, E. J., "Overview of a Naval Combat Survivability Program," *Proceedings of the 13th International Ship Control Systems Symposium*, Orlando, FL, 2003.

[75] Pressman, R. S., *Software Engineering: A Practitioner's Approach*, McGraw-Hill, Fifth edition, 2001.

[76] PSpice A/D Simulator, *MicroSim Corporation*, Version 8.0, July 1997.

[77] Rehtanz, C., *Autonomous Systems and Intelligent Agents in Power System Control and Operation*, Springer-Verlag Berlin Heidelberg, 2003.

[78] Maturana, F. P., Staron, R. J., Hall, K. H., *Rockwell Automation*, "Methodologies and Tools for Intelligent Agents in Distributed Control," *IEEE Intelligent Systems*, January 2005.

[79] Russel, S., Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice Hall Series in Artificial Intelligence, 2004.

[80] Saberbook Navigator, *Analogy Inc.*, Version 4.3.

[81] Scheidt, D. H., "Intelligent Agent-Based Control," *Johns Hopkins Apl Technical Digest*, 23(4), 2002.

[82] Schoonderwoerd, R., Holland, O., Bruten, J., "Ant-like agents for load balancing in telecommunications networks," *Proceedings of the First International Conference on Autonomous Agents*, 209-216, Marina del Rey, CA, 1997.

[83] Searle, J. R., *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press, 1970.

[84] Sen, S., "Multiagent Systems: Milestones and New Horizons," *Trends in Cognitive Science*, 1(9), 334–339, 1997.

[85] Shachter, R., "Evaluating Influence Diagrams", *Operations Research*, 34(6), November-December 1986.

[86] Srivastava, S. K., Butler-Purry, K. L., Sarma, N. D. R., "Shipboard power Restored for Active Duty," *IEEE Computer Applications in Power*, 16–23, July 2002.

[87] Stankovic, A. M., Calvic, M. S., "Graph-oriented algorithm for the steady state security enhancement in distributed networks,", *IEEE Trans. Power Delivery*, 4(1), 539–544, 1989.

[88] Sudhoff, S. D., Glover, S. F., Zak, S. H., Pekarek, S. D., Zivi, E. J., Delisle, D. E., "Stability Analysis Methodologies for DC Power Distribution Systems," *13th International Ship Control Systems Symposium*, Orlando, FL, April 7–9 2003.

[89] Sun, L., Cartes, D. A., "Reconfiguration of Shipboard Radial Power System using Intelligent Agents," *ANSE Electric Machine Technology Symposium*, Philadelphia, PA, 2004.

[90] Sycara, K., Decker, K., Pannu, A., Williamson, M., Zeng, D., "Distributed Intelligent Agents," *IEEE Expert*, 11(6), 1996.

[91] Tel, G., *Introduction to Distributed Algorithms*, Cambridge University Press, 1994.

[92] Trappl, R., Petta, P., *Creating Personalities for Synthetic Actors*, Springer-Verlag: Berlin, Germany, 1997.

[93] Tseng, K. J., Foo, C. F., Palmer, P. R., "Implementing Power Diode Models in SPICE and Saber," *Proceedings of the 25th IEEE Annual Power Electronics Specialists Conference*, PESC'94, 59–63, June 1994.

[94] Tucker, A. J., "Opportunities and Challenges in Ship Systems and Control at ONR," *IEEE Conference on Decision and Control*, December 2001.

[95] US Navy ONR/NSF EPNES, "ONR Control Challenge Problem (white paper)," http://www.usna.edu/EPNES/Challenge Problem.htm, 2002.

[96] Wang, X., Vittal, V., "System Islanding Using Minimal Cutsets with Minimum Net Flow," *Power Systems Conference and Exposition, IEEE PES*, 1, 379–384, October 2004.

[97] Wavish, P., Graham, M., "A situated action approach to implementing characters in computer games," *Applied Artificial Intelligence*, 10(1), 53-74, 1996.

[98] Weiss, G., *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 1999.

[99] Wooldridge, M., Jennings, N. R., "Intelligent agents: theory and practice," *The Knowledge Engineering Review*, 10(2), 115–152, 1995.

[100] Wooldridge, M., "Agent-Based Software Engineering," *IEE Proceedings on Software Engineering*, 144, 26–37, 1997.

[101] Wooldridge, M., *An introduction to Multiagent Systems*, John Wiley & Sons, 2002.

[102] Xiao, H., Adediran, A. T., Butler, K. L., "Fault scenario studies based on geographical information for shipboard power systems," *Proceedings of North American Power Symposium*, 436-442, October 2001.

[103] Yusof, S. B., Rogers, G. J., Alden, R. T. H., "Slow coherency based network partitioning including load buses,", *IEEE Transactions on Power Systems*, 8(3), 1375-1382, August 1993.

[104] Zivi, E. L., McCoy, T. J., "Control of a Shipboard Integrated Power System," *Proceedings of the 33rd Annual Conference on Information Sciences and Systems*, Baltimore, MD, 1999.

[105] Zhang, H., Butler, K. L., "Simulation of ungrounded shipboard power systems in PSpice," *Proceedings Midwest Symposium on Circuits and Systems*, 58-62, 1998.

[106] Zhang, H., Butler, K. L., Sarma, N. D. R., DoCarmoand, H., Gopalakrishnanand, S., Adediran, A., "Analysis of tools for simulation of shipboard electric power systems," *Electric Power Systems Research*, 58, 111-122, 2001.