Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Robin Agung Kusmanto

Entitled Modeling and Simulation of an Optimized Wireless Network in a Naval Ship System of Systems

For the degree of Master of Science in Mechanical Engineering

Is approved by the final examining committee:

Douglas E. Adams
Chair

Alok R. Chaturvedi

Daniel A. DeLaurentis

Peter H. Meckl

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Douglas E. Adams

Approved by: Anil K. Bajaj 7/7/2009
Head of the Graduate Program Date

Graduate School Form 20
(Revised 6/09)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

Modeling and Simulation of an Optimized Wireless Network in a Naval Ship System of Systems

For the degree of Master of Science in Mechanical Engineering

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Robin Agung Kusmanto

Printed Name and Signature of Candidate

07/07/2009

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

MODELING AND SIMULATION OF AN OPTIMIZED WIRELESS NETWORK IN A NAVAL SHIP SYSTEM OF SYSTEMS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Robin Agung Kusmanto

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Mechanical Engineering

August 2009

Purdue University

West Lafayette, Indiana

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

To my dearest family and friends.
Thank you for being source of inspiration throughout this whole journey.

ACKNOWLEDGMENTS

The author would like to thank the following for their contributions and invaluable advice during the course of this work:

- Prof. Douglas E. Adams for his support, advice, dedication, and patience which has mostly contributed to the successful completion of this thesis. Our innumerable conversations and emails kept me motivated throughout the course of my studies.
- Prof. Peter H. Meckl, Prof. Daniel A. DeLaurentis, and Prof. Alok Chaturvedi for taking time from their busy schedules to serve on my advisory committee and for their helpful suggestions that have helped improve this thesis.
- Prof. Edward J. Delp, Prof. Oleg Wasynczuk from Electrical Engineering Department, and Matthew M. Wilson from Simulex, Inc for taking time from their busy schedules and sharing many insightful conversations during the weekly meetings that helped in the development of the ideas presented in this thesis.
- Faculty and staff (Judy, Donna, Bob, Gil, Frank, Fritz, Dave) of the Ray W. Herrick Laboratories and Center for Systems Integrity for the help extended ever cheerfully and graciously.
- Faculty of Purdue University whose courses helped me strengthen my fundamentals in this area of engineering.
- Prof. Anil K. Bajaj and the ME graduate office and business staff for their assistance during my study.
- Tejas H. Bhatt, Chih-Hui Hsieh, and Angela K. Mellema from SEAS Laboratory and Marc Bosch-Ruiz, Gagan R. Gupta, and Abdallah A. Khreishah from Electrical Engineering Department for their help with some of the experiments reported in this work.

- My fellow labmates and other students at Ray W. Herrick Laboratory and Purdue Center for Systems Integrity, including Shawn McKay and Vishal Mahulkar for their time and suggestions which have helped me completing this work.
- My roommates and fellow friends who have helped me and made my life at West Lafayette enjoyable.

None of this would have been possible without the unstinting support and encouragement of family and friends; especially my parents, Sulistyo and Mary Kusmanto and my lovely sister, Angela Kusmanto.

Thanks to the countless others who have helped me in my endeavors. Above all, Thank You God for the countless blessings you have bestowed upon me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Kusmanto, Robin A. M.S.M.E., Purdue University, August 2009. Model Identification and Simulation of a Naval Ship System of Systems Containing a Wireless Network. Major Professor: Dr. Douglas E. Adams, School of Mechanical Engineering.

The United States Navy would like to reduce operational bottlenecks and manning costs on ships to enable efficient naval operations. The main objective of this thesis was to develop a model-based simulation for evaluating the operational capability of a Navy ship and its crew as a system of systems. In agent-based simulations of this system of systems model, a wireless computer network enabled interactions amongst the crew throughout the ship environment. The effects of normal operating scenarios as well as emergency scenarios, such as compartment fires and equipment failures, were studied.

Physical experiments consisting of signal strength measurements and data transfers were performed on a small network, and the non-linear relationships between the locations of crewmembers, signal strengths, and corresponding data rates were recorded and analyzed. The model that was developed describes the behavior of the signal due to several obstacles that were used to represent a simulated ship environment. The number of crewmembers, access points, and failure scenarios corresponding to a specific schedule was used to examine characteristics related to access point utilization and data transfer times. In the set of simulated ship operational scenarios, the wireless network communication performance was shown to be robust to failure of either the highest or lowest utilized access points. Finally, a reduced number of crewmembers communicating over the wireless network was shown to enable the ship to complete its defined missions.

CHAPTER 1. INTRODUCTION

1.1. Background

Civilian and military organizations are becoming increasingly more complex as the technology for conducting business and communicating across organizations becomes more sophisticated. For example, the introduction of telephones to allow two individuals at a distance to communicate in real time enhanced the productivity of businesses. The introduction of the Internet along with its capability to support electronic mail ushered in a new means of communication across multiple organizations. Today, wireless communication devices enable workers to communicate via telephone and electronic mail simultaneously with other workers regardless of geographic location.

Despite the merits of these new modes of communication, new technologies for communication can also result in organizational challenges. For example, wireless local area networks are being introduced on some United States Navy warfighters to enable unprecedented capabilities in communications among the crew, communications between the ship's commanders and the crew, and communications from ship to shore to access subject matters experts that can help the crew resolve critical issues dealing with everything from maintenance to warfare. Congestion as multiple crewmembers attempt to communicate using the same access points within this network could potentially cause reductions in their productivity if the workflows for crew are not designed in accordance with the technologies they are using for communication over the network. These types of unanticipated interactions are referred to as "emergent phenomena" within this thesis. Crewmembers are also being equipped with personal data devices in many instances to increase the bandwidth of communication that is possible given the wireless nature of these networks. As the crew's access to the network is enhanced, so too is the

likelihood for subtle, possibly disruptive interactions, or emergent phenomena, between crewmembers who are simultaneously attempting to access maintenance records, submit their reports, and conduct other ships' business.

Likewise, technologies for monitoring of machines onboard the ship are also enabling these machines to communicate their condition to the crew so that proactive maintenance actions can be taken to prevent machine failures and the associated costly downtime. As these machines communicate their data over the wireless network, there is the possibility for yet additional conflicts between the crew's intended use of the wireless network and a machine's usage of the network for conveying its operational condition to the crew.

Wireless systems have several advantages in a shipboard environment. Electric or fiber optic cables add significant weight to ships. They add some complexity to the design process and are expensive to procure, maintain, and replace. The passing of cables from compartment to compartment requires that holes be cut in the bulkhead, which weakens the bulkhead. In order to reinforce the bulkhead, it must be made thicker, which increases the weight and cost of the ship. Furthermore, the cables are susceptible to damage. A damaged cable could cause the information path to be damaged or destroyed. This situation would require the crewmembers to repair wiring infrastructure.

Wireless networks utilize radio waves, which rely less on cable management. The network configuration, which determines the number and location of server and access points to be deployed, is the main design challenge encountered when switching from the wired network. However, wireless systems are more easily and inexpensively reconfigured than their wired counterparts. Overall, wireless systems can be more cost effective and have higher levels of survivability (Estes, 2001).

The United States Navy has the desire to improve the evaluation of new technologies before they are implemented on ships. The ability to predict how these technologies will affect crew performance for various manning levels is of particular interest. The high

costs of new technology, additional training of crewmembers, and other infrastructural changes need to be justified prior to the deployment of new technologies. Seventy percent of the total ownership cost of a ship is in operation and support, and 51% of these costs are associated with manning levels and personnel related costs. When decisions are made regarding the reduction of manning levels to contain these high costs, the ship's capability to conduct a given mission or a set of missions must be evaluated through simulations to justify these decisions. For example, it may be possible to reduce manning levels because of the more efficient communications that are enabled by a wireless network; however, it must be verified that this manning level for the crew can support all possible missions (e.g., maintenance, emergency event involving fire onboard). This verification of crew performance across all possible missions must take into account the emergent phenomena that are brought about by the strong and weak coupling amongst the crew, technology, the ship's infrastructure, and the ship's geography. Stated in another way, manning requirements and the total ownership cost are defined using a single variable, the number of crewmembers and the dollar value, respectively; whereas a ship's capability embodies a multiplicity of variables, which change for different missions.

There is a minimum total ownership cost for which the associated manning level is said to be optimal for a particular performance level in warfighting. The locus of all such minimum ownership costs, considering all different mission requirements, is a curve along a surface (see line labeled 'optimized manning' in Figure 1-1 taken from Spindel et al., 2000).

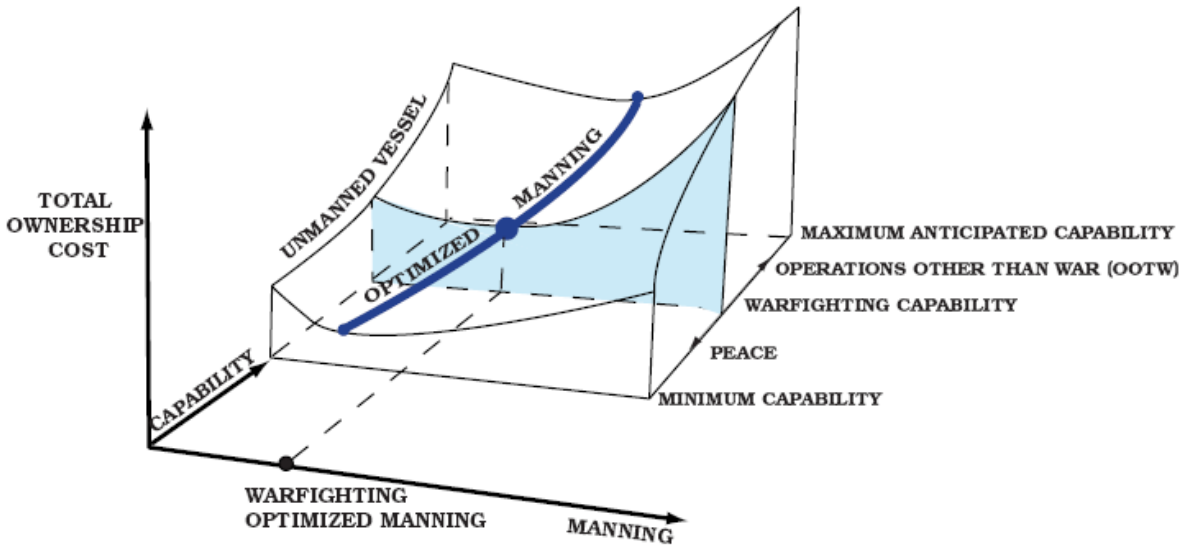


Figure 1-1. Optimized Manning Design (Spindel et al., 2000)

Technology implementation plans must be quickly developed and updated before, between, and during missions to optimize the decision-making process related to manning levels. In order to develop predictive models for simulating the impact of these new technologies and processes on the crew and mission, the entire ship System-of-Systems (SoS) must be considered. DeLaurentis et al.(2004) point out that there is a tradeoff between the design of a product to optimize its own performance and the design of a network to optimize its overall performance. System-of-systems approaches are aimed at analyzing this tradeoff by examining subtle interactions, which take place between the various systems (Mahulkar, 2006).

1.2. Literature Survey

The idea behind SoS simulations was brought forward on October 1995 by scientists in the United States Marine Corps and at MITRE in what is currently known as Project Albert. Project Albert used a series of new models and tools, multidisciplinary teams, and a scientific method to explore various questions about SoS. It was a research effort that

emphasized the process of looking at the whole of a system rather than reducing such systems into parts. In summary, Project Albert utilized agent-based simulations to develop new tools to capture emergent behavior in synthetic environments that would evolve into more effective warriors (Brandstein et al., 2000).

Hoffman and Horne (1998; Brandstein et al., 2000) described the results of the initial efforts by the United States Marine Corps to understand the potential mesh of nonlinear and complex adaptive systems within the context of warfare. One such effort was the development of an agent-based model called ISAAC (Irreducible Semi-Autonomous Adaptive Combat), a mobile cellular automata model in which the individual fighting entities, called agents, moved through a lattice carrying information with them as they moved. The agents were given characteristics that included the following: a default local rule set to specify how an agent should act in a generic environment, goals directing behavior, sensors generating an internal map of the environment, and an internal mechanism to alter behavior.

Another model developed by the United States Marine Corps was Swarrior, an interactive model based on Hunter Warrior Advanced Warfighting Experiments (Brandstein et al., 2000). "The Hunter Warrior experiment was designed to examine three specific areas ... The first objective area covered tactical operations on the dispersed, noncontiguous battlefield. The performance of small units against a numerically superior force on a battlefield that has no front, flank, or rear areas, was a key component of this objective. The second area was command, control, communications, computers, and intelligence, and the single battle concept. The ability to create and utilize a shared, digital communications network will be crucial … on the battlefield and on naval ships. Experiments examined the digital network and information-sharing. The third objective was the enhancement of fire support and improved targeting … This experiment evaluated the ability of sea-based forces to operate successfully on a digital and extended battlefield using these new concepts and technologies. These concepts and technologies

were intended to result in additional capabilities beyond the core competencies that already existed" (Cox, 1997; Brandstein et al., 2000; Pike, 2000).

Niraj et al. (2005) presented computer based simulations as effective tools for human system integration optimization, as well as for studying the risks associated with complex interactions between crewmembers and systems. The proposed modular simulation environment enabled analysts to choose and integrate the best combination of agent, discrete event, and physics-based simulations to address questions of manning. The environment incorporated advances in complexity theory for simulating non-linear systems, knowledge discovery for data analysis, and distributed computing.

A complex system is defined as a system composed of a large number of entities that have many interactions both within and among the entities. Recently, there has been an increasing recognition that new methods are needed to deal with the particular challenges presented in designing, integrating, protecting, and optimizing collections of independently operating complex systems, or Systems-of-Systems (SoS).

The precise definition of SoS is still being discussed. Experts have different perspectives on SoS as discussed by Sage and Cuppan (2001), DeLaurentis et al.(2007), Jamshidi (2008), Carlock and Fenton (2001), Pei (2000), Lukasik (1998), and Manthorpe Jr (1996). Maier (1998) emphasized that a SoS is different from a complex, monolithic system in terms of five characteristics:

a. "Operational Independence of the Elements: If the SoS is disassembled into its component systems the component systems must be able to usefully operate independently. The SoS is composed of systems, which are independent and useful in their own right.
b. Managerial Independence of the Elements: The component systems not only can operate independently, they do operate independently. The component systems are separately acquired and integrated but maintain a continuing operational existence independent of the SoS.

c. Evolutionary Development: The SoS does not appear fully formed. Its development and existence is evolutionary with functions and purposes added, removed, and modified with experience.
d. Emergent Behavior: The system performs functions and carries out purposes that do not reside in any component system. These behaviors are emergent properties of the entire SoS and cannot be localized to any component system. The principal purposes of the SoS are fulfilled by these behaviors.
e. Geographic Distribution: The geographic extent of the component systems is large. Large is a nebulous and relative concept as communication capabilities increase, but at a minimum it means that the components can readily exchange only information and not substantial quantities of mass or energy." (Maier, 1998)

Some of the current applications of a SoS in the military arethe United States Navy defense system and Thousand Ship Navy program (Gilbertson, 2007). This SoS consists of a variety of ships that have different attack and defense capabilities and operate throughout multiple regions of the globe. Those ships are integrated with weapon systems to enable a diverse range of capabilities that no individual system, such as a Carrier, Destroyer, Sealift, Amphibious Warfare Ship, and Submarine, can possess. Each of the individual systems is designed and developed individually and can function alone or in sync with other systems to satisfy the requirements for a defense SoS, which has the ability to provide complete information, track movement at sea, and guide weapon systems to enemy threats. The separate design and management of the individual systems, in conjunction with the emergent capabilities that no single system possesses, but which emerges when the systems are used in combination, make the defense system a SoS.

Despite being one element of this enterprise wide SoS, the United States Navy Warfighter ship also comprises a complex interconnected SoS consisting of

geographically distributed infrastructures for power generation and distribution systems, weapons systems, transmission systems, communication systems, hydraulic systems, etc., which interact directly with crew or indirectly over the shipboard network as the crew executes its workflow as illustrated in Figure 1-2. These interconnected entities can function independently in some cases but must interact with one another to efficiently execute mission objectives. Wireless networks in the ship would be very useful as a means of communicating among the crewmembers and the servers. The crewmembers are able to write and submit reports through the wireless network and notify each other should an emergency occur. The machines and equipment operate independently as systems with respect to the wireless network. However, the wireless network, equipped with wireless sensors, must interact with the machines so that the crew members are able to be informed about failures and problems in the machines. Such a SoS can exhibit emergent phenomena that may not be observed in Sea Trial experiments. Furthermore, infrastructure is being continuously repaired and replaced, and new hardware and software are being installed to develop new naval capabilities, showing the evolutionary development of the systems (Mahulkar et al., 2008).

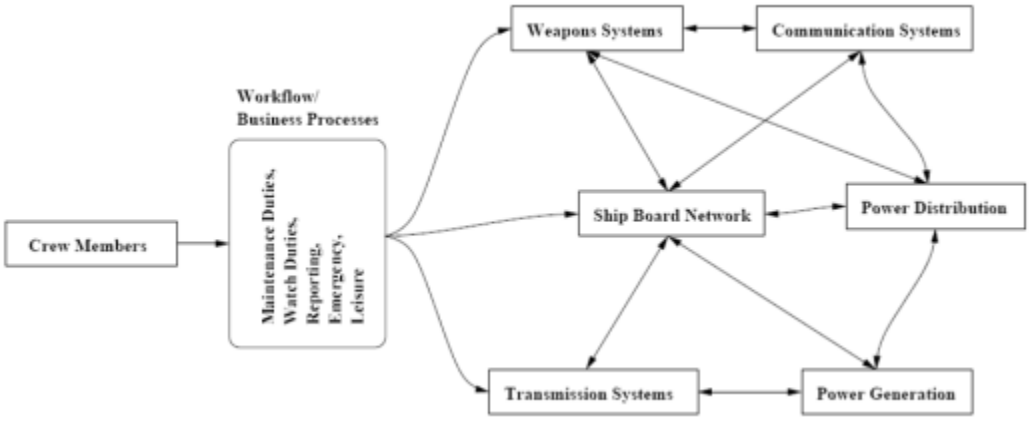


Figure 1-2. Navy Warfighter ships' system of systems (Mahulkar et al., 2008).

In this thesis, the modeling of a wireless network in the ship environment and the validation of this model are the main focus of the research. This model is then integrated into the ship SoS model to enable interactions among all entities on the ship and to

evaluate the impact of the interactions between all entities in the ship. The ship becomes a network centric system where all entities are connected in a robust network, thus increasing the situational awareness of the crewmembers. For example, when a fire emergency occurs, the sensors in the room would trigger the fire alarm and inform all agents about the situation. Furthermore, the workflow of the agent would be analyzed and two of the least utilized agents in the area would be assigned for the fire fighting task. This shared situational awareness enables self synchronization between the crew members and shortens the chain of command for emergency protocols, hence, increasing the operational effectiveness of the ship.

The interdependence among large-scale, distributed systems has increased due to the advent of modern telecommunications. This increase in interdependence among complex systems has increased the risk to systems from disturbances that rapidly propagate through networks causing damage to critical infrastructure and processes. Military and defense industries have used the metric of survivability as a criterion for determining the robustness of SoS to these kinds of disturbances. The Joint Technical Coordinating Group on Aircraft Survivability (2001) defines survivability as "the capability of a system and crew to avoid or withstand a man-made hostile environment without suffering an abortive impairment of its ability to accomplish its designated mission. Survivability consists of susceptibility, vulnerability, and recoverability." Whereas robust systems are able to accommodate permanent changes in the system, such as a torpedo impact of the ship, survivable systems are able to recover from finite changes in their infrastructure, such as due to a wireless router glitch or temporary communication failure. Therefore, survivability can be considered a special case of robustness.

1.3. Problem Statement

The main objective of this thesis is to develop a model and simulation for evaluating the operational capability of a Navy ship, which contains a wireless communication network through which the crew interacts. The simulation must take into account the SoS

interactions throughout the ship environment and must consider the effects of normal operating scenarios as well as emergency scenarios, such as compartment fires and equipment failures.

The following research questions are investigated:

1. How can a wireless network be modeled and simulated along with the crew workflow on a Navy DDG class ship?
2. Using this wireless network, how can its configuration of access points and overall structure be optimized to satisfy the optimized manning requirements?
3. How can the robustness of the wireless network configuration be evaluated in the modeling and simulation environment in the event of emergency scenarios?

1.4. Scope of Research and Methodology

A Modeling and Simulation (M&S) Verification, Validation, and Accreditation (VV&A) Implementation Handbook published by the United States Navy describes the process for developing simulation models and performing verification, validation, and accreditation of these models for the United States Navy. The M&S process encompasses the entire M&S lifecycle from requirements definition through development, use, and support. VV&A complements the M&S process by gathering and examining its products to make an informed decision regarding the use of the M&S for a specific purpose. For successful VV&A, M&S development requires parallel VV&A activities during design, implementation, testing, and analysis of M&S results. The primary purpose for conducting VV&A is to establish credibility and confidence in the use of the M&S results. Figure 1-3 summarizes the whole process of M&S and VV&A.

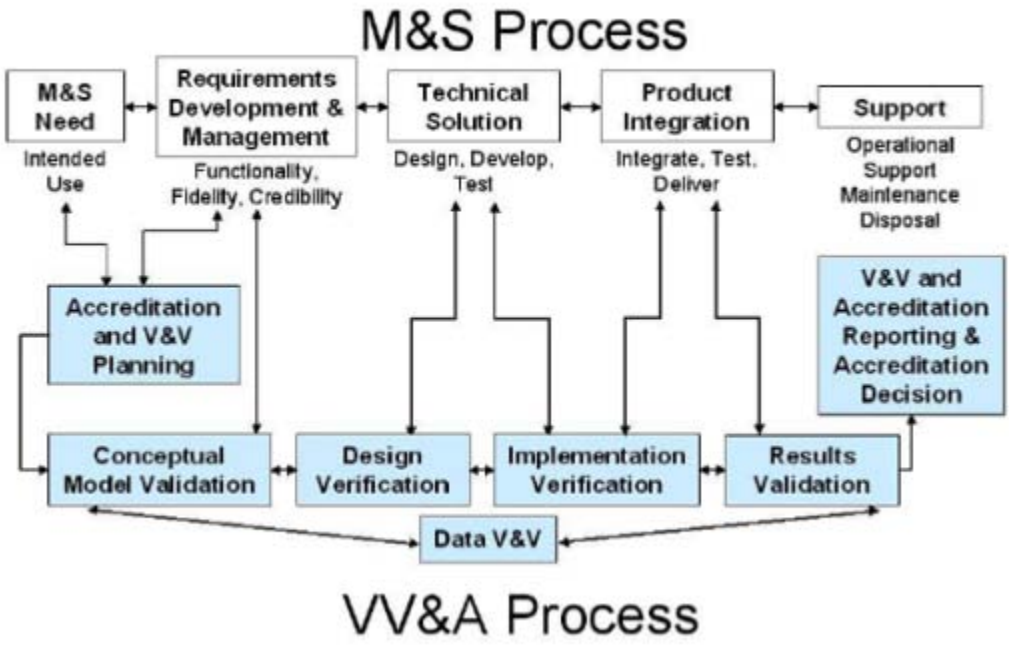


Figure 1-3. Modeling & Simulation, Verification, Validation, & Accreditation process of U.S. Department of the Navy.

Table 1-1 below summarizes the activities that were performed in this research to align with Navy VV&A requirements. The need for modeling and simulation that is relevant to naval platforms requires that technical requirements are specified based on an extensive literature review and interviews with subject matter experts (SME) from Simulex pertaining to technology insertion into a Navy warfighter. An earlier version of the Zonal Model was developed by Mahulkar and McKay; however, the model needed to be updated with a wireless network model to satisfy the need for verification and validation. Experiments were conducted at the former Burtsfield Elementary School in West Lafayette, Indiana to obtain comparison data that was used to improve the existing wireless network model and then verify and validate the results from the Zonal Model. Further accreditation processes would need to be addressed by Simulex in the future upon the integration of the Zonal Model into the Synthetic Environment for Smartship (SES) developed by Simulex.

Table 1-1. Research Framework (Adapted from US Dept. of Navy Modeling and Simulation Verification, Validation, and Accreditation Process, Navy MSMO 2004).

M&S VV&A Steps	Research Activities
M&S Need, Requirements Development & Management	Literature Review, Interview with Subject Matter Experts
Technical Solution	Updated Wireless Network Model
Product Integration	Updated Zonal Model
Support Accreditation and V&V Planning	Performed by Simulex and Crane
Conceptual Model Validation	Literature Review
Design & Implementation Verification	Experiments at Burtsfield Elementary School
Results Validation, Data V&V	Comparison with existing wireless model in the Zonal Model, Sensitivity Analysis
V&V and Accreditation Reporting & Accreditation Decision	To be performed by Simulex and Crane

1.5. Organization

This thesis consists of five chapters. Chapter 1 describes the problem statement and motivation of this research in terms of complex organizational decision-making to achieve optimal reductions in manning for Navy warfighters using new communication technology. Chapter 2 provides an overview of a Zonal Model that models the various systems aboard a Navy ship environment using an agent-based modeling formulation. Chapter 3 presents the modeling process performed for the updated wireless network model. Chapter 4 presents the results of Zonal Model simulations with the implementation of updated network model and analyzes the results of these simulations using sensitivity analysis and multiple objective optimization approaches to optimize the ship's operation. Chapter 5 summarizes the research conducted and discusses the contributions, limitations, and recommendations for future research related to work in this area.

CHAPTER 2. ZONAL MODEL (MODELING AND SIMULATION)

2.1. Agent Based Modeling and Simulation

Agent-based modeling (ABM) is gaining in popularity as a method to more accurately model the increasing complexity and interdependence among systems. Agent-based models are particularly popular in research conducted within the social sciences, which model the behavior of individuals and societies, and in military related simulations.

ABM consists of models of multiple entities that sense and individually respond to conditions in their local environments. The interactions amongst these entities produce complex large-scale system behaviors. ABM has been proposed for many situations involving a large number of heterogeneous individuals in various scenarios, such as vehicles and pedestrians in traffic, people in crowds, artificial characters in computer games, agents in financial markets, and humans and machines on battlefields. The aggregate behavior of the simulated system is the result of the dense interaction of the relatively simple behaviors of the individually simulated agents. ABM has two essential components: agents and their environment. Agents are problem-solving entities with well defined boundaries and interfaces, situated in a synthetic or virtual environment over which they have partial control and follow distinct rules. The utilization of agent-based models is practical for modeling the crew of a Navy ship SoS environment.

According to Bonabeau (2002), ABM is a mindset more than a technology. It consists of a description of the system from a perspective of its constituent units. An agent-based model is essentially a set of differential equations, each describing the dynamics of one of the system's constituents.

There are three major benefits of using ABM: (1) ABM captures emergent phenomena; (2) ABM provides a natural description of the system based on rules for individual entities; and (3) ABM is flexible. First, emergent phenomena result from interactions among the individual entities. When the behavior of an individual entity is nonlinear, it is difficult to describe this behavior using systems of differential equations and this is where ABM becomes most useful.

Second, ABM can handle individual entities whose behaviors exhibit learning and adaptation. Interactions among such entities are heterogeneous and lead to network effects that can cause significant deviations from the predicted aggregate behavior. A reinforced effect in the interaction between two agents does not necessarily produce a reinforced effect in the entire system.

Third, ABM provides the natural description of a system simulation in many cases. ABM works well to simulate the complex, nonlinear, discontinuous, or discrete interactions between entities. For example, it is more natural to describe how shoppers move in a supermarket than to derive equations that describe the complex dynamics of the density of shoppers. In this way, ABM enables the user to study aggregate properties and also optimizes the usage of data describing customer behaviors based on customer surveys, buying trends, etc., in the development of the model. Similarly, an ABM provides a more natural means of describing how crew members aboard a ship interact with one another using a wireless communication network. ABM has the flexibility for tuning the complexity of the agents: behavior, degree of rationality, ability to learn, evolve, and adapt, and rules of interactions.

ABM is also advantageous relative to conventional mathematical models because the dynamics of the SoS as a whole do not need to be well understood before the simulation is conducted. Instead, ABM uses the simulation to develop this understanding of the SoS dynamics. However, ABM has one significant disadvantage over traditional mathematical modeling approaches: “Each run in ABM could yield a sufficiency theorem,

but that does not provide any information on the robustness of the theorem. The only way to treat this problem is through multiple runs, systematically varying initial conditions or parameters to verify the robustness of results" (Axtell, 2000). Despite the ability of ABM to capture the emergent behavior of systems, it also becomes a challenge to analyze the modeling and simulation results due to the inability to predict what kinds of phenomena are expected.

2.2. Ship Infrastructure and Environment Modeling

A ship environment consists of complex interconnected systems such as the infrastructure, the crew members, and the workflow, which require careful systems planning. By studying real-time interactions between entities within such systems, one can make better decisions about the impact of new technology insertion on a ship. The Zonal Model described below is a platform developed by Mahulkar and McKay to model these types of ship systems (Mahulkar, 2006; Mahulkar, 2006; Mahulkar et al., 2008).

The simulation model is built based on the ABM methodology to simulate the workflow scenarios for maintenance, troubleshooting, and watch duties within zones of the ship; thus, the model is called a Zonal Model. The environment consists of crew agents modeled with limited intelligence and behavioral traits, machines with sensors, mobile and stationary network nodes, and models for data transfer over the network and crew mobility.

A simulation of agents' movements and interactions on the ship is run in a virtual environment that utilizes the floor plan of a single deck on a Guided Missile Destroyer (DDG) class Navy ship as shown in Figure 2-1. There are eleven zones in this deck and two zones of this deck are used as the virtual environment for the agents. According to Navy Program Executive Ships office (Office-of-the-Assistant-Secretary-of-Defense 2008), the latest ships have a crew size of 323, which consists of 23 officers and 300 enlisted crew. By distributing the crewmembers evenly in the ship's three-level decks and

zones, the current manning level is 9 to 10 crew members in each zone. The actual dimension of the ship is 142 meters (465.87 feet) long and 18 meters (59.05 feet) wide while the size of the ship layout image is 2155 pixels by 277 pixels. Thus, the conversion factor of 4.26 pixels per feet is used throughout the virtual environment for the adjustment of the locations of agents, machines, equipments, access points and the actual agents walking speed.

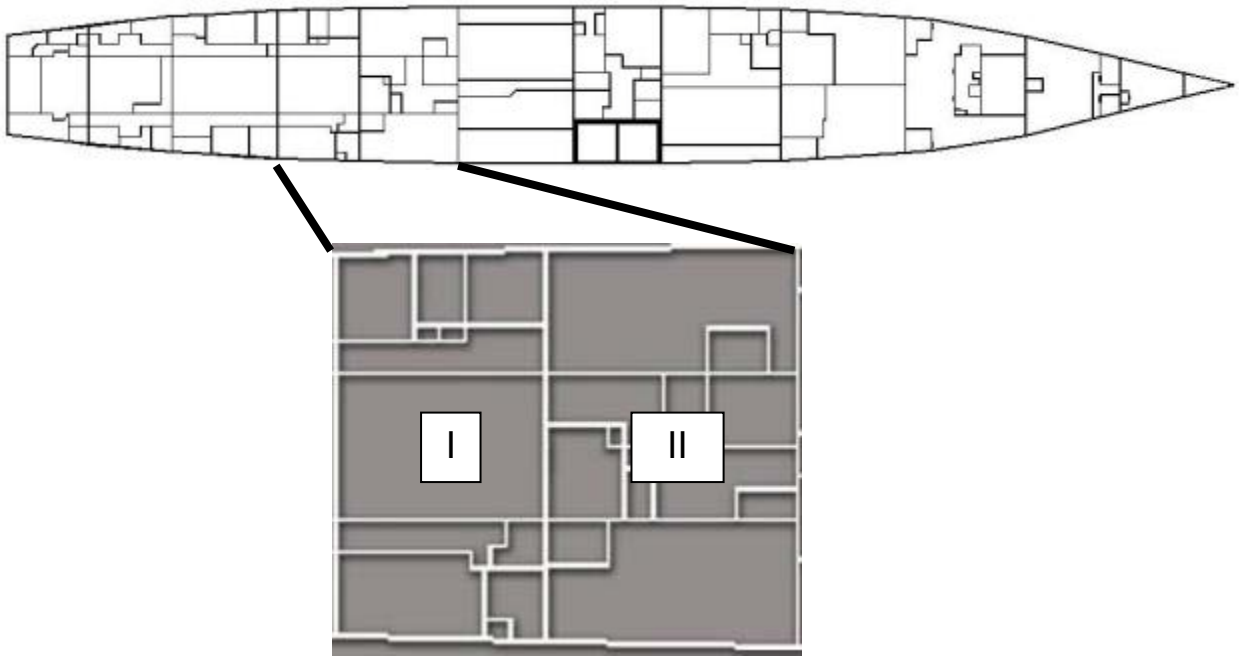


Figure 2-1. DDG floor plan layout.

The Zonal Model environment allows a user to load the ship layout image in TIFF format. Based on this layout, the location of doors and rooms can be automatically extracted. These locations are used as waypoints for crew members who navigate the virtual environment. The simulation scenarios in the Zonal Model are defined based on 7-10 crewmembers, 2-4 access points, 0-2 workstations, 1 server, 5 machines, 5 watch locations, 3 fire fighting equipments, and 4 miscellaneous pieces of equipment. In order to simulate ship operation, the models for the wireless network, crew, and emergency scenarios are developed to qualitatively describe the realistic behavior of the system. The following section outlines the wireless network model, the agents' model, and the

emergency scenarios model. Figure 2-2 shows the main panel of the Zonal Model, which is used to setup and launch the simulation.

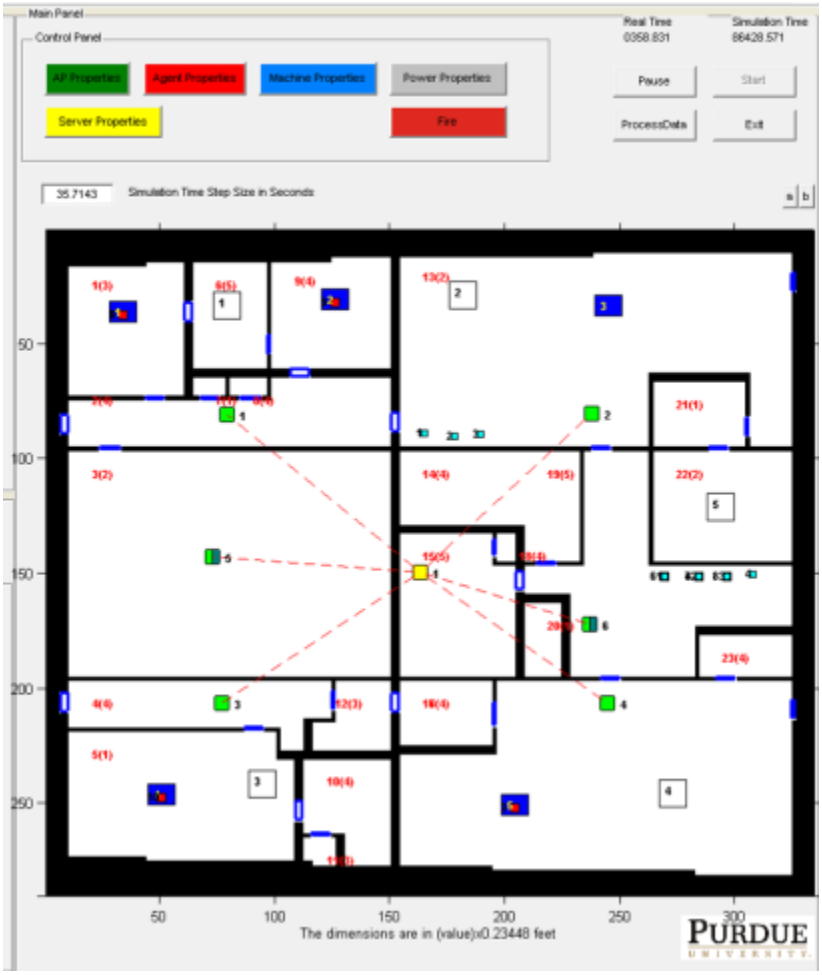


Figure 2-2. Simulation environment of Zonal Model.

Table 2-1. Legend Figure 2-2.

Legend	Description
Little red squares	Crew members
Green squares	Access points
Yellow squares	Server
White squares	Watch Locations
Blue squares	Machines
Little light blue squares	Fire fighting equipments

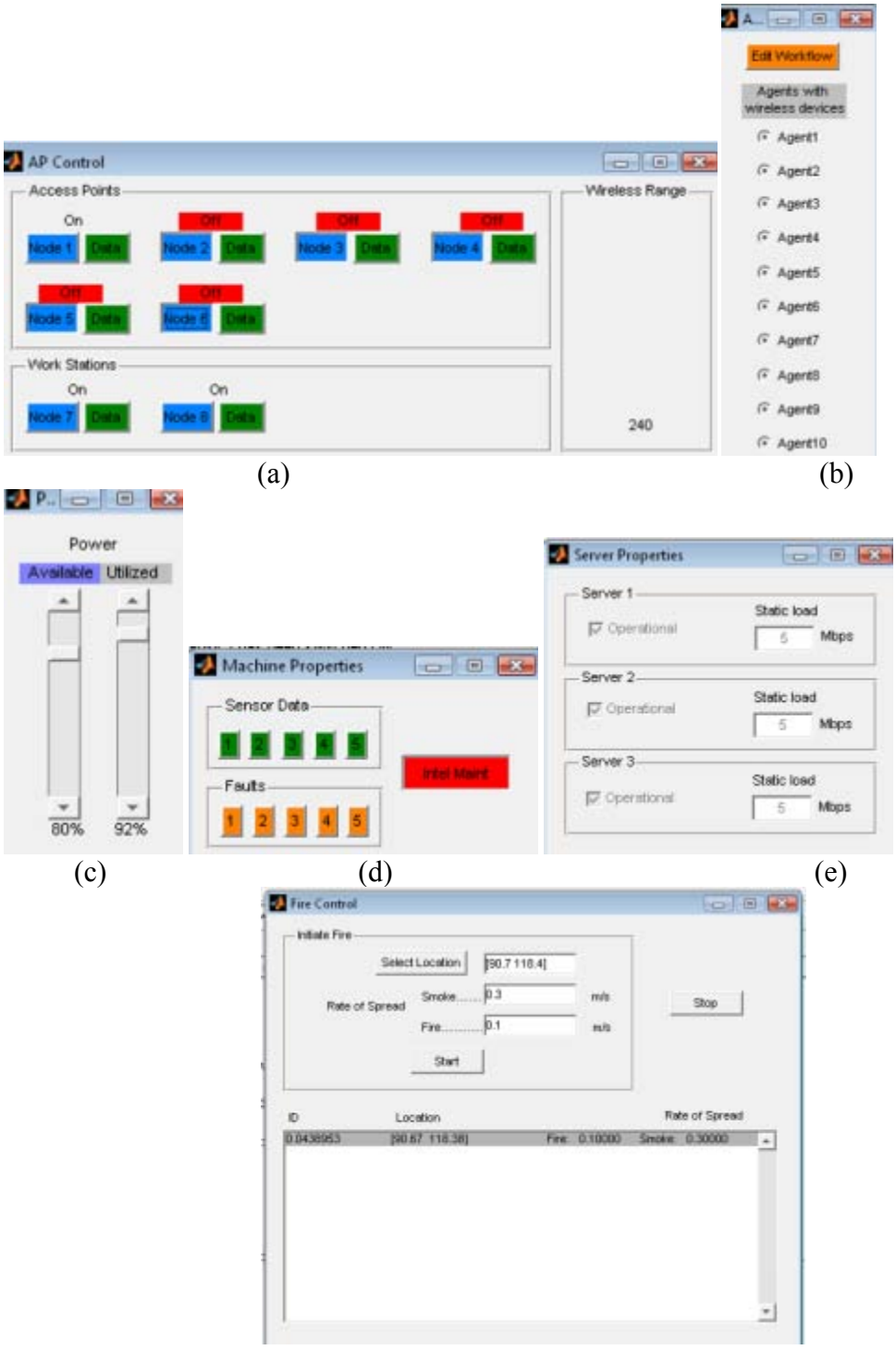


Figure 2-3. Zonal Model control panels and dialog boxes of: a) APs and workstations, b) agents, c) power status, d) machines, e) servers, and f) fire.

Figure 2-3 above shows multiple control panels available to the user of the Zonal Model for adjusting the parameters used in the simulation. Agents' PDAs or wireless devices are set to be available for each of them; however, during the simulation, a PDA failure can be implemented in the scenario. Control panels for machines, servers, and AP enable the user to manually turn on or off the functionality during a simulation run. The fire control panel enables the user to set the location of the fire and initiate fire. The details of the models that explain the interactions between the agents, machines, APs, servers, and the emergency scenarios are explained in the following sections.

2.3. Wireless Network Model

Currently, the United States Navy implements a wired local area network (LAN) in ships as the major means of communication. The objective is to identify the advantages/disadvantages of introducing new technology that is wireless through the modeling and simulation process before committing to the investment. The bottleneck in a LAN plus Wireless LAN (WLAN) combination is usually the WLAN because the LAN operates at a higher bandwidth compared to the WLAN. Hence, only the WLAN becomes the bottleneck in the data transmission process and will be examined in the current network model. An 802.11b backbone is assumed with a maximum data rate of 11 Mbit/s operating in the frequency range with center frequency 2.4 GHz. The typical indoor range is 30 m at 11 Mbit/s and 90 m at 1 Mbit/s. In the current Zonal Model, it is assumed that all access occurs within 30 m of an access point and the signal strength goes to zero beyond that range.

Before a data transfer is established, all clients compete for a connection. The client that gets connected is chosen randomly and all others are told to wait for the next round of competition. This process continues until no more clients are present (Bianchi, 2000). In effect, only one client is connected at a time to one channel or access point. The setup is that each agent would need to find free or unused APs in order to send data. Multiple accesses to a particular AP are not enabled in this model. Figure 2-4 illustrates the

network structure in the Zonal Model. No more than one crewmember is able to connect to any of the APs and initiate data transfer. The crewmembers marked with a red circle represent those that have to queue or look for other available APs before they can connect and initiate data transfer to the server.

Wireless sensor networks have been utilized to assist people in monitoring processes in various situations, such as medical/vital sign monitoring (Tia et al., 2005), structural integrity monitoring (Ning et al., 2004; Sukun et al., 2007), and machine monitoring (Kevan, 2006). Ning et al. (2004) developed a wireless sensor network system to assist in the acquisition of detailed data sets that record the response of different structures to ambient vibration caused by earthquakes, wind, or passing vehicles, or forced excitations delivered by large shakers. Currently, structural engineers use wired data acquisition systems to acquire such data sets. These systems consist of a device that collects and stores vibration measurements from a small number of sensors. However, power and wiring constraints imposed by these systems can increase the cost of acquiring these data sets, impose significant setup delays, and limit the number and location of sensors. Wireless sensors enable more sensing equipment to be deployed in building, or Naval ships, which guarantees continuous monitoring faults at a lower investment cost than wired sensors.

A previous project by British Petroleum developed a new predictive maintenance system capable of monitoring critical rotating machinery, such as the pumps and motors in the oil tanker's starboard engine room, using vibration data to evaluate operating conditions and wireless communications to send alerts when wear and tear was detected. The Loch Rannoch project produced an efficient automated data collection system for machine monitoring and predictive maintenance that eliminated many of the manual processes used in the past. Condition monitoring data from many types of rotating machines was typically captured by operators using handheld devices, resembling PDAs. In the conclusion of the project, sensor networks were found to work well in a hostile environment. The omission of manual inspection processes would help to detect early

equipment failures and could reduce a repetitive workload of the agents (Kevan, 2006; Li, 2006).

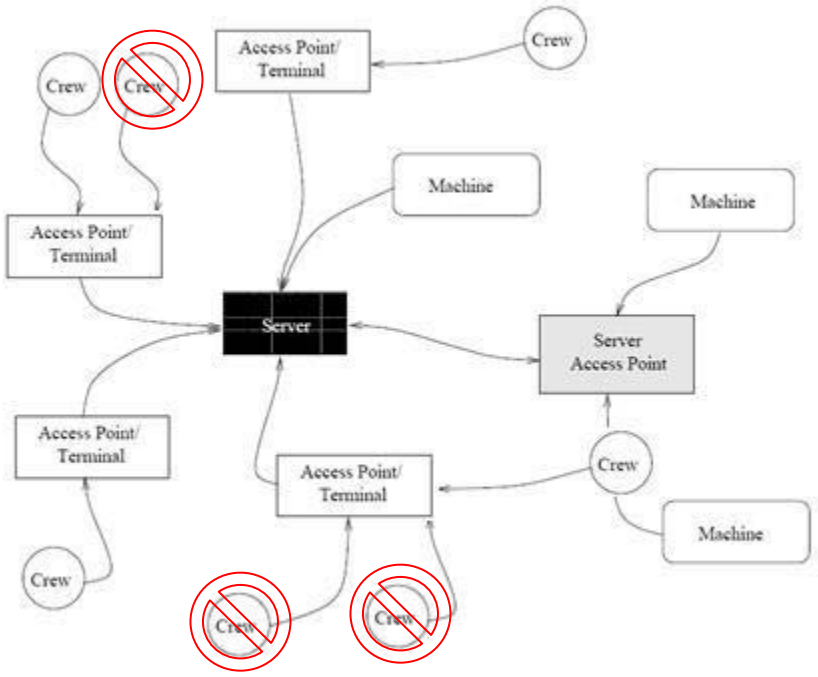


Figure 2-4. Network model (Adapted from Mahulkar et al., 2008).

The machines in the Zonal Model are connected to the APs by using wireless sensors to detect failures and allow early notifications for troubleshooting or repair to the agents before maintenance is scheduled. Some of the sensors commonly used in monitoring are temperature sensors, position sensors, pressure sensors, and vibration (acceleration) sensors. The number of sensors varies for different types of machines. If data from all sensors on all machines are being transmitted, this will lead to a large load on the network, which in turn imposes constraints on the amount of sensor information that can be transferred over the network and the amount of signal processing that must be performed locally prior to data transfer. The wireless sensor network adds additional network loads. A static load is included to account for applications like distance support, which involves ship to shore communications on a regular basis. For the Zonal Model implementation, this load is assumed to consume 20% of the bandwidth available in the APs.

Priority on the scale of 1 to 5 can be assigned to particular data transmissions depending on their importance. For example, tactical information like radar or sonar data can be assigned the highest priority while web browsing can be assigned the lowest priority. The priority determines the amount of bandwidth allocated for a particular transmission. Higher priority transmissions are allocated higher bandwidths compared to the lower priority transmissions. Equal bandwidths are allocated for transmissions with equal priority.

2.4. Agents Model

There are numerous roles and responsibilities for crew onboard a navy ship aligned towards maintaining the integrity of the ship as well as achieving its mission. The model focuses on a small subset of all the different crew positions and ranks possible to simulate the effect of a hierarchical operational environment with multiple superiors and multiple subordinates at any level. Figure 2-5 visualizes one example of the simplified hierarchy. The Division Officer (DO) is responsible for assigning tasks to the crew as described in the Watch, Station and Duty Bill (2004). The Division Damage Control Petty Officer (DDCPO) is responsible for distribution of crew in case of an emergency. The Division Repair Parts Petty Officer (DRPPO) is responsible for ordering parts from the suppliers, managing the supply chain, and being a point of contact for unscheduled maintenance issues. Finally, the crewmembers are responsible for executing any tasks assigned to them including watch duties, regular maintenance and inspection as well as responding to emergency scenarios like fire and flooding.

The crewmembers aboard a navy ship are responsible for regular maintenance of space and machinery. They are usually assigned a weekly maintenance schedule, but because the simulation spans the time frame of 24 hours, the weekly schedule would be implemented as a daily maintenance schedule. Examples of maintenance of space include cleaning the floors, ladders, storerooms as well as the water tight compartments. Examples of maintenance of machinery include checking equipment and machinery on a

regular basis (Department-of-Navy, 2003). Upon completion of the tasks, a reporting process would be performed. Depending on the priority of the reporting duty, the crew could be required to report directly right after finishing the task or wait until finishing higher priority assignments before performing the reporting process.

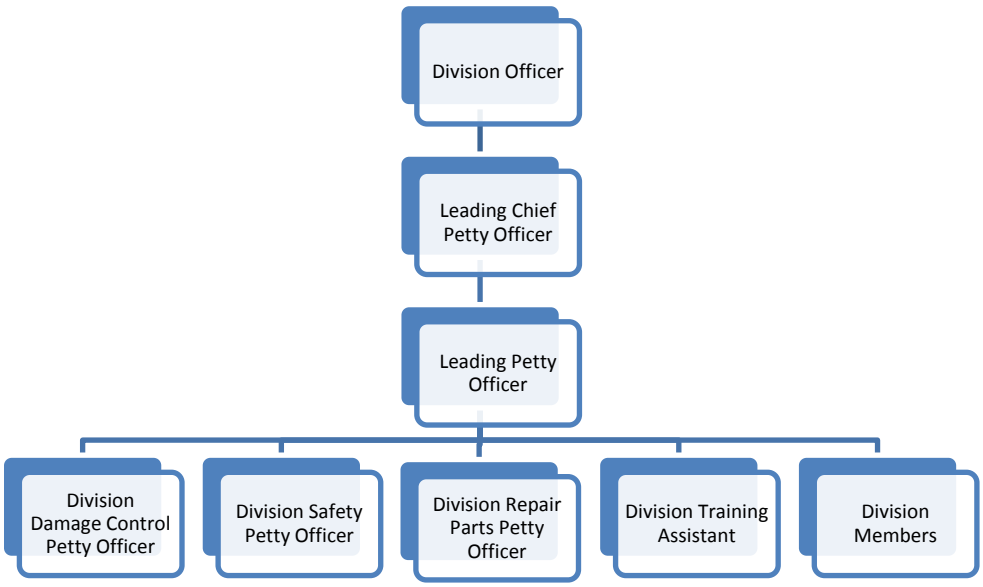


Figure 2-5. Division officer hierarchy (Stavridis and Girrier, 2004).

Chaturvedi et al. (2005) developed a conceptual representation and specification for ABM used for developing rule-based and event-driven behavioral models. The agents in the model are visualized as a double-helix DNA, one strand of which contains a finite list of attributes and limited intelligence, and the other strand contains the agent behavior resulting from a set of rules of engagement based on the interaction of the crewmembers and the infrastructure in the virtual environment. Agents have the following distinct attributes and properties that serve as rules of interaction within the environment: maximum walking speed, awareness of their location, machines, equipment, rooms, and APs, information of the assignments and tasks that need to be completed, and the exhaustion level conditions.

Figure 2-6 below shows the representation of various states an agent could occupy and the conditions for entering and exiting those states. The crewmembers will start from idle and receive work orders. They will then move to a new state. During the movement from one location to another in the zones, the agents navigate themselves using Dijkstra's algorithm by finding the shortest path between two rooms. The path from room to room has been established in the Zonal Model by establishing the wall, partitions, and doors from the imported ship layout. The speed of movement is limited by the maximum walking speed of 2 m/s.

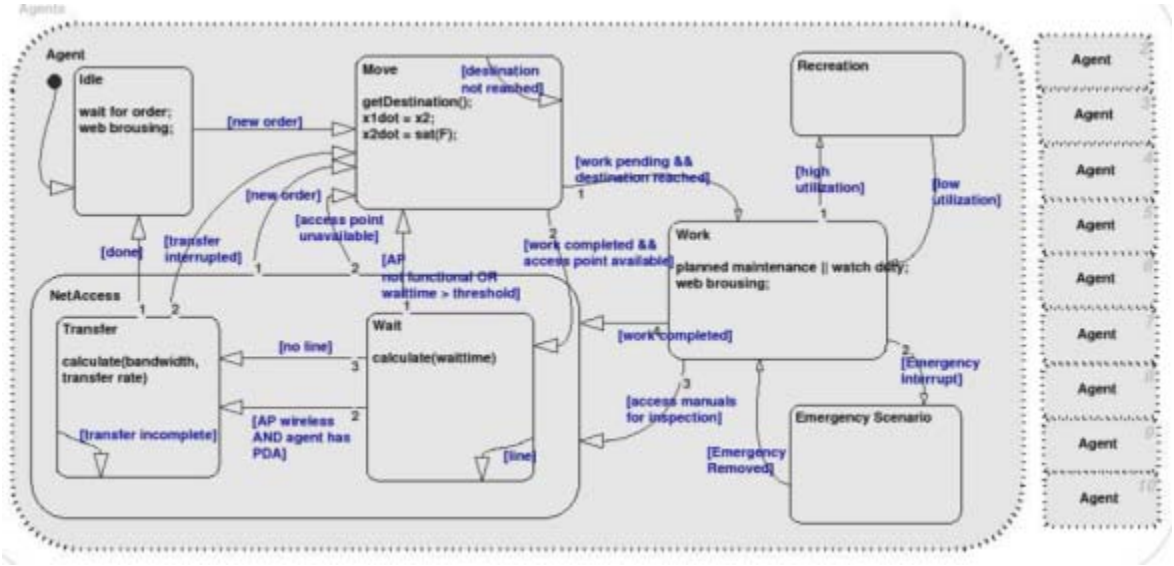


Figure 2-6. Agents intelligence and behavior workflow (Mahulkar et al., 2008).

A study by Mahulkar et al. (2008) on how to improve crew efficiency and reduce manning requirements onboard ships using wireless network technologies results in the introduction of a strategy for scheduling and reassigning crew workflows. They demonstrated this improvement using a scenario in the Zonal Model implemented with a specified number of failures, preventive maintenance jobs, and watch duties over a period of 24 hours in one section of the ship. In general, the initial agent utilization is relatively high at 80% whereas it is more desired to maintain the crew utilization around 60% to avoid overworking some of the crewmembers and ensure that agents maintain top performance during each task. A rescheduling strategy was used to minimize the variance

of lengths of workflows of the crewmembers by allocating recreational activities to crewmembers with high utilization. The algorithm also reassigned tasks from crewmembers with a large backlog to other crewmembers so that all crewmembers were equally utilized. This rescheduling mechanism has been established as the default setting in the development of the modified Zonal Model.

2.5. Power Generation and Trimming Model

During the standard operation of Navy ships, power distribution plays a vital role in ensuring the interactions between crewmembers and the infrastructure they utilize goes according to plan. Therefore, there is a need to develop and implement a dynamic simulation of the electric power subsystem of a Navy warfighter, especially the DDG class guided missile destroyer. Wasynczuk et al. (1997) developed a DC zonal electric distribution system (ZEDS) architecture for future shipboard applications. This system includes alternator/rectifier power sources, a dc distribution system, and distributed dc/ac inverters for zonal AC loads. System stability, fault tolerance, and detection were the main objectives in the development of this model.

The simulation of actual shipboard power distribution models has shown that transients occur on the order of a few milliseconds to a few seconds (Walters, 2001). Since the data transfer time is measured in seconds and an SoS approach focuses on a high level view of the system, a simplified model of power redistribution and trimming was created based on the model mentioned above. The details of the model are described below according to Walters (2001):

- "The amount of power generated and available for this zonal environment can be controlled using the interface provided to the user or can be varied as a function of time.
- Each of the rooms is assumed to consume a predefined amount of power. If the required amount of power is not available, all equipment in that room is taken off line.

- Priorities are also assigned to each room depending on their importance.
- If the power level falls below the amount required to power all rooms, power trimming is performed starting with rooms having lowest priority as is done in reality on a ship.
- The network equipment is assumed to be powered over the Ethernet, i.e., the wireless access points are powered over the Ethernet cable by the server to which they are connected. A power outage in that zone of the ship does not necessarily affect the network equipment unless the server is affected." (Mahulkar et al., 2008)

2.6. Emergency Scenarios

The zonal model simulates numerous crisis scenarios and models the impact of such events on the crew utilization, network utilization, power utilization and overall readiness of the ship. One scenario consists of a fire starting in one zone of the ship and spreading throughout the ship until fire fighting crew intervened to bring the fire under control.

In the fire and smoke scenario, the user has the capability to initiate and terminate a fire at any location within the zones of the ship at a specific time during the simulation. Once the fire model is initiated, it starts spreading in all directions at a rate of 1 foot per second. The smoke is usually set to spread faster than the fire with the smoke propagation rate set at 2 feet per second. As soon as a fire scenario has been started, two agents working close to the incident would leave their current work, obtain the fire fighting equipment, and fight the fire. During the fire fighting process, fire recedes at a rate of 2.6 feet per second and smoke recedes at a rate of 0.7 foot per second.

According to Daley and Gani (1999), existing fire growth models assume that the growth of a fire is a function of the amount of unburned inflammables and the rate with which inflammables catch fire. The model includes the feature of the combustibility of each elementary item, and once a unit is burned out, it cannot burn again. Both ignition and burn-out rates have random fluctuations during a fire and from fire to fire. In the Zonal

Model, the rates of fire and smoke propagation are simplified based on the simulation created by Brazilian Navy officer and United States Naval Postgraduate School students (Andrade et al., 2002). Utilizing the fire growth model by Daley and Gani, they created an agent simulation that addresses the fire fighting situation. The average results of fire and smoke propagation rate as well as the fire fighting rate shown above are used in the Zonal Model. Figure 2-7 below shows the visualization of agents fighting fire and smoke in the Zonal Model simulation. A dark purple circle represents the fire and the light purple circle represents the smoke. In the fire model implemented, the possibility of injured or loss of the agents is not considered; thus, the assigned duties of each agent would be maintained unless reassignment occurs to assist in the reduction of crew utilization.

Another emergency scenario analyzed in this project is the wireless AP failure. The failure would likely occur in the case of gun shock and ship collision. It is also possible that the APs have performance loss due to exposure to extreme heat and humidity, commonly in a fire and smoke situation. During a simulation, the AP failure could either be manually switched on or automatically inserted in the workflow. When it occurs, the AP is completely inaccessible and the agents have to search for other available APs or workstations.

2.7. Summary

This chapter presented ABM as a new approach in modeling naval ship SoS with several advantages. ABM captures emergent phenomena and provides a natural description of the system based on rules for individual entities. ABM allows the dynamics of the naval ship SoS to be captured during the simulations; however, extensive numbers of simulation runs are required to verify the robustness of the results.

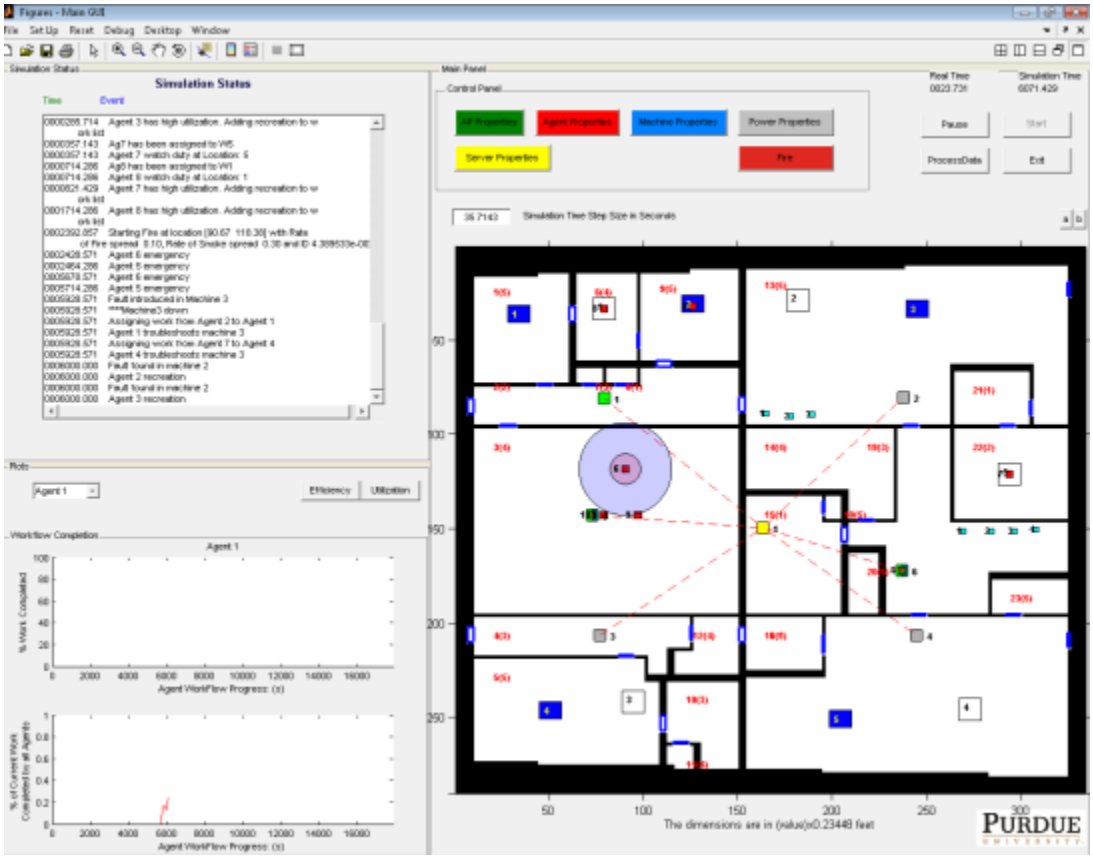


Figure 2-7. Fire scenario initiated during simulation.

A Zonal Model was created that integrated complex interconnected systems, such as the infrastructure, the crewmembers, and the workflow to understand the impact of new technology insertion on a ship. The simulation model was built based on the ABM methodology to simulate the workflow scenarios for maintenance, troubleshooting, and watch duties within zones of a DDG class ship. The environment consists of crew agents modeled with limited intelligence and behavioral traits, machines with sensors, mobile and stationary network nodes, and models for data transfer over the network and crew mobility.The existing models for the wireless network, crew, and emergency scenarios have been developed to qualitatively and quantitatively describe the realistic behavior of the system.

CHAPTER 3. WIRELESS NETWORK MODEL

3.1. Wireless Network Technology Overview

Wireless Local Area Networks (WLANs) have been developed to provide high bandwidth communications to users in a limited geographical area. The WLAN has been proposed as an alternative technology for mitigating the high installation and maintenance costs incurred when traditional wired LAN infrastructures are modified. Thus, the WLAN has been widely deployed across the enterprise, home, and public environments sectors because WLANs offer several advantages such as the elimination of wiring around the building, roaming, support for several types of services provided to a large number of users, the flexibility of relocating equipment, and easier maintenance. Current standards, such as IEEE 802.11g, provide a throughput of 54Mbps while operating in the 2.4 GHz band (Ferro and Potorti, 2005; Kuran and Tugcu, 2006).

The IEEE 802.11 WLAN is a communication protocol standard that defines a physical layer (PHY) and a medium access control layer (MAC) for wireless communication within short range, from a few meters up to 100 meters with low power consumption, from less than 1mW up to 100mW. It is also a data transmission system designed to provide a location-independent network usually implemented as the final link between the existing wired network and a group of client computers, giving these users wireless access to the full resources and services of the corporate network across a building or campus setting. A WLAN is based on a cellular architecture; a typical WLAN configuration consists of Access Points (AP) and users connected to these APs. However, there is the possibility that the stations communicate with each other directly in an ad-hoc fashion. The available bandwidth is divided into 14 channels where only channels 1, 6, and 11 are not overlapped. The 802.11 specification for wireless LANs was ratified by

the IEEE in 1997, which specifies the characteristics of devices with a signal rate of 1 and 2 Mbps. In 2003, the IEEE approved 802.11g as a further evolution of the 802.11 standard. Some of the signal rates provided by 802.11g vary from 6 and 54 Mbps while working in the 2.4GHz band. Throughout this paper, these signal rates are referred to as the signal rate modes of the access point (AP). This standard guarantees compatibility with other 802.11 standards such as 802.11b.

It is well known that the IEEE 802.11g standard switches data rates by means of different modulation schemes from one signal rate mode to another according to the wireless propagation conditions using the signal strength as a performance metric, i.e., the minimum signal strength required to handle arriving frames at a given data rate. The performance of a communication link can, therefore, be estimated by knowing the received signal strength from the AP. Furthermore, the performance of the overall network on a Navy ship could also be estimated given these signal strengths. This concept is the basis of the experiments conducted herein and the models of the wireless system that can be used in the subsequent simulations to evaluate ship wireless network performance.

3.2. Wireless Network Modeling

A key point for ensuring proper integration of WLAN technology into the ship system is to optimize the placement of the APs according to the operational scenario conditions and the needs of the user. The terms “optimal configuration” refers to the AP configuration that provides a maximum data transfer rate capacity to the majority of the users using the minimum number of APs with 100% wireless signal coverage. For this purpose, the relationship between the received signal strength (RSS) and the data rate that one AP can serve according to that RSS must be known.

There are two major factors that affect the RSS / data rate relationship in a particular location: the channel being used and the mode that the AP is operating on. Theoretically,

each signal rate mode has different RSS intervals, but for a given RSS value the corresponding data rate may be different for each transmission due to the factors mentioned above, commonly referred to as the variation of the data rate. It becomes critical for the performance of the network to anticipate and to locate the areas where the variation component is larger, and thus, avoid these areas of large variation.

Various wireless modeling approaches have been proposed in the literature (Rappaportet al., 1991; Andersen et al., 1995; Panjwani et al., 1996). Propagation mechanisms are very complex and diverse; signals propagate by means of diffraction, scattering, reflection, transmission, and refraction. Indoor environments do not guarantee direct line-of-sight propagation between the AP and the user leading to a notable signal loss phenomenon called shadowing. This phenomenon occurs because the diffracted field can reach a receiver even if it is shadowed by an obstruction. Therefore, it is essential in the design of any wireless system to characterize the propagation method and develop a model. This model describes both temporal and spatial channel responses. Channel modeling is useful for characterizing the behavior of the RSS to anticipate the approximated data rate value that will be obtained. A model for estimating the mean path loss $g(a_j, r_i)$ is given by:

$$g(a_j, r_i) = g(d_0) + 10 \times \eta \times log\left(\frac{d(a_j, r_i)}{d_0}\right) [dBm] \quad (3.1)$$

where $g(d_0)$ is the path loss at reference distance d_0, $d(a_j, r_i)$ is the distance from the transmitter, and η specifies the path loss behavior for a particular type of building that will be estimated to fit the actual data from the experiments. This model is a distance-dependent path loss model and does not depend explicitly on the location of partitions or obstructions in the proximity.

In order to obtain the values for parameters in Equation 3.1, a living laboratory was setup at the East Wing of formerly Burtsfield Elementary School in West Lafayette, IN. Figure 3-1 lays out the placement of the access points in the wing. The equipment used for the experiments were a Dell Inspiron laptop with Dell Wireless 1450 Dual Band WLAN

MINI-PCI Card, Linksys WAP54G access points, Linksys EtherFast Cable DSL Router, and 3Com Baseline Switch.

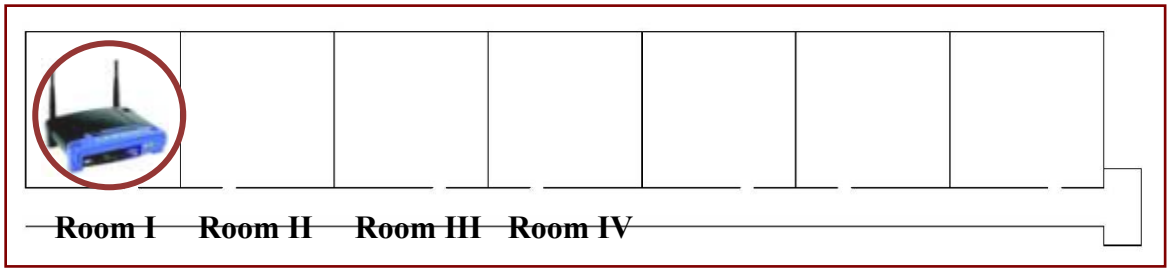


Figure 3-1. Formerly Burtsfield Elementary School wing layout.

An access point marked by the red circle in Figure 3-1 is used in the experiments to find out the path loss coefficient of the environment. Using a laptop, the received signal strength of various locations in the wing is measured. Figure 3-2 below shows the relationship between signal strength and the distance between the user and the access point. The blue markers indicate the real data points from experiments and the red markers are the estimated signal strengths as calculated using Equation 3.1. The computed η value is 2.1, which is bounded by the lower limit of 2 (for free space) and upper limit of 4 (for flat earth).

After mapping the RSS to the distance of a user from an AP, the corresponding data rate must be calculated for each particular value of signal strength. Two sets of experiments with a single user and multiple users sending data to server were performed. The effect of multiple users accessing the AP in the same time would be captured to properly adjust the corresponding data rate for a certain situation. For every experiment, a 100Mb file transfer was initiated by the users to the server in order to obtain a more consistent data transfer rate. A constant data rate of 2863.2 kBps was found for a signal strength from -30dBm to -59 dBm and was 389.68 kBps for a signal strength lessthan -76dBm. For signal strengths between -60 dBm and -76 dBm, the data rate was estimated using Equation 3.2 below:

$$DR = DR_{min} + rand\# * (DR_{max} - DR_{min}) \quad (3.2)$$

where DR_{min} is the minimum data rate for a corresponding signal strength recorded during the experiment, DR_{max} is the maximum data rate for a corresponding signal strength recorded during the experiment, and $rand\#$ is a random number between 0 and 1. The random number generator is used because of the uncertainty in the data transfer rate for its corresponding signal strength levels. A second-order polynomial curve fit was also performed but the random probabilistic model fit the actual data better. The comparison is presented further in Figure 3-5.

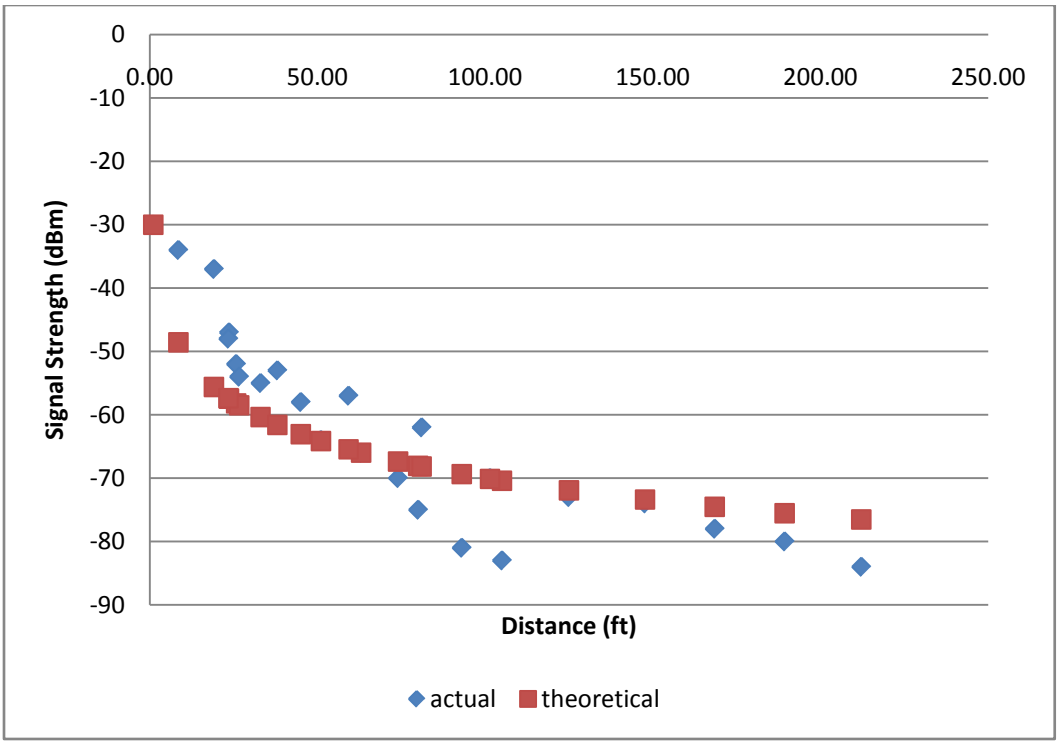


Figure 3-2. Received signal strength as a function of distance of one user from a single access point.

For the experiments with multiple users accessing the AP, the data rate that each user gets is inversely proportional to the number of users sending files through this AP. Based on the observation of the results shown in Figure 3-3, it is deduced that as the number of the users increases, the data rate for each user would be the maximum data rate at that location divided by the number of users accessing the AP and sending data. Room I has the AP inside and Room II, III, and IV are the adjacent rooms with no AP.

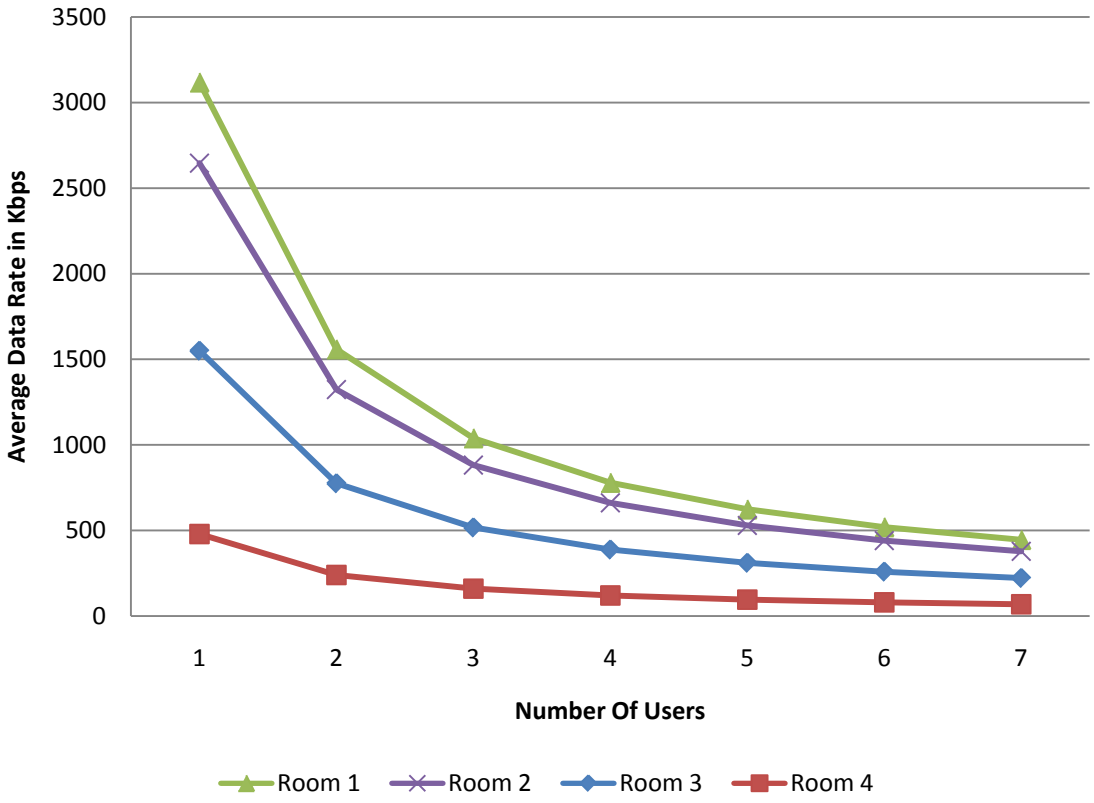


Figure 3-3. Average data rate for multiple users accessing one AP

The hybrid automaton that summarizes the model for each AP is shown in Figure 3-4. According to the hybrid automaton, in order to determine the mode that the AP will operate on, the current signal strength received by the user must be known according to the separation distance between both entities. After the separation distance is computed and the path loss is estimated using Equation 3.1, the simulation uses the hybrid automaton and switches among the modes as the condition changes.

Figure 3-5 shows the comparison of the relationship of signal strength and data rate of the experimental data, polynomial curve fit, and random probability model. From Figure 3-5, it is observed that for a signal strength higher than -60 dBm (region I) and lower than -76 dBm (region III), both the curve fit model and random probabilistic model work well producing small error. For a signal strength ranging from -60 dBm to -76 dBm (region II), the random probabilistic model, even though it performs better than the curve fit

model, shows variability in the data rate estimation. Overall, both of the models work best in regions I and III.

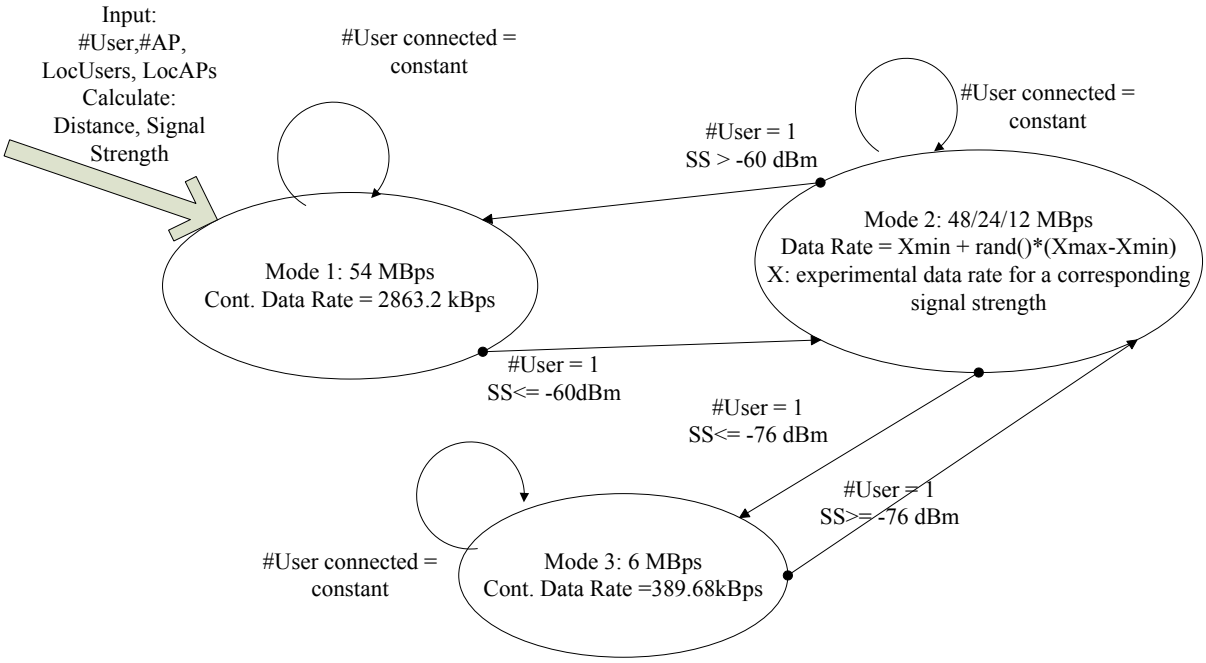


Figure 3-4. Hybrid Automaton for various modes in wireless data transfer.

For further analysis, a Matlab and Excel simulation was created to demonstrate the capability of the model developed for predicting the distance – signal strength – data rate mapping. Using the curve fit model, for a consecutive data transfer of 100MB at 24 different locations, the total actual time from experiments at former Burtsfield elementary school was found to be 1771.7 s and the model estimated 1581.5 s. The error was 190.1 s

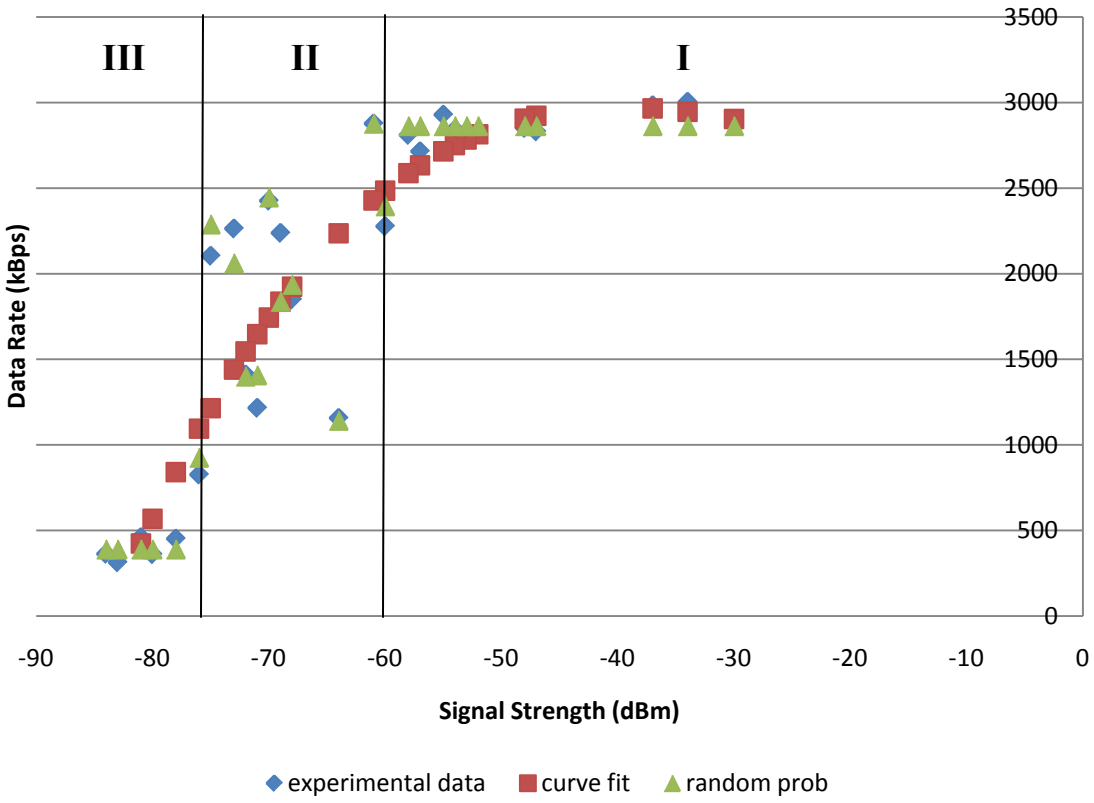


Figure 3-5. Experimental data rate vs. signal strength.

From the results of the simulation using the random probabilistic model, as shown in Table 3-1, it was found that the average error in time was 51.9 s. Due to the nature of the random number generator, the estimated time was different during every run; however, it could be concluded that the magnitude was much lower compared to the magnitude obtained using the curve fit model. It is clear that the random probabilistic model has a better performance than the curve fit model.

Table 3-1. Random Probabilistic Model Results.

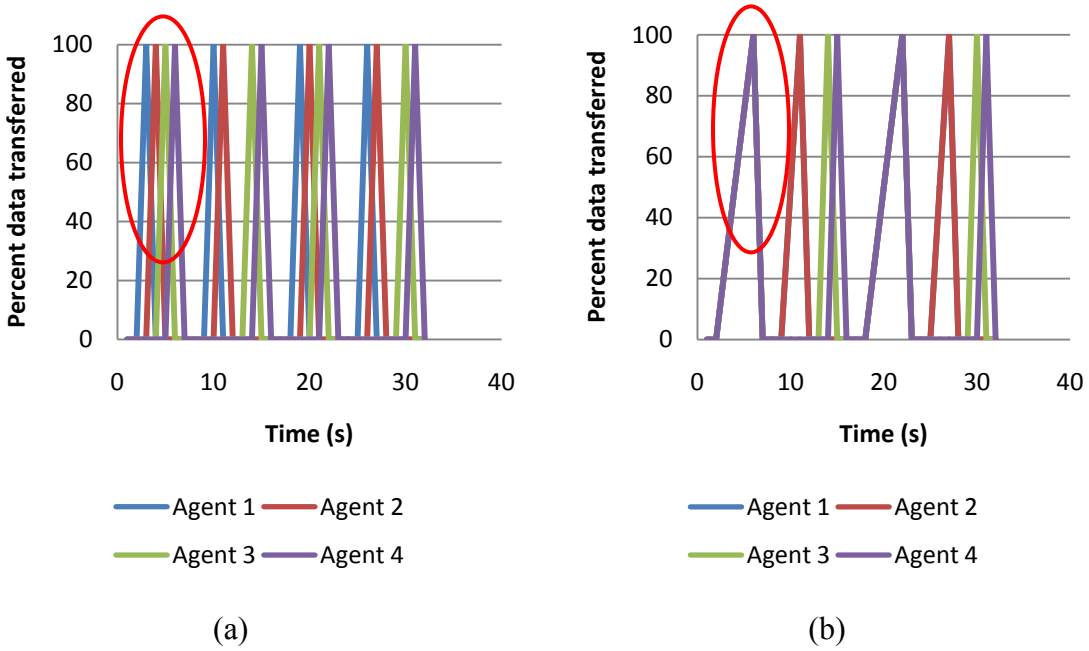
Actual Time (s)	Est. Time (s)	Error (s)
1771.66	1807.87	36.21
1771.66	1842.60	70.94
1771.66	1825.99	54.33
1771.66	1810.50	38.84
1771.66	1830.83	59.17
	Average	51.90

3.3. Comparison with Existing Wireless Network Model

The performance of the updated wireless network model must also be compared with the existing model to quantify the change in the agents' workflow and overall network performance. The existing model only allows 1 agent to access 1 AP at a time. The coverage range for an AP is 30 feet, the signal drops to zero beyond that boundary and a fixed data transfer rate of 10 Mbps is provided when the agent is in the range of an AP. However, the updated model has far more complex dynamics than the existing one as explained in the previous section. Figure 3-6 shows a comparison of results in an environment with 1 AP. At a simulation time of 4 seconds, 4 agents initiate 2 Mb file transfers and the existing model does not allow multiple access so that each agent will need to wait until the AP is available; thus, the process for data transfer is delayed. In the updated model, all agents experience the same data transfer rate that is given to all agents because they are in the same region I (refer to previous section), which yields the maximum data transfer rate.

Figure 3-7 shows the comparison of results in an environment with 2 APs. At a simulation time of 2 seconds, 4 agents initiate 2 Mb file transfers. Agent 4 is located in region II of the updated model and able to initiate data transfer with a lower data rate while agent 4 is out of the network in the simulation using the existing model; therefore, agent 4 must walk to the nearest AP with the existing model before being able to initiate

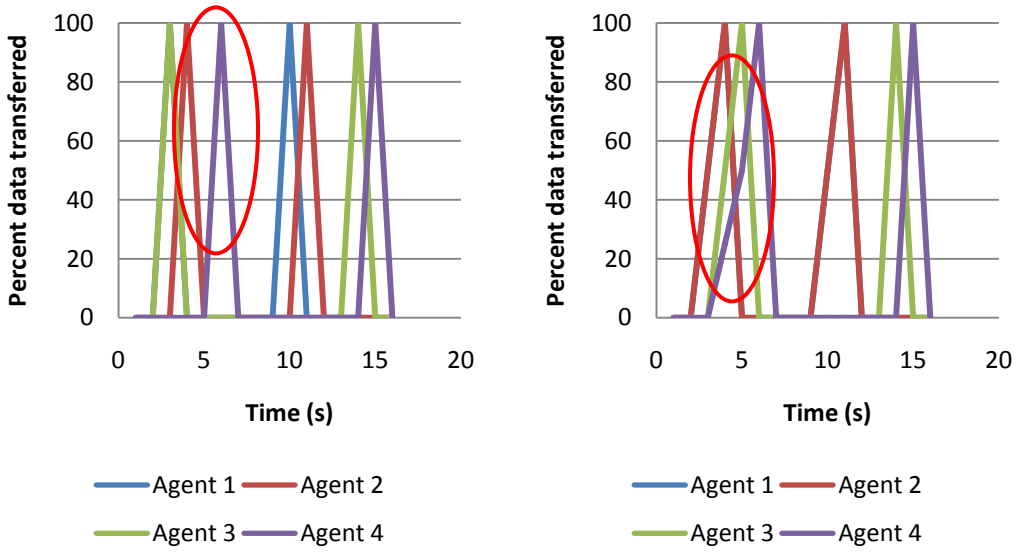
data transfer. The data transfer time for the updated wireless network model is longer than that of the existing model but it does not mean that the updated model is inferior to the existing one. The updated model represents a more realistic behavior of the wireless network, which does not allow a fast data transfer as described by the existing model.



(a) (b)

Figure 3-6. Comparison of data transfer rate from 4 agents accessing 1 AP: a) existing model and b) updated model.

Depending on the granularity of the simulation time of the other interacting models, the results from the existing model and the updated model are very similar. However, the updated model is superior in enabling multiple agents to access a single AP or multiple APs in the same amount of time. Further implementation of the updated wireless network model will be discussed in the next chapter.



(a) (b)

Figure 3-7. Comparison of data transfer rate from 4 agents accessing 2 APs: a) existing model and b) updated model.

3.4. Access Point Placement Optimization

The variation in data transfer rate is caused by the distance and obstructions between the user/receiver and the AP. Hence, the planning of the access point location would play a crucial role in the maximization of the transmission coverage and communication throughput. Good placement of the AP location would certainly result in wide-area transmission coverage and the maximization of the coverage area with a high transmission rate. However, the complexity in the design of an indoor environment, which contains various types of obstacles, generally leads to a difficult planning and placement problem. In practice, the system designer must survey the location in order to take all necessary readings prior to the actual system installation. The survey is a costly and time-consuming process; therefore, in order to eliminate some of the costs involved in the process, the use of a computer-aided planning technique is one of the most widely used approaches. However, before a simulation to study optimization can be carried out, more characteristics of the hand-off process that occurs as agents move throughout the environment with multiple APs is required.

Another set of experiments was performed at formerly Burtsfield Elementary School to identify the hand-off requirements for a user connected to an access point to switch to another access point of the same network. Three APs were placed with different channel settings in the same network separated by equal distances. When a user is connected to an access point with signal strength worse than -70dBm and there is another access point with signal strength better than -45dBm, the user will switch to the latter access point. The APs were positioned so that when the users need to send or receive data from the server, they should switch to the access point with better signal strength satisfying the conditions above.

As visualized in Figure 3-5, for optimal performance, it is suggested to place the laptops or workstations in regions I and III, due to variation issues in the data rate, than in region II. The ideal situation, however, in terms of data rate is to place the APs in areas such that all the workstations under coverage can exploit the maximum data rate available, i.e., region I. However, this strategy implies the usage of a large number of APs because the region I coverage is only guaranteed in the close proximity of the AP.

Another issue that needs to be taken into account in the design problem is the mobility of the agents. In scenarios with high mobility patterns exhibited by the agents, the network needs to be designed in a way that satisfies the hand-off process; particularly for the 802.11 protocol. This means that the hand-off requirements need to be guaranteed even if a loss of data rate is experienced. In the case when the agent is continuously moving (monitoring tasks), all the coverage areas need to ensure that a good hand-off process will take place, even if it is impossible to completely exploit the data rate of the APs, region I in particular. The hand-off process usually occurs when the user receives a signal strength lower than -70dBm and there is another AP in the proximity that gives a better signal strength.

The goal is to place the APs such that, when sending data, the users are located in areas with a signal strength better than –45dBm with the closest AP and worse than –70dBm

with the other APs. In order to guarantee the stability in the data rate, the previous requirements can be slightly modified by stating that all the workstations need to be located in areas where the signal strength is better than –45dBm with respect to one AP, and worse than –76dBm (region III) with respect to other APs, where one AP provides the data rate corresponding to region I and the other APs provide the data rate corresponding to region III.

In this situation, the capacity of the access point is not fully utilized because those areas with signal strength worse than –45dBm but still in region I (better than –60dBm) that provide high data rates are not considered. For instance, if the network configuration is designed in such a way that the users are connected to an AP with a received signal strength of –70dBm in the current location and none of the other APs within the coverage area provides a better received signal strength than –45dBm, even though this is better than –70dBm, the users will not switch to these available APs with better signal strength, and, thus, these users will fail to obtain a better data rate, simply because the network configuration does not satisfy the hand-off requirements, and they will still be connected to a further away access point, diminishing the performance of the network.

Nagy and Farkas (2000) proposed ideas using Genetic Algorithms to optimize the placement of AP implementing regression parameters related to the level of obstructions in the indoor environment. The main emphasis is in the adjustment of path loss of the wireless signal by subtracting the signal strength depending on the material and thickness of obstructions, e.g., concrete wall would attenuate the signal more then a glass wall. Genetic Algorithms result in more precision of the AP placement but it has higher costs in developing the problem formulation and is computationally more expensive. Wright (1998) managed to utilize the Nelder-Mead Simplex method in the AP placement optimization and it was reasonably efficient and reliable at finding local optima. The results and experiences with producing improved or optimal base station placements suggest that optimization can enhance the understanding of propagation models as well as improve wireless network system designs. Battiti et al. (2003) successfully utilized the

DIRECT (DIviding RECTangles) search aimed at maximizing signal coverage. This algorithm is a version of the Nelder-Mead simplex method, and it implements a pattern search algorithm that considers the minimization of the cost function. In particular, this algorithm is useful when the cost function might reach local minima during the search.

A simulation in Matlab was created to obtain the optimal placement of APs in an indoor environment based on the ideas gathered during the experiment at formerly Burtsfield Elementary School. The objective was to assist the user in deciding the placement of a set of APs in a virtual environment to ensure the minimum signal path loss to each of the agents in the room. The objective function was non-linear, discontinuous and might have local minima, so global, non-smooth optimization method was used. The Nead-Melder (NM) Simplex method was used to evaluate this problem since it was comparably easy to formulate and computationally less expensive compared to Simulated Annealing and Genetic Algorithms. However, if there were local minima, the NM Simplex would not be well suited for this scenario. The objective function was coupled with the constraint functions to create a pseudo-objective function that allowed Matlab's *fminsearch* to work effectively to identify the minima of interest. The step linear penalty method was used with a penalty multiplier value of 1. Matlab's *fminsearch* was used with default options for this problem (Arora, 2004).

$$f(a) = \frac{1}{M}\sum_{i=1}^{M} \min_{j} p(a_j, r_i) \tag{3.3}$$

$$g(k) = -a_j \leq 0, k = 1,2, \dots, 2 * j \tag{3.4}$$

$$\Phi(a) = \frac{1}{M}\sum_{i=1}^{M} \min_{j} p(a_j, r_i) - r_p \sum_{i=1}^{2*j} a_j \tag{3.5}$$

$$p(a_j, r_i) = p(d_0) + 10log\left(\frac{d(a_j, r_i)}{d_0}\right)^n \tag{3.6}$$

$$p(d_0) = 20log\frac{4\pi d_0 f}{c} \tag{3.7}$$

$$d(a_j, r_i) = \sqrt{(a_{jx} - r_{ix})^2 + (a_{jy} - r_{iy})^2} \tag{3.8}$$

$$\Phi(a) = \frac{1}{M}\sum_{i=1}^{M} \min_{j} d(a_j, r_i) - r_p \sum_{i=1}^{2*j} a_j \tag{3.9}$$

where a_j is the vector of x- and y-coordinates of the j[th] AP, r_i is the vector of x- and y-coordinates of i[th] agent, r_p is the penalty multiplier, M is the total number of agents, n is the path loss coefficient, d_0 is the reference distance (1 meter), f is the carrier frequency (2.4x 10^9 Hz), and c is the speed of light (3 x 10^8 m/s).

The detailed optimization setup is summarized in the equations above. Equation 3.3 is the objective function of the average path loss for all agents. Equation 3.4 is the constraint equation to ensure a positive value for the calculated locations of the APs. Equation 3.5 is the pseudo objective function that integrates the original objective function with the constraints using a linear penalty function. Equation 3.6 explains the path loss function as a logarithmic function of the ratio of the reference distance to the current distance from an agent to an AP. Equation 3.7 is the formula for computing the path loss at the reference point. Equation 3.8 is the basic formula for computing the distance between two objects.

The objective function was defined in decibels, but during the search, the algorithm occasionally encountered log 0; therefore, the pseudo-objective function used in the Matlab code was modified. The path loss equation was expressed as a function of the distance between the APs and the agents. Another possible solution was to enforce constraints that prevented the distance between the APs and the agents to be near zero; each access point could not be exactly placed in the "same" place as the agents. The former solution was used because the addition of extra constraints into the pseudo-objective function generally causes scaling problems. The constraint functions were used to ensure that the optimal solution would not be negative in value.

According to the existing Zonal Model, when two zones were used in the simulation, four APs would provide enough coverage for all agent operations. A set of AP placement optimization simulations for 10 agents with 2-4 APs was performed in a space the size of two zones in the ship and the results are shown in Figure 3-8. The agents are represented with blue circles and the APs are represented with red crosses. The circle around each AP

indicates that Region I of the signal strength to data rate mapping was the relevant region for the wireless transmission model and is desired by all agents. The larger the number of APs, the better the coverage area; however, the hand-off process would be less likely to occur in this case and the agent would be connected with an AP that provides a lower data rate. In this particular simulation, all agents are not moving with respect to the APs and further observations will be made during the implementation of the wireless network model in the Zonal Model.

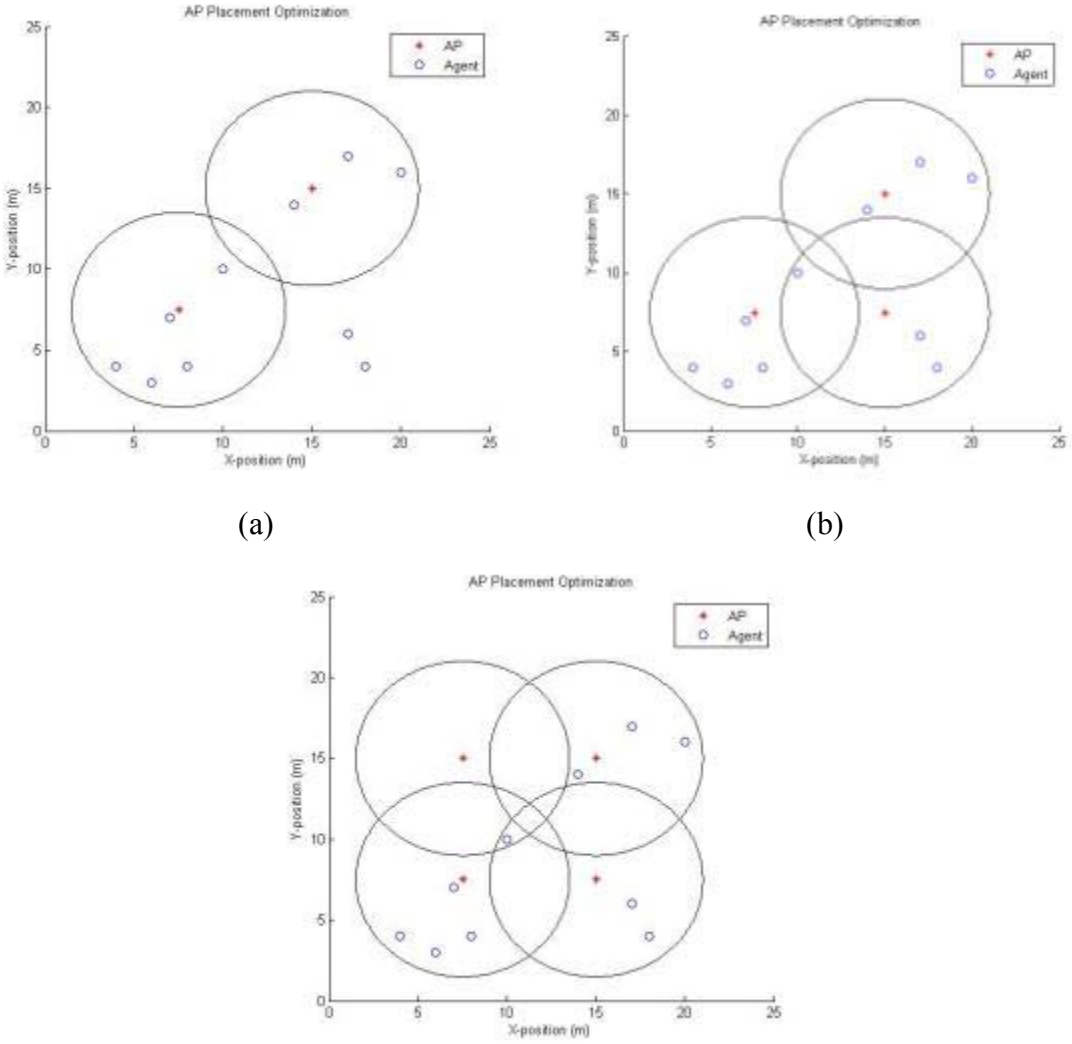


Figure 3-8. AP placement optimization results for 10 agents with a) 2 APs, b) 3 APs, and c) 4 APs.

3.5. Summary

This chapter presented an in depth development of an updated wireless network model for the Zonal Model. Proper integration of WLAN technology into the ship system must entail the optimal placement of the APs. From experiments at the former Burtsfield Elementary School, the relationship between the signal strength of the wireless APs and the data transfer rate that each AP can deliver was determined. A hybrid automaton summarizing the model was created. It determined the mode on which each AP will operate according to the configuration of the agents and the APs.

The performance of the updated wireless network model and a previously developed model was compared to determine how the performance of the model impacted the agents' workflow and the overall performance of the network. The results from the previous model and the updated model were similar. However, the updated model was shown to be superior in enabling multiple agents to access a single AP or multiple APs simultaneously.

The variation in data transfer rate is caused by the distance and obstructions between the user/receiver and the AP. Hence, the optimal positioning of the AP location would play a crucial role in maximizing the transmission coverage and communication throughput. An AP placement strategy was formulated that requires each AP to be separated so that the users would receive a signal strength better than -45 dBm when they are located near an AP. As a user moves towards another AP and he/she began to receive a signal strength worse than -70 dBm from the currently connected AP, a hand-off process would take place and a better signal strength would be obtained. A Matlab simulation was created for implementation of this strategy and results of optimized AP placement for various numbers of APs were presented.

CHAPTER 4. SIMULATION RESULTS AND ANALYSIS

4.1. Simulation Setup

Based on the models presented in the previous chapters, a Zonal Model simulation scenario was developed and is presented in the tables below. There are two schedules available depending on the number of agents in the zones. Table 4-1 and Table 4-2 show the schedules for 7 agents and 10 agents with the green blocks representing the maintenance tasks, the red blocks representing the repair duties, and the yellow blocks representing the watch duties. The schedule is designed using one-shift per day that is executed by all crewmembers. There are 6 maintenance tasks, 12 watch duties, and 15 repair duties distributed among the available agents in the zones.

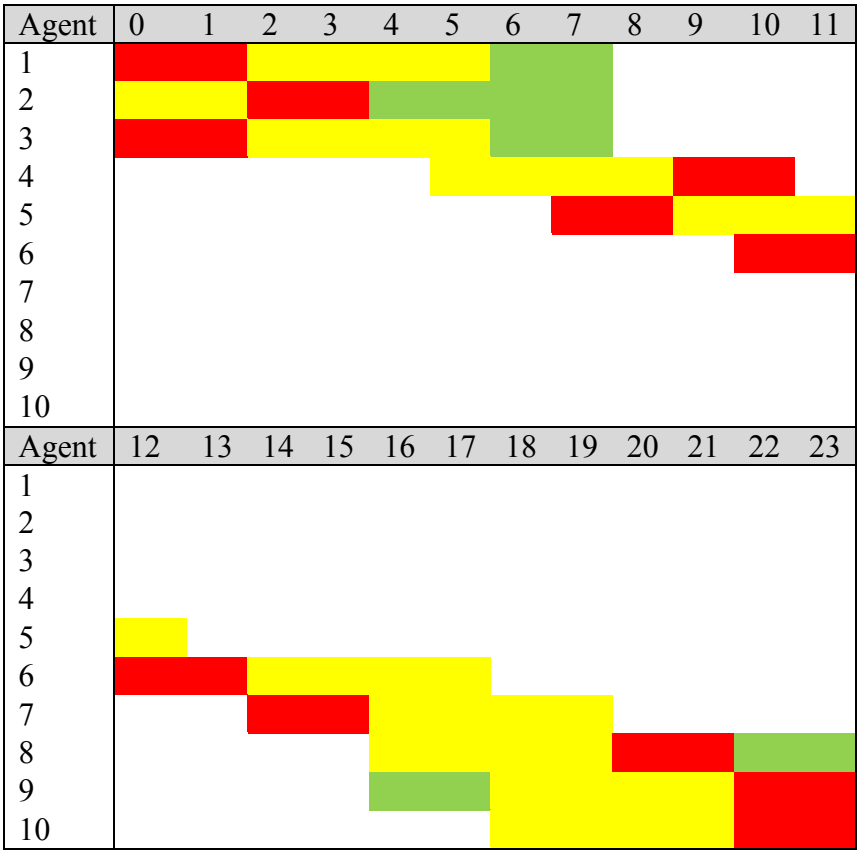
Table 4-1. Schedule for 7 agents.

Agent	0	1	2	3	4	5	6	7	8	9	10	11
1												
2												
3												
4												
5												
6												
7												

Agent	12	13	14	15	16	17	18	19	20	21	22	23
1												
2												
3												
4												
5												
6												
7												

Besides the scheduled maintenance, repair, and watch duties, there are additional random machine and equipment failure occurring throughout the day. The random failures are designed to simulate unanticipated troubleshooting of machines and equipment throughout the day. Agents are required to write and submit a report after completing each duty; however, depending on the priority of task reporting, agents might be required to perform more work before being allowed to write and submit the report.

Table 4-2. Schedule for 10 agents.



4.2. Simulation Results

The results of the simulation for 10 agents operating in the environment with 4 APs and 2 wired workstations are presented in this section. Figure 4-1 shows the agent work completion progress. Agents 4, 7, and 8 are not scheduled to work until the middle of the

day; however, as the results show, they start early due to the rescheduling mechanism to handle machine failures that occur during the early part of the simulation. The reporting process priority is set to be high; therefore, if an agent is not immediately assigned to repair machines, or perform fire fighting activities, their reporting process will be initiated. Agent 9 performed seven tasks during the simulation; however, only three reporting processes were initiated. Based on the results in Figure 4-2, the average agent utilization was maintained at 55%, but some agents have significantly more work than the others. According to schedule I, agents 8, 9, and 10 should start their duties in the last third of the simulation; however, agent 8 starts work earlier to assist in the repair of the machine. Thus, agent 8's utilization is higher than for agents 9 and 10. The visualization of work completion is divided into two main parts: completion of assigned duties and completion of data transfer. Each completion indicator for assigned duties will be shown as 50% on the graph and as soon as the agent finishes the reporting process, 100% will be shown on the graph.

The network performance is visualized in Figure 4-3. Each AP provides a maximum data transfer rate for each agent since there is no condition where the data transfer takes place simultaneously (refer to agents' activities in Table 4-1 and 4-2).

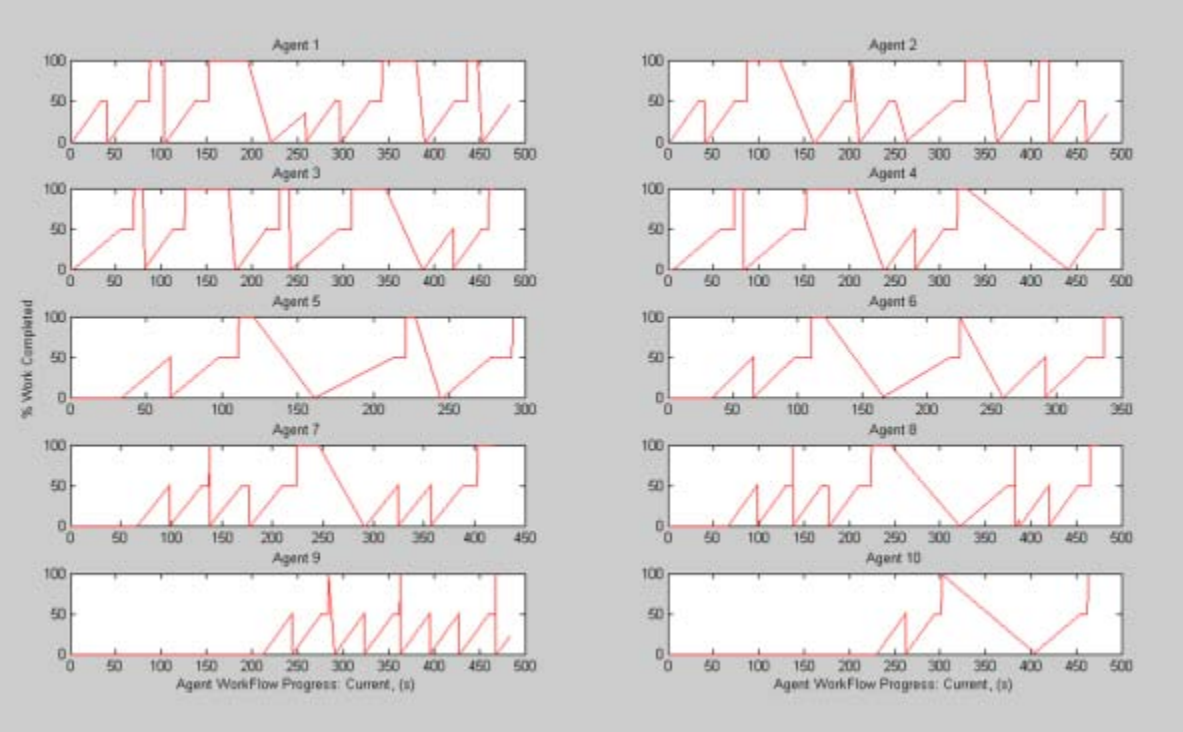


Figure 4-1. Agent work completion progress (10 Agents, 4 APs, and 2 Workstations).

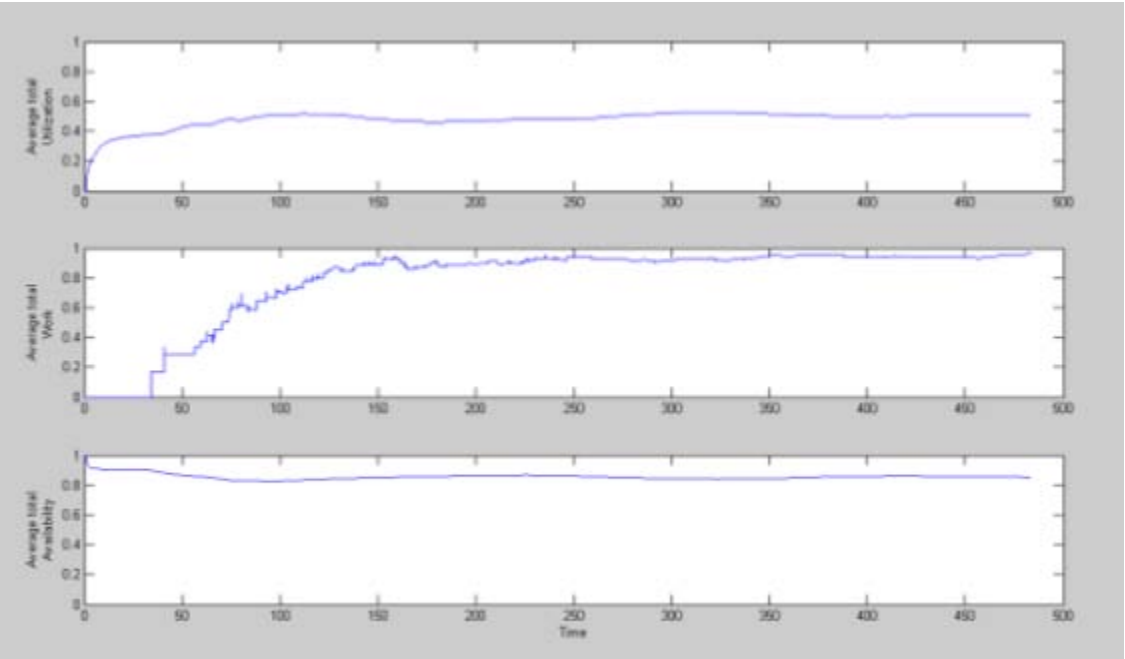
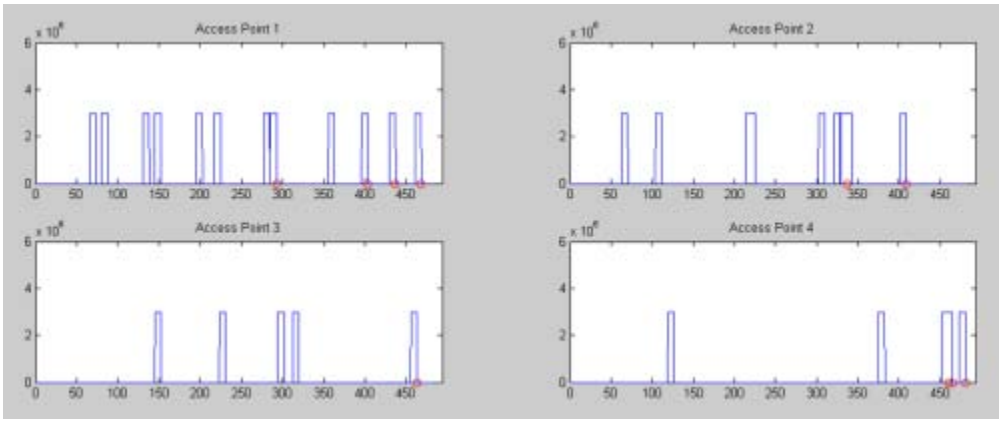
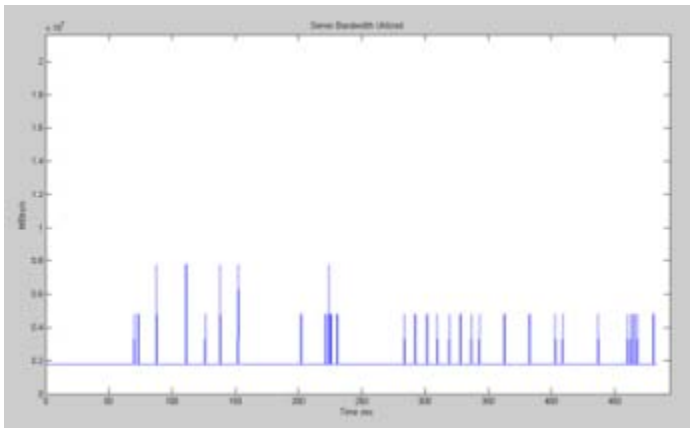


Figure 4-2. Agent total utilization, total percentage of completed work, and availability during the entire simulation (10 Agents, 4 APs, and 2 Workstations).



(a)

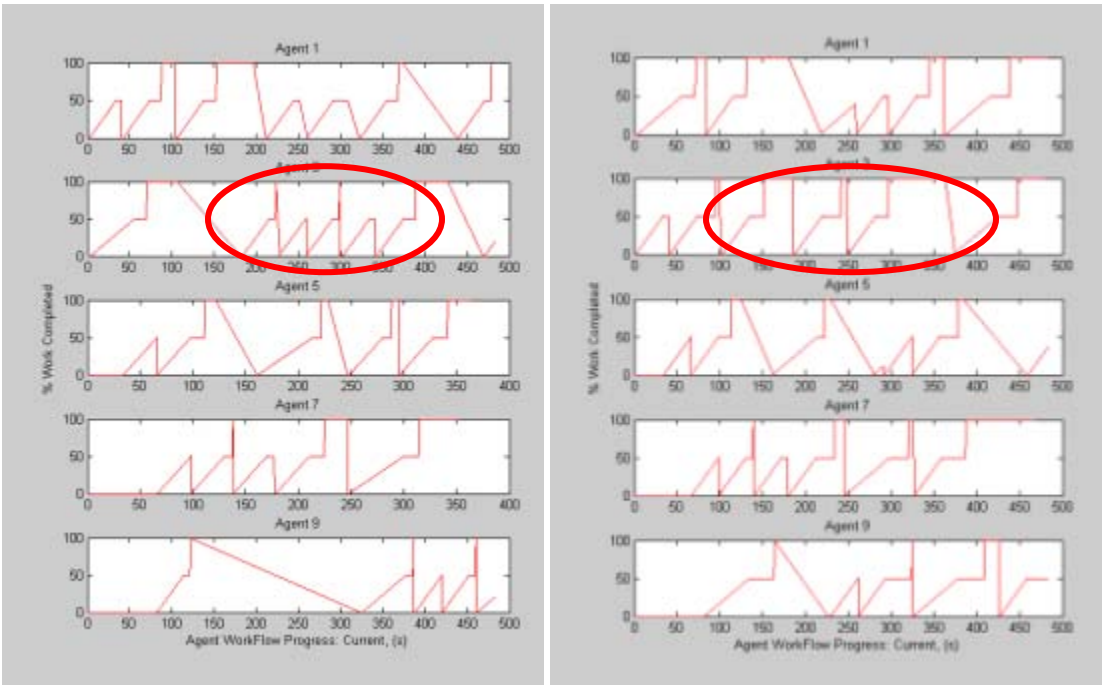


(b)

Figure 4-3. Bandwidth utilization of: a) APs and b) server (10 Agents, 4 APs, and 2 Workstations).

Variation of the network setup to account for a wired network only, wireless network only, and combination of wired and wireless networks was also implemented in the simulation. The wired network represents the current baseline against which the performance of agents in the ship will be measured. A wireless network implementation without any wired network was tested and analyzed to determine the possibility of replacing the existing system. The simulation involving a combination of wired and wireless networks was designed so that in the case of wireless network failure, backup wired network would be available. The availability of wired workstations also enables the agent that is located close to a particular workstation to carry out reporting processes there without using the PDA. Figure 4-4 shows a comparison between the agent work completion for both the wireless network only and wired network only. Among all 10

agents working in the zones of the ship, odd numbered agents are used for analysis of the workflow since most variation in the workflow dynamics are observed in those agents. When analyzing the workflow of agent 3, it is clear that for the implementation of a purely wired network, agent 3 experiences additional waiting time due to the queue at the workstation, shown as a flat line for 100% work completion. When the wireless network was implemented, agent 2 completed the duties in a shorter period of time. The scale being used in the graph represents the actual simulation time where each 24-hour simulation run corresponds to 484 seconds in real time.



(a) (b)

Figure 4-4. Comparison of agent work completion with: a) wireless network only and b) wired network only.

It was also critical to evaluate the impacts of emergency scenarios, e.g., fire and AP failure, on the workflow of the crewmembers. The effects of a fire on the workflow completion and crew utilization varies depending on the location and severity of the event. A fire was initiated at hour 6 in different locations and took about 2-4 hours to extinguish depending on the location of the fire and crew availabilities. Figure 4-5 shows

the three locations considered: in general room (room number 7 in this simulation), near AP 1, and near the server. When the server fails, the network relies on a single workstation that is assumed to be the only working terminal for writing and submitting the emergency report.

A fire incident near the server yielded the lowest average agent utilization at 54%, while the fire near AP1 yielded an average agent utilization of 56%, and the fire at room 7 yielded an average agent utilization of 59%. The total work completion rate diminished as the severity of the fire increased. A fire incident near the server caused the work completion rate to drop to 92% while a fire in room 7 still yielded a work completion rate of 98%.

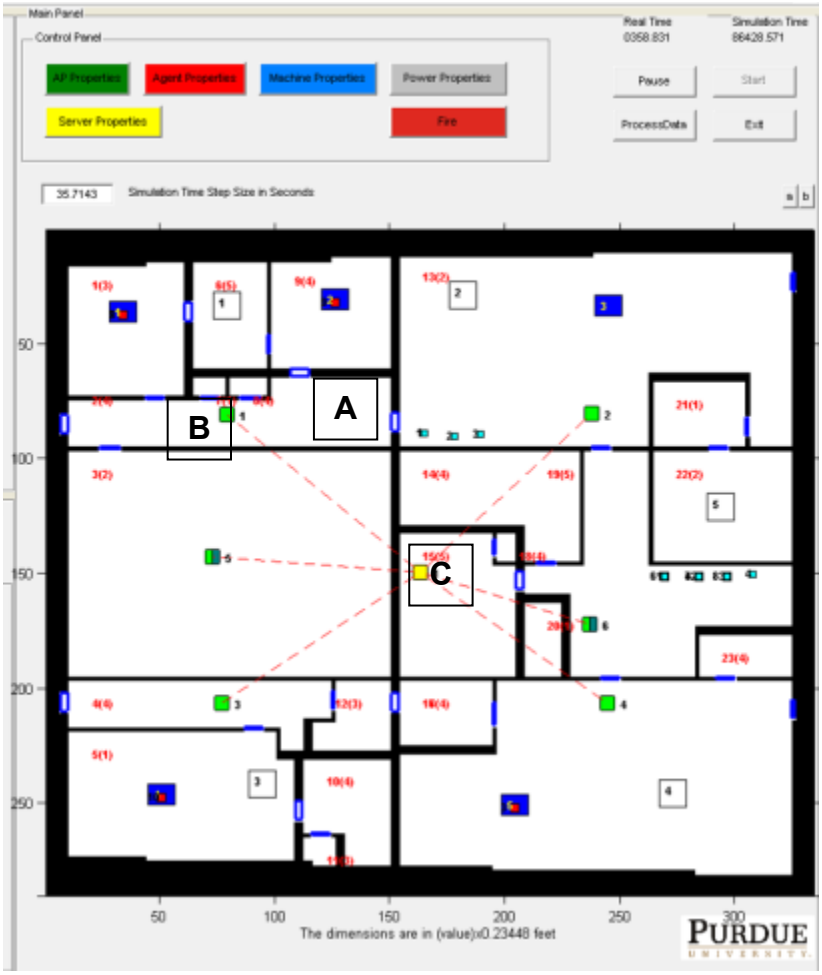
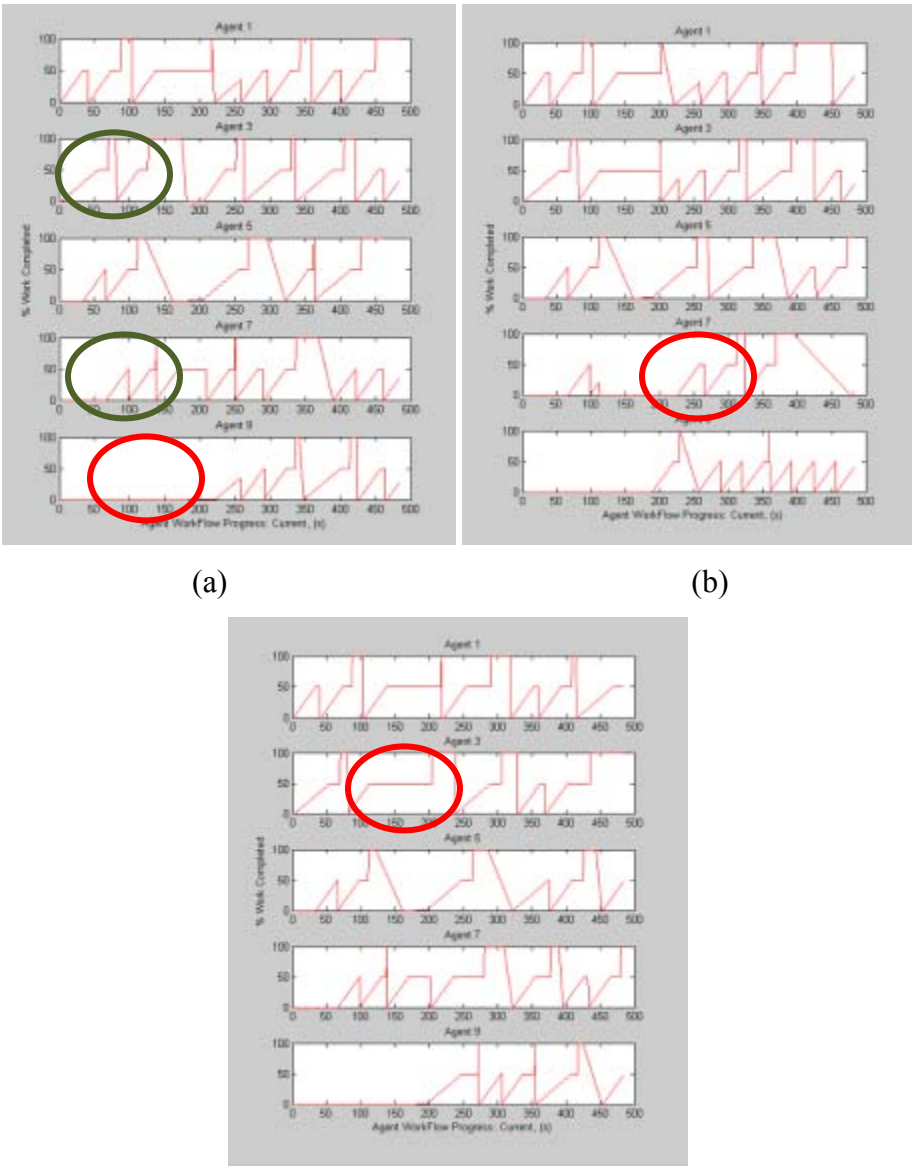


Figure 4-5. Fire location: a) room number 7, b) near AP 1, and c) near the server.

Figure 4-6 shows the direct impact of a fire on the agents' workflow. Agents 9, 7, and 3 were involved in fire fighting for a fire started in room 7, AP1, and near the server, respectfully, at three different and independent simulation runs. Only odd numbered agents are shown to simplify the visualization of workflow changes. Their workflows have been adjusted due to the fire fighting, which has the highest job priority. The agents that are assigned to fight fire will have higher increases in their utilization rate and, thus, some of their duties would be assigned to other agents that do not participate in the fire fighting task. According to agent 7's workflows from Figure 4-6, during the fire fighting process, agent 7 does not manage to complete two tasks (shown as green circles), which are performed when the agent is not on fire fighting duty. Because the fire emergency has taken agent 7 off of the assigned duties, other agents have picked up the duties; therefore, the workflow of agent 7 becomes lighter in the performance of fire fighting duties relative to the scenario that does not involve a fire. Agent 3's workflow is also modified due to the involvement in fire fighting. In Figure 4-6.a, when agent 3 executes the original schedule, agent 3 has a recreation assigned while when agent 3 participates in the fire fighting task, the recreation is taken away as the part of fire fighting assignment, as shown in Figure 4-6.c.

During an AP failure scenario, the AP is shut down and stays non-functional for the remainder of the simulation. For a simulation setup with four APs and two wired workstations, one of the four APs is switched off on four different simulation runs. The average agent utilization drops on average to 50.3% from 59% when there is no AP failure. Due to the existence of wired workstations, the work completion rates for the entire AP failure scenario do not fall below 98%.



(a) (b)

(c)

Figure 4-6. Agent work completion progress during fire at: a} room number 7, b) near AP 1, and c) near the server.

4.3. Design of Experiments Analysis

In order to understand the impact of the AP configuration, number of APs, AP failure, and fire emergency on the time to 90% completion of all tasks and the average agent and AP utilization rates, a design of experiments (DOE) was performed using seven factors,

each of which with two levels. Instead of using a full factorial design, a level IV fraction factorial design developed with thirty-two different scenarios that systematically explored the simulation space was used to reduce the number of simulation runs. The numbers of APs used in the two levels of simulation were two and four APs. The numbers of working agents were seven and ten agents. The file size of the report being transferred was 1 Mb and 2 Mb. The reporting task priority was two or four out of five, with five being the highest priority job. The numbers of fire and AP failure incidents were zero and one. The number of wired workstations was zero and two workstations.

Sensitivity analysis (SA) can be employed to identify the essential factors and critical interactions in a model. It is known from the parsimony principle that only a few factors will affect the output while the rest will be of no consequence (Montgomery, 2005). Based on the average of ten replications of the thirty-two designed scenarios, Pareto charts shown in Figure 4-7, Figure 4-8, and Figure 4-9 were generated. The completion of at least 90% of the work depends heavily on the number of wired workstations and the existence of a fire emergency. The average agent utilization was dependent on the total number of working agents. AP utilization was highly affected by the number of APs and wired workstations. For example, the number of agents has a significant impact on the overall agent utilization, which was anticipated because the more agents that are available, the less work each of these agents must perform.

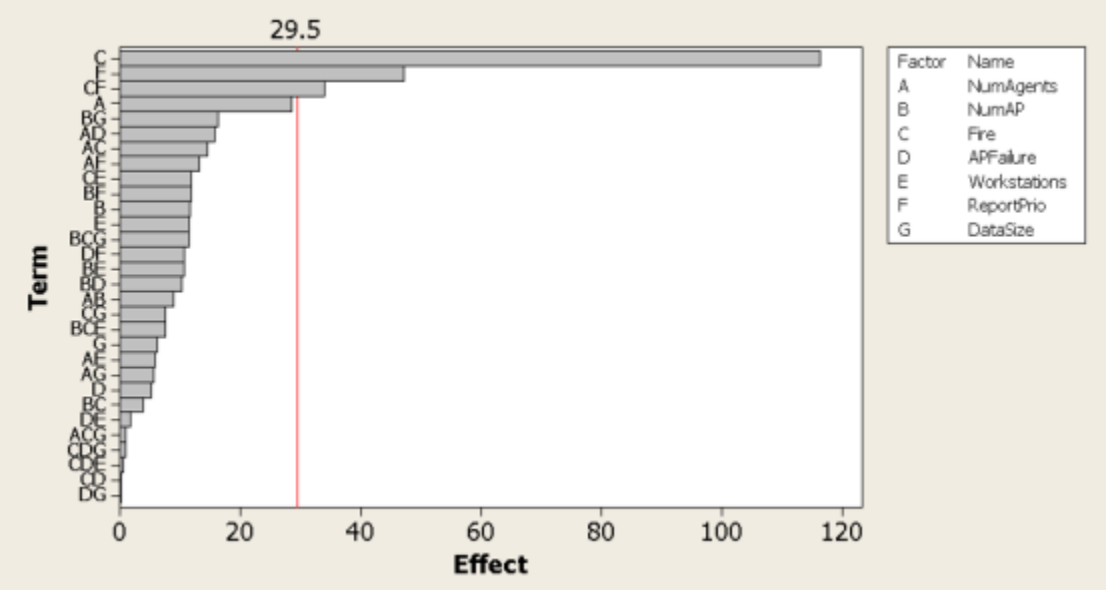


Figure 4-7. Pareto chart of the effects on90% completion of work.

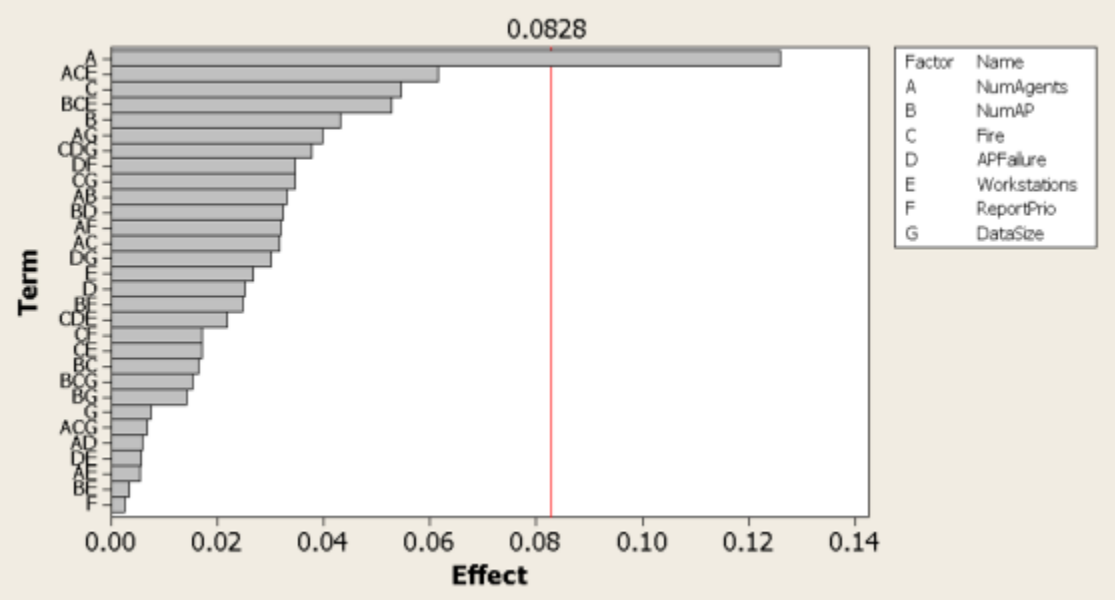


Figure 4-8. Pareto chart of the effects onaverage agent utilization.

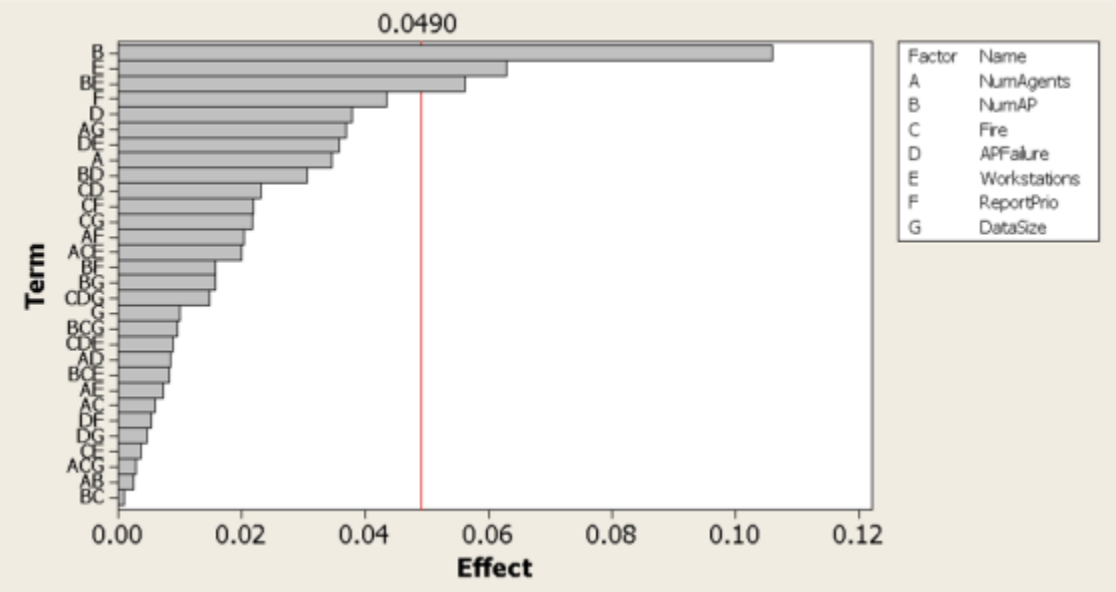


Figure 4-9. Pareto chart of the effects onaverage access point utilization.

Among all factors identified as significant in the 90% completion of work for each simulation run, the fire emergency led to the most delay in reaching 90% work completion. On average, the fire causes almost 100% delays to reach the 90% completion time. As the priority of reporting is increased from 2 to 4, the time to reach 90% completion of work is significantly reduced by 30%. These results are clearly shown in Figure 4-10.

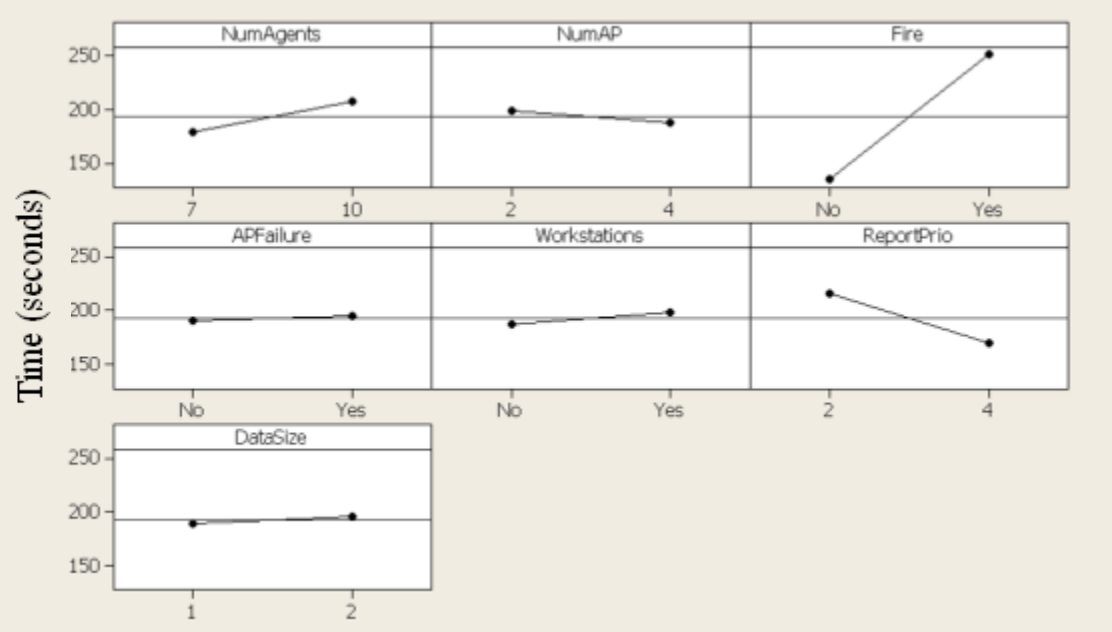


Figure 4-10. Main effects plot for 90% completion of work.

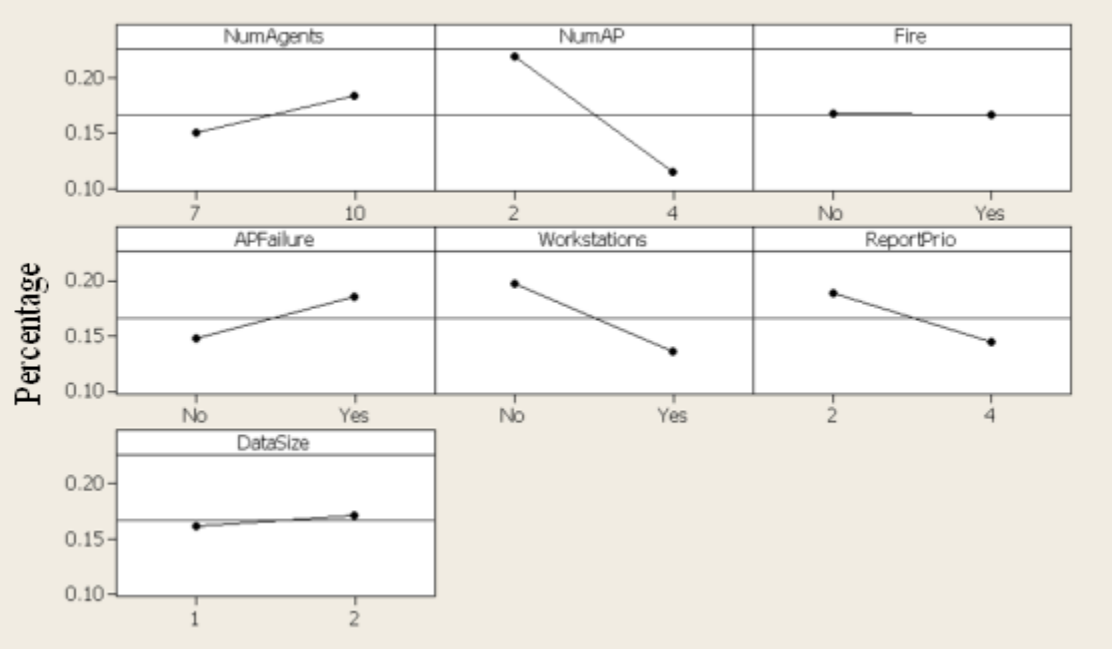


Figure 4-11. Main effects plot for average AP utilization.

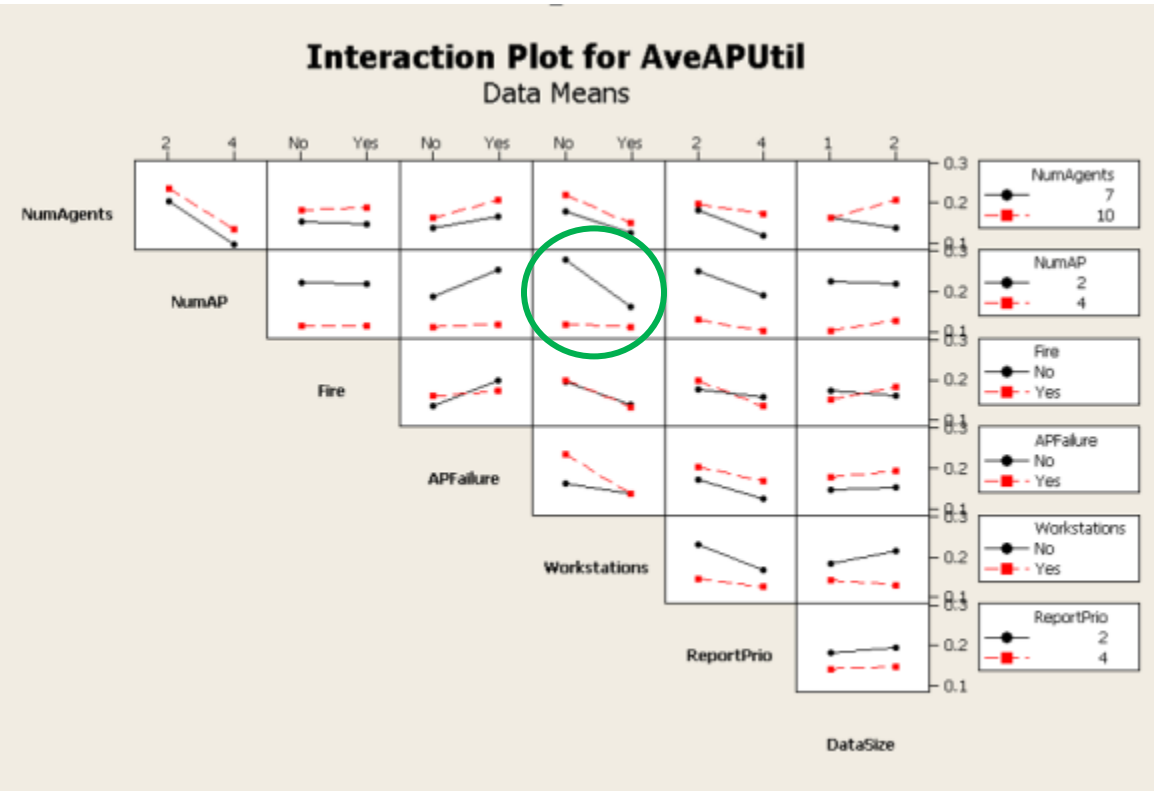


Figure 4-12. Interaction plot for average AP utilization.

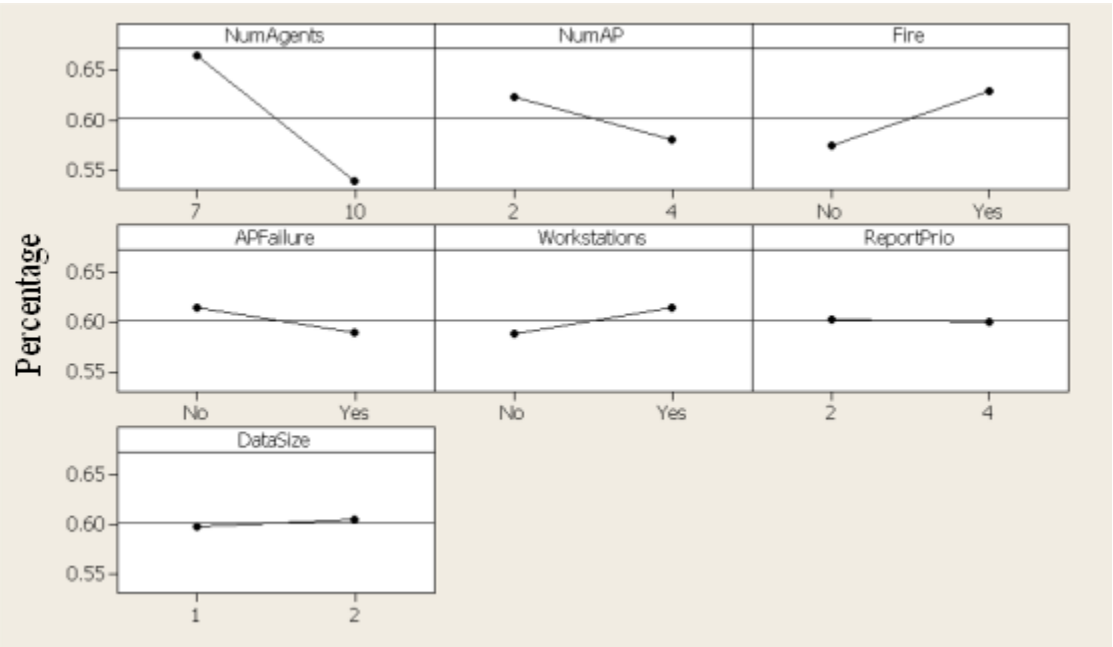


Figure 4-13. Main effects plot for average agent utilization.

Based on the results in Figure 4-11, when the number of APs increased from two to four, it is evident that the average AP utilization was reduced by 50% - in other words, AP utilization is inversely proportional to the number of APs available. When wired workstations were implemented in the simulation, the average AP utilization was reduced. The existence of wired workstations when only two APs are available helped to lower the average AP utilization; however, when four APs were available, the wired workstations did not contribute to lower AP utilization because the agents could do the reporting task without having to visit the wired workstations. These phenomena are shown in the interaction plot for the factors marked by green circles in Figure 4-12. However, referring to the simulation results from the previous section, the 100% completion of assigned tasks was the same for the second level of reporting priority. From Figure 4-13, the average agent utilization drops to around 15% when the number of agents was increased from seven to ten agents. Regardless of whether seven or ten agents are deployed in the zones, the agent utilization remained near 65% or lower as desired in order to prevent agents from getting overworked. Therefore, a reduction of the number of agents from ten to seven would be reasonable

4.4. Summary

This chapter presents the setup for the simulation scenarios that were investigated, results of those simulations, and the analysis of the results. Two schedules for seven and ten agents were presented and results showing network utilization, agent work completion, agent utilization, and agent availability were produced. A comparison between wired, wireless, and a combination of wired and wireless networks was made. The existence of a wireless network in the system allows agents to finish the tasks and duties more quickly; however, the wired network is still required to maintain backup service in the case of a wireless network failure. Furthermore, the combination between wired and wireless network is crucial in ensuring the agents always have available terminals for sending and receiving data at all times.

A level IV fractional factorial design was implemented with seven factors i.e., the number of APs, the number of agents, the size of files transferred, the reporting task priority, the number of wired workstations, and the existence of fire incidents and AP failure, with two levels each. Among all factors identified as significant towards the 90% completion of work for each simulation run, when a fire emergency occurs, there will be a significant delay in reaching the 90% work completion as anticipated. The completion of at least 90% of the work depends heavily on the number of wired workstations and the occurrence of a fire emergency. The average agent utilization is largely dependent on the total number of working agents. The desired network configuration combines the existing wired workstations with wireless access points and sensors to maintain 100% work completion.

CHAPTER 5. CONCLUSIONS AND RECOMMENDATIONS

5.1. Summary of Work Done

The objective of this thesis was to model and simulate the implementation of wireless technology in naval ship system-of-systems. More specifically, this research presented tools that facilitate the consideration of optimized manning with reduced manning level by utilizing the latest wireless communication technology. A Zonal Model previously developed by Mahulkar and McKay served as the platform for the developing the updated wireless network model. The limitations of the existing wireless network model in recognizing the actual interactions between multiple users and APs lead to the need for further development of the wireless network model to be implemented into the Zonal Model.

The first objective was to model and simulate the wireless technology operations in naval ships. Physical experiments consisting of signal strength measurement and data transfers at former Burtsfield Elementary School were performed and the nonlinear relationship between the location of crewmembers, signal strength, and the corresponding data rate was recorded and analyzed. A relationship between the distance between users and APs, received signal strength, and the corresponding data rate was formulated into a hybrid automaton. Despite various other wireless signal strength and data rate measurement techniques available, the model developed could capture the behavior of the signal under various emergency scenarios and simulated ship environments.

The second objective was to optimize the wireless network configuration to satisfy the manning requirement set forth by the United States Navy. Among all factors that determine the optimized manning, the ship's capability establishes the limit on the

minimum number of crewmembers that must be readily available at all times. At least 60% of the current manning level is required for combat readiness and other vital operations on ships. The wireless network could meet the demand and functionality required.

The third objective was to quantify the robustness of configuration due to emergency scenarios. The quantity of crewmembers, APs, and failure scenarios corresponding to specific schedules contributes to the AP utilization and data transfer time. The design of experiments determined that the existence of fire emergencies and variations in the reporting process priority both play important roles in determining the time to reach the 90% completion time for the assigned duties. The network configuration was robust from failure of the APs and the availability of wired workstations helped to maintain the completion of the duties. Further reduction in the number of agents increased the average agent utilization; however, it was still under the desired utilization rate of 65%.

The original crew size of DDG class ship is 323 personnel of officers and enlisted crews. There are 3 decks with 11 zones in the ship, so with the manning level of 7 crews per 2 zones, there are total of 116 crew members required to accomplished the designed mission with fire and AP or server failure scenarios. Therefore, the implementation of a wireless network into an existing DDG ship would ensure reduction of the current manning level in the ship.

5.2. Future Work

Further research in the area of the impacts of topology, agent relationships, and physical infrastructure has tremendous potential for future research in academia and application to the military world. A combat scenario could be introduced to further assess the performance of the current network configuration. A human interaction model integrated with a social network would provide more realistic agent interactions and enable virtual human decision in the simulation. Simulations of the integration of wireless

communication in fleets of ships as a part of a Navy defense system-of-systems would be feasible with an extension of the Zonal Model.

There are many potential future research issues as an extension of this study in the area of decision support in the implementation of new technology insertion into a System of Systems. The development of wireless technology is rapidly growing and the current Zonal Model provides the modular platform for new technology evaluation.

BIBLIOGRAPHY

BIBLIOGRAPHY

Andersen, J. B., T. S. Rappaport, S. Yoshida (1995). "Propagation measurements and models for wireless communications channels." Communications Magazine, IEEE 33(1): 42-49.

Andrade, S. F., N. C. Rowe, D. P. Gaver, P. A. Jacobs. (2002). Analysis of Shipboard Firefighting-Team Efficiency Using Intelligent-Agent Simulation. Monterey, CA, Naval Postgraduate School.

Arora, J. S. (2004). Introduction to Optimum Design. San Diego, CA, Elsevier Academic Press.

Axtell, R. (2000). Why Agents ? on the Varied Motivations for Agent Computing in the Social Sciences. Washington, D. C., Center on Social and Economic Dynamics - The Brookings Institution. Technical Report 17.

Battiti, R., M. Brunato, A. Delai. (2003). Optimal WIreless Access Point Placement for Location Dependent Services. Trento, Italy, Universita di Trento.

Bianchi, G. (2000). "Performance analysis of the IEEE 802.11 distributed coordination function." Selected Areas in Communications, IEEE Journal on 18(3): 535-547.

Brandstein, A., G. Horne, H. Friman. (2000). Project Albert + ROLF 2010 = Red Orm. Proceedings of 5th International Command and Control Research and Technology Symposium, Canberra ACT, Australia.

Brandstein, A. G., G. E. Horne, M. Leonardi. (2000). Project Albert Overview. Colorado Springs, CO.

Carlock, P. G. and R. E. Fenton (2001). "System of Systems (SoS) enterprise systems engineering for information-intensive organizations." Systems Engineering 4(4): 242-261.

Chaturvedi, A., S. Mehta, D. Dolk, R. Ayer. (2005). "Agent-based simulation for computational experimentation: Developing an artificial labor market " European Journal of Operational Research 166(3): 694-716.

Cox, C. W. (1997). Findings Released on Hunter Warrior Advanced Warfighting Experiment. D. o. P. Affairs. Quantico, VA.

Daley, P. J. and J. Gani (1999). Epidemic Modelling: An Introduction. Cambridge, Cambridge University Press.

DeLaurentis, D. and R. K. Callaway (2004). "A system-of-systems perspective for future public policy." Review of Policy Research 21(6): 829-837.

DeLaurentis, D., C. Dickerson, M. DiMario, P. Gartz, M. M. Jamshidi, S. Nahavandi, A. P. Sage, E. B. Sloane, D. R. Walker. (2007). "A case for an international consortium on system-of-systems engineering." Systems Journal, IEEE 1(1): 68-73.

Department-of-Navy (2003). Ship's Maintenance And Material Management (3-M) Manual. Washington, D. C., Naval Sea Systems Command.

Estes, D. R. J. (2001). Assessment of Radio Frequency Propagation in a Naval Shipboard Environment. Annapolis, MD, US Naval Academy.

Ferro, E. and F. Potorti (2005). "Bluetooth and Wi-Fi wireless protocols: a survey and a comparison." Wireless Communications, IEEE 12(1): 12-26.

Gilbertson, C. J. (2007). The Thousand Ship Navy: Creating a Maritime System of Systems. Newport, RI, Joint Military Operations Department, Naval War College.

Hoffman, F. G. and G. E. Horne (1998). Maneuver Warfare Science. Quantico, VA, Marine Corps Combat Development Command.

Jamshidi, M. (2008). "System of systems engineering - new challenges for the 21st century." Aerospace and Electronic Systems Magazine, IEEE 23(5): 4-19.

Joint-Technical-Coordinating-Group-on-Aircraft-Survivability (2001). Aerospace Systems Survivability Handbook Series. Arlington, VA. 1.

Kevan, T. (2006, 1 Feb 2006). "Shipboard Machine Monitoring for Predictive Maintenance."
from http://www.sensorsmag.com/sensors/article/articleDetail.jsp?id=314716.

Kuran, M. S. and T. Tugcu (2006). "A survey on emerging broadband wireless access technologies." Computer Networks: The international Journal of Computer and Telecommunications Networking 51: 3013-3046.

Li, J. (2006). Shipboard Machine Monitoring for Predictive Maintenance, Ohio State University.

Lukasik, S. J. (1998). "Systems, systems of systems, and the education of engineers." Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12(1): 55-60.

Mahulkar, V., L. Lin, N. Shroff, A. Chaturvedi, O. Wasynczuk. (2006). "Modeling and simulation of zonal ship system of systems using agent-based approach." Dynamic Systems,Measurement, and Control, in preparation.

Mahulkar, V., S. McKay, et al. (2008). "System of Systems Modeling and Simulation of aShip Environment with Wireless and Intelligent Maintenance Technologies." IEEE Transactions on Systems Man & Cybernetics, Part A.

Mahulkar, V.,D. E. Adams, L. Lin, N. Shroff, A. Chaturvedi. (2006). Modeling and Simulation of Ship System of Systems with Wireless Network Inserted for On-Ship Communications. American Society of Mechanical Engineers International Mechanical Engineering Congress and Exposition. Chicago, IL.

Maier, M. W. (1998). "Architecting Principles for System of Systems." Systems Engineering 1(4): 267-284.

Manthorpe Jr, W. H. J. (1996). "The emerging joint system of systems: Asystem engineering challenge and opportunity for APL." Johns Hopkins APL Technical Digest (Applied Physiscs Laboratoratory) 17.

Montgomery, D. C. (2005). Design and Analysis of Experiments. NJ, John Wiley & Sons.

Nagy, L. and L. Farkas (2000). Indoor base station location optimization using genetic algorithms. The 11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, 2000.

Ning, X., R. Sumit, C. K. Kant, G. Deepak, B. Alan, G. Ramesh, E. Deborah. (2004). A Wireless Sensor Network for Structural Monitoring. Proceedings of the 2nd international conference on Embedded networked sensor systems. Baltimore, MD, USA, ACM.

Niraj, S., F. Pietryka, G. Horne, M. Theroff. (2005). Simulation Environment to Assess Technology Insertion Impact and Optimized Manning. Simulation Conference, 2005 Proceedings of the Winter.

Office-of-the-Assistant-Secretary-of-Defense (2008). Navy Secretary Names New Guided-Missile Destroyer USS Michael Murphy. P. Affairs.

Panjwani, M. A., A. L. Abbott, T. S. Rappaport. (1996). "Interactive computation of coverage regions for wireless communication in multifloored indoor environments." Selected Areas in Communications, IEEE Journal on 14(3): 420-430.

Pei, R. S. (2000). System of Systems Integration (SOSI) - A Smart Way of Acquiring Army C4I2WS System. Summer Computer Simulation Conference.

Pike, J. (2000). "Hunter Warrior." from http://www.globalsecurity.org/military/ops/hunter-warrior.htm.

Rappaport, T. S., S. Y. Seidel, K. Takamizawa. (1991). "Statistical Channel Impulse Response Models for Factory and Open Plan Building Radio Communicate System Design." Communications, IEEE Transactions on 39(5): 794-807.

Sage, A. P. and C. D. Cuppan (2001). "On the Systems Engineering and Management of Systems of Systems and Federations of Systems." Information, Knowledge, Systems Management 2(4): 325-345.

Spindel, R. C., S. Laska, J. A. Cannon-Bowers, D. L. Cooper, K. C. Hegmann, R. J. Hogan, J. E. Hubbard, J. A. Johnson, D. J. Katz, R. Kohn, Jr., K. A. Roberts, T. B. Sheridan, A. M. Skalka, J. A. Smith. (2000). Optimizing Surface Ship Manning. O. o. t. A. S. o. t. Navy.

Stavridis, J. and R. Girrier (2004). Division Officer's Guide. Annapolis, MD, Naval Institute Press.

Sukun, K., P. Shamim, C. David, D. James, F. Gregory, G. Steven, T. Martin. (2007). Health monitoring of civil infrastructures using wireless sensor networks. Proceedings of the 6th International Conference on Information Processing in Sensor Networks. Cambridge, Massachusetts, USA, ACM.

Tia, G., D. Greenspan, M. Welsh, R. Juang, A. Alm. (2005). Vital Signs Monitoring and Patient Tracking Over a Wireless Network. Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the.

Walters, E. A. (2001). Automated averaging techniques for power electronic-based systems. Electrical and Computer Engineering. West Lafayette, IN, Purdue University. PhD.

Wasynczuk, O., E. A. Walters, H. J. Hegner. (1997). Simulation of a Zonal Electric Distribution System for Shipboard Applications. Proceedings of the 32nd Intersociety Energy Conversion Engineering Conference, 1997.

Wright, M. H. (1998). Optimization methods for base station placement in wireless applications. 48th IEEE Vehicular Technology Conference, 1998.

APPENDICES

APPENDIX A. MATLAB CODES FOR ZONAL MODEL SIMULATION

```
function main()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% function initializes all the GUIs
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear all;

warning offall;
parameterSetUp1();

%==================================Gui
Init==============================
g = mainGUI();
set(g, 'Renderer', 'opengl');
guidata(g);

function parameterSetUp1()
%
% PARAMETERSETUP This function initializes all the parameters and GUIs
%
% USAGE:
%   parameterSetUp()
%
% Created by: Vishal Mahulkar
% Created on: 13 October 2006
% Modified by: Robin Kusmanto
%
% Last Modified: 25 May 2009
%
%% PARAMETERS

delete(timerfindall);

% path should be added only when it is not deployed otherwise problems
with
% exe
if( ~isdeployed )
    addpath('./xmltree')
    addpath('./XMLparser')
    addpath('./GenerateGeometry')
    addpath('./matlab_bgl')
    addpath('./xlsData')
    addpath('./XMLSchema')
end

% ship size in actual in meters and pixel size of the image
shipSizeMtr = [154 19.8]*3.2808;
```

```
shipSizePix = [2155 277];

% calculate the conversion factor
global CONVERSIONFACT;
CONVERSIONFACT =
mean([shipSizePix(1)/shipSizeMtr(1),shipSizePix(2)/shipSizeMtr(2)]);

global STARTSIMULATION; STARTSIMULATION = false;

%% Load cases information
global parameterValues;
load PARAMVALUES;
global pValueIdx; pValueIdx = 1

%==================================Time
Keeping============================
% real time
global time; time = clock;

% simulation time keeping
global simulationTime; simulationTime = 0;
global deltaTime; deltaTime = .2;

%==================================Constants=========================
====
global R; R = .4;                                    % Radius of min
approach
global N; N = 10;       % number of agents
global NT; NT = 4;                                   % number of APs
global NW; NW = 2;                                   % number of
workstations
global NM; NM = 5;                                   % number of
machines
global NWL; NWL = 5;                                 % number of Watch
Locations
global NE; NE = 4;                                   % number of
equipment
global NS; NS = 1;                                   % number of servers
global NFE; NFE = 3;                                 % number of fire
equipment
global G; G = 1200;                                  % gravitational
constant for Artificial Physics
global MAXFORCE; MAXFORCE = 50;                      % maximum
permisible force
global FRICTION; FRICTION = 5;                       % frictional force
constant for damping
global MAXSPEED; MAXSPEED = 2*CONVERSIONFACT;        % maximum allowable
speed of movement

global WIRELESSRANGE;                                % maximum wireless
range (80 m)
global WIRELESSRANGE2;                               % wireless range
(60 m)
```

```
global WIRELESSRANGE3;                                  % wireless range
(15 m)
WIRELESSRANGE  = 80*CONVERSIONFACT;
WIRELESSRANGE2 = 60*CONVERSIONFACT;
WIRELESSRANGE3 = 15*CONVERSIONFACT;
global BANDWIDTH; BANDWIDTH = (NT+NW) * 3e6;            % Overall Network
Speed (bytes/s) - proportional to number of AP

global RECREATIONTIME; RECREATIONTIME = 5;               % Time spent at
recreation locations
global WORKSTACK;WORKSTACK = [];                        % global workload
will be distributed
global COST; COST = 0;                                  % number of
iteration fo the mail while loop
global BOUNDARY; BOUNDARY = .7;                         % maximum distance
of approach for activation (equipment)
global WIDTH;WIDTH = 6;                                 % distance between
agents waiting in line
global PACKETS;PACKETS = [];                            % global list of
network transmissions
global PACKETSSENT;PACKETSSENT = [];                    % global list of
network transmissions completed
global staticLoad; staticLoad = 0.5;                    % static load on
the servers

global emergencyScenarios;emergencyScenarios = [];      % global list of
emergency scenarios

global recedenceRateS; recedenceRateS = 0.2;            % rate at which the
smoke recedes when an agent is fighting it (m/s)
global recedenceRateF; recedenceRateF = .08;            % rate at which the
fire recedes when an agent is fighting it (m/s)
global REASSIGN; REASSIGN = true;                       % reassignmnet of
workflow

%==================================Global Exit
Condition=================
global OPTION; OPTION = 1;
global EXITPRESSED; EXITPRESSED = false;

%==================================Power Status
Change====================
%% recalculate the power levels if this flag is set
global POWERSTATUSCHANGE; POWERSTATUSCHANGE = true;

global time_reduction;

%% Time reduction
if parameterValues(pValueIdx,3) == 1
    time_reduction = .0014;  % real value is: 0.125;
else
    time_reduction =  .0014*4;  %real value is: .5;
end
```

```
%% AGENTS

%==================================Agents=============================
====
global ag;
global agCurrentTaskCData;
global agCurrentTaskCTime;
global agTotalTaskCData;
global agTotalTaskCTime;

agCurrentTaskCData(1) = 0;
agCurrentTaskCTime(1) = 0;
agTotalTaskCData(1) = 0;
agTotalTaskCTime(1) = 0;

ag = agents(N);

%% APS AND WORKSTATIONS

%==================================APs and
Workstations==================
global terminal;
terminal = terminals(NT,NW);

global bwTermTime;
global markerTerm;
markerTerm = 0;

%% MACHINES

%==================================Machines===========================
====
global machine;
machine = machines(NM);

global INTELLIGENTMAINTENANCE;
INTELLIGENTMAINTENANCE = 1;

%% EQUIPMENT

%==================================Equipment==========================
====
global equip;
equip = equipments(NE);

%% WATCH LOCATIONS

%==================================Watch
Locations=======================
global watchLoc;
watchLoc = watchLocations(NWL);
```

```
%% SERVERS

%==================================Servers============================
====
global server;
global bandwServ;
global markerServ;

server = servers(NS);
markerServ = zeros(1,3);

%% FIRE EQUIPMENT

%==============================Fire
Equipment============================
global fireEquip;
fireEquip = fireEquipments(NFE);

%% NETWORK SETUP

%==================================Network
Setup=========================
global BANDWIDTHWIRELESS;                          % Overall Network
Speed (bytes/s) Not Used
BANDWIDTHWIRELESS = 10e6;

global BANDWIDTHWIRED;                             % Overall Network
Speed (bytes/s) Not used
BANDWIDTHWIRED = 10e6;

global servCon;                                    % Current
connections to the Server (upload or download)
global servPrio;
servCon  = zeros(3,1);
servPrio = zeros(3,1);

global termCon;                                    % Current
connections to the Terminal
global termPrio;
termCon  = zeros(3,NT+NW);
termPrio = zeros(3,NT+NW);

function ag = agents(N)
%
% AGENTS function to initialize agent structure
%
% USAGE:
%   ag = agents(N)
%
%       ag        the output structure
%       N         the number of agents
%
```

```
% -----------
% Created by: Vishal Mahulkar
% Created on: 17 July 2006
% Version History:
%
% Last Modified: 18 January 2007
%

global CONVERSIONFACT;
global parameterValues;
global pValueIdx;

for i = 1:N
    ag(i).x = 290;                          % current x position
    ag(i).y = 75;                           % current y position
    ag(i).mass = 1;                         % inertia
    ag(i).forcex = 0;                       % force x direction
    ag(i).forcey = 0;                       % force y direction
    ag(i).velx = 0;                         % velocity x direction
    ag(i).vely = 0;                         % velocity y direction
    ag(i).roomNumber = 0;                   % current room number

    ag(i).finalx(1) = 0;                     % intermediate destination
xlocation
    ag(i).finaly(1) = 0;                     % intermediate destination
ylocation
    ag(i).terminalx = 0;                     % final detination
    ag(i).terminaly = 0;                     % final detination
    ag(i).roommap = [];
    ag(i).doormap = [];

    ag(i).terminalreached = false;           % terminal reached
    ag(i).timetermreached = 0;               % time at which the
terminal was reached
    ag(i).destinationreached = false;        % position near terminal if
waiting
    ag(i).waittime = 0;                      % waiting time
    ag(i).waitno = 0;                        % wait number
    ag(i).datainputtime = 0;                 % time required to input
data calculated run time
    ag(i).preterminal = 0;                   % previous terminal
    ag(i).finalterminal = 0;                 % final terminal

    ag(i).busy = false;                      % busy status
    ag(i).timer = 0;                         % for producing delays
    ag(i).status = 2;                        % 1 : moving to machine 2:
moving to terminal 3: watch duty 4:
    ag(i).orderstatus = true;                % order status
complete/incomplete
    ag(i).hasorders = false;                 % currently has orders
    ag(i).timeOrderCompleted = 0;            % time at which order has
been completed
    ag(i).lastTerminalAccessed = 0;          % last terminal used for
network access
```

```
    ag(i).machine = 0;                      % number of machine
currently being inspected
    ag(i).machinereached = 0;               % machine reached status
    ag(i).inspectiondone = false;           % inspection done
    ag(i).timemcreached = 0;

    ag(i).watchStarted = false;             % On watch duty
    ag(i).watch_step = 1;                   % Counter for watch path
    ag(i).watchLocation = [];               % current watch locations

    ag(i).equip = [];                       % equipment should replace
terminal also used for recreation
    ag(i).equipreached = 0;                 % equipment reached
    ag(i).timeequipreached = 0;             % time at which equipment
reached
    ag(i).timetogetequip = 5;

    ag(i).emergency = [];
    ag(i).emergencyLocation = [];           % emergency location number
    ag(i).emergencyLocreached = 0;          % location reached?
    ag(i).timeemergencyLocreached = 0;      % time at which the
location is reached

    ag(i).pda = true;                       % agents has pdas
    ag(i).pdaAddress = [];                  % unique address of the pda
    ag(i).carryEquip = [];                  % equipment carried by
agent

    ag(i).maxSpeed = 2*CONVERSIONFACT;      % max speed
    ag(i).stress = 0;                       % Stress level for agent
(from fire, etc.)
    ag(i).training = mod(i-1,3);            % training level is  0, 1
or 2

    ag(i).alive = true;                     % Alive / Dead - Result of
fire

    ag(i).inclagtimedata = 0;               % Increased time for data
input due to training/stress
    ag(i).inclagtimeinsp = 0;               % Increased time for
inspection input due to training/stress
    ag(i).datarettime = 5;                  %
    ag(i).getinfo = false;                  % more information required
    ag(i).inforeceived = false;             % more information received

    ag(i).coordinate = false;               % coordinate with other
agents boolean
    ag(i).coordinatewith = 0;               % agents to coordinate with

% can be/should be changed during runtime
```

```
if parameterValues(pValueIdx,40) == 1
if i == 1 || 4 || 8
            pr = 1;
else
            pr = 2;
end
else
        pr = 2;
end
    ag(i).packet = newPacket(parameterValues(pValueIdx,17),0,-
1,5,i,pr,[]);

    ag(i).message = [];                     % message received
    ag(i).packetOld = ag(i).packet;         % internal use

    ag(i).deltaTime = .15;                  % future individual deltat
for agents
    ag(i).simulationTime = 0;               % individual simulation
time

    ag(i).percentTaskComp = [];             % record for task
completion rate
    ag(i).ptcTime = [];                     % rate
    ag(i).utilization = [];                 % utilization
    ag(i).utiTime = [];                     % corresponding time

    ag(i).interrupt = true;                 % not used
    ag(i).path = [];                        % path of agent
    ag(i).pathstep = 1;                     % counter for path

    ag(i).workData = [];                    % currrent work data
    ag(i).memory.data = [];                 % memory - information
    ag(i).memory.timeStamp = [];            % memory - timestamp
    timeReduction = 1;
    ag(i).memory.decayRate = {1,10/timeReduction};
    ag(i).workstack = [];                   % workstack

    ag(i).handle = 0;                       % graphics
    ag(i).numberHandle = 0;                 % graphics
end

function varargout = agcntr(varargin)
% AGCNTR M-file for agcntr.fig
%      AGCNTR, by itself, creates a new AGCNTR or raises the existing
%      singleton*.
%
%      H = AGCNTR returns the handle to a new AGCNTR or the handle to
%      the existing singleton*.
%
%      AGCNTR('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in AGCNTR.M with the given input
arguments.
%
```

```
%      AGCNTR('Property','Value',...) creates a new AGCNTR or raises
the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before agcntr_OpeningFunction gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to agcntr_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help agcntr

% Last Modified by GUIDE v2.5 13-Jul-2006 13:59:46

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @agcntr_OpeningFcn, ...
'gui_OutputFcn',  @agcntr_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before agcntr is made visible.
function agcntr_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to agcntr (see VARARGIN)

% Choose default command line output for agcntr

handles.output = hObject;

% Update handles structure
```

```
guidata(hObject, handles);
% UIWAIT makes agcntr wait for user response (see UIRESUME)
% uiwait(handles.agcntr);


% --- Outputs from this function are returned to the command line.
function varargout = agcntr_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in a1pda.
function a1pda_Callback(hObject, eventdata, handles)
% hObject    handle to a1pda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of a1pda
global ag;
ag(1).pda = get(hObject,'Value');
displayAgentStatus(1);


% --- Executes on button press in a2pda.
function a2pda_Callback(hObject, eventdata, handles)
% hObject    handle to a2pda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of a2pda
global ag;
ag(2).pda = get(hObject,'Value');
displayAgentStatus(2);


% --- Executes on button press in a3pda.
function a3pda_Callback(hObject, eventdata, handles)
% hObject    handle to a3pda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of a3pda
global ag;
ag(3).pda = get(hObject,'Value');
displayAgentStatus(3);


% --- Executes on button press in a4pda.
```

```
function a4pda_Callback(hObject, eventdata, handles)
% hObject    handle to a4pda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of a4pda
global ag;
ag(4).pda = get(hObject,'Value');
displayAgentStatus(4);


% --- Executes on button press in a5pda.
function a5pda_Callback(hObject, eventdata, handles)
% hObject    handle to a5pda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of a5pda
global ag;
ag(5).pda = get(hObject,'Value');
displayAgentStatus(5);


% --- Executes on button press in a6pda.
function a6pda_Callback(hObject, eventdata, handles)
% hObject    handle to a6pda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of a6pda
global ag;
ag(6).pda = get(hObject,'Value');
displayAgentStatus(6);


% --- Executes on button press in a7diobutton7.
function a7pda_Callback(hObject, eventdata, handles)
% hObject    handle to a7diobutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of a7diobutton7
global ag;
ag(7).pda = get(hObject,'Value');
displayAgentStatus(7);


% --- Executes on button press in a8pda.
function a8pda_Callback(hObject, eventdata, handles)
% hObject    handle to a8pda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of a8pda
global ag;
ag(8).pda = get(hObject,'Value');
displayAgentStatus(8);


% --- Executes on button press in a9pda.
function a9pda_Callback(hObject, eventdata, handles)
% hObject    handle to a9pda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of a9pda
global ag;
ag(9).pda = get(hObject,'Value');
displayAgentStatus(9);


% --- Executes on button press in a10pda.
function a10pda_Callback(hObject, eventdata, handles)
% hObject    handle to a10pda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of a10pda
global ag;
ag(10).pda = get(hObject,'Value');
displayAgentStatus(10);


% --- Executes on button press in editWorkflow.
function editWorkflow_Callback(hObject, eventdata, handles)
% hObject    handle to editWorkflow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
workflowGUI();


% --- display agent Status
function displayAgentStatus(i)
% i          Agent Number
%
%
global ag;
if( ag(i).pda )
    displayString(['Agent ',num2str(i),' has PDA == TRUE']);
else
    displayString(['Agent ',num2str(i),' has PDA == FALSE']);
end
```

```
% --- Executes when user attempts to close agCntr.
function agCntr_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to agCntr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
set(hObject,'Visible','off')

function varargout = apcntr(varargin)
% APCNTR M-file for apcntr.fig
%      APCNTR, by itself, creates a new APCNTR or raises the existing
%      singleton*.
%
%      H = APCNTR returns the handle to a new APCNTR or the handle to
%      the existing singleton*.
%
%      APCNTR('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in APCNTR.M with the given input
arguments.
%
%      APCNTR('Property','Value',...) creates a new APCNTR or raises
the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before apcntr_OpeningFunction gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to apcntr_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help apcntr

% Last Modified by GUIDE v2.5 18-Sep-2006 13:10:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @apcntr_OpeningFcn, ...
'gui_OutputFcn',  @apcntr_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before apcntr is made visible.
function apcntr_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to apcntr (see VARARGIN)

% Choose default command line output for apcntr

handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

global WIRELESSRANGE;
global CONVERSIONFACT;
set(handles.rangevalue,'String',num2str(WIRELESSRANGE/CONVERSIONFACT));
% UIWAIT makes apcntr wait for user response (see UIRESUME)
% uiwait(handles.apcntr);


% --- Outputs from this function are returned to the command line.
function varargout = apcntr_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in MyButton1.
function MyButton1_Callback(hObject, eventdata, handles)
% hObject    handle to MyButton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of MyButton1
OnOffTerminal(1,hObject,handles);


% --- Executes on button press in MyButton2.
```

```
function MyButton2_Callback(hObject, eventdata, handles)
% hObject    handle to MyButton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of MyButton2
OnOffTerminal(2,hObject,handles);


% --- Executes on button press in MyButton3.
function MyButton3_Callback(hObject, eventdata, handles)
% hObject    handle to MyButton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of MyButton3
OnOffTerminal(3,hObject,handles);


% --- Executes on button press in MyButton4.
function MyButton4_Callback(hObject, eventdata, handles)
% hObject    handle to MyButton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of MyButton4
OnOffTerminal(4,hObject,handles);


% --- Executes on button press in MyButton5.
function MyButton5_Callback(hObject, eventdata, handles)
% hObject    handle to MyButton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of MyButton5
OnOffTerminal(5,hObject,handles);


% --- Executes on button press in MyButton6.
function MyButton6_Callback(hObject, eventdata, handles)
% hObject    handle to MyButton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of MyButton6
OnOffTerminal(6,hObject,handles);


% --- Executes on button press in MyButton7.
function MyButton7_Callback(hObject, eventdata, handles)
% hObject    handle to MyButton7 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of MyButton7
OnOffTerminal(7,hObject,handles);


% --- Executes on button press in MyButton8.
function MyButton8_Callback(hObject, eventdata, handles)
% hObject    handle to MyButton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of MyButton8
OnOffTerminal(8,hObject,handles);

%------
function OnOffTerminal(tnum,hObject,handles)

global ag;
global terminal;
global server;
global machine;

machineCntrHandle = findall(0,'tag','machineCntr');
machineCntrHandleList = guidata(machineCntrHandle);

stat = get(hObject,'Value');
str = ['MyStatic',num2str(tnum)];

if stat == 1
    terminal(tnum).operational = false;
    displayTerminalStatus(tnum);

if(terminal(tnum).agent ~= 0 )
        ag(terminal(tnum).agent).destinationreached = false;
end
    set(handles.(str),'String','Off');
    set(handles.(str),'BackgroundColor','r');
    set(terminal(tnum).handle,'FaceColor',[.7 .7 .7]);

    servNum = terminal(tnum).server;
    packetNums = findPackets(tnum);
for i=1:length(packetNums)
if server(servNum).packets(packetNums(i)).percent < 100
            server(servNum).packets(packetNums(i)).percent = 100;
            server(servNum).packets(packetNums(i)).delay = 0;

            mcn = -server(servNum).packets(packetNums(i)).agent;
            machine(mcn).sensorDataInterrupted = true;
            displayString(['Sensor Data transfer Interrupted from
machine ',...
```

```
                num2str(mcn)]);
            displayString('Trying to route through a new AP');

            str = ['Machine',num2str(mcn)];

machinecntr(strcat(str,'_Callback'),machineCntrHandleList.(str),1,machi
neCntrHandleList);
end
end
else
    terminal(tnum).operational = true;
    displayTerminalStatus(tnum);

    set(handles.(str),'String','On');
    set(handles.(str),'BackgroundColor',[0.9255    0.9137    0.8471]);
    set(terminal(tnum).handle,'FaceColor',[0 1 0]);

for i=1:length(machine)
        str = ['Machine',num2str(i)];
if get(machineCntrHandleList.(str),'Value') ...
&& machine(i).sensorDataInterrupted

            displayString(['Data from Machine ',num2str(i),' was
interrupted',...
'Trying to route sensor data through a new AP']);


machinecntr(strcat(str,'_Callback'),machineCntrHandleList.(str),1,machi
neCntrHandleList);
end
end
end


% --- displays terminal status
function displayTerminalStatus(i)
% i          terminal number
%
%
global terminal;
if( ~terminal(i).operational )
    displayString(['AccessPoint ',num2str(i),' has been switched OFF']);
else
    displayString(['AccessPoint ',num2str(i),' has been switched ON']);
end


% --- Executes during object creation, after setting all properties.
function slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject,'Backgroundcolor',[.9 .9 .9]);
else

set(hObject,'Backgroundcolor',get(0,'defaultUicontrolBackgroundcolor'));
end
global WIRELESSRANGE;
global CONVERSIONFACT;
set(hObject,'Value',WIRELESSRANGE/CONVERSIONFACT);


% --- Executes on slider movement.
function slider_Callback(hObject, eventdata, handles)
% hObject    handle to slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
global WIRELESSRANGE;
global CONVERSIONFACT;
global terminal;
global ag;

WIRELESSRANGE = get(hObject,'Value')*CONVERSIONFACT;
set(handles.rangevalue,'String',num2str(WIRELESSRANGE/CONVERSIONFACT));
displayString(['Wireless Range changed to
',num2str(WIRELESSRANGE/CONVERSIONFACT)]);

global BOUNDARY;
if( WIRELESSRANGE < BOUNDARY )
    WIRELESSRANGE = BOUNDARY;
end

for i = 1:length(terminal)
for j = 1:length(terminal(i).wirelessagent)
if( terminal(i).wirelessagent(j) ~= 0 )
            ag(terminal(i).wirelessagent(j)).terminalreached = false;
end
end
    terminal(i).wirelessoccupied = 0;
end


% --- Executes on button press in data1.
function data1_Callback(hObject, eventdata, handles)
% hObject    handle to data1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
sendData(1,get(hObject,'Value'));


% --- Executes on button press in data2.
function data2_Callback(hObject, eventdata, handles)
% hObject    handle to data2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sendData(2,get(hObject,'Value'));


% --- Executes on button press in data3.
function data3_Callback(hObject, eventdata, handles)
% hObject    handle to data3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sendData(3,get(hObject,'Value'));


% --- Executes on button press in data4.
function data4_Callback(hObject, eventdata, handles)
% hObject    handle to data4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sendData(4,get(hObject,'Value'));


% --- Executes on button press in data5.
function data5_Callback(hObject, eventdata, handles)
% hObject    handle to data5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sendData(5,get(hObject,'Value'));


% --- Executes on button press in data6.
function data6_Callback(hObject, eventdata, handles)
% hObject    handle to data6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sendData(6,get(hObject,'Value'));


% --- Executes on button press in data7.
function data7_Callback(hObject, eventdata, handles)
% hObject    handle to data7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sendData(7,get(hObject,'Value'));
```

```
% --- Executes on button press in data8.
function data8_Callback(hObject, eventdata, handles)
% hObject    handle to data8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sendData(8,get(hObject,'Value'));


% --- Send Data from terminals
function sendData(tnum,val)
% tnum       terminal number
% val        true false
global server;
global terminal;
persistent packetnumber;
if( terminal(tnum).operational )
if( val )
        displayString(['Data transfer from Access Point ',
num2str(tnum) ,' introduced']);
        size = 100000000000;
        origin = tnum;
        destination = -1;
        delay = 10;
        agent = 0;
        priority = 2;
        bandwidth = [];

        packet =
newPacket(size,origin,destination,delay,agent,priority,bandwidth);
        queuePacket(packet,-1);
        packetnumber = length(server(terminal(tnum).server).packets);

else
        displayString(['Data transfer from Access Point ',
num2str(tnum) ,' stopped']);
        server(terminal(tnum).server).packets(packetnumber).percent =
100;
end
end


% --- Executes when user attempts to close apCntr.
function apCntr_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to apCntr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
set(hObject,'Visible','off');

function packetNums = findPackets(tnum)

global terminal;
```

```
global server;

packetNums = [];
servNum = terminal(tnum).server;
for i=1:length(server(servNum).packets)
if server(servNum).packets(i).origin == tnum
if server(servNum).packets(i).agent < 0
            packetNums = [packetNums, i];
end
end
end
function equip = equipments(NE)
%
% EQUIPMENTS This function initializes the equipments
%
% USAGE:
%   equip = equipments(NE)
%
%       equip       output structure
%       NE          number of equipment
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 13 October 2006
% Version History:
%
% Last Modified: 13 October 2006
%
for i = 1:NE
    equip(i).x = 0;                     % x location
    equip(i).y = 0;                     % y location
    equip(i).power = 1e4;               % power consumption
    equip(i).status = false;            % on off
    equip(i).type = [];                 % type: general fore fighting
etc....
    equip(i).handle = 0;                % graphics
    equip(i).numberHandle = 0;          % graphics
end

function varargout = firecntr(varargin)
% FIRECNTR M-file for firecntr.fig
%      FIRECNTR, by itself, creates a new FIRECNTR or raises the
existing
%      singleton*.
%
%      H = FIRECNTR returns the handle to a new FIRECNTR or the handle
to
%      the existing singleton*.
%
%      FIRECNTR('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in FIRECNTR.M with the given input
arguments.
%
```

```
%      FIRECNTR('Property','Value',...) creates a new FIRECNTR or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before firecntr_OpeningFunction gets called.
An
%      unrecognized property name or invalid value makes property
application
%       stop.  All inputs are passed to firecntr_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help firecntr

% Last Modified by GUIDE v2.5 19-Sep-2006 09:46:20

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @firecntr_OpeningFcn, ...
'gui_OutputFcn',  @firecntr_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before firecntr is made visible.
function firecntr_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to firecntr (see VARARGIN)

% Choose default command line output for firecntr
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
% UIWAIT makes firecntr wait for user response (see UIRESUME)
% uiwait(handles.fireCntr);


% --- Outputs from this function are returned to the command line.
function varargout = firecntr_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in listoffires.
function listoffires_Callback(hObject, eventdata, handles)
% hObject    handle to listoffires (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns listoffires contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
listoffires


% --- Executes during object creation, after setting all properties.
function listoffires_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listoffires (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% % --- Executes on button press in initiateFire.
% function initiateFire_Callback(hObject, eventdata, handles)
% % hObject    handle to initiateFire (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
% fireDetails();


% --- Executes on button press in stop.
```

```
function stop_Callback(hObject, eventdata, handles)
% hObject    handle to stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
simulationCommands('StopFire');


function dispLocation_Callback(hObject, eventdata, handles)
% hObject    handle to dispLocation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of dispLocation as text
%        str2double(get(hObject,'String')) returns contents of
dispLocation as a double


% --- Executes during object creation, after setting all properties.
function dispLocation_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dispLocation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in selectLocation.
function selectLocation_Callback(hObject, eventdata, handles)
% hObject    handle to selectLocation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
mainSimGUIHandle = findall(0, 'tag', 'mainGUI');
mainSimGUIHandleList = guidata(mainSimGUIHandle);
figure(mainSimGUIHandle);

k = waitforbuttonpress;
point = get(mainSimGUIHandleList.mainAxis,'CurrentPoint');
handles.UserData{1} = [point(1,1) point(1,2)];
str = sprintf('[%3.1f %3.1f]',handles.UserData{1});
set(handles.dispLocation,'String',str);

guidata(hObject,handles);

fireCntrHandle = findall(0, 'tag', 'fireCntr');
figure(fireCntrHandle);
```

```
function rosSmoke_Callback(hObject, eventdata, handles)
% hObject    handle to rosSmoke (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of rosSmoke as text
%        str2double(get(hObject,'String')) returns contents of rosSmoke
as a double


% --- Executes during object creation, after setting all properties.
function rosSmoke_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rosSmoke (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function rosFire_Callback(hObject, eventdata, handles)
% hObject    handle to rosFire (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of rosFire as text
%        str2double(get(hObject,'String')) returns contents of rosFire
as a double


% --- Executes during object creation, after setting all properties.
function rosFire_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rosFire (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in startFire.
function startFire_Callback(hObject, eventdata, handles)
% hObject    handle to startFire (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
simulationCommands('StartFire');
set(handles.output,'Visible','off');


% --- Executes when user attempts to close fireCntr.
function fireCntr_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to fireCntr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
set(hObject,'Visible','off');

function fireEquip = fireEquipments(NE)
%
% FIREEQUIPMENTS This function initializes all the booleans of fire
% fighting equipments
%
% USAGE:
%   equip = equipments(NE)
%
%       equip       output structure
%       NE          number of equipment
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 28 November 2006
% Version History:
%
% Last Modified: 12 December 2006
%

for i = 1:NE
    fireEquip(i).x = 0;                     % x location
    fireEquip(i).y = 0;                     % y location
    fireEquip(i).power = 0;                 % power consumption
    fireEquip(i).status = true;             % working not working
    fireEquip(i).type = 'firefighting';     % type: general fire
fighting etc....
    fireEquip(i).handle = 0;                % graphics
    fireEquip(i).numberHandle = 0;          % graphics
end

function machine = machines(NM)
%
% MACHINES This function initializes all the booemans associated with
% machine in the simulation environment
%
```

```
% USAGE:
%   machine = machines(NM)
%
%       machine         output structure
%       NM              number of machines
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 17 July 2006
% Version History:
%
% Last Modified: 7 January 2007
%

for i = 1:NM
    machine(i).status = {'on'};             % on, repair, off
    machine(i).timeforinspection = 15;      % time required for
inspection from xml
    machine(i).inspectionreqd = false;      % is inspection required
    machine(i).agent = 0;                   % agent ispecting
    machine(i).occupied = false;            % machine is being
inspected
    machine(i).sensorData = false;          % sensor data transfer
started?
    machine(i).sensorDataInterrupted...
        = false;                            % sensor data interrupted
% search for new route
    machine(i).fault = false;               % is there a fault inthe
machine
    machine(i).justOnce = false;            % internal use
    machine(i).power = 1e4;                 % power consumption
    machine(i).dataSizeTrans = 0;           % internal use

    machine(i).bwRecord(1) = 0;             % record of bandwidth
availabel to the sensor
    machine(i).marker = 0;                  % internal use
    machine(i).bwRecordTime = 0;            % record time

    machine(i).health = 1;                  % health
    machine(i).healthDecayRate = .1;        % x100 percent/s
    machine(i).on(1) = 1;                   % on off satus record
    machine(i).onTime = 0;                  % time

    machine(i).handle = 0;                  % graphics
    machine(i).numberHandle = 0;            % graphics

    machine(i).maintenanceData =
xmlInfoMaintenance('Maintenance_EquipID_No1.xml',i); % maintenance
information
    machine(i).failureData = [];
    machine(i).faultfound = 0;
end
```

```
function varargout = machinecntr(varargin)
% MACHINECNTR M-file for machinecntr.fig
%      MACHINECNTR, by itself, creates a new MACHINECNTR or raises the
existing
%      singleton*.
%
%      H = MACHINECNTR returns the handle to a new MACHINECNTR or the
handle to
%      the existing singleton*.
%
%      MACHINECNTR('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in MACHINECNTR.M with the given input
arguments.
%
%      MACHINECNTR('Property','Value',...) creates a new MACHINECNTR or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before machinecntr_OpeningFunction gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to machinecntr_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help machinecntr

% Last Modified by GUIDE v2.5 02-Aug-2006 13:10:14

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @machinecntr_OpeningFcn, ...
'gui_OutputFcn',  @machinecntr_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before machinecntr is made visible.
function machinecntr_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to machinecntr (see VARARGIN)

% Choose default command line output for machinecntr

handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes machinecntr wait for user response (see UIRESUME)
% uiwait(handles.machinecntr);

% --- Outputs from this function are returned to the command line.
function varargout = machinecntr_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in Machine1.
function Machine1_Callback(hObject, eventdata, handles)
% hObject    handle to Machine1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
startSensorData(1,hObject);

% --- Executes on button press in Machine2.
function Machine2_Callback(hObject, eventdata, handles)
% hObject    handle to Machine2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
startSensorData(2,hObject);

% --- Executes on button press in Machine3.
function Machine3_Callback(hObject, eventdata, handles)
% hObject    handle to Machine3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
startSensorData(3,hObject);


% --- Executes on button press in Machine4.
```

```
function Machine4_Callback(hObject, eventdata, handles)
% hObject    handle to Machine4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
startSensorData(4,hObject);

% --- Executes on button press in Machine5.
function Machine5_Callback(hObject, eventdata, handles)
% hObject    handle to Machine5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
startSensorData(5,hObject);

% --------------------------------
function startSensorData(mcn,hObject)

global server;
global terminal;
global machine;
global WIRELESSRANGE;

global pValueIdx;
global parameterValues;

machineCntrHandle = findall(0,'tag','machineCntr');
machineCntrHandleList = guidata(machineCntrHandle);

machine(mcn).sensorData = get(hObject,'Value');

distMat = [];
termMat = [];
for i=1:length(terminal)
if distanceBetween(terminal(i),machine(mcn)) < WIRELESSRANGE ...
&& strcmpi(terminal(i).type,'Access Point')
        distMat = [distMat, distanceBetween(terminal(i),machine(mcn))];
        termMat = [termMat, i];
end
end

[dist,idx1] = sort(distMat);

flag = 0;
for i=1:length(dist)
    tnum = termMat(idx1(i));
if terminal(tnum).operational ...
&& distanceBetween(terminal(tnum),machine(mcn)) < WIRELESSRANGE
if machine(mcn).sensorData
            size = 1e15;
            origin = tnum;
            destination = -1;
            delay = 10;
            agent = -mcn;
            priority = parameterValues(pValueIdx,23);
```

```
            bandwidth = [];

            packet =
newPacket(size,origin,destination,delay,agent,priority,bandwidth);
            packet = queuePacket(packet,-1);

            machineCntrHandleList.packetID(mcn) = packet.id;
            guidata(hObject,machineCntrHandleList);

            machine(mcn).sensorDataInterrupted = false;
            displayMachineSensorData(mcn,tnum,true);
            updateNetworkGraph(machine(mcn),terminal(tnum),1);
else
            servNum = terminal(tnum).server;
            handles = guidata(hObject);
            idNumber = handles.packetID(mcn);

            r = find([server(servNum).packets.id] == idNumber);

if ~isempty(r)
                server(servNum).packets(r).percent = 100;
                server(servNum).packets(r).delay = 0;
                displayMachineSensorData(mcn,tnum,false);
                updateNetworkGraph(machine(mcn),terminal(tnum),0);
end
end
        flag = 1;
break;
end
end

if ~flag
if get(hObject,'Value')
        displayString(['AP not operational or out of Wireless Range.'...
' Sensor data cannot be transferred']);
else
        displayMachineSensorData(mcn,false);
end
end


% --- Executes on button press in faultmac1.
function faultmac1_Callback(hObject, eventdata, handles)
% hObject    handle to faultmac1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of faultmac1
enableFault(1,get(hObject,'Value'));

% --- Executes on button press in faultmac2.
function faultmac2_Callback(hObject, eventdata, handles)
% hObject    handle to faultmac2 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of faultmac2
enableFault(2,get(hObject,'Value'));

% --- Executes on button press in faultmac3.
function faultmac3_Callback(hObject, eventdata, handles)
% hObject    handle to faultmac3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of faultmac3
enableFault(3,get(hObject,'Value'));

% --- Executes on button press in faultmac4.
function faultmac4_Callback(hObject, eventdata, handles)
% hObject    handle to faultmac4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of faultmac4
enableFault(4,get(hObject,'Value'));

% --- Executes on button press in faultmac5.
function faultmac5_Callback(hObject, eventdata, handles)
% hObject    handle to faultmac5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of faultmac5
enableFault(5,get(hObject,'Value'));

% --- enableFault function
function enableFault(mcn,val)
% mcn        machine number
% val        value 1, 0
global machine
machine(mcn).fault = val;
if( val )
    displayString(['Fault introduced in Machine ', num2str(mcn)]);
%     if ~machine(mcn).sensorData

    freeAgents = find2Agents(mcn);

    machine(mcn).justOnce = true;
    machine(mcn).status = {'off','repair'};

    object = 'machine';
    data =
insertWorkflow(struct('subject','agent','sno',freeAgents,'object',objec
t,...
```

```
'ono',mcn,'priority',4,'type','troubleshoot','coordinate',0,'status','t
oStart','userData',[]));
    setupOrderStack(object,data);

%     end
else
    displayString(['Fault from Machine ', num2str(mcn) ,' removed']);
end


% --- Display mahcine sensor data
function displayMachineSensorData(varargin)
% macNum      Machine numner
% value       true/false
%
if nargin == 3
    macNum = varargin{1};
    tnum = varargin{2};
    value = varargin{3};

if( value )
        displayString(['Sensor Data transfer from Machine
',num2str(macNum),' routed through AP ',num2str(tnum)]);
else
        displayString(['Sensor Data transfer from Machine
',num2str(macNum),' stopped']);
end
elseif nargin == 2
    macNum = varargin{1};
    value = varargin{2};

if( value )
        displayString(['Sensor Data transfer from Machine
',num2str(macNum),' started']);
else
        displayString(['Sensor Data transfer from Machine
',num2str(macNum),' stopped']);
end
end




% --- Executes when user attempts to close machineCntr.
function machineCntr_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to machineCntr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
set(hObject,'Visible','off');
```

```
% --- Executes on button press in intelMaint.
function intelMaint_Callback(hObject, eventdata, handles)
% hObject    handle to intelMaint (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of intelMaint
global INTELLIGENTMAINTENANCE;

INTELLIGENTMAINTENANCE = get(hObject,'Value');

function server = servers(NS)
%
% SERVERS This function initializes the servers in the simulation
% environment
%
% USAGE:
%   server = servers(NS)
%
%       server      output structure
%       NS          number of servers
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 17 July 2006
% Version History:
%
% Last Modified: 5 January 2007
%
for i = 1:NS
    server(i).packets(1) = newPacket(0,0,-1,0,0,2,[]);

    server(i).operational = true;                               % server
operational
    server(i).transmitted = false;                              %
    server(i).time = clock;                                     %
    server(i).percent = 0;                                      %
    server(i).delay = 1;                                        %
    server(i).data = 1000;                                      %
    server(i).received = false;                                 %
    server(i).terminals = [];                                   % terminals
connected to the server

    server(i).bwRecord(1,1) = 0;                                % bandwidth
ustilized record
    server(i).bwRecord(2,1) = 0;                                % record

end

function varargout = servcntr(varargin)
% SERVCNTR M-file for servcntr.fig
%      SERVCNTR, by itself, creates a new SERVCNTR or raises the
existing
```

```
%      singleton*.
%
%      H = SERVCNTR returns the handle to a new SERVCNTR or the handle
to
%      the existing singleton*.
%
%      SERVCNTR('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in SERVCNTR.M with the given input
arguments.
%
%      SERVCNTR('Property','Value',...) creates a new SERVCNTR or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before servcntr_OpeningFunction gets called.
An
%      unrecognized property name or invalid value makes property
application
%       stop.  All inputs are passed to servcntr_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help servcntr

% Last Modified by GUIDE v2.5 13-Jul-2006 16:27:45

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @servcntr_OpeningFcn, ...
'gui_OutputFcn',  @servcntr_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before servcntr is made visible.
```

```
function servcntr_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to servcntr (see VARARGIN)

% Choose default command line output for servcntr
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes servcntr wait for user response (see UIRESUME)
% uiwait(handles.servCntr);

% --- Outputs from this function are returned to the command line.
function varargout = servcntr_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in serv1Operational.
function serv1Operational_Callback(hObject, eventdata, handles)
% hObject    handle to serv1Operational (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of serv1Operational
changeServStatus(1,get(hObject,'Value'))

% --- Executes on button press in serv2operational.
function serv2Operational_Callback(hObject, eventdata, handles)
% hObject    handle to serv2operational (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of serv2operational
changeServStatus(2,get(hObject,'Value'))

% --- Executes on button press in serv3operational.
function serv3Operational_Callback(hObject, eventdata, handles)
% hObject    handle to serv3operational (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of serv3operational
changeServStatus(3,get(hObject,'Value'));
```

```
% --- change operational status
function changeServStatus(num,val)
% num         server number
% val         true/false
%
global server;
global terminal;

apCntrHandle  = findall(0,'tag','apCntr');
apCntrHandleList = guidata(apCntrHandle);
servCntrHandle = findall(0,'tag','servCntr');
servCntrHandleList = guidata(servCntrHandle);
str = ['serv',num2str(num),'Operational'];

if( val ~= server(num).operational )
if( val )
        server(num).operational = true;
        set(servCntrHandleList.(str),'BackgroundColor',[0.9255
0.9137    0.8471]);
        set(server(num).handle,'FaceColor',[1 1 0]);
for i=1:length(server(num).terminals)
            terminal(server(num).terminals(i)).operational = true;
            string2 = ['MyButton',num2str(server(num).terminals(i))];
            set(apCntrHandleList.(string2),'Value',0);

apcntr([string2,'_Callback'],apCntrHandleList.(string2),1,apCntrHandleL
ist);
end
else
        server(num).operational = false;
        set(servCntrHandleList.(str),'BackgroundColor','r');
        set(server(num).handle,'FaceColor',[.7 .7 .7]);
for i=1:length(server(num).terminals)
            terminal(server(num).terminals(i)).operational = false;
            string2 = ['MyButton',num2str(server(num).terminals(i))];
            set(apCntrHandleList.(string2),'Value',1);

apcntr([string2,'_Callback'],apCntrHandleList.(string2),1,apCntrHandleL
ist);
end
end
end


function serv1staticLoad_Callback(hObject, eventdata, handles)
% hObject    handle to serv1staticLoad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of serv1staticLoad as
text
```

```
%        str2double(get(hObject,'String')) returns contents of
serv1staticLoad as a double


% --- Executes during object creation, after setting all properties.
function serv1staticLoad_CreateFcn(hObject, eventdata, handles)
% hObject    handle to serv1staticLoad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function serv2staticLoad_Callback(hObject, eventdata, handles)
% hObject    handle to serv2staticLoad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of serv2staticLoad as
text
%        str2double(get(hObject,'String')) returns contents of
serv2staticLoad as a double


% --- Executes during object creation, after setting all properties.
function serv2staticLoad_CreateFcn(hObject, eventdata, handles)
% hObject    handle to serv2staticLoad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function serv3staticLoad_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to serv3staticLoad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of serv3staticLoad as
text
%        str2double(get(hObject,'String')) returns contents of
serv3staticLoad as a double


% --- Executes during object creation, after setting all properties.
function serv3staticLoad_CreateFcn(hObject, eventdata, handles)
% hObject    handle to serv3staticLoad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes when user attempts to close servCntr.
function servCntr_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to servCntr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
set(hObject,'Visible','off');

function terminal = terminals(NT,NW)
%
% TERMINALS function to initialize terminals structure
%
% USAGE:
%   terminal = TERMINALS(NT,NW)
%
%       terminal      the output structure
%       NT            number of wireless Access Points
%       NW            number of workstations
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 17 July 2005
% Version History:
%
% Last Modified: 14 December 2006
```

```
%
global WIRELESSRANGE;

for i = 1:NT
    terminal(i).occupied = false;                   % is terminal
occupied
    terminal(i).operational = true;                 % is terminal
operational
    terminal(i).agent = 0;                          % agent accessing
the terminal
    terminal(i).waitno = 0;                         % number of agents
waiting in line
    terminal(i).waitagent(1) = 0;                   % lits of agents
waiting in line
    terminal(i).wirelessagent(1) = 0;               % list of wireless
agents accessing the terminal
    terminal(i).wireless = true;                    % terminal wireless
hub with range
    terminal(i).wirelessoccupied = 0;               % number of agents
accessing the wireless terminal
    terminal(i).wirelessrange = WIRELESSRANGE;      % wireless range of
a terminal can be varied individually

    terminal(i).packets(1).size = 0;                % Size in bytes
    terminal(i).packets(1).origin = 0;              % Transmitting
Terminal
    terminal(i).packets(1).destination = 0;         % Recieving
Terminal
    terminal(i).packets(1).percent = 0;             % Percent sent
    terminal(i).packets(1).transmitted = false;     % Packets sent to
this terminal
    terminal(i).packets(1).time = clock;            % time of last
iteration
    terminal(i).packets(1).delay = 0;               % Delay due to
stages of progression
    terminal(i).packets(1).agent = 0;               % ower fo the
packet
    terminal(i).packets(1).priority = 2;            % priority of the
packet

    terminal(i).server = [];                        % server the
terminal is connected to
    terminal(i).bwRecord = 0;                       % record of the
bandwidth availabel to the connected entity
    terminal(i).bwRecordTime = 0;                   % record time
    terminal(i).markerTerm = 0;                     % record
    terminal(i).agRecord = [];                      % agent
transmitting
    terminal(i).type = 'Access Point';              % type of terminal
AP,workstation
    terminal(i).handle = 0;                         % graphics

    terminal(i).Util = 0;
    terminal(i).PercentUtil = 0;
```

```
end

for i=NT+1:NT+NW
    terminal(i).occupied = false;                    % is terminal
occupied
    terminal(i).operational = true;                  % is terminal
operational
    terminal(i).agent = 0;                           % agent accessing
the terminal
    terminal(i).waitno = 0;                          % number of agents
waiting in line
    terminal(i).waitagent(1) = 0;                    % lits of agents
waiting in line
    terminal(i).wirelessagent(1) = 0;                % list of wireless
agents accessing the terminal
    terminal(i).wireless = false;                    % terminal wireless
hub with range
    terminal(i).wirelessoccupied = 0;                % number of agents
accessing the wireless terminal
    terminal(i).wirelessrange = WIRELESSRANGE;

    terminal(i).packets(1).size = 0;                 % Size in bytes
    terminal(i).packets(1).origin = 0;               % Transmitting
Terminal
    terminal(i).packets(1).destination = 0;          % Recieving
Terminal
    terminal(i).packets(1).percent = 0;              % Percent sent
    terminal(i).packets(1).transmitted = false;      % Packets sent to
this terminal
    terminal(i).packets(1).time = clock;             % time of last
iteration
    terminal(i).packets(1).delay = 0;                % Delay due to
stages of progression
    terminal(i).packets(1).agent = 0;
    terminal(i).packets(1).priority = 2;

    terminal(i).server = [];                         % server the
terminal is connected to
    terminal(i).bwRecord = 0;                        % record of the
bandwidth availabel to the connected entity
    terminal(i).bwRecordTime = 0;                    % record time
    terminal(i).markerTerm = 0;                      % record
    terminal(i).type = 'Workstation';               % type of terminal
AP,workstation
    terminal(i).handle = 0;                          % graphics
    terminal(i).numberHandle = 0;                    % graphics
end

function varargout = powercntr(varargin)
% POWERCNTR M-file for powercntr.fig
%      POWERCNTR, by itself, creates a new POWERCNTR or raises the
existing
%      singleton*.
%
```

```
%      H = POWERCNTR returns the handle to a new POWERCNTR or the
handle to
%      the existing singleton*.
%
%      POWERCNTR('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in POWERCNTR.M with the given input
arguments.
%
%      POWERCNTR('Property','Value',...) creates a new POWERCNTR or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before powercntr_OpeningFunction gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to powercntr_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help powercntr

% Last Modified by GUIDE v2.5 13-Jul-2006 14:35:31

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @powercntr_OpeningFcn, ...
'gui_OutputFcn',  @powercntr_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before powercntr is made visible.
function powercntr_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to powercntr (see VARARGIN)

% Choose default command line output for powercntr

handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes powercntr wait for user response (see UIRESUME)
% uiwait(handles.powercntr);

% --- Outputs from this function are returned to the command line.
function varargout = powercntr_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on slider movement.
function pAvailable_Callback(hObject, eventdata, handles)
% hObject    handle to pAvailable (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of
%        slider
global powerGenerated;
global powerGeneratedPercent;
global deckMax;
global POWERSTATUSCHANGE;

powerGeneratedPercent = get(hObject,'Value');
set(handles.pAvailabletxt,'String',[num2str(powerGeneratedPercent),'%']
);
powerGenerated = deckMax.power*powerGeneratedPercent/100;
POWERSTATUSCHANGE = true;

displayString(['Power Availability changes to
',num2str(powerGenerated),' W']);

% --- Executes during object creation, after setting all properties.
function pAvailable_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pAvailable (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject,'Backgroundcolor',[.9 .9 .9]);
else

set(hObject,'Backgroundcolor',get(0,'defaultUicontrolBackgroundcolor'));
end
set(hObject,'Value',0);

% --- Executes on slider movement.
function pUtilized_Callback(hObject, eventdata, handles)
% hObject    handle to pUtilized (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


% --- Executes during object creation, after setting all properties.
function pUtilized_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pUtilized (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject,'Backgroundcolor',[.9 .9 .9]);
else

set(hObject,'Backgroundcolor',get(0,'defaultUicontrolBackgroundcolor'));
end
set(hObject,'Value',0);

% --- Executes when user attempts to close powerCntr.
function powerCntr_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to powerCntr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
set(hObject,'Visible','off');
```

```
function out = insertWorkflow(data)
%
% INSERTWORKFLOW This function inserts the new workflows into the
existing
% list
%
% USAGE:
%   out = insertWorkflow(data)
%
%       data                    data relating to the current scenario
%       out                     same data with ID
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 13 October 2006
% Version History:
%
% Last Modified: 14 December 2006
%

global WORKSTACK;

id = rand;
for i=1:length(WORKSTACK)
while id == WORKSTACK(i).id
        id = rand;
end
end

if isempty(WORKSTACK)
    tempw = data;
    tempw.('id') = id;
    WORKSTACK = tempw;
else
    tempw = data;
    tempw.('id') = id;
    WORKSTACK(end+1) = tempw;
end

out = WORKSTACK(end);

function out = insertEmergencyScenario(emergencyData)
%
% INSERTEMERGENCYSCENARIOS This function inserts the new emergency
scenario
% to the existing list. It also initiates the plotting
%
% USAGE:
%   out = insertEmergencyScenario(emergencyData)
%
%       emergencyData           data relating to the current scenario
%       out                     same data with ID
%
% -----------
```

```
% Created by: Vishal Mahulkar
% Created on: 15 November 2006
% Version History:
%
% Last Modified: 12 December 2006
%

global emergencyScenarios;

mainSimGUIHandle = findall(0,'tag','mainGUI');
mainSimGUIHandleList = guidata(mainSimGUIHandle);

id = single(rand);
for i=1:length(emergencyScenarios)
while id == emergencyScenarios(i).id
        id = rand;
end
end

if strcmpi(emergencyData.type,'fire')
    emergencyData.userData.radiusF = 0;
    emergencyData.userData.radiusS = 0;

% plotting fire and smoke
    xycircledataF =
plot_circle(emergencyData.x,emergencyData.y,emergencyData.userData.radi
usF,100,'interval');
    xycircledataS =
plot_circle(emergencyData.x,emergencyData.y,emergencyData.userData.radi
usS,100,'interval');

    axes(mainSimGUIHandleList.mainAxis);hold on;
    emergencyData.handleF =
fill(xycircledataF(:,1),xycircledataF(:,2),'r');
    emergencyData.handleS =
fill(xycircledataS(:,1),xycircledataS(:,2),'bl');
    set(emergencyData.handleF,'FaceAlpha',0.2);
    set(emergencyData.handleS,'FaceAlpha',0.2);
    hold off;
end

if isempty(emergencyScenarios)
    tempw = emergencyData;
    tempw.('id') = id;
    emergencyScenarios = tempw;
else
    tempw = emergencyData;
    tempw.('id') = id;
    emergencyScenarios(end+1) = tempw;
end

out = emergencyScenarios(end);
```

```
function removeCompletedWorkflow(agn)
%
% REMOVECOMPLETEDWORKFLOW This function is used to remove the completed
% workflow of an agent and also the corresponding workflow in the
global
% WORKSTACK is marked completed.
%
% USAGE:
%   removeCompletedWorkflow(agn)
%
%       agn         agent number
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 13 October 2006
% Version History:
%
% Last Modified: 14 December 2006
%

global WORKSTACK;

global ag;

global simulationTime;

data = ag(agn).workstack(1);

r = [];
if ~isempty(WORKSTACK)
    r = find([WORKSTACK.id] == data.id);
end

if ~isempty(r)
    WORKSTACK(r).status = 'done';
    WORKSTACK(r).userData = [WORKSTACK(r).userData,simulationTime];
%     if r == 1
%         if length(workstack) == 1
%             workstack = [];
%         else
%             workstack = workstack(2:end);
%         end
%     elseif r == length(workstack)
%         workstack = workstack(1:end-1);
%     else
%         workstack = [workstack(1:r-1),workstack(r+1:end)];
%     end
end


if( length(ag(agn).workstack) == 1 )
    ag(agn).workstack = [];
else
```

```
    ag(agn).workstack = ag(agn).workstack(2:end);
end

function removeEmergencyScenario(fireID,varargin)
%
% REMOVEEMERGENCYSCENARIO This function is used to remove the emergency
% scenario dealt with from the list
%
% USAGE:
%   removeEmergencyScenario(fireID)
%
%       fireID      id of the fire to be removed
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 15 November 2006
% Version History:
%
% Last Modified: 15 November 2006
%

global emergencyScenarios;

r = [];
if ~isempty(emergencyScenarios)
    r = find( abs([emergencyScenarios.id] - single(fireID)) < 1e-7);
end

if ~isempty(r)
if r == 1
if length(emergencyScenarios) == 1
            emergencyScenarios = [];
else
            emergencyScenarios = emergencyScenarios(2:end);
end
elseif r == length(emergencyScenarios)
        emergencyScenarios = emergencyScenarios(1:end-1);
else
        emergencyScenarios = [emergencyScenarios(1:r-
1),emergencyScenarios(r+1:end)];
end
end

function varargout = mainGUI(varargin)
% MAINGUI M-file for mainGUI.fig
%      MAINGUI, by itself, creates a new MAINGUI or raises the existing
%      singleton*.
%
%      H = MAINGUI returns the handle to a new MAINGUI or the handle to
%      the existing singleton*.
%
%      MAINGUI('CALLBACK',hObject,eventData,handles,...) calls the
local
```

```
%      function named CALLBACK in MAINGUI.M with the given input
arguments.
%
%      MAINGUI('Property','Value',...) creates a new MAINGUI or raises
the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before mainGUI_OpeningFunction gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to mainGUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help mainGUI

% Last Modified by GUIDE v2.5 28-Jun-2007 16:41:04

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @mainGUI_OpeningFcn, ...
'gui_OutputFcn',  @mainGUI_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before mainGUI is made visible.
function mainGUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to mainGUI (see VARARGIN)

% Choose default command line output for mainGUI
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

global time_reduction;
% Efficiency
axes(handles.axes1)
h = plot(0,0,'r');
set(h,'Tag','Current');
title('Agent 1','FontSize',8)
ylabel('% Work Completed','FontSize',8)
xlabel('Agent WorkFlow Progress: (s)','FontSize',8)
set(handles.axes1,'FontSize',8)
set(handles.axes1,'XLim',[0 100]/time_reduction,'YLim',[0 100]);

axes(handles.axes2)
totalPlotHandle = plot(0,0,0,0,'r');
ylabel({'% of Current Work','Completed by all Agents'},'FontSize',8);
xlabel('Agent WorkFlow Progress: (s)','FontSize',8);
set(totalPlotHandle(1),'LineStyle','-.','Tag','Baseline')
set(totalPlotHandle(2),'Tag','Current')
set(handles.axes2,'FontSize',8)
set(handles.axes2,'XLim',[0 100]/time_reduction,'YLim',[0 1]);

axes(handles.indUtilization)
h = plot(0,0,'r');
set(h,'Tag','Current');
title('Agent 1','FontSize',8)
ylabel('% Utilization','FontSize',8)
xlabel('Time','FontSize',8)
set(handles.indUtilization,'FontSize',8)
set(handles.indUtilization,'XLim',[0 100]/time_reduction,'YLim',[0 1]);

axes(handles.totUtilization)
h = plot(0,0,'r');
set(h,'Tag','Current');
ylabel('% Utilization Total','FontSize',8)
xlabel('Time','FontSize',8)
set(handles.totUtilization,'FontSize',8)
set(handles.totUtilization,'XLim',[0 100]/time_reduction,'YLim',[0 1]);

set(handles.utiPanel,'visible','off');

RGB = imread('PULOGO.jpg');
axes(handles.puLogo1);
imshow(RGB);
axis('tight')
axes(handles.puLogo2);
imshow(RGB);
axis('tight')
% % reposition
% set(0,'Units','Normalized');
% set(hObject,'Units','Normalized')
```

```
% scnsize = get(0,'ScreenSize');
% pos1  = [0,...
%     -.01,...
%     1,...
%     .9];
% set(hObject,'Position',pos1)

% UIWAIT makes mainGUI wait for user response (see UIRESUME)
% uiwait(handles.mainGUI);


% --- Outputs from this function are returned to the command line.
function varargout = mainGUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
% reposition
if 1
set(0,'Units','Normalized');
set(hObject,'Units','Normalized')
scnsize = get(0,'ScreenSize');
pos1  = [0,...
    0,...
    1,...
    .955];
set(hObject,'Position',pos1)
end
if 0
    pos = get(hObject,'Position');
    posNew = [10,3,pos(3)-pos(1),pos(4)-pos(2)];
    set(hObject,'Position',posNew)
end
% Main
setupCommands('Init');


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1
global ag;
global agTotalUtiData;
global agTotalUtiTime;
```

```
global agTotalTaskCTime;
global agTotalTaskCData;

global time_reduction;

currAgentSel = get(handles.popupmenu1, 'Value');
if strcmpi(get(handles.effPanel,'Visible'),'on')
    axes(handles.axes1);
    child = get(handles.axes1,'Children');

set(child,'Xdata',ag(currAgentSel).ptcTime/time_reduction,'Ydata',ag(cu
rrAgentSel).percentTaskComp);
    xlabel('Agent WorkFlow Progress: Current, (s)')
    ylabel('% Work Completed')
    title(['Agent ',num2str(currAgentSel)])

    child = get(handles.axes2,'Children');

set(child,'Xdata',agTotalTaskCTime/time_reduction,'Ydata',agTotalTaskCD
ata);
    xlabel('Time')
    ylabel('% Work Completed')
elseif strcmpi(get(handles.utiPanel,'Visible'),'on')
    axes(handles.indUtilization);
    child = get(handles.indUtilization,'Children');

set(child,'Xdata',ag(currAgentSel).utiTime/time_reduction,'Ydata',ag(cu
rrAgentSel).utilization);
    xlabel('Time')
    ylabel('% Utilization')
    title(['Agent ',num2str(currAgentSel)])

    child = get(handles.totUtilization,'Children');

set(child,'Xdata',agTotalUtiTime/time_reduction,'Ydata',agTotalUtiData);
    xlabel('Time')
    ylabel('Average Utilization')
end


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
set(hObject, 'String', {'Agent 1', 'Agent 2', 'Agent 3', 'Agent 4',
'Agent 5', ...
'Agent 6', 'Agent 7', 'Agent 8', 'Agent 9', 'Agent 10'} );


% --- Executes on button press in effPanelButton.
function effPanelButton_Callback(hObject, eventdata, handles)
% hObject    handle to effPanelButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.effPanel,'visible','on');
set(handles.utiPanel,'visible','off');


% --- Executes on button press in utiPanelButton.
function utiPanelButton_Callback(hObject, eventdata, handles)
% hObject    handle to utiPanelButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.effPanel,'visible','off');
set(handles.utiPanel,'visible','on');


% --- Executes on selection change in simStatusList.
function simStatusList_Callback(hObject, eventdata, handles)
% hObject    handle to simStatusList (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns simStatusList
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
simStatusList


% --- Executes during object creation, after setting all properties.
function simStatusList_CreateFcn(hObject, eventdata, handles)
% hObject    handle to simStatusList (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in mainDisplay.
```

```
function mainDisplay_Callback(hObject, eventdata, handles)
% hObject    handle to mainDisplay (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.mainPanel,'visible','on');
set(handles.networkPanel,'visible','off');


% --- Executes on button press in networkDisplay.
function networkDisplay_Callback(hObject, eventdata, handles)
% hObject    handle to networkDisplay (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.mainPanel,'visible','off');
set(handles.networkPanel,'visible','on');


% --- Executes on button press in serverControl.
function serverControl_Callback(hObject, eventdata, handles)
% hObject    handle to serverControl (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
servcntr;


% --- Executes on button press in apControl.
function apControl_Callback(hObject, eventdata, handles)
% hObject    handle to apControl (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apcntr;


% --- Executes on button press in agControl.
function agControl_Callback(hObject, eventdata, handles)
% hObject    handle to agControl (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
agcntr;


% --- Executes on button press in machineControl.
function machineControl_Callback(hObject, eventdata, handles)
% hObject    handle to machineControl (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
machinecntr;


% --- Executes on button press in powerControl.
function powerControl_Callback(hObject, eventdata, handles)
% hObject    handle to powerControl (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
powercntr;


% --- Executes on button press in fireControl.
function fireControl_Callback(hObject, eventdata, handles)
% hObject    handle to fireControl (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
firecntr;


% --- Executes on button press in exit.
function exit_Callback(hObject, eventdata, handles)
% hObject    handle to exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
simulationCommands('Exit');


% --- Executes on button press in pauseButton.
function pauseButton_Callback(hObject, eventdata, handles)
% hObject    handle to pauseButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
simulationCommands('Pause');


% --- Executes on button press in start.
function start_Callback(hObject, eventdata, handles)
% hObject    handle to start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'enable','off');
simulationCommands('Start');


% --- Executes on button press in process.
function process_Callback(hObject, eventdata, handles)
% hObject    handle to process (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
postProcessing();


% --------------------------------------------------------------------
function saveFigure_Callback(hObject, eventdata, handles)
% hObject    handle to saveFigure (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fileformat = ['fig';'jpg';'eps'];
```

```
[filename, pathname, filterindex] = uiputfile( ...
{'*.fig','Figures (*.fig)'; ...
'*.jpg','Jpeg (*.jpg)'; ...
'*.eps','EPS (*.eps)';}, ...
'Save as');
if( filename )
    saveas(handles.output,filename,fileformat(filterindex,:));
end


% --------------------------------------------------------------------
function File_Callback(hObject, eventdata, handles)
% hObject    handle to File (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function loadSetUp_Callback(hObject, eventdata, handles)
% hObject    handle to loadSetUp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
setupCommands('Load');


% --------------------------------------------------------------------
function setUp_Callback(hObject, eventdata, handles)
% hObject    handle to setUp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function newSetUp_Callback(hObject, eventdata, handles)
% hObject    handle to newSetUp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
setupCommands('New');


% --------------------------------------------------------------------
function reset_Callback(hObject, eventdata, handles)
% hObject    handle to reset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function saveSetUp_Callback(hObject, eventdata, handles)
% hObject    handle to saveSetUp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
setupCommands('Save');


% --------------------------------------------------------------------
function resetSetup_Callback(hObject, eventdata, handles)
% hObject    handle to resetSetup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
setupCommands('Reset');


function stepSize_Callback(hObject, eventdata, handles)
% hObject    handle to stepSize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of stepSize as text
%        str2double(get(hObject,'String')) returns contents of stepSize
as a double
global deltaTime;
deltaTime = str2double(get(hObject,'String'));


% --- Executes during object creation, after setting all properties.
function stepSize_CreateFcn(hObject, eventdata, handles)
% hObject    handle to stepSize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
global deltaTime;
global time_reduction;
set(hObject,'String',num2str(deltaTime/time_reduction));




% --- Executes when user attempts to close mainGUI.
function mainGUI_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to mainGUI (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
setupCommands('Close');
```

```
function setupCommands(command, varargin)
%
% SETUPCOMMANDS Tis function has commands to set up various aspects of
the
% simulation. All these functions are called up from the main GUI.
%
% USAGE:
%   setupCommands('Init')
%       Initializes the simulation
%
%   setupCommands('Load')
%       Load geometry through a userinput
%
%   setupcommands('New')
%       Initializes for setting up of a new geometry
%
%   setupCommands('Reset')
%       Reset the geometry and make it ready to load new geometry
%
%   setupcommands('Save')
%       Save the current Geometry
%
%   setupCommands('Close')
%       Close the simulation windows
%
%   setupCommands('GenerateNetworkStructure')
%       To set up the network structure. This is called from
%       setupcommands('New')
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 17 July 2006
% Modified by: Robin Kusmanto
%
%
% Last Modified: 20 May 2009
%
global ag;
global terminal;
global server;
global machine;
global watchLoc;
global fireEquip;
global equip;

global tsze;

global deckMax;
global deckAct;

global powerGeneratedPercent;
global powerGenerated;
global powerUtilized;
global POWERSTATUSCHANGE;
```

```
global roomNumbering;
global connectionMatrix;
global doorMatrix;
global doorCenters;
global networkGraph;

global CONVERSIONFACT;

global pValueIdx
global parameterValues;

mainSimGUIHandle = findall(0,'tag','mainGUI');
mainSimGUIHandleList = guidata(mainSimGUIHandle);
%% INIT

if strcmpi(command,'Init')
% Location set up
    set(mainSimGUIHandleList.networkPanel,'visible','off');
    axes(mainSimGUIHandleList.mainAxis);

    displayString('Simulation SetUp Begins');

    apCntrHandle = apcntr();
    apCntrHandleList = guidata(apCntrHandle);
    set(apCntrHandle,'Visible','Off');

    agCntrHandle = agcntr();
    agCntrHandleList = guidata(agCntrHandle);
    set(agCntrHandle,'Visible','Off');
    set(agCntrHandleList.a1pda,'Value',ag(1).pda);
    set(agCntrHandleList.a2pda,'Value',ag(2).pda);
    set(agCntrHandleList.a3pda,'Value',ag(3).pda);
    set(agCntrHandleList.a4pda,'Value',ag(4).pda);
    set(agCntrHandleList.a5pda,'Value',ag(5).pda);
    set(agCntrHandleList.a6pda,'Value',ag(6).pda);
    set(agCntrHandleList.a7pda,'Value',ag(7).pda);
    set(agCntrHandleList.a8pda,'Value',ag(8).pda);
    set(agCntrHandleList.a9pda,'Value',ag(9).pda);
    set(agCntrHandleList.a10pda,'Value',ag(10).pda);


    machineCntrHandle = machinecntr();
    set(machineCntrHandle,'Visible','Off');

    powerCntrHandle = powercntr();
    set(powerCntrHandle,'Visible','Off');

    servCntrHandle = servcntr();
    set(servCntrHandle,'Visible','Off');

    workflowGUICntrHandle = workflowGUI();
```

```
    set(workflowGUICntrHandle,'Visible','Off');

    fireCntrHandle = firecntr();
    set(fireCntrHandle,'Visible','Off');

%% LOAD

elseif strcmpi(command,'Load')
% --- Load an existing geometry

    roomNumbering = [];
    connectionMatrix = [];
    doorMatrix = [];
    doorCenters = [];

    cla(mainSimGUIHandleList.mainAxis);
    file = 'try1.mat';
if ~isequal(file, 0)
        displayString(['Loading file -> ', file,' : Predefined setup
with geometry and crew, terminal, server and machine locations']);

% load File
        load(file);
        displayString('Checking for necessary variables');

% check for vairables
if( ismember({'selectedBW','finalBW','finalBoundary','doorsBW','numNode
s','roomNumbers','agLoc',...
'termLoc','servLoc','macLoc','wLoc','fireEquipLoc','equipLoc','serv2ter
mConn'},who) )
            displayString('Variables found - displaying Geometry');

% calculate the geometry and connectivity
            [connectionMatrix] =
generateGraph(numNodes,roomNumbers,selectedBW,finalBW,finalBoundary);
            [doorMatrix,doorCenters,doorsBW] =
getDoors(numNodes,roomNumbers,selectedBW,finalBW);

% set up the size of objects in simulation
            [r,c] = find(finalBW == 1);
            tsze = floor(max( max(r)-min(r), max(c)-min(c) )/90);

% Display the environment
            axes(mainSimGUIHandleList.mainAxis);

            hlayout = imshow(finalBW);hold on;
            set(hlayout,'tag','Environment');
            axis tight;
            axis on;
            xlabel(['The dimensions are in
(value)x',num2str(1/CONVERSIONFACT),' feet']);
```

```
% room numbers and boundaries
            roomNumbering = roomNumbers;
            [fB,fL,fN,fA] = bwboundaries(finalBW);
            [sB,sL,sN,sA] = bwboundaries(selectedBW,'noholes');
            [dB,dL,dN,dA] = bwboundaries(doorsBW,'noholes');
            colors=['b''g''r''c''m''y'];

% initialize network graph
            [networkGraph] = getNetworkGraph();

% initialize power structure
            deckMax = deckStructureMaximum();
            deckAct = deckStructure();

% Display the environment
            axes(mainSimGUIHandleList.mainAxis);hold on;

% display doors
            doorcen = [];
for k = 1:length(dB)
                boundary = dB{k};
                doorcen(k).x = mean(boundary(:,2));
                doorcen(k).y = mean(boundary(:,1));
                cidx = mod(k,length(colors))+1;
                hdoors = plot(boundary(:,2), boundary(:,1),
'b','LineWidth',2);
                set(hdoors,'tag','Door');
                rndRow = ceil(length(boundary)/(mod(rand*k,7)+1));
                col = boundary(rndRow,2); row = boundary(rndRow,1);
end
% keyboard
% display aps, ags, servers, machines, watch locations
for i = 1:length(terminal)
                terminal(i).x = termLoc(i).x;
                terminal(i).y = termLoc(i).y;
if strcmpi(terminal(i).type,'Access Point')
                    terminal(i).handle = fill([terminal(i).x-
tsze,terminal(i).x-tsze,terminal(i).x+tsze,terminal(i).x+tsze],...
                        [terminal(i).y-
tsze,terminal(i).y+tsze,terminal(i).y+tsze,terminal(i).y-tsze],'g');
                    htext = text(terminal(i).x+2*tsze,
terminal(i).y+2*tsze/8,num2str(i));
elseif strcmpi(terminal(i).type,'Workstation')
                    fill([terminal(i).x-tsze,terminal(i).x-
tsze,terminal(i).x,terminal(i).x],...
                        [terminal(i).y-
tsze,terminal(i).y+tsze,terminal(i).y+tsze,terminal(i).y-tsze],'g',...

[terminal(i).x,terminal(i).x,terminal(i).x+tsze,terminal(i).x+tsze],...
                        [terminal(i).y-
tsze,terminal(i).y+tsze,terminal(i).y+tsze,terminal(i).y-tsze],'b');

                    terminal(i).handle = patch([terminal(i).x-
tsze,terminal(i).x-tsze,terminal(i).x+tsze,terminal(i).x+tsze],...
```

```
                        [terminal(i).y-
tsze,terminal(i).y+tsze,terminal(i).y+tsze,terminal(i).y-tsze],'g');
                    alpha(terminal(i).handle,.5);
                    htext = text(terminal(i).x+2*tsze,
terminal(i).y+2*tsze/8,num2str(i));
end

set(htext,'FontSize',8,'FontWeight','bold','Color','black','tag',['term
inal(',num2str(i),')']);

                set(terminal(i).handle,'UserData','Terminal');

set(terminal(i).handle,'tag',['terminal(',num2str(i),')']);

set(terminal(i).handle,'ButtonDownFcn',{@displayAPInfo,i});

deckMax.room(roomNumbering(round(terminal(i).y),round(terminal(i).x))).
terminals ...
                    =
[deckMax.room(roomNumbering(round(terminal(i).y),round(terminal(i).x)))
.terminals, i];

deckAct.room(roomNumbering(round(terminal(i).y),round(terminal(i).x))).
terminals ...
                    =
[deckAct.room(roomNumbering(round(terminal(i).y),round(terminal(i).x)))
.terminals, i];


updateNetworkGraph(terminal(i),roomNumbering(round(terminal(i).y),round
(terminal(i).x)),1);
end
for i = 1:length(server)
                server(i).x = servLoc(i).x;
                server(i).y = servLoc(i).y;
                server(i).handle = fill([server(i).x-tsze,server(i).x-
tsze,server(i).x+tsze,server(i).x+tsze],...
                    [server(i).y-
tsze,server(i).y+tsze,server(i).y+tsze,server(i).y-tsze],'y');
                htext = text(server(i).x+2*tsze,
server(i).y+2*tsze/8,num2str(i));

set(htext,'FontSize',8,'FontWeight','bold','Color','black','tag',['serv
er(',num2str(i),')']);

                set(server(i).handle,'UserData','Server');
                set(server(i).handle,'tag',['server(',num2str(i),')']);

set(server(i).handle,'ButtonDownFcn',{@displayServerInfo,i});
                server(i).terminals = serv2termConn{i};

for k = 1:length(server(i).terminals)
                    firstPt.x = server(i).x;
                    firstPt.y = server(i).y;
```

```
                    secondPt.x = terminal(server(i).terminals(k)).x;
                    secondPt.y = terminal(server(i).terminals(k)).y;
                    terminal(server(i).terminals(k)).server = i;
                    hnetwork = line([firstPt.x
secondPt.x],[firstPt.y,secondPt.y],'LineStyle','--','color','red');
                    set(hnetwork,'tag','Network Structure');

updateNetworkGraph(server(i),terminal(server(i).terminals(k)),1)
end


deckMax.room(roomNumbering(round(server(i).y),round(server(i).x))).serv
ers ...
                    =
[deckMax.room(roomNumbering(round(server(i).y),round(server(i).x))).ser
vers, i];

deckAct.room(roomNumbering(round(server(i).y),round(server(i).x))).serv
ers ...
                    =
[deckAct.room(roomNumbering(round(server(i).y),round(server(i).x))).ser
vers, i];


updateNetworkGraph(server(i),roomNumbering(round(server(i).y),round(ser
ver(i).x)),1);
end
for i = 1:length(machine)
                machine(i).x = macLoc(i).x;
                machine(i).y = macLoc(i).y;
                machine(i).handle = fill([machine(i).x-
2*tsze,machine(i).x-2*tsze,machine(i).x+2*tsze,machine(i).x+2*tsze],...
                    [machine(i).y-
2*tsze,machine(i).y+tsze,machine(i).y+tsze,machine(i).y-2*tsze],'b');
                htext = text(machine(i).x-tsze, machine(i).y-
2*tsze/8,num2str(i));

set(htext,'FontSize',8,'FontWeight','bold','Color','yellow','tag',['mac
hine(',num2str(i),')']);

                set(machine(i).handle,'UserData','Machine');

set(machine(i).handle,'tag',['machine(',num2str(i),')']);

set(machine(i).handle,'ButtonDownFcn',{@displayMachineInfo,i});


updateNetworkGraph(machine(i),roomNumbering(round(machine(i).y),round(m
achine(i).x)),1);
end
for i = 1:length(watchLoc)
                watchLoc(i).x = wLoc(i).x;
                watchLoc(i).y = wLoc(i).y;
```

```
                watchLoc(i).handle = fill([watchLoc(i).x-
2*tsze,watchLoc(i).x-
2*tsze,watchLoc(i).x+2*tsze,watchLoc(i).x+2*tsze],...
                    [watchLoc(i).y-
2*tsze,watchLoc(i).y+2*tsze,watchLoc(i).y+2*tsze,watchLoc(i).y-
2*tsze],'w');
                htext = text(watchLoc(i).x-tsze, watchLoc(i).y-
2*tsze/8,num2str(i));

set(htext,'FontSize',8,'FontWeight','bold','Color','black','tag',['watc
hLoc(',num2str(i),')']);

                set(watchLoc(i).handle,'UserData','Watch');

set(watchLoc(i).handle,'tag',['watchLoc(',num2str(i),')']);

set(watchLoc(i).handle,'ButtonDownFcn',{@displayWLInfo,i});


updateNetworkGraph(watchLoc(i),roomNumbering(round(watchLoc(i).y),round
(watchLoc(i).x)),1);
end
for i = 1:length(ag)
                ag(i).x = agLoc(i).x;
                ag(i).y = agLoc(i).y;
                ag(i).handle = fill([ag(i).x-2*tsze/3,ag(i).x-
2*tsze/3,ag(i).x+2*tsze/3,ag(i).x+2*tsze/3],...
                    [ag(i).y-
2*tsze/3,ag(i).y+2*tsze/3,ag(i).y+2*tsze/3,ag(i).y-2*tsze/3],'r');
                ag(i).numberHandle = text(ag(i).x-tsze,
ag(i).y,num2str(i));

set(ag(i).numberHandle,'FontSize',8,'FontWeight','bold','Color','k');

                set(ag(i).handle,'UserData','Agent');
                set(ag(i).handle,'tag',['ag(',num2str(i),')']);
                set(ag(i).handle,'ButtonDownFcn',{@displayAgentInfo,i});


updateNetworkGraph(ag(i),roomNumbering(round(ag(i).y),round(ag(i).x)),1
);
end
for i = 1:length(fireEquip)
                fireEquip(i).x = fireEquipLoc(i).x;
                fireEquip(i).y = fireEquipLoc(i).y;
                fireEquip(i).handle = fill([fireEquip(i).x-
2*tsze/4,fireEquip(i).x-
2*tsze/4,fireEquip(i).x+2*tsze/4,fireEquip(i).x+2*tsze/4],...
                    [fireEquip(i).y-
2*tsze/4,fireEquip(i).y+2*tsze/4,fireEquip(i).y+2*tsze/4,fireEquip(i).y
-2*tsze/4],'c');
                fireEquip(i).numberHandle = text(fireEquip(i).x-tsze,
fireEquip(i).y,num2str(i));
```

```
set(fireEquip(i).numberHandle,'FontSize',8,'FontWeight','bold','Color',
'k');

                set(fireEquip(i).handle,'UserData','fireEquip');

set(fireEquip(i).handle,'tag',['fireEquip(',num2str(i),')']);

set(fireEquip(i).handle,'ButtonDownFcn',{@displayFireEquipInfo,i});


updateNetworkGraph(fireEquip(i),roomNumbering(round(fireEquip(i).y),rou
nd(fireEquip(i).x)),1);
end
for i = 1:length(equip)
                equip(i).x = equipLoc(i).x;
                equip(i).y = equipLoc(i).y;
                equip(i).handle = fill([equip(i).x-2*tsze/4,equip(i).x-
2*tsze/4,equip(i).x+2*tsze/4,equip(i).x+2*tsze/4],...
                    [equip(i).y-
2*tsze/4,equip(i).y+2*tsze/4,equip(i).y+2*tsze/4,equip(i).y-
2*tsze/4],'c');
                equip(i).numberHandle = text(equip(i).x-tsze,
equip(i).y,num2str(i));

set(equip(i).numberHandle,'FontSize',8,'FontWeight','bold','Color','k');

                set(equip(i).handle,'UserData','Equip');
                set(equip(i).handle,'tag',['equip(',num2str(i),')']);

set(equip(i).handle,'ButtonDownFcn',{@displayEquipInfo,i});


updateNetworkGraph(equip(i),roomNumbering(round(equip(i).y),round(equip
(i).x)),1);
end

% power level calculations
            powerGeneratedPercent = 80;
            powerGenerated = deckMax.power*powerGeneratedPercent/100;

            deckAct = calcDeckStatus(powerGenerated,deckAct);
            powerUtilized = deckAct.power*100/powerGenerated;

            powerSimPOE(mainSimGUIHandleList);

            POWERSTATUSCHANGE = false;

% display room Number and Priority
for k = 1:length(sB)
                boundary = sB{k};
                rndRow = ceil(length(boundary)/(mod(rand*k,7)+1));
                col = boundary(rndRow,2); row = boundary(rndRow,1);
```

```
                h = text(min(boundary(:,2))+10, min(boundary(:,1))+10,
[num2str(sL(row,col)),'(',num2str(deckMax.room(k).priority),')']);

set(h,'color',colors(3),'FontSize',8,'FontWeight','bold');
end

% initial movement of the agents in absense of orders
for i=1:length(ag)
                ag(i).finalterminal = getClosestTerminalforAgent(i);
end

            displayString(['Simulation number ',num2str(pValueIdx)]);
% set factors
            set_factors;

% Enable Controls

set(findobj(findall(0,'tag','mainGUI'),'enable','off'),'enable','on')

set(findobj(findall(0,'tag','apCntr'),'enable','off'),'enable','on')

set(findobj(findall(0,'tag','agCntr'),'enable','off'),'enable','on')

set(findobj(findall(0,'tag','machineCntr'),'enable','off'),'enable','on
')

set(findobj(findall(0,'tag','powerCntr'),'enable','off'),'enable','on')

set(findobj(findall(0,'tag','servCntr'),'enable','off'),'enable','on')

set(findobj(findall(0,'tag','fireCntr'),'enable','off'),'enable','on')

else
            displayString('Error loading file, the file is not in
specified format or may not contain necessary data');
end
end
% ---

%% NEW

elseif strcmpi(command, 'New')
% --- Create a new geometry

    roomNumbering = [];
    connectionMatrix = [];
    doorMatrix = [];
    doorCenters = [];

    cla(mainSimGUIHandleList.mainAxis);
% get the rooms and calculate geometry and connectivity
    [numNodes,roomNumbers,selectedBW,finalBW,finalBoundary,selectFig] =
sRoom();
```

```
    [connectionMatrix] =
generateGraph(numNodes,roomNumbers,selectedBW,finalBW,finalBoundary);
    [doorMatrix,doorCenters,doorsBW] =
getDoors(numNodes,roomNumbers,selectedBW,finalBW);

% set up the size of objects in simulation
    [r,c] = find(finalBW == 1);
    tsze = floor(3200/max( max(r)-min(r), max(c)-min(c) ));

% show the selection
    axes(mainSimGUIHandleList.mainAxis);
    imshow(finalBW);hold on;
    axis tight;
    axis on;
    xlabel(['The dimensions are in
(value)x',num2str(1/CONVERSIONFACT),' feet']);

% room numbers and boundaries
    roomNumbering = roomNumbers;
    [fB,fL,fN,fA] = bwboundaries(finalBW);
    [sB,sL,sN,sA] = bwboundaries(selectedBW,'noholes');
    [dB,dL,dN,dA] = bwboundaries(doorsBW,'noholes');
    colors=['b''g''r''c''m''y'];

% initialize network graph
    [networkGraph] = getNetworkGraph();

% initialize power structure
    deckMax = deckStructureMaximum();
    deckAct = deckStructure();

% locate various objects
for k = 1:length(dB)
        boundary = dB{k};
        doorcen(k).x = mean(boundary(:,2));
        doorcen(k).y = mean(boundary(:,1));
        cidx = mod(k,length(colors))+1;
        hdoors = plot(boundary(:,2), boundary(:,1), 'b','LineWidth',2);
        set(hdoors,'tag','Door');
        rndRow = ceil(length(boundary)/(mod(rand*k,7)+1));
        col = boundary(rndRow,2); row = boundary(rndRow,1);
end

    hquest = questdlg('Select Access point Locations','Access
Points','Ok','Ok');
    termStr = 'No';

if( strcmp(termStr,'No' ) )
        i = 1;
while (i < length(terminal)+1 )
            [xpos,ypos] = ginput(1);
if ( selectedBW(round(ypos),round(xpos)) ~= 0)
                terminal(i).x = xpos(1);
```

```
                terminal(i).y = ypos(1);

if strcmpi(terminal(i).type,'Access Point')
                    terminal(i).handle = fill([terminal(i).x-
tsze,terminal(i).x-tsze,terminal(i).x+tsze,terminal(i).x+tsze],...
                        [terminal(i).y-
tsze,terminal(i).y+tsze,terminal(i).y+tsze,terminal(i).y-tsze],'g');
                    htext = text(terminal(i).x+2*tsze,
terminal(i).y+2*tsze/8,num2str(i));
elseif strcmpi(terminal(i).type,'Workstation')
                    fill([terminal(i).x-tsze,terminal(i).x-
tsze,terminal(i).x,terminal(i).x],...
                        [terminal(i).y-
tsze,terminal(i).y+tsze,terminal(i).y+tsze,terminal(i).y-tsze],'g',...

[terminal(i).x,terminal(i).x,terminal(i).x+tsze,terminal(i).x+tsze],...
                        [terminal(i).y-
tsze,terminal(i).y+tsze,terminal(i).y+tsze,terminal(i).y-tsze],'b');

                    terminal(i).handle = patch([terminal(i).x-
tsze,terminal(i).x-tsze,terminal(i).x+tsze,terminal(i).x+tsze],...
                        [terminal(i).y-
tsze,terminal(i).y+tsze,terminal(i).y+tsze,terminal(i).y-tsze],'g');
                    alpha(terminal(i).handle,.5);

                    htext = text(terminal(i).x+2*tsze,
terminal(i).y+2*tsze/8,num2str(i));
end

set(htext,'FontSize',8,'FontWeight','bold','Color','black');

                set(terminal(i).handle,'UserData','Terminal');

set(terminal(i).handle,'tag',['terminal(',num2str(i),')']);

set(terminal(i).handle,'ButtonDownFcn',{@displayAPInfo,i});


deckMax.room(roomNumbering(round(terminal(i).y),round(terminal(i).x))).
terminals ...
                    =
[deckMax.room(roomNumbering(round(terminal(i).y),round(terminal(i).x)))
.terminals, i];

deckAct.room(roomNumbering(round(terminal(i).y),round(terminal(i).x))).
terminals ...
                    =
[deckAct.room(roomNumbering(round(terminal(i).y),round(terminal(i).x)))
.terminals, i];


updateNetworkGraph(terminal(i),roomNumbering(round(terminal(i).y),round
(terminal(i).x)),1);
```

```
                i = i + 1;
end
end
end

    hquest = questdlg('Select Server Locations','Servers','Ok','Ok');
    serverStr = 'No';

if( strcmp(serverStr,'No' ) )
        i = 1;
while (i < length(server)+1 )
            [xpos,ypos] = ginput(1);
if ( selectedBW(round(ypos),round(xpos)) ~= 0)
                server(i).x = xpos(1);
                server(i).y = ypos(1);

                server(i).handle = fill([server(i).x-tsze,server(i).x-
tsze,server(i).x+tsze,server(i).x+tsze],...
                    [server(i).y-
tsze,server(i).y+tsze,server(i).y+tsze,server(i).y-tsze],'y');
                htext = text(server(i).x+2*tsze,
server(i).y+2*tsze/8,num2str(i));

set(htext,'FontSize',8,'FontWeight','bold','Color','black');

                set(server(i).handle,'UserData','Server');
                set(server(i).handle,'tag',['server(',num2str(i),')']);

set(server(i).handle,'ButtonDownFcn',{@displayServerInfo,i});

deckMax.room(roomNumbering(round(server(i).y),round(server(i).x))).serv
ers ...
                    =
[deckMax.room(roomNumbering(round(server(i).y),round(server(i).x))).ser
vers, i];

deckAct.room(roomNumbering(round(server(i).y),round(server(i).x))).serv
ers ...
                    =
[deckAct.room(roomNumbering(round(server(i).y),round(server(i).x))).ser
vers, i];


updateNetworkGraph(server(i),roomNumbering(round(server(i).y),round(ser
ver(i).x)),1);
                i = i + 1;
end
end
end

    hquest = questdlg('Select Machine Locations','Machines','Ok','Ok');
    machineStr = 'No';
```

```
if( strcmp(machineStr,'No' ) )
        i = 1;
while (i < length(machine)+1 )
            [xpos,ypos] = ginput(1);
if ( selectedBW(round(ypos),round(xpos)) ~= 0)
                machine(i).x = xpos(1);
                machine(i).y = ypos(1);

                machine(i).handle = fill([machine(i).x-
2*tsze,machine(i).x-2*tsze,machine(i).x+2*tsze,machine(i).x+2*tsze],...
                    [machine(i).y-
2*tsze,machine(i).y+tsze,machine(i).y+tsze,machine(i).y-2*tsze],'b');
                htext = text(machine(i).x-tsze, machine(i).y-
2*tsze/8,num2str(i));

set(htext,'FontSize',8,'FontWeight','bold','Color','yellow');

                set(machine(i).handle,'UserData','Machine');

set(machine(i).handle,'tag',['machine(',num2str(i),')']);

set(machine(i).handle,'ButtonDownFcn',{@displayMachineInfo,i});


updateNetworkGraph(machine(i),roomNumbering(round(machine(i).y),round(m
achine(i).x)),1);
                i = i + 1;
end
end
end

    hquest = questdlg('Select Watch Locations','Watch','Ok','Ok');
    watchStr = 'No';

if( strcmp(watchStr,'No' ) )
        i = 1;
while (i < length(machine)+1 )
            [xpos,ypos] = ginput(1);
if ( selectedBW(round(ypos),round(xpos)) ~= 0)
                watchLoc(i).x = xpos(1);
                watchLoc(i).y = ypos(1);

                watchLoc(i).handle = fill([watchLoc(i).x-
2*tsze,watchLoc(i).x-
2*tsze,watchLoc(i).x+2*tsze,watchLoc(i).x+2*tsze],...
                    [watchLoc(i).y-
2*tsze,watchLoc(i).y+2*tsze,watchLoc(i).y+2*tsze,watchLoc(i).y-
2*tsze],'w');
                htext = text(watchLoc(i).x-tsze, watchLoc(i).y-
2*tsze/8,num2str(i));

set(htext,'FontSize',8,'FontWeight','bold','Color','black');
```

```
                set(watchLoc(i).handle,'UserData','Watch');

set(watchLoc(i).handle,'tag',['watchLoc(',num2str(i),')']);

set(watchLoc(i).handle,'ButtonDownFcn',{@displayWLInfo,i});


updateNetworkGraph(watchLoc(i),roomNumbering(round(watchLoc(i).y),round
(watchLoc(i).x)),1);
                i = i + 1;
end
end
end

    hquest = questdlg('Select Agent Locations','Agents','Ok','Ok');
    agStr = 'No';

if( strcmp(agStr,'No' ) )
        i = 1;
while (i < length(ag)+1 )
            [xpos,ypos] = ginput(1);
if ( selectedBW(round(ypos),round(xpos)) ~= 0)
                ag(i).x = xpos(1);
                ag(i).y = ypos(1);

                ag(i).handle = fill([ag(i).x-2*tsze/3,ag(i).x-
2*tsze/3,ag(i).x+2*tsze/3,ag(i).x+2*tsze/3],...
                    [ag(i).y-
2*tsze/3,ag(i).y+2*tsze/3,ag(i).y+2*tsze/3,ag(i).y-2*tsze/3],'r');
                ag(i).numberHandle = text(ag(i).x-tsze,
ag(i).y,num2str(i));

set(ag(i).numberHandle,'FontSize',8,'FontWeight','bold','Color','k');

                set(ag(i).handle,'UserData','Agent');
                set(ag(i).handle,'tag',['ag(',num2str(i),')']);
                set(ag(i).handle,'ButtonDownFcn',{@displayAgentInfo,i});


updateNetworkGraph(ag(i),roomNumbering(round(ag(i).y),round(ag(i).x)),1
);
                i = i + 1;
end
end
end

    hquest = questdlg('Select Fire Equipment
Locations','Equipments','Ok','Ok');
    fireEquipStr = 'No';

if( strcmp(fireEquipStr,'No' ) )
        i = 1;
while (i < length(fireEquip)+1 )
```

```
            [xpos,ypos] = ginput(1);
if ( selectedBW(round(ypos),round(xpos)) ~= 0)
                fireEquip(i).x = xpos(1);
                fireEquip(i).y = ypos(1);

                fireEquip(i).handle = fill([fireEquip(i).x-
2*tsze/4,fireEquip(i).x-
2*tsze/4,fireEquip(i).x+2*tsze/4,fireEquip(i).x+2*tsze/4],...
                    [fireEquip(i).y-
2*tsze/4,fireEquip(i).y+2*tsze/4,fireEquip(i).y+2*tsze/4,fireEquip(i).y
-2*tsze/4],'c');
                fireEquip(i).numberHandle = text(ag(i).x-tsze,
ag(i).y,num2str(i));

set(fireEquip(i).numberHandle,'FontSize',8,'FontWeight','bold','Color',
'k');

                set(fireEquip(i).handle,'UserData','FireEquip');

set(fireEquip(i).handle,'tag',['fireEquip(',num2str(i),')']);

set(fireEquip(i).handle,'ButtonDownFcn',{@displayFireEquipInfo,i});


updateNetworkGraph(fireEquip(i),roomNumbering(round(fireEquip(i).y),rou
nd(fireEquip(i).x)),1);
                i = i + 1;
end
end
end

    hquest = questdlg('Select Equipment
Locations','Equipments','Ok','Ok');
    equipStr = 'No';

if( strcmp(equipStr,'No' ) )
        i = 1;
while (i < length(equip)+1 )
            [xpos,ypos] = ginput(1);
if ( selectedBW(round(ypos),round(xpos)) ~= 0)
                equip(i).x = xpos(1);
                equip(i).y = ypos(1);

                equip(i).handle = fill([equip(i).x-2*tsze/4,equip(i).x-
2*tsze/4,equip(i).x+2*tsze/4,equip(i).x+2*tsze/4],...
                    [equip(i).y-
2*tsze/4,equip(i).y+2*tsze/4,equip(i).y+2*tsze/4,equip(i).y-
2*tsze/4],'c');
                equip(i).numberHandle = text(ag(i).x-tsze,
ag(i).y,num2str(i));

set(equip(i).numberHandle,'FontSize',8,'FontWeight','bold','Color','k');
```

```
                set(equip(i).handle,'UserData','Equip');
                set(equip(i).handle,'tag',['equip(',num2str(i),')']);

set(equip(i).handle,'ButtonDownFcn',{@displayEquipInfo,i});


updateNetworkGraph(equip(i),roomNumbering(round(equip(i).y),round(equip
(i).x)),1);
                i = i + 1;
end
end
end

    delete(selectFig);

for i = 1:length(ag)
        agLoc(i).x = ag(i).x;
        agLoc(i).y = ag(i).y;
end
for i = 1:length(terminal)
        termLoc(i).x = terminal(i).x;
        termLoc(i).y = terminal(i).y;
end
for i = 1:length(server)
        servLoc(i).x = server(i).x;
        servLoc(i).y = server(i).y;
end
for i = 1:length(machine)
        macLoc(i).x = machine(i).x;
        macLoc(i).y = machine(i).y;
end
for i = 1:length(watchLoc)
        wLoc(i).x = watchLoc(i).x;
        wLoc(i).y = watchLoc(i).y;
end
for i = 1:length(fireEquip)
        fireEquipLoc(i).x = fireEquip(i).x;
        fireEquipLoc(i).y = fireEquip(i).y;
end
for i = 1:length(equip)
        equipLoc(i).x = equip(i).x;
        equipLoc(i).y = equip(i).y;
end

    hquest = questdlg('Select Access points and Servers which are to be
connected to each other','Network Structure','Ok','Ok');
% Generate the Network Structure
    setupCommands('GenerateNetworkStructure');

% power level calculations
    powerGeneratedPercent = 80;
    powerGenerated = deckMax.power*powerGeneratedPercent/100;
```

```
    deckAct = myPower(powerGenerated,deckAct);
    powerUtilized = deckAct.power*100/powerGenerated;

    powerSimPOE(mainSimGUIHandleList);

    POWERSTATUSCHANGE = false;

% display room Number and Priority
for k = 1:length(sB)
        boundary = sB{k};
        rndRow = ceil(length(boundary)/(mod(rand*k,7)+1));
        col = boundary(rndRow,2); row = boundary(rndRow,1);
        h = text(min(boundary(:,2))+10, min(boundary(:,1))+10,
[num2str(sL(row,col)),'(',num2str(deckMax.room(k).priority),')']);
        set(h,'color',colors(3),'FontSize',8,'FontWeight','bold');
end

for i = 1:length(server)
        serv2termConn{i} = server(i).terminals;
end

    save
tempMatselectedBWfinalBWfinalBoundarydoorsBWnumNodesroomNumbersagLocter
mLocservLocmacLocwLocfireEquipLocequipLocserv2termConn;

% initial movement of the agents in absense of orders
for i=1:length(ag)
        ag(i).finalterminal = getClosestTerminalforAgent(i);
end

% get the scenario to run
    displayString(['Simulation number ',num2str(pValueIdx)]);

% set factors
    set_factors;

% enable all graphical controls

set(findobj(findall(0,'tag','mainGUI'),'enable','off'),'enable','on')
    set(findobj(findall(0,'tag','apCntr'),'enable','off'),'enable','on')
    set(findobj(findall(0,'tag','agCntr'),'enable','off'),'enable','on')

set(findobj(findall(0,'tag','machineCntr'),'enable','off'),'enable','on
')

set(findobj(findall(0,'tag','powerCntr'),'enable','off'),'enable','on')

set(findobj(findall(0,'tag','servCntr'),'enable','off'),'enable','on')

set(findobj(findall(0,'tag','fireCntr'),'enable','off'),'enable','on')
% ---
%% RESET SETUP
```

```
elseif strcmpi(command, 'Reset')
% --- Reset the system
    cla(mainSimGUIHandleList.mainAxis);
    cla(mainSimGUIHandleList.networkAxis);

    childEffInd =
findobj(get(mainSimGUIHandleList.axes1,'Children'),'Tag','Current');
    set(childEffInd,'XData',0,'YData',0);
    childEffTot =
findobj(get(mainSimGUIHandleList.axes2,'Children'),'Tag','Current');
    set(childEffTot,'XData',0,'YData',0);
    childUtiInd =
findobj(get(mainSimGUIHandleList.indUtilization,'Children'),'Tag','Curr
ent');
    set(childUtiInd,'XData',0,'YData',0);
    childUtiTot =
findobj(get(mainSimGUIHandleList.totUtilization,'Children'),'Tag','Curr
ent');
    set(childUtiTot,'XData',0,'YData',0);

    HsimStat = findall(0, 'tag', 'simulationStatus');
    Hlist = findobj(HsimStat, 'tag', 'simStatusList');
    str{1,1} = sprintf('%011.3f \t \t \t %s',0.0,'Reseting Simulation');
    set(Hlist, 'string',  str, 'listboxtop', max(1, length(str)-1));%,
'value', length(newStr)-offset)
    clear global;
    parameterSetUp1();

set(findobj(findall(0,'tag','mainGUI'),'enable','off'),'enable','off')

set(findobj(findall(0,'tag','apCntr'),'enable','off'),'enable','off')

set(findobj(findall(0,'tag','agCntr'),'enable','off'),'enable','off')

set(findobj(findall(0,'tag','machineCntr'),'enable','off'),'enable','of
f')

set(findobj(findall(0,'tag','powerCntr'),'enable','off'),'enable','off')

set(findobj(findall(0,'tag','servCntr'),'enable','off'),'enable','off')

set(findobj(findall(0,'tag','fireCntr'),'enable','off'),'enable','off')
%     setupCommands('Close');
%     SA_full();

% ---
%% SAVE GEOMETRY

elseif strcmpi(command, 'Save')
% --- Save generated geometry
if ( exist('tempMat.mat') )
        load tempMat.mat;
        delete tempMat.mat;
```

```
uisave({'selectedBW','finalBW','finalBoundary','doorsBW','numNodes','ro
omNumbers','agLoc','termLoc',...
'servLoc','macLoc','wLoc','fireEquipLoc','equipLoc','serv2termConn'});
end;

% ---
%% CLOSE

elseif strcmpi(command, 'Close')
% ---
%     simulationCommands('Exit');
    delete(mainSimGUIHandle);
    ob = findall(0,'tag','fireCntr');
    delete(ob);
    ob = findall(0,'tag','fireDetails');
    delete(ob);
    ob = findall(0,'tag','apCntr');
    delete(ob);
    ob = findall(0,'tag','agCntr');
    delete(ob);
    ob = findall(0,'tag','machineCntr');
    delete(ob);
    ob = findall(0,'tag','powerCntr');
    delete(ob);
    ob = findall(0,'tag','servCntr');
    delete(ob);
    ob = findall(0,'tag','simulationStatus');
    delete(ob);
    ob = findall(0,'tag','efficiency');
    delete(ob);
    ob = findall(0,'tag','workflowGUI');
    delete(ob);
% ---
%% NETWORK STRUCTURE

elseif strcmpi(command, 'GenerateNetworkStructure')
% --- Generate network structure

    displayString('Set up the network structure');
    displayString('Select terminals and servers which are to be
connected to each other');

for i = 1:length(server)
        server(i).terminals = [];
end

    countLine = 0;
    listTerms = [];

while ( countLine ~= length(terminal) )
        displayString('');
        countTerm = 0;
```

```
        countServ = 0;
        k = waitforbuttonpress;

% get objects to be connected
        object1 = get(mainSimGUIHandleList.mainGUI,'CurrentObject');
for i = 1: length(terminal)
if( strcmp('Terminal',get(object1,'UserData')) )
if( (object1 == terminal(i).handle && isempty(listTerms)) || ...
                        (object1 == terminal(i).handle &&
isempty(find(i==listTerms,1))) )
                    firstPt(countLine+1).x = terminal(i).x;
                    firstPt(countLine+1).y = terminal(i).y;
                    termNum = i;
                    countTerm = countTerm + 1;
                    obj1Ocolor = get(object1,'facecolor');
                    set(object1,'facecolor','r');
                    displayString(['Selected terminal no ',num2str(i)]);
break
end
end
end
for i = 1:length(server)
if( strcmp('Server',get(object1,'UserData')) )
if( object1 == server(i).handle )
                    firstPt(countLine+1).x = server(i).x;
                    firstPt(countLine+1).y = server(i).y;
                    servNum = i;
                    countServ = countServ + 1;
                    obj1Ocolor = get(object1,'facecolor');
                    set(object1,'facecolor','r');
                    displayString(['Selected server no ',num2str(i)]);
break
end
end
end

        k = waitforbuttonpress;
        object2 = get(mainSimGUIHandleList.mainGUI,'CurrentObject');
for i = 1: length(terminal)
if( strcmp('Terminal',get(object2,'UserData')) )
if( (object2 == terminal(i).handle && isempty(listTerms)) || ...
                        (object2 == terminal(i).handle &&
isempty(find(i==listTerms,1))) )
                    secondPt(countLine+1).x = terminal(i).x;
                    secondPt(countLine+1).y = terminal(i).y;
                    termNum = i;
                    countTerm = countTerm + 1;
                    obj2Ocolor = get(object2,'facecolor');
                    set(object2,'facecolor','r');
                    displayString(['Selected terminal no ',num2str(i)]);
break
end
end
end
```

```
for i = 1:length(server)
if( strcmp('Server',get(object2,'UserData')) )
if( object2 == server(i).handle )
                    secondPt(countLine+1).x = server(i).x;
                    secondPt(countLine+1).y = server(i).y;
                    servNum = i;
                    countServ = countServ + 1;
                    obj2Ocolor = get(object2,'facecolor');
                    set(object2,'facecolor','r');
                    displayString(['Selected server no ',num2str(i)]);
break
end
end
end
        pause(.1);

% check the objects and make connections
if ( strcmp('Server',get(object1,'UserData')) ||
strcmp('Terminal',get(object1,'UserData')) )
            set(object1,'facecolor',obj1Ocolor);
end
if ( strcmp('Server',get(object2,'UserData')) ||
strcmp('Terminal',get(object2,'UserData')) )
            set(object2,'facecolor',obj2Ocolor);
end
if( countTerm == 1 && countServ == 1 )
            displayString(['Connecting Terminal Number ',
num2str(termNum),' and Server ',num2str(servNum)]);


server(servNum).terminals(length(server(servNum).terminals)+1) =
termNum;
            terminal(termNum).server = servNum;
            listTerms = [listTerms,termNum];
            countLine = countLine + 1;
else
            displayString('Selection was not proper please try again');
end

end

% draw the structure
for i = 1:countLine
        hnetwork = line([firstPt(i).x
secondPt(i).x],[firstPt(i).y,secondPt(i).y],'LineStyle','--
','color','red');
        set(hnetwork,'tag','Network Structure');
end
    displayString('Network Setup done Simulation is ready. Press Start
to begin');

% ---
```

```
%% ELSE

else
% ---
    disp('Command not found');
% ---
end

function premovement(agn,n,init)
%
% PREMOVEMENT This function is used to setup various booleans and final
% destination and path when moving towards an Access Point or a
workstation
%
% USAGE
%   premovement(agn,n,init)
%
%       agn       agent number
%       n         AP number
%       init      1 internal use
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 17 July 2006
% Version History:
%
% Last Modified: 21 December 2006
%
global ag;
global N;
global terminal;
global WIDTH;
global machine;
global BOUNDARY;

% if( agn == 10 || agn == 8 )
%     1;
% end

if(init == 1)
for i = 1:N
%intermediategoals(i);
        ag(i).waittime = 0;
        ag(i).status = 2;
end
else
if( ~ag(agn).pda || ~terminal(n).wireless)
if( n ~= ag(agn).finalterminal || (ag(agn).terminalreached == 0))
            ag(agn).status = 2;
if( n ~= ag(agn).finalterminal)
                ag(agn).preterminal = ag(agn).finalterminal;
end
            ag(agn).finalterminal = n; % input
            ag(agn).destinationreached = false;
```

```
            ag(agn).terminalreached = false;
            ag(agn).machinereached = false;
            ag(agn).equipreached = false;
            ag(agn).emergencyLocreached = false;
            ag(agn).waittime = 0;
            ag(agn).timetermreached = 0;

            ag(agn).timemcreached = 0;

if( ag(agn).machine )
                machine(ag(agn).machine).occupied = false;
end

% previous terminal agents - reduce the wait number
if( ag(agn).preterminal ~= 0 )
                ~isempty(find(terminal(ag(agn).preterminal).waitagent
== agn ));
                terminal(ag(agn).preterminal).agent == agn;
end
if( ~(ag(agn).preterminal == ag(agn).finalterminal &&
distanceBetween(ag(agn),terminal(n)) < BOUNDARY) )
if( ag(agn).preterminal ~= 0 &&
( ~isempty(find(terminal(ag(agn).preterminal).waitagent == agn )) || ...
                    terminal(ag(agn).preterminal).agent == agn ) )
if( ~ag(agn).pda || ~terminal(ag(agn).preterminal).wireless )

for i=1:terminal(ag(agn).preterminal).waitno
if( (agn ~= terminal(ag(agn).preterminal).waitagent(i)) &&...
                                    ag(agn).waitno <
ag(terminal(ag(agn).preterminal).waitagent(i)).waitno)

ag(terminal(ag(agn).preterminal).waitagent(i)).waitno = ...

ag(terminal(ag(agn).preterminal).waitagent(i)).waitno -1;

ag(terminal(ag(agn).preterminal).waitagent(i)).destinationreached =
false;

ag(terminal(ag(agn).preterminal).waitagent(i)).waittime = 0;

if( isempty(ag(terminal(ag(agn).preterminal).waitagent(i)).path))

ag(terminal(ag(agn).preterminal).waitagent(i)).finalx(1) = ...
                                        terminal(ag(agn).preterminal).x
+ ...

ag(terminal(ag(agn).preterminal).waitagent(i)).waitno*WIDTH;

ag(terminal(ag(agn).preterminal).waitagent(i)).finaly(1) = ...
                                        terminal(ag(agn).preterminal).y;
else

ag(terminal(ag(agn).preterminal).waitagent(i)).path(end).x = ...
```

```
                                        terminal(ag(agn).preterminal).x
+ ...

ag(terminal(ag(agn).preterminal).waitagent(i)).waitno*WIDTH;

ag(terminal(ag(agn).preterminal).waitagent(i)).path(end).y = ...
                                        terminal(ag(agn).preterminal).y;
end
end
end
                        waitagent = 0;
                        count = 0;
% update the waitlist
for i=1:terminal(ag(agn).preterminal).waitno
if( (agn ~= terminal(ag(agn).preterminal).waitagent(i)) )
                                count = count +1;
                                waitagent(count) =
terminal(ag(agn).preterminal).waitagent(i);
end
                            np = ag(agn).preterminal;
end

if(terminal(ag(agn).preterminal).waitno ~= 0)
if( terminal(ag(agn).preterminal).waitno == length(waitagent) )
if ( length(waitagent) > 1)

terminal(ag(agn).preterminal).waitagent =
waitagent(2:length(waitagent));
else

terminal(ag(agn).preterminal).waitagent = 0;
end
else
                                terminal(ag(agn).preterminal).waitagent
= waitagent;
end
                            terminal(ag(agn).preterminal).waitno = ...
                                terminal(ag(agn).preterminal).waitno  -1;
else
if( agn == terminal(ag(agn).preterminal).agent )
                                terminal(ag(agn).preterminal).occupied
= false;
                                terminal(ag(agn).preterminal).agent  =  0;
end
end
else
                        wirelessagent = 0;
                        count = 0;
% refresh the list of wireless agents accessing the
% terminal
for i=1:terminal(ag(agn).preterminal).wirelessoccupied
if( (agn ~= terminal(ag(agn).preterminal).wirelessagent(i)) )
                                count = count +1;
```

```
                                    wirelessagent(count) =
terminal(ag(agn).preterminal).wirelessagent(i);
end
end
                          terminal(ag(agn).preterminal).wirelessagent =
wirelessagent;
if( wirelessagent == 0)

terminal(ag(agn).preterminal).wirelessoccupied = 0;
else

terminal(ag(agn).preterminal).wirelessoccupied = count;
end
end
end
end

% if the terminal is occupied, make the agent wait
if( terminal(n).occupied && (terminal(n).agent ~= agn))
                  count = 0;
if( terminal(n).waitno ~= 0)
for i = 1:terminal(n).waitno
if( agn ~= terminal(n).waitagent(terminal(n).waitno) )
                               count = count + 1;
end
end
else
                       count =1;
end
if( count ~= 0 )
                       terminal(n).waitno = terminal(n).waitno+1;
                       terminal(n).waitagent(terminal(n).waitno) = agn;
                       ag(agn).waitno = terminal(n).waitno;

                       ag(agn).path =
getPathforAgent(agn,terminal(n),ag(agn).waitno);
                       ag(agn).watch_step = 1;
                       ag(agn).terminalx = terminal(n).x;
                       ag(agn).terminaly = terminal(n).y;
end
else
                  ag(agn).path = getPathforAgent(agn,terminal(n));
                  ag(agn).watch_step = 1;
                  ag(agn).terminalx = 0;
                  ag(agn).terminaly = 0;
                  ag(agn).waitno = 0;
end
end
else
if( n ~= ag(agn).finalterminal || (ag(agn).terminalreached == 0))
              ag(agn).status = 2;
if( n ~= ag(agn).finalterminal)
                  ag(agn).preterminal = ag(agn).finalterminal;
end
```

```
            ag(agn).finalterminal = n; % input
            ag(agn).destinationreached = false;
            ag(agn).terminalreached = false;
            ag(agn).waittime = 0;
            ag(agn).waitno = 0;
            ag(agn).timetermreached = 0;
            ag(agn).equipreached = false;
            ag(agn).machinereached = false;
            ag(agn).timemcreached = 0;

if( ag(agn).machine )
                machine(ag(agn).machine).occupied = false;
end

% previous terminal agents - reduce the wait number
if( ag(agn).preterminal ~= 0 )
if( ~terminal(ag(agn).preterminal).wireless )
if ( ~isempty(find(terminal(ag(agn).preterminal).waitagent == agn ))
|| ...
                            terminal(ag(agn).preterminal).agent == agn )

for i=1:terminal(ag(agn).preterminal).waitno
if( (agn ~= terminal(ag(agn).preterminal).waitagent(i)) &&...
                                    ag(agn).waitno <
ag(terminal(ag(agn).preterminal).waitagent(i)).waitno)

ag(terminal(ag(agn).preterminal).waitagent(i)).waitno = ...

ag(terminal(ag(agn).preterminal).waitagent(i)).waitno -1;

ag(terminal(ag(agn).preterminal).waitagent(i)).destinationreached =
false;

ag(terminal(ag(agn).preterminal).waitagent(i)).waittime = 0;
end
end
                        waitagent = 0;
                        count = 0;
% move the waiting agents forward
for i=1:terminal(ag(agn).preterminal).waitno
if( (agn ~= terminal(ag(agn).preterminal).waitagent(i)) )
                                count = count +1;
                                waitagent(count) =
terminal(ag(agn).preterminal).waitagent(i);
end

                            np = ag(agn).preterminal;

if( isempty(ag(terminal(ag(agn).preterminal).waitagent(i)).path) )

ag(terminal(ag(agn).preterminal).waitagent(i)).finalx(1) = ...
                                    terminal(ag(agn).preterminal).x
+ ...
```

```
ag(terminal(ag(agn).preterminal).waitagent(i)).waitno*WIDTH;

ag(terminal(ag(agn).preterminal).waitagent(i)).finaly(1) = ...
                                    terminal(ag(agn).preterminal).y;
else

ag(terminal(ag(agn).preterminal).waitagent(i)).path(end).x = ...
                                    terminal(ag(agn).preterminal).x
+ ...

ag(terminal(ag(agn).preterminal).waitagent(i)).waitno*WIDTH;

ag(terminal(ag(agn).preterminal).waitagent(i)).path(end).y = ...
                                    terminal(ag(agn).preterminal).y;
end
end

if(terminal(ag(agn).preterminal).waitno ~= 0)
if( terminal(ag(agn).preterminal).waitno == length(waitagent) )
if ( length(waitagent) > 1)

terminal(ag(agn).preterminal).waitagent =
waitagent(2:length(waitagent));
else

terminal(ag(agn).preterminal).waitagent = 0;
end
else
                                    terminal(ag(agn).preterminal).waitagent
= waitagent;
end
                            terminal(ag(agn).preterminal).waitno = ...
                                terminal(ag(agn).preterminal).waitno  -1;
else
                            terminal(ag(agn).preterminal).occupied =
false;
                            terminal(ag(agn).preterminal).agent = 0;
end
end
else
                    wirelessagent = 0;
                    count = 0;
% refresh the list of wireless agents accessing the
% terminal
for i=1:terminal(ag(agn).preterminal).wirelessoccupied
if( (agn ~= terminal(ag(agn).preterminal).wirelessagent(i)) )
                            count = count +1;
                            wirelessagent(count) =
terminal(ag(agn).preterminal).wirelessagent(i);
end
end
                    terminal(ag(agn).preterminal).wirelessagent =
wirelessagent;
```

```
if( wirelessagent == 0)
                        terminal(ag(agn).preterminal).wirelessoccupied
= 0;
else
                        terminal(ag(agn).preterminal).wirelessoccupied
= count;
end

if ag(agn).lastTerminalAccessed ~= 0
                        wirelessagent = 0;
                        count = 0;
% refresh the list of wireless agents accessing the
% terminal
for i=1:terminal(ag(agn).lastTerminalAccessed).wirelessoccupied
if( (agn ~= terminal(ag(agn).lastTerminalAccessed).wirelessagent(i)) )
                                count = count +1;
                                wirelessagent(count) =
terminal(ag(agn).lastTerminalAccessed).wirelessagent(i);
end
end

terminal(ag(agn).lastTerminalAccessed).wirelessagent = wirelessagent;
if( wirelessagent == 0)

terminal(ag(agn).lastTerminalAccessed).wirelessoccupied = 0;
else

terminal(ag(agn).lastTerminalAccessed).wirelessoccupied = count;
end
end
end
end

            ag(agn).path = getPathforAgent(agn,terminal(n));
            ag(agn).watch_step = 1;
            ag(agn).terminalx = 0;
            ag(agn).terminaly = 0;
            ag(agn).waitno = 0;

end
end
end

function moveover(agn)
%
% MOVEOVER This function is used to make agents move away from a
% workstation after the reporting is completed and if there is other
agents
% waiting in line.
%
% USAGE:
%   moveover(agn)
%
%       agn     agent number that needs to be moved
```

```
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 17 July 2006
% Version History:
%
% Last Modified: 21 December 2006
%

global ag;
global terminal;
global WIDTH;

if( ~ag(agn).pda || ~terminal(ag(agn).finalterminal).wireless )
if(ag(agn).waitno == 0)
        ag(agn).preterminal = ag(agn).finalterminal;

        waitagent = 0;
        count = 0;
for i=1:terminal(ag(agn).finalterminal).waitno
if( (agn ~= terminal(ag(agn).finalterminal).waitagent(i)) &&...
                    ag(agn).waitno <
ag(terminal(ag(agn).finalterminal).waitagent(i)).waitno)
                ag(terminal(ag(agn).finalterminal).waitagent(i)).waitno
= ...

ag(terminal(ag(agn).finalterminal).waitagent(i)).waitno -1;

ag(terminal(ag(agn).finalterminal).waitagent(i)).destinationreached =
false;

ag(terminal(ag(agn).finalterminal).waitagent(i)).waittime = 0;

                count = count +1;
                waitagent(count) =
terminal(ag(agn).finalterminal).waitagent(i);

                np = ag(agn).finalterminal;

if( isempty(ag(terminal(ag(agn).finalterminal).waitagent(i)).path))

ag(terminal(ag(agn).finalterminal).waitagent(i)).finalx(1) = ...
                        terminal(ag(agn).finalterminal).x + ...

ag(terminal(ag(agn).finalterminal).waitagent(i)).waitno*WIDTH;

ag(terminal(ag(agn).finalterminal).waitagent(i)).finaly(1) = ...
                        terminal(ag(agn).finalterminal).y;
else

ag(terminal(ag(agn).finalterminal).waitagent(i)).path(end).x = ...
                        terminal(ag(agn).finalterminal).x + ...
```

```
ag(terminal(ag(agn).finalterminal).waitagent(i)).waitno*WIDTH;

ag(terminal(ag(agn).finalterminal).waitagent(i)).path(end).y = ...
                        terminal(ag(agn).finalterminal).y;
end
end
end

if(terminal(ag(agn).finalterminal).waitno ~= 0)
if( terminal(ag(agn).finalterminal).waitno == length(waitagent) )
if ( length(waitagent) > 1)
                    terminal(ag(agn).finalterminal).waitagent =
waitagent(2:length(waitagent));
else
                    terminal(ag(agn).finalterminal).waitagent = 0;
end
else
                terminal(ag(agn).finalterminal).waitagent = waitagent;
end
end

% checks necessary
        ag(agn).destinationreached = false;
        ag(agn).terminalreached = false;
        ag(agn).waitno = terminal(ag(agn).finalterminal).waitno;

        terminal(ag(agn).finalterminal).agent = 0;

terminal(ag(agn).finalterminal).waitagent(terminal(ag(agn).finaltermina
l).waitno) = agn;
        terminal(ag(agn).finalterminal).occupied = false;

        np = ag(agn).finalterminal;

        ag(agn).finalx(1) = terminal(ag(agn).finalterminal).x +
ag(agn).waitno*WIDTH;
        ag(agn).finaly(1) = terminal(ag(agn).finalterminal).y;
        ag(agn).path = [];
        ag(agn).watch_step = 1;
end
end

function simulationCommands(command,varargin)
%
% SIMULATIONCOMMANDS The function has commands to set up various
aspects
% of the simulation
%
% USAGE:
%   simulationCommands('Start')
%       To star the simulation
%
```

```
%   simulationCommands('Order',orderStruct)
%       To set up initial order. This is not used now, a function
exists.
%       Check setupOrderStack.
%
%   simulationCommands('Pause')
%       To pause the simulation
%
%   simulationCommands('StartFire')
%       Used to start fire
%
%   simulationCommands('StopFire')
%       Used to stop fire
%
%   simulationCommands('ObjectsInfo')
%       Prints information about the object type selected
%
%   simulationCommands('PostProcessing')
%       Post processing the data collected.
%
%
simulationCommands('Order',struct('subject','agent','sno',agentNumber,'
object',object,...
%
'ono',objectNumber,'priority',priority,'type',type,'coordinate',coordin
ate,'status',...
% 'toStart','userData',[]));
%
%
% subject       - agent
% sno           - 1-10
% object        - machine
% ono           - 1-5
% priority      - 1-5
% type          - inspect, troubleshoot, repair
% coordinate    - 0
% status        - 'toStart', 'started'
% userData      - can be any structure
%
% subject       - agent
% sno           - 1-10
% object        - watch
% ono           - 1-5
% priority      - 1-5
% type          - inspect
% coordinate    - 0
% status        - 'toStart', 'started'
% userData      - can be any structure
%
% subject       - agent
% sno           - 1-10
% object        - firefighting
% ono           - id
% priority      - 5
% type          - emergency
```

```
% coordinate    - 0
% status        - 'toStart', 'started'
% userData      - can be any structure
%
% subject       - agent
% sno           - 1-10
% object        - getfireequip
% ono           - id
% priority      - 5
% type          - emergency
% coordinate    - 0
% status        - 'toStart', 'started'
% userData      - can be any structure
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 13 October 2006
% Version History:
%
% Last Modified: 5 January 2007
%

global STARTSIMULATION;
global EXITPRESSED;

global time;
global simulationTime;

global ag;
global machine;
global watchLoc;
global terminal;
global server;

global emergencyScenarios;

global estcost;

%% START

if strcmpi(command,'Start')
% --- Start Simulation
    mainSimGUIHandle = findall(0, 'tag', 'mainGUI');
    mainSimGUIHandleList = guidata(mainSimGUIHandle);
    set(mainSimGUIHandleList.start,'enable','off');

for agn =1:length(ag)
        premovement(agn,ag(agn).finalterminal,0);
end

    machineCntrHandle = findall(0,'tag','machineCntr');
    machineCntrHandleList = guidata(machineCntrHandle);
```

```
    time = clock;
    simulationTime = 0;

    displayString('Reseting Simulation time to 0');

if( isempty(STARTSIMULATION) || STARTSIMULATION == false )
        numtimes = 1;

        workflowTimer = timer;
        set(workflowTimer,'ExecutionMode','fixedrate', 'Period',
250,'tag','workflowTimer');

set(workflowTimer,'TimerFcn',@initialOrder,'TasksToExecute',numtimes);
        start(workflowTimer);

% % start sensor data from machine 5
% set(machineCntrHandleList.Machine5,'Value',1);
% machinecntr('Machine5_Callback',machineCntrHandleList.Machine5, 1,
machineCntrHandleList);

        movement();
if( ~EXITPRESSED )
while( get(workflowTimer,'TasksExecuted') ~= numtimes )
                movement();
end
            movement();
end

        plottingNrecording(simulationTime);
        displayString('Simulation Done');

        estcost = simulationTime;

        displayString(['Simulation Cost is ', num2str(estcost),'
seconds']);

        delete(timerfindall);
end
% ---

%% ORDER

elseif strcmpi(command,'Order')
% --- order
if( nargin ~= 2 )
        error('Unknown Command');
end

    data = varargin{1};
if strcmpi(data.subject,'agent')
        agn = data.sno;
if agn == 0
```

```
            error('Agent number should be non zero')
end

if strcmpi(data.object,'machine')
            mcn = data.ono;
if mcn == 0
                error('Machine number should non zero')
end

if length(agn) > 1
                setupOrderStack('coordinate',data);
else
                setupOrderStack('machine',data);
end

elseif strcmpi(data.object,'watch')
            watchLocation = data.ono;
if watchLocation == 0
                error('WatchLocation number should non zero')
end

if length(agn) > 1
                error('Number of agents should be one');
end

            setupOrderStack('watch',data);
end
end
% ---

%% START FIRE

elseif strcmpi(command,'StartFire')
% ---
% start Fire
    fireCntrHandle = findall(0,'tag','fireCntr');
    fireCntrHandleList = guidata(fireCntrHandle);
    allParam = false;
while ~allParam
        rosSmoke =
str2double(get(fireCntrHandleList.rosSmoke,'String'));
        rosFire = str2double(get(fireCntrHandleList.rosFire,'String'));
        location = fireCntrHandleList.UserData{1};
if ~isempty(rosSmoke) && ~isempty(rosFire) && ~isempty(location)
            allParam = true;
end
end
    listStr = get(fireCntrHandleList.listoffires,'String');

% data for emergency scenario
    emergencyData =
insertEmergencyScenario(struct('type','fire','x',location(1),'y',locati
on(2),...
```

```
'rosFire',rosFire,'rosSmoke',rosSmoke,'priority',6,'agentsSent',[],'age
nts',[],'userData',[],'handleF',0,'handleS',0));

% print in the fire command window
    str{1} = sprintf('%9.7f                [%05.2f \t %05.2f]
Fire: %9.5f      Smoke: %9.5f',...
        emergencyData.id, location(1),location(2), rosFire, rosSmoke);

    newStr = [listStr;str];
    set(fireCntrHandleList.listoffires,'String',newStr,'listboxtop',
max(1, length(newStr)-1), 'value', length(newStr));

% print in the status window
    dispStr = sprintf('Starting Fire at location [%05.2f \t %05.2f]
with Rate of Fire spread %5.2f, Rate of Smoke spread %5.2f and
ID %d',...
        location(1),location(2), rosFire, rosSmoke, emergencyData.id);

    displayString(dispStr);

%% STOP FIRE

elseif strcmpi(command,'StopFire')
% --- stop fire
if isempty(varargin)
        fireCntrHandle = findall(0,'tag','fireCntr');
        fireCntrHandleList = guidata(fireCntrHandle);
        listStr = get(fireCntrHandleList.listoffires,'String');
        listID = get(fireCntrHandleList.listoffires,'Value');

        [fireID,locx,locy,rosF,rosS] =
strread(listStr{listID},'%f %*c%f%f%*c %*s %f %*s %f');

        r = [];
if ~isempty(emergencyScenarios)
            r = find( abs([emergencyScenarios.id] - single(fireID)) <
1e-7);
end

if ~isempty(r)
            emergencyScenarios(r).userData.radiusF = 0;
            emergencyScenarios(r).userData.radiusS = 0;
            emergencyScenarios(r).rosFire = 0;
            emergencyScenarios(r).rosSmoke = 0;

else
        fireCntrHandle = findall(0,'tag','fireCntr');
        fireCntrHandleList = guidata(fireCntrHandle);
        listStr = get(fireCntrHandleList.listoffires,'String');
for i=1:length(listStr)
            [fireID,locx,locy,rosF,rosS] =
strread(listStr{i},'%f %*c%f%f%*c %*s %f %*s %f');
if abs(fireID-varargin{1}) < 1e-7
```

```
                listID = i;
end
end

        [fireID,locx,locy,rosF,rosS] =
strread(listStr{listID},'%f %*c%f%f%*c %*s %f %*s %f');
        removeEmergencyScenario(fireID);

if listID == 1
            newStr = listStr(2:end);
elseif listID == length(listStr)
            newStr = listStr(1:end-1);
else
            newStr = [listStr(1:listID-1);listStr(listID+1:end)];
end

        set(fireCntrHandleList.listoffires,'String',newStr,'listboxtop',
max(1, length(newStr)-1), 'value', length(newStr));
        dispStr = sprintf('Stopping fire with ID %d',fireID);
        displayString(dispStr);
end
% ---

%% OBJECTS INFORMATION

elseif strcmpi(command,'ObjectsInfo')
% ---
    obj = varargin{1};
if ~isempty(obj)
        str{1} = sprintf('Selected Object is %s',obj);
end
% ---

%% PAUSE

elseif strcmpi(command,'Pause')
% --- Pause
    pause
% ---

%% EXIT

elseif strcmpi(command,'Exit')
% --- Exit
% xitFunction();
    displayString('Exit Pressed stoping simulation.');
    EXITPRESSED = true;
% ---

%% POST PROCESSING

elseif strcmpi(command,'PostProcessing')
```

```
% --- post processing
    postProcessing;
% ---

%% ELSE

else
% ---
    disp('Command not found');
% ---
end

function movement()
%
% MOVEMENT This function is responsible for the main simulation loop.
It
% shouses all other functions and checks.
%
% USAGE:
%   movement()
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 17 July 2006
% Version History:
%
% Last Modified: 5 January 2007
%

global ag;
global terminal;
global machine;
global server;
global pValueIdx;
global parameterValues;

global time;
global simulationTime;
global deltaTime;

global cost;
global OPTION;
global EXITPRESSED
global isEmpty;
global REASSIGN;
global WIRELESSRANGE;

global BOUNDARY;
global WIDTH;
global tsze;

global agCurrentTaskCData;
global time_reduction;
```

```
persistent mainSimGUIHandle;
persistent mainSimGUIHandleList;
persistent machineCntrHandleList;
persistent workflowGUIHandleList;

persistent simTimeStringHandle;
persistent wallTimeStringHandle;
global numDays;

if( isempty(machineCntrHandleList) )
    machineCntrHandle = findall(0,'tag','machineCntr');
    machineCntrHandleList = guidata(machineCntrHandle);
end

if( isempty(mainSimGUIHandle) )
    mainSimGUIHandle = findall(0, 'tag', 'mainGUI');
    mainSimGUIHandleList = guidata(mainSimGUIHandle);
    simTimeStringHandle = findobj(mainSimGUIHandle, 'tag', 'simTime');
    wallTimeStringHandle = findobj(mainSimGUIHandle, 'tag', 'wallTime');
end

if( isempty(machineCntrHandleList) )
    workflowGUIHandle = findall(0,'tag','workflowGUI');
    workflowGUIHandleList = guidata(workflowGUIHandle);
end


simulationTime = simulationTime + deltaTime;
curtime = simulationTime;
cost = cost + 1;

% to check if the workstack is empty
if( isempty(isEmpty) )
    isEmpty = false;
end

% Data recording
for agn =1:length(ag)
if( length(ag(agn).percentTaskComp) > 1 )
        ag(agn).percentTaskComp(length(ag(agn).percentTaskComp) + 1) ...
            = ag(agn).percentTaskComp(length(ag(agn).percentTaskComp));
        ag(agn).ptcTime(length(ag(agn).ptcTime) + 1) = curtime;
else
        ag(agn).percentTaskComp(length(ag(agn).percentTaskComp) + 1) =
0;
        ag(agn).ptcTime(length(ag(agn).ptcTime) + 1) = curtime;
end
end

% Draw now
drawnow
```

```
%==================main loop
while ( OPTION ~= 0 || ~isEmpty )

if parameterValues(pValueIdx,27) == 7
        ag(8).workstack = [];
        ag(9).workstack = [];
        ag(10).workstack = [];
end

    simulationTime = simulationTime + deltaTime;
    curtime = simulationTime;

    str = sprintf('%08.3f',simulationTime/time_reduction);
    set(simTimeStringHandle,'string',str);
    str = sprintf('%08.3f',etime(clock,time));
    set(wallTimeStringHandle,'string',str);

if( EXITPRESSED )
return;
end

if( OPTION == 0 && isEmpty )
return;
end

%% add functions here
    SA_tasks;
    utilization;

if abs(simulationTime - 24*60*60*time_reduction*numDays)< .1
        simulationCommands('Exit');
end

    plottingNrecording(curtime);
    networkSim();
    powerSimPOE();
    emergencySim();
if REASSIGN
        workflowSim();
end

% loop for agents
    count = 0;
for agn = 1:length(ag)

        makeChecks(agn);
%         exchangeInfo(agn);
        updateMemory(0,[]);
        makeChecks(agn);

        ag(agn).busy = false;
```

```
        ag(agn).forcex = 0;
        ag(agn).forcey = 0;

% calculate forces on each agent
        [dx,dy,dfx,dfy] = getforce(agn);
        ag(agn).forcex = dfx + ag(agn).forcex;
        ag(agn).forcey = dfy + ag(agn).forcey;

% move agent depending on forces acting
        [velx,vely,dx,dy] = incrementalmove(agn,dx,dy);

% update agent velocity and coordinates
if( dx ~= 0 || dy ~= 0 )
            ag(agn).velx = velx;
            ag(agn).vely = vely;
            ag(agn).x = ag(agn).x + dx;
            ag(agn).y = ag(agn).y + dy;

            ax1 = ag(agn).x+2*tsze/3;
            ax2 = ag(agn).x-2*tsze/3;
            ay1 = ag(agn).y+2*tsze/3;
            ay2 = ag(agn).y-2*tsze/3;
            set(ag(agn).handle,'XData',[ax1,ax2,ax2,ax1]);
            set(ag(agn).handle,'YData',[ay1,ay1,ay2,ay2]);
            set(ag(agn).numberHandle,'Position',[ag(agn).x-2*tsze
ag(agn).y]);
end

if( isempty(ag(agn).workstack) )
            1;
else
            count = count + 1;
end
end
if strcmpi(get(mainSimGUIHandleList.mainAxis,'Visible'),'on')
        drawnow
end

if( count == 0 )
        isEmpty = true;
else
        isEmpty = false;
end

if parameterValues(pValueIdx,27) == 7
        ag(8).workstack = [];
        ag(9).workstack = [];
        ag(10).workstack = [];
end

end

function SA_tasks
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SA_tasks()
%
% Assign  all agents duties
%
% Created by : Shawn McKay
% Modified by: Robin Kusmanto
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global pValueIdx;
global parameterValues;
global repairnum;
global time_reduction;
global simulationTime;
global failureTimes;
global numberOfRepairs;
global watchnumber;
global PMnumber;
global agnr;
global agnw;
global agnp;
global watchnum;
global PMnum;
global watchtimes;
global PMtimes;
global priorityr;
global priorityw;
global priorityp;
global dr;
global dw;
global dwq;
global dp;
global agntestw;
global repairstep;
global PMtag;
global ag;
global repairs;
global PM;
global machinedown;
global PMdownTotal;
global PMdownend;
global PMdownstart;
global availability;
global time_step_availability;
global availabilityData;
global availabilityTime;
global machine;
global simTimerepair;
global Totalmachinedown;

global simulationTime;

curtime = simulationTime;

persistent machinecntrHandleList;
```

```
persistent workflowGUIHandleList;

if isempty(machinecntrHandleList)
    machinecntrHandle = findall(0,'tag','machineCntr');
    machinecntrHandleList = guidata(machinecntrHandle);

    workflowGUIHandle = findall(0,'tag','workflowGUI');
    workflowGUIHandleList = guidata(workflowGUIHandle);
end

machineinspecttime = 0.5;  %  time to inspect a machine in hours
intelfactor = .5;   % inpection reduction time due to sensor data

for i = 1:numberOfRepairs  %repairs
if ( abs(simulationTime -failureTimes(i)) <= .2);
if abs(failureTimes(i) - dr)> .9
            dr = failureTimes(i);
            repair = repairnum(i)-1;

if parameterValues(pValueIdx,10) == 1
if parameterValues(pValueIdx,31) == 30  % takes the 30% of the failure
times and shifts them forward by the correct ammount
if i == 2 || i == 8 || i == 9 || i == 10 || i == 13 || i == 14 || i ==
15 || i == 16 || i == 19
                        machine(repair).faultfound = 1;

machine(repair).maintenanceData.inspectionTimeWired =
parameterValues(pValueIdx,30)*machineinspecttime*intelfactor*3600*time_
reduction;

end
else
if i == 2 || i == 8 || i == 9 || i == 10 || i == 13 || i == 14 || i ==
15 || i == 16 || i == 19 || i == 3 || i == 7 || i ==  11  || i == 12 ||
i == 15 || i == 23 || i == 24 || i == 26
                        machine(repair).faultfound = 1;

machine(repair).maintenanceData.inspectionTimeWired =
parameterValues(pValueIdx,30)*machineinspecttime*intelfactor*3600*time_
reduction;
end
end
else
                machine(repair).maintenanceData.inspectionTimeWired =
parameterValues(pValueIdx,30)*machineinspecttime*3600*time_reduction;
end

if repair == 1
if machine(1).fault == 1
                    failureTimes(i) = failureTimes(i) + 1;
else
                    set(machinecntrHandleList.faultmac1,'value',1);
```

```
machinecntr('faultmac1_Callback',machinecntrHandleList.faultmac1,1,mach
inecntrHandleList);
                    displayString(['****Machine',num2str(repair),'
down']);
end
elseif repair == 2
if machine(2).fault == 1
                    failureTimes(i) = failureTimes(i) + 1;
else
                    set(machinecntrHandleList.faultmac2,'value',1);

machinecntr('faultmac2_Callback',machinecntrHandleList.faultmac2,1,mach
inecntrHandleList);
                    displayString(['****Machine',num2str(repair),'
down']);
end
elseif repair == 3
if machine(3).fault == 1
                    failureTimes(i) = failureTimes(i) + 1;
else
                    set(machinecntrHandleList.faultmac3,'value',1);

machinecntr('faultmac3_Callback',machinecntrHandleList.faultmac3,1,mach
inecntrHandleList);
                    displayString(['****Machine',num2str(repair),'
down']);
end
elseif repair == 4
if machine(4).fault == 1
                    failureTimes(i) = failureTimes(i) + 1;
else
                    set(machinecntrHandleList.faultmac4,'value',1);

machinecntr('faultmac4_Callback',machinecntrHandleList.faultmac4,1,mach
inecntrHandleList);
                    displayString(['****Machine',num2str(repair),'
down']);
end
elseif repair == 5
if machine(1).fault == 1
                    failureTimes(i) = failureTimes(i) + 1;
else
                    set(machinecntrHandleList.faultmac5,'value',1);

machinecntr('faultmac5_Callback',machinecntrHandleList.faultmac5,1,mach
inecntrHandleList);
                    displayString(['****Machine',num2str(repair),'
down']);
end
else
                error('machine number is incorrect');
end
end
end
```

```
end

for i = 1:numberOfRepairs
if
abs(failureTimes(i)+parameterValues(pValueIdx,35)*3600*time_reduction-
simulationTime)<=.2
        repair = repairnum(i)-1;
        machine(repair).faultfound = 0;
end
end

count = 0;
for i = 1:5
if machine(i).fault == 1
if machine(i).faultfound ~=1
            count = count + 1;
end
end
end
deltaDowntime = count*(simulationTime - simTimerepair);
Totalmachinedown = Totalmachinedown + deltaDowntime ;
Totalmachinedown_time(time_step_availability,1) = Totalmachinedown;


for i = 1:PMnumber*3  %  PMs
if abs(PMtimes(i) - dp)>1
if ( abs(simulationTime -PMtimes(i)) <=.2);

            PMvalue = PMnum(i)-1;
            agnp(i) = agnp(i);
            displayString(['Ag',num2str(agnp(i)),' has been assigned to
PM M',num2str(PMvalue)]);
            agnp(i) = agnp(i)+1;
            set(workflowGUIHandleList.agNumber,'value',agnp(i));

workflowGUI('agNumber_Callback',workflowGUIHandleList.agNumber,1,workfl
owGUIHandleList);

            set(workflowGUIHandleList.workflowType,'value',1);

workflowGUI('workflowType_Callback',workflowGUIHandleList.workflowType,
1,workflowGUIHandleList);

            set(workflowGUIHandleList.priority,'value',priorityp);

workflowGUI('priority_Callback',workflowGUIHandleList.priority,1,workfl
owGUIHandleList);

            set(workflowGUIHandleList.ono,'value',PMvalue);

workflowGUI('ono_Callback',workflowGUIHandleList.ono,1,workflowGUIHandl
eList);
```

```
            set(workflowGUIHandleList.okButton,'value',1);

workflowGUI('okButton_Callback',workflowGUIHandleList.okButton,1,workfl
owGUIHandleList);

            dp = PMtimes(i);

            PM(i,1) = simulationTime;
            PM(i,2) = agnp(i);
            PM(i,3) = priorityp;
            PM(i,4) = PMnum(i)- 1;
            PM(i,5) = 2;

end
end
end
[length_PM,width_PM] = size(PM);

for i = 1:length_PM
if ag(PM(i,2)).machine == PM(i,4)
if ag(PM(i,2)).machinereached == 1
if ag(PM(i,2)).inspectiondone == 1
if ag(PM(i,2)).timemcreached > 0 && PMtag(i) ~= 1
                    PMdownstart(i) = simulationTime;
                    PMtag(i) = 1;
                    PM(i,5) = 3;

end
end
end
end
end
for i = 1:length_PM
if ag(PM(i,2)).machine == PM(i,4)
if ag(PM(i,2)).machinereached == 1
if ag(PM(i,2)).inspectiondone == 1
if ag(PM(i,2)).timemcreached > 0
                    PMdownend(i) = simulationTime;
                    PMdownTotal(i) = PMdownend(i) - PMdownstart(i);
                    PM(i,5) = 4;
end
end
end
end
end

for i = 1:length_PM
if PM(i,5) == 4 && ag(PM(i,2)).machinereached == 0
        PM(i,5) = 5;
end
end

for i = 1:length_PM
```

```
for t = (i+1):length_PM
if PM(i,4) == PM(t,4)
if PM(i,5) == 3 || PM(i,5) == 4  % changed the latter from repairs to
PM
if PM(t,5) == 3 || PM(t,5) == 4
                    PMdownTotal(i) = 0;
end
end
end
end
end

PM_Total = sum(PMdownTotal);

availability = (10*simulationTime -(PM_Total +
Totalmachinedown))/(10*simulationTime);
availabilityData(time_step_availability,1) = availability;
availabilityTime(time_step_availability,1) = curtime;


reliability = [Totalmachinedown PM_Total availability];

for i = 1:watchnumber*3  %watch
if ( abs(simulationTime -watchtimes(i)) <= .2);
if abs(watchtimes(i) - dw)>1  ;

            watch = watchnum(i)-1;
            agnw(i) = agnw(i)+1 ;
            displayString(['Ag',num2str(agnw(i)-1),' has been assigned
to W',num2str(watch)]);
            set(workflowGUIHandleList.agNumber,'value',agnw(i));

workflowGUI('agNumber_Callback',workflowGUIHandleList.agNumber,1,workfl
owGUIHandleList);

            set(workflowGUIHandleList.workflowType,'value',2);

workflowGUI('workflowType_Callback',workflowGUIHandleList.workflowType,
1,workflowGUIHandleList);

            set(workflowGUIHandleList.priority,'value',priorityw);

workflowGUI('priority_Callback',workflowGUIHandleList.priority,1,workfl
owGUIHandleList);

            set(workflowGUIHandleList.ono,'value',watch);

workflowGUI('ono_Callback',workflowGUIHandleList.ono,1,workflowGUIHandl
eList);

            set(workflowGUIHandleList.okButton,'value',1);
```

```
workflowGUI('okButton_Callback',workflowGUIHandleList.okButton,1,workfl
owGUIHandleList);

            dw = watchtimes(i);

end
end
end

time_step_availability = time_step_availability +1;
simTimerepair = simulationTime;

function SA_tasks_values
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SA_tasks_values()
%
% Assign the number of duties and times
%
% Created by : Shawn McKay
% Modified by: Robin Kusmanto
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global pValueIdx;
global parameterValues;
global repairnum;
global time_reduction;
global simulationTime;
global failureTimes;
global numberOfRepairs;
global watchnumber;
global PMnumber;
global agnr;
global agnw;
global agnp;
global watchnum;
global PMnum;
global watchtimes;
global PMtimes;
global priorityr;
global priorityw;
global priorityp;

priorityr = 3;
priorityw = 3;
priorityp = 3;

if parameterValues(pValueIdx,29) == 18
    watchnumber = 12;
else
    watchnumber = 6;
end
```

```
PMnumber =  6;

repairnum_high = [2 3 1 1 3 1 5 4 3 1 1 4 5 2 1] +1;
repairnum_low = [2 3 1 1 5 3 1 4 2 ]+1;

watchnum_high = [5 1 1 2 2 2 5 4 5 3 3 4 ] +1;
watchnum_low = [5 1 2 2 5 3 ] + 1;

PMnum_high = [5 4 2 4 3 5]+1;
PMnum_low =  [5 4 2 4 3 5]+1;
a = [ 1 2 3 4].*.3;

watchtimes_high = [.1 .2 4 8 8.1 8.2 12 16 16.1 16.2 20
24.9].*(time_reduction*3600);
watchtimes_low = [.1 .2 4 8 8.1 12 16 16.1 20].*(time_reduction*3600);

PMtimes_high = [4 5 8 8.1 9 16 16.1 17 23].*(time_reduction*3600);
PMtimes_low = [4 5 8 9 16 17].*(time_reduction*3600);


repairnum = repairnum_high;
watchnum = watchnum_high;
PMnum = PMnum_high;
watchtimes = watchtimes_high;
PMtimes = PMtimes_low;

watchnum = cat(2,watchnum,watchnum,watchnum);
repairnum = cat(2,repairnum,repairnum,repairnum);
PMnum = cat(2,PMnum,PMnum,PMnum);
watchtimes = cat(2,watchtimes,watchtimes
+24*time_reduction*3600,watchtimes+48*time_reduction*3600);
PMtimes = cat(2,PMtimes,PMtimes + 24*time_reduction*3600,PMtimes +
48*time_reduction*3600);

if parameterValues(pValueIdx,27) == 10       %  For 10 crew in the zone
scenario 3
            agnr(1) = 1;
            agnr(2) = 2;
            agnr(3) = 3;
            agnr(4) = 3;
            agnr(5) = 4;
            agnr(6) = 4;
            agnr(7) = 5;
            agnr(8) = 6;
            agnr(9) = 6;
            agnr(10) = 7;
            agnr(11) = 8;
            agnr(12) = 8;
            agnr(13) = 9;
            agnr(14) = 10;
            agnr(15) = 10;

            agnp(1) = 1;
```

```
            agnp(2) = 1;
            agnp(3) = 2;
            agnp(4) = 3;
            agnp(5) = 7;
            agnp(6) = 8;

            agnw(1) = 2;
            agnw(2) = 3;
            agnw(3) = 4;
            agnw(4) = 5;
            agnw(5) = 5;
            agnw(6) = 6;
            agnw(7) = 7;
            agnw(8) = 8;
            agnw(9) = 9;
            agnw(10) = 9;
            agnw(11) = 10;
            agnw(12) = 10;
else%  For 7 crew in the zone scenario 3
            agnr(1) = 1;
            agnr(2) = 1;
            agnr(3) = 2;
            agnr(4) = 2;
            agnr(5) = 3;
            agnr(6) = 4;
            agnr(7) = 4;
            agnr(8) = 4;
            agnr(9) = 4;
            agnr(10) = 5;
            agnr(11) = 5;
            agnr(12) = 6;
            agnr(13) = 6;
            agnr(14) = 7;
            agnr(15) = 7;

            agnp(1) = 1;
            agnp(2) = 1;
            agnp(3) = 2;
            agnp(4) = 5;
            agnp(5) = 6;
            agnp(6) = 7;

            agnw(1) = 1;
            agnw(2) = 1;
            agnw(3) = 1;
            agnw(4) = 2;
            agnw(5) = 2;
            agnw(6) = 3;
            agnw(7) = 5;
            agnw(8) = 5;
            agnw(9) = 6;
            agnw(10) = 6;
            agnw(11) = 7;
            agnw(12) = 7;
```

```
end

agnr = cat(2,agnr,agnr,agnr);
agnw = cat(2,agnw,agnw,agnw);
agnp = cat(2,agnp,agnp,agnp);

function utilization()
%
% UTILIZATION This function calculates the utilization of crew members.
% This ustilization will be used to reschedule the workflow to maintain
the
% utilization within a particular band.
%
% USAGE:
%   utilization()
%
%       The function plots a figure showing utilization as a % plotted
with
%       time.
%
% -----------
% Created by: Shawn McKay
% Created on: 26 Jan 2007
% Modification History
% Old Date      New Date            Modified by         Changes
% 26 Jan 2007   26 Jan 2007         Vishal Mahulkar
%
% Version History:
%
% Last Modified: 26 January 2007
%
global ag;

global simulationTime;
global deltaTime;

global deltat;
global ag_utilization;

global watchtimes;
global PMtimes;
global failureTimes;

global aveTask;
global agTotalUtiData;
global agTotalUtiTime;
global time_reduction;

global parameterValues;
global pValueIdx;

persistent time_step_utilization;
```

```
if isempty(time_step_utilization)
    time_step_utilization = 1;
end

% calculate the busy times for an agent
for i = 1:length(ag)
    [a,b] = size(ag(i).percentTaskComp);
if ag(i).orderstatus == 0
        deltat(i) = deltat(i) + deltaTime;
        ag_utilization(i,time_step_utilization) =
deltat(i)/simulationTime;

% My modifications
        ag(i).utilization(time_step_utilization) =
deltat(i)/simulationTime;
        ag(i).utiTime(time_step_utilization) = simulationTime;
%
else
        ag_utilization(i,time_step_utilization) =
deltat(i)/simulationTime;

% My modifications
        ag(i).utilization(time_step_utilization) =
deltat(i)/simulationTime;
        ag(i).utiTime(time_step_utilization) = simulationTime;
%
end
end

agTotalUtiData(1,time_step_utilization) =
sum(ag_utilization(:,time_step_utilization))/parameterValues(pValueIdx,
27);

agTotalUtiTime(1,time_step_utilization) = simulationTime;

if watchtimes(1,length(watchtimes)) > simulationTime
    w = find(watchtimes>simulationTime-1,1) - 1;
else
    w = length(watchtimes);
end
if failureTimes(1,length(failureTimes)) > simulationTime
    r = find(failureTimes>simulationTime-1,1) - 1;
else
    r = length(failureTimes);
end
if PMtimes(1,length(PMtimes)) > simulationTime
    p = find(PMtimes>simulationTime-1,1) - 1;
else
    p = length(PMtimes);
end

if w + r + p < 1
    aveTask(1,time_step_utilization) = 0;
```

```
    aveTask(2,time_step_utilization) = simulationTime*time_reduction;
else
    aveTask(1,time_step_utilization) = sum(deltat)/(w + p + r);
    aveTask(2,time_step_utilization) = simulationTime*time_reduction;
end
time_step_utilization = time_step_utilization +1;

function plottingNrecording(curtime)
%
% PLOTTINGNRECORDING This function plots the data based on the
simulation
% time. It plots the workflow completion times of individual agents as
well
% as the total workflow completion. Plots of bandwidth at each terminal
are
% also plotted
%
% USAGE:
%   plottingNrecording(curtime)
%
%       curtime         currtent simulation time
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 12 October 2006
% Modification History:
% Old Date      New Date            Modified by         Changes
% 22 Feb 2007   3 Mar 2007          Vishal Mahulkar     Added % of
total
%                                                       workflow
%
% Version History:
%
% Last Modified: 3 Mar 2007
%

global ag;

global agCurrentTaskCData;
global agCurrentTaskCTime;
global agTotalTaskCData;
global agTotalTaskCTime;
global agTotalUtiTime;
global agTotalUtiData;

global totalPlotHandle;
global OPTION;
global WORKSTACK;

global networkGraph;
global clusteringCoeff;
global degreeDistribution;
persistent mainSimGUIHandleList;
persistent childEffInd;
```

```
persistent childEffTot;
persistent childUtiInd;
persistent childUtiTot;

global time_reduction;

% get handles to the graphics
if isempty(mainSimGUIHandleList)
    mainSimGUIHandle = findall(0,'tag','mainGUI');
    mainSimGUIHandleList = guidata(mainSimGUIHandle);
end

if isempty(childEffInd)
    childEffInd =
findobj(get(mainSimGUIHandleList.axes1,'Children'),'Tag','Current');
end

if isempty(childEffTot)
    childEffTot =
findobj(get(mainSimGUIHandleList.axes2,'Children'),'Tag','Current');
end

if isempty(childUtiInd)
    childUtiInd =
findobj(get(mainSimGUIHandleList.indUtilization,'Children'),'Tag','Curr
ent');
end

if isempty(childUtiTot)
    childUtiTot =
findobj(get(mainSimGUIHandleList.totUtilization,'Children'),'Tag','Curr
ent');
end

% current selection in the graphics
currAgentSel = get(mainSimGUIHandleList.popupmenu1, 'Value');

% Calculating % of current task completed
workSum = 0;
totalWork = 0;
for i =1:length(ag)
if( ag(i).hasorders )
        workSum = workSum + 1;
        indexNums = find(ag(i).ptcTime<=curtime,1,'last');
try
            totalWork = totalWork + ag(i).percentTaskComp(indexNums);
catch
            disp(lasterr);
            disp(indexNums);
            disp(i);
            disp(curtime);
            OPTION = 0;
return;
```

```
end
end
end

% record the % of total task completed
if( workSum ~= 0)
    agCurrentTaskCData(length(agCurrentTaskCData) + 1) =
totalWork/workSum;
else
    agCurrentTaskCData(length(agCurrentTaskCData) + 1) =
agCurrentTaskCData(length(agCurrentTaskCData));
end
agCurrentTaskCTime(length(agCurrentTaskCTime) + 1) = curtime;

% Calculate the amount of total.
workLen = 0;
for i=1:length(WORKSTACK)
    workLen = workLen + length(WORKSTACK(i).sno);
end
agTotalTaskCData(end+1) = 1-length([ag.workstack])/workLen;
agTotalTaskCTime(end+1) = curtime;

if strcmpi(get(mainSimGUIHandleList.effPanel,'Visible'),'on')


set(childEffTot,'XData',agTotalTaskCTime/time_reduction,'YData',agTotal
TaskCData);
    maxX =
max(max(agTotalTaskCTime/time_reduction),max(get(mainSimGUIHandleList.a
xes2,'XLim')));
    set(mainSimGUIHandleList.axes2,'XLim',[0 maxX]);

if length(ag(currAgentSel).ptcTime) > 0

set(childEffInd,'Xdata',ag(currAgentSel).ptcTime/time_reduction,'Ydata'
,ag(currAgentSel).percentTaskComp);

set(mainSimGUIHandleList.axes1,'XLim',get(mainSimGUIHandleList.axes2,'X
Lim'),'YLim',[0 100])
end

elseif strcmpi(get(mainSimGUIHandleList.utiPanel,'Visible'),'on')


set(childUtiTot,'XData',agTotalUtiTime/time_reduction,'YData',agTotalUt
iData);
    maxX =
max(max(agTotalUtiTime/time_reduction),max(get(mainSimGUIHandleList.tot
Utilization,'XLim')));
    set(mainSimGUIHandleList.totUtilization,'XLim',[0 maxX]);

if length(ag(currAgentSel).ptcTime) > 0
```

```
set(childUtiInd,'Xdata',ag(currAgentSel).utiTime/time_reduction,'Ydata'
,ag(currAgentSel).utilization);

set(mainSimGUIHandleList.indUtilization,'XLim',get(mainSimGUIHandleList
.totUtilization,'XLim'),'YLim',[0 1])
end

end

function powerSimPOE(varargin)
%
% POWERSIMPOE This function manages power trimming if the available
power
% drops below required. The network nodes are assumed to be powered
over
% ethernet
%
% USAGE:
%   powerSimPOE()
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 17 July 2006
% Version History:
%
% Last Modified: 21 December 2006
%

global terminal;
global server;

global deckAct;
global powerGeneratedPercent;
global powerGenerated;
global powerUtilized;
global POWERSTATUSCHANGE;

powerCntrHandle = findall(0,'tag','powerCntr');
apCntrHandle  = findall(0,'tag','apCntr');
servCntrHandle = findall(0,'tag','servCntr');
powerCntrHandleList = guidata(powerCntrHandle);
apCntrHandleList = guidata(apCntrHandle);
servCntrHandleList = guidata(servCntrHandle);

if( POWERSTATUSCHANGE )
if( powerGenerated ~= 0)
        deckAct = calcDeckStatus(powerGenerated,deckStructure());
        powerUtilized = deckAct.power*100/powerGenerated;
else
        powerUtilized = 0;
end
```

```
if( powerUtilized > 100)
        powerUtilized = 100;
end

    set(powerCntrHandleList.pUtilized,'Value',powerUtilized);

set(powerCntrHandleList.pUtilizedtxt,'String',[num2str(floor(powerUtili
zed)),'%']);

    set(powerCntrHandleList.pAvailable,'Value',powerGeneratedPercent);

set(powerCntrHandleList.pAvailabletxt,'String',[num2str(powerGeneratedP
ercent),'%']);

    rmNum = findRoomsWServers();
for i=1:length(rmNum)
if( ~deckAct.room(rmNum(i)).state )
for servNum = 1:length(deckAct.room(rmNum(i)).servers)
                str1 =
strcat('serv',num2str(deckAct.room(rmNum(i)).servers(servNum)),'Operati
onal');
                set(servCntrHandleList.(str1),'Value',0);

servcntr(strcat(str1,'_Callback'),servCntrHandleList.(str1),1,servCntrH
andleList);
end
else
for servNum = 1:length(deckAct.room(rmNum(i)).servers)
                str1 =
strcat('serv',num2str(deckAct.room(rmNum(i)).servers(servNum)),'Operati
onal');
                set(servCntrHandleList.(str1),'Value',1);

servcntr(strcat(str1,'_Callback'),servCntrHandleList.(str1),1,servCntrH
andleList);
end
end
end
end
POWERSTATUSCHANGE = false;

guidata(powerCntrHandle,powerCntrHandleList);
guidata(apCntrHandle,apCntrHandleList);

function emergencySim()
%
% EMERGENCYSIM This function is used to update emergency situation
status
%
% USAGE:
%   emergencySim()
%
% -----------
% Created by: Vishal Mahulkar
```

```
% Created on: 13 October 2006
% Version History:
%
% Last Modified: 5 January 2007
%

global ag;
global terminal;
global server;

global emergencyScenarios;
global deltaTime;

global recedenceRateS;
global recedenceRateF;

global CONVERSIONFACT;

persistent serverCntrHandleList;
persistent apCntrHandleList;

if isempty(serverCntrHandleList)
    serverCntrHandle = findall(0,'tag','servCntr');
    serverCntrHandleList = guidata(serverCntrHandle);
end

if isempty(apCntrHandleList)
    apCntrHandle = findall(0,'tag','apCntr');
    apCntrHandleList = guidata(apCntrHandle);
end

i = 1;
while i <= length(emergencyScenarios)
if strcmpi(emergencyScenarios(i).type,'fire')
        rateF = emergencyScenarios(i).rosFire -
length(emergencyScenarios(i).agents)*recedenceRateF;
        emergencyScenarios(i).userData.radiusF =
emergencyScenarios(i).userData.radiusF ...
            + rateF*deltaTime*CONVERSIONFACT;
        rateS = emergencyScenarios(i).rosSmoke -
length(emergencyScenarios(i).agents)*recedenceRateS;
        emergencyScenarios(i).userData.radiusS =
emergencyScenarios(i).userData.radiusS ...
            + rateS*deltaTime*CONVERSIONFACT;

if emergencyScenarios(i).userData.radiusF <= 0
            emergencyScenarios(i).userData.radiusF = 0;
            emergencyScenarios(i).rosFire = 0;
end
if emergencyScenarios(i).userData.radiusS <= 0
            emergencyScenarios(i).userData.radiusS = 0;
            emergencyScenarios(i).rosSmoke = 0;
end
```

```
        xycircledataF = plot_circle(emergencyScenarios(i).x,...
emergencyScenarios(i).y,emergencyScenarios(i).userData.radiusF,100,'int
erval');
set(emergencyScenarios(i).handleF,'Xdata',xycircledataF(:,1),'YData',xy
circledataF(:,2))
        xycircledataS = plot_circle(emergencyScenarios(i).x,...
emergencyScenarios(i).y,emergencyScenarios(i).userData.radiusS,100,'int
erval');
set(emergencyScenarios(i).handleS,'Xdata',xycircledataS(:,1),'YData',xy
circledataS(:,2))

% send more agents if fire is still spreading
        rateFSim = emergencyScenarios(i).rosFire -
length(emergencyScenarios(i).agentsSent)*recedenceRateF;
        rateSSim = emergencyScenarios(i).rosSmoke -
length(emergencyScenarios(i).agentsSent)*recedenceRateS;

% check equipment in fire and turn it off
for servNum=1:length(server)
            str = strcat('serv',num2str(servNum),'Operational');
if distanceBetween(server(servNum),emergencyScenarios(i)) <...
                    emergencyScenarios(i).userData.radiusF &&...
                    get(serverCntrHandleList.(str),'Value')
                set(serverCntrHandleList.(str),'Value',0);

servcntr(strcat(str,'_Callback'),serverCntrHandleList.(str),1,serverCnt
rHandleList);
end
end
for tnum =1:length(terminal)
            str = strcat('MyButton',num2str(tnum));
if distanceBetween(terminal(tnum),emergencyScenarios(i)) <...
                    emergencyScenarios(i).userData.radiusF &&...
                    ~get(apCntrHandleList.(str),'Value');
                set(apCntrHandleList.(str),'Value',1);

apcntr(strcat(str,'_Callback'),apCntrHandleList.(str),1,apCntrHandleLis
t);
end
end

if rateFSim > 0 || rateSSim > 0
            agn =
findAgentforEmergency([emergencyScenarios(i).x,emergencyScenarios(i).y]
);
            object = 'getFireEquip';
            priority = 6;
            ono = emergencyScenarios(i).id;
            type = 'emergency';
```

```
            data =
insertWorkflow(struct('subject','agent','sno',agn,'object',object,'ono'
,...

ono,'priority',priority,'type',type,'coordinate',0,'status','toStart','
userData',[]));
            setupOrderStack(object,data);
end

if emergencyScenarios(i).rosSmoke == 0 && emergencyScenarios(i).rosFire
== 0
            agNums = emergencyScenarios(i).agentsSent;
for j=1:length(agNums)
                removeCompletedWorkflow(agNums(j));
if ~isempty(ag(agNums(j)).workstack)
                    setAgentOrder(agNums(j),ag(agNums(j)).workstack(1));
end
end
            simulationCommands('StopFire',emergencyScenarios(i).id);
% remove emergency scenario
else
            i = i+1;
end
end
end

function workflowSim()
%
% WORKFLOWSIM This function is used to reschedule workflow depending on
the
% utilization
%
% USAGE:
%   workflowSim()
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 22 Feb 2007
% Version History:
%
%
% Modification History:
% Earlier Date      Modified        Type
%
%
% Last Modified: 22 Feb 2007
%

global ag

for agn=1:length(ag)-1
if ~isempty(ag(agn).utilization)
if ag(agn).utilization(end) > .6
```

```
% keyboard
            lenVector = [];
            agnVector = [];
for ii=1:length(ag)-1
                workLen = 0;
                lenVector = [lenVector, length(ag(ii).workstack)];
                agnVector = [agnVector, ii];
end
            meanLen = ceil(mean(lenVector));

if length(ag(agn).workstack) > meanLen && meanLen > 0;
                workstack = ag(agn).workstack(meanLen+1:end);

% sort the agent numbers according to the length of
% workstack length
                [slenVector,idxslenVector] = sort(lenVector);

% counter
                agCounter = 1;

% decrease the current agents workstack size
                ag(agn).workstack = ag(agn).workstack(1:meanLen);

% loop over the excess workflow and reassign
for jj = 1:length(workstack)
if ~strcmpi(workstack(jj).type,'recreation') &&...
                            ~strcmpi(workstack(jj).type,'report') &&...
                            ~strcmpi(workstack(jj).type,'emergency')
%                         keyboard
                        workstackLen =
length(ag(agnVector(idxslenVector(agCounter))).workstack);

% find and replace the agent number in the workflow
                        r = find(workstack(jj).sno == agn);
                        workstack(jj).sno(r) =
agnVector(idxslenVector(agCounter));

                        displayString(['Assigning work from Agent
',num2str(agn),...
' to Agent ',num2str(agnVector(idxslenVector(agCounter)))]);
% reassign the workflow
if workstackLen == 0

ag(agnVector(idxslenVector(agCounter))).workstack = workstack(jj);
% call this if this is the first workflow

setAgentOrder(agnVector(idxslenVector(agCounter)),workstack(jj));
else

ag(agnVector(idxslenVector(agCounter))).workstack(workstackLen+1) =
workstack(jj);
end
                        agCounter = agCounter+1;
```

```
else
                        ag(agn).workstack(end+1) = workstack(jj);
end
end
else
                workflowType = [];
if ~isempty(ag(agn).workstack)
                    workflowType = {ag(agn).workstack.type};
end

if isempty(strmatch('recreation',workflowType,'exact'))
%                     keyboard
                    displayString(['Agent ',num2str(agn),' has high
utilization. Adding recreation to work list']);
                    object = 'recreation';
                    type = 'recreation';
                    priority = 5;
                    ono = mod(agn,3)+1;
                    data =
insertWorkflow(struct('subject','agent','sno',agn,'object',object,'ono'
,...

ono,'priority',priority,'type',type,'coordinate',0,'status','toStart','
userData',[]));
if length(ag(agn).workstack) > 1
                        ag(agn).workstack = [ag(agn).workstack(1), data,
ag(agn).workstack(2:end)];
elseif length(ag(agn).workstack) == 1
                        ag(agn).workstack = [ag(agn).workstack(1),
data];
else
                        ag(agn).workstack = data;
                        setAgentOrder(agn,ag(agn).workstack);
end
end
end
end
end
end

function makeChecks(agn)
%
% MAKECHECKS This function makes various checks during workflow to set
up new
% workflows
%
% USAGE:
% makeChecks(agn)
%
% agn       agent number
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 13 October 2006
```

```
% Version History:
%
% Last Modified: 5 Januayry 2007
%

global ag;

global simulationTime;

curtime = simulationTime;

if ~isempty(ag(agn).workstack)
    data = ag(agn).workstack(1);

    subject = data.subject;
    sno = data.sno;
    object = data.object;
    ono = data.ono;
    priority = data.priority;
    type = data.type;

if strcmpi(data.subject,'agent')
switch data.object
case'machine'
                machineChecks(agn);
case'watch'
                watchlocationChecks(agn);
case'terminal'
                apChecks(agn);
case'firefighting'
                firefightingChecks(agn);
case'getFireEquip'
                getEquipChecks(agn);
case'recreation'
                recreationChecks(agn);
case'emergencyProtocol'
                ;
end
end
else
    apChecks(agn);
end

function apChecks(agn)
%
% APCHECKS This function makes checks on agents reporting data
%
% USAGE:
%   apChecks(agn)
%
%       agn           agent number
%
% -----------
```

```
% Created by: Vishal Mahulkar
% Created on: 13 October 2006
% Modified by:  Robin Kusmanto
%
% Last Modified: 21 May 2009
%

global ag;
global terminal;

global WIRELESSRANGE;
global BOUNDARY;
global WIDTH;

global simulationTime;

curtime = simulationTime;

if ag(agn).finalterminal == 0
    ag(agn).finalterminal = getClosestTerminalforAgent(agn);
    premovement(agn,ag(agn).finalterminal,0);
end

if ag(agn).pda && terminal(ag(agn).finalterminal).wireless
    distApp = WIRELESSRANGE;
else
    distApp = BOUNDARY;
end

if( ag(agn).terminalreached &&  (terminal(ag(agn).finalterminal).waitno >
0) &&...
        ag(terminal(ag(agn).finalterminal).waitagent(1)).waittime > 1
&&...
        ( ( ( (curtime - ag(agn).timetermreached) >
(ag(agn).datainputtime + ag(agn).inclagtimedata) ) ...
        || ag(agn).orderstatus ) &&...

distanceBetween(ag(terminal(ag(agn).finalterminal).waitagent(1)),termin
al(ag(agn).finalterminal)) < 1.2*WIDTH ))
    moveover(agn);
end

% waiting for a long time
if( ag(agn).waittime > 0)
if( (curtime - ag(agn).waittime) > 5 )
        newTerm = searchNewTerminal(agn);
        modifyCurrentWorkflow(agn,newTerm);
end
end

% terminal not operational
if( ~terminal(ag(agn).finalterminal).operational )
if( ~ag(agn).pda || ~terminal(ag(agn).finalterminal).wireless )
```

```
if( distanceBetween(ag(agn),terminal(ag(agn).finalterminal)) <
2*BOUNDARY  )
            newTerm = searchNewTerminal(agn);
            modifyCurrentWorkflow(agn,newTerm);
end
else
if( distanceBetween(ag(agn),terminal(ag(agn).finalterminal)) <
WIRELESSRANGE )
            newTerm = searchNewTerminal(agn);
            modifyCurrentWorkflow(agn,newTerm);
end
end
end

% set booleans for destination reached may be a mchine or a terminal
if( terminal(ag(agn).finalterminal).wireless && ag(agn).pda )
if( (abs(ag(agn).x - ag(agn).finalx(1)) < distApp) &&...
            (abs(ag(agn).y - ag(agn).finaly(1)) < distApp))
if( ag(agn).waittime == 0)
            ag(agn).destinationreached = true;
            ag(agn).waittime = curtime;
            setBoolsWireless(agn);
end
end
else
if( (abs(ag(agn).x - ag(agn).finalx(1)) < distApp) &&...
            (abs(ag(agn).y - ag(agn).finaly(1)) < distApp))
if( ag(agn).waittime == 0)
            ag(agn).destinationreached = true;
            ag(agn).waittime = curtime;
            setBoolsWired(agn);
end
end
end



%
function setBoolsWired(agn)
%
% function setBoolsWired(agn)
%
% This function sets various booleans
%
% agn                 agent number
%
global ag;
global terminal;

global roomNumbering;
global BOUNDARY;

global simulationTime;
```

```
if( ((abs(ag(agn).x - terminal(ag(agn).finalterminal).x)) < BOUNDARY)
&&...
        ((abs(ag(agn).y - terminal(ag(agn).finalterminal).y)) <
BOUNDARY))

    curtime = simulationTime;
    ag(agn).waittime = 0;

if( ag(agn).getinfo && ~ag(agn).inforeceived )
        ag(agn).inforeceived = true;
        ag(agn).datarettime =
machine(ag(agn).machine).maintenanceData.wiredDelay;

        ag(agn).inspectiondone = true;
        ag(agn).packet.size =
machine(ag(agn).machine).maintenanceData.infoDataSize;
        ag(agn).packet.delay =
machine(ag(agn).machine).maintenanceData.wiredDelay;
end

if( ~terminal(ag(agn).finalterminal).occupied &&
terminal(ag(agn).finalterminal).operational )

        ag(agn).terminalreached = true;
        ag(agn).timer = clock;
        ag(agn).busy = true;
        ag(agn).timetermreached = curtime;

        terminal(ag(agn).finalterminal).occupied = true;
        terminal(ag(agn).finalterminal).agent = agn;

        ag(agn).inclagtimedata = ag(agn).stress*2 + (3-
ag(agn).training)*2;

if( ag(agn).inspectiondone )
            ag(agn).inspectiondone = false;
            ag(agn).busy = false;


%Added for network testing
            ag(agn).packet.origin = ag(agn).finalterminal;
            ag(agn).packet.percent = 0;
            ag(agn).packet.transmitted = 0;

updateNetworkGraph(ag(agn),terminal(ag(agn).finalterminal),1);
            queuePacket(ag(agn).packet,ag(agn).packet.destination);
end

if( ag(agn).watchStarted )
            ag(agn).watchStarted = false;
end
```

```
else
if( terminal(ag(agn).finalterminal).agent ~= agn)
            newTerm = searchNewTerminal(agn);
            modifyCurrentWorkflow(agn,newTerm);
end
end
else
    ag(agn).machinereached = false;
    ag(agn).terminalreached = false;

if( ag(agn).waitno ~= 0 )
        1;
else
        ag(agn).destinationreached = false;
        ag(agn).waittime = 0;
end

if( roomNumbering(round(ag(agn).y),round(ag(agn).x))...
            ~=
roomNumbering(round(terminal(ag(agn).finalterminal).y),...
            round(terminal(ag(agn).finalterminal).x)) )
        ag(agn).destinationreached = false;
        ag(agn).waittime = 0;
end
end



%
function setBoolsWireless(agn)
%
% function setBoolsWireless(agn)
%
% This function sets various booleans
%
% agn                 agent number
%
global ag;
global terminal;

global WIRELESSRANGE;

global simulationTime;

curtime = simulationTime;

if( distanceBetween(ag(agn),terminal(ag(agn).finalterminal)) <
WIRELESSRANGE &&...
        terminal(ag(agn).finalterminal).operational )

if( ~(terminal(ag(agn).finalterminal).wirelessoccupied > 99) )
        ag(agn).waittime = 0;
```

```
if( ag(agn).getinfo && ~ag(agn).inforeceived )
            ag(agn).inforeceived = true;
            ag(agn).datarettime =
machine(ag(agn).machine).maintenanceData.wirelessDelay;

            ag(agn).inspectiondone = true;
            ag(agn).packet.size =
machine(ag(agn).machine).maintenanceData.infoDataSize;
            ag(agn).packet.delay =
machine(ag(agn).machine).maintenanceData.wirelessDelay;
end

        val = 0;
for i = 1:terminal(ag(agn).finalterminal).wirelessoccupied
if(agn == terminal(ag(agn).finalterminal).wirelessagent(i) )
                val = 1;
end
end

if( agn == terminal(ag(agn).finalterminal).agent )
            val = 1;
end

if( ~val )
            ag(agn).terminalreached = true;
            ag(agn).timer = clock;
            ag(agn).busy = true;
            ag(agn).timetermreached = curtime;

            terminal(ag(agn).finalterminal).wirelessoccupied =
terminal(ag(agn).finalterminal).wirelessoccupied + 1;
            count = terminal(ag(agn).finalterminal).wirelessoccupied;
            terminal(ag(agn).finalterminal).wirelessagent(count) = agn;

            ag(agn).inclagtimedata = ag(agn).stress*2 + (3-
ag(agn).training)*2 ;

if( ag(agn).inspectiondone )
                ag(agn).inspectiondone = false;
%Added for network testing
                ag(agn).packet.origin = ag(agn).finalterminal;
                ag(agn).packet.percent = 0;
                ag(agn).packet.transmitted = 0;

updateNetworkGraph(ag(agn),terminal(ag(agn).finalterminal),1);
                queuePacket(ag(agn).packet,ag(agn).packet.destination);
end
end
else
if( terminal(ag(agn).finalterminal).agent ~= agn)
            newTerm = searchNewTerminal(agn);
            modifyCurrentWorkflow(agn,newTerm);
```

```
end
end
end



function modifyCurrentWorkflow(agn,newTerm)
%
% function modifyCurrentWorkflow(agn,newTerm)
%
% This function modifies the finalterminal in the workflow
%
%   agn           agent number
%   newTerm       new Access Point/Workstation
%
global ag;

global WORKSTACK;

global simulationTime;

curtime = simulationTime;

r = [];
if ~isempty(WORKSTACK) && ~isempty(ag(agn).workstack)
    r = find([WORKSTACK.id] == ag(agn).workstack(1).id);
end

if ~isempty(r)
    WORKSTACK(r).ono = newTerm;
    ag(agn).workstack(1).ono = newTerm;

    ag(agn).inspectiondone = true;
    ag(agn).packet.percent = 0;
if( ~ag(agn).getinfo )
        ag(agn).inforeceived = false;
end
if( ag(agn).getinfo )
        ag(agn).percentTaskComp(length(ag(agn).percentTaskComp) + 1) =
0;
else
        ag(agn).percentTaskComp(length(ag(agn).percentTaskComp) + 1) =
50;
end
    ag(agn).ptcTime(length(ag(agn).ptcTime) +  1) = curtime;
    setAgentOrder(agn,ag(agn).workstack(1));
else
    premovement(agn,newTerm,0);
end

function newTerm = searchNewTerminal(agn)
%
```

```
% SEARCHNEWTERMINAL This function is used to search new terminals for
the
% agents if the current terminal is occupied or not functional
%
% USAGE:
%       newTerm = searchNewTerminal(agn)
%
%           agn         agent number
%           newTerm     new terminal
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 17 July 2006
% Modified by: Robin Kusmanto
%
% Last Modified: 21 May 2009
%

global ag;
global terminal;

if ( ag(agn).pda )
    flagA = false;
for nterm =1:6
if terminal(nterm).operational
           flagA = true;
end
end
if flagA
switch ag(agn).finalterminal
case 1
                newTerm = 2;
case 2
if( ag(agn).preterminal == 5 )
                    newTerm = 1;
else
                    newTerm = 5;
end
case 3
if( ag(agn).preterminal == 5 )
                    newTerm = 4;
else
                    newTerm = 5;
end
case 4
if( ag(agn).preterminal == 3 )
                    newTerm = 6;
else
                    newTerm = 3;
end
case 5
if( ag(agn).preterminal == 3 )
                    newTerm = 2;
else
```

```
                    newTerm = 3;
end
case 6
                newTerm = 4;
otherwise
                newTerm = getClosestTerminalforAgent(agn);
end
else
switch ag(agn).finalterminal
case 7
                newTerm = 8;
case 8
                newTerm = 7;
otherwise
                newTerm = 7;
end
end
elseif( ~ag(agn).pda )
switch ag(agn).finalterminal
case 7
            newTerm = 8;
case 8
            newTerm = 7;
otherwise
            newTerm = getClosestTerminalforAgent(agn);
end
end

function term = getClosestTerminalforAgent(agn)
%
% GETCLOSESTTERMINALFORAGENT This function finds an AP or Workstation
% closest to an agent
%
% USAGE:
%   term = getClosestTerminalforAgent(agn)
%
%       agn         agent Number
%       term        AP or Workstation number
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 17 July 2006
% Modified by: Robin Kusmanto
%
% Last Modified: 21 May 2009
%

global ag;
global terminal;

min = 1e10;
term = 0;
if( ag(agn).pda )
for i = 1:length(terminal)
```

```
        path = getPathforAgent(agn,terminal(i));
        dist = sqrt( (ag(agn).x-path(1).x)^2 + (ag(agn).y-
path(1).y)^2 );
for k=1:length(path)-1
            dist = sqrt( (path(k).x-path(k+1).x)^2 + (path(k).y-
path(k+1).y)^2  ) + dist;
end
if( dist < min )
            min = dist;
            term = i;
end
end
elseif( ~ag(agn).pda )
for i = 1:length(terminal)
if strcmpi(terminal(i).type,'Workstation')
            path = getPathforAgent(agn,terminal(i));
            dist = sqrt( (ag(agn).x-path(1).x)^2 + (ag(agn).y-
path(1).y)^2 );
for k=1:length(path)-1
                dist = sqrt( (path(k).x-path(k+1).x)^2 + (path(k).y-
path(k+1).y)^2  ) + dist;
end
if( dist < min )
                min = dist;
                term = i;
end
end
end
end

function machineChecks(agn)
%
% MACHINECHECKS This function makes checks on agents inspecting
machines
%
% USAGE:
%   machineChecks(agn)
%
%       agn           agent number
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 13 October 2006
% Modification History:
% Old Date      New Date            Modified by         Changes
% 7 Jan 2007    29 Jan 2007         Vishal Mahulkar
%
% Version History:
%
% Last Modified: 7 January 2007
%
global ag;
global machine;
```

```
global BOUNDARY;
global simulationTime;

persistent machineCntrHandleList;

curtime = simulationTime;

if( isempty(machineCntrHandleList) )
    machineCntrHandle = findall(0,'tag','machineCntr');
    machineCntrHandleList = guidata(machineCntrHandle);
end

if( ((abs(ag(agn).x - machine(ag(agn).machine).x)) < BOUNDARY) ...
&& ((abs(ag(agn).y - machine(ag(agn).machine).y)) < BOUNDARY) )

    ag(agn).destinationreached = true;
    ag(agn).waittime = 0;

if ~ag(agn).machinereached || ag(agn).coordinate
if( ~ag(agn).machinereached )
            ag(agn).timemcreached = curtime;
            ag(agn).finalterminal = getClosestTerminalforAgent(agn);

            ag(agn).percentTaskComp(length(ag(agn).percentTaskComp)  +  1)
= 0;
            ag(agn).ptcTime(length(ag(agn).ptcTime) + 1) = curtime;
            updateNetworkGraph(ag(agn),machine(ag(agn).machine),1);
end

        ag(agn).machinereached = true;
        ag(agn).timer = clock;
        ag(agn).inclagtimeinsp = ag(agn).stress * 2 + (3-
ag(agn).training) * 2;
end
end

if ag(agn).pda
    inspectionTime = ag(agn).workData.inspectionTimeWireless;
else
    inspectionTime = ag(agn).workData.inspectionTimeWired;
end

if ~ag(agn).coordinate && ag(agn).machinereached &&
~ag(agn).inspectiondone
if curtime ~= ag(agn).ptcTime(length(ag(agn).ptcTime))
        ag(agn).percentTaskComp(length(ag(agn).percentTaskComp) + 1)
= ...
            .5*100*abs(curtime - ag(agn).timemcreached)/...
            (inspectionTime + ag(agn).inclagtimeinsp);
        ag(agn).ptcTime(length(ag(agn).ptcTime) + 1) = curtime;
end
end
```

```
if( ~ag(agn).coordinate &&...
        ag(agn).machinereached &&...
        (curtime - ag(agn).timemcreached) > inspectionTime +
ag(agn).inclagtimeinsp )

    ag(agn).inspectiondone = true;

% display data
if strcmpi(ag(agn).workstack(1).type,'inspect')
        displayString(['Machine ',num2str(ag(agn).machine),'
inspected']);

% new workflow
        removeCompletedWorkflow(agn);
        object = 'terminal';
        data =
insertWorkflow(struct('subject','agent','sno',agn,'object',object,...
'ono',ag(agn).finalterminal,'priority',4,'type','report','coordinate',0
,'status','toStart','userData',[]));
        setupOrderStack(object,data);

elseif strcmpi(ag(agn).workstack(1).type,'troubleshoot')
        displayString(['Fault found in machine
',num2str(ag(agn).machine)]);

% new workflow
        removeCompletedWorkflow(agn);
        object = 'machine';
        data =
insertWorkflow(struct('subject','agent','sno',agn,'object',object,...
'ono',ag(agn).machine,'priority',4,'type','repair','coordinate',0,'stat
us','toStart','userData',[]));
        setupOrderStack(object,data);

elseif strcmpi(ag(agn).workstack(1).type,'repair')
        displayString(['Machine ',num2str(ag(agn).machine),'
repaired']);
        machine(ag(agn).machine).justOnce = false;
        machine(ag(agn).machine).status = {'on'};
        str = strcat('faultmac',num2str(ag(agn).machine));
        set(machineCntrHandleList.(str),'Value',0);

machinecntr(strcat(str,'_Callback'),machineCntrHandleList.(str),1,machi
neCntrHandleList);

% new workflow
        removeCompletedWorkflow(agn);
        object = 'terminal';
        data =
insertWorkflow(struct('subject','agent','sno',agn,'object',object,...
'ono',ag(agn).finalterminal,'priority',4,'type','report','coordinate',0
,'status','toStart','userData',[]));
```

```
        setupOrderStack(object,data);
end
end

% if two or more agents have to coordinate
if ag(agn).coordinate && ag(agn).machinereached
    count1 = 0;
    count2 = 0;

% both have reached machines
for index = 1:length(ag(agn).coordinatewith)
if( ag(ag(agn).coordinatewith(index)).machinereached )
            count1 = count1 + 1;
end
end

% both have finished inspection
if( count1 == length(ag(agn).coordinatewith) )
for index = 1:length(ag(agn).coordinatewith)
            ag(ag(agn).coordinatewith(index)).timetermreached =
max([ag(ag(agn).coordinatewith).timetermreached]);
if( (length(ag(ag(agn).coordinatewith(index)).timemcreached) > 0) &&...
                    ( (curtime -
ag(ag(agn).coordinatewith(index)).timemcreached) >...

ag(ag(agn).coordinatewith(index)).workData.inspectionTimeWireless + ...
                    ag(ag(agn).coordinatewith(index)).inclagtimeinsp) )
                count2 = count2 + 1;
end
end
end

if( count2 == length(ag(agn).coordinatewith) )
for index = 1:length(ag(agn).coordinatewith)
            ag(ag(agn).coordinatewith(index)).coordinate = false;
            ag(ag(agn).coordinatewith(index)).inspectiondone = true;

% new workflow
            removeCompletedWorkflow(ag(agn).coordinatewith(index));
            object = 'terminal';
            data =
insertWorkflow(struct('subject','agent','sno',ag(agn).coordinatewith(in
dex),'object',object,...
'ono',ag(ag(agn).coordinatewith(index)).finalterminal,'priority',4,'typ
e','report','coordinate',0,'status','toStart','userData',[]));
            setupOrderStack(object,data);
end

% display data
if ~isempty(ag(agn).workstack)
if strcmpi(ag(agn).workstack(1).type,'inspect')
                displayString(['Machine ',num2str(ag(agn).machine),'
inspected']);
```

```
elseif strcmpi(ag(agn).workstack(1).type,'troubleshoot')
                displayString(['Fault found in machine
',num2str(ag(agn).machine)]);
elseif strcmpi(ag(agn).workstack(1).type,'repair')
                displayString(['Machine ',num2str(ag(agn).machine),'
repaired']);
                machine(ag(agn).machine).justOnce = false;
                machine(ag(agn).machine).status = {'on'};
                str = strcat('faultmac',num2str(ag(agn).machine));
                set(machineCntrHandleList.(str),'Value',0);

machinecntr(strcat(str,'_Callback'),machineCntrHandleList.(str),1,machi
neCntrHandleList);
end
end
end

if ag(agn).coordinatewith(index) == 5
        1;
end
% use larger of two time machine reached
if( count1 == length(ag(agn).coordinatewith) )

if curtime ~= ag(agn).ptcTime(length(ag(agn).ptcTime))
            ag(agn).percentTaskComp(length(ag(agn).percentTaskComp)  +  1)
= ...
                min(.5*100*abs(curtime-ag(agn).timemcreached)/...
                (inspectionTime + ag(agn).inclagtimeinsp),...
                50);
            ag(agn).ptcTime(length(ag(agn).ptcTime) + 1) = curtime;
end
end
end

function watchlocationChecks(agn)
%
% WATCHLOCATIONCHECKS This function makes checks on agents on watch
duty
%
% USAGE:
%   watchlocationChecks(agn)
%
%       agn           agent number
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 13 October 2006
% Version History:
%
% Last Modified: 28 December 2006
%
global ag;
global watchLoc;
```

```
global BOUNDARY;

global simulationTime;

curtime = simulationTime;

if( ((abs(ag(agn).x - watchLoc(ag(agn).watchLocation).x)) <
BOUNDARY) ...
&& ((abs(ag(agn).y - watchLoc(ag(agn).watchLocation).y)) < BOUNDARY) )

    ag(agn).destinationreached = true;
    ag(agn).waittime = 0;

if( ~ag(agn).machinereached || ag(agn).coordinate )
if( ~ag(agn).machinereached )
            ag(agn).timemcreached = curtime;
            ag(agn).finalterminal = getClosestTerminalforAgent(agn);

updateNetworkGraph(ag(agn),watchLoc(ag(agn).watchLocation),1);
end

        ag(agn).machinereached = true;
        ag(agn).timer = clock;
        ag(agn).inclagtimeinsp = ag(agn).stress*2 + (3-
ag(agn).training)*2;
end

if( ag(agn).machinereached && ~ag(agn).inspectiondone )
        ag(agn).percentTaskComp(length(ag(agn).percentTaskComp) + 1)
= ...
            min(.5*100*abs(curtime - ag(agn).timemcreached)/...
            (ag(agn).workData.watchTime + ag(agn).inclagtimeinsp),50);
        ag(agn).ptcTime(length(ag(agn).ptcTime) + 1) = curtime;
end

if( ag(agn).machinereached && (curtime - ag(agn).timemcreached) >
ag(agn).workData.watchTime + ag(agn).inclagtimeinsp )
        ag(agn).inspectiondone = true;

        removeCompletedWorkflow(agn);

        object = 'terminal';
        data =
insertWorkflow(struct('subject','agent','sno',agn,'object',object,...
'ono',ag(agn).finalterminal,'priority',4,'type','report','coordinate',0
,'status','toStart','userData',[]));

        setupOrderStack(object,data);
end
end

function firefightingChecks(agn)
```

```
%
% FIREFIGHTINGCHECKS This function makes checks on agents getting
equipment
%
% USAGE:
%   getEquipChecks(agn)
%
%       agn           agent number
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 12 December 2006
% Version History:
%
% Last Modified: 28 December 2006
%
global ag;
global emergencyScenarios;

global simulationTime;

curtime = simulationTime;

r = [];
if ~isempty(emergencyScenarios)
    r = find([emergencyScenarios.id] == ag(agn).workstack(1).ono);
end

if ~isempty(r)

if distanceBetween(ag(agn),emergencyScenarios(r)) <
(emergencyScenarios(r).userData.radiusF+...
            emergencyScenarios(r).userData.radiusS)/2
if ~ag(agn).emergencyLocreached
            ag(agn).emergencyLocreached = true;
            ag(agn).timeemergencyLocreached = curtime;
            emergencyScenarios(r).agents =
[emergencyScenarios(r).agents, agn];
end
end
end

function getEquipChecks(agn)
%
% GETEQUIPMENTCHECKS This function make checks on agents getting
equipment
%
% USAGE:
%   getEquipChecks(agn)
%
%       agn           agent number
%
% -----------
```

```
% Created by: Vishal Mahulkar
% Created on: 12 December 2006
% Version History:
%
% Last Modified: 21 December 2006
%
global ag;
global fireEquip;
global equip;

global BOUNDARY;

global simulationTime;

curtime = simulationTime;

if( ((abs(ag(agn).x - fireEquip(ag(agn).equip).x)) < BOUNDARY) ...
&& ((abs(ag(agn).y - fireEquip(ag(agn).equip).y)) < BOUNDARY) )

    ag(agn).destinationreached = true;
    ag(agn).waittime = 0;

if( ~ag(agn).equipreached )
        ag(agn).timeequipreached = curtime;
        updateNetworkGraph(ag(agn),fireEquip(ag(agn).equip),1);
end

    ag(agn).equipreached = true;
    ag(agn).timer = clock;
    ag(agn).inclagtimeinsp = ag(agn).stress*2 + (3-ag(agn).training)*2;

if( ag(agn).equipreached && (curtime - ag(agn).timeequipreached) >
ag(agn).timetogetequip + ag(agn).inclagtimeinsp )
        ag(agn).carryEquip = 'firefighting';
        ono = ag(agn).workstack(1).ono;

        removeCompletedWorkflow(agn);

        object = 'firefighting';
        priority = 6;
        type = 'emergency';

        data =
insertWorkflow(struct('subject','agent','sno',agn,'object',object,'ono'
,...

ono,'priority',priority,'type',type,'coordinate',0,'status','toStart','
userData',[]));
        setupOrderStack(object,data);
end
end
```

```
function recreationChecks(agn)
%
% RECREATIONCHECKS function used check status of agents at recreation
% locations
%
% USAGE:
%   recreationChecks(agn)
%       agn           agent number
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 15 January 2007
% Version History:
%
% Last Modified: 15 January 2007
%
global ag;
global fireEquip;
global equip;

global BOUNDARY;
global RECREATIONTIME;

global simulationTime;

curtime = simulationTime;

if( ((abs(ag(agn).x - equip(ag(agn).equip).x)) < BOUNDARY) ...
&& ((abs(ag(agn).y - equip(ag(agn).equip).y)) < BOUNDARY) )

    ag(agn).destinationreached = true;
    ag(agn).waittime = 0;

if( ~ag(agn).equipreached )
        ag(agn).timeequipreached = curtime;
        updateNetworkGraph(ag(agn),equip(ag(agn).equip),1);
end

    ag(agn).equipreached = true;
    ag(agn).timer = clock;
    ag(agn).inclagtimeinsp = ag(agn).stress*2 + (3-ag(agn).training)*2;

if( ag(agn).equipreached && (curtime - ag(agn).timeequipreached) ...
> RECREATIONTIME )
        ag(agn).carryEquip = '';

        removeCompletedWorkflow(agn);
if ~isempty(ag(agn).workstack)
            setAgentOrder(agn,ag(agn).workstack(1));
end
else
        ag(agn).percentTaskComp(length(ag(agn).percentTaskComp) + 1) =
ag(agn).percentTaskComp(length(ag(agn).percentTaskComp));
```

```
        ag(agn).ptcTime(length(ag(agn).ptcTime) + 1) = curtime;
end
end

function postProcessing(varargin)
%
% POSTPRECESSING This function post processes and displays some of the
data
% collected during simulation
%
% USAGE:
%   poatProcessing()
%
% -----------
% Created by: Vishal Mahulkar
% Created on: 17 July 2006
% Modified by: Robin Kusmanto
%
% Last Modified: 15 May 2009
%

global ag;
global estcost;
global cost;
global N;
global bwTermTime;
global terminal;
global server;
global ag1dataptc ag2dataptc ag3dataptc ag4dataptc ag5dataptc
ag6dataptc...
    ag7dataptc ag8dataptc ag9dataptc ag10dataptc;
global agCurrentTaskCTime;
global agCurrentTaskCData;
global agTotalTaskCTime;
global agTotalTaskCData;
global agTotalUtiTime;
global agTotalUtiData;
global availabilityData;
global availabilityTime;
global staticLoad;
global BANDWIDTH;

if nargin == 1
    eval(['save ',varargin{1},' ag bwTermTime terminal server
agCurrentTaskCTime agCurrentTaskCData agTotalTaskCTime agTotalTaskCData
agTotalUtiTime agTotalUtiData BANDWIDTH staticLoad '])
end

bwRecord = 1;
agRecord = 0;
workflowRecord = 1;
utiRecord = 1;
totalWorkflow = 1;
```

```
minDiff = .1;
maxTime = 0;
for agn=1:10
if( ag(agn).ptcTime(end) > maxTime )
        maxTime = ag(agn).ptcTime(end);
end
end

numberofplots = 3;
fignum = 1;
i1=0;
i2=0;
i3=0;
i4=0;
i5=0;
i6=0;
i7=0;
i8=0;
for i = 1:length(ag)
if( ag(i).lastTerminalAccessed == 1)
        i1=i1+1;
        timereached1(i1) = ag(i).timeOrderCompleted;
end
if( ag(i).lastTerminalAccessed == 2)
        i2=i2+1;
        timereached2(i2) = ag(i).timeOrderCompleted;
end
if( ag(i).lastTerminalAccessed == 3)
        i3=i3+1;
        timereached3(i3) = ag(i).timeOrderCompleted;
end
if( ag(i).lastTerminalAccessed == 4)
        i4=i4+1;
        timereached4(i4) = ag(i).timeOrderCompleted;
end
if( ag(i).lastTerminalAccessed == 5)
        i5=i5+1;
        timereached5(i5) = ag(i).timeOrderCompleted;
end
if( ag(i).lastTerminalAccessed == 6)
        i6=i6+1;
        timereached6(i6) = ag(i).timeOrderCompleted;
end
if( ag(i).lastTerminalAccessed == 7)
        i7=i7+1;
        timereached7(i7) = ag(i).timeOrderCompleted;
end
if( ag(i).lastTerminalAccessed == 8)
        i8=i8+1;
        timereached8(i8) = ag(i).timeOrderCompleted;
end
end

%     num = bwTermTime(length(bwTermTime))+10;
```

```
num = [];
for i = 1:8
    terminal(i).bwRecordTime = bwTermTime;
    num = [num terminal(i).bwRecordTime(end)];
end
num = max(num) + 10;

%% bandwidth record
if bwRecord
    figure;
    title('Terminal Bandwidths');
    subplot(numberofplots,2,1);
if( exist('timereached1','var'))
        plot(timereached1,zeros(1,length(timereached1)),'ro');
        axis([0 num 0 6e6]);
        hold;
end
    plot(terminal(1).bwRecordTime,terminal(1).bwRecord);
    axis([0 num 0 6e6]);
    title('Access Point 1');

    subplot(numberofplots,2,2);
if( exist('timereached2','var'))
        plot(timereached2,zeros(1,length(timereached2)),'ro');
        axis([0 num 0 6e6]);
        hold;
end
    plot(terminal(2).bwRecordTime,terminal(2).bwRecord);
    axis([0 num 0 6e6]);
    title('Access Point 2');

    subplot(numberofplots,2,3);
if( exist('timereached3','var'))
        plot(timereached3,zeros(1,length(timereached3)),'ro');
        axis([0 num 0 6e6]);
        hold;
end
    plot(terminal(3).bwRecordTime,terminal(3).bwRecord);
    axis([0 num 0 6e6]);
    title('Access Point 3');

    subplot(numberofplots,2,4);
if( exist('timereached4','var'))
        plot(timereached4,zeros(1,length(timereached4)),'ro');
        axis([0 num 0 6e6]);
        hold;
end
    plot(terminal(4).bwRecordTime,terminal(4).bwRecord);
    axis([0 num 0 6e6]);
    title('Access Point 4');

    figure
    plot(server(1).bwRecord(1,:), server(1).bwRecord(2,:)+
staticLoad*BANDWIDTH);
```

```
    axis([0 num 0 BANDWIDTH*1.2]);
    title('Server Bandwidth Utilized');
    xlabel('Time sec');
    ylabel('MBits/s')
end

%% agent record
if agRecord
    figure;
    numberofplots = 4;
    title('Terminal Bandwidths');
    subplot(numberofplots,2,1);
if( exist('timereached1','var'))
        plot(timereached1,zeros(1,length(timereached1)),'ro');
        hold;
end
    plot(terminal(1).bwRecordTime,terminal(1).agRecord);
    title('Access Point 1');

    subplot(numberofplots,2,2);
if( exist('timereached2','var'))
        plot(timereached2,zeros(1,length(timereached2)),'ro');
        hold;
end
    plot(terminal(2).bwRecordTime,terminal(2).agRecord);
    title('Access Point 2');

    subplot(numberofplots,2,3);
if( exist('timereached3','var'))
        plot(timereached3,zeros(1,length(timereached3)),'ro');
        hold;
end
    plot(terminal(3).bwRecordTime,terminal(3).agRecord);
    title('Access Point 3');

    subplot(numberofplots,2,4);
if( exist('timereached4','var'))
        plot(timereached4,zeros(1,length(timereached4)),'ro');
        hold;
end
    plot(terminal(4).bwRecordTime,terminal(4).agRecord);
    title('Access Point 4');

end

%% workflow
if workflowRecord
    figure;
for currAgentSel = 1:length(ag)
        subplot(5,2,currAgentSel)
switch currAgentSel
case 1
```

```
plot(ag(currAgentSel).ptcTime,ag(currAgentSel).percentTaskComp,'r');
                title('Agent 1')
%             legend('Baseline','Current')
case 2

plot(ag(currAgentSel).ptcTime,ag(currAgentSel).percentTaskComp,'r');
                title('Agent 2')
%             legend('Baseline','Current')
case 3

plot(ag(currAgentSel).ptcTime,ag(currAgentSel).percentTaskComp,'r');
                title('Agent 3')
%             legend('Baseline','Current')
case 4

plot(ag(currAgentSel).ptcTime,ag(currAgentSel).percentTaskComp,'r');
                title('Agent 4')
%             legend('Baseline','Current')
case 5

plot(ag(currAgentSel).ptcTime,ag(currAgentSel).percentTaskComp,'r');
                title('Agent 5')
                ylabel('% Work Completed')
%             legend('Baseline','Current')
case 6

plot(ag(currAgentSel).ptcTime,ag(currAgentSel).percentTaskComp,'r');
                title('Agent 6')
%             legend('Baseline','Current')
case 7

plot(ag(currAgentSel).ptcTime,ag(currAgentSel).percentTaskComp,'r');
                title('Agent 7')
%             legend('Baseline','Current')
case 8

plot(ag(currAgentSel).ptcTime,ag(currAgentSel).percentTaskComp,'r');
                title('Agent 8')
%             legend('Baseline','Current')
case 9

plot(ag(currAgentSel).ptcTime,ag(currAgentSel).percentTaskComp,'r');
                title('Agent 9')
                xlabel('Agent WorkFlow Progress: Current, (s)')
%             legend('Baseline','Current')
case 10

plot(ag(currAgentSel).ptcTime,ag(currAgentSel).percentTaskComp,'r');
                title('Agent 10')
                xlabel('Agent WorkFlow Progress: Current, (s)')
%             legend('Baseline','Current')
end
end
end
```

```
if utiRecord
    figure
for agn = 1:10
        subplot(6,2,agn)
        plot(ag(agn).utiTime,ag(agn).utilization);
        title(['Agent ',num2str(agn)]);
if agn>8
            xlabel('Time');
end
if agn==5 || agn == 6
            ylabel('Utilization')
end
end
    subplot(6,1,6)
    plot(agTotalUtiTime,agTotalUtiData);
    grid
    title('Total Utilization')
    xlabel('Time')
    ylabel('Utilization')
end

if totalWorkflow
    figure
    plot(agTotalTaskCTime,agTotalTaskCData)
    grid
    title('Total workflow completion progress')
    xlabel('Time (s)')
    ylabel('% Workflow Completion')
    set(gca,'YLim',[0 1]);
end

if 0
global aveTask
    figure;
    plot(aveTask(2,:),aveTask(1,:));
    grid
end

figure
subplot(311)
h1 = plot(agTotalUtiTime,agTotalUtiData);
set(gca,'YLim',[0 1]);
ylabel({'Average total','Utilization'});
subplot(312)
h2 = plot(agTotalTaskCTime,agTotalTaskCData);
set(gca,'YLim',[0 1]);
ylabel({'Average total','Work'});
subplot(313)
h3 = plot(availabilityTime,availabilityData);
set(gca,'YLim',[0 1]);
xlabel('Time')
ylabel({'Average total','Availability'});
```

```
%%%%%%%%%%%%%%% AP Utilization%%%%%%%%%%%%%%%%%
terminalUtil = [0 0 0 0];
for i = 1:4
for j = 1:length(terminal(i).bwRecordTime)
if terminal(i).bwRecord(j) ~= 0
            terminalUtil(i) = terminalUtil(i) + 1;
end
end
    terminalPercentUtil(i) = terminalUtil(i) /
length(terminal(i).bwRecordTime);
end
terminalPercentUtil
MaxUtil = max(terminalPercentUtil)
MinUtil = min(terminalPercentUtil)
AveUtil = mean(terminalPercentUtil)
FinalAgUtilization = agTotalUtiData(end)
```

APPENDIX B. ZONAL MODEL PARAMETER SETUP

Factor name	Parameters	Low	High (or nominal)	units
X1	AP1 operational	FALSE	TRUE	na
X2	AP2 operationa	FALSE	TRUE	na
X3	Agent walking speed (single value)	1	4	ft/s
X4	AP range	200	240	feet
X5	AP3 operational	FALSE	TRUE	na
X6	AP4 operational	FALSE	TRUE	na
X7	AP5 operationa	FALSE	TRUE	na
X8	Data transfer to AP 1	TRUE	FALSE	bs
X9	AP locations	2	2	1-At nearest internet port 2 - Nominal
X10	Intellegent Maintenance (w/sensor data)	TRUE	FALSE	1 - together 2 - seperated
X11	Workstation location	2	2	1 - center 2- far end of zone
X12	Bandwidth of the entire network	2.E+07	2.E+07	bs
X13	Data transfer to AP 2	TRUE	FALSE	
X14	Bandwidth for Wireless	5.50E+06	2.80E+07	bs
X15	Fixed network load	10	10	%of the network bandwidth
X16	Time to write report	5	20	minutes
X17	Report data size	1000000	2000000	bs
X18	Data transfer to AP 3	TRUE	FALSE	
X19	Data transfer to AP 4	TRUE	FALSE	
X20	Fire start time	2	14	military time
X21	Max power for the ship (define the max power for the rooms)	70	80	
X22	Data transfer to AP 5	TRUE	FALSE	
X23	Sensor data priority	2	4	kbs
X24	Data transfer to AP 6	TRUE	FALSE	

X25	Training level (varies time to complete a task)	0	2	training level
X26	Information needed for machine inspection	1	2	1- all machines need data 2- two random machines need data
X27	Total number of crew in zone	7	10	# of crew
X28	Maintaince- unschedule	9	15	# of failures
X29	Scheduled events (.5 PM and .5 watches)	12	18	# of PMs
X30	Repair/watch/duration	2	4	1 - all tasks take half the time 2 - nominal
X31	% of failures intellengent maintenance detects	25	50	# of times
X32	Failure times	2	2	1- psuedo random 2 - equal space
X33	Stress level (varies time)	1	5	setting
X34	Maintenance aid device (laptop, PDA)	1	2	1- Agents 1-5 PDA 2 - all agents PDA
X35	Amount of time prior to failure intellegent maintenance identifies error	1	2	hours
X36	AP6 operational	FALSE	TRUE	na
X37	workstations operational	1	2	1 - Work station 7 failed 2 - all work stations operational
X38	Server operational	1	2	1 - sev 1 fail 2 - all serv operational
X39	Inspection Data needed	75	150	kbs
X40	Agents data priority	1	2	1 - agents 1, 4, 8 have low priority 2 - all agents have same priority

X41	Machine inspection required	1	2	1 - only serv 3 has high priority 2- all servers have same priority
X42	N/A	N/A	N/A	N/A
X43	N/A	N/A	N/A	N/A
X44	Fire	1	2	1 - nominal w/o fire 2- fire
X45	Power failure	2	2	1 - nominal 2- power failure for 3 hrs
X46	N/A	N/A	N/A	N/A
X47	Room power priority	1	2	1 - nominal with default room priority 2 -room priority set psuedorandom
X48	Task Priority	1	2	1 - nominal 2 - all task have same priority
X49	# of doors	19	25	# of doors
X50	Smoke rate	0.3	0.3	m/s
X51	fire rate	0.1	0.1	m/s
X52	Fire start location	1	2	1 - center of ship 2 - corner of ship
X53	Power failure	2	2	1 - nominal 2- power failure for 3 hrs
X54	Time of power failure	1	1	military time
X55	Duration of power failure	1	2	hours
X56	zonal model area	1	2	1 - is nominal configuration 2 - zone is extended by two rooms
X57	Model Steady-State	1	2	1- model is executed in a trasient state 2- model is executed in a steady-state

APPENDIX C. DESIGN OF EXPERIMENTS RESULTS

RunOrder	NumAgents	NumAP	Fire	APFailure	Workstations	ReportPrio	DataSize
1	10	4	No	Yes	Yes	2	2
2	7	2	No	Yes	No	2	1
3	7	4	Yes	No	No	4	1
4	7	2	Yes	No	No	2	2
5	7	2	No	No	Yes	4	1
6	7	4	No	Yes	No	4	2
7	10	4	No	No	No	4	2
8	7	4	Yes	Yes	Yes	2	1
9	7	2	No	Yes	Yes	2	2
10	10	2	No	Yes	No	4	2
11	7	2	Yes	Yes	No	4	1
12	10	2	No	No	No	2	1
13	10	4	No	Yes	No	2	1
14	7	4	No	No	Yes	2	2
15	7	4	No	No	No	2	1
16	10	4	No	No	Yes	4	1
17	7	4	Yes	Yes	No	2	2
18	10	4	Yes	No	No	2	2
19	10	4	Yes	Yes	No	4	1
20	7	4	Yes	No	Yes	4	2
21	10	2	No	No	Yes	2	2
22	7	2	Yes	No	Yes	2	1
23	7	4	No	Yes	Yes	4	1
24	10	2	Yes	No	Yes	4	2
25	10	2	Yes	No	No	4	1
26	7	2	Yes	Yes	Yes	4	2
27	10	4	Yes	No	Yes	2	1
28	10	2	Yes	Yes	No	2	2
29	10	2	Yes	Yes	Yes	2	1
30	7	2	No	No	No	4	2
31	10	4	Yes	Yes	Yes	4	2
32	10	2	No	Yes	Yes	4	1

RunOrder	90 Complete	AveAP Utilization	MaxAP Utilization	MinAP Utilization	Ag Utilization	100% Complete	EmergencyAg
1	155	0.1438	0.1902	0.0881	0.5518	0	
2	155	0.4061	0.4061	N/A	0.6994	1	
3	150	0.0598	0.0951	0.0165	0.6301	1	3,5
4	230	0.24835	0.3143	0.1824	0.6576	1	4,6
5	125	0.1187	0.158	0.0794	0.771	0	
6	123	0.1092	0.1361	0.0922	0.6556	1	
7	130	0.112	0.158	0.0637	0.5358	0	
8	275	0.0947	0.11	0.0662	0.7991	0	4,7
9	130	0.1439	0.1439	N/A	0.729	0	
10	137	0.349	0.349	N/A	0.5326	0	
11	225	0.225	0.225	N/A	0.6762	1	1,2
12	149	0.22415	0.2361	0.2122	0.3855	1	
13	140	0.1329	0.1505	0.1303	0.3753	1	
14	122	0.1164	0.2163	0.0277	0.5785	1	
15	125	0.0893	0.1266	0.0314	0.7049	1	
16	130	0.1167	0.1654	0.0749	0.5358	0	
17	278	0.1307	0.139	0.1158	0.54	0	3,5
18	317	0.1952	0.3739	0.0629	0.6082	0	3,9
19	145	0.1081	0.1183	0.098	0.4976	0	9,10
20	223	0.0742	0.1352	0.0294	0.6852	1	3,5
21	155	0.165	0.2179	0.1121	0.5119	0	
22	250	0.2217	0.3797	0.0637	0.746	0	3,4
23	122	0.0878	0.1108	0.074	0.4202	1	
24	250	0.1441	0.1902	0.098	0.5783	0	3,9
25	248	0.2127	0.2448	0.1766	0.6389	0	1,7
26	205	0.1158	0.1158	N/A	0.674	1	1,7
27	320	0.1181	0.189	0.0562	0.5919	0	3,7
28	322	0.4194	0.4194	N/a	0.6069	0	2,7
29	340	0.1663	0.1663	N/A	0.5598	0	3,7
30	120	0.1497	0.1505	0.1489	0.6673	1	
31	238	0.1319	0.165	0.0951	0.5726	0	5,6
32	138	0.2064	0.2064	N/A	0.5359	0	

APPENDIX D. BURTSFIELD ELEMENTARY SCHOOL EXPERIMENTS RESULTS

Scenario		User 1			User 2			User 3		
# of Users	Room(s)	Rate	Room	Signal	Rate	Room	Signal	Rate	Room	Signal
1	1	1618		40 - 45						
2	1	763		40 - 45	750		40 - 45			
3	1	523		40 - 45	442		40 - 45	440		40 - 45
4	1	261		40 - 45	248		40 - 45	247		40 - 45
5	1	130		40 - 45	125		40 - 45	121		40 - 45
6	1	125		40 - 45	119		40 - 45	118		40 - 45
7	1	60		40 - 45	59		40 - 45	58		40 - 45
1	2	1646		55 - 60						
2	2	761		55 - 60	747		55 - 60			
3	2	429		55 - 60	429		55 - 60	423		55 - 60
4	2	266		55 - 60	259		55 - 60	257		55 - 60
5	2	127		55 - 60	120		55 - 60	117		55 - 60
6	2	104		55 - 60	86		55 - 60	81		55 - 60
7	2	55		55 - 60	49		55 - 60	47		55 - 60
1	3	1050		70 - 75						
2	3	667		70 - 75	534		70 - 75			
3	3	651		70 - 75	349		70 - 75	298		70 - 75
4	3	272		70 - 75	263		70 - 75	255		70 - 75
5	3	177		70 - 75	165		70 - 75	155		70 - 75
6	3	95		70 - 75	82		70 - 75	81		70 - 75
7	3	60		70 - 75	56		70 - 75	53		70 - 75
1	4	379		80 - 85						
2	4	281		80 - 85	247		80 - 85			
3	4	167		80 - 85	161		80 - 85	154		80 - 85
4	4	274		80 - 85	226		80 - 85	170		80 - 85
5	4	261		80 - 85	200		80 - 85	174		80 - 85
6	4	103		80 - 85	100		80 - 85	89		80 - 85
7	4	121		80 - 85	93		80 - 85	85		80 - 85
4	Mixed	542	1	40 - 45	410	2	55 - 60	372	3	70 - 75
5	Mixed	427	1	40 - 45	363	1	40 - 45	338	2	55 - 60

Scenario		User 4			User 5			User 6		User 7	
# of Users	Room(s)	Rate	Room	Signal	Rate	Room	Signal	Rate	Signal	Rate	Signal
1	1										
2	1										
3	1										
4	1	241		40 - 45							
5	1	120		40 - 45	120		40 - 45				
6	1	117		40 - 45	117		40 - 45	117	40 - 45		
7	1	57		40 - 45	56		40 - 45	56	40 - 45	55	40 - 45
1	2										
2	2										
3	2										
4	2	254		55 - 60							
5	2	117		55 - 60	117		55 - 60				
6	2	81		55 - 60	80		55 - 60	79	55 - 60		
7	2	47		55 - 60	45		55 - 60	45	55 - 60	45	55 - 60
1	3										
2	3										
3	3										
4	3	244		70 - 75							
5	3	153		70 - 75	116		70 - 75				
6	3	81		70 - 75	79		70 - 75	69	70 - 75		
7	3	52		70 - 75	52		70 - 75	50	70 - 75	50	70 - 75
1	4										
2	4										
3	4										
4	4	161		80 - 85							
5	4	154		80 - 85	136		80 - 85				
6	4	85		80 - 85	84		80 - 85	77	80 - 85		
7	4	84		80 - 85	79		80 - 85	73	80 - 85	70	80 - 85
4	Mixed	232	4	80 - 85							
5	Mixed	287	3	70 - 75		4	80 - 85				

Scenario		Average	Total Data Rate	Average Signal
# of Users	Room(s)	(kBps)	(kBps)	(dBm)
1	1	1618	1618	-42.5
2	1	756.5	1513	-42.5
3	1	468.3333333	1405	-42.5
4	1	249.25	997	-42.5
5	1	123.2	616	-42.5
6	1	118.8333333	713	-42.5
7	1	57.28571429	401	-42.5
1	2	1646	1646	-57.5
2	2	754	1508	-57.5
3	2	427	1281	-57.5
4	2	259	1036	-57.5
5	2	119.6	598	-57.5
6	2	85.16666667	511	-57.5
7	2	47.57142857	333	-57.5
1	3	1050	1050	-72.5
2	3	600.5	1201	-72.5
3	3	432.6666667	1298	-72.5
4	3	258.5	1034	-72.5
5	3	153.2	766	-72.5
6	3	81.16666667	487	-72.5
7	3	53.28571429	373	-72.5
1	4	379	379	-82.5
2	4	264	528	-82.5
3	4	160.6666667	482	-82.5
4	4	207.75	831	-82.5
5	4	185	925	-82.5
6	4	89.66666667	538	-82.5
7	4	86.42857143	605	-82.5
4	Mixed	389	1556	-63.75
5	Mixed	353.75	1415	-59.5