ABSTRACT

Title of document: REAL-TIME DECISION AID DISPLAY

Jennifer Au, Anthony Bonomo, Laura Freyman
Brian Kwong, Benjamin Li, Jessica Lieberman
Levon Mkrtchyan, Michael Price, Andrew Skoda
Mary Tellers, Andrew Tomaschko, Johnny Wu

Directed by: Dr. Frederick W. Mowrer, Associate Professor Emeritus
Department of Fire Protection Engineering

Fire sensor systems effectively monitor the state of the building, detect fire, and alert occupants in the event of an emergency. However, fire sensor technology is limited in its ability to convey information to firefighters. Even though all of the necessary information can be obtained through Fire Annunciator Control Panels (FACPs), it is difficult to use them to track the progression of fire.

We designed and prototyped a decision aid system to illustrate our approach to this problem. Our goal was to create a tactical decision aid display that can present building information through an intuitive interface in real time. We used previous research on the information needs of firefighters in designing the interface. Our key insight was to use a floor plan with a sensor information overlay to organize information. We implemented a prototype that interfaces with FACPs using existing facilities systems management communication protocols.

REAL-TIME DECISION AID DISPLAY

by

Team Future Firefighting Advancements (FFA)
Jennifer Au, Anthony Bonomo, Laura Freyman
Brian Kwong, Benjamin Li, Jessica Lieberman
Levon Mkrtchyan, Michael Price, Andrew Skoda
Mary Tellers, Andrew Tomaschko, Johnny Wu

Thesis submitted in partial fulfillment of the
Gemstone Program, University of Maryland, 2011

Advisory Committee:
Dr. Frederick W. Mowrer, Chair
Mr. Millard B. Holmes
Dr. James A. Milke, Ph.D. P.E.
Dr. James Purtilo
Dr. Peter B. Sunderland
Mr. Scott Wood

Acknowledgments

We would like to thank SFPE ESF for their generous grant and the opportunity to present at the SFPE conference. In addition, on campus we'd like to thank David Doucette, Jim Robinson, and Olga Zeller for their time, patience, and helpfulness throughout the project, as well as other facilities management staff who lent their time. We are also grateful to Millard Holmes and SimplexGrinnell for the donation of hardware and the expertise to help us configure it for our project. We would also like to thank Honeywell for their donation of hardware as well. We also appreciate the Gemstone program's support throughout this four-year process. Finally, we would like to thank the Fire Protection Engineering Department for the space they gave us for meetings and lab work.

Table of Contents

List of Figures

List of Abbreviations

Antibody Identification Assistant (AIDA)
The American Society of Heating, Refrigeration, and Air-conditioning Engineers (ASHRAE)
Building automation system (BAS)
Building information services and control system (BISACS)
Building services interface (BSI)
Chemical, biological, radiation (CBR)
Command Decision Support Interface (CODSI)
Comma-separated value (CSV)
Decision support system (DSS)
Emergency first responders (EFRs)
Fire Alarm Control Panel (FACP)
Fire Dynamics Simulator (FDS)
Future Firefighting Advancements (FFA)
Geographic information system (GIS)
Gallons per minute (gpm)
Global positioning system (GPS)
Graphical user interface (GUI)
Hazardous materials (hazmats)
Human-computer interaction (HCI)
Incident commanders (ICs)
James M. Patterson (JMP)
Maryland Fire and Rescue Institute (MFRI)
National Electrical Manufacturers Association (NEMA)
National Fire Protection Association (NFPA)
National Institute of Standards and Technology (NIST)
Occupational Safety and Health Administration (OSHA)
Personal Identity Verification (PIV)
Regional Crime Analysis Program (RECAP)
Radio-frequency identification (RFID)
Records management system (RMS)
Self-contained breathing apparatus (SCBA)
Sensor Driven Fire Model (SDFM)
Society of Fire Protection Engineers (SFPE)
SFPE Education and Scientific Foundation (SFPE ESF)
Signaling line circuit (SLC)
Ultra wideband (UWB)
Very Early Smoke Detection Apparatus (VESDA)
Extensible markup language (XML)

Chapter 1

Introduction

For decades, building sensor manufacturers have pursued the goal of detecting emergency situations earlier while reducing the occurrence of nuisance alarms. In this respect, great progress has been made. Modern building sensors are much more reliable and accurate than their predecessors. At the same time, the goal of using building sensors as an aid to emergency first responders (EFRs) has been given less attention. Building sensor data could provide important information, and history has shown that having that data in a timely manner could aid in emergency response.

The Pang Seattle Warehouse fire is an example of how having information immediately can change the outcome of an emergency. The fire started on July 5, 1995 in the warehouse as a result of arson, and firefighters were deployed inside the building in order to suppress the flames. The firefighters on the first floor were able to successfully put out the fire in that area and believed that they had the emergency under control. However, the fire raged on in the basement, with another dispatched group of firefighters trying to suppress it. The firefighters on the first level did not communicate with the firefighters on the lower level and vice versa, and neither communicated with incident command. Consequently, neither the incident commander nor the firefighters on the first floor had any idea that there was a raging fire in the basement. There was a sudden collapse of the first floor of the warehouse,

resulting in the deaths of four firefighters (Routley, 1995).

If the information of where the fire was located was immediately available, the incident commander would have been much more informed when formulating an attack plan for the fire. The need for more pre-planning data, as well as more data communication during an emergency was made very clear in this incident (Thiel, 1999).

The First Interstate Bank building fire in Los Angeles on May 4, 1988 is another example of an incident where more information could have been provided to the personnel involved. At the time, this incident was considered to be one of the most devastating fires in history. The blaze destroyed four floors and damaged a fifth, resulting in over $50 million in property damage. One person was killed, and thirty-five occupants and fourteen firefighters were injured during the incident.

The fire, which is believed to have been caused by an electrical source, originated on the twelfth floor of the building. Shortly after, a manual pull station was activated and silenced by security personnel within minutes; they believed the alarm was the result of repair work being performed in the building. A series of smoke detector alarms were activated next, but each one was also reset by security personnel. Eventually, an employee went to twelfth floor to determine the source of the alarms, only to be engulfed by the flames in the lobby. The multiple alarm resets delayed the notification of the fire department, leading to a much larger fire by the time firefighters arrived. This scenario is one where a display of the alarm history may have been helpful to security personnel, who were unable to piece together the multiple alarms and recognize the extent of the fire.

In addition, the main fire pumps were shut down, resulting in poor water pressure in the first minutes of response. Firefighters eventually found the building fire pumps, mainly because the sprinkler installation supervisor was present and able to inform the incident commander of the existence of these pumps (Routley, 1998). In this case, critical static information could have been made available to the incident commander prior to arriving on the scene, which would have prevented any delays in response. Again, this highlights the need for more pre-planning data.

The MGM Grand Hotel fire took place on November 21, 1980. The fire began early in the morning at approximately 7:15 AM. The blaze began as a result of an electrical fault that had been smoldering in the walls of a deli on the first floor of the hotel. The fire soon caused flashover in the deli, causing a fireball to expand rapidly throughout the first floor casino at about nineteen feet per second (Puit, 2000). The casino was full of fuel for the fire, including a highly flammable adhesive used to attach ceiling tiles. The fire had engulfed the first floor by the time the first responders arrived to the scene four minutes after the start of the fire.

In Figure 1.1, copious amounts of smoke can be seen pouring from the building at the top and some of the sides. However, that smoke is the limit of visual information that can be gained from an external view. While it may appear that the fire has spread throughout the entire building by looking at the smoke, the fire is only on the first floor. The massive amount of smoke generated by the fire quickly rose through ventilation shafts, moved through the hallways in the top floor and escaped out the windows. Because this smoke movement is very dependent on environmental factors, it becomes hard to visually ascertain exactly where a fire is

within a building just from seeing the smoke pour out of it.

Figure 1.1: Smoke can be seen pouring out of the top of the building as the fire continues to burn on the first floor.

There were many critical mistakes made that led to the severity of the fire at the MGM Grand Hotel, including a lack of fire detectors of any form and a lack of sprinklers. Either of these fire safety measures could have provided valuable data to personnel and emergency first responders. In the event that fire spread could not be limited by sprinklers, data from building sensors could have provided information as to where the fire was spreading within the building and how to further plan for effective suppression efforts.

Currently, sensor data is typically presented sequentially and out of context, making the data less useful for decision making. Most modern building sensor systems still use a text-only display, which is hard to interpret and takes too much

effort to access and understand the information one is seeking. In order to assess the situation in a building, it is necessary to contextualize the sensor data, that is it must be presented in the context of related data. For example, the state of any one sensor may not say much about the progression of smoke in the building but the states of all of the smoke detectors on a floor does.

There are a number of barriers to contextualizing building data. Each sensor manufacturer uses their own protocols for communication between sensors and the annunciator panels. This lack of standardization makes it difficult to create a product that can work with systems from different manufacturers. Another problem is that, frequently, the sensor data is not easily obtained from the annunciator panel. Retrieving analog sensor values such as room temperature and smoke obscuration can be more difficult than simply obtaining sensor trouble or alarm states. The extent of these problems was revealed during work on our project, but we did not attempt to address them since there are known solutions.

Our team explored the approach of data contextualization through an emergency visualization system. Overlaying data on a floorplan is the simplest way to contextualize data. We focused on making the system useful as a decision support tool during an emergency. We limited the prototype to only work with fire sensor systems due to resource limitations, but intend the concept to be easily extensible to all building sensor systems. In this thesis we describe the design, prototyping, and evaluation of our system.

Our intent is for this project to serve as a proof-of-concept. We want to show that building sensor data can be more useful when contextualized. The decision

support tool we developed demonstrates our approach to visualizing the state of a building. Finally, we wanted to find out whether modern fire sensor systems can be incorporated into such a visualization system without any changes to sensor technology. To satisfy this goal we focused on the design stages of the process but left the implementation as a prototype rather than as a finished product.

We are aware that in order to put a building state visualization system to practical use, a number of infrastructure questions must be resolved. How are building floor plans stored and made available for use in our system? Who annotates the floor plans with sensor locations and when? How is the sensor data made externally available without compromising security? How do EFRs incorporate use of the system into their existing practices? It is not the aim of this thesis to address any of these questions. These questions can be answered once a fully functional decision support tool has been realized.

Chapter 2

Literature Review

2.1 Fire Emergency Responder Standard Operating Procedures

2.1.1 Size-Up

In order to effectively combat any fire emergency, firefighters are required to obtain and process a great deal of information for every fire situation. According to the *Fire Officer's Handbook of Tactics* by John Norman, there is a thirteen-point outline used by fire chiefs that covers a majority of fireground considerations, conveniently summed up by the mnemonic COAL WAS WEALTH: Construction, Occupancy, Apparatus and manpower, Life hazard, Water supply, Auxiliary appliances, Street conditions, Weather, Exposures, Area and height, Location and extent of fire, Time, and Hazardous materials. These points are interrelated and each is not simply considered separately when sizing up a fire situation.

Norman establishes that life hazard is the most important factor in determining fire operations and tactics. Life hazard comes in two forms - civilians and firefighters. Tactics employed by firefighters will usually be more aggressive if there are civilians in need of rescue, but incident commanders (ICs) will not risk the safety of firefighters if there is no significant civilian life hazard. Risk to civilian life is best prevented before an incident occurs by imposing occupancy restrictions, specifying

fire doors and exits, and installing an automatic sprinkler system throughout the building.

Occupancy has considerable bearing on life hazard, and is usually dependent on the time of day and the type of building. For example, hospitals and residential buildings create a high life hazard at all hours of the day, while storage warehouses pose a uniformly low life hazard, while the life hazard of a school varies greatly with the time of day. The time of year affects fire operations as well. Certain times of year, such as the holiday season or hunting and fishing seasons in some regions, can affect the amount of manpower available to ICs, especially in volunteer departments (Norman, 2005).

Another important aspect of time in fire operations is the elapsed time of a fire incident. ICs have some rules of thumb to gauge how long a fire has been burning before their arrival to the fire situation, such as looking to see if fire is venting out of windows, but these are typically inexact and require very experienced ICs to implement these rules based on any given situation. ICs also have mechanisms in place to track the time spent fighting a fire, such as the dispatcher requesting a status report from the IC five minutes after arrival and every ten minutes for the first hour of the incident (Norman, 2005).

Construction of the building is another important factor in determining the fire tactics utilized by ICs. Buildings are classified into five general types: fire-resistive, noncombustible, ordinary construction, heavy timber, and wood frame. These classifications are based on four criteria: the degree of compartmentation a building provides, the degree to which a building contributes to the fire load, the

number of hidden voids in the building, and the ability of the building to resist collapse (Norman, 2005).

Area and height of a building are concerns during size-up for several reasons. Most notably, they indicate the maximum possible fire area. Building height can also give clues as to what kind of construction the building is and what auxiliary appliances are available. For instance, if the building is over a certain height, it might be required by law to be built of Class I fire-resistive construction or have a sprinkler or standpipe system (Norman, 2005).

The physical location and extent of the fire has an influence on the tactics used to control it. In general, the lower a fire is in a building, the more problematic the incident due to more potential for vertical spread. Other locations that create firefighting problems include the top floors of buildings with ordinary brick and wood-frame constructions; fires below grade, such as in a cellar, a tunnel, or below deck on a ship; and fires beyond the reach of ladders (Norman, 2005). Determining the extent of a fire can be difficult, as certain factors such as central air conditioning and moving elevators can cause smoke to move in ways it otherwise would not.

ICs must also identify and protect exposures, or areas around the building that could create additional life and property hazards. Firefighters often use a numbering system to identify exposures, labeling the sides of the building as one through four, starting at the front of the building and proceeding clockwise around the perimeter (Norman, 2005).

The IC must then identify the resources available at the scene and start to determine a plan of action for attacking the incident. The IC must identify the

apparatus and manpower at his disposal and determine how to utilize them. Closely related to these factors is the available water supply, which not only includes the sources of water but also the devices and manpower used to transport the water to extinguish the fire such as hoses and pumpers. ICs must also determine how much water is needed to put out the fire, which is usually measured in gallons per minute (gpm) based on the square footage of the fire. This can range from 10 gpm for 100 square feet of light fire load to 50 gpm for 100 square feet of heavy fire load. ICs must also determine the presence and status of auxiliary appliances such as sprinkler systems, standpipe systems, and foam suppression systems (Norman, 2005).

Several other factors should be accounted for when determining firefighting tactics. Weather conditions can affect an IC's strategy; high temperatures and humidity will fatigue firefighters more quickly, while high winds might make ventilation of a fire impractical. Certain street conditions such as construction and illegally parked cars can hamper maneuvering and apparatus placement and utilization. Hazardous materials (hazmats) present in and around a building present a variety of problems, ranging from health hazards to acceleration of the fire extension. Identifying and attacking all of the above factors make the IC's job very challenging (Norman, 2005).

2.1.2 Operations - High-Rise Buildings

The basic strategic plan for a high-rise fire incident starts with determining and verifying the specific fire floor before committing hose lines. This information

is critical but difficult to ascertain, as the initial information is often vague, such as smoke seen from the fifth, sixth, and seventh floors. Next, firefighters must begin a controlled evacuation, evacuating those in immediate danger first, preventing a panicked exit by those not endangered, and searching the fire floor and all floors above the fire. Firefighters must then gain control of the building systems, including HVAC, elevators, public announcement, and other vital systems, and proceed to confine and extinguish the fire (Norman, 2005).

A tremendous amount of resources and experience is required to fight a high-rise building fire. In New York City, the signal of a possible working fire in a high-rise calls for a deputy chief and four battalion chiefs to respond, and a second alarm calls for another battalion chief, another deputy chief, and a senior staff chief. The command post is usually set up in the lobby of the high-rise, while the operations officer, usually one of the first responding chiefs other than the IC, goes to the floor below the fire to assume direct command of the fire operations. The operations officer must communicate constantly with the command post, keep the line of attack moving forward and maintain resources (Norman, 2005).

Firefighters must take steps to remove the smoke and heat buildup caused by a fire, which is usually done through ventilation of the building. One way this can be achieved is through vertical ventilation, or opening the top of the building using a stairwell or elevator shaft. The IC should send two EFRs to the top of the building to open the stairwell door on the roof. The EFRs should check with the IC to see how the fire responds to this door opening. If favorable, the door to the fire area should then be opened, after which smoke and heat will be drawn to the roof of the

building, with the stairwell acting as a chimney (Norman, 2005).

Another way to vent smoke, heat and gas out of the building is horizontal ventilation. This technique involves breaking windows on the fire floor to allow smoke to exit out the sides of the building. This tactic is much more complex than vertical ventilation because if done improperly or in the wrong weather conditions, it can cause smoke to blow into the building and upward to the top floors. This tactic also has the added danger of glass falling to the street outside, which can injure bystanders and fire personnel as well as sever hose lines. The IC should only authorize horizontal ventilation after careful consideration (Norman, 2005).

After ventilation of the fire, gaining access to the fire area is the next fire-fighting concern. For large high-rises, using the elevators is considered a "necessary evil" because a firefighter will not be very effective at fire suppression after walking up over twenty stories carrying forcible entry tools, hoses, and other gear. Certain safety precautions must be taken when using elevators in a fire situation, which include attempting to accurately determine the fire floor, ensuring each team using the elevator has been documented and properly equipped, using firemen's service elevators if possible, pressing the call cancel button when boarding to eliminate any other selections made, and being prepared to don facemasks when arriving at the destination. If elevators become unavailable, extra supplies must be transported manually to the operations post. One firefighter would be assigned to transport equipment for every two floors from ground floor to the fire floor. If there is not enough fire personnel to transport equipment, non-fire personnel such as police can be used as long as they are not put in threatening situations (Norman, 2005).

The first-arriving units are charged with locating the fire, determining size and likely paths of travel, and redeploying to prevent extension. The attack crews on the fire floor, however, will have a difficult time putting out the fire, as it might be challenging to attack the base of the fire due to obstructions such as desks and partitions. There is also an issue of having enough pressure and water flow from standpipes that are so high above the ground floor. A fully involved floor that is over 30,000 square feet requires a minimum of 3,000 gpm to extinguish a fire at a light fire load, but the NFPA 14 standard's 1993 revision requires only 1,250 gpm of water flow from standpipes. The next step would be to get above the fire floor and contain the fire until it consumes enough fuel to be fought manually. The tactics used for this process are considered last resort measures to get water onto a fire that is otherwise untenable (Norman, 2005).

The tactics outlined above are used by essentially all firefighters and ICs across the country. Despite the advancements in fire protection and building construction made in recent years, the possibility of a high-rise building disaster remains a very real possibility (Norman, 2005).

2.2 NFPA Standards

The National Fire Protection Association (NFPA) is responsible for creating and advocating three hundred consensus codes and standards. In general, the purpose of these standards is to decrease the potential risks and negative effects due to fire through the establishment of criteria for building, processing, design, ser-

vice, and installation (National Fire Protection Association, 2010). The NFPA uses committees of volunteers from the fire protection industry to create and update these standards. While wording from NFPA standards have been incorporated into particular federal or state Occupational Safety and Health Administration (OSHA) regulations, compliance with most of the three hundred codes and standards is voluntary unless adopted into law (National Volunteer Fire Council, n.d.). Three NFPA standards, NFPA 72, NFPA 170 and NFPA 1620, are of particular significance to our research project.

NFPA 72 contains the National Fire Alarm and Signaling Code, which provides standards for the installation and use of fire alarm systems. Requirements for initiating devices (smoke and heat detectors, radiant energy sensing fire detectors, gas detectors, water flow alarm-initiating devices, and other fire detectors) and notification appliances are thoroughly outlined. The NEMA (National Electrical Manufacturers Association) SB 30 Fire Service Annunciator and Interface standards included in Annex E are especially important to our research. NEMA SB 30 provides a cross-industry standard for the symbols and other elements involved in interface systems related to fire protection (*NFPA 72: National Fire Alarm and Signaling Code*, 2010). Our research team plans to implement the symbols suggested by NEMA SB 30 whenever possible.

NFPA 170 contains the Standard for Fire Safety and Emergency Symbols, which presents a standard set of symbols to be used in representing fire safety, emergency and associated hazards (*NFPA 170: Standard for Fire Safety and Emergency Symbols*, 2009). Included are symbols for general use, for use by the fire service, for

use in architectural and engineering drawings and for use in pre-incident planning sketches. We plan to use the symbols contained in NFPA 170 to augment Annex E of NFPA 72, as NFPA 170 includes symbols for elements of a fire emergency not addressed by the NEMA SB 30 standard.

NFPA 1620, the Standard for Pre-Incident Planning, establishes the essential features of any pre-planning system. It addresses the method of pre-planning, occupancy, fire protection and suppression systems, specific hazards, emergency operations and plan testing and maintenance. It also addresses the site evaluation and other physical considerations. Some essential aspects of the building assessment portion are construction, building management, external site conditions, and internal and external security measures (*NFPA 1620: Standard for pre-incident planning*, 2010). NFPA 1620 details exactly what information about a building needs to be included in pre-planning materials. A standard like NFPA 1620 allows for cooperation between multiple departments on the same event.

2.3 BACnet

In order to facilitate communications between our system and third-party peripherals, our team has decided to focus on using the BACnet protocol to receive and interpret data output from building information systems. The BACnet protocol was developed by ASHRAE, the American Society of Heating, Refrigeration, and Air-conditioning Engineers in 1995 as a communications protocol to aid cross-platform messaging between disparate building information systems(Bushby, 1996). It was

internationally recognized and formalized by the ISO standard 16484-5 in 2003. Prior to the development of BACnet, building information systems exclusively used proprietary communication protocols, and they were incapable of directly communicating or working with each other. Without a universal protocol like BACnet, systems reading output from multiple building information systems, such as ours, would be extremely difficult, if not impossible, to implement, for both technical and legal reasons. The different protocols would have to be handled individually in the code of the program to enable it to understand the output from the building information systems, requiring a great deal of space, time, and effort to implement an interpreter for each protocol. In addition, since these protocols are proprietary, there could be legal issues associated with making such an interpreter, since it could involve reverse-engineering these protocols to attain a sufficient understanding of them (Newman, 2000).

BACnet is an object-oriented protocol with a client-server communications paradigm. Sensors within a BACnet-compatible system are represented as "devices." Every device is made up of one or more "objects," which encapsulate a specific function of that device, such as binary input or analog output. These objects contain properties which represent the state of that function, such as whether it is in alarm, or the value of an analog sensor. These properties are accessed by the BACnet server through "services," which query the objects about the state of their properties. Every BACnet object can respond meaningfully to a subset of the available services, with additional services supported as required (Bushby, 1996). Sensors and systems that cannot natively communicate through BACnet can utilize

a BACnet gateway, which converts the proprietary data output into a format that is legible by BACnet. This BACnet gateway must be provided by the manufacturer of the sensor.

Our team decided to use BACnet because it is relatively widely supported among building information systems in general. Much of our operating test equipment supports BACnet as an output protocol, which allows our system to gather data from them simultaneously and symmetrically. By comparison, output using the RS-232 ports requires a specialized interpreter to be coded for each individual system; without a common protocol like BACnet, there can be no generalization of data interpretation from these systems. Also, since BACnet was originally designed to be a universal, general protocol, it is not limited to use with fire detection systems; it has the capability to work with a diverse array of products, such as air-conditioning systems, smart elevators, and other building sensors. This wide applicability theoretically enables our decision support tool to receive and use data from any BACnet-compatible device present within a building, even if it is not directly related to fire detection or prevention (Bushby, 1996).

2.4 Building Tactical Information Project

The ongoing Building Tactical Information project being conducted by the National Institute of Standards and Technology (NIST) is concerned with the development of technology that aims to make real-time building information available to EFRs (Holmberg, Treado, & Reed, 2006). The project has four main objectives:

1) determining which data would be most useful to EFRs in emergency response situations, 2) developing a standard method for dispersing the data collected by buildings to EFRs, 3) demonstrating the effectiveness of the technology proposed by the project and 4) addressing the security issues in the system proposed.

In order to make a system of complexity and sophistication that would be capable of the goals of our project, it must be determined what information is key to emergency response. To make a complex system for a target group of people such as EFRs, it is a necessary step of product development to talk to the end users to determine their needs and specifications. To this end, NIST held a workshop for emergency responders to define what information they needed during a building emergency to aid them in making decisions. The results of this workshop are presented in the paper entitled "Workshop to Define Information Needed for Emergency First Responders During Building Emergencies," by Jones, Holmberg, Davis, Evans, Bushby, and Reed.

It is important for our project to have the feedback and information generated by users from NIST's workshop because it was conducted on a scale that would not be feasible for a Gemstone Team. The following paragraphs introduce information found from the workshop that directly pertains to specifications we need to address.

First, the participants in the workshop identified two essential categories of information based on the time frame typically associated with emergency response: "En-Route," defined as the "first five minutes" and "On the Scene," after the EFRs arrive at the site of the emergency. These two time frames represent the important phases of emergency response in which decisions are made. Second, the workshop

identified three major areas of information important to creating a system: static information, dynamic information, and the display of information. Static information is defined as information available before an emergency occurs. This information includes building plans, sensor layouts, and pre-established escape routes. Also included are the more specific qualities of a structure such as where access points of the building are, locations of elevators, nearby hazard conditions, and the types of power systems that are running in the building. Dynamic information is defined as the real-time information that would be received from the sensor systems. This information includes direct sensor readings from building fire panels as well as information from decision support tools which take in and analyze data in order to give a useful conclusion such as fire location and spread (Jones et al., 2005). All of this information would have to be managed in a simple, yet comprehensive display.

For the En-Route display, to be used during the first five minutes of an emergency when responders are on the way to the scene, a comprehensive list of static and dynamic information to be displayed was decided on by the EFRs who took part in the workshop. However, it was noted that given the circumstances of the first five minutes, with the commander and other first responders all en-route to the incident, only so much information could be effectively communicated due to difficulties of reading from computers while in motion. With that in mind, a list of static information was generated; aspects of this list that are relevant to our project are presented below:

- Building condition (let burn, unsafe to enter, dangerous roof, sprinklered and other suppression systems)

- Building type (single family, commercial, gas storage, school)
- Building style (one story, two story, n story, auditorium, sublevels, etc.)
- Building construction (type I, II, III, IV or V; fire resistive, noncombustible or limited combustible, ordinary, heavy timber, or wood frame)
- Roof construction (light weight metal or wood trusses)
- Hazardous materials or unusual hazards (above ground propane tank, gas lines, chemicals, etc.)
- Location of fire hydrants on map with building outline, nonstandard thread sizes included
- Location of fire department hookups for sprinkler system/standpipes
- Other sources of water nearby
- Location of staging areas and entrances and exits to building
- History of location in case fire stages before police arrive
- Routing information for emergency equipment to reach the building in case of construction

(Jones et al., 2005)

Additionally, the dynamic information generated includes:

- Confidence in the incident being real (based on number of sensors in alarm and/or calculated fire size)
- Approximate location of fire within building
- Fire size and duration
- Sprinklers are flowing/no sprinklers or other working systems
- Fire growth (fast, medium, or slow)
- CBR (chemical, biological, radiation) sensors present and in alarm
- Police on the scene
- Presence of occupants in the building
- Stairwell smoke/heat conditions for positioning
- Standpipes to use to get to the fire

- Exposures

(Jones et al., 2005)

The above points of information that would aid in tactical decisions will most likely be available in large commercial structures that have the infrastructure to supply it. In smaller structures, only some of the information is likely to be available. However, any and all of the above information would aid emergency response. Normally, none of the above information can be obtained en-route and in the best-case scenario, only fractions of it are obtained on scene without the aid of technology.

Once at the scene, incident commanders need more information to make their decisions. Information crucial to decision making lies in the floor plans of a building. Ideally, these floor plans would be easily viewable, with relevant information shown in the correct location with respect to the floor being viewed. "The floor plan (static data) would include layers/overlays that would allow the incident commander to locate:

- Doors, windows (with types and which can be used for egress), stairwell risers, fire walls (with ratings and area separation), roof access, fire sensors.
- Security sensors, closed circuit TV cameras, occupancy sensors, security control room.
- Fire alarm panel and remote annunciator panels.
- Utility shutoff.
- Building generator (with indication of what it powers).
- Building system controls (HVAC, smoke control, others), areas covered, special operating systems, and which ones should and should not be used by the responders.
- Evacuation quality elevators, floors served, and location of elevator overrides and how to control.
- Convenience stairs/evacuation stairs.
- Areas (zone boundaries) protected by sprinklers or other devices.
- Vertical openings.

- Extremely valuable materials.

(Jones et al., 2005)

Currently, the ONYX FirstVision fire panel is being used by NIST to incorporate this kind of information overlay. It clearly labels all floors, all sensor locations, important points of entry and exit, and other aspects of a building useful to emergency responders. Where the FirstVision panel excels is in the processing of dynamic information on the scene, as determined by the workshop as the following:

- Location of fire detectors in alarm
- Location of CBR sensors in alarm
- Location and size of fire(s)
- Duration of the fire(s)
- Location and condition of smoke
- Presence of smoke in elevator shafts or stairwells
- Identification of activation of sprinklers or other devices
- Location of elevators used during the incident
- Location of people in need of rescue (911 calls or visual sightings)
- Warnings of structural collapse based on material type, fire location, fire size and duration
- Location of operational elevators
- Alarm, occupant, and system histories of the building

(Jones et al., 2005)

The kind of system that could take into account all of this information cannot be limited to just fire sensor technology though the most pertinent information about the movement and severity of the fire comes from such sensors; it would require the incorporation of security system data, HVAC information, facilities management information, such as elevator location, and tracking systems to monitor the location

of EFRs within a building. A normal fire sensor system would be inadequate on its own. However, fire sensor systems have an important component, the fire panel annunciator, which, if sophisticated enough, can be used to process and analyze information, and, given the right inputs, incorporate other system technologies by using a common communication protocol such as BACnet.

Finally, an incident commander would need to know information about what is going on outside the building, such as who has responded to the emergency (medical personnel, police, firefighters, etc.), what apparatus has been dispatched and where personnel are located, in order to make fully educated decisions. Static information regarding what is going on outside the building to be included on the plot of the floor plan consists of:

- Building location with street designations
- Location of fire fighting obstacles such as street widths, overhead clearance and elevations
- Location of underground pipelines and other utilities
- Name and phone numbers of building owners and managers
- Name and phone numbers of utility contact people
- Location of police line necessary to isolate the incident
- Indicated runoff or water table problems
- Helicopter landing areas
- Evacuation routes

Dynamic information displayed on the floor plan would include:

- Location of responding units (fire, police, and EMS)
- Location of units responding but not yet on scene
- Hospital availability
- Helicopter availability
- Hazmat response

- Location of police line necessary to isolate the incident
- Location of triage or evacuation area
- Suggested hazard perimeter
- Local weather conditions and predicted spread directions
- Wind direction and velocity

The data that could be provided and used by EFRs is extensive and the need for a system that can distribute all of this information is very real. The needs for EFRs in a building emergency determined by NIST's workshop serve as the backbone for the standards set for our system. There are, however, concerns that must be considered in designing such a system. Specifically, standardization is a key issue. EFRs expressed concerns with usability of a system, mentioning a need for every system control display to be the same, so there is no new learning curve from building to building, incident to incident. Also, the system has to be fully functional, not partially, as the Fire Service will not adopt a new, questionable system. The Fire Service cannot be expected to pay for new technology, thus the system must also be affordable enough to be installed and used in real-time when buildings are first constructed (Jones et al., 2005).

Further literature published by NIST addresses the remaining objectives of the Building Tactical Information project. To address the second objective, NIST proposes a building information server to provide access to tactical information in real-time to EFRs (Holmberg et al., 2006). This tactical information includes static building information on a given building's construction, occupancy type, floor plans, and system schematics, as well as dynamic information collected including alarms and changes in a building's status as they occur. According to the project, the desirable features for such a building information server system include the ability to:

- Forward alarms and status messages as they occur

- Deliver descriptive building information
- Allow only authorized access
- Allow authentication by multiple accessing connections
- Allow asynchronous access
- Present information in a common format
- Minimize the volume of transmitted information
- Maximize the use of layered communications protocols
- Continue to provide information in the event of partial system failure

The project also proposes the use of extensible markup language (XML) in messages received and dispatched by the server in addition to the implementation of the Internet TCP/IP suite as the base communication protocol (Holmberg et al., 2006).

In addition to the implementation of a building information server, the project also advocates the use of the Sensor Driven Fire Model (SDFM) support system. The SDFM is a prototype decision support system that converts sensor signals to predictions and issues warnings based on these predictions. In order to present the information collected by the building information server and interpreted by the SDFM system, the project proposes both en-route and on-scene information presentation screens. The en-route screen provides information about the area in the immediate vicinity of the building while the on-scene screen presents detailed information about the building's interior (Holmberg et al., 2006).

In February 2005, a demonstration of much of the technology described above was held at NIST, accomplishing the project's third objective. The demonstration was documented on video. The presentation included demonstrations of building sensor data and fire modeling data formatted in XML schema, the transfer of building data to the building information server, and the use of the proposed software client and interface software. At the demonstration, it was determined that EFRs

often are not aware what information is available from the control systems contained in buildings; an educational video was produced to address this deficiency (Holmberg et al., 2006).

To accomplish the fourth objective of the Building Tactical Information Project, NIST outlines a method for allowing only secure access to building information for use in emergency response. The methodology described would permit public safety officials remote access to real-time information. In addition, it would also grant access to EFRs en-route (Treado, Vinh, Holmberg, & Galler, 2007).

As proposed, dynamic information would be provided to emergency personnel through a building automation system, which coordinates the activities of HVAC, lighting and access controllers, and fire detection and security systems. The sensor data and video streams provided by these systems would be delivered to EFRs through enhancements to the BACnet building automation system communication protocol. Static information such as floor plans and equipment schematics would already be available to public safety officials in building information models (Treado et al., 2007).

To access the information being provided by the building automation system, a secure proof-of-identity credential, such as a federal PIV (Personal Identity Verification) would be required. The right to varying levels of the hierarchical database structure would be governed by factors such as identity, jurisdiction, duty status, incident need, and chain-of-command. The secure connection to these databases would be facilitated by a building information services and control system (BISACS), a system lending itself to standardized protocols and secure communication. The BISACS would be centrally located in a given building and credential reader interfaces at access nodes would permit authenticated internal and external user access. A secure web-based link between the BISACS and the BACnet building automation system (BAS) using a building services interface (BSI) would allow information and

resources to be exchanged and shared securely. To maintain security, information would only be permitted to travel in one direction, from BAS to BISACS (Treado et al., 2007).

The system proposed by the Building Tactical Information Project provided a sound foundation for the design and implementation of our team's own prototype decision support tool. The prototype our team created aims to satisfy many of the same design criteria. The background information utilized, data collected, and results to date of the Building Tactical Information Project were invaluable to the design of our decision support tool.

2.5 FireGrid

The FireGrid project, lead by the School of Engineering and Electronics at the University of Edinburgh, is attempting to achieve the same goal we sought to accomplish with our research project: provide firefighters with valuable building sensor data in a usable format in real-time. However, unlike our project, the FireGrid consortium has placed heavy emphasis on the modeling of a fire's progression to predict its future course using high performance computers, grid-enabled distributed computing capabilities, computational fluid dynamics models, and finite element structural models (Berry et al., 2005). According to Potter and Wickler (2008), incident commanders would use one of two command and control interfaces to send a request concerning the future development of a particular fire emergency to the Query Manager which would search for available models that can answer the query, run the most appropriate model to obtain the information requested, and return this information to the incident commander. While the FireGrid consortium's attempt to integrate predictive modeling into a command and control decision support tool for firefighters is certainly compelling, we feel that it is rather far-sighted and thus did not find the work they are doing to be particularly helpful to the advancement

of our project.

2.6 Current Building Technology

2.6.1 Overview

Most of the fire detection systems on the market feature a fire alarm control panel that is permanently installed near the entrance of a building, and uses a loop to monitor and control a number of devices located throughout the building. These devices are often point-addressable smoke detectors, thermal sensors, flow switches, or other detectors that can relay its current status back to the control panel. Some detectors are intelligent in that they have some remote programming capabilities that allow a user to specify alarm profiles into the detector. Device information is usually displayed on the control panel itself, whose display can be as simple as a set of LED lights that indicate a zone in alarm to a more sophisticated one, such as a multi-character display with icon and graphics support. History logs are often available on control panels and are readily available to technicians and firefighters who respond to an alarm, trouble, or supervisory signal.

Many manufacturers offer methods to monitor these systems through a network. A network can connect a number of fire detection systems together, but the diversity of these systems, as well as the modularity of the control panels, are usually limited by the specific manufacturer. In some cases, manufacturers provide separately-sold devices or modules to allow control panels to use BACnet, a communications protocol. BACnet has the potential to allow different brands of systems to communicate with each other.

Some manufacturers also offer monitoring from remote locations. This usually comes in the form of a computer workstation that can provide a visual representation of an emergency. These workstations typically feature a floor plan of the building

that contains multiple icons that represent different types of signals and detectors. The location of these workstations is fixed, since the computer is usually installed in a specific area.

Other types of building technology are HVAC and other monitoring systems. HVAC systems have been used to monitor fire alarm panels, with variable success. Many of these have given way to systems like those of Keltron, which uses wireless transceivers to monitor a variety of building systems through radio or Ethernet.

2.6.2 Major Companies of the Fire Alarm and Detection Industry

In the United States, Tyco International Ltd. is a major player in the security, burglar, and fire alarm industry, making up 26.2 percent of the market share (Culbert, 2010). Other major companies that play a role in this industry include the Honeywell International, Siemens AG, and UTC Fire and Security.

2.6.2.1 SimplexGrinnell

The subsidiaries of Tyco International include several companies that manufacture safety, security, and electrical products, as well as SimplexGrinnell, which manufactures many of the fire alarm systems in the United States. SimplexGrinnell's addressable fire alarm panels are the 4100U, the 4010, and the 4008 (*Addressable Fire Alarm Panels*, 2011).

While the 4010 and 4008 panels are most applicable to small to mid-sized buildings, the 4100U is designed for larger applications. It has a maximum capacity of 2000 addressable points, with up to 250 points that support TrueAlarm sensors, allowing sensor analog values to be communicated back to the control panel (*Simplex 4100U Fire Alarm Control Panel*, 2010). For TrueAlarm smoke sensors, the control panel maintains a current value, peak value, and average value for each sensor and tracks the status of each sensor by comparing the current values to the average value.

This allows the system to account for dirty sensors that could result in false alarms. The 4100U also features a two-line by 40-character LCD display for information readout, a numeric keypad, six system status indicator LEDs, three programmable LEDs, and five programmable function switches. The panel's alarm and trouble signal history log can maintain up to 1,300 total events, which are viewable on the LCD display or printed by connecting a printer to the panels RS-232 port.

The 4100U is also compatible with other communication devices, such as the BACpac Ethernet module and the SafeLINC internet interface. The BACpac Ethernet module converts data from the panel to the BACnet protocol, allowing communication with other BACnet-compatible devices (*BACPAC BACnet Products*, 2010). The module is connected to the panel through the RS-232 port and has an output port for Ethernet LAN connection. The SafeLINC internet interface allows devices such as personal computers and PDAs to remotely monitor the fire alarm panels through an internet connection (*SafeLINC Fire Panel Internet Interface*, 2010). The module provides access for up to 20 different user accounts, but only one user can access the interface at a single time. The interface is modeled after that of an internet browser and includes a system snapshot of the control panel status, access to history logs and reports, the local time, a side menu with additional links and a page of panel details.

SimplexGrinnell also provides a line of annunciators that are compatible with the 4100U. One particular annunciator is the 4190 PC annunciator. The interface is text-based and essentially provides a printout of each sensor's state as it changes. Historical logs and reports are also accessible. Figure 2.1 shows its interface.

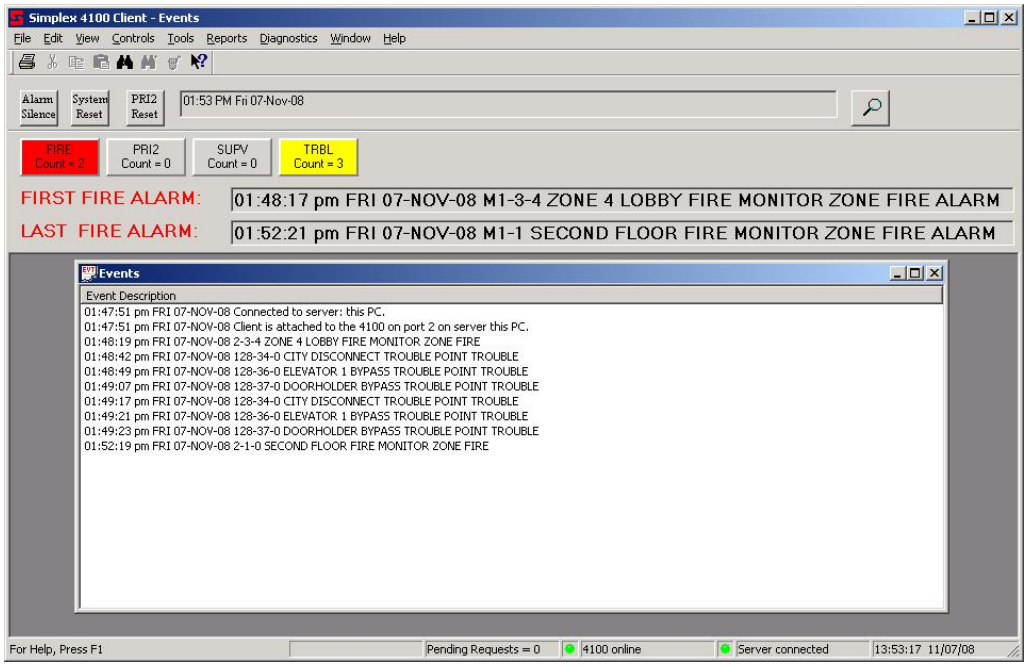


Figure 2.1: 4190 PC Annunciator User Interface

SimplexGrinnell's other annunciator is called the TrueSite Workstation, which provides a graphical display of the building's floor plan and a textual display of the alarm states (*TrueSite Workstation with Multi-Client Capability*, 2010). The graphical display has navigational tools to view the floor plan in greater or lesser detail and supports standard fire service annunciation icons. Custom alarm and system messages, as well as historical logs, are viewable on the TrueSite Workstation. Figure 2.2 and Figure 2.3 show both of the Workstation's interfaces.

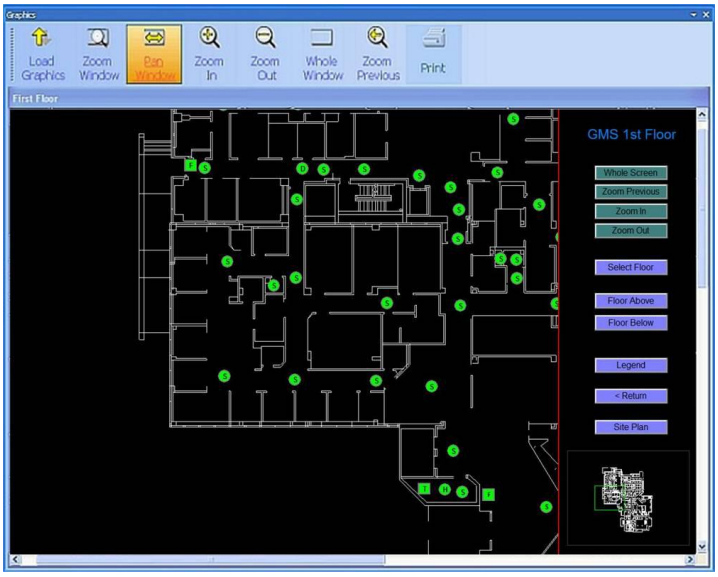


Figure 2.2: TrueSite Floor Plan Window

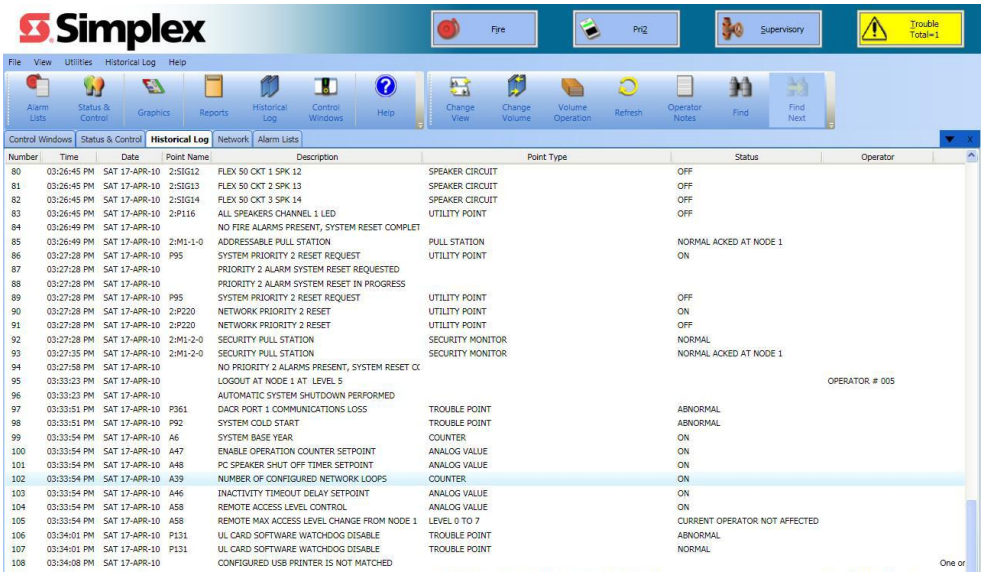


Figure 2.3: TrueSite Historical Log Window

SimplexGrinnell also manufactures initiating devices that are compatible with its fire alarm panels (*SimplexGrinnell Initiating Devices*, 2011). This includes the TrueAlarm sensors, manual pull stations, carbon monoxide detectors, and modules that provide addressability to conventional devices. It also manufactures speciality devices, such as an infrared smoke detector, a VESDA VLC-600 TrueAlarm Laser,

and a video smoke and flame detector which provides smoke and flame video in real-time (*SimplexGrinnell Specialized Detection Devices*, 2011). The Very Early Smoke Detection Apparatus (VESDA) periodically samples the air in an area for smoke (*VESDA VLC-600 TrueAlarm Laser COMPACT*, 2010).

2.6.2.2 Honeywell International, Inc.

Honeywell has a number of subsidiaries that manufacture fire alarm systems. These include Silent Knight, Fire Lite Alarms, Gamewell Fire Control Instruments and Notifier. Notifier is one of the acceptable manufacturers on the University of Maryland campus (*Design Criteria/Facilities Standard Manual*, 2005), and for this reason, the products of this subsidiary are explored in depth.

The intelligent, addressable control panels of Notifier include the ONYX series, FireWarden series, and Spartan-25, the latter two being applicable in smaller facilities. Of the four different products in the ONYX series, two of them, the NFS2-640 and NFS2-3030, fit the requirements of medium to large facilities (Honeywell International, 2010a). The NFS2-640 features one signaling line circuit, or SLC, which can be expanded to two SLCs (Honeywell International, 2010d). The NFS2-3030 can be expanded to ten SLCs (Honeywell International, 2010c). Each SLC can hold up to 159 intelligent detectors and 159 addressable modules (which includes pull stations), giving the NFS2-640 a maximum capacity of 636 addressable points and the NFS2-3030 a maximum capacity of 3,180 addressable points. Both panels contain a 640-character large display with a full QWERTY keyboard, as well as an EIA-232 printer port for communication to external devices. The NFS2-640 has a history log that can hold 800 events and 200 alarm-only events, while the NFS2-3030's history log can hold 4,000 events and 1,000 alarm-only events.

The NFS2-640 and NFS2-3030 both utilize FlashScan and ONYX intelligent sensing (Honeywell International, 2010c, 2010d). FlashScan is a detector protocol

that allows a panel to poll up to 318 devices in less than two seconds and activate up to 159 outputs in less than five seconds, allowing for near real-time annunciation. ONYX intelligent sensing is a software algorithm for detector readings. It allows a user to program different levels of sensitivity adjustment and perform sensitivity measurements to avoid false alarms and determine if detector maintenance is required. The algorithm also gives the detectors a self-optimization capability. With this feature, the detector takes normal analog readings over an extended period of time and then sets its pre-alarm values just above those readings. Cooperative multi-detector sensing, the ability of a smoke detector to use readings from nearby sensors in alarm decisions, is also possible with ONYX intelligent sensing.

Many of the Notifier panels use networks and modules for monitoring and controlling. The NFS2-640 and NFS2-3030 support the FireWatch Internet monitoring modules, which use the Internet to monitor alarm signals (Honeywell International, 2010c, 2010d). Another option is NOTI-FIRE-NET, which is a network that connects multiple Notifier panels using a combination of fiber optic and wire communication paths (Honeywell International, 2004). Each panel on the network is a node, and each node can monitor and control other nodes. The failure of one node does not affect communication of other nodes in the network. NOTI-FIRE-NET can be monitored on either a network command annunciator, or an ONYXWorks workstation. The ONYXWorks workstation, which is an industrial, high-performance computer, can integrate a number of life safety and building systems, which include fire, security, CCTV, and other facility information (Honeywell International, 2010f). It was designed to be modular and allow one workstation to manage a combination of different manufacturers, technologies, and networks. Communication between the workstation and its systems can occur over local Ethernet or wide-area TCP/IP networks. The interface of ONYXWorks is supported by Windows XP and features a floor plan, real-time event printing of system-wide events, control of security and

fire panels, a history log, and the capability for mass notification.

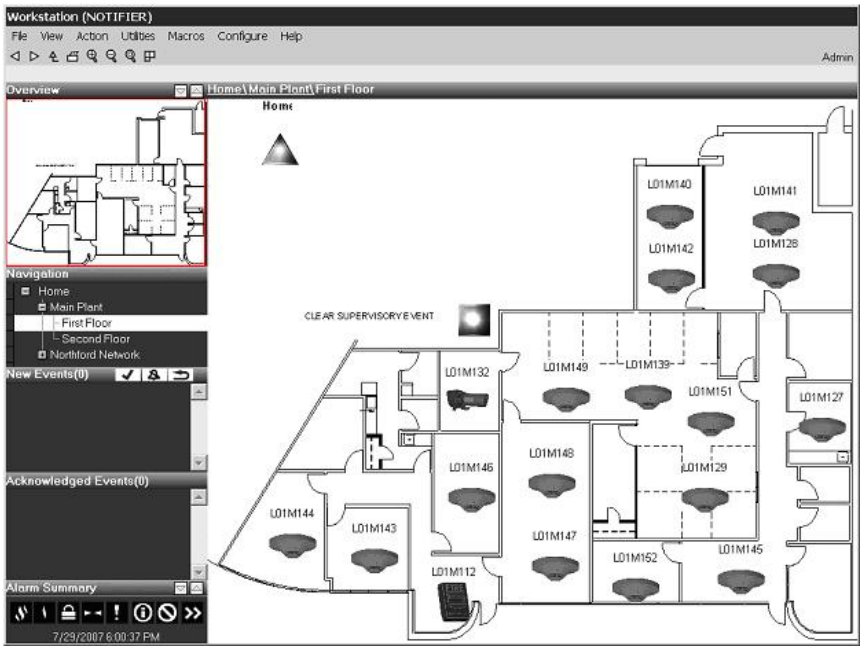


Figure 2.4: ONYXWorks Workstation software

Other network devices include the BACnet gateway and ONYX FirstVision. The BACnet gateway allows a single panel or panels on NOTI-FIRE-NET to communicate with a network using the BACnet/IP communication protocol (Honeywell International, 2009). On NOTI-FIRE-NET, the BACnet gateway can support up to 14 other nodes and 15,000 objects. The ONYX FirstVision is a building-installed, touch screen display that acts as a navigational tool for EFRs (Honeywell International, 2010e). It uses CAD drawings and the VeriFire tools database to display a floor plan annotated with the location of activated detectors and the sequence of activation. The ONYX FirstVision also includes the location of water supplies, evacuation routes, special hazards, and the shutoff valves for gas, power, and HVAC systems.

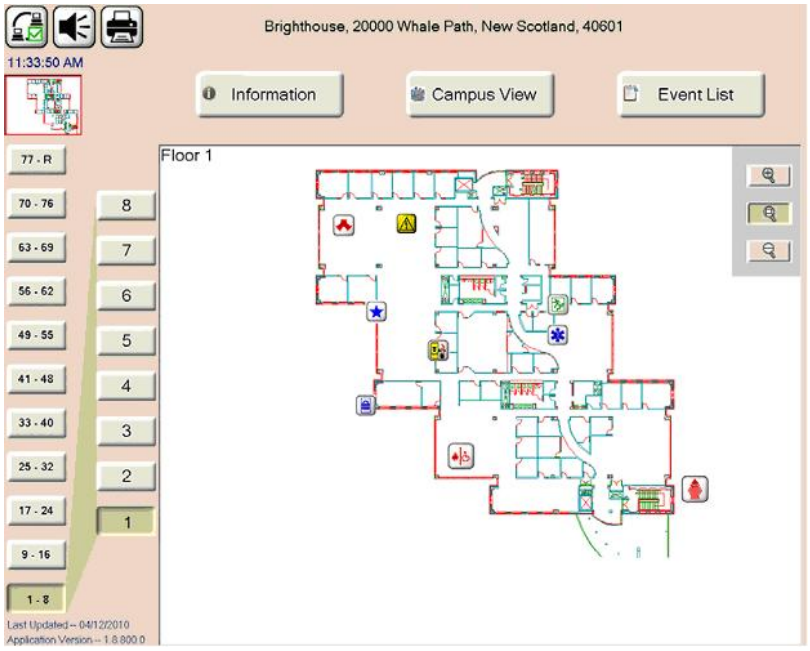


Figure 2.5: Interface of ONYX FirstVision

Notifier offers a number of conventional and intelligent detectors. Its conventional detectors include ionization and photoelectric smoke detectors, fixed-temperature and rate-of-rise thermal detectors, ultraviolet and infrared flame detectors, and reflective beam detectors (Honeywell International, 2010b). The intelligent detectors utilize the FlashScan capability described above.

2.6.2.3 Siemens AG

Siemens' line of addressable alarm panels includes the FireFinder XLS, MXL, and FireSeeker 250. While the FireSeeker 250 is mainly meant for small or medium-sized facilities, the FireFinder XLS and MXL series are capable of handling larger facilities, such as commercial or industrial buildings. The FireFinder XLS has the capacity for 2,500 addressable points, which can include both analog and conventional detection devices (Siemens Industry, Inc., 2010a). It also provides manual control of a building's HVAC system. The addressable panel features a 6" backlit, LCD display with touch screen capabilities. The system interface has the capacity

for hundreds of large-font text characters, hazmat icons, NFPA fire safety symbols, and graphic maps, and provides access to a history log that is capable of holding 5,000 events. This series also includes a remote printer module that provides an output port that, when configured, can be used to communicate with external systems.

In the MXL series, the MXL is an analog fire protection system and is one of the acceptable fire alarm systems on the University of Maryland campus. It is capable of handling more than 2,000 intelligent input-devices and is capable of monitoring security devices (Siemens Industry, Inc., 2006). Some of its features are not as sophisticated as those of the XLS. The panel only includes an 80-character backlit alphanumeric display and an 800-event history log. The MXLV has similar capabilities as the MXL, but also includes a voice evacuation system (Siemens Industry, Inc., 2005). Both the XLS and MXL can be combined into a network of other XLS and MXL systems that can be monitored and controlled using a network color-graphics command center.

Siemens also provides three types of detection devices: conventional, intelligent, and specialized. The conventional detection devices are similar to those of SimplexGrinnell and Honeywell. The thermal detectors can be rate-of-rise, fixed-temperature, a combination of the two, or Detect-A-Fire (Siemens Industry, Inc., 2002). Detect-A-Fire thermal detectors are rate-compensating and can accurately detect the surrounding air temperature regardless of the fire growth rate.

The intelligent detectors include photoelectric detectors, ionization detectors, FirePrint detectors, and thermal detectors. These differ from their conventional counterparts in that each contains a microprocessor that allows more complex detection, error-checking, and supervision algorithms to be remotely programmed into the detectors (Siemens Industry, Inc., 2010b). This allows the detectors to distinguish between false alarms and actual fire hazards. The FirePrint detector has

photoelectric and thermal sensing capabilities and can use fire hazard profiles programmed from the control panel to compare data.

Specialized detectors include a VESDA air sampling system, the Series 3/X3 air duct detector, and linear beam smoke detector. The VESDA LaserPLUS detector draws air from a network of air sampling pipes, with each pipe containing a sensor that detects changes in airflow (Siemens Industry, Inc., 2003). Before the air reaches a laser light sources inside the VESDA detector, it passes through a dual-stage air filtration system that filters dust from the air and prevents the inner chamber from being contaminated. The VESDA LaserPlus detector can communicate with other VESDA detectors through a communications protocol called VESDAnet. It is separate from NOTI-FIRE-NET. The Series 3/X3 air duct detector operates independently, while the linear beam smoke detector is compatible with the MXL series of fire alarm control panels.

2.6.3 Other Relevant Technology

2.6.3.1 Keltron

The University of Maryland Facilities Management recently installed a Keltron Life Safety Event Management System to monitor the more than 200 fire alarm systems located on its campus (Robinson, 2009). For decades, the University had been using its HVAC system for this purpose, and this presented a number of issues. First, only sixty percent of the campus alarm systems could be monitored using the HVAC systems, forcing the management to rely on people to call in whenever an alarm was set off. Second, the HVAC system was not designed to handle the variety of brands and models that are represented on the campus. This greatly limited the amount of information available to the technicians and firefighters, all of whom had to defer to the building's control panel for further details. Third, HVAC systems are

not held to the same strict codes of fire alarm systems, leaving the University with major flaws in the communication pathways between the detector and the central monitoring system.

The solution to these issues was the installation of the Keltron Active Network Radio System (Corporation, 2009). It utilizes wireless transceivers with 8 programmable inputs that allow for two-way transmission between the detector and the central monitoring location. With the Keltron DataTap, transceivers can take point-specific information from fire alarm control panels and relay it through the network. This device is connected to the control panel's RS-232 port and is compatible with the four manufacturers discussed above.

Keltron also offers Ethernet networking systems that have similar capabilities as those of its radio system. The company also features its fire and security alarm monitoring systems called the DMP703 (Corporation, 2008). The DMP703 can handle 20,000 direct connect inputs for continuous supervised monitoring and includes a touch screen. The system can also interface with other fire alarm control panels via the RS-232 ports.

2.7 Current Firefighter Technology

In addition to the technology of building sensor systems, EFRs use, or are beginning to use, a variety of technologies for preplanning, firefighter locators and vital signs indicators. These technologies can be applied to EMS and hazmat incidents as well as fires. However, most of the technology is aimed at firefighters. In an emergency situation, knowing the hazards of the building and the location and health of firefighters is essential to a successful resolution to the event.

2.7.1 Pre-planning Software

Pre-planning is an essential tool for firefighters. Pre-plans locate fire hydrants, stairwells, hazardous materials in a building, and fire alarm control panels. In addition, pre-plans make note of building construction in terms of flammability, known difficulties in access, and surrounding traffic patterns (*NFPA 1620: Standard for pre-incident planning*, 2010). While pre-planning has traditionally been done on paper, several companies provide software to aid in creating pre-plans. The CAD Zone, Iron Compass, RealView, and FDM are some of the producers of pre-planning software. Each pre-planning aid aims to comply with NFPA 1620.

The CAD Zone was one of the first companies to begin producing software for pre-planning purposes. Their Fire Zone product helps fire service personnel draw site plans and add pertinent pre-incident planning information. It uses simple drawing tools to construct the floor plans, as well as using pre-made symbols to represent stairs, hydrants, and hazards (Zone, 2010a). In order to make the pre-plans accessible on the scene of the incident, The CAD Zone also sells First Look Pro, a mobile pre-plan organizer. It is designed to be used on mobile computers and even touch-screen computers. It uses a search function to find addresses and displays the information input into Fire Zone. It also includes a mobile mapping function to show the fire service vehicle's progress toward the location of the event through global positioning system (GPS) tracking (Zone, 2010b).

The next producer of pre-incident planning technology is Iron Compass, a company known for making maps. Their OnScene Xplorer product also uses a search function to find addresses or intersections, with "spell-right" technology ensuring proper street name entry. Depending on the zoom of the image, different information is displayed to simplify read-out. In addition GPS technology can be used to find fire hydrants and give precise coordinates to MedEvac helicopters. It also allows for space to note important information such as occupancy, shut-off valve locations,

hazardous materials, and fire suppression systems. Rather than having a separate mobile program, OnScene Xplorer can be used to create and view pre-plans en route (Compass, 2010).

RealView produces Command Scope, their mobile pre-planning software, available as a yearly subscription. All the pre-planning data is synchronized across a department through a RealView server. Command Scope allows for both commercial and residential pre-plans by asking residents to submit their own pre-plan information online. Like the other softwares, Command Scope shows building site information, hazmat details, and geographic information system (GIS) maps (LLC, 2010).

Another producer of pre-planning software is FDM. FDM produces records management system (RMS) modules for a variety of applications including hydrant tracking, personnel management, maintenance records, training details, and property information. The properties module includes records from prior incidents, permit information and inspection results. The properties module also includes information on occupancy, building construction, hazards, important contacts and built-in safety features. All of FDM's modules are fully customizable, allow for image viewing and connect with the fire department's CAD program so that dispatchers can have access to information and building plans (FDM, 2010).

2.7.2 Locator and Vital Sign Indicator Technology

Tracking firefighters within an incident allows the incident commander to be certain that all firefighters are in their intended locations to better combat the fire as well as avoid injury. There are several ways that firefighters can be tracked within a building. Pedometry uses calibrated step lengths to estimate distance and direction, as well as vertical movement. It counts steps to provide the relative location and display the estimated path over the floor plan. In ad hoc self-forming networks,

also known as mesh networks, radio networks form that uses the time of signal from other users to track location and direction. Like pedometry, it also shows a relative location and can be overlaid on the floor plan. If the incident commander can be linked in, he can see the locations of all the firefighters. Unfortunately, linkage from the inside of the building to the outside can be difficult with ad hoc networks due to the need for a grid layout in order to operate correctly. Another method is through triangulation. Using transmitters with known coordinates, such as those in fire service vehicles outside the building, the locations are pinpointed and can be displayed on the floor plan. Triangulation can successfully transmit through different materials and the incident commander can see the entire team. Other technology used to track firefighters includes ultra wideband (UWB) and radio-frequency identification (RFID) technologies (Bryner, 2008).

Scott Health and Safety produces the Pak-Tracker, a firefighter-locating device. The Pak-Tracker operates on a directional 2.4 GHz radio frequency. The handheld monitor can connect with up to thirty-six individual units. It is designed to be operated with gloves on and weighs only 2.2 lbs. It can be used alone or incorporated into a self-contained breathing apparatus (SCBA) system. If the system does not detect movement for a period of time, it goes into full transmission mode so that other firefighters can find the immobile firefighter. When using the handheld module, audio and visual tools help locate the firefighter (Health & Safety, 2010).

Harris Corp's GR-100 firefighter tracker uses a SCBA-mounted transmitter and a radio network at 700 and 800 MHz frequencies to track firefighters. The data is then displayed on a graphic user interface (GUI) so that an incident commander can track the firefighters in 3D, updated every few seconds. The system also uses accuracy-enhancing algorithms, inertial information, and GPS tracking. In order to address some challenging structures for fire departments, Harris aimed the product at single-family residences, strip malls, and small office buildings. The GR-100

is able to track the identity, location, and path of individual firefighters in three dimensions (Roberts, 2010).

Knowing the location of a firefighter is important, but knowing whether that firefighter is under physical duress is top priority. In the fire service, half of all fatalities of firefighters come from heart attacks. Several companies produce undershirts that are intended to track the vital signs of the firefighter. At the National Personal Protective Laboratory, a vest called a "lifeshirt" uses five sensors to track body temperature, heart rate, and breathing ("Vitals vest – physiologists create undergarment to measure vital signs of firefighters", 2008). Zephyr Technology developed their BioHarness First Responder System to monitor body temperature, heart rate, and breathing in the context of the firefighter's activity level and posture, which could alert an incident commander to a dangerous situation. Using a monitoring system, the vital signs can be assessed at a glance on a computer that can provide real-time alerts or assist in post-event analysis (Thompson, 2009). Vital signs monitoring can provide essential information in order to protect firefighters on scene.

2.8 Human Factors

As the goal of our project is to develop and demonstrate the effectiveness of a tactical decision aid display benefitting firefighters and other EFRs, it was important for us to keep the needs and limitations of the user in mind throughout the design process. The field of human factors is concerned with the study of the capabilities and limitations of human beings and how these factors affect human-computer interaction (HCI), making research being done in this field very relevant to our project. Since the decision support tool we have developed is a relatively simple decision support system (DSS) with a graphical user interface (GUI), human factor research dealing with both DSS and GUI design was studied and used to help

guide our design choices.

2.8.1 Decision Support Systems (DSS)

We define decision support systems to be those computer systems dedicated to aiding the decision-making process of the user by providing him or her with information gained through analysis of available data. Research concerning DSS designed to support emergency response is particularly relevant to our project. Unfortunately, literature dealing with emergency DSS is lacking when compared to the research that has been conducted concerning DSS with more conventional, corporate applications. It should be noted, however, that the body of research on emergency DSS has been steadily growing in recent years (Dai, Wang, & Yang, 1994). Dai et al. (1994) studied the design and development of computer support systems intended to aid emergency decision-making. Their paper addresses three main issues: the differences between emergency decision-making and what the authors call conventional decision-making, the kinds of aid DSS are capable of providing for emergency decision-making, and the specific requirements that should be considered in the design and development of emergency DSS. According to Dai et al., emergency decision-making problems differ from conventional decision-making problems in the following five ways:

1. Attributes of an emergency-decision making problem are often uncertain.
2. Environmental factors are often changing rapidly in emergency scenarios.
3. Emergency decisions are often time-dependent and are often made with incomplete or inexact information.
4. Usually only one or two goals are important in an emergency situation.
5. Even the soundest decision may involve substantial risks when faced with an emergency scenario.

(Dai et al., 1994)

The paper also emphasizes that the user of an emergency DSS is often under duress and that the DSS should be designed to ease the user's stress as much as possible. According to the authors, this can be achieved by ensuring the information presented by an emergency DSS is concise, adaptable, and insensitive to imprecision of input data and that limited interaction is required between the user and the AI governing the DSS during the course of an emergency (Dai et al., 1994). Dai et al. then go on to explain how the emergency DSS should function differently at different stages of the emergency scenario. During the pre-planning stage, the emergency DSS should be able to provide the following functions:

- The ability to collect and store information dealing with possible emergencies
- The ability to acquire expert knowledge about an emergency
- The ability to aid managers in developing an emergency plan
- The ability to run simulations evaluating the effectiveness of the standing emergency policy
- The ability to process and use real-time data to predict and monitor developing emergencies

(Dai et al., 1994)

Meanwhile, the emergency DSS should be able to perform these functions during the course of an emergency situation:

- The ability to retrieve information about a developing emergency in real-time
- The ability to analyze an emergency scenario as it progresses
- The ability to convey expert-level advice and make proposals to the DSS user
- The ability to aid the user in making and implementing a decision

(Dai et al., 1994)

Furthermore, Dai et al. state that the response time of emergency DSS should be minimized, even at the cost of storage capacity. The paper concludes with a

discussion of the basic framework of an emergency DSS and a description of a prototype emergency DSS used for disaster response in coalmines.

Ye, Wang, Li, and Dai (2008) studied the development of an emergency DSS based on the general decision process of emergency management personnel. The paper outlines nine key elements present in the general emergency decision process: (O, A, C, E, R, RES, S, T, V), where O is the set of roles of those involved in emergency response, C is the environment in which the emergency takes place, E is the set containing the events of the emergency, R is the relationship of the events contained in E, RES is the set of all emergency resources, S is set of statistics describing the emergency system, T is time, and V is geographic space (Ye et al., 2008). Figure 2.6, reproduced from Ye et al. (2008), illustrates the relationship of these nine factors.

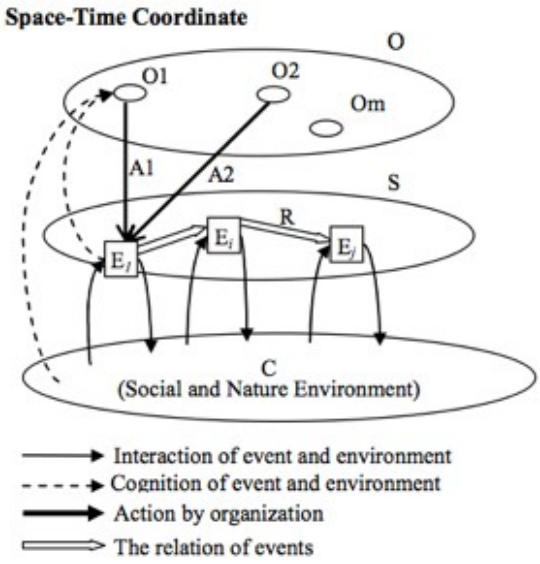


Figure 2.6: Relationship of nine key elements of the emergency decision process

The authors then use this model of the general emergency decision process as a basis for their proposal of a general architecture for an emergency DSS. It should be noted that, unlike in the previous paper discussed, the architecture proposed dictates a constant interaction between the DSS and the human user throughout

the emergency scenario.

Drury, Klein, Pfaff, and More (2009) studied DSS that utilize simulations to predict how a given decision will affect the course of an emergency situation. A cost metric was developed to compare a decision to its alternatives. This cost comparison was visually conveyed to the user through the display of box-and-whisker plots. To assess the usefulness of this developed DSS, a test was designed where two groups had to determine the number of emergency-response vehicles to dispatch to a developing emergency situation. While both groups were given a textual description the emergency scenario, only one group was given access to the developed DSS. It was found that the DSS group was able to make decisions closer to the normatively correct decision. Furthermore, the DSS group reported a higher degree of confidence in their decisions and rated the amount of decision support they received higher than the control group.

While not focusing on emergency DSS specifically, other human factors research concerning DSS contain conclusions relevant to our project. Research dealing with DSS designed to aid command decisions is especially relevant, as the target user of our tactical decision aid display is the incident commander directing an emergency response effort. Breton, Paradis, and Roy (2002) studied the work being done at the Defence R & D Canada - Valcartier toward developing a command and control DSS design tool called CODSI (Command Decision Support Interface). CODSI is used to evaluate and validate design concepts from both a human factors perspective and an operational perspective. To illustrate the effectiveness of CODSI, the design of a tactical DSS intended for naval threat-assessment applications was studied. Through this study, CODSI was shown to accurately model and assess human factors considerations of DSS with command and control applications.

Another facet of DSS research relevant to our project concerns the display of uncertain information. Bisantz, Marsiglio, and Munch (2005) conducted four

studies regarding the display of probabilistic information. The first three studies dealt with user performance of stock purchase tasks where information regarding the profitability of a given stock was probabilistic. Two aspects of the display were varied in these studies: the display format and the specificity level of probabilistic information. From these studies it was concluded that while performance did not vary with display format, increased specificity improved performance. The fourth study had participants create membership functions that corresponded to three distinct display formats. From this study it was found that users can successfully interpret probabilistic information displayed in a non-numeric manner and use this information to make effective decisions.

A good portion of the human factors research being conducted on DSS deals with the design of so-called "intelligent" systems, systems with a high degree of automation. Guerlain, Brown, and Mastrangelo (2000) studied characteristics that are common to three intelligent DSS shown to be successful in practice: the Antibody Identification Assistant (AIDA), the Regional Crime Analysis Program (RECAP), and the River Flooding Forecasting System. Six attributes representative of successful intelligent DSS were determined from this study and are given below:

1. Interactivity - the system is able work well with both human users and databases. Users are given a wide variety of possibilities instead of a single "optimal" decision.
2. Event and Change Detection - the system is able to effectively communicate status changes and important events.
3. Representation aiding - the system is able to convey information in an informative yet human-like way.
4. Error Detection and Recovery - the system is able to check for typical reasoning errors made by users. The system is also able to identify situations that are beyond its own capabilities.
5. Information out of Data - The system is able to extract information from large data sets and provide the user tools for dealing with smaller data sets, outliers, and possible sources of error.

6. Predictive Capabilities - the system is able to predict both the future consequences of a given action and the future state of an environment.

(Guerlain et al., 2000)

Despite the appeal of intelligent DSS, it is important to keep the limitations of such systems in mind. Parasuraman and Wickens (2008) argue that, despite the high degree of automation of current DSS, the human user is still indispensable to the decision-making process. Though current technology allows almost complete autonomy by some DSS, the authors argue that operator input must be factored in to high-risk decisions. Without this input, it is possible that the user will follow the decision made by the system incorrectly due to a lack of understanding. The authors also point out how a lack of involvement by the user may lead to a lack of trust and, in turn, a breakdown of the decision-making process.

Rovira, McGarry, and Parasuraman (2007) studied the effects imperfect automation has had on command and control decisions. To assess these effects, the authors designed a test in which eighteen participants performed a targeting simulation based on sensor data. It was found that while reliable automation reduced decision-making time considerably, imperfect automation lead to a compromised decision-making process with high costs. Based on this study, the authors recommend that automation features should not be included in a DSS unless fully dependable automation is ensured.

Sarter and Schroeder (2001) studied how uncertainty of information and time pressure affect DSS aided high-risk decision making. The authors examined the effectiveness of two types of DSS: status displays and command and control displays. In their study, twenty-seven pilots flew twenty simulated flight patterns with a possibility of in-flight icing while the accuracy of the DSS provided and the experience of the pilot was varied. It was shown that while accurate information significantly improved how pilots approached and handled in-flight icing, inaccu-

rate information lead to the performance of pilots falling below those with no DSS. Based on this study, the authors recommend the use of DSS that simply provide information relating to a particular scenario over those DSS that make particular command suggestions for applications where perfect reliability of information cannot be guaranteed.

Another limitation of DSS involves those with displays that can become cluttered with symbols and other information. When DSS displays become cluttered, sensory overload can occur and the performance of the user can suffer greatly. John, Smallman, Manes, Feher, and Morrison (2005) studied how heuristic automation can be used to "declutter" displays with this problem. To assess the effectiveness of heuristic automation, twenty-seven Navy users with experience related to DSS with tactical displays monitored a simulated airspace and were told to find and respond to aircrafts that posed a significant threat. Meanwhile, a heuristic algorithm that dimmed the symbols of aircraft it deemed less threatening was constantly run in the background of some DSS. While the DSS users did not let the dimming of symbols affect which aircraft they deemed to be threatening, the decluttering algorithm was shown to improve their time response to threatening aircraft by twenty-five percent. Furthermore, twenty-five of the twenty-seven users said they preferred systems with the decluttering heuristic. The authors concluded that properly designed heuristic decluttering of a display may prove valuable in many applications where the display can easily become cluttered, such as in air-traffic control or crisis team management.

2.8.2 GUI Design

A significant component of our research project involves the design of a graphical user interface to provide EFRs with useful information concerning a developing fire emergency in a timely manner. For this reason, it was important that we were aware of the human factors work related to GUI design. According to *An Intro-*

duction to Human Factors Engineering (2004) by Wickens et al., there are thirteen principles of display design to consider when creating an effective GUI. These thirteen principles are as follows:

1. Make displays legible (or audible): The characters or objects being displayed must be discernible.
2. Avoid absolute judgment: Users should not be required to judge the level of a variable on the basis of a single sensory variable.
3. Top-down processing: People will perceive and interpret signals based on prior experience.
4. Redundancy gain: The more times a message is presented, the more likely it is to be correctly interpreted.
5. Discriminability: Similar appearing signals will most likely be confused.
6. Principle of pictorial realism: Symbols employed should look like the object or element that it represents.
7. Principle of the moving part: The moving elements should move consistently with the user's expectations.
8. Minimizing information access costs: Keep frequently accessed sources in a location such that the cost of traveling between them is small.
9. Proximity compatibility principle: Sometimes two or more separate sources of information are required to complete a given task and must be mentally integrated.
10. Principle of multiple resources: Information should be divided across resources to allow easier processing by the user.
11. Principle of predictive aiding: A display that predicts future events will usually enhance human performance.
12. Replace memory with visual information: knowledge in the world: Place specific cues in the task environment to remind the user what needs to be done.
13. Principle of consistency: Symbols and other design elements should be consistent with other displays the user may be using.

(Wickens, Lee, & Liu, 1997)

While some of these principles are not applicable to the project at hand, the majority of these guidelines should be considered to ensure the design of an effective GUI not hindered by the limitations and preconceived notions of the human user.

Priegnitz et al. (1997) wrote a paper on the human factors considerations involved in the design of a GUI for open systems radar product generation. The authors studied how a GUI could be used to improve upon the existing unit control position interface. The paper provides a good understanding of the merits of a graphical display and paints a clear picture of exactly what steps go into GUI design.

Talcott, Bennett, Martinez, Shattuck, and Stansifer (2007) studied the GUI design of military command and control decision aids and used the principles of direct perception, direct motivation, and perception-action loops to develop a system of perception-action icons. Participants of their study used GUIs with and without these perception-action icons implemented to provide estimates of combat resources. The results showed that GUIs with these perception-action icons improved user performance significantly. This paper demonstrates how a system of standardized icons with incorporated color changes can be used to effectively communicate status changes to the user that demand attention.

A particularly useful source related to GUI design is the Federal Aviation Administration's *Human Factors Criteria for Displays* (2007). While too extensive to summarize here, Chapter 5 of the technical report covers criteria for displays varying from office use to heads-up and tactical displays (Ahlstrom & Kudrick, 2007). While dealing more with hardware than GUI programming considerations, the suggestions made by the FAA proved valuable in the design of our decision support tool.

Chapter 3

Methodology

3.1 Case Study Methodology

Before our team began the formulation of a decision support visualization tool, a thorough understanding of the current capabilities of building information systems needed to be established through a case study. The objective of the case study was to better understand the types of fire alarm systems found in current, state-of-the-art buildings, as well as different functioning modes for purpose-specific buildings. Each building is intended for different events and uses, so there are different requirements and operation techniques for each fire alarm system. The case studies included an observation of the Jeong H. Kim Engineering Building, a laboratory and classroom building, the Comcast Center, an event arena with office space and Keltron, the central monitoring system on the University of Maryland campus. All of these buildings were located on the University of Maryland, College Park campus. An additional office building, the CareSource Building in Dayton, OH, was also studied. This building was chosen because it is a state-of-the-art building aimed for single-occupant office use that was available to our team.

The following is a list of building or fire alarm system features that were noted during the case study, and important aspects of each feature:

- Building Type and Usage: This information provided a context for each building case study. Each building was purpose-specific and the fire alarm control panels and emergency operations were customized around the specific needs of the building and its occupants. This knowledge gave a framework for the study and provided insight into the different operations of current, high-tech buildings.
- Type of Fire Alarm Control Panel (FACP): An observation of FACPs allowed

Team FFA to understand what types of panels are currently used in state-of-the-art buildings. These findings were used to direct our efforts in requesting donations of equipment. It was necessary to obtain commonly used panels for further testing.

- Location of FACP and Other Control Panels: To understand the current operations of firefighters during emergency situations, it was necessary to note typical locations of control panels in buildings. The current locations emphasized a need for external access to FACP information.
- Current Floor Plans or Maps: An examination of the floor plans and maps provided insight into current systems. Current floor plans and maps found in these buildings were simplistic and vague, indicating a possibility for improvement to the information provided on floor plans.
- Sensor Numbering and Addressability: Many current sensors are point-addressable. Sensor location data were pinpointed as possibly useful yet underutilized aspects of current technology.
- Specialty Sensors: Chemical sensors, high-tech smoke detection systems, pressurized stairwells and elevators were all noted as items of particular interest.

3.2 System Design

3.2.1 Assessing EFR Needs

In order to adequately demonstrate how a decision support visualization tool can effectively utilize, contextualize, and present building sensor data, our team saw it necessary to construct a prototype decision support tool. In accordance with most established design methodologies, we began our prototype design by considering the needs and limitations of the end users, namely firefighters and other EFRs. In 2004, NIST held a workshop to identify and inventory the information needs of EFRs during the course of an emergency situation (Jones et al., 2005). Our team used the results of this workshop as a starting point toward determining what types of information should be included in and conveyed by our prototype visualization tool.

Appendix C of the technical report "Workshop to Define Information Needed by Emergency Responders during Building Emergencies" contains several lists of

the kinds of information that firefighters would like to know both en route to a fire emergency and on the scene of the fire (Jones et al., 2005), as discussed in Section 2.4 of this thesis. These lists are quite exhaustive, so our team felt it was important to prioritize the items requested and include only the most important information in our prototype. After consolidating all the information requested by EFRs during the course of the workshop into two lists (static and dynamic information), we categorized each item requested as either high priority (information essential to understanding the nature and progression of a fire emergency that must be included if available), medium priority (potentially useful information that is either not entirely necessary or would be hard to integrate into our prototype), or low priority (information that is excessive, unnecessary, or not currently available). In order to avoid clutter, we decided to include only the information classified as high priority. The resulting list of static information derived from the workshop includes:

- Building style (e.g., one story, two story, n-story, auditorium, sublevels, etc.)
- Building construction (e.g., type I, II, III, IV or V; fire resistive, noncombustible or limited combustible, ordinary, heavy timber, or wood frame)
- Presence and Location of Hazardous materials (e.g., above ground propane tank, gas lines, chemicals, etc.)
- Location of fire department hookups for sprinkler systems and/or standpipes
- Location of staging areas and entrances and exits to buildings
- Location of doors, windows, stairwell risers, fire walls (with ratings and area separation), roof access, and fire sensors
- Location of fire alarm panels and remote annunciator panels
- Location of utility shutoffs
- Location of building system controls (e.g., HVAC, smoke control, etc.), areas of effect, and special operating systems
- Location of evacuation quality elevators, floors served, and location of elevator overrides and how to control

- Location of convenience stairs and evacuation stairs
- Areas (i.e. zone boundaries) protected by sprinklers or other devices

The list of dynamic information included from the workshop is as follows:

- Approximate location of fire within a building
- Fire size and duration
- Location of fire detectors in alarm
- Location of sprinklers activated
- Location of CBR sensors in alarm
- Location and condition of smoke
- Location of elevators (and state of operation) during an incident

3.2.2 Evaluation of Prior Work

In the NIST technical report "Demonstration of Real-Time Tactical Decision Aid Displays" by Davis et al., 2007, a prototype tactical decision aid display interface is proposed to meet the EFR needs determined during NIST's previously held workshop. Before starting to formulate the features of our own prototype decision support visualization tool, our team thought it best to review NIST's proposed design and evaluate where it succeeded and how it could be improved.

NIST's prototype system consisted of two decision aid display interfaces: one to be used by EFRs en-route to a fire emergency and one to aid the incident commander on site. The proposed en-route GUI would display a plan view of the building and surrounding area and would provide information such as the location of building standpipes, the building's name and address, and the location of entrances to the building. After reviewing this en-route display, our team determined that, while the information displayed is indeed useful, the manner in which it is displayed is only a marginal improvement over the pre-planning and dispatch software that is

already in use by many fire departments. For this reason, we decided to devote our attention solely to the proposed on-site interface.

The top of NIST's on-site display (reproduced in Figure 3.1) is populated with several buttons that allow the user to choose which kinds of information should be overlaid on a floor plan of the building. The types of information that can be called by each button are as follows: security (e.g., information from security sensors), elevator (e.g., number and location of elevators), fire-fighting equipment (e.g., location of fire extinguishers), utilities (e.g., location of utility shutoffs), building generator (i.e., location of building generator), fire walls (i.e., location and ratings of fire walls), standpipes (i.e., location of interior fire department standpipe connections), hazardous materials (e.g., location of hazardous materials within the building), building hazards (i.e. information of potential hazards to EFRs), and valuable materials (i.e., location of extremely valuable materials) (Davis et al., 2007). When a button is selected, the associated information is symbolically displayed on the floor plan in real-time. Next to the building's floor plan are buttons that allow the user to zoom in and out, pan, and reset the view to the default setting. There are also buttons that allow the user to change floors. On the bottom of the display, three buttons are present: the first displays the building information from the en-route display, the second allows the user to view a log of the alarms and changing building conditions, and the third displays a legend of the symbols used in the map overlay.

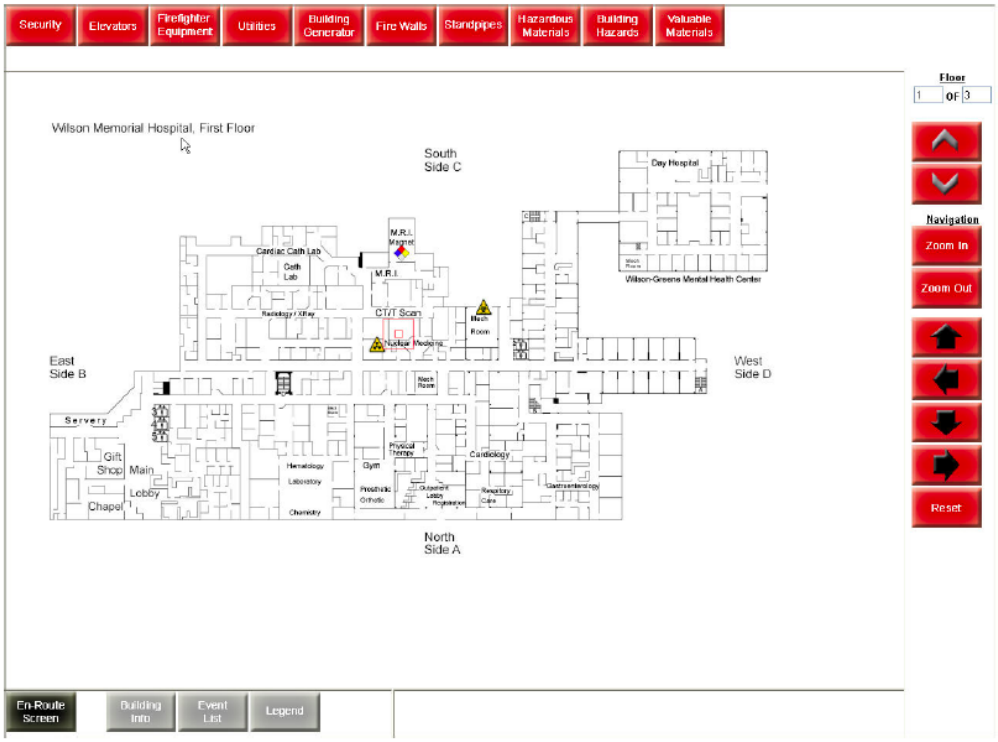


Figure 3.1: On-site Screen of NIST's Prototype Tactical Decision Aid Display (Davis et al., 2007)

This on-site tactical decision aid display interface is successful in many ways. From the description of the on-site interface's features, it is clear that all of the EFR information needs highlighted to be included in our prototype are addressed. Furthermore, overlaying symbols on the floor plan in real-time is an effective way of conveying spatial and temporal information about the building's state. The fact that an event log exists to complement the visual display is also a positive aspect. Finally, the buttons on the right side of the interface allow for easy manipulation of the map.

Despite the successes listed above, there are many ways in which this interface could be improved. First of all, the fact that the user is required to select buttons to toggle which information is displayed on the floor plan requires too much interaction. From a human factors perspective, it is unreasonable to expect an emergency responder to devote significant amounts of time to manipulating the controls of the

GUI while managing and reacting to a developing fire emergency (Dai et al., 1994). However, the rationale behind the toggle buttons, to prevent information overload, is clear and also a valid concern. Our team believes that a better way to avoid information overload would be to simply only display the information most pertinent to conveying the status of the building. In addition, the fact that the event log and the building pre-planning information are not displayed at all times on the interface diminishes the effectiveness of the system. Our team also noted that a function that allowed the user to replay the progression of the emergency would be very useful as more details about a fire's development will lead to more informed tactical decisions. One last improvement would be graphically highlighting on which floors the fire emergency is occurring to allow the user to quickly return to the floors of greatest importance.

3.2.3 Design Goals and Basic Layout

After finishing our assessment of the NIST prototype tactical decision aid display, considering the standard operating procedures of firefighters, and reviewing the available literature on human factors, our team formulated our broad design goals for our prototype system. These goals are as follows:

- Our decision support visualization tool should effectively achieve temporal and spatial contextualization of the information obtained from building sensor systems.
- The interface should be intuitive to view and manipulate. This goal should be accomplished by utilizing standardized symbols and familiar visuals.
- The display should be designed in such a way as to optimize the amount of information that can be gleaned at a glance with particular emphasis placed on the most important data.
- The prototype's design should be modular to allow future technology to be integrated with relative ease.

Once we established these goals, our team created a basic layout that we thought would best meet them. We decided that our interface should be divided into four main components: a static information (or pre-planning) panel, a dynamic information panel, a floor plan with overlaid visual information, and an alert pop-up window. The static information panel was to textually display building and pre-planning information that is difficult to illustrate graphically. The dynamic information panel was designed to textually display a queue of alarms and events as they happen in real-time to complement the visually displayed data overlaid on the floor plan. Like in the NIST prototype display, the floor plan was to be overlaid with symbols and other visual cues to effectively show the state of the building. The alarm pop-up window was devised to ensure that the user is aware of key events such as the fire spreading to another floor. After outlining the basic layout of our prototype's display, a rough mockup of our GUI was created and shown to a panel of incident commanders at the Maryland Fire and Rescue Institute (MFRI) for feedback.

3.2.4 Final GUI Design

Our final prototype GUI was developed based on our design goals, existing building technology, our prioritized list of EFR needs, our assessment of NIST's prototype display, and the feedback we received from MFRI. A screen capture of our final GUI mockup is shown in Figure 3.2 below.

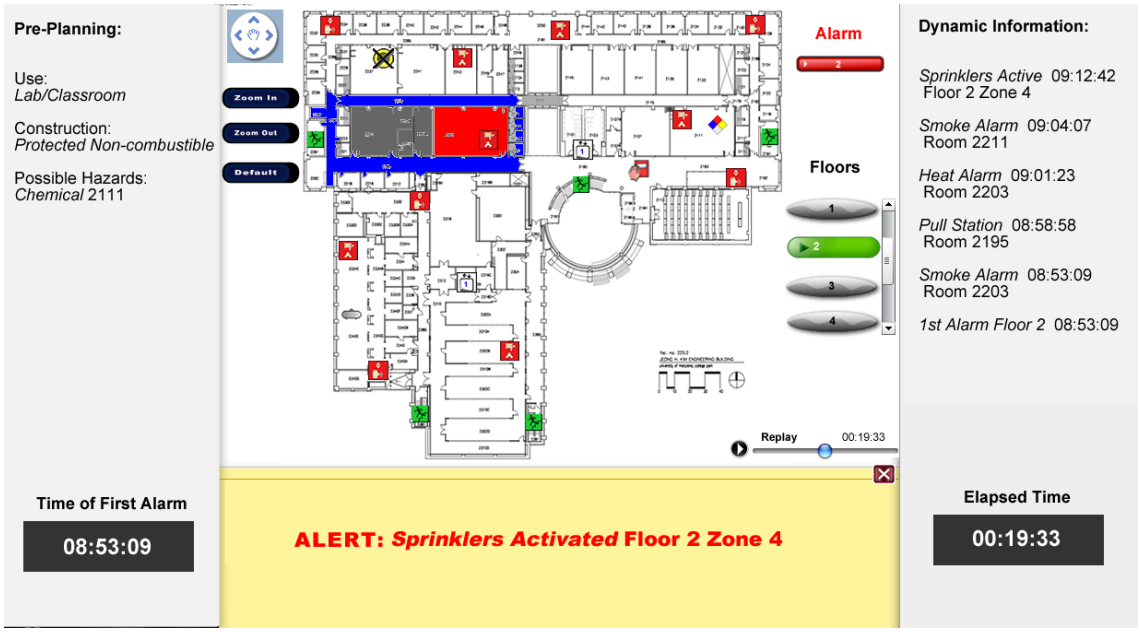


Figure 3.2: Team FFA Final GUI Mockup

The above GUI shows the four main parts in our design: the floor plan display, the pre-planning panel, the dynamic information panel, and the alert window. Located prominently at the center of the design, the floor plan display is the GUI's most important feature; its purpose is to visually display the location of sensors in alarm and other items of importance. Dynamic information is graphically overlaid in real-time on to the static, two-dimensional floor plan. Only the most useful information is included to prevent the display from becoming cluttered. In order to allow the prototype to potentially fit into the standard operating procedures of firefighters seamlessly, NEMA SB 30 icons and NFPA 170 symbols are utilized as deemed necessary. To display the state of a particular location on the map, a color-shading scheme was decided upon in favor of the existing NEMA SB 30 icons. This shading is quite intuitive (e.g., red shading for heat detectors in alarm, grey shading for active smoke detectors in alarm, etc.) and calls attention to the development of the emergency more readily than the existing icons. One other item included in the floor plan display of particular note is the replay control bar. As designed, the

developed visualization tool includes a feature that allows emergency responders to watch an animation on the floor plan showing the progression of the emergency up until that point at several factors faster than real-time. Please refer to Appendix A for a full list of features included in the floor plan display.

The pre-planning and dynamic information panels are located to the left and right of the floor plan, respectively. The pre-planning panel provides a synopsis of the building involved in the emergency to the firefighters before they arrive on the scene. For the sake of simplicity and to prevent information overload, the information displayed on this panel is limited to the use and construction of the building, a list of possible hazards in the building and their locations, and the time of the first alarm. On the other side of the floor plan, the dynamic information panel displays any important alerts as they occur in real-time. All events listed are complete with location and time stamp to ensure that an accurate picture of the emergency situation is painted both temporally and spatially. In addition to complementing the floor plan display in conveying the progression of the emergency, this information allows a log documenting the emergency situation to be generated by the system with minimal effort. The dynamic information panel also shows the amount of time that has elapsed since the beginning of the emergency situation. Refer to Appendix A for the full features lists for these two panels.

One final main feature of the prototype's GUI is the pop-up alert window. When an event of particular importance occurs (such as the fire spreading to a new floor), a pop-up window appears below the floor plan accompanied with a sound and does not disappear until acknowledged. This feature is designed to bring events of particular importance to the attention of the incident commander despite the frenzy of the emergency situation. See Appendix A for further detail.

3.3 Software Implementation

The task of implementing the software consisted of accessing and parsing sensor data and displaying the information according to design specifications. Accordingly, the largest modules of the parser are 1) the data parser, 2) the building representation and 3) the interface. The data parser processes the raw data obtained from the sensor panel and updates the building representation. The building representation organizes the received information to make it easy to find the state of all sensors on a given floor at a given time. The interface displays the state of the building at the floor and time selected by the user. The overarching modules can, in turn, consist of a number of sub-modules. The desired behavior of the data parser and the visualization interface is already well-defined: the parser needs to accept the protocols used by building sensor systems, and the design of the visualization interface was described in the previous section.

Throughout this implementation phase, we focused on developing software that could be used to evaluate the effectiveness of the proposed approach to the problem of data visualization. In the prototype, some functionality that would be necessary for a full-featured application was omitted. We chose not to provide solutions to previously solved problems, such as software distribution questions and the networking necessary to make the obtained sensor data remotely available.

3.3.1 Software Design Choices

The visualization display interface was implemented in Java. The Object Oriented paradigm employed in the programming environment Java was conducive to a logical representation of building information: the building, floors, and sensors were represented by separate object classes. More importantly, Java's type polymorphism – a feature that enables a uniform interface to handle different complex

data types – simplified the task of writing modular code.

Since one of the goals of the project was to create a system that could interface with building sensor systems regardless of type or manufacturer, the software was designed to be able to receive input from a variety of different sources. For each protocol used, a separate parser module is needed, but the other modules can be used without modification. We wrote parsers for data received over RS-232 and BACnet, since those were the two transmission methods used by the hardware donated to our team. Additionally, we wrote a parser for Fire Dynamics Simulator output for testing purposes.

3.3.2 Annotation Tool

Whereas the implementation of the data parser and the interface is simply done according to specifications, the building representation required further consideration. It was necessary to associate the sensor labels used by the annunciator panels with their locations on the floor plan. Also, it was necessary to know the areas of effect for each sensor in order to highlight the appropriate parts of the floor plan when the sensors go into alarm. Unfortunately, this information is rarely stored in a way that would allow us to directly use it in the software, so it is necessary to input this information by hand for each building. Since this is a time-consuming and error-prone task, we decided to develop an annotation tool, a separate application to simplify the task of entering building information by hand.

The annotation tool uses an interface that is very similar to our decision support tool's interface. The user marks the sensor location and area of effect directly on the floor plan. Using a familiar graphical interface to enter information makes it possible to quickly and accurately annotate the floor plan. The annotation tool also allows for text-based information to be entered for any sensor, building floor or for the entire building. The text information is organized into parameters and values

to allow comparison of different sensors or floors based on the values entered.

The variety of data that can be entered makes the annotation tool a robust pre-planning utility that can be used for many different purposes. The annotation is stored in XML format, which is well-defined, robust and can be handled by many existing tools. We propose that the annotation tool or similar pre-planning software be used to standardize the way that building information is stored and accessed.

3.4 Hardware Testing

3.4.1 Overview

The goal of the hardware testing was to demonstrate that the decision support tool is capable of processing actual sensor inputs as well as display the information accurately. We used mockups of fire sensor equipment donated to our team by Honeywell and SimplexGrinnell to obtain these inputs.

The hardware setup consisted of two different FACPs, one Notifier NFS-640 and one Simplex4100U, both of which are approved for use in buildings on campus. Each FACP was hooked up to the following sensors:

- 3 photoelectric smoke sensors
- 3 ionization smoke sensors
- 3 heat sensors
- 1 flow switch
- 1 tamper switch
- 1 manual pull station

The FACPs were in turn connected to a laptop. The laptop used to run the decision support software had the following key specifications:

- Processor: Intel Pentium M, 1.73 GHz

- Harddrive: Fujitsu MHV2060AH ATA Device, 5400 RPM
- RAM: 1024 MB at 266 MHz

It is possible that due to the age of the parts, the specifications of the laptop contributed to delay in the response time of the decision support tool.

Using the sensors available to each FACP, a floor plan was devised placing each sensor on one of three floors. The floor plan itself, shown below, is a basic hallway with three rooms, a door on the first floor and a stairwell leading up to each floor.

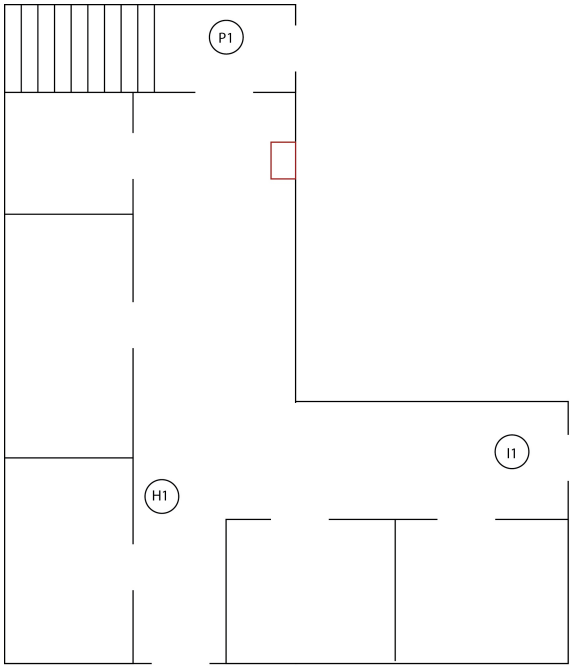


Figure 3.3: A mockup floor plan for real lab hardware to simulate.

Using the devised sensor layout, the floor plan was then formatted for input using our developed annotation tool. This annotation identified each sensor on the floor plan and its approximate area of effect that will be visually displayed on the system should the sensor go into alarm.

In order to use the hardware to simulate an emergency scenario, sensors were manually triggered according to a pre-written plan of events. This pre-written plan,

provided in Appendix B, established an emergency scenario, identifying what sensors go off at what time steps. By controlling the input to the sensors, the output of the system can be analyzed to identify if the real sensor hardware will perform as expected. For this experiment, it is expected that the system will accurately denote which sensors are going off on a floor plan and that there will be little lag time (less than ten seconds) between when a sensor goes into alarm and when it is displayed as being in alarm by the prototype system.

3.4.2 Panel Output Protocols

The Simplex 4100U panel and the Notifier NFS-640 panel are significant because they both have BACnet gateways installed. The BACnet gateway allowed for the output of each system to be standardized, so that sensor states from each system are identical. BACnet transferred this data to the system over an Ethernet cable through a network card on a computer.

The default settings on the BACnet card provided with the Simplex panel were not configured to monitor analog values. The card was reprogrammed to specifically monitor analog values of sensors over time. Heat sensors that measured room temperature would then display their current readings in degrees Fahrenheit, and smoke sensors would read their values on the obscuration of the air.

The output of the sensors via BACnet now become the input to the system. As previously mentioned, BACnet is convenient because it is a standard protocol that is installed on a variety of FACPs across several manufacturers in addition to Honeywell and Simplex Grinnell. However, in the interest of thorough experimentation, another communication protocol was used – RS-232. RS-232 is a well-established way to transfer serial data from one device to the next. Virtually every FACP has RS-232 output; serial data transfer is the primary method for communicating with the printer. With very basic wiring, the printer data stream can be tapped and fed

into a computer's serial data port or a USB port with a DB9 serial data adapter. Because of the simplicity of serial data provides, and the ubiquity of RS-232 output from FACPs, RS-232 was also explored as a method for taking the output of the FACPs and making that the input for the decision support tool.

3.4.3 Hardware and Scenario Mockup

Each FACP in the mockup was hooked up to twelve different detectors and modules in a Type A loop, meaning a single wire connected each sensor in a loop that began and ended at the FACP. Each flow switch and tamper switch was connected to manual toggle switches, allowing both the manual pull stations and the tamper and flow switches to be instantly activated as necessary. Each photoelectric and ion smoke sensor, as well as the heat sensors, was activated by applying a magnet at a specific point near the LED light. The sensors triggered on application of the magnet; however, there was a significant delay between the application of the magnet and when the sensor actually went into alarm. It also appeared that the delay was different from sensor to sensor. Brief experimentation was required to obtain the average response time of each sensor to ensure reliable system input. Appendix C outlines the testing procedure and results for this experimentation. Additionally, for the Simplex 4100U panel, analog values were measured by the sensors. The specific output from the FACPs includes: time of sensor alarm; what sensor on the loop went into alarm, identified by its loop address; its custom ID (inputted specifically to match with the system input); the alarm state of the sensor (supervisory warning, a trouble signal, or in a fire alarm state); and what the analog value of the sensor is, if it has one. In order to test for analog values (which magnet tests do not affect), canned smoke was applied in specified bursts to each smoke sensor, and a heat gun was placed beneath each heat sensor in order to trigger the sensor and measure changes in analog readings. With these methods, test inputs could be performed in

accordance to a testing plan.

A basic description of the scenario is: at time equals zero, the fire starts on the second floor in the bottom left corner room on the floor plan (shown earlier in Figure 3.3). A hot smoke layer soon forms on the ceiling of the room and begins to exit into the hallway through an open door. As the hot gases move over the second floor heat detector, the heat detector goes into alarm roughly sixty seconds from the start of the scenario, and the audio/visual alarms have activated to let personnel in the building be aware of the emergency. The smoke layer expands quickly through the small hallway and the second floor ionization smoke sensor detects the smoke layer at time equals ninety seconds. At this point, personnel have begun evacuating and going down the stairwell. As one person moves to the fire escape, they pull the manual pull station on the way out at time equals 120 seconds. The smoke layer on the second floor now begins to extend all the way to the end of the hallway towards the stairwell, and the photoelectric smoke detector on that floor goes into alarm at time equals 150 seconds. Because the stairwell doors have been left open, the smoke enters the stairwell and rises, entering the third floor. Because of the increased burning conditions of the initial fire, the smoke layer is hot and dense, thus moving up the stairwell at a rapid rate. At time equals 180 seconds, the third floor photoelectric smoke detector has been triggered. By time equals 210 seconds, the third floor ionization smoke sensor has gone into alarm as the smoke rapidly fills the third floor. At time equals 240 seconds, the third floor heat sensor has been triggered. The scenario has concluded at this point. At time equals 240 seconds, the third floor heat sensor has been triggered. The scenario has concluded at this point.

3.4.4 Procedure

In order to carry out tests of the scenario, two people were required. The scenario was experimentally demonstrated as follows:

1. At the start of the test, a stopwatch was triggered by Person 1.
2. At each time point that a sensor was activated in the scenario, a sensor was triggered on the sensor mockup by Person 2. Smoke and heat sensors were activated with the application of a magnet; tamper and flow switches, as well as manual pull stations, were manually triggered by their respective toggle switches.
3. (a) In order to accurately follow the scenario, the magnets were applied by Person 2 after a cue from Person 1. The experimentally determined lead times were taken into account. These lead times are provided in Appendix C.
   (b) For the Simplex Panel, canned smoke was applied in a five second burst to the smoke sensors, and a small flame was placed under each heat sensor according to appropriate trigger times. The same call-out procedure by Person 1 was employed.
4. The data output by the FACP as a result of the sensor triggers were sent to our prototype using the appropriate communication protocol (BACnet or RS-232).
5. Our prototype interpreted the data and displayed the information on the user interface.
6. A video of the decision support tool was recorded as it responded to the sensor input.
7. The video was then compared to the pre-designed simulation (which was artificially entered to have zero delay).

.

3.4.4.1 Analysis

We compared the video output from the hardware mockup with the pre-designed video of the scenario. This comparison achieved two things: 1) whether or not the sensor technology and communication protocols worked appropriately with

the designed software, and 2) it identified if there were significant delays between the hardware response time and the ideal case of no delay. Comparing the video output to the pre-designed scenario allowed us to determine if the system's performance was acceptable. The analysis is qualitative in nature due to the uncertainty associated with the triggering of each sensor.

3.5 Fire Dynamics Simulator Tests

3.5.1 FDS Overview

Fire Dynamics Simulator (FDS) is a program developed by NIST used for fire dynamics simulation ("Smokeview (Version 5) - A Tool for Visualizing Fire Dynamics Simulation Data Volume I: Users Guide", n.d.). The input files for this program are written in Fortran. Information contained within the input file can include simulated building obstructions, ignition sources, sensors, and the mesh that is computed. Sensors that are readily available in FDS include smoke detectors, heat detectors, and sprinklers. The device properties can be manually set for the different sensors (for our scenarios, the values from the FDS user guide were utilized). Smokeview is software that was developed by NIST that displays the numeric solution found by FDS graphically. By default, Smokeview will display the obstructions written in the input file, with the ability to show the movement of the smoke and other particles within the simulation. Smokeview is also capable of displaying temperatures at any point within the simulation. Simulating in FDS and displaying the results in Smokeview eliminated the need to test the prototype in an actual building, since performing such a test would not be feasible.

3.5.2 Simulations

A simple four-room case was used to demonstrate that the prototype could process the data. This simulation consisted of four rooms connected by a hallway in the middle. The rooms contained sprinklers, smoke detectors and heat detectors. This simulation was created to establish what information was output by the FDS program and to check that the devices and fire source worked as intended. The simulation's results were also employed as a preliminary test of the prototype's capabilities, ensuring that the prototype was capable of handling data generated from FDS.

FDS simulations of greater complexity were utilized to test the final functionality of our prototype decision support tool. These simulations included a multilevel case and several cases modeling a wing of the James M. Patterson (JMP) building on the University of Maryland campus. Initially, the simulations were run in the single processor mode of FDS with computers utilizing Intel dual core processors to check if the obstructions in the scenarios were dimensionally correct. Then the scenarios were run in full on a dedicated Linux server.

Each scenario contained a single continuous burner that increased heat over time. This choice of fire source simplified the simulation, allowing for continuous smoke to set off the different sensors and the cases to not be dependent on location and quantity of combustibles. The intent of the FDS simulation was to show that the different sensors being triggered into alarm at different times could be displayed by the prototype. It was not about the realism of the fire simulation itself.

The multilevel case contained three rooms stacked up on each other with a single smoke detector, heat detector and sprinkler on each floor. The floors were connected by a staircase on the side of the building. The design of the obstructions in the input file was created by first sketching out the floor plan and then translating that information into the proper obstructions. The multilevel simulation was created

to test the prototype's ability to handle multiple floors. If the prototype could not switch between floors or accurately display the situation in this scenario, the prototype would need to be modified. The scenario was run for thirty minutes in simulation time, with the burner in the bottom level to allow the smoke to rise to the other levels.

The JMP simulations were created to demonstrate how the prototype would function when faced with larger building sizes and more complex geometries. This simulation contained multiple rooms and hallways in order to provide a more complex path for the smoke to flow through. The floor plan for the wing of JMP was drawn in AutoCAD and imported into FDS code through an AutoCAD plugin. The obstructions that were imported were utilized as the base floor plan for all the input files.

Multiple sensor placement variations on the same floor plan were used to establish how sensor density affected the display capabilities of the prototype. Originally there was to be three sensor setups: one representing the minimal amount of sensors, another representing the actual as-built setup of JMP, and the last containing a heat and smoke detector in every room. For this test, the maximum sensor density and the as-built setup of JMP were the only two variants used. The reason for this decision is that, for commercial buildings, smoke detectors are not typically required as long as sprinklers are present (*NFPA 101: Life Safety Code*, 2009). The simulation was fitted with a single sprinkler, representing a flow switch for the entire floor; multiple sprinklers were not placed since the wing of JMP modeled is a single zone for the sprinklers as built. This use of one sprinkler also alleviated issues with the added complexity of water droplets lengthening the simulation's computation time.

3.5.3 Testing Procedure

There were two different objectives for testing with simulations. The first was to determine if the prototype could accurately show the scenario. The results of the multi-level case and the regular JMP simulation were used for this objective. The second test was to determine the sensor density for which our prototype decision support tool performs most effectively. The placement variations described earlier for JMP were used for this objective. First, the prototype was set up to use the FDS output in place of sensor data. This setup took several steps:

1. Annotate the floor plan used to design the FDS simulation with the sensor location and areas of effect.
2. Label the sensors so that the sensor labels match the data field labels in the FDS output.
3. Set threshold values for each sensor type to determine when the sensors go into alarm.
4. Parse the FDS output to read in sensor values and find when each sensor goes into alarm.
5. Use the timestamps in the FDS output to simulate sensor activation at appropriate time intervals.
6. Pass the sensor activations as input to the visualization software.

As the prototype was displaying the changing state of the FDS simulation, a video of its display was created by utilizing an existing function within Smokeview. This function exported the individual frames that Smokeview displayed as jpegs. Those jpegs were then spliced together into a single video of the scenario. The time between frames was determined by the numerical solutions found by FDS. Both displays have timestamps located on the screen; timing comparisons were gauged based on those stamps. Pictures were taken at one minute intervals in order to examine whether or not our prototype accurately displayed the progression of the fire scenario.

3.5.3.1 Test 1

The first test compared the information given by the prototype to the changing state of the simulation. The test was to determine whether the progression of the smoke movement, representing our fire scenario, was followed by the sensor activations in the prototype. If a sensor had been activated in the simulation and not the prototype, it was noted. A visual comparison of the activations was assessed at this point. The videos showing the display of both the prototype and the simulation were placed side by side. At one minute intervals, the two videos were compared to check if the prototype was accurately describing the scenario displayed in Smokeview.

3.5.3.2 Test 2

For test 2, JMP was annotated in two different ways: a full sensor layout and a sparse sensor layout. The full sensor layout was identical to the one used in Test 1; each room and hallway was equipped with a heat sensor and a smoke sensor. These sensors were physically located close to the center of each room. There was also a single flow switch monitoring the sprinklers for the whole floor located close to the fire source. In the sparse sensor layout, only the major rooms (1, 2, 3, 4, and 26) were equipped with sensors. The sensors were positioned identically to the corresponding sensors in the full case. Our original plan was to analyze the results of this test at one-minute intervals. However, the sparse case does not change very frequently. Four minutes proved to be a more appropriate interval to describe the development of that scenario.

Chapter 4

Results

4.1 Hardware Test

We tested our prototype with two different fire sensor systems. The purpose of these tests was to demonstrate that existing hardware can support the proposed visualization system. A second goal was to determine what limitations current hardware imposes on the system and how those limitations may be worked around. Please refer to Section 3.4.4 for the procedure of each test.

4.1.1 Honeywell Mockup Test

The Honeywell mockup test was performed by tapping the RS-232 port on theNFS-640 panel to obtain streaming information about what alarms were triggered. At t = 0, the first alarm, H2, the heat sensor on the second floor, went off. There wasa four second delay between when the sensor went off and when the GUI actually displayed the alarm. The prototype did react immediately and kept track of thefour seconds it took to display the data, so the time elapsed clock was still correct. The ion smoke sensor I2 went o approximately six seconds early at t = 29 due to thevariation on trigger times associated with activating the sensors by magnet. After that, the manual pull station was pulled at t = 62 and the screen shifted to lookat the first floor. The second floor was then manually reselected. The photoelectric smoke sensor P2 went off approximately on time at t = 88. Next, the photoelectricsmoke sensor P3 at the top of the stairwell on the third floor went off at t = 127. Once again, the four second delay can be seen here as the system jumps to a new floor, but the clock still follows the correct time after the four seconds.

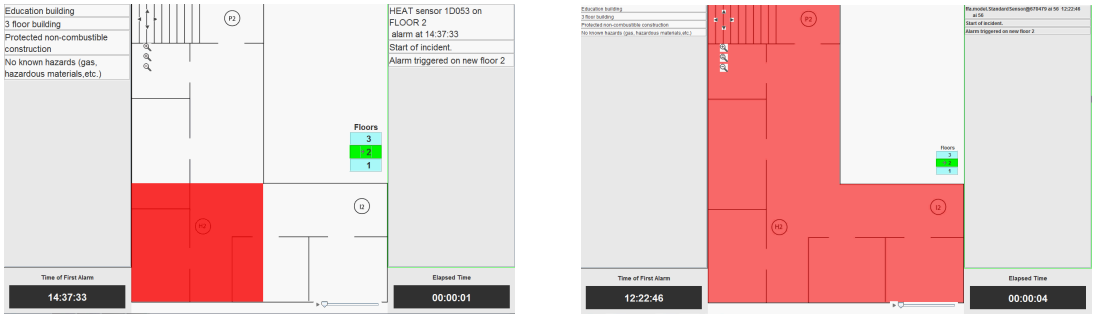
Finally, the third floor ion smoke sensor I3 triggered 12 seconds late at t = 145, and the third floor heat sensor H3 triggered at t = 174.

4.1.2 SimplexGrinnell Mockup Test

The SimplexGrinnell mockup test was performed by programming the BACnet card on the Simplex 4100U panel to monitor changes in analog values of smoke and heat sensors. As a result of this programming, sensor states could not be obtained and only analog values could be read. At t = 0, the first alarm, H2, the heat sensor on the second floor, went off. There was a four second delay between when the sensor went off and when the system actually displayed the alarm. As with the Honeywell test, the system did react immediately and kept track of the four seconds it took to display the data, so the time elapsed clock was still correct. The ion smoke sensor I2 went off approximately 5 seconds late at t = 35 due to the variation on trigger times associated with triggering the sensors by magnet. The manual pull station was not pulled because the BACnet gateway was programmed to monitor changes in analog data only; manual pull stations do not provide analog data and polling such a device for analog data is an invalid call in the Simplex 4100U panel that shuts off the BACnet card. The second floor photoelectric smoke sensor P2 went off 4 seconds early at t = 86. Next, the photoelectric smoke sensor at the top of the stairwell on the third floor went off fifteen seconds late at t = 135. Once again, the four second delay can be seen here as the system jumps to a new floor, but the clock still follows the correct time after the four seconds. Finally, the third floor heat sensor H3 triggered nine seconds early at t = 159, and the third floor ion smoke sensor I3 triggered nineteen seconds late at t = 169.

4.1.3 Fire Progression

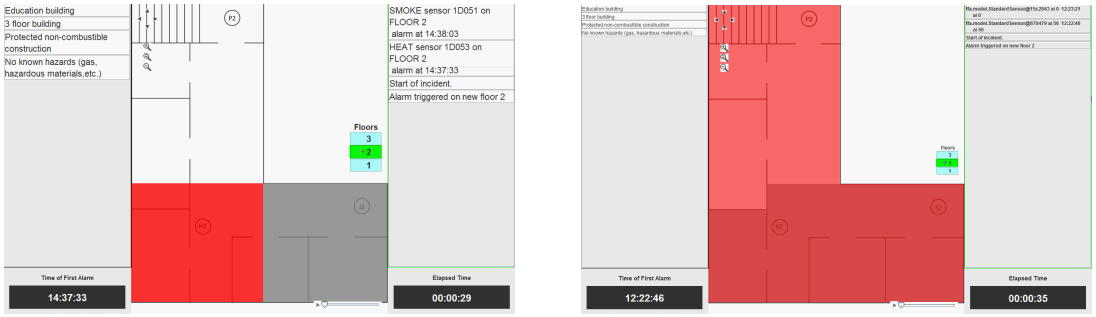
Key screenshots from the hardware tests have been selected and are presented here. The selected screenshots show what the prototype displayed immediately after each sensor was triggered. From these screenshots, a story of the fire has been constructed as it would be understood by an observer who is familiar with the system and has no information about the fire other than what is presented in the screenshots.



(a) Honeywell (b) Simplex

Figure 4.1: Frame A

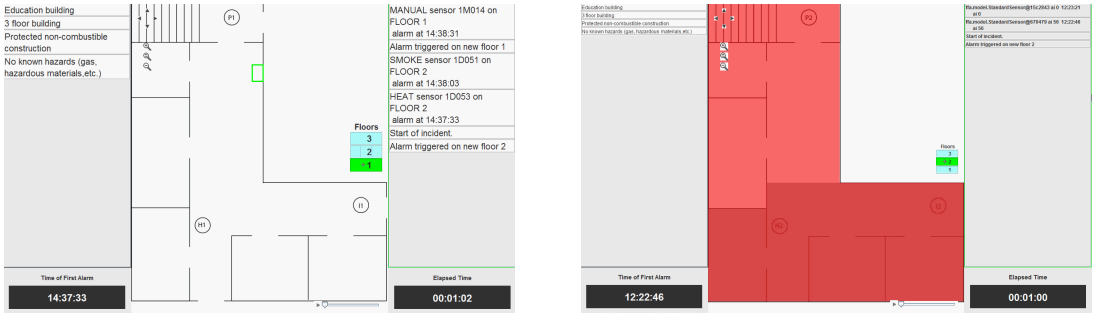
The above screenshots show that the fire originated on the second floor. The fire probably started with a clean-burning material that did not trip the smoke sensors. In the NFS-640 annotation file, only the corner appears in red, identifying the zone of effect for the heat detector. In the 4100U annotation, the zone of effect for the sensor is the whole floor, making it more difficult to visually understand where the fire initially started, but still clearly shows what floor where the incident began.



(a) Honeywell (b) Simplex

Figure 4.2: Frame B

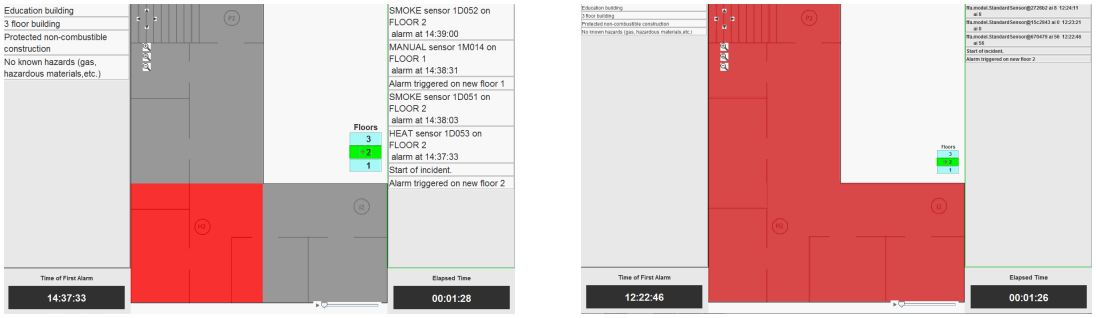
The east wing of the second floor is filling with smoke. The fire is continuing to spread and probably originated in the east wing.



(a) Honeywell (b) Simplex

Figure 4.3: Frame C

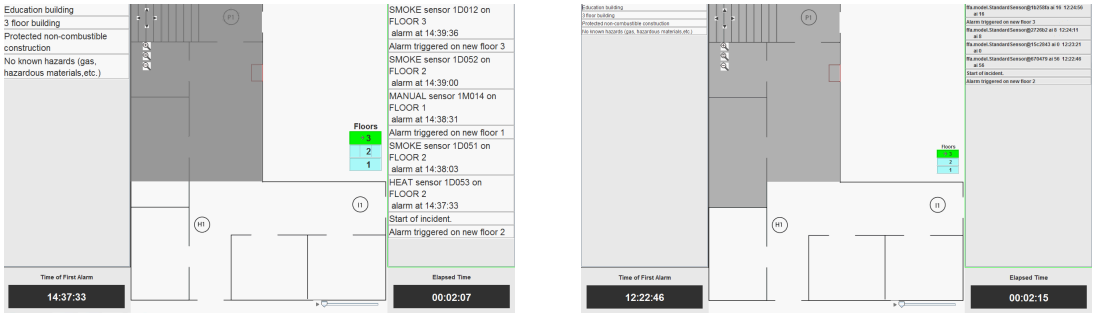
In the Honeywell test, the manual pull station on floor one has been activated. This activation indicates that there are people on floor one, and evacuation of the building has begun. In the Simplex test, the manual pull station was not pulled due to difficulty reading non-analog alarm states via BACnet, as described earlier.



(a) Honeywell (b) Simplex

Figure 4.4: Frame D

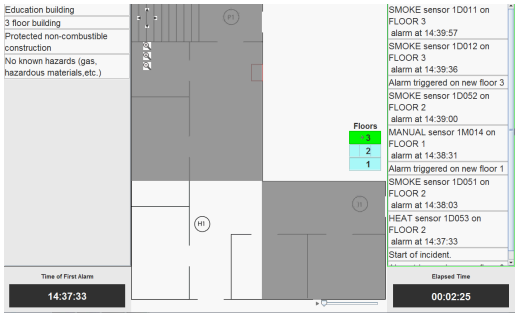
In the Simplex test, we can see that smoke has spread throughout the second floor and has reached the stairwell. In both trials, the text-based display on the right side of the screenshots shows that a new sensor has been activated. However, in the Honeywell test the first floor is still being shown as a result of the manual pull station being triggered, so it had to be manually switched to the second floor.



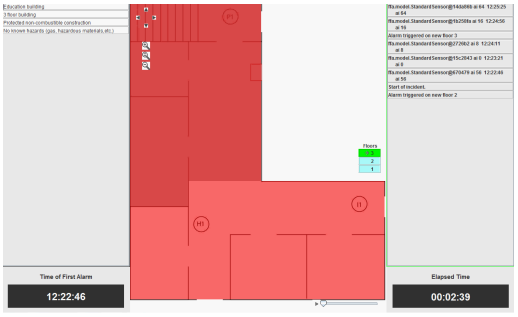
(a) Honeywell (b) Simplex

Figure 4.5: Frame E

The smoke has spread up the stairwell from the second floor to the north wing of the the third floor.



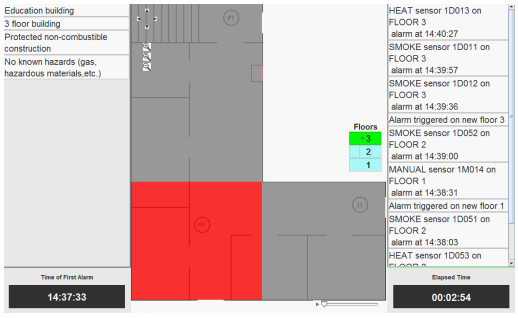
(a) Honeywell



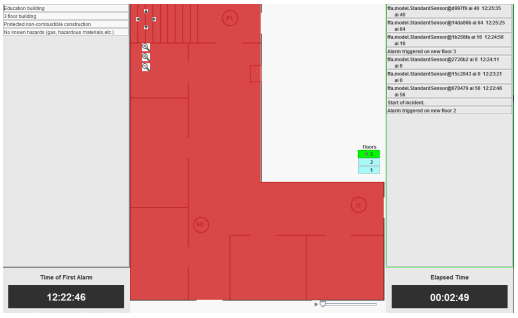
(b) Simplex

Figure 4.6: Frame F

Smoke has spread throughout the third floor.



(a) Honeywell



(b) Simplex

Figure 4.7: Frame G

Material on the third floor has probably ignited.

4.1.4 Analysis

The tests have shown that the system can interface with existing Honeywell and Simplex hardware, though with some difficulty. It is easy to track the progression of the fire by looking at the displayed information assuming that there are multiple sensors present in the building. The text-based display on the right side of the screen should be improved for readability and display information such as sensor floor and sensor type.

An observation to note in the results is that in the Simplex test, there were a few occasions where the prototype system detected an alarm before the Simplex

4100U panel did. This anomaly is most likely due to the fact that the BACnet plugin for the system monitors the changes in analog values and detects alarms based on the threshold values of the sensors being reached. The 4100U most likely has more sophisticated measures that prevent it from picking up nuisance alarms, thus the slight delay in setting off the alarm to be more confident that it is real.

In addition to the output of the prototype system, the normal output of the printer port from the Notifier NFS-640 Panel was taken and is shown below:

```
ALARM: HEAT(FIXED)   DETECTOR ADDR 1D053                Z001    04:15P
    012411 1D053

ACKNOWLEDGE                                                      04:16P
    012411 Mon

ALARM: SMOKE (ION)   DETECTOR ADDR 1D051                Z002    04:16P
    012411 1D051

ACKNOWLEDGE                                                      04:16P
    012411 Mon

ALARM: MONITOR       MODULE ADDR 1M014                   Z004    04:16P
    012411 1M014

ACKNOWLEDGE                                                      04:17P
    012411 Mon

ALARM: SMOKE(PHOTO) DETECTOR ADDR 1D052                  Z003    04:17P
    012411 1D052

ACKNOWLEDGE                                                      04:17P
    012411 Mon

ALARM: SMOKE(PHOTO) DETECTOR ADDR 1D012                  Z003    04:18P
    012411 1D012

ACKNOWLEDGE                                                      04:18P
    012411 Mon

ALARM: SMOKE (ION)   DETECTOR ADDR 1D011                Z002    04:18P
    012411 1D011

ACKNOWLEDGE                                                      04:18P
    012411 Mon

ALARM: HEAT(FIXED)   DETECTOR ADDR 1D013                Z001    04:18P
    012411 1D013
```

As can be seen in the default configuration of the panel, simply reading the alarm states as they go off makes it hard to gain a visual perspective of the scenario, and harder to get a visual picture of the progression. Custom labeling of the sensors could aid in this process, since this is a small building with few sensors, but in larger configurations, it would be much more difficult to get a full visual picture of how the fire started and how it has progressed. The visual nature of the prototype, as well as its replay feature, can potentially aid in providing an image of the spread of the fire in a building.

4.2 FDS Testing of GUI

4.2.1 Test 1 Results

A qualitative analysis of the correlation between the fire activity in the FDS simulation and the GUI of our decision support tool prototype was undertaken. Images from the FDS and the GUI were compared at one-minute intervals to assess the accuracy of the view of fire progression through the decision support tool. This assessment was done by matching rooms with smoke visible in Smokeview with rooms showing smoke detector activations in the GUI. The timeline used was based on the SmokeView simulation. The GUI time-of-first-alarm is approximately 65 seconds after the Smokeview timeline begins. Refer to Section 3.5.3 for the full procedure of this test.

4.2.1.1 JMP Scenario

A visual assessment of the decision support tool's display for the JMP simulation in comparison to the SmokeView images showed only two incongruities. At

the three-minute mark, smoke was visible at a low concentration in room 9 but had not yet been detected by the decision aid display. This discrepancy can be seen in figure 4.8, where there is no indication on the GUI display that an alarm has activated – an entry in the right dynamic panel or graphical change in the floor plan overlay. However, by the four minute mark, the GUI showed smoke in room 9.

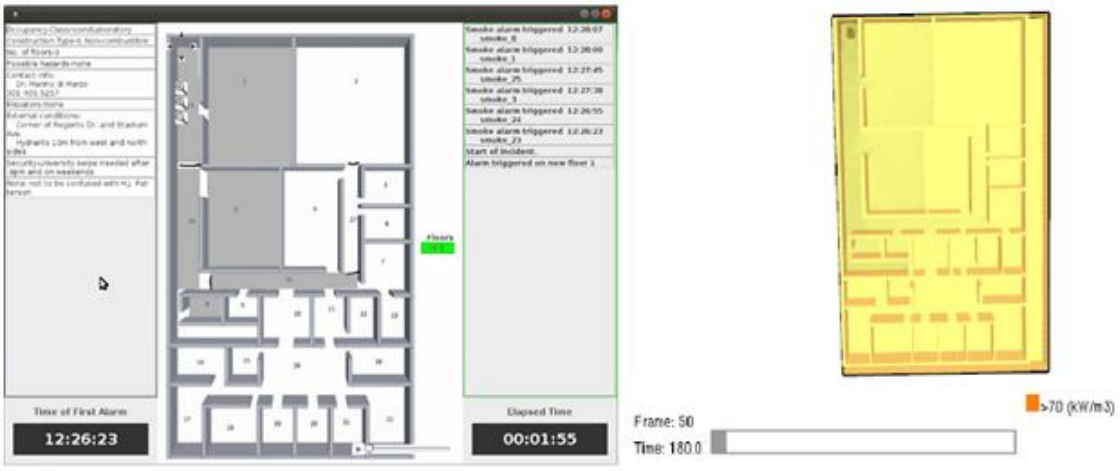


Figure 4.8: Side by Side comparison of JMP simulation to GUI at the three minute mark

At the seven-minute mark, room 22 showed smoke in Smokeview but not in the GUI. Again, by eight minutes, the GUI indicated the alarm. Also by eight minutes, the prototype's GUI showed smoke in every room. It is at this point that visual assessment of smoke progress was halted because the simulation would primarily show an increase in intensity of smoke concentration. However, the GUI is not able to show these changes in intensity once all the smoke detectors are activated; it is only capable of displaying whether or not a sensor is triggered, not the analog sensor output. Figure 4.9 displays the simulation at the eight minute mark while figure 4.10 shows the simulation at the twelve minute mark. As evidenced in the figure, the display of activated smoke sensors on the GUI saw no change. However, in the simulation, all the rooms have an increase in smoke concentration. A visual assessment of heat progress was not carried out.

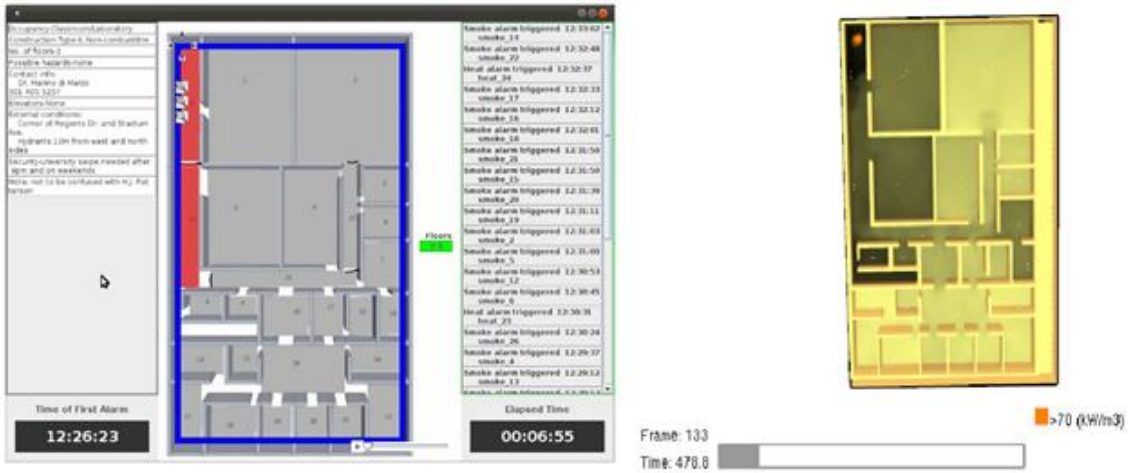


Figure 4.9: Side by Side comparison of JMP simulation to GUI at the eight minute mark

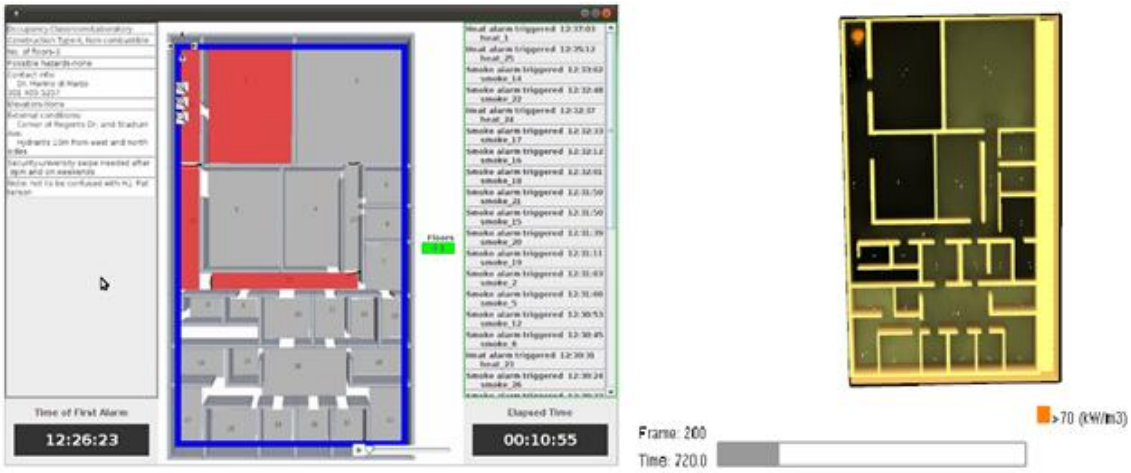


Figure 4.10: Side by Side comparison of JMP simulation to GUI at the twelve minute mark

The discrepancies described above are likely due to the threshold values for the smoke detectors. Smoke can be seen in the Smokeview simulation before the alarm triggers. Each alarm triggers somewhere within the fairly large time gap of one minute. For room 9, the GUI activation appeared seconds after the image that was used as the comparison between the GUI and the simulation. For room 22, the GUI activation time was at 7.5 minutes. Because the activation times are between the minute markers, these discrepancies do not indicate any error in the decision

support tool. A visual assessment of the smoke progression in the JMP simulation shows a successful rendering of the smoke progression in the building.

4.2.1.2 Multilevel Scenario

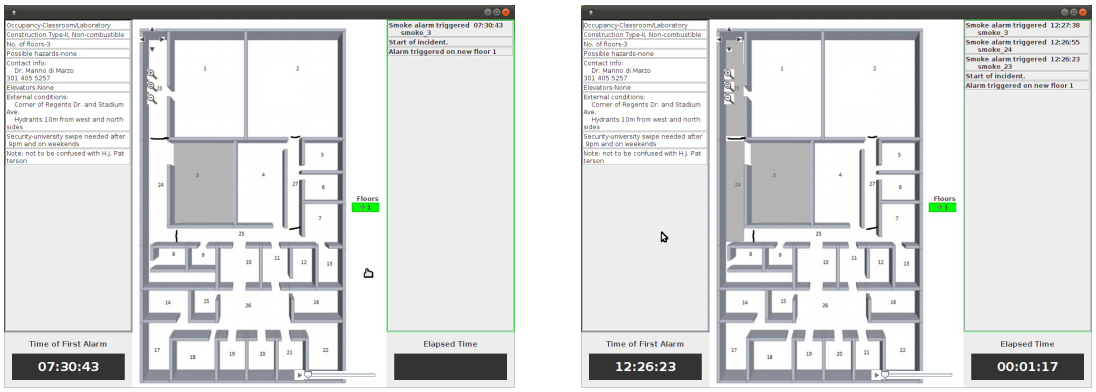
In the multilevel simulation, the visual assessment shows a mostly accurate depiction of the smoke progression. At three minutes, smoke is visible in the stairs, but no detectors are located there in the simulation. At four minutes, smoke can be seen entering the second and third floors, but has not encompassed the entire room, so it is reasonable that the smoke alarm has not yet triggered. At five minutes, both the first and second floors show smoke detected in the GUI. However, the third floor has a significant quantity of smoke but the smoke detector has not yet been activated. By looking at the activation times, the third floor smoke detector triggers twenty seconds after the second floor. This occurrence is somewhat surprising because from a horizontal view, the smoke on floor three appears to be more highly concentrated than floor two. The height of the sensor on the third floor is 0.1 m higher than the other sensors and located further from the door than on the other two floors, likely explaining the delay in sensor activation on the third floor. However, the discrepancy is only five seconds. At six minutes, all of the floors show smoke in both the GUI and Smokeview and the visual assessment was completed. Considering the minute-long intervals, the visual assessment concludes that the GUI is sufficiently accurate in portraying the smoke progression across multiple floors.

4.2.2 Test 2 Results

The second test determined how sensor density affected the prototype performance. This test involved comparing the videos of the two different JMP sensor placement cases previously described. The test examines the difference between sensor densities to see if more visual information can be obtained from buildings

with more sensors. Refer to Section 3.5.3 for the full procedure of this test.

4.2.2.1 Start of incident

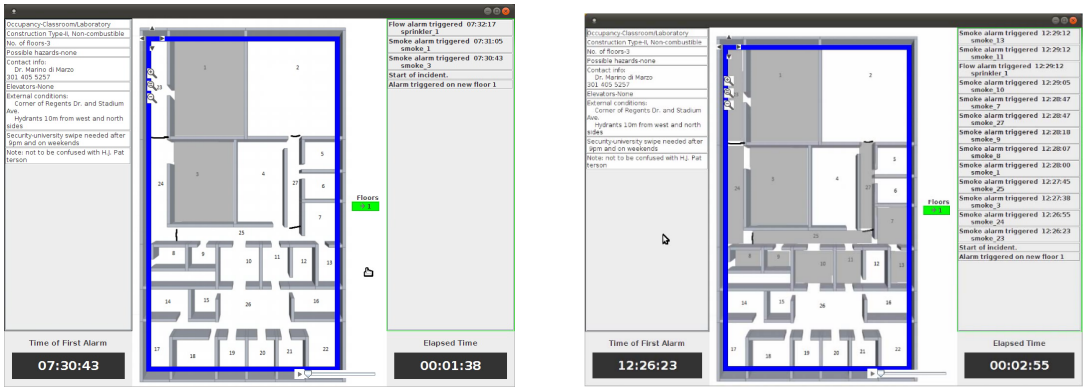


(a) sparse (b) full

Figure 4.11: incident start

The sparse sensor layout detects the fire over a minute after the full sensor layout has gone into alarm. Room 3, where the initial sensor on the sparse layout was located, is still fairly close to the fire source. Due to this proximity, locating the initial source from the data obtained from the sparse sensor layout is still possible. However, the requires an inference that is not necessary when viewing the data from the full sensor layout.

4.2.2.2 Flow switch activation: Four minutes from start of fire



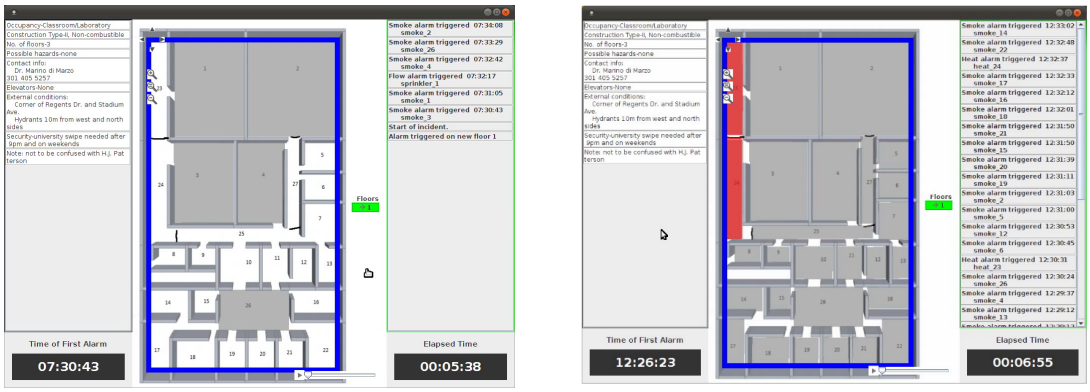
(a) sparse (b) full

Figure 4.12: flow switch activation

By this point, the full layout shows that the smoke has spread across the building. The sparse layout shows a very different picture, with sensors in alarm only in the top left corner of the building. The sparse case does not correctly represent the spread of smoke. The movement of smoke is more clearly seen by the full sensor layout.

About half of the building's smoke sensors are in alarm in both scenarios – two out of five are in alarm for the sparse case, and twelve out of twenty-seven for the full case. However, without an indication for which rooms have sensors, the sparse case appears to under-represent the extent to which smoke has spread. The fact that only two sensors are in alarm for the sparse case could be deceiving.

4.2.2.3 All smoke sensors in alarm: Eight minutes from start of fire



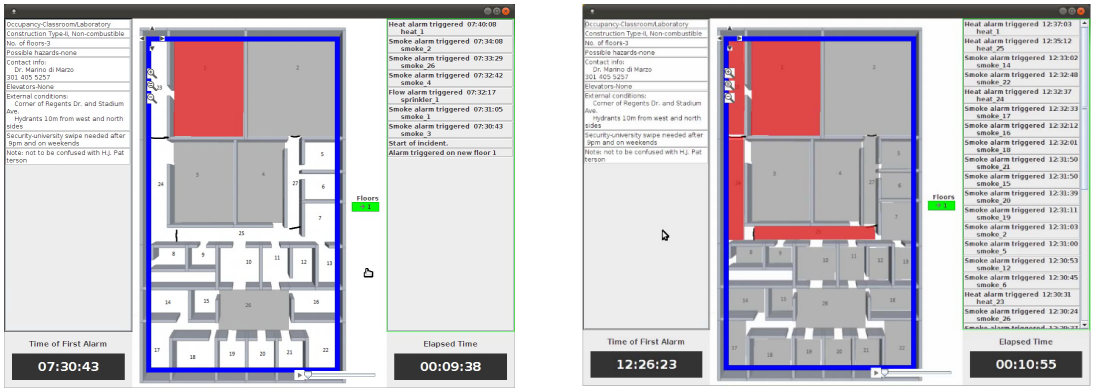
(a) sparse (b) full

Figure 4.13: all smoke sensors in alarm

In both sensor layouts, all smoke sensors are now in alarm. However, on the display showing the sparse layout, less of the area is marked to indicate smoke. The effect is quite pronounced; at a glance, the sparse case appears to be only half-filled with smoke, while with the full case, it is clear that smoke is present in every room. Another aspect of this effect is that the source of smoke in room 26 is not readily apparent. It raises the question of a separate source of smoke. From the full case it is clear that the smoke did in fact propagate as expected.

The full case now has two heat sensors in alarm, signifying a more serious incident. The sparse case has only smoke sensors in alarm, and no new sensors have gone into alarm for the past two minutes. The full case is indicating further developments in the fire which the sparse case cannot represent.

4.2.2.4 All sensors in alarm for sparse case: Twelve minutes from start of fire



(a) sparse (b) full

Figure 4.14: all sensors in alarm

All the sensors from the sparse case that are triggered by the simulation are now in alarm. The sparse case finally gives an indication that temperatures have climbed much higher close to the source of the fire. However, it is still missing information compared the full case, which shows that the high temperatures have already spread to the center of the building.

Chapter 5

Conclusions

Currently, EFRs accessing building sensor data are given sensor activation data sequentially and without context. A clear picture of the situation in the building is virtually inaccessible from textual input of this sort. We proposed a decision support tool to interpret building sensor data and visualize the progression of a developing fire emergency situation. The goal was to provide a visual representation of the emergency as accurately as possible to key decision making personnel using only technology already available in buildings. We developed a prototype decision support tool to show that a decision aid display can be an effective and feasible way to provide valuable information to incident commanders. The prototype was tested by using sensor activation inputs as well as virtual simulation inputs to verify the prototypes capability to handle physical sensor data and the prototypes ability to track fire emergency development accurately.

Using a sequence of triggered alarms, the prototype demonstrated that it could successfully handle sensor data inputs. In current systems, RS-232 has been a common form of data output for printers commonly connected to FACPs. Basic printer output from FACPs did not provide any useful visual context of the fire progression. When the same data from the printer output is used as an input for the prototype decision support tool, the GUI clearly shows the events of the emergency scenario in real-time. The delay between sensor activation and visualization of the GUI was minimal (four second maximum) and was in large part due to inefficiencies of the chosen programming language, Java, and the hardware limitations of the lab laptop. Tests with physical sensor inputs demonstrate that a decision support tool connect to existing building sensor systems and provide more effective information.

In order to perform experiments on a larger scale, virtual simulations of fire scenarios were created in FDS. The virtual sensor output of the FDS simulations was used as the input to the prototype. Then the visual outputs of Smokeview and the GUI were compared. There were two virtual experiments. One experiment was an analysis of a residential sized building with three sensors per floor and a total of three levels. This multi-level experiment was of a scale similar in size and sensor density to the hardware tests. The virtual experiments matched the general results of the hardware experiments and reached the same conclusions.

The next FDS experiment was of a significantly larger scale and sensor density, in a replication of the J.M. Patterson building at the University of Maryland. This comparison demonstrated that the progression of a fire emergency situation in a full-scale building can be accurately portrayed by a visualization tool. The experiment was run in two parts, one with a low sensor density (the real, as-built sensor configuration) and one with a high sensor density, closer to an ideal case. In the case of the low density test, the results showed that it was very difficult to get a detailed view of the progression of the fire. However, in the high density experiment, the progression of the fire was clear and identifiable. The high density scenario provided more at-a-glance information for EFRs. In addition, the stage of development of the fire was more readily apparent by comparing the progression of activated smoke sensors to heat sensors. While a higher senor density leads to a more precise picture, even in low sensor density configurations, the visual display provides more at-a-glance information than the plain text output of an FACP.

5.0.3 Future Directions

The system proposed in this project has many potential implications for its own portion of the industry as well as others. The future directions inspired by this project fall into two distinct categories: advances that can be made to improve the

decision support tool itself, and advances to the industry that vastly increase the potential of incorporating such a system.

For advances to the system itself, there are many improvements to be made. Further work can be done to streamline the user interface (UI) and maximize the human interfacing components of potential hardware that could be used. Currently, the software is in a proof-of-concept phase, running on any computer installed. It is envisioned to be installed on touchscreen devices like Toughbooks and other tablet devices for fast, intuitive touch interaction. In order to get to the envisioned state, more research on human interfacing must be considered. Usability testing needs to be done to determine what kind of difference this tool makes in EFR response to an emergency. Questions such as "Do EFRs make more effective decisions or make decisions significantly faster with the decision support tool?" are paramount in justifying its advanced development. Also, research to evaluate whether or not too much information is displayed is imperative. Optimizing the hardware and software for use by first responders is a key step in order for a system to be practically adopted.

Other advances to the decision support tool include increasing the variety of kinds of input it can adopt. The tool is designed to easily add new sensor technologies in a modular way. Annotations of floor plans can be updated to include new devices. The result is that there is room for new technology to get more and increasingly sophisticated data for analysis by the decision support tool. Inclusions that could be made based on current technology include adding in HVAC sensors, security systems and structural health monitoring devices. HVAC sensors can provide further information on the air flows within a building structure, monitor humidity, and track water flows. Security systems offer access to motion sensors which could be used to monitor motion in rooms due to either smoke current or persons still inside a building. Additionally, they provide security cameras that have video

feeds that could be used to provide an internal view of the building. Such cameras combined with infrared camera technology or new devices such as fire and smoke sensing cameras could bring significant visual data. Through cameras, the progression of fire and the location of civilians and EFRs within a building could be seen in detail. Additional technology that could be included is GPS tracking of individual EFRs within a building. While it is most likely a more complicated inclusion, it is potentially valuable information when responding to an emergency. Thorough and properly visualized inclusion of current technology could provide a whole new realm of data to aid first responders in analyzing an emergency scenario.

With modularity being a key goal of the system, taking in new types of data as new technologies evolve is also crucial. Ideally, new sensors that are able to more accurately monitor temperature, structural integrity and overall building health could be used to predict dangerous scenarios like flashover or building collapse. Predictive analysis based on accurate sensor data could potentially save lives. It is essential that, as the proposed decision support tool evolves, modularity stays at the forefront of priorities in its future direction.

For the industry, significant changes could lead to innovation and development in advances for emergency technology. An important recommendation for a new standard is increasing the requirements for building sensor configurations. As shown in the sensor density experiments, more sensors installed in a building allows for a more accurate visualization of the fire scenario. Requiring more senors would greatly increase the benefits of decision aid displays like the tested prototype, but may not be cost effective. Further cost-benefit analysis should be undertaken. The single biggest recommendation for the future that could be made is the adoption of a unified standard for a protocol of sensor and other building data. Such a standard is essential for the decision support tool to be able to take in new inputs. BACnet, a protocol used in the experiments by our team, is a good start at such a standard.

It makes the outputs of sensors unified and currently works across fire sensors and HVAC sensors. However, it does not have the capacity to deal with new kinds of sensor systems such as security or EFR tracking systems. Also, our experimentation with BACnet was limited to a sensor mockup of only twelve input devices. On a full sized building, there could be orders of magnitude more sensors to incorporate. The refresh rate of the BACnet protocol may not be fast enough to handle real-time monitoring of data on this scale. Further experimentation is necessary to confirm this fact.

Another important standard for the proposed system is NEMA SB 30, the standard that outlines how visuals are shown on an interface for EFRs. A key point for adoption of a new technology by first responders is a consistent visual interface. All layouts, symbols, colors and indicators need to be standardized across every display in order for there to be no new learning curve moving from device to device.

As another direction, the industry could work to advance technology in order to increase the potential of the proposed tool. Making sensors more robust in emergency scenarios (e.g., more heat resistant so they fail at a later time in fire scenarios), making them take in higher ranges of data at faster rates, and increasing the variety of data available could contribute to increasing the information available to first responders. Ultimately, the goal of the system is to provide better data for EFRs to make more informed decisions.

While discussing future directions of this complex system, there are also challenges to consider. Widespread implementation of the proposed decision support tool would require initial training of EFRs. There are both logistical and financial costs associated with such a significant amount of training. The economic challenges of mass hardware implementation must also be recognized. Conveniently, the system is primarily software in nature and could run on already existing hardware in the field such as Toughbooks. However, not all fire departments have the necessary

technology. Furthermore, if a new standard protocol is implemented, the fire sensor industry and all additional sensor industries that would become included will need to adopt that protocol and make appropriate hardware accommodations. While this project did not attempt to address these issues, in future work on this kind of system, it would be important to consider them.

Appendix A

Features List

Static/Pre-planning Sidebar

- The purpose of the static sidebar is to provide a synopsis of the building involved in the emergency situation. Firefighters will use this information before they arrive at the scene. This information will be displayed on the left side of the display in plain text. The following information, if available, MUST be displayed in this sidebar in the following order.
- Occupancy (notation) assembly, business, educational, factory, high-hazard, institutional, mercantile, residential, storage, utilities, miscellaneous (International Building Code 2006)
- Construction (notation) fire-resistive non-combustible, protected non-combustible, unprotected non-combustible, protected combustible, unprotected combustible, heavy timber, protected wood frame, unprotected wood frame
- Number of floors
- Possible hazards- chemical, gas tanks, biological, radioactive
- Display time of first alarm in 24-hour mode in hours-minutes-seconds (i.e., 20:15:30, as opposed to 08:15:30 P.M.).

Dynamic Alert Sidebar

- The purpose of the dynamic sidebar is to display all important alerts that have occurred during the duration of the emergency situation. This information will be displayed on the right side of the display in plain text.
- Display elapsed time since first alarm
- Display alarms chronologically with newest alarms on top. Alarms notated as: sprinklers active, smoke alarm, heat alarm, chemical alarm, chem. suppression, pull station
- Should state location of alarms by room, sprinklers by zone
- Should time stamp in 24 hour time
- Scroll bar if space of window for display is exceeded

- Display first alarm of each floor and time occurred

Floor Plan

- The purpose of the floor plan is to allow Incident Commanders to visually pinpoint the locations of sensors in alarm and other items of importance. The dynamic information displayed will be overlaid on a static, two-dimensional floor plan. The system will utilize NEMA SB 30 icons and icons created by our team. Only sensors in alarm will be displayed on the floor plan; idle sensors will be hidden from view.
- The floor plan should be draggable (if possible) but at minimum, move based on arrows.
- The following items must be displayed on the floor plan:
- Location and state of smoke detectors (NEMA SB 30)
- Rooms with active smoke detectors will be shaded grey.
- If a room is determined to contain both smoke and heat, the room will be shaded red. The presence of fire will take precedence over the presence of smoke.
- Location of activated heat sensors (NEMA SB 30)
- Rooms determined to have heat detector going off will be shaded red
- Location of activated hazardous material (hazmat) detectors (NEMA SB 30)
- Rooms determined to have activated hazmat detectors will be shaded green
- Rooms with hazmat and heat detectors activated will be striped green and red
- Rooms with hazmat and smoke detectors activated will be striped green and gray
- Location of hazardous materials; chemical, biological, radioactive (NEMA SB 30)
- Location of troubled sensors (NEMA SB 30)
- Sensors in alarm that stop reporting will display most recent state and troubled symbol
- Location of high-pressure gas storage (NEMA SB 30)
- Locations of elevators/elevator controls (NEMA SB 30)

  - Floor elevators on should be displayed on elevator icon

- Locations of stand pipes/fire hydrants (NEMA SB 30)
- Location of utility shutoff (gas, electric, water)
- Location of smoke vents and exhaust fans (NEMA SB 30)
- Locations of evacuation/pressurized stairs (NEMA SB 30)
- Location of standpipes and sprinkler shutoff (NEMA SB 30?)
- Locations of activated sprinkler systems
- Zones with activated sprinklers will be outlined in blue. Zones in a building will not otherwise be displayed.
- Displayed zones will have centered label
- Locations of activated chemical suppression systems
- Outline room with activated chemical suppression systems will be outline in purple.

Zoom (like Google Maps):

- Three button control
  - Zoom in: Moves in a set magnification (will have to figure actual amount later)
  - Zoom out: Moves out a set magnification (Max zoom out is default view)
  - Default: This button should set the map so that the entire floor is in view.
- Buttons will be placed at the upper left of the map

Replay:

- Scroll bar will allow flip through of the recorded map images
- Animation needs to be faster (several factors) than real time
- Animation will replace current map (Needs to be distinguished from real time image)
- Wherever scroll bar is dragged and dropped, it will continue playing till it reaches current situation

- Pressing the play button starts the animation at the beginning and plays to the end
- Only the map area will be affected by replay; all other areas functions as normal

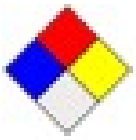
Floor control:

- Two areas to pick floors; first is floors in alarm, 2nd is complete list of floors
- Incident Commanders will have the ability to change floors through a scroll function, point-and-click on the floor of interest
- The system will default to the floor where the first alarm occurred.

Pop-up Alert Window Area

- The purpose of the popup alert window is to display current special alerts and crucial warnings to Incident Commanders. The pop-up alert window will be located in the lower section of the screen directly below the floor plan.
- Pop-up will appear with sound and will not close until acknowledged
- Newest alert will be in front of stack of pop ups
- Information on the pop-up will show up on the dynamic sidebar when pop up appears
- The following alerts will be displayed in the pop-up window:
  - When an alarm goes off on another floor (notated as: Alert: Alarm on floor X)
  - When a sprinkler systems activates (notated as: Alert: Sprinklers activated, floor X, zone Y)
  - When a chemical/special hazard occurs (notated as: Alert: K hazard, floor X, room Z)
  - When a chemical suppression system activates (notated as: Chemical suppression systems activate, floor X, room Z)

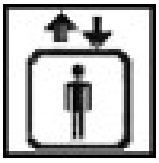
(a) Troubled Alarm



(b) Chemical Hazard

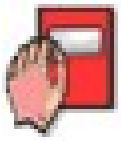
(c) Biological Hazard

(d) Radioactive Hazard



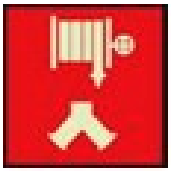
(e) Elevator (Displayed with floor number in the middle)

(f) Gas Tank



(g) Pull Station (displayed when in alarm)

(h) Stairs



(i) Standpipe

(j) Sprinkler Shutoff

Figure A.1: NEMA SB30 Symbols Used

Appendix B

Hardware Test: Emergency Scenario

T = 0 - fire starts

T = 0:00 = 60 seconds - H2 goes into alarm, A/V alarms sound (A/V sensors not visualized)

T = 0:30 = 90 seconds - I2 goes into alarm

T = 1:00 = 120 seconds - Manual Pull station activated

T = 1:30 = 150 seconds - P2 goes into alarm

T = 2:00 = 180 seconds - P3 Goes into alarm

T = 2:30 = 210 seconds - I3 go into alarm

T = 3:00 = 240 seconds - H3 Goes into Alarm

At time equals zero, the fire starts on the second floor in the bottom left corner room on the floor plan (all floors shown below). The fire is very sooty, producing a very hot, dense smoke. A hot smoke layer soon forms on the ceiling of the room and begins to exit into the hallway through an open door. As significant quantities of hot gases move over the second floor heat detector, the heat detector goes into alarm roughly sixty seconds from the start of the scenario, and the audio/visual alarms have activated to let personnel in the building be aware of the emergency. The smoke layer expands quickly through the small hallway and the ionization smoke sensor to the right detects the smoke layer at time equals ninety seconds. At this point, personnel have begun evacuating and going down the stairwell. As one person moves to the fire escape, they pull the manual pull station on the way out at time equals 120 seconds. The smoke layer on the second floor now begins to extend all the way to the end of the hallway towards the stairwell, and the photoelectric smoke detector on that floor goes into alarm at time equals 150 seconds. Because

the stairwell doors have been left open, the smoke enters the stairwell and rises, entering the third floor. Because of the increased burning conditions of the initial fire, the smoke layer is very hot and very dense, thus moving up the stairwell at a rapid rate. At time equals 180 seconds, the third floor photoelectric smoke detector has been triggered. Soon, it starts spreading into the hallway due to the stairwell door being left open during the evacuation process. By time equals 210 seconds, the ionization smoke sensor has gone into alarm as the smoke rapidly fills the third floor. At time t=240, the third floor heat detector ha been triggered. The scenario has concluded at this point.

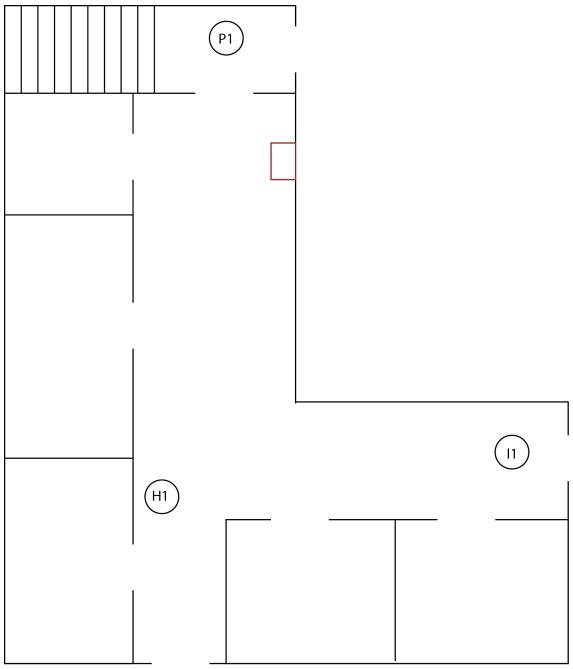


Figure B.1: Floor 1

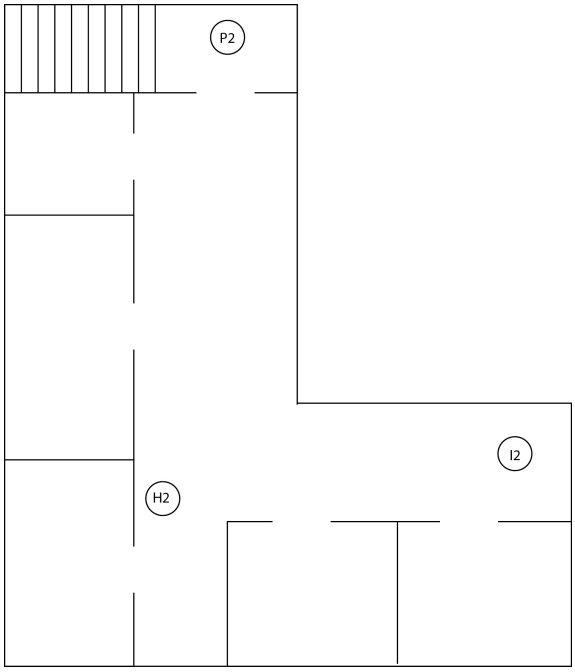


Figure B.2: Floor 2

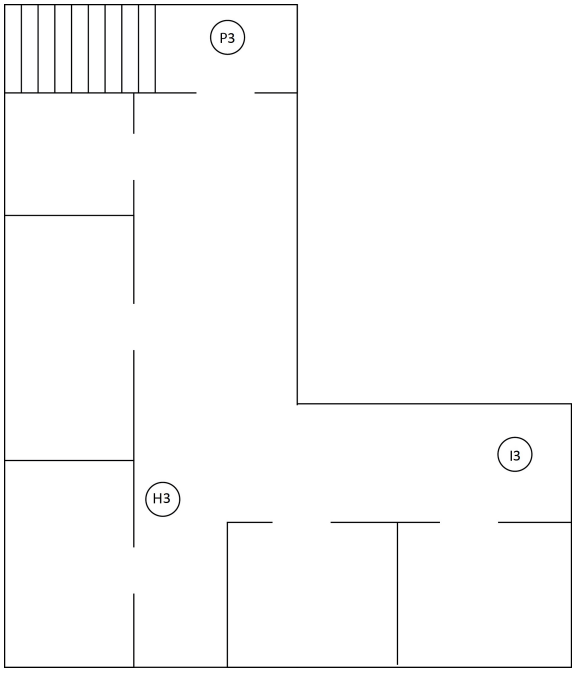


Figure B.3: Floor 3

Appendix C

Sensor Triggering Testing

Because alarms did not trigger immediately upon application of the magnet test trigger, there needed to be a way to account for the delay when testing the system. Each Honeywell sensor was triggered a total of fifteen times with a magnet. Each Simplex-Grinnell sensor was triggered a total of seven times with a flame or with canned smoke in light of the fact that those sensors were activated by changes in their analog values, not by magnets. They were triggered in random order to reduce systematic errors potentially introduced into the system by consistently triggering the sensors in similar time intervals. The time between when the sensor was triggered and the FACP registered the event was recorded. For each sensor the mean and standard deviation of its response times were taken. These numbers were then used to test for normality and perform a two tailed t-test for means with 95% certainty. This test was done to determine if a standard response time for each sensor could be assumed when testing the whole system. The results can be seen in the tables below.

Sensor Key			
Detector type	First Floor	Second Floor	Third Floor
Photo	L1D12	L1D52	
Ion	L1D11	L1D51	L1D91
Heat	L1D13	L1D53	L1D93

Sensor Name	Time (seconds)
L1D12	16.78
	13.76
	16.97
	16.62
	14.60
	18.93
	18.72
	16.90
	17.53
	17.67
Mean	16.85
STDEV	1.62

Sensor Name	Time (seconds)
L1D52	14.39
	17.46
	9.52
	15.57
	16.90
	8.24
	14.74
	17.04
	14.40
	17.83
Mean	14.61
STDEV	3.29

Sensor Name	Time (seconds)
L1D11	8.31
	7.96
	8.17
	6.70
	6.84
	9.09
	7.12
	7.12
	7.28
	7.47
Mean	7.61
STDEV	0.76

Sensor Name	Time (seconds)
L1D51	7.12
	6.91
	7.82
	7.47
	14.88
	8.16
	6.84
	9.92
	6.84
Mean	8.44
STDEV	2.60

Sensor Name	Time (seconds)
L1D13	6.00
	5.93
	5.86
	7.40
	6.56
	6.70
	6.49
	8.45
	9.22
	8.59
Mean	7.12
STDEV	1.23

Sensor Name	Time (seconds)
L1D53	6.78
	7.26
	6.07
	8.24
	7.57
	9.71
	7.47
	9.01
	4.40
	6.70
Mean	7.32
STDEV	1.50

Sensor Name	Time (seconds)
L1D91	7.05
	8.24
	8.11
	9.08
	7.41
	8.66
	8.45
	7.40
	9.71
	6.84
Mean	8.10
STDEV	0.92

Sensor Name	Time (seconds)
L1D93	9.08
	7.40
	7.54
	8.38
	6.84
	9.22
	8.87
	8.45
	8.24
	11.39
	7.48
Mean	8.44
STDEV	1.24

Appendix D

FDS Test 1: Full Results

D.1 Side by Side comparison of JMP simulation to GUI

Figure D.1: Side by Side comparison of JMP simulation to GUI at the one minute mark

Figure D.2: Side by Side comparison of JMP simulation to GUI at the two minute mark

Figure D.3: Side by Side comparison of JMP simulation to GUI at the three minute mark

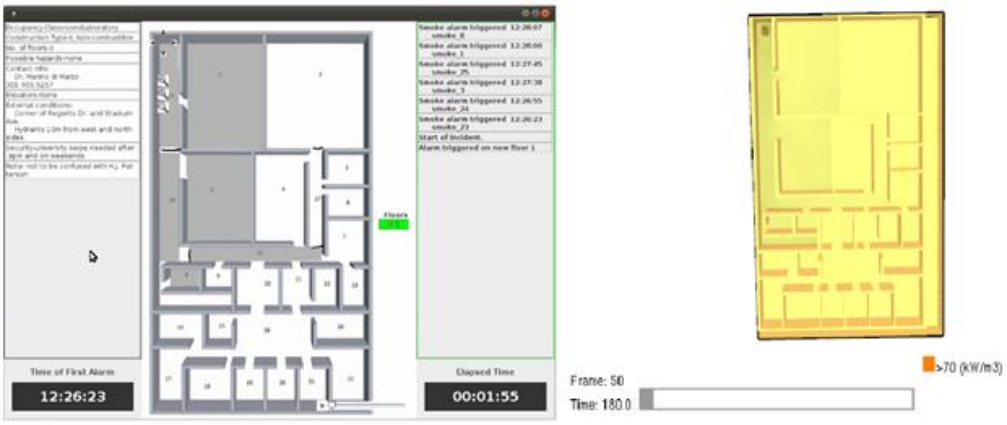


Figure D.4: Side by Side comparison of JMP simulation to GUI at the four minute mark

Figure D.5: Side by Side comparison of JMP simulation to GUI at the five minute mark

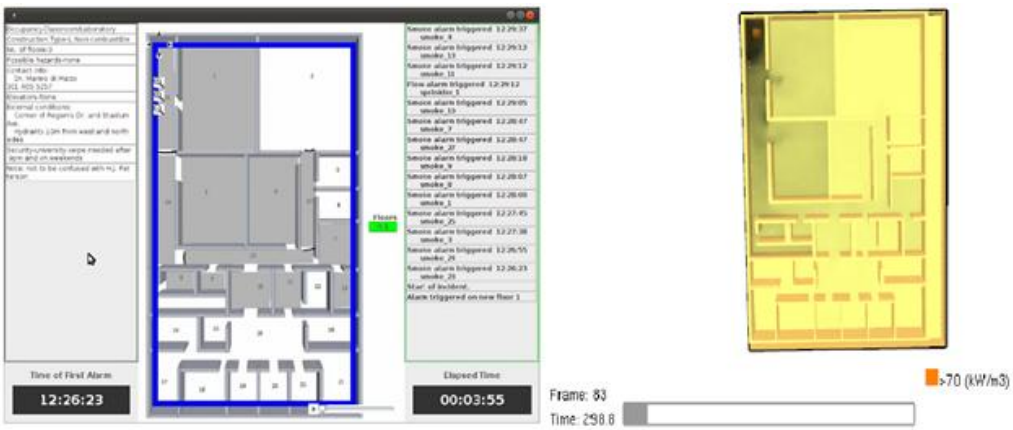


Figure D.6: Side by Side comparison of JMP simulation to GUI at the six minute mark

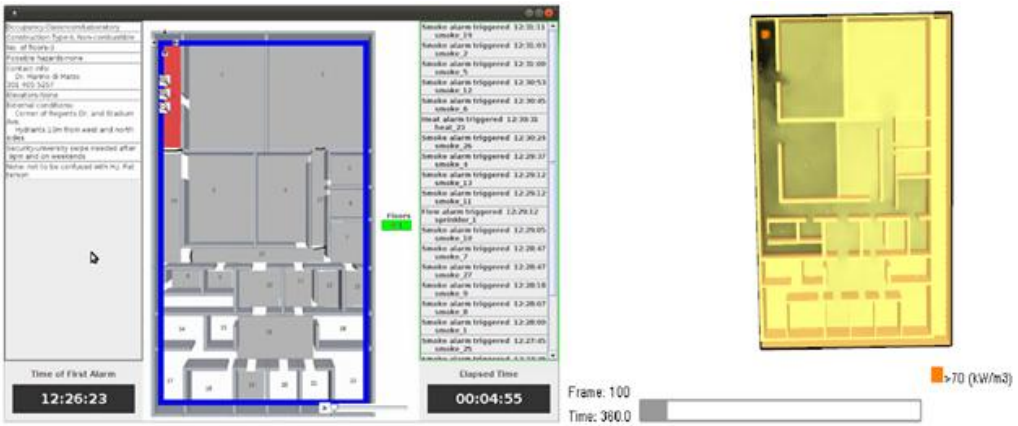


Figure D.7: Side by Side comparison of JMP simulation to GUI at the seven minute mark

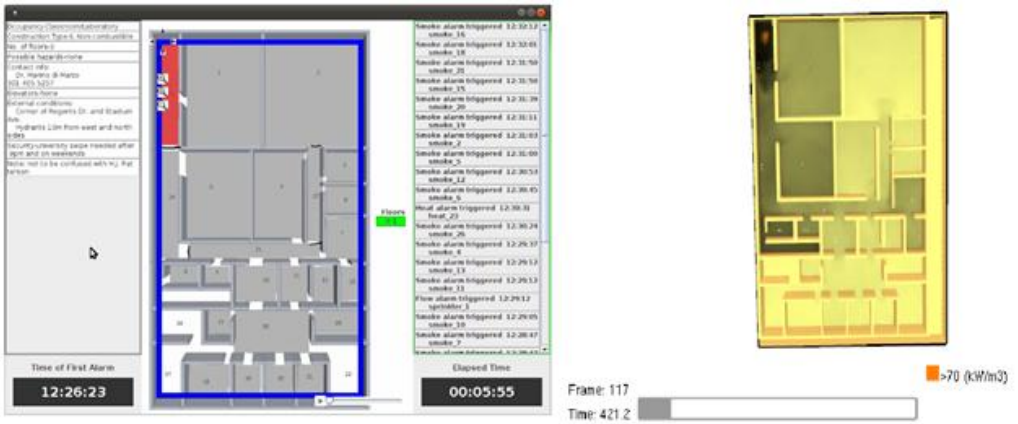


Figure D.8: Side by Side comparison of JMP simulation to GUI at the eight minute mark

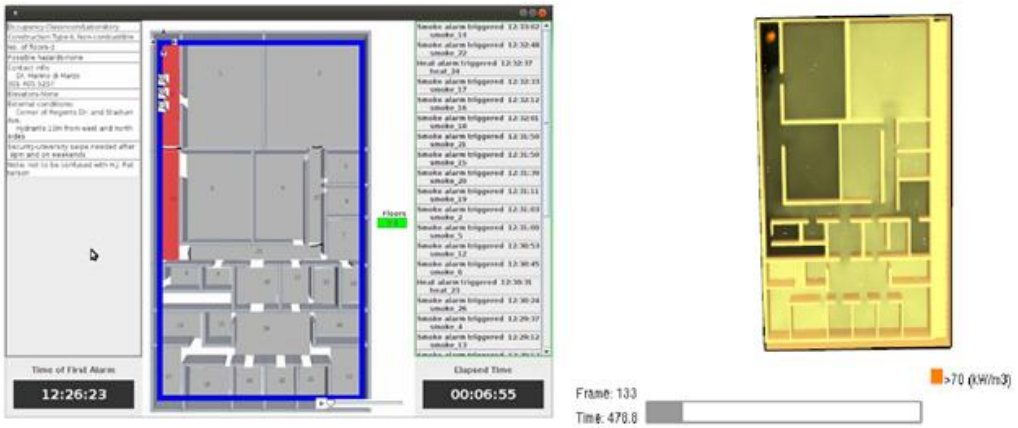


Figure D.9: Side by Side comparison of JMP simulation to GUI at the nine minute mark

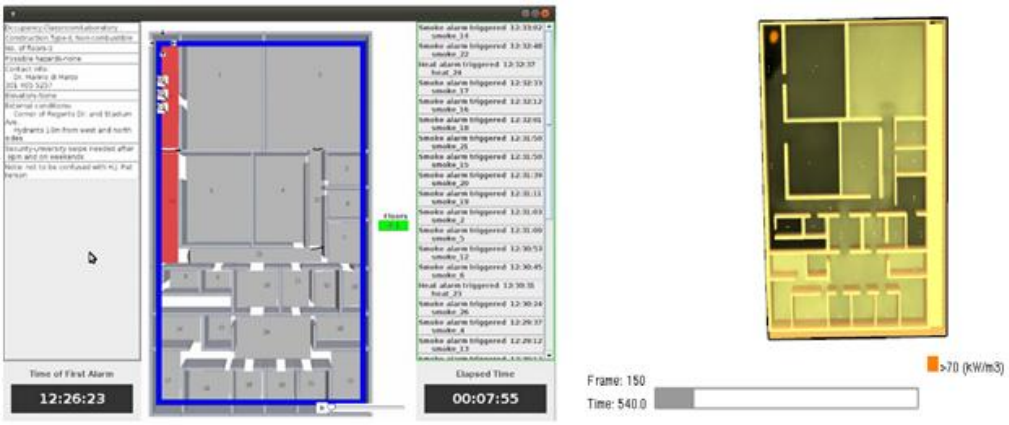


Figure D.10: Side by Side comparison of JMP simulation to GUI at the ten minute mark

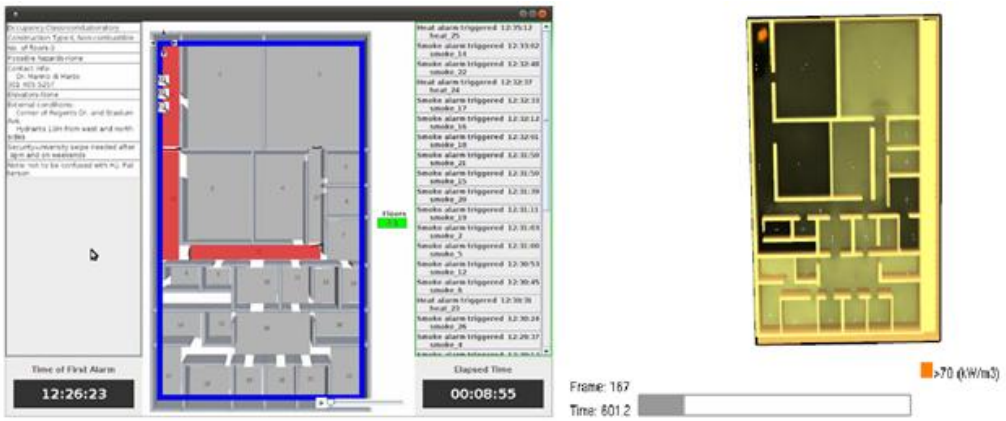


Figure D.11: Side by Side comparison of JMP simulation to GUI at the eleven minute mark

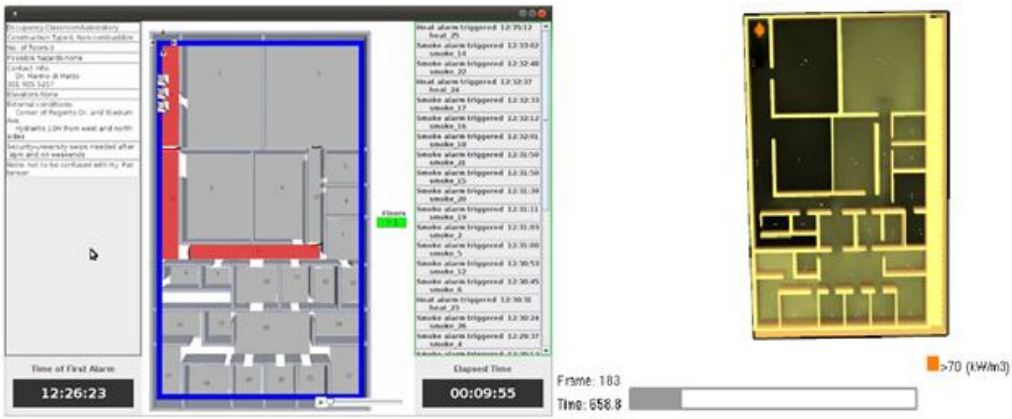


Figure D.12: Side by Side comparison of JMP simulation to GUI at the twelve minute mark

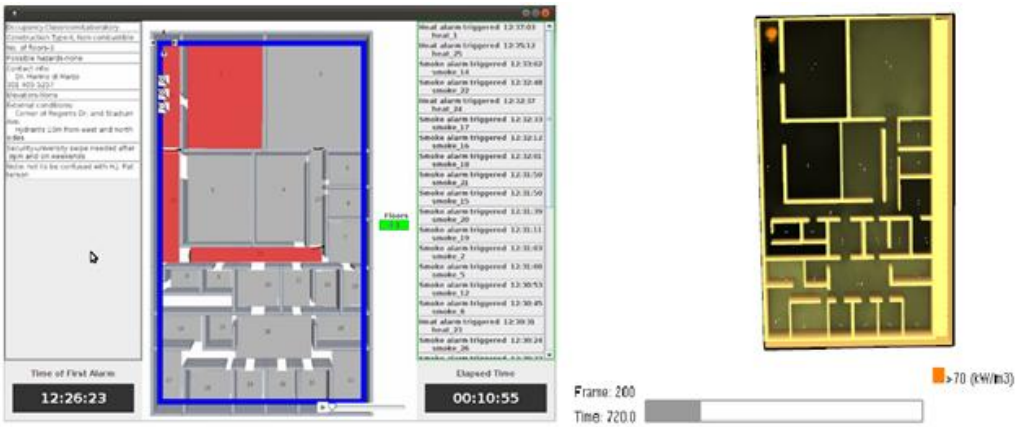


Figure D.13: Side by Side comparison of JMP simulation to GUI at the thirteen minute mark

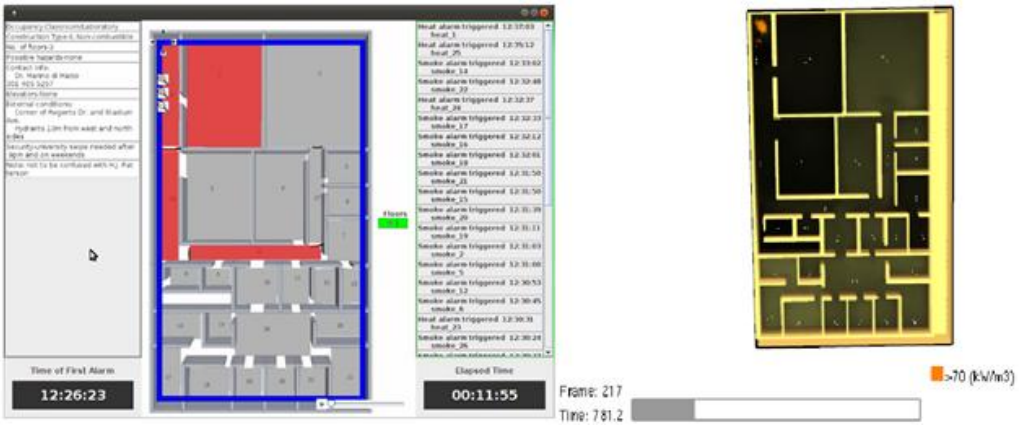


Figure D.14: Side by Side comparison of JMP simulation to GUI at the fourteen minute mark

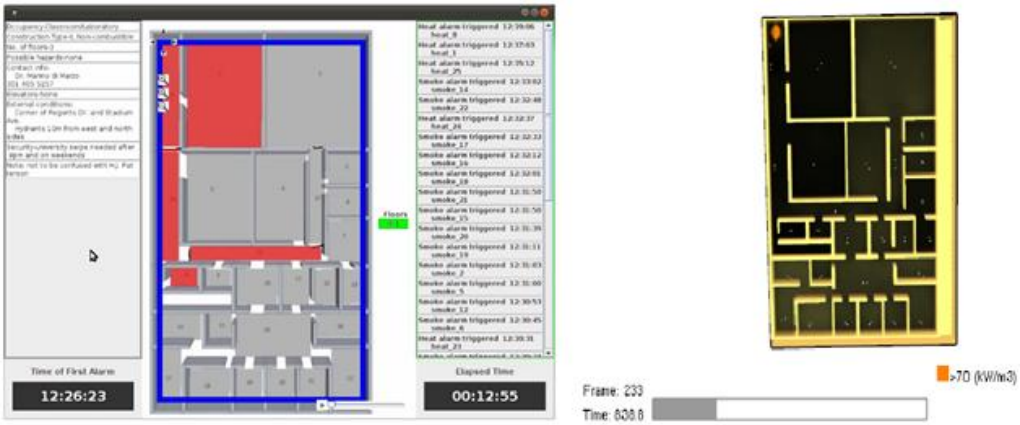


Figure D.15: Side by Side comparison of JMP simulation to GUI at the fifteen minute mark

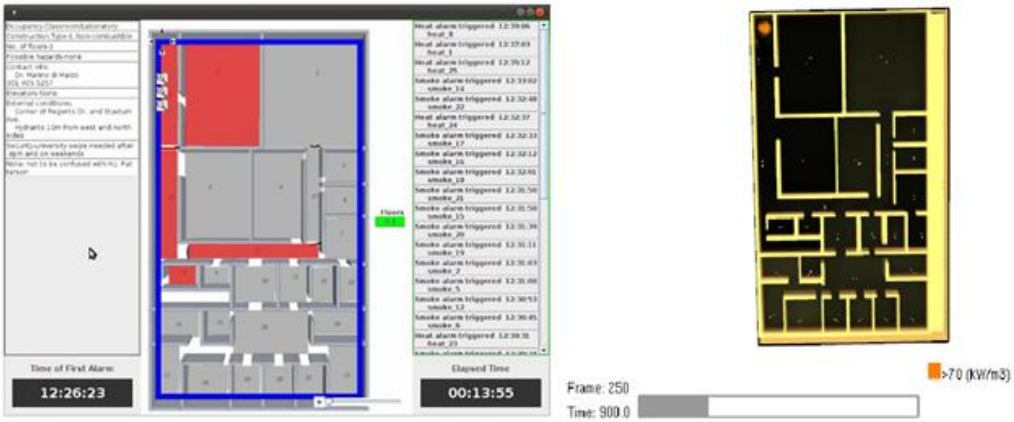


Figure D.16: Side by Side comparison of JMP simulation to GUI at the sixteen minute mark

Figure D.17: Side by Side comparison of JMP simulation to GUI at the seventeen minute mark

D.2 Side by Side comparison of Multilevel simulation to GUI

Figure D.18: Side by Side comparison of Multilevel simulation to GUI at the one minute mark

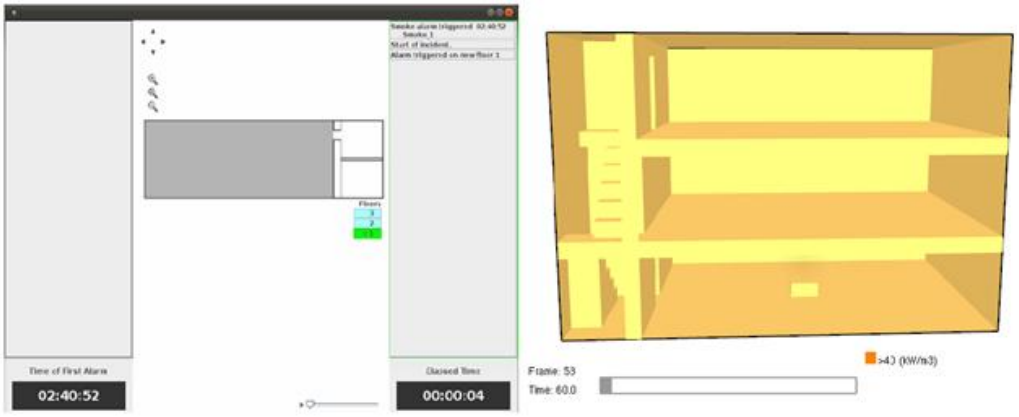


Figure D.19: Side by Side comparison of Multilevel simulation to GUI at the two minute mark

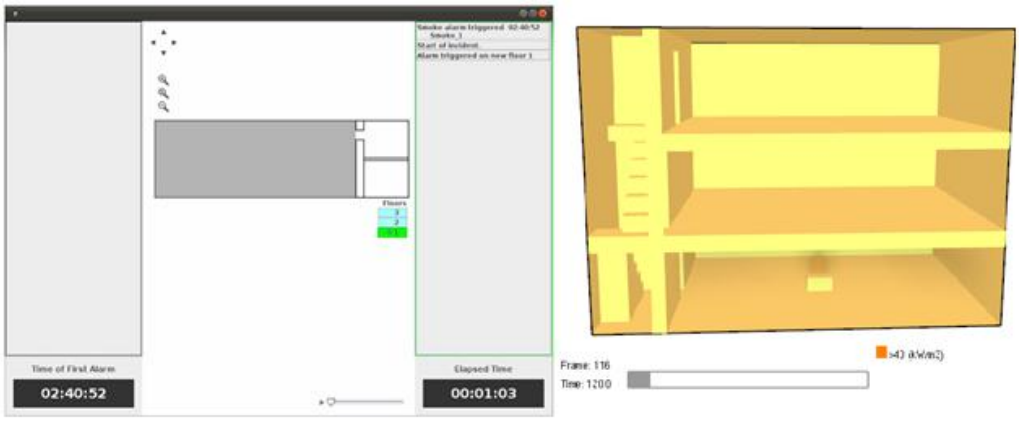


Figure D.20: Side by Side comparison of Multilevel simulation to GUI at the three minute mark

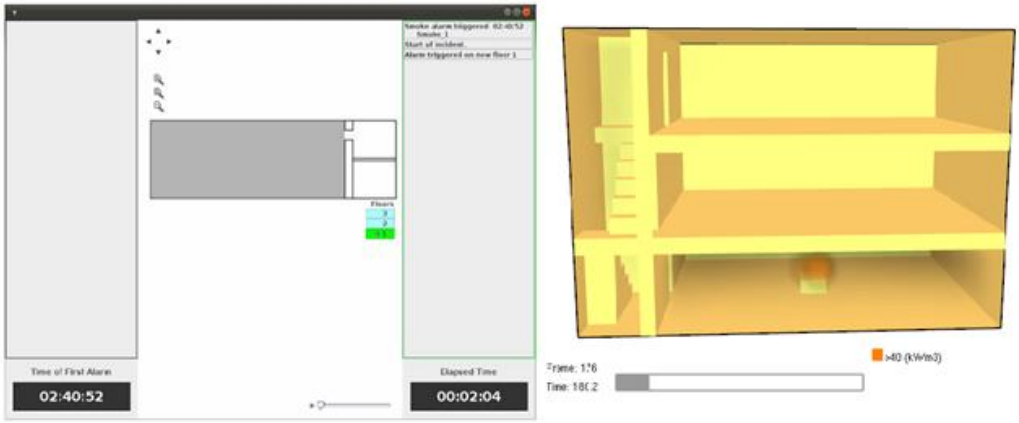


Figure D.21: Side by Side comparison of Multilevel simulation to GUI at the four minute mark

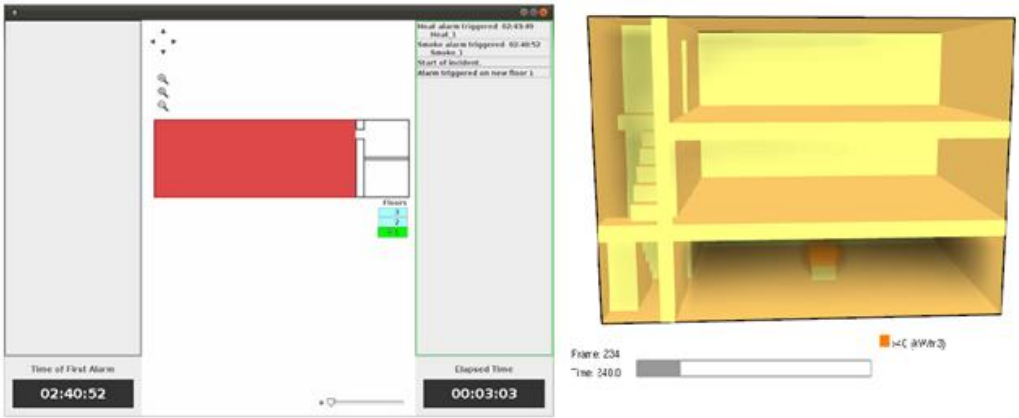
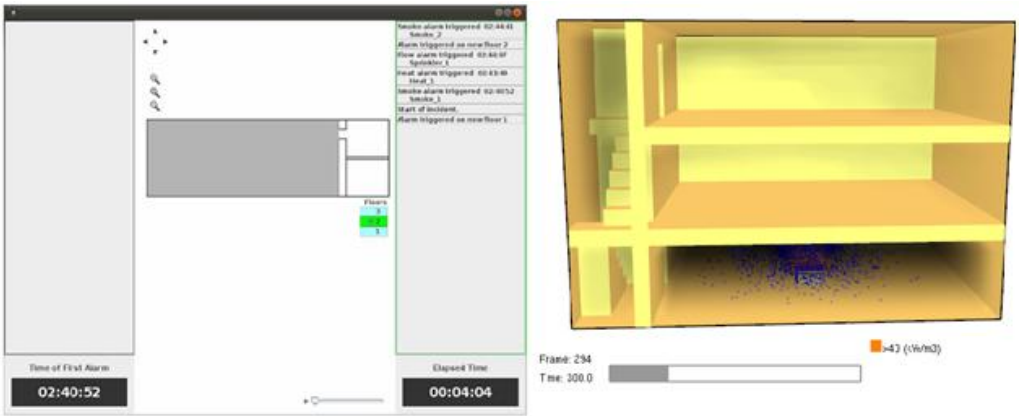


Figure D.22: Side by Side comparison of Multilevel simulation to GUI at the five minute mark



Appendix E

FDS Code

E.1 JMP FDS code

```
&HEAD CHID='jmp_fs'
&MESH IJK=120,120,12, XB=0.0,40,0.0,25,0.0,4.0, COLOR='BLACK'/
&TIME T_END=3600. /
*single burner, no objects, 30 min run

&SURF ID='FIRE',HRRPUA=10000.0,TAU_Q=-600.00/
&OBST XB=1.5,2,1.5,2,0,1.5,SURF_IDS='FIRE','INERT','INERT' /
&PART ID='water drops', WATER=.TRUE., SAMPLING_FACTOR=1 /

&PROP ID='Acme Sprinkler', QUANTITY='SPRINKLER LINK TEMPERATURE', RTI
    =95, C_FACTOR=0.4,
      ACTIVATION_TEMPERATURE=68, OFFSET=0.10,PART_ID='water drops',
          FLOW_RATE=189.3,
      DROPLET_VELOCITY=10., SPRAY_ANGLE=30.,80.    /

&PROP ID='Acme Smoke Detector', QUANTITY='CHAMBER OBSCURATION', LENGTH
    =1.8, ACTIVATION_OBSCURATION=3.28 /

&PROP ID='Acme Heat', QUANTITY='LINK TEMPERATURE', RTI=132.,
    ACTIVATION_TEMPERATURE=74. /

*smoke detector
&DEVC XYZ=6,17,3.9, ID='smoke_2', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=6,7,3.9, ID='smoke_1', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=6,1,3.9, ID='smoke_23', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=14,20.5,3.9, ID='smoke_5', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=17.5,20.5,3.9, ID='smoke_6', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=22,20.5,3.9, ID='smoke_7', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=17,17,3.9, ID='smoke_27', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=17.5,13,3.9, ID='smoke_4', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=17.5,6.333,3.9, ID='smoke_3', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=18,1,3.9, ID='smoke_24', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=25.5,3,3.9, ID='smoke_8', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=25.5,6.5,3.9, ID='smoke_9', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=26.5,10.5,3.9, ID='smoke_10', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=26.5,15,3.9, ID='smoke_11', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=26.5,18,3.9, ID='smoke_12', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=26.5,21.3333,3.9, ID='smoke_13', PROP_ID='Acme Smoke Detector
    ' /
&DEVC XYZ=31,2.5,3.9, ID='smoke_14', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=31,7,3.9, ID='smoke_15', PROP_ID='Acme Smoke Detector' /
```

```
&DEVC XYZ=32,12.5,3.9, ID='smoke_26', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=31,20,3.9, ID='smoke_16', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=36.5,1.5,3.9, ID='smoke_17', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=37,5.5,3.9, ID='smoke_18', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=37,10,3.9, ID='smoke_19', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=37,13.5,3.9, ID='smoke_20', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=37,16.5,3.9, ID='smoke_21', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=37,20,3.9, ID='smoke_22', PROP_ID='Acme Smoke Detector' /
&DEVC XYZ=23.25,9,3.9, ID='smoke_25', PROP_ID='Acme Smoke Detector' /

*sprinkler
&DEVC XYZ=4,1,3.9, ID='sprinkler_1', PROP_ID='Acme Sprinkler' /

*heat detector
&DEVC XYZ=5.5,1,3.9, ID='heat_23', PROP_ID='Acme Heat' /
&DEVC XYZ=5.5,7.5,3.9, ID='heat_1', PROP_ID='Acme Heat' /
&DEVC XYZ=5.5,16.5,3.9, ID='heat_2', PROP_ID='Acme Heat' /
&DEVC XYZ=17.5,1,3.9, ID='heat_24', PROP_ID='Acme Heat' /
&DEVC XYZ=18,16,3.9, ID='heat_3', PROP_ID='Acme Heat' /
&DEVC XYZ=18,13.5,3.9, ID='heat_4', PROP_ID='Acme Heat' /
&DEVC XYZ=18,17,3.9, ID='heat_27', PROP_ID='Acme Heat' /
&DEVC XYZ=14.333,20.5,3.9, ID='heat_5', PROP_ID='Acme Heat' /
&DEVC XYZ=17.333,20.5,3.9, ID='heat_6', PROP_ID='Acme Heat' /
&DEVC XYZ=22.333,20.5,3.9, ID='heat_7', PROP_ID='Acme Heat' /
&DEVC XYZ=23,9,3.9, ID='heat_25', PROP_ID='Acme Heat' /
&DEVC XYZ=25.5,3.25,3.9, ID='heat_8', PROP_ID='Acme Heat' /
&DEVC XYZ=25.5,6.75,3.9, ID='heat_9', PROP_ID='Acme Heat' /
&DEVC XYZ=27,10.5,3.9, ID='heat_10', PROP_ID='Acme Heat' /
&DEVC XYZ=27,15.25,3.9, ID='heat_11', PROP_ID='Acme Heat' /
&DEVC XYZ=27,18,3.9, ID='heat_12', PROP_ID='Acme Heat' /
&DEVC XYZ=27,21.5,3.9, ID='heat_13', PROP_ID='Acme Heat' /
&DEVC XYZ=31.33,3,3.9, ID='heat_14', PROP_ID='Acme Heat' /
&DEVC XYZ=31.33,6.75,3.9, ID='heat_15', PROP_ID='Acme Heat' /
&DEVC XYZ=32,13,3.9, ID='heat_26', PROP_ID='Acme Heat' /
&DEVC XYZ=31.33,20.5,3.9, ID='heat_16', PROP_ID='Acme Heat' /
&DEVC XYZ=36.25,2,3.9, ID='heat_17', PROP_ID='Acme Heat' /
&DEVC XYZ=37.25,5.25,3.9, ID='heat_18', PROP_ID='Acme Heat' /
&DEVC XYZ=37.25,10.25,3.9, ID='heat_19', PROP_ID='Acme Heat' /
&DEVC XYZ=37.25,13.25,3.9, ID='heat_20', PROP_ID='Acme Heat' /
&DEVC XYZ=37.25,16.75,3.9, ID='heat_21', PROP_ID='Acme Heat' /
&DEVC XYZ=37.25,20.25,3.9, ID='heat_22', PROP_ID='Acme Heat' /

*floorplan
&OBST XB=34.1316,34.4649,3.59167,7.59167,0.0,4.0/
&OBST XB=38.7983,39.1316,0.333334,3.27333,0.0,4.0 /
&OBST XB=34.1316,38.7983,3.25833,3.59167,0.0,4.0/
&OBST XB=38.7983,39.1316,3.25833,8.25833,0.0,4.0/
&OBST XB=32.4667,39.1333,0,0.333333,0.0,4.0 /
&OBST XB=34.1316,34.465,8.5917,9.5917,0.0,4.0 /
&OBST XB=34.1316,34.4649,10.5917,11.5917,0.0,4.0 /
&OBST XB=34.1316,38.7983,8.25833,8.59167,0.0,4.0 /
&OBST XB=38.7983,39.1316,8.25833,11.5917,0.0,4.0/
&OBST XB=34.1316,34.4649,11.925,12.925,0.0,4.0 /
&OBST XB=34.1316,34.4649,13.925,14.925,0.0,4.0 /
```

```
&OBST XB=34.1316,38.7983,11.5917,11.925,0.0,4.0 /
&OBST XB=38.7983,39.1316,11.5917,14.925,0.0,4.0 /
&OBST XB=30.0,32.5,8.333,8.66633,0.0,4.0 /
&OBST XB=28.885,32.135,5.333,5.66633,0.0,4.0 /
&OBST XB=32.1333,32.4667,5.333,8.333,0.0,4.0 /
&OBST XB=32.135,32.4683,0.333333,3.99999,0.0,4.0 /
&OBST XB=28.885,32.4683,0.0,0.333333,0.0,4.0 /
&OBST XB=34.1316,34.4649,15.2584,15.5917,0.0,4.0 /
&OBST XB=34.1316,38.7983,14.925,15.2584,0.0,4.0 /
&OBST XB=38.7983,39.1316,14.925,18.2584,0.0,4.0 /
&OBST XB=34.137,34.4703,16.666,18.2533,0.0,4.0 /
&OBST XB=34.137,38.803,18.2533,18.5867,0.0,4.0 /
&OBST XB=38.7983,39.1317,18.2583,22.3417,0.0,4.0 /
&OBST XB=32.4667,39.1333,22.3333,22.6667,0.0,4.0 /
&OBST XB=29.8888,32.4721,16.666,16.9994,0.0,4.0 /
&OBST XB=32.1388,32.4721,16.9994,22.3327,0.0,4.0 /
&OBST XB=28.8888,32.4721,22.3333,22.6667,0.0,4.0 /
&OBST XB=28.55,28.8833,0.332967,8.33297,0.0,4.0 /
&OBST XB=23.885,28.885,0.0,0.333333,0.0,4.0 /
&OBST XB=12.335,23.885,0.0,0.333333,0.0,4.0 /
&OBST XB=23.8867,25.5533,1.16666,1.49999,0.0,4.0 /
&OBST XB=23.8833,24.2167,1.49999,4.99999,0.0,4.0 /
&OBST XB=26.55,26.8833,1.16334,4.99667,0.0,4.0 /
&OBST XB=26.5533,26.8867,5.33315,8.33315,0.0,4.0 /
&OBST XB=23.8833,26.8833,4.99999,5.33333,0.0,4.0 /
&OBST XB=23.8833,24.2167,7.33333,8.33333,0.0,4.0 /
&OBST XB=23.8833,24.2167,5.3333,6.3333,0.0,4.0 /
&OBST XB=28.55,28.8833,8.6663,10.1663,0.0,4.0 /
&OBST XB=23.8833,28.8833,8.33297,8.6663,0.0,4.0 /
&OBST XB=23.8833,24.2167,8.6663,10.1663,0.0,4.0 /
&OBST XB=28.55,28.8833,11.6663,13.1663,0.0,4.0 /
&OBST XB=23.8833,24.2167,11.6663,13.1663,0.0,4.0 /
&OBST XB=28.55,28.8833,13.4997,14.4997,0.0,4.0 /
&OBST XB=28.5533,28.8867,15.5,16.5,0.0,4.0 /
&OBST XB=23.8833,28.8833,13.1663,13.4997,0.0,4.0 /
&OBST XB=23.8833,24.2167,13.5,14.5,0.0,4.0 /
&OBST XB=23.8867,24.22,15.5,16.5,0.0,4.0 /
&OBST XB=24.22,27.5533,16.5033,16.8367,0.0,4.0 /
&OBST XB=28.5533,28.8867,16.5033,19.3367,0.0,4.0 /
&OBST XB=15.0,21.6667,3.0,3.33333,0.0,4.0 /
&OBST XB=21.6667,22.0,2.99934,10.3327,0.0,4.0 /
&OBST XB=12.3333,21.6667,10.3333,10.6667,0.0,4.0 /
&OBST XB=21.6667,22.0,10.3333,14.6667,0.0,4.0 /
&OBST XB=13.3333,22.0,15.9019,16.2352,0.0,4.0 /
&OBST XB=23.8867,24.22,16.5033,19.3367,0.0,4.0 /
&OBST XB=23.9133,24.2467,20.82,22.32,0.0,4.0 /
&OBST XB=23.8867,28.5533,19.3333,19.6667,0.0,4.0 /
&OBST XB=28.5533,28.8867,19.3333,22.3333,0.0,4.0 /
&OBST XB=24.2221,28.8888,22.3333,22.6667,0.0,4.0 /
&OBST XB=19.3267,22.9933,17.8333,18.1667,0.0,4.0 /
&OBST XB=19.3333,24.2221,22.3333,22.6667,0.0,4.0 /
&OBST XB=16.6666,18.9999,17.8333,18.1667,0.0,4.0 /
&OBST XB=18.9999,19.3333,17.83,22.33,0.0,4.0 /
&OBST XB=16.0,19.3333,22.3333,22.6667,0.0,4.0 /
```

```
&OBST XB=12.3333,15.0,17.8333,18.1667,0.0,4.0 /
&OBST XB=15.6667,16.0,17.83,22.33,0.0,4.0 /
&OBST XB=12.3334,16.0,22.3333,22.6667,0.0,4.0 /
&OBST XB=7.52333,12.3567,3.0,3.33333,0.0,4.0 /
&OBST XB=0.333333,6.33333,3.0,3.33333,0.0,4.0 /
&OBST XB=12.0,12.3333,3.32667,11.4933,0.0,4.0 /
&OBST XB=0.333333,12.3333,0.0,0.333333,0.0,4.0 /
&OBST XB=12.0,12.3333,11.83,16.33,0.0,4.0 /
&OBST XB=0.3333,12.3333,11.5,11.8333,0.0,4.0 /
&OBST XB=12.0,12.3333,17.83,22.33,0.0,4.0 /
&OBST XB=0.0,0.333333,11.66,22.66,0.0,4.0 /
&OBST XB=0.0,0.333333,3.0,11.6667,0.0,4.0 /
&OBST XB=0.0,0.333333,0.0,3.0,0.0,4.0 /
&OBST XB=0.333333,12.3333,22.3332,22.6668,0.0,4.0 /
&HOLE XB=0.25,0.5,.5,.75,0,0.25/

&tail/
```

E.2 Multi-level FDS code

```
&HEAD CHID='multilevel' , TITLE='Multi-level Case'

&TIME T_END=1500. /

&MESH IJK=28,16,18, XB=0.0,14.0,0.0,8.0,0.0,9.0 /

&OBST XB=11.5,12.0,0.0,8.0,0.0,9.0 /
&OBST XB=12.0,14.0,3.5,4.0,0.0,9.0 /
&OBST XB=0.0,11.5,0.0,8.0,2.5,3.0 /
&OBST XB=0.0,11.5,0.0,8.0,5.5,6.0 /
&OBST XB=13.0,14.0,4.5,5.0,0.0,0.5/
&OBST XB=13.0,14.0,5.0,5.5,0.0,1.0/
&OBST XB=13.0,14.0,5.5,6.0,0.0,1.5/
&OBST XB=13.0,14.0,6.0,6.5,0.0,2.0/
&OBST XB=13.0,14.0,6.5,7.0,0.0,2.5/
&OBST XB=12.0,14.0,7.0,8.0,2.5,3.0 /
&OBST XB=12.0,13.0,7.0,7.5,3.0,3.5 /
&OBST XB=12.0,13.0,6.5,7.0,3.0,4.0 /
&OBST XB=12.0,13.0,6.0,6.5,3.0,4.5 /
&OBST XB=12.0,13.0,5.5,6.0,3.0,5.0 /
&OBST XB=12.0,13.0,5.0,5.5,3.0,5.5 /
&OBST XB=12.0,14.0,4.0,5.0,5.5,6.0 /

&HOLE XB=11.5,12.0,4.0,5.0,0.0,2.5 /
&HOLE XB=11.5,12.0,7.0,8.0,3.0,5.5 /
&HOLE XB=11.5,12.0,4.0,5.0,6.0,8.5 /
&HOLE XB=0.25,0.5,0.25,0.5,0.0,0.25/

&PROP ID='Acme Smoke Detector', QUANTITY='CHAMBER OBSCURATION', LENGTH
    =1.8,
ACTIVATION_OBSCURATION=3.28 /
```

```
&PROP ID='Acme Heat', QUANTITY='LINK TEMPERATURE', RTI=132.,
    ACTIVATION_TEMPERATURE=74. /

&DEVC ID='Smoke_1', PROP_ID='Acme Smoke Detector', XYZ=5.75,4.0,2.4 /
&DEVC ID='Smoke_2', PROP_ID='Acme Smoke Detector', XYZ=5.75,4.0,5.4 /
&DEVC ID='Smoke_3', PROP_ID='Acme Smoke Detector', XYZ=5.75,4.0,9.0 /

&DEVC ID='Heat_1', PROP_ID='Acme Heat', XYZ=6.0,4.0,2.4 /
&DEVC ID='Heat_2', PROP_ID='Acme Heat', XYZ=6.0,4.0,5.4 /
&DEVC ID='Heat_3', PROP_ID='Acme Heat', XYZ=6.0,4.0,9.0 /

&PART ID='water_drops', WATER=.TRUE., SAMPLING_FACTOR=1, QUANTITIES='
    DROPLET_DIAMETER', DIAMETER=2000. /
&PROP ID='K-11', QUANTITY='SPRINKLER LINK TEMPERATURE', RTI=95,
    C_FACTOR=0.4,
ACTIVATION_TEMPERATURE=68, OFFSET=0.10,PART_ID='water_drops', FLOW_RATE
    =189.3,
DROPLET_VELOCITY=10., SPRAY_ANGLE=30.,80. /

&DEVC ID='Sprinkler_1', XYZ=6.0,3.0,2.4, PROP_ID='K-11' /
&DEVC ID='Sprinkler_2', XYZ=6.0,3.0,5.4, PROP_ID='K-11' /
&DEVC ID='Sprinkler_3', XYZ=6.0,3.0,9.0, PROP_ID='K-11' /


&SURF ID='FIRE',HRRPUA=2000.0, TAU_Q=-600.00 /
&OBST XB=5.5,6.5,3.5,4.5,0.0,0.5,SURF_IDS='FIRE','INERT','INERT' /
&HOLE XB=5,5.25,3.5,3.75,0.0,0.25/
```

Appendix F
Prototype Source Code

```
/*****************************************
*
* ./ffa/annotation/AnnotationIO.java
*
*****************************************/
package ffa.annotation;

import java.awt.Polygon;
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;
import java.util.TreeMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.imageio.ImageIO;

import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import org.jdom.output.Format;
import org.jdom.output.XMLOutputter;

import ffa.model.Building;
import ffa.model.Floor;
import ffa.model.SensorLayer;
import ffa.model.SensorType;

public class AnnotationIO {

  private static File annotationFile(File root) {
    return new File(root, "annotation");
  }

  private static File floorFile(File root, int floor) {
    return new File(root, String.format("floor%d.png", floor));
  }

  public static Building newAnnotation(File folder) {
```

```
    if(annotationFile(folder).exists()) {
      return importAnnotation(folder);
    }

    File[] img = folder.listFiles();
    Pattern floorNum = Pattern.compile("floor([0-9]+)\\.png");
    int floorCount = 0;

    for (File f : img) {
      String name = f.getName();
      Matcher matcher = floorNum.matcher(name);
      if(matcher.matches()) {
        int floor = Integer.parseInt(matcher.group(1));
        if (floor > floorCount) {
          floorCount = floor;
        }
      }
    }

    Building building = new Building();
    for (int i = 0; i < floorCount; i++) {
      building.floors.put(i, new Floor());
    }

    setupZoneMap(building);

    loadFloorImages(building, folder);

    return building;
  }

  public static void exportAnnotation(Building building, File save) {
    Element root = new Element("building");
    Document doc = new Document(root);

    for (Integer index : building.floors.keySet()) {
      writeFloor(root, index, building.getFloor(index));
    }

    XMLOutputter out = new XMLOutputter(Format.getPrettyFormat());

    try {
      out.output(doc, new PrintWriter(save));
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }

  private static void writeFloor(Element root, Integer index, Floor
      floor) {
    Element elem = new Element("floor");
    root.addContent(elem);
```

```
    elem.addContent(new Element("index").addContent(index.toString()));
    for (SensorLayer layer : floor.layers.values()) {
      writeLayer(elem, layer);
    }
  }

  private static void writeLayer(Element root, SensorLayer layer) {
    Element elem = new Element("layer");
    root.addContent(elem);
    elem.setAttribute("type", layer.getType().name());

    for(Zone zone : layer.getZones()) {
      writeZone(elem, zone);
    }
  }

  private static void writeZone(Element root, Zone zone) {
    Element zoneElem = new Element("zone");
    zoneElem.addContent(new Element("label").setText(zone.getLabel()));
    Polygon shape = zone.getZone();
    Element shapeElem = new Element("zone");
    zoneElem.addContent(shapeElem);
    for(int i = 0; i < shape.npoints; i++) {
      shapeElem.addContent(new Element("vertex").
          addContent(new Element("x").setText(String.valueOf(shape.
              xpoints[i]))).
          addContent(new Element("y").setText(String.valueOf(shape.
              ypoints[i]))));
    }

    root.addContent(zoneElem);
  }

  @SuppressWarnings("unchecked")
  public static Building importAnnotation(File folder) {
    Building building = new Building();
    building.floors = new TreeMap<Integer, Floor>();


    Document doc = null;
    try {

      doc = new SAXBuilder().build(annotationFile(folder));

      for (Element ele : (List<Element>)doc.getRootElement().
          getChildren()) {

        if(ele.getName().equals("static")) {
          building.staticInfo.add(readStatic(ele));
        } else if (ele.getName().equals("floor")) {
          IndexedFloor floor = readFloor(ele);
          building.floors.put(floor.index, floor.floor);
        } else {
```

```
            throw new UnsupportedOperationException("Unknown XML tag: " +
                ele.getName());
          }
        }
      } catch (JDOMException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }

      setupZoneMap(building);

      loadFloorImages(building, folder);

      return building;
    }

    private static void loadFloorImages(Building building, File folder) {
      for(int i : building.floors.keySet()){
        try{
          File floorPlanFile = floorFile(folder, i+1);
          if(!floorPlanFile.exists()){
            System.out.println("file: " + floorPlanFile + " does not
                exist!");
          }

          building.getFloor(i).image = ImageIO.read(floorPlanFile);
        }catch(Exception e){
          System.out.println("Cannot read floor plans. Fatal error");
          e.printStackTrace();
          System.exit(0);
        }
      }//end for loop
    }

    private static void setupZoneMap(Building building) {
      building.zones = new TreeMap<String,Zone>();

      for (Floor floor : building.floors.values()) {
        for (SensorLayer layer : floor.layers.values()) {
          for (Zone zone : layer.getZones()) {
            building.zones.put(zone.getLabel(), zone);
          }
        }
      }
    }

    private static String readStatic(Element root) {
      return root.getTextTrim();
    }

    @SuppressWarnings("unchecked")
```

```
  private static IndexedFloor readFloor(Element root) {
    Floor floor = new Floor();
    Integer index = Integer.parseInt(root.getChildText("index"));


    for (Element element : (List<Element>)root.getChildren("layer")) {
      SensorLayer layer = readLayer(element);
      floor.layers.put(layer.getType(), layer);
    }

    return new IndexedFloor(floor, index);
  }

  @SuppressWarnings("unchecked")
  private static SensorLayer readLayer(Element root) {
    SensorType type = SensorType.valueOf(root.getAttribute("type").
        getValue());
    SensorLayer layer = new SensorLayer(type);

    for (Element element : (List<Element>)root.getChildren()) {
      if (element.getName().equals("zone")) {
        layer.addZone(readZone(element));
      }
    }

    return layer;
  }

  @SuppressWarnings("unchecked")
  private static Zone readZone(Element root) {
    Zone zone = new Zone();
    zone.setLabel(root.getChildText("label"));
    Polygon shape = zone.getZone();
    for (Element vertex : (List<Element>)root.getChild("zone").
        getChildren("vertex")) {
      int x = Integer.parseInt(vertex.getChildText("x"));
      int y = Integer.parseInt(vertex.getChildText("y"));

      shape.addPoint(x, y);
    }

    return zone;
  }

  private static class IndexedFloor {
    Floor floor;
    Integer index;

    public IndexedFloor(Floor floor, Integer index) {
      this.floor = floor;
      this.index = index;
    }
  }
}
```

```
/*****************************************
*
* ./ffa/annotation/ModeChangedListener.java
*
*****************************************/
package ffa.annotation;

import ffa.model.SensorType;

public interface ModeChangedListener {
  public void modeChanged(AnnotationMode mode);
  public void layerChanged(SensorType layer);
}
```

```
/*****************************************
*
* ./ffa/annotation/AnnotationWindow.java
*
*****************************************/
package ffa.annotation;

import info.clearthought.layout.TableLayout;

import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;

import ffa.model.Building;
import ffa.ui.MapDisplay;
import ffa.ui.Util;

@SuppressWarnings("serial")
public class AnnotationWindow extends JFrame {
  private MapDisplay mapDisplay;
  private AnnotationLayer aLayer;
  private EditPanel editPanel;
  private Toolbox toolbox;
```

```
//Three columns, left and right are 25%, one row, fills screen
private static final double size[][] = {{.25, TableLayout.FILL,
    .25},{TableLayout.FILL}};

private File last = new File("staticInfo");

public AnnotationWindow() {}

public void initGUI(Building building) {
  getContentPane().removeAll();
  Util.checkEDT();
  /******** Initialize Components **********/
  editPanel  = new EditPanel();
  File annotationFolder = new File("staticInfo");

  if (building == null) {
    building = AnnotationIO.newAnnotation(annotationFolder);
  }

  mapDisplay = new MapDisplay(this, building);
  aLayer = new AnnotationLayer(mapDisplay, editPanel);

  toolbox  = new Toolbox();
  toolbox.addModeChangedListener(aLayer);

  JMenuBar menuBar = new JMenuBar();
  JMenu menu = new JMenu("File");
  menuBar.add(menu);
  JMenuItem item = new JMenuItem("Save");
  menu.add(item);
  item.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
      JFileChooser chooser = new JFileChooser(last);
      int ret = chooser.showSaveDialog(AnnotationWindow.this);
      if (ret == JFileChooser.APPROVE_OPTION) {
        AnnotationIO.exportAnnotation(aLayer.getBuilding(), chooser.
            getSelectedFile());

        last = chooser.getSelectedFile();
      }
    }
  });
  item = new JMenuItem("Load");
  menu.add(item);
  item.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
      JFileChooser chooser = new JFileChooser(last);
      chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
      int ret = chooser.showOpenDialog(AnnotationWindow.this);
      if (ret == JFileChooser.APPROVE_OPTION) {
        Building building = AnnotationIO.importAnnotation(chooser.
            getSelectedFile());
```

```
            setVisible(false);
            initGUI(building);
            repaint();
            setVisible(true);
          }
        }
      });

      /******** Initialize Frame **********/
      this.setJMenuBar(menuBar);

      // I believe this is the standard screen size we're working with
      this.setSize(1024, 768);

      //Centers the window
      this.setLocationRelativeTo(null);

      // This is the main app window
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      setLayout(new TableLayout(size));

      getContentPane().add(mapDisplay, "1,0,1,0");
      getContentPane().add(toolbox, "0,0,0,0");
      getContentPane().add(editPanel, "2,0,2,0");
    }

    public void initGUI() {
      initGUI(null);
    }

    public static void main(String[] args) {
      final AnnotationWindow window = new AnnotationWindow();
      EventQueue.invokeLater(new Runnable(){
        public void run() {
          window.initGUI();
          window.setVisible(true);

        }
      });
    }
}
```

```
/*****************************************
*
* ./ffa/annotation/AnnotationLayer.java
*
*****************************************/
/*
Levon K. Mkrtchyan
```

```
This Component displays annotation stuffs and
accepts annotation commands
*/

package ffa.annotation;

import java.awt.Color;
import java.awt.Cursor;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Polygon;
import java.awt.Rectangle;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.geom.AffineTransform;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.util.Iterator;

import javax.swing.JComponent;

import ffa.model.Building;
import ffa.model.Floor;
import ffa.model.SensorLayer;
import ffa.model.SensorType;
import ffa.ui.MapDisplay;

public class AnnotationLayer extends JComponent implements
    ModeChangedListener{
  static final long serialVersionUID = 1L;

  private EditPanel editor;

  private Zone curZone = null;
  private int curZoneFloor = 0;
  private int curVertex = -1;

  private MapDisplay display = null;

  private Building building;
  //private Integer curFloor = 0;

  private SensorType layer = SensorType.HEAT;

  private ComponentListener cl = new ComponentAdapter(){
    public void componentResized(ComponentEvent e){
```

```
    AnnotationLayer.this.setSize(e.getComponent().getWidth(), e.
        getComponent().getHeight());
  }
};

private KeyListener kl = new KeyAdapter(){
  public void keyTyped(KeyEvent e){
    if(e.getKeyChar()==KeyEvent.VK_DELETE){
      Zone nz = nextZone(curZone);
      if(curVertex!=-1){
        deleteVertex(curZone,curVertex);
        curVertex--;
        if(curVertex<0)
          curVertex=curZone.getZone().npoints-1;

        if(curZone.getZone().npoints==0){
          curZone=nz;
          curVertex=-1;
        }
      }else if(curZone!=null){
        deleteZone(curZone);
        curZone=nz;
      }
    }else if(e.getKeyChar()==KeyEvent.VK_TAB){
      Zone nz;
      if(e.isShiftDown()){
        nz = prevZone(curZone);
      }else{
        nz = nextZone(curZone);
      }
      curVertex=-1;
      curZone=nz;
    }
    display.repaint();
  }
};

private MouseAdapter moveML = null;
private MouseAdapter zoneML = new MouseAdapter(){
  public void mousePressed(MouseEvent e){
    if(curVertex==-1){
      editor.clear();
      curZone = new Zone();
      curZoneFloor=display.getCurrentFloor();
      getBuilding().getLayer(display.getCurrentFloor(), layer).
          addZone(curZone);
      editor.edit(curZone);
    }

    curVertex = curZone.getZone().npoints;
    Point2D pt = display.screenToMap(e.getPoint());
    curZone.getZone().addPoint((int) pt.getX(), (int) pt.getY());

    display.repaint();
```

```
    }

    public void mouseClicked(MouseEvent e){
      if(e.getClickCount()==2)
        curVertex=-1;
      display.repaint();
    }

    public void mouseDragged(MouseEvent e){
      Polygon poly = curZone.getZone();
      Point2D pt = display.screenToMap(e.getPoint());
      poly.xpoints[curVertex]=(int) pt.getX();
      poly.ypoints[curVertex]=(int) pt.getY();
      display.repaint();
    }
  };
  private MouseAdapter editML = new MouseAdapter(){
    double zoneOffsetX = 0f;
    double zoneOffsetY = 0f;

    public void mousePressed(MouseEvent e){
      Point2D pt = display.screenToMap(e.getPoint());
      if(curZone!=null){
        Polygon poly = curZone.getZone();
        for(int n=0;n<poly.npoints;n++)
          if(pt.distance(poly.xpoints[n], poly.ypoints[n])*display.
              getZoom()<=4){
            curVertex=n;
            display.setCursor(new Cursor(Cursor.MOVE_CURSOR));
            display.repaint();
            return;
          }

        if(new Line2D.Double(
            poly.xpoints[poly.npoints-1],poly.ypoints[poly.npoints-1],
            poly.xpoints[0],poly.ypoints[0]).ptLineDist(pt)
            *display.getZoom()<=3){
          poly.addPoint((int) pt.getX(), (int) pt.getY());
          curVertex=poly.npoints-1;
          display.setCursor(new Cursor(Cursor.MOVE_CURSOR));
          display.repaint();
          return;
        }
        for(int n=1;n<poly.npoints;n++)
          if(new Line2D.Double(
              poly.xpoints[n-1],poly.ypoints[n-1],
              poly.xpoints[n],poly.ypoints[n]).ptLineDist(pt)
              *display.getZoom()<=3){
            poly.addPoint(1, 1);
            for(int i=poly.npoints-1;i>n;i--){
              poly.xpoints[i]=poly.xpoints[i-1];
              poly.ypoints[i]=poly.ypoints[i-1];
            }
            poly.xpoints[n]=(int) pt.getX();
```

```
            poly.ypoints[n]=(int) pt.getY();
            curVertex=n;
            display.setCursor(new Cursor(Cursor.MOVE_CURSOR));
            display.repaint();
            return;
          }
      }

      curZone=null;
      curVertex=-1;
      editor.clear();

      SensorLayer sLayer = getBuilding().getLayer(display.
          getCurrentFloor(), layer);

      for(Zone z : sLayer.getZones()){
        if(z.getZone().contains(pt)){
          curZone=z;
          curZoneFloor=display.getCurrentFloor();

          Rectangle bounds = curZone.getZone().getBounds();
          zoneOffsetX=pt.getX()-bounds.x;
          zoneOffsetY=pt.getY()-bounds.y;

          editor.edit(z);
          display.setCursor(new Cursor(Cursor.MOVE_CURSOR));
          display.repaint();
          return;
        }
      }
      display.repaint();
    }

    public void mouseReleased(MouseEvent e){
      display.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    }

    public void mouseDragged(MouseEvent e){
      Point2D pt = display.screenToMap(e.getPoint());
      if(curZone!=null && curVertex==-1){
        Rectangle bounds = curZone.getZone().getBounds();
        curZone.getZone().translate(
            (int) (pt.getX()-zoneOffsetX-bounds.x),
            (int) (pt.getY()-zoneOffsetY-bounds.y));
        display.repaint();
      }else if(curZone!=null && curVertex>=0){
        curZone.getZone().xpoints[curVertex]=(int) pt.getX();
        curZone.getZone().ypoints[curVertex]=(int) pt.getY();
        display.repaint();
      }
    }
};

public void modeChanged(AnnotationMode mode){
```

```
    curZone=null;
    editor.clear();

    this.removeMouseListener(moveML);
    this.removeMouseMotionListener(moveML);
    this.removeMouseListener(zoneML);
    this.removeMouseMotionListener(zoneML);
    this.removeMouseListener(editML);
    this.removeMouseMotionListener(editML);

    switch(mode){
    case MOVE:
      this.addMouseListener(moveML);
      this.addMouseMotionListener(moveML);
      this.addMouseWheelListener(moveML);
      break;
    case EDIT:
      this.addMouseListener(editML);
      this.addMouseMotionListener(editML);
      break;
    case ZONE:
      this.addMouseListener(zoneML);
      this.addMouseMotionListener(zoneML);
      break;
    }

    display.repaint();
  }
  @Override
  public void layerChanged(SensorType layer) {
    this.layer=layer;
    curZone=null;
    curVertex=-1;
    editor.clear();
    display.repaint();
  }

  public AnnotationLayer(MapDisplay display, EditPanel editor){
    this.editor = editor;
    this.display = display;
    this.setOpaque(false);
    display.add(this,new Integer(1));
    moveML = display.mapMouseListener;
    display.addComponentListener(cl);
    this.setBackground(new Color(255,255,255,Color.TRANSLUCENT));
    this.setForeground(new Color(255,255,255,Color.TRANSLUCENT));

    setBuilding(display.building);
    layerChanged(SensorType.SMOKE);


    /******* Sample Key Listener Code *********/
    setFocusable(true); // Component needs focus for keys
    setFocusTraversalKeysEnabled(false); // Disable tab and shift-tab
```

```
    // Request focus when mouse enters or when mouse is clicked.
    // TODO: Not sure which of these two is most appropriate
    addMouseListener(new MouseAdapter() {
      @Override
      public void mouseEntered(MouseEvent e) {
        AnnotationLayer.this.requestFocusInWindow();
        //System.out.println("hi");
      }

      @Override
      public void mouseClicked(MouseEvent e) {
        AnnotationLayer.this.requestFocusInWindow();
        //System.out.println("hallo");
      }
    });

    this.addKeyListener(kl);
  }

  public void paint(Graphics g){
    if(curZoneFloor!=display.getCurrentFloor())
      curZone=null;
    Graphics2D g2d = (Graphics2D) g;

    AffineTransform original = g2d.getTransform();

    g2d.transform(display.getTransform());

    float zoom = display.getZoom();

    Color opaque = Color.black;
    Color transparent;
    switch(layer){
    case HEAT:
      opaque = Color.red;
      break;
    case SMOKE:
      opaque = Color.gray;
      break;
    case FLOW:
      opaque = Color.blue;
      break;
    case MANUAL:
      opaque = Color.green;
      break;
    }
    transparent = new Color(opaque.getRed(),opaque.getGreen(),opaque.
        getBlue(),50);

    if(curZone!=null){
      g.setColor(transparent);
      g.fillPolygon(curZone.getZone());
    }
```

```
    g.setColor(opaque);
    SensorLayer sLayer = getBuilding().getLayer(display.getCurrentFloor
        (), layer);

    for(Zone z : sLayer.getZones()){
      g.drawPolygon(z.getZone());
      for(int n=0;n<z.getZone().npoints;n++){
        g.fillRect(z.getZone().xpoints[n]-(int)(2/zoom), z.getZone().
            ypoints[n]-(int)(2/zoom), (int)(5/zoom), (int)(5/zoom));
      }
    }

    if(curVertex>=0){
      g.setColor(Color.black);
      g.fillRect(curZone.getZone().xpoints[curVertex]-(int)(2/zoom),
          curZone.getZone().ypoints[curVertex]-(int)(2/zoom), (int)(5/
          zoom), (int)(5/zoom));
    }

    g2d.setTransform(original);
  }
  /*
  public void setLayers(Map<SensorType,SensorLayer> layers) {
    this.layers = layers;
    curZone = null;
    editor.clear();

    display.repaint();
  }*/

  private Zone prevZone(Zone z){
    Zone ret=null;
    Iterator<Zone> itr = getBuilding().getLayer(display.getCurrentFloor
        (), layer).getZones().iterator();
    while(itr.hasNext()){
      Zone next = itr.next();
      if(next==z && ret!=null)
        return ret;
      else
        ret=next;
    }

    itr = getBuilding().getLayer(display.getCurrentFloor(), layer).
        getZones().iterator();
    if (itr.hasNext() && itr.next()==z)
      return ret;
    else
      return null;
  }

  private Zone nextZone(Zone z){
    if(z==null)
      return null;
```

```
    Zone ret=null;
    Iterator<Zone> itr = getBuilding().getLayer(display.getCurrentFloor
        (), layer).getZones().iterator();
    while(itr.hasNext()){
      if(ret==z)
        return itr.next();
      else
        ret=itr.next();
    }
    if(ret==z)
      return getBuilding().getLayer(display.getCurrentFloor(), layer).
          getZones().iterator().next();
    else
      return null;
  }

  private void deleteZone(Zone z){
    getBuilding().getLayer(display.getCurrentFloor(), layer).removeZone
        (z);
  }

  private void deleteVertex(Zone z, int v){
    Polygon p = z.getZone();
    if(p.npoints<=v || v<0)
      return;

    if(p.npoints==1){
      p.npoints=0;
      deleteZone(z);
      return;
    }

    p.npoints--;
    for(int i=v;i<p.npoints;i++){
      p.xpoints[i]=p.xpoints[i+1];
      p.ypoints[i]=p.ypoints[i+1];
    }
    return;
  }
  public void setBuilding(Building building) {
    this.building = building;
  }
  public Building getBuilding() {
    return building;
  }
}




/*****************************************
*
* ./ffa/annotation/EditPanel.java
*
```

```
*****************************************/
package ffa.annotation;

import info.clearthought.layout.TableLayout;
import info.clearthought.layout.TableLayoutConstraints;

import java.awt.CardLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

import ffa.ui.Util;

@SuppressWarnings("serial")
public class EditPanel extends JPanel implements ActionListener {
  private Zone curZone = null;

  private CardLayout layout = new CardLayout();

  private JPanel zonePanel = new JPanel();
  JTextField zoneLabel = new JTextField();

  private JLabel zoneName = new JLabel();

  public EditPanel() {
    Util.checkEDT();

    this.setLayout(layout);

    add(new JPanel(), "blank");
    add(zonePanel, "zone");

    zonePanel.setLayout(new TableLayout(
        new double[][] {
            {.3,.7},
            {TableLayout.PREFERRED, TableLayout.PREFERRED, 15,
              TableLayout.PREFERRED, TableLayout.PREFERRED,
              TableLayout.PREFERRED}
        }));
    zonePanel.add(new JLabel("ZONE"), "0,0,1,0");
    zonePanel.add(new JLabel("Label: "), "0,1");
    zonePanel.add(zoneName, "1,1");
    zonePanel.add(new JLabel("Edit zone properties"), "0,3,1,3");
    zonePanel.add(new JLabel("Label:"), "0,4");
    zonePanel.add(zoneLabel, "1,4");

    JButton zoneOk = new JButton("OK");
    zoneOk.setActionCommand("zone");
    zoneOk.addActionListener(this);
```

```
    zonePanel.add(zoneOk, new TableLayoutConstraints(0,5,1,5,
        TableLayout.RIGHT, TableLayout.TOP));

  }

  public void clear() {
    curZone = null;
    layout.show(this, "blank");
  }
  public void edit(Zone zone) {
    zoneName.setText(zone.getLabel());
    zoneLabel.setText(zone.getLabel());
    layout.show(this, "zone");

    curZone = zone;
  }

  @Override
  public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals("zone")) {
      curZone.setLabel(zoneLabel.getText());
      edit(curZone); // Refresh window
    }
  }
}
```

```
/*****************************************
*
* ./ffa/annotation/Zone.java
*
*****************************************/
package ffa.annotation;

import java.awt.Polygon;
import java.util.Iterator;
import java.util.TreeSet;
import java.util.Comparator;
import ffa.model.Event;

import org.joda.time.DateTime;

public class Zone {
  private Polygon zone;
  private String label;
  private TreeSet<Event> events;//this field not used by
      AnnotationLayer, it is only used by MetaInfoLayer in the main
      visualization software.
  final static long displayDuration = 15000;

  public void addEvent(Event e){
```

```
    events.add(e);
  }

  public Zone() {
    events = new TreeSet<Event>(new Comparator<Event>(){
      public int compare(Event a, Event b){
        return a.getTime().compareTo(b.getTime());
      }
    });
    zone = new Polygon();
  }

  public int getOpacity(DateTime dt){
    long sysMillis = dt.getMillis();
    Event recentEvent = null;
    Iterator<Event> itr = events.descendingIterator();
    while(itr.hasNext() && recentEvent==null){
      Event e = itr.next();
      if(sysMillis > e.getTime().getTime())
        recentEvent = e;
    }

    if(recentEvent==null)// || sysMillis-recentEvent.getTime().getTime
        ()>displayDuration)
      return 0;

    return 200;//(int) ((displayDuration-sysMillis+recentEvent.getTime
        ().getTime())*200/displayDuration);
  }

  public Polygon getZone() {
    return zone;
  }

  protected String getLabel() {
    return label;
  }

  protected void setLabel(String label) {
    this.label = label;
  }
}
```

```
/*****************************************
*
* ./ffa/annotation/Toolbox.java
*
*****************************************/
package ffa.annotation;
```

```
import info.clearthought.layout.TableLayout;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.ButtonGroup;
import javax.swing.JPanel;
import javax.swing.JToggleButton;

import ffa.model.SensorType;
import ffa.ui.Util;

@SuppressWarnings("serial")
public class Toolbox extends JPanel implements ActionListener {
  private List<ModeChangedListener> modeChangedList = null;

  public Toolbox() {
    modeChangedList = new ArrayList<ModeChangedListener>();
    Util.checkEDT();
    setLayout(new TableLayout(new double[][]{{TableLayout.FILL
        },{.5,.5}}));

    JPanel buttonPanel = new JPanel(
        new TableLayout(new double[][]{
            {.5,.5},
            {TableLayout.PREFERRED, TableLayout.PREFERRED, TableLayout.
                PREFERRED}
        }));
    JPanel layerPanel = new JPanel();

    add(buttonPanel, "0,0");
    add(layerPanel, "0,1");

    ButtonGroup toolboxGroup = new ButtonGroup();

    ToolboxButton button = new ToolboxButton(AnnotationMode.MOVE,"Move
        ");
    button.addActionListener(this);
    toolboxGroup.add(button);
    buttonPanel.add(button, "0,0");
    button.doClick();//selects the move tool by default

    button = new ToolboxButton(AnnotationMode.EDIT, "Edit");
    button.addActionListener(this);
    toolboxGroup.add(button);
    buttonPanel.add(button, "1,0");

    button = new ToolboxButton(AnnotationMode.ZONE, "Zone");
    button.addActionListener(this);
    toolboxGroup.add(button);
    buttonPanel.add(button, "1,1");
```

```
/*    button = new ToolboxButton(AnnotationMode.DELETE, "Delete");
    button.addActionListener(this);
    toolboxGroup.add(button);
    buttonPanel.add(button, "1,2");*/

    ButtonGroup layerGroup = new ButtonGroup();
    for(SensorType type : SensorType.values()) {
      LayerButton layerButton = new LayerButton(type);
      layerButton.addActionListener(this);
      layerPanel.add(layerButton);
      layerGroup.add(layerButton);
    }
    layerGroup.getElements().nextElement().doClick(); //selects a layer
        by default - we don't really care which one
  }

  public void actionPerformed(ActionEvent e) {
    if (e.getSource() instanceof ToolboxButton) {
      ToolboxButton source = (ToolboxButton)e.getSource();

      for (ModeChangedListener listener : modeChangedList) {
        listener.modeChanged(source.getMode());
      }
    } else if (e.getSource() instanceof LayerButton) {
      LayerButton button = (LayerButton)e.getSource();

      for (ModeChangedListener listener : modeChangedList) {
        listener.layerChanged(button.type);
      }

      System.out.println(button.type);
    }
  }

  public void addModeChangedListener(ModeChangedListener l) {
    modeChangedList.add(l);
  }

  private class ToolboxButton extends JToggleButton{
    private AnnotationMode mode;
    public ToolboxButton(AnnotationMode mode, String text) {
      this.mode = mode;
      this.setText(text);
    }

    public AnnotationMode getMode() {
      return mode;
    }
  }
  private class LayerButton extends JToggleButton {
    public final SensorType type;
    public LayerButton(SensorType type) {
      this.type = type;
      setText(type.title);
```

```
    }
  }
}
```

```
/*****************************************
*
* ./ffa/annotation/AnnotationMode.java
*
*****************************************/
package ffa.annotation;

public enum AnnotationMode {
  MOVE, EDIT, SENSOR, ZONE, LINK, DELETE;
}
```

```
/*****************************************
*
* ./ffa/model/Sensor.java
*
*****************************************/
package ffa.model;


/**
 * @author jwu
 *
 */
public abstract class Sensor {
  String ID = "";
  Location location = null;

  boolean alarm = false;
  boolean trouble = false;

  public Sensor(Location location) {
    super();
    this.location = location;
  }
  public Sensor(String id)
  {
    super();
    this.ID = id;
  }
  public Sensor(Location loc, String id)
  {
    super();
```

```
    this.location = loc;
    this.ID = id;
  }

  public boolean inAlarm()
  {
    return alarm;
  }

  public String getID()
  {
    return ID;
  }
  public boolean inTrouble()
  {
    return trouble;
  }

  public void setAlarm(boolean status)
  {
    this.alarm = status;
  }

  public void setTrouble(boolean status)
  {
    this.alarm = trouble;
  }
}
```

```
/*****************************************
*
* ./ffa/model/RS232Demo.java
*
*****************************************/
package ffa.model;

import java.io.*;
import gnu.io.*;

/* Demo of input from RS232.  This requires a proper implementation
 * of the javax.comm library for a given operating system.  The
 * one currently used is the Windows version, found in RXTXcomm.jar
 */
public class RS232Demo {

  public static void main(String[] args)
  {
    CommPortIdentifier ident = null;
    String portName = "COM3";
```

```
  if(args.length > 0)
    portName = args[0];
  try{
    ident = CommPortIdentifier.getPortIdentifier(portName);
  }
  catch(NoSuchPortException e)
  {
    System.err.println("No such port: " + portName);
    System.exit(0);
  }
  if(ident == null || ident.isCurrentlyOwned())
  {
    System.err.println("Port " + portName + " in use");
    System.exit(0);
  }
      else
      {
        try{
          CommPort commPort = ident.open("Demo", 2000);

          if ( commPort instanceof SerialPort )
          {
            SerialPort serialPort = (SerialPort) commPort;
            serialPort.setSerialPortParams(2400,SerialPort.DATABITS_7
                ,SerialPort.STOPBITS_1,SerialPort.PARITY_EVEN);

            InputStream in = serialPort.getInputStream();

            serialPort.addEventListener(new SerialReader(in));
            serialPort.notifyOnDataAvailable(true);

          }
          else
          {
            System.out.println("Error: Only serial ports are handled
                by this example.");
          }
        }
        catch(Exception e)
        {
          e.printStackTrace();
        }
      }

}
public static class SerialReader implements SerialPortEventListener
  {
      private InputStream in;
      private byte[] buffer = new byte[1024];

      public SerialReader ( InputStream in )
      {
          this.in = in;
      }
```

```
public void serialEvent(SerialPortEvent arg0) {
        int data;
        String message;

        try
        {
            int len = 0;
            while ( ( data = in.read()) > -1 )
            {
                if ( data == '\n' ) {
                    break;
                }
                buffer[len++] = (byte) data;
            }

            message = parseString(new String(buffer,0,len));
            if(message != null)
              System.out.println("ALERT: " + message);
        }
        catch ( IOException e )
        {
            e.printStackTrace();
            System.exit(-1);
        }
    }
    public String parseString(String input)
    {
      String str = null;
      String[] inputArray;

      inputArray = input.split("\\s+", 7);

      if(inputArray[0].equals("ALARM:"))
      {
        str = inputArray[1] + " alarm with ID: " + inputArray[2] +
            " in zone " + inputArray[3] + " went off at " +
               inputArray[4] +
            " on " + inputArray[5].substring(0,2) + "/" +
            inputArray[5].substring(2,4) + "/" + inputArray[5].
               substring(4,6);
      }
      else if(inputArray[0].equals("TROUBL"))
      {
        if(inputArray.length == 7)
          str = "System in trouble at " + inputArray[4] +
              " on " + inputArray[5].substring(0,2) + "/" +
                inputArray[5].substring(2,4) + "/" + inputArray[5].
                   substring(4,6);
        else if(inputArray.length == 6)
          str = "System in trouble on " + inputArray[4].substring
             (0,2) + "/" +
          inputArray[4].substring(2,4) + "/" + inputArray[4].
             substring(4,6);
```

```
            else if(inputArray.length < 4)
            {
              str = "System in trouble";
            }
          }
          return str;
        }
    }

}
```

```
/*****************************************
*
* ./ffa/model/FireFilter.java
*
*****************************************/
package ffa.model;

/*Interface for fire filters.  This declares only the generic
 * filter() method, which does the preprocessing with help
 * from internal variables.
 */
public interface FireFilter
{
  public Alert filter(Event e);
}
```

```
/*****************************************
*
* ./ffa/model/BACNetPluginRip.java
*
*****************************************/
/*
 *
    ============================================================================

 * GNU Lesser General Public License
 *
    ============================================================================

 *
 * Copyright (C) 2006-2009 Serotonin Software Technologies Inc. http://
    serotoninsoftware.com
 * @author Matthew Lohbihler
 *
 * This library is free software; you can redistribute it and/or
```

```
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
    02111-1307, USA.
 */
package ffa.model;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;

import com.serotonin.bacnet4j.LocalDevice;
import com.serotonin.bacnet4j.RemoteDevice;
import com.serotonin.bacnet4j.RemoteObject;
import com.serotonin.bacnet4j.event.DeviceEventListener;
import com.serotonin.bacnet4j.exception.BACnetException;
import com.serotonin.bacnet4j.obj.BACnetObject;
import com.serotonin.bacnet4j.service.confirmed.
    ReinitializeDeviceRequest.ReinitializedStateOfDevice;
import com.serotonin.bacnet4j.service.unconfirmed.WhoIsRequest;
import com.serotonin.bacnet4j.type.Encodable;
import com.serotonin.bacnet4j.type.constructed.Choice;
import com.serotonin.bacnet4j.type.constructed.ObjectPropertyReference;
import com.serotonin.bacnet4j.type.constructed.PropertyValue;
import com.serotonin.bacnet4j.type.constructed.SequenceOf;
import com.serotonin.bacnet4j.type.constructed.TimeStamp;
import com.serotonin.bacnet4j.type.enumerated.EventState;
import com.serotonin.bacnet4j.type.enumerated.EventType;
import com.serotonin.bacnet4j.type.enumerated.MessagePriority;
import com.serotonin.bacnet4j.type.enumerated.NotifyType;
import com.serotonin.bacnet4j.type.enumerated.PropertyIdentifier;
import com.serotonin.bacnet4j.type.notificationParameters.
    NotificationParameters;
import com.serotonin.bacnet4j.type.primitive.Boolean;
import com.serotonin.bacnet4j.type.primitive.CharacterString;
import com.serotonin.bacnet4j.type.primitive.ObjectIdentifier;
import com.serotonin.bacnet4j.type.primitive.UnsignedInteger;
import com.serotonin.bacnet4j.util.PropertyReferences;
import com.serotonin.bacnet4j.util.PropertyValues;

/**
```

```
 * Discovers and devices and print all properties of all objects found.
 * this is done by using PropertyIdentifier.all so the Device will send
     all propertys that are set.
 * if you want poll all PropertyId {@link ReadPropertyRangeTest}.
 *
 * @author Matthew Lohbihler
 * @author Arne Plse
 */
public class BACNetPluginRip {
    public static String BROADCAST_ADDRESS = "192.168.1.255";
        //"127.0.0.255";
    private LoopDevice loopDevice;
    private final LocalDevice localDevice;
    // remote devices found
    private final List<RemoteDevice> remoteDevices = new ArrayList<
        RemoteDevice>();

    //maps devices to states
    private Map<RemoteDevice, String> deviceMap = new TreeMap<
        RemoteDevice, String>();

    public BACNetPluginRip(String broadcastAddress, int port) throws
        IOException {
        localDevice = new LocalDevice(1, broadcastAddress);

      localDevice.setPort(port);
        localDevice.getEventHandler().addListener(new
            DeviceEventListener() {

            public void listenerException(Throwable e) {
                System.out.println("DiscoveryTest listenerException");
            }

            public void iAmReceived(RemoteDevice d) {
                System.out.println("DiscoveryTest iAmReceived!");
                remoteDevices.add(d);
                synchronized (BACNetPluginRip.this) {
                    BACNetPluginRip.this.notifyAll();
                }
            }

            public boolean allowPropertyWrite(BACnetObject obj,
                PropertyValue pv) {
                System.out.println("DiscoveryTest allowPropertyWrite");
                return true;
            }

            public void propertyWritten(BACnetObject obj, PropertyValue
                pv) {
                System.out.println("DiscoveryTest propertyWritten");
            }

            public void iHaveReceived(RemoteDevice d, RemoteObject o) {
                System.out.println("DiscoveryTest iHaveReceived");
```

```
        }

        public void covNotificationReceived(UnsignedInteger
            subscriberProcessIdentifier, RemoteDevice
            initiatingDevice, ObjectIdentifier
            monitoredObjectIdentifier, UnsignedInteger
            timeRemaining, SequenceOf<PropertyValue> listOfValues)
            {
             System.out.println("DiscoveryTest
                 covNotificationReceived");
        }

        public void eventNotificationReceived(UnsignedInteger
            processIdentifier, RemoteDevice initiatingDevice,
            ObjectIdentifier eventObjectIdentifier, TimeStamp
            timeStamp, UnsignedInteger notificationClass,
            UnsignedInteger priority, EventType eventType,
            CharacterString messageText, NotifyType notifyType,
            Boolean ackRequired, EventState fromState, EventState
            toState, NotificationParameters eventValues) {
          System.out.println("BACNetPlugin
              eventNotificationReceived: ID=" + processIdentifier.
              longValue() + " EventType= " + eventType.toString() +
               " fromState=" + fromState.TYPE_ID + " toState= " +
              toState.TYPE_ID);

        }

        public void textMessageReceived(RemoteDevice
            textMessageSourceDevice, Choice messageClass,
            MessagePriority messagePriority, CharacterString
            message) {
             System.out.println("DiscoveryTest textMessageReceived")
                 ;
        }

        public void privateTransferReceived(UnsignedInteger
            vendorId, UnsignedInteger serviceNumber, Encodable
            serviceParameters) {
             System.out.println("DiscoveryTest
                 privateTransferReceived");
        }

    @Override
    public void reinitializeDevice(ReinitializedStateOfDevice arg0) {
      // TODO Auto-generated method stub

    }
      });
      localDevice.initialize();

}

/**
```

```
     * Send a WhoIs request and wait for the first to answer
     * @throws java.lang.Exception
     */
    public void doDiscover() throws Exception {
        // Who is
        System.out.println("Send Broadcast WhoIsRequest() ");
        // Send the broadcast to the correct port of the LoopDevice !!!
        //localDevice.sendBroadcast(loopDevice.getPort(), new
            WhoIsRequest(null, null));

        localDevice.sendBroadcast(47808, new WhoIsRequest(null, null));


        // wait for notification in iAmReceived() Timeout 5 sec
        synchronized (this) {
            final long start = System.currentTimeMillis();
            this.wait(5000);
            System.out.println(" waited for iAmReceived: " + (System.
                currentTimeMillis() - start) + " ms");
        }

        // An other way to get to the list of devices
        // return localDevice.getRemoteDevices();
    }

    @SuppressWarnings("unchecked")
    private void printDevices() throws BACnetException {
        for (RemoteDevice d : remoteDevices) {

            localDevice.getExtendedDeviceInformation(d);

            List<ObjectIdentifier> oids = ((SequenceOf<ObjectIdentifier
                >) localDevice.sendReadPropertyAllowNull(
                     d, d.getObjectIdentifier(), PropertyIdentifier.
                         objectList)).getValues();

            //xxx (10/19)
            System.out.println("removeDevice: " + d);

            // and now from all objects under the device object >> ai0,
                 ai1,bi0,bi1...
            for (ObjectIdentifier oid : oids) {
              PropertyReferences refs = new PropertyReferences();
                // add the property references of the "device object"
                    to the list
                refs.add(d.getObjectIdentifier(), PropertyIdentifier.
                    all);

                refs.add(oid, PropertyIdentifier.all);


              System.out.println("Start read properties");
                final long start = System.currentTimeMillis();
```

```
                System.out.println("JWU: refs = " + refs.size()); //xxx

                PropertyValues pvs = localDevice.readProperties(d, refs);
                System.out.println(String.format("Properties read done in
                    %d ms", System.currentTimeMillis() - start));
                String temp =printObject(d.getObjectIdentifier(), pvs);

                System.out.println("print1: " + temp);

                temp = printObject(oid, pvs);
                System.out.println("print2: " + temp);
              }
            }

            System.out.println("Remote devices done...");
        }

        //xxx - hacked to return state (10/19)
        private String printObject(ObjectIdentifier oid, PropertyValues pvs
            ) {
            System.out.println(String.format("\t%s", oid));

            Set<String> testing = new TreeSet<String>();
            testing.add("Object identifier");
            testing.add("Object type");
            testing.add("Object name");
            testing.add("state");
            testing.add("value");
            String temp;

            for (ObjectPropertyReference opr : pvs) {

//              if (oid.equals(opr.getObjectIdentifier())) {
//                  System.out.println(String.format("\t\t%s = %s", opr.
    getPropertyIdentifier().toString(), pvs.getNoErrorCheck(opr)));
//              }
              temp = opr.getPropertyIdentifier().toString();
              for(String s : testing) {
                if(temp.contains(s))
                  System.out.println(String.format("\t\t%s = %s", opr.
                      getPropertyIdentifier().toString(), pvs.
                      getNoErrorCheck(opr)));

                if(temp.contains("state"))
                  return pvs.getNoErrorCheck(opr).toString();
              }
            }

            return "No State";
        }

        /**
         * Note same Bropadcast address, but different ports!!!
         * @param args
```

```
     * @throws java.lang.Exception
     */
    @SuppressWarnings("unchecked")
    public static void main(String[] args) throws Exception {
        BACNetPluginRip dt = new BACNetPluginRip(BROADCAST_ADDRESS,
            47808);
        try {
            dt.setLoopDevice(new LoopDevice(BROADCAST_ADDRESS, 47808 +
                1));
        } catch (RuntimeException e) {
            dt.localDevice.terminate();
            throw e;
        }
//        try {
            dt.doDiscover();
            dt.printDevices();
//        } finally {
//            dt.localDevice.terminate();
//            System.out.println("Cleanup loopDevice");
//            dt.getLoopDevice().doTerminate();
//        }

            while(true) {

            }
    }

    /**
     * @return the loopDevice
     */
    public LoopDevice getLoopDevice() {
        return loopDevice;
    }

    /**
     * @param loopDevice the loopDevice to set
     */
    public void setLoopDevice(LoopDevice loopDevice) {
        this.loopDevice = loopDevice;
    }
}
```

```
/*****************************************
*
* ./ffa/model/SensorLayer.java
*
*****************************************/
package ffa.model;

import java.util.Comparator;
import java.util.TreeSet;
```

```
import java.util.Collection;
import java.awt.Rectangle;

import ffa.annotation.Zone;

public class SensorLayer {
  private SensorType type;
  private Collection<Zone> zoneList;

  public SensorLayer(SensorType type) {
    this.type = type;
    zoneList = new TreeSet<Zone>(new Comparator<Zone>(){
      public int compare(Zone a, Zone b){
        Rectangle aBounds = a.getZone().getBounds();
        Rectangle bBounds = b.getZone().getBounds();

        int aCenterX = aBounds.x+aBounds.width/2;
        int aCenterY = aBounds.y+aBounds.height/2;

        int bCenterX = bBounds.x+bBounds.width/2;
        int bCenterY = bBounds.y+bBounds.height/2;

        if((bCenterX>aCenterX) && (bCenterY>aBounds.y))
          return -1;

        if((bCenterX<aCenterX) && (bBounds.y<aCenterY))
          return 1;

        if(aCenterY<bCenterY)
          return -1;

        if(aCenterY>bCenterY)
          return 1;

        return new Integer(bCenterX).compareTo(new Integer(aCenterX));
      }
    });
  }

  public void addZone(Zone zone) {
    zoneList.add(zone);
  }

  public void removeZone(Zone zone) {
    zoneList.remove(zone);
  }

  public Iterable<Zone> getZones() {
    return zoneList;
  }
  public SensorType getType() {
    return type;
  }
}
```

```
/*****************************************
*
* ./ffa/model/LoopDevice.java
*
*****************************************/
/*
 *
    ============================================================================

 * GNU Lesser General Public License
 *
    ============================================================================

 *
 * Copyright (C) 2006-2009 Serotonin Software Technologies Inc. http://
    serotoninsoftware.com
 * @author Matthew Lohbihler
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
    02111-1307, USA.
 */
package ffa.model;

import java.io.IOException;

import com.serotonin.bacnet4j.LocalDevice;
import com.serotonin.bacnet4j.RemoteDevice;
import com.serotonin.bacnet4j.RemoteObject;
import com.serotonin.bacnet4j.event.DeviceEventListener;
import com.serotonin.bacnet4j.exception.BACnetServiceException;
import com.serotonin.bacnet4j.obj.BACnetObject;
import com.serotonin.bacnet4j.service.confirmed.
   ReinitializeDeviceRequest.ReinitializedStateOfDevice;
import com.serotonin.bacnet4j.type.Encodable;
import com.serotonin.bacnet4j.type.constructed.Choice;
import com.serotonin.bacnet4j.type.constructed.PropertyValue;
import com.serotonin.bacnet4j.type.constructed.SequenceOf;
```

```
import com.serotonin.bacnet4j.type.constructed.TimeStamp;
import com.serotonin.bacnet4j.type.enumerated.BinaryPV;
import com.serotonin.bacnet4j.type.enumerated.EngineeringUnits;
import com.serotonin.bacnet4j.type.enumerated.EventState;
import com.serotonin.bacnet4j.type.enumerated.EventType;
import com.serotonin.bacnet4j.type.enumerated.MessagePriority;
import com.serotonin.bacnet4j.type.enumerated.NotifyType;
import com.serotonin.bacnet4j.type.enumerated.ObjectType;
import com.serotonin.bacnet4j.type.enumerated.PropertyIdentifier;
import com.serotonin.bacnet4j.type.enumerated.Reliability;
import com.serotonin.bacnet4j.type.notificationParameters.
    NotificationParameters;
import com.serotonin.bacnet4j.type.primitive.Boolean;
import com.serotonin.bacnet4j.type.primitive.CharacterString;
import com.serotonin.bacnet4j.type.primitive.ObjectIdentifier;
import com.serotonin.bacnet4j.type.primitive.Real;
import com.serotonin.bacnet4j.type.primitive.UnsignedInteger;

/**
 *
 * software only device default local loop ;-)
 *
 * @author mlohbihler
 * @author aploese
 */
public class LoopDevice implements Runnable {

    public static void main(String[] args) throws Exception {
        LoopDevice ld = new LoopDevice("127.0.0.255", LocalDevice.
            DEFAULT_PORT + 1);
        Thread.sleep(12000); // wait 2 min
        ld.doTerminate();
    }

    private boolean terminate;
    private LocalDevice localDevice;
    private BACnetObject ai0;
    private BACnetObject ai1;
    private BACnetObject bi0;
    private BACnetObject bi1;
    private BACnetObject mso0;
    private BACnetObject ao0;

    public LoopDevice(String broadcastAddress, int port) throws
        BACnetServiceException, IOException {
        localDevice = new LocalDevice(1968, broadcastAddress);
        try {
            localDevice.setPort(port);
            localDevice.getEventHandler().addListener(new
                DeviceEventListener() {

                public void listenerException(Throwable e) {
                    System.out.println("loopDevice listenerException");
                }
```

```
public void iAmReceived(RemoteDevice d) {
    System.out.println("loopDevice iAmReceived");
}

public boolean allowPropertyWrite(BACnetObject obj,
    PropertyValue pv) {
    System.out.println("loopDevice allowPropertyWrite")
        ;
    return true;
}

public void propertyWritten(BACnetObject obj,
    PropertyValue pv) {
    System.out.println("loopDevice propertyWritten");
}

public void iHaveReceived(RemoteDevice d, RemoteObject
    o) {
    System.out.println("loopDevice iHaveReceived");
}

public void covNotificationReceived(UnsignedInteger
    subscriberProcessIdentifier, RemoteDevice
    initiatingDevice, ObjectIdentifier
    monitoredObjectIdentifier, UnsignedInteger
    timeRemaining, SequenceOf<PropertyValue>
    listOfValues) {
    System.out.println("loopDevice
        covNotificationReceived");
}

public void eventNotificationReceived(UnsignedInteger
    processIdentifier, RemoteDevice initiatingDevice,
    ObjectIdentifier eventObjectIdentifier, TimeStamp
    timeStamp, UnsignedInteger notificationClass,
    UnsignedInteger priority, EventType eventType,
    CharacterString messageText, NotifyType notifyType,
     Boolean ackRequired, EventState fromState,
    EventState toState, NotificationParameters
    eventValues) {
    System.out.println("loopDevice
        eventNotificationReceived");
}

public void textMessageReceived(RemoteDevice
    textMessageSourceDevice, Choice messageClass,
    MessagePriority messagePriority, CharacterString
    message) {
    System.out.println("loopDevice textMessageReceived
        ");
}
```

```
        public void privateTransferReceived(UnsignedInteger
            vendorId, UnsignedInteger serviceNumber, Encodable
            serviceParameters) {
             System.out.println("loopDevice
                 privateTransferReceived");
        }

@Override
public void reinitializeDevice(ReinitializedStateOfDevice arg0)
    {
  // TODO Auto-generated method stub

}
    });

    // for valid property values with valid datatypes see com.
        serotonin.bacnet4j.obj.ObjectProperties and ther look
        for the big static block at the end;
    // properties of device object
    localDevice.getConfiguration().setProperty(
        PropertyIdentifier.modelName, new CharacterString("
        BACnet4J LoopDevice"));

    // Set up a few objects.
    ai0 = new BACnetObject(localDevice, localDevice.
        getNextInstanceObjectIdentifier(ObjectType.analogInput)
        );

    //mandatory properties
    ai0.setProperty(PropertyIdentifier.objectName, new
        CharacterString("G1-RLT03-TM-01")); // this is a
        cryptic encoded name of a temp sensor from a drawing...
         (ahm. actually taken from a book ;-))
    ai0.setProperty(PropertyIdentifier.presentValue, new Real
        (11));
    ai0.setProperty(PropertyIdentifier.outOfService, new
        Boolean(false));
    ai0.setProperty(PropertyIdentifier.units, EngineeringUnits.
        degreesCelsius);

    //some optional properties
    ai0.setProperty(PropertyIdentifier.description, new
        CharacterString("temperature"));
    ai0.setProperty(PropertyIdentifier.deviceType, new
        CharacterString("random values"));
    ai0.setProperty(PropertyIdentifier.reliability, Reliability
        .noFaultDetected);
    ai0.setProperty(PropertyIdentifier.updateInterval, new
        UnsignedInteger(10));

    ai0.setProperty(PropertyIdentifier.minPresValue, new Real
        (-70));
    ai0.setProperty(PropertyIdentifier.maxPresValue, new Real
        (120));
```

```
ai0.setProperty(PropertyIdentifier.resolution, new Real((
    float)0.1));
ai0.setProperty(PropertyIdentifier.profileName, new
    CharacterString("funny reader"));

localDevice.addObject(ai0);

ai1 = new BACnetObject(
        localDevice, localDevice.
            getNextInstanceObjectIdentifier(ObjectType.
            analogInput));
ai1.setProperty(PropertyIdentifier.units, EngineeringUnits.
    percentObscurationPerFoot);
localDevice.addObject(ai1);

bi0 = new BACnetObject(
        localDevice, localDevice.
            getNextInstanceObjectIdentifier(ObjectType.
            binaryInput));
localDevice.addObject(bi0);
bi0.setProperty(PropertyIdentifier.objectName, new
    CharacterString("Off and on"));
bi0.setProperty(PropertyIdentifier.inactiveText, new
    CharacterString("Off"));
bi0.setProperty(PropertyIdentifier.activeText, new
    CharacterString("On"));

bi1 = new BACnetObject(
        localDevice, localDevice.
            getNextInstanceObjectIdentifier(ObjectType.
            binaryInput));
localDevice.addObject(bi1);
bi1.setProperty(PropertyIdentifier.objectName, new
    CharacterString("Good and bad"));
bi1.setProperty(PropertyIdentifier.inactiveText, new
    CharacterString("Bad"));
bi1.setProperty(PropertyIdentifier.activeText, new
    CharacterString("Good"));

mso0 = new BACnetObject(
        localDevice, localDevice.
            getNextInstanceObjectIdentifier(ObjectType.
            multiStateOutput));
mso0.setProperty(PropertyIdentifier.objectName, new
    CharacterString("Vegetable"));
mso0.setProperty(PropertyIdentifier.numberOfStates, new
    UnsignedInteger(4));
mso0.setProperty(PropertyIdentifier.stateText, 1, new
    CharacterString("Tomato"));
mso0.setProperty(PropertyIdentifier.stateText, 2, new
    CharacterString("Potato"));
mso0.setProperty(PropertyIdentifier.stateText, 3, new
    CharacterString("Onion"));
```

```
            mso0.setProperty(PropertyIdentifier.stateText, 4, new
                CharacterString("Broccoli"));
            mso0.setProperty(PropertyIdentifier.presentValue, new
                UnsignedInteger(1));
            localDevice.addObject(mso0);

            ao0 = new BACnetObject(
                    localDevice, localDevice.
                        getNextInstanceObjectIdentifier(ObjectType.
                        analogOutput));
            ao0.setProperty(PropertyIdentifier.objectName, new
                CharacterString("Settable analog"));
            localDevice.addObject(ao0);


            // Start the local device.
            localDevice.initialize();
            new Thread(this).start();
        } catch (RuntimeException e) {
            System.out.println("Ex in LoopDevice() ");
            e.printStackTrace();
            localDevice.terminate();
            localDevice = null;
            throw e;
        }
    }

    public void run() {
        try {
            System.out.println("LoopDevice start changing values" +
                this);

            // Let it go...
            float ai0value = 0;
            float ai1value = 0;
            boolean bi0value = false;
            boolean bi1value = false;

            getMso0().setProperty(PropertyIdentifier.presentValue, new
                UnsignedInteger(2));
            while (!isTerminate()) {
                System.out.println("Change values of LoopDevice " +
                    this);


                // Update the values in the objects.
                ai0.setProperty(PropertyIdentifier.presentValue, new
                    Real(ai0value));
                ai1.setProperty(PropertyIdentifier.presentValue, new
                    Real(ai1value));
                bi0.setProperty(PropertyIdentifier.presentValue,
                    bi0value ? BinaryPV.active : BinaryPV.inactive);
                bi1.setProperty(PropertyIdentifier.presentValue,
                    bi1value ? BinaryPV.active : BinaryPV.inactive);
```

```
            synchronized (this) {
                wait(1000); // 1 second or notified (faster exit
                    then stupid wait for 1 second)
            }
        }
        System.out.println("Close LoopDevive " + this);
    } catch (Exception ex) {
    }
    localDevice.terminate();
    localDevice = null;
}

@Override
protected void finalize() throws Throwable {
    if (localDevice != null) {
        localDevice.terminate();
        localDevice = null;
    }
}

/**
 * @return the terminate
 */
public boolean isTerminate() {
    return terminate;
}

/**
 * @param terminate the terminate to set
 */
public void doTerminate() {
    terminate = true;
    synchronized (this) {
        notifyAll(); // we may wait for this in run() ...
    }
}

/**
 * @return the broadcastAddress
 */
public String getBroadcastAddress() {
    return localDevice.getBroadcastAddress();
}

/**
 * @return the port
 */
public int getPort() {
    return localDevice.getPort();
}

/**
 * @return the localDevice
```

```
     */
    public LocalDevice getLocalDevice() {
        return localDevice;
    }

    /**
     * @return the ai0
     */
    public BACnetObject getAi0() {
        return ai0;
    }

    /**
     * @return the ai1
     */
    public BACnetObject getAi1() {
        return ai1;
    }

    /**
     * @return the bi0
     */
    public BACnetObject getBi0() {
        return bi0;
    }

    /**
     * @return the bi1
     */
    public BACnetObject getBi1() {
        return bi1;
    }

    /**
     * @return the mso0
     */
    public BACnetObject getMso0() {
        return mso0;
    }

    /**
     * @return the ao0
     */
    public BACnetObject getAo0() {
        return ao0;
    }
}



/*****************************************
*
* ./ffa/model/Event.java
```

```
*
*****************************************/
package ffa.model;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * @author jwu
 * @date   Nov 16, 2009
 */
public class Event{
  private Date time;
  private Sensor trigger;
  DateFormat timeFormat = new SimpleDateFormat("kk:mm:ss");

  public Event(Date time, Sensor trigger) {
    this.time = time;
    this.trigger = trigger;
  }

  protected void setTime(Date time) {
    this.time = time;
  }

  protected void setTrigger(Sensor trigger) {
    this.trigger = trigger;
  }

  public  String getText() {
    String msg="<HTML>";
    msg = msg+trigger.toString()+" "+trigger.getID()+ "  "+
        timeFormat.format(time)+"<BR>"+
    "      "+
    trigger.location.getDescription();
    return msg;
  }

  public Date getTime() {
    return time;
  }

  public Sensor getTrigger() {
    return trigger;
  }

  /**
   * @author jwu
   * @return this event's location
   */
  public Location getLocation() {
    return this.trigger.location;
  }
```

```
  public boolean equals(Object o) {
    if (this == o)
      return true;
    // Also checks o==null
    if (! (o instanceof Event))
      return false;

    //Cast is now safe
    Event e = (Event)o;

    if (!e.getTime().equals(getTime()))
      return false;

    if (!e.getText().equals(getText()))
      return false;

    if (!e.getTrigger().equals(getTrigger()))
      return false;

    return true;
  }
  public String getID()
  {
    return trigger.getID();
  }
}
```

```
/*****************************************
*
* ./ffa/model/DummyLord.java
*
*****************************************/
/*Used for testing the RS232 plugin; it could easily be used for
   testing any other plugin, as well.
 *Basically, instead of actually doing anything with the input from the
     plugin, it simply prints out information about it.
 *
 */


package ffa.model;

public class DummyLord extends EventLord
{
  public DummyLord()
  {

  }
  public void updateIncident(Event e)
```

```
  {
    System.out.print("Event received at " + e.getTime());
    System.out.println(" : name is " + e.getID());
  }
}
```

```
/*****************************************
*
* ./ffa/model/FloorFilter.java
*
*****************************************/
/*Filter that determines whether a sensor has gone off on a
 * new floor.  Implements FireFilter.
 *
 * TODO: Figure out how to send an alert
 */
package ffa.model;

public class FloorFilter implements FireFilter
{
  private boolean[] floors;
  private int numFloors;

  /* Constructs the filter, creating and initializing the
   * floor array.  Takes as an argument the number of
   * floors in the building.
   *
   * I don't think arrays need to be initialzed in java,
   * but better safe than sorry.
   */
  public FloorFilter(int n)
  {
    numFloors = n;
    floors = new boolean[n];
    for(int i = 0; i < n; i++)
      floors[i] = false;
  }
  /* Processes the given event.  Checks the value at the
   * index that corresponds to the floor of the event, and
   * if that value is false, set it to true and return an
   * alarm.  Else return null
   *
   */
  public Alert filter(Event e)
  {
    int f = e.getLocation().getFloor();
    if(!floors[f])
    {
      floors[f]=true;
```

```
      return new Alert(e.getTime(), e.getTrigger(), "Alarm triggered on
          new floor " + (f+1));
    }
    return null;
  }
}
```

```
/*****************************************
*
* ./ffa/model/EventLord.java
*
*****************************************/
/* Eventlord class.  Handles input from the network through
 * NetworkClient, preprocesses it for alerts, then passes it
 * to the Incident and notifies Window of an update.
 */

package ffa.model;
import java.util.ArrayList;
import java.util.ListIterator;

public class EventLord {
  private Incident incident;
  private ArrayList<FireFilter> filters;

  /* Constructor.  Takes as arguments the incident of
   * the system, and an arraylist of all desired
   * filters to be applied to new events.
   */
  public EventLord()
  {

  }
  public EventLord(Incident stub, ArrayList<FireFilter> filter)
  {
    incident = stub;
    filters = filter;
  }
  /* Update the incident.  This function takes an argument
   * from the NetworkClient, adds it to the event list,
   * applies all filters to it, and notifies the incident.
   * Each individual filter generates the actual alert,
   * which eventLord then updates the incident with.
   *
   *TODO: special handling of alerts as opposed to events?
   */
  public void updateIncident(Event e)
  {
    Alert a;
    ListIterator<FireFilter> iter = filters.listIterator();
```

```
    FireFilter f;

    incident.addEvent(e);
    while(iter.hasNext())
    {
      f = iter.next();
      a = f.filter(e);
      if(a != null)
      {
        incident.addEvent(a);
      }
    }
    incident.update();
  }
}
```

```
/*****************************************
*
* ./ffa/model/ThresholdSensor.java
*
*****************************************/
package ffa.model;


/**
 * @author jwu
 * @date   Nov 16, 2009
 */
public class ThresholdSensor extends Sensor{

  public ThresholdSensor(Location location, String ID) {
    super(location, ID);
  }

  public String toString() {
    return "<i>Threshold Sensor</i>";
  }

}
```

```
/*****************************************
*
* ./ffa/model/Building.java
*
*****************************************/
package ffa.model;
```

```
import java.util.ArrayList;
import java.util.Map;
import java.util.TreeMap;

import ffa.annotation.Zone;

public class Building {
  public Map<Integer, Floor> floors;
  public Map<String, Zone> zones;
  public ArrayList<String> staticInfo = new ArrayList();

  public Building() {
    floors = new TreeMap<Integer, Floor>();
  }

  //Convenience method
  public SensorLayer getLayer(Integer floor, SensorType type) {
    Floor fl = floors.get(floor);
    SensorLayer sl = fl.getLayer(type);
    return sl;
  }

  public Floor getFloor(Integer floor){
    return floors.get(floor);
  }

  public void addEvent(Event e){
    Sensor s = e.getTrigger();
    Zone z = zones.get(s.getID());
    if(z!=null){
      z.addEvent(e);
    }else{
      System.err.println("nonexistant sensor-id: " + e.getID());
    }
  }
}
```

```
/*****************************************
*
* ./ffa/model/BACNetPlugin.java
*
*****************************************/
package ffa.model;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
```

```
import com.serotonin.bacnet4j.LocalDevice;
import com.serotonin.bacnet4j.RemoteDevice;
import com.serotonin.bacnet4j.RemoteObject;
import com.serotonin.bacnet4j.event.DeviceEventListener;
import com.serotonin.bacnet4j.exception.BACnetException;
import com.serotonin.bacnet4j.obj.BACnetObject;
import com.serotonin.bacnet4j.service.confirmed.
    ReinitializeDeviceRequest.ReinitializedStateOfDevice;
import com.serotonin.bacnet4j.service.unconfirmed.WhoIsRequest;
import com.serotonin.bacnet4j.type.Encodable;
import com.serotonin.bacnet4j.type.constructed.Choice;
import com.serotonin.bacnet4j.type.constructed.ObjectPropertyReference;
import com.serotonin.bacnet4j.type.constructed.PropertyValue;
import com.serotonin.bacnet4j.type.constructed.SequenceOf;
import com.serotonin.bacnet4j.type.constructed.TimeStamp;
import com.serotonin.bacnet4j.type.enumerated.EventState;
import com.serotonin.bacnet4j.type.enumerated.EventType;
import com.serotonin.bacnet4j.type.enumerated.MessagePriority;
import com.serotonin.bacnet4j.type.enumerated.NotifyType;
import com.serotonin.bacnet4j.type.enumerated.PropertyIdentifier;
import com.serotonin.bacnet4j.type.notificationParameters.
    NotificationParameters;
import com.serotonin.bacnet4j.type.primitive.CharacterString;
import com.serotonin.bacnet4j.type.primitive.ObjectIdentifier;
import com.serotonin.bacnet4j.type.primitive.UnsignedInteger;
import com.serotonin.bacnet4j.util.PropertyReferences;
import com.serotonin.bacnet4j.util.PropertyValues;

public class BACNetPlugin extends Plugin{

  EventLord lord; //EventLord

  //basic constructor
  public BACNetPlugin(EventLord theLord, String broadcastAddress, int
      port) throws IOException
  {
    super.events = true;
    lord = theLord;

    localDevice = new LocalDevice(1, broadcastAddress);
    localDevice.setPort(port);
    localDevice.getEventHandler().addListener(new DeviceEventListener()
        {

      //TODO: edit the effects of these methods

      public void listenerException(Throwable e) {
        System.out.println("DiscoveryTest listenerException");
      }

      public void iAmReceived(RemoteDevice d) {
        System.out.println("DiscoveryTest iAmReceived");
        remoteDevices.add(d);
        synchronized (BACNetPlugin.this) {
```

```
    BACNetPlugin.this.notifyAll();
  }
}

public boolean allowPropertyWrite(BACnetObject obj, PropertyValue
    pv) {
  System.out.println("DiscoveryTest allowPropertyWrite");
  return true;
}

public void propertyWritten(BACnetObject obj, PropertyValue pv) {
  System.out.println("DiscoveryTest propertyWritten: " + pv.
      getValue());
}

public void iHaveReceived(RemoteDevice d, RemoteObject o) {
  System.out.println("DiscoveryTest iHaveReceived");
}

public void covNotificationReceived(UnsignedInteger
    subscriberProcessIdentifier, RemoteDevice initiatingDevice,
    ObjectIdentifier monitoredObjectIdentifier, UnsignedInteger
    timeRemaining, SequenceOf<PropertyValue> listOfValues) {
  System.out.println("DiscoveryTest covNotificationReceived");
}



public void textMessageReceived(RemoteDevice
    textMessageSourceDevice, Choice messageClass, MessagePriority
     messagePriority, CharacterString message) {
  System.out.println("DiscoveryTest textMessageReceived");
}

public void privateTransferReceived(UnsignedInteger vendorId,
    UnsignedInteger serviceNumber, Encodable serviceParameters) {
  System.out.println("DiscoveryTest privateTransferReceived");
}


public void eventNotificationReceived(UnsignedInteger
    processIdentifier,
    RemoteDevice initiatingDevice, ObjectIdentifier
        eventObjectIdentifier, TimeStamp timeStamp,
    UnsignedInteger notificationClass, UnsignedInteger priority,
        EventType eventType,
    CharacterString messageText, NotifyType notifyType,
    com.serotonin.bacnet4j.type.primitive.Boolean ackRequired,
        EventState fromState, EventState toState,
        NotificationParameters eventValues) {
  System.out.println("BACNetPlugin eventNotificationReceived: ID
      =" + processIdentifier.longValue() + " EventType= " +
      eventType.toString() + " fromState=" + fromState.TYPE_ID +
      " toState= " + toState.TYPE_ID);
```

```
        Date d = new Date(timeStamp.getDateTime().getTimeMillis());
        String id = initiatingDevice.getName()+initiatingDevice.
            getInstanceNumber();
        Sensor s = new SystemSensor(new Location(null, 0, eventType.
            toString()), id);
        Event e = new Event(d, s);
        lord.updateIncident(e);
        //type_id = 1 for change of state eventType
        //type_id = 2 for change of value eventType
      }

      @Override
      public void reinitializeDevice(ReinitializedStateOfDevice arg0) {
        // TODO Auto-generated method stub

      }


    });
    localDevice.initialize(); //starts event listening
  }

  @Override
  public void prepare() {
    // TODO Auto-generated method stub
    //does not need to do anything
  }

  @Override
  public void start() {
    // TODO Auto-generated method stub

    try {
      TestUseListener();
      //TestUsePolling();
      //TestUseNaive();
    } catch (Exception e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

  }



  private LoopDevice loopDevice;
  private final LocalDevice localDevice;
  // remote devices found
  private final List<RemoteDevice> remoteDevices = new ArrayList<
      RemoteDevice>();

  //3 implementation styles to be tested
  private void TestUseListener() throws Exception {
```

```
  try {
    this.setLoopDevice(new LoopDevice("192.168.1.255", 47808 + 1));
        //TODO not hardcoded
  } catch (RuntimeException e) {
    this.localDevice.terminate();
    throw e;
  }


  try {
          this.doDiscover();
          this.printDevices();
      } finally {
          this.localDevice.terminate();
         System.out.println("Cleanup loopDevice");
         this.getLoopDevice().doTerminate();
      }

}

/**
 * Send a WhoIs request and wait for the first to answer
 * @throws java.lang.Exception
 */
public void doDiscover() throws Exception {
  // Who is
  System.out.println("Send Broadcast WhoIsRequest() ");
  // Send the broadcast to the correct port of the LoopDevice !!!
  //localDevice.sendBroadcast(loopDevice.getPort(), new WhoIsRequest(
      null, null));

  localDevice.sendBroadcast(47808, new WhoIsRequest(null, null));


  // wait for notification in iAmReceived() Timeout 5 sec
  synchronized (this) {
    final long start = System.currentTimeMillis();
    this.wait(5000);
    System.out.println(" waited for iAmReceived: " + (System.
        currentTimeMillis() - start) + " ms");
  }

  // Another way to get to the list of devices
  // return localDevice.getRemoteDevices();
}

@SuppressWarnings("unchecked")
private void printDevices() throws BACnetException {
  for (RemoteDevice d : remoteDevices) {

    localDevice.getExtendedDeviceInformation(d);

    List<ObjectIdentifier> oids = ((SequenceOf<ObjectIdentifier>)
        localDevice.sendReadPropertyAllowNull(
```

```
        d, d.getObjectIdentifier(), PropertyIdentifier.objectList)).
            getValues();

    PropertyReferences refs = new PropertyReferences();
    // add the property references of the "device object" to the list
    refs.add(d.getObjectIdentifier(), PropertyIdentifier.all);

    // and now from all objects under the device object >> ai0, ai1,
        bi0,bi1...
    for (ObjectIdentifier oid : oids) {
      refs.add(oid, PropertyIdentifier.all);
    }

    System.out.println("Start read properties");
    final long start = System.currentTimeMillis();

    PropertyValues pvs = localDevice.readProperties(d, refs);
    System.out.println(String.format("Properties read done in %d ms",
        System.currentTimeMillis() - start));
    printObject(d.getObjectIdentifier(), pvs);
    for (ObjectIdentifier oid : oids) {
      printObject(oid, pvs);
    }


  }

}

private void printObject(ObjectIdentifier oid, PropertyValues pvs) {
  System.out.println(String.format("\t%s", oid));
  for (ObjectPropertyReference opr : pvs) {
    if (oid.equals(opr.getObjectIdentifier())) {
      System.out.println(String.format("\t\t%s = %s", opr.
          getPropertyIdentifier().toString(), pvs.getNoErrorCheck(opr
          )));
    }

  }
}



/**
 * @return the loopDevice
 */
public LoopDevice getLoopDevice() {
  return loopDevice;
}

/**
 * @param loopDevice the loopDevice to set
 */
public void setLoopDevice(LoopDevice loopDevice) {
```

```
    this.loopDevice = loopDevice;
  }

  private static void TestUsePolling() {
    //get list of devices, loop through checking states
  }

  private static void TestUseNaive() {

  }

}
```

```
/*****************************************
*
* ./ffa/model/Plugin.java
*
*****************************************/
package ffa.model;

public abstract class Plugin {

  boolean events;

  public boolean isEventDriven()
  {
    return events;
  }

  public abstract void prepare();
  public abstract void start();
}
```

```
/*****************************************
*
* ./ffa/model/RSPluginDemo.java
*
*****************************************/
package ffa.model;

public class RSPluginDemo {
  public static void main(String[] args)
  {
    EventLord lord = new DummyLord();
    RS232Plugin plugin = new RS232Plugin(lord, null);
    plugin.prepare();
```

```
    plugin.start();
  }
}
```

```
/*****************************************
*
* ./ffa/model/RS232Plugin.java
*
*****************************************/
package ffa.model;

/*Implementation of the RS232 plugin.  This enables communication
   between a fire panel and the eventlord.
 * This implementation is hardware-specific; some of the serial port
    settings (e.g. baud rate), as well as the string parsing
 * are dependent on at least the manufacturing brand, if not the
    individual machine.
 *
 * It takes the ASCII output of the fire panel, parses it for
    information about the date, time, sensor ID, etc., puts this
 * data into an Event object, and passes it along to the EventLord,
    which then performs the appropriate preprocessing and
 * distributing.
 *
 * At startup, it displays a window that offers the choice of all
    serial ports displayed on a system; need to work on displaying the
 * names of these ports, rather than their Object.toString(), which is
    just a memory address (yikes).
 *
 * Credit is due to rxtx.org; most of this code is based off their
    examples, and I got the implementation of the javax.comm API from
 * them.
 */


import java.io.IOException;
import java.io.InputStream;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Enumeration;
import java.util.HashSet;
import javax.swing.*;

import ffa.ui.Window;

import gnu.io.*;

public class RS232Plugin extends Plugin
{
```

```
EventLord lord; //EventLord
CommPortIdentifier ident; //Name of the serial port that will be used
Window window; // Window of the GUI; used to anchor the dialog box

//basic constructor
public RS232Plugin(EventLord theLord, Window theWindow)
{
  super.events = true;
  lord = theLord;
  window = theWindow;
}

/*
 * This is where the magic happens.  This method sets up the port and
     establishes the event listener.
 * The basic structure of this code comes from http://rxtx.qbang.org/
    wiki/index.php/Event_based_two_way_Communication, although
 * it is modified; since only one-way communication is necessary, and
     it performs more intensive manipulation of the data.
 */
public void start()
{
  ident = null;
  HashSet<CommPortIdentifier> commPorts = getAvailableSerialPorts();
     //Discovers available serial ports and puts them in an array
  if(commPorts == null || commPorts.isEmpty())  //If there are no
     serial ports available, exit
  {
    System.out.println("No ports available!");
    System.exit(0); //TODO: something more elegant to do?
  }

  String[] names = new String[commPorts.size()];
  int i = 0;
  for(CommPortIdentifier c : commPorts)
  {
    names[i] = c.getName();
    i++;
  }

  //Ask the user which serial port he wishes to use; TODO: find a
     better way to display port names
  try
  {
    ident = CommPortIdentifier.getPortIdentifier((String)JOptionPane.
       showInputDialog(window, "Select the serial port to use:", "
       Serial Port Selection", JOptionPane.QUESTION_MESSAGE, null,
       names, null));
  }
  catch(NoSuchPortException e)
  {
    System.out.println("Error: no such port");
    System.exit(0);
  }
```

```
  //houston, we have a problem
  if(ident == null || ident.isCurrentlyOwned())
  {
    System.err.println("Port nonexistent or in use");
    System.exit(0);
  }
  else
  {
    try{
      //Attempt to open the port, using the name Plugin to reserve it
          .
      CommPort commPort = ident.open("Plugin", 2000);

      if ( commPort instanceof SerialPort )
      {
        //Associate the serial port with the proper object, and set
            its properties to match the fire panel's
        //NOTE WELL: THESE SETTINGS ARE POTENTIALLY MACHINE-SPECIFIC
        SerialPort serialPort = (SerialPort) commPort;
        serialPort.setSerialPortParams(2400,SerialPort.DATABITS_7,
            SerialPort.STOPBITS_1,SerialPort.PARITY_EVEN);
        serialPort.setFlowControlMode(SerialPort.
           FLOWCONTROL_XONXOFF_IN);

        //Get input stream of serial port, associate it with the
            event listener, and start listening
        InputStream in = serialPort.getInputStream();

        serialPort.addEventListener(new SerialReader(in, lord));
        serialPort.notifyOnDataAvailable(true);

      }
      else
      {
        System.out.println("Error: Only serial ports are handled by
            this plugin.");
        System.exit(0);
      }
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }
}
/*
 * This inner class handles the input events of the Serial port.
 */
public static class SerialReader implements SerialPortEventListener
{
  private InputStream in;
  private byte[] buffer = new byte[1024];
  private EventLord lord;
```

```
//basic constructor
public SerialReader ( InputStream in, EventLord theLord)
{
  this.in = in;
  lord = theLord;
}

/*
 * When a serial event is activated, this event method is called.
    It reads the string until the EOL, parses the information,
 * and passes it to the event lord for further handling.
 */
public void serialEvent(SerialPortEvent arg0) {
  int data;
      String input;
      Event event = null;

  try
  {
    //Read the input
    int len = 0;
    while ( ( data = in.read()) > -1 )
    {
      if ( data == '\n' ) {
        break;
      }
      buffer[len++] = (byte) data;
    }
    input = new String(buffer,0,len);
    event = parseString(input);//Parse the string and create a new
       event
    lord.updateIncident(event);//pass the event to the event lord
       and we're done
  }
  catch ( IOException e )
  {
    e.printStackTrace();
    System.exit(-1);
  }
}
/*
 * This helper method for the SerialEvent method parses the actual
    string received from the serial port.
 * NOTE WELL: THIS PARSING IS POTENTIALLY MACHINE-SPECIFIC
 */
public Event parseString(String input)
{
  String[] inputArray;
  Date d = null;
  Event e = null;
  String date = null;
  SimpleDateFormat df = new SimpleDateFormat("MMddyyhh:mma");

  inputArray = input.split("\\s", 7);
```

```
    date = "" + inputArray[5].substring(0,6) + inputArray[4].
        substring(0,4);
    System.out.println(inputArray[4]);
    if(inputArray[4].charAt(4) == 'a')
      date += "am";
    else date += "pm";
    if(inputArray[0].equals("ALARM:"))
    {
      try {
        d = df.parse(date);
      } catch (ParseException ex) {
        // TODO Auto-generated catch block
        ex.printStackTrace();
      }
      e = new Event(d, new SystemSensor(new Location(null, 0, "
          location"),inputArray[2]));

    }
    return e;
  }
}

/*
 * This is a helper method for start(); it gathers a HashSet
     containing the CommPortIdentifiers associated with all available
 * serial ports.  This is lifted directly from http://rxtx.qbang.org/
     wiki/index.php/Discovering_available_comm_ports; no changes
 * were made.
 */
  public static HashSet<CommPortIdentifier> getAvailableSerialPorts()
      {
      HashSet<CommPortIdentifier> h = new HashSet<CommPortIdentifier
          >();
      Enumeration thePorts = CommPortIdentifier.getPortIdentifiers();
      while (thePorts.hasMoreElements()) {
          CommPortIdentifier com = (CommPortIdentifier) thePorts.
              nextElement();
          switch (com.getPortType()) {
          case CommPortIdentifier.PORT_SERIAL:
              try {
                  CommPort thePort = com.open("CommUtil", 50);
                  thePort.close();
                  h.add(com);
              } catch (PortInUseException e) {
                  System.out.println("Port, "  + com.getName() + ",
                      is in use.");
              } catch (Exception e) {
                  System.err.println("Failed to open port " +  com.
                      getName());
                  e.printStackTrace();
              }
          }
      }
      return h;
```

```
    }

  /*
   * This function is required by the abstract superclass Plugin; for
       this specific implementation, it does nothing.
   */
  public void prepare() {
    // TODO Auto-generated method stub

  }

}
```

```
/*****************************************
*
* ./ffa/model/Incident.java
*
*****************************************/
package ffa.model;

import java.util.Comparator;
import java.util.Date;
import java.util.concurrent.ConcurrentSkipListSet;

import ffa.ui.Util;


public class Incident {
  /*
   * First attempt at a signaling mechanism.
   */
  private boolean updated = false;
  private ConcurrentSkipListSet<Event> eventList = new
     ConcurrentSkipListSet<Event> (new EventComparator());
  private Date start;

  private class EventComparator implements Comparator<Event> {
    @Override
    public int compare(Event o1, Event o2) {
      int i = o1.getTime().compareTo(o2.getTime());
      if (i == 0)
        i = o1.getText().compareTo(o2.getText());
      if (i == 0)
        i = new Integer(o1.getTrigger().hashCode()).compareTo(o2.
            getTrigger().hashCode());

      return i;
    }
  }
```

```
public Incident () {} //default constructor -jwu

/**
 * @author jwu
 * @param start
 * Constructor for the incident with a start date
 */
public Incident(Date start) {
  this.eventList = new ConcurrentSkipListSet<Event> (new
     EventComparator());
  this.start = start;
}

/**
 * @author jwu
 * Currently like this to just copy functionality of testing
    previously in DynamicDisplay
 * @return the ConcurrentSkipListSet of events
 */
public ConcurrentSkipListSet<Event> getEvents() {
  return this.eventList;
}


public synchronized void waitForUpdate(){
  while(!updated) {
    try {
      wait();
    } catch (InterruptedException e) {
      //Should not be interrupted afaik
      if(Util.DEBUG)
        e.printStackTrace();
    }
  }

  updated = false;
}

public synchronized void update() {
  updated = true;
  notifyAll();
}

/**
 * @author jwu
 * @param event - The event that has updated the incident
 * Updates the incident by adding the event to the eventList
 */
public synchronized void update(Event event) {
  updated = true;
  this.eventList.add(event);
  notifyAll();
}
```

```
  /*Add an event WITHOUT updating, so that we don't have to
   *refresh the screen every time
   */
  public synchronized void addEvent(Event e)
  {
    this.eventList.add(e);
  }
}




/*****************************************
*
* ./ffa/model/SystemSensor.java
*
*****************************************/
package ffa.model;


/**
 * Created originally to give the alert that the building has burned
    down
 *  for testing.
 *
 * @author jwu
 * @date Feb. 16, 2010
 */
public class SystemSensor extends Sensor{

  public SystemSensor(Location location) {
    super(location);
  }
  public SystemSensor(String id)
  {
    super(id);
  }

  public SystemSensor(Location loc, String id)
  {
    super(loc, id);
  }
  public String toString() {
    return "<i>System Event</i>";
  }

}




/*****************************************
```

```
*
* ./ffa/model/Location.java
*
*****************************************/
package ffa.model;

import java.awt.Point;

/**
 * @author jwu
 * @date   Nov 16, 2009
 */
public class Location{

  private Point point;
  private int floor;
  private String description;

  public Location(Point point, int floor, String description) {
    super();
    this.point = point;
    this.floor = floor;
    this.description = description;
  }
  public void setPoint(Point point){//should only be used in annotation
      .
    this.point = point;
  }
  public Point getPoint() {
    return point;
  }
  public int getFloor() {
    return floor;
  }
  public String getDescription() {
    return description;
  }



}




/*****************************************
*
* ./ffa/model/Alert.java
*
*****************************************/
package ffa.model;

import java.util.Date;
```

```
/**
 * @author jwu
 * @date   Nov 16, 2009
 */
public class Alert extends Event {
  String message;

  public Alert(Date time, Sensor trigger, String message)
  {
    super(time,trigger);
    this.message = message;
  }

  @Override
  public String getText() {
    // TODO Auto-generated method stub
    return message;
  }

  @Override
  public String toString() {
    return message;
  }
}
```

```
/*****************************************
*
* ./ffa/model/BACNetPluginDemo.java
*
*****************************************/
package ffa.model;

import java.io.IOException;

public class BACNetPluginDemo {

  //public static String BROADCAST_ADDRESS = "192.168.1.24"; //TODO
     user provided
  public static String BROADCAST_ADDRESS = "192.168.1.255"; //TODO user
      provided


  public static void main(String[] args)
  {
    EventLord lord = new DummyLord();
    BACNetPlugin plugin = null;
    try {
      plugin = new BACNetPlugin(lord, BROADCAST_ADDRESS, 47808);
    } catch (IOException e) {
```

```
      e.printStackTrace();
    }

    plugin.prepare();
    plugin.start();
  }
}
```

```
/*****************************************
*
* ./ffa/model/Floor.java
*
*****************************************/
package ffa.model;

import java.awt.image.BufferedImage;
import java.util.HashMap;
import java.util.Map;

public class Floor {
  public BufferedImage image;

  public Map<SensorType, SensorLayer> layers;

  public Floor() {
    layers = new HashMap<SensorType,SensorLayer>();

    for(SensorType type : SensorType.values())
      layers.put(type,new SensorLayer(type));
  }

  public SensorLayer getLayer(SensorType type) {
    return layers.get(type);
  }
}
```

```
/*****************************************
*
* ./ffa/model/SensorType.java
*
*****************************************/
package ffa.model;

public enum SensorType{
  SMOKE("Smoke"),
  HEAT("Heat"),
```

```
  FLOW("Flow"),
  MANUAL("Manual");

  public final String title;

  private SensorType(String title){
    this.title = title;
  }
}
```

```
/*****************************************
*
* ./ffa/testing/PluginTesting.java
*
*****************************************/
package ffa.testing;

import java.awt.EventQueue;
import java.io.File;
import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;
import java.util.Date;

import ffa.model.*;
import ffa.annotation.*;
import ffa.ui.*;

public class PluginTesting {

  private static Window window;
  private static Plugin plugin;
  private static EventLord theLord;

  public static void initialize(int floors) {
    Incident incident = new Incident(new Date());
    window = new Window(incident);

    //TODO: Currently using just a new ArrayList<FireFilter>, change
        later
    ArrayList<FireFilter> fireFilters = new ArrayList<FireFilter>();

    theLord = new EventLord(incident, fireFilters);

    plugin = new RS232Plugin(theLord, window);

    try {
      EventQueue.invokeAndWait(new Runnable(){
        public void run() {
          window.initGUI();
          window.setVisible(true);
```

```
        }
      });
    } catch (InterruptedException e) {
      e.printStackTrace();
    } catch (InvocationTargetException e) {
      e.printStackTrace();
    }

  }

  public static void main(String[] args) {
    initialize(4);
    plugin.prepare();
    plugin.start();

    window.await();
  }
}
```

```
/*****************************************
*
* ./ffa/testing/FDSSensor.java
*
*****************************************/
package ffa.testing;

/*
 *  Temporary sensor class for FDS integration
 */
public class FDSSensor {
  private String xyz;
  private String name;
  private String type;
  private double alarmThreshold;
  private double troubleThreshold;

  public FDSSensor() {
    this.xyz = null;
    this.name = null;
    this.type = null;
    this.alarmThreshold = 0.0;
    this.troubleThreshold = 0.0;
  }

  public FDSSensor(String loc, String nm, String ty, double alarm,
     double trouble) {
    this.xyz = loc;
    this.name = nm;
    this.type = ty;
    this.alarmThreshold = alarm;
```

```
    this.troubleThreshold = trouble;
  }

  public String getXyz() {return xyz;}
  public void setXyz(String xyz) {this.xyz = xyz;}
  public String getName() {return name;}
  public void setName(String name) {this.name = name;}
  public String getType() {return type;}
  public void setType(String type) {this.type = type;}
  public double getAlarmThreshold() {return alarmThreshold;}
  public void setAlarmThreshold(double alarmThreshold) {this.
     alarmThreshold = alarmThreshold;}
  public double getTroubleThreshold() {return troubleThreshold;}
  public void setTroubleThreshold(double troubleThreshold) {this.
     troubleThreshold = troubleThreshold;}

  public String toString() {
    return "name=" + this.name + ",type=" + this.type + ",alarm=" +
       this.alarmThreshold + ",trouble=" + this.troubleThreshold;
  }
}
```

```
/*****************************************
*
* ./ffa/testing/EventTesting.java
*
*****************************************/
package ffa.testing;

import java.awt.EventQueue;
import java.awt.FileDialog;
import java.awt.Frame;
import java.awt.Point;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.Random;
import java.util.Scanner;
import java.util.TreeMap;

import junit.framework.TestCase;
import ffa.model.Alert;
import ffa.model.Event;
import ffa.model.EventLord;
```

```
import ffa.model.FireFilter;
import ffa.model.FloorFilter;
import ffa.model.Incident;
import ffa.model.Location;
import ffa.model.Sensor;
import ffa.model.SystemSensor;
import ffa.model.ThresholdSensor;
import ffa.ui.Window;

/**
 * @author jwu
 *
 */
public class EventTesting extends TestCase {

  private static final String TEST1 = "src/ffa/testing/test1.txt";
  private static final String PULSE1 = "src/ffa/testing/pulse1";
  private static final String KIMBUILDINGBURN = "src/ffa/testing/jmp.
     ffa";//"src/ffa/testing/Conf_video.ffa";//"src/ffa/testing/
     KimBuildingBurn";

  private ArrayList<Event> events = new ArrayList<Event>();
  private ArrayList<Long> timeBetweenEvents = new ArrayList<Long>();
    //timeBetweenEvents.get(i) = time between events.get(i-1) and
       events.get(i)

  private EventLord theLord;
  private Window window;

  public EventTesting() {
  }

  /**
   * @author jwu
   * initializes testing by creating the Incident, creating a Window,
      creating an EventLord
   * Starts the window and then returns
   */
  public void initialize(int floors) {
    Incident incident = new Incident();
    window = new Window(incident);

    //TODO: Currently using just a new ArrayList<FireFilter>, change
       later
    ArrayList<FireFilter> fireFilters = new ArrayList<FireFilter>();
    fireFilters.add(new FloorFilter(floors));

    theLord = new EventLord(incident, fireFilters);

    try {
      EventQueue.invokeAndWait(new Runnable(){
        public void run() {
          window.initGUI();
          window.setVisible(true);
```

```
        }
      });
    } catch (InterruptedException e) {
      e.printStackTrace();
    } catch (InvocationTargetException e) {
      e.printStackTrace();
    }

  }

  /**
   * @author jwu
   * Basic test to start the testing
   * parse event information from a file
   * start a thread to simulate the events
   * and have the window await
   */
/*  public void test1()
  {
    initialize(2);

    Thread t = getTestThread(new File(TEST1));
    t.start();

    window.await();
  }
*/
  /**
   * Generates a test file...not quite working due to not writing to
      file for some reason...
   * Will add more parameters later to change how the simulation runs
   * Very basic right now
   */
/*  public void testGeneratePulse()
  {
    pulseGenerate(4, 400, new File(KIMBUILDINGBURN));
  }
*/


  /**
   * Basically the same as other test, but uses pulse file instead
   */
/*  public void testPulse2()
  {
    initialize(10);

    Thread t = getTestThread(new File(PULSE1));
    t.start();

    window.await();
  }
*/
```

```
public void testBurnKim() {
  initialize(4);
  Thread t = getTestThread(new File(KIMBUILDINGBURN));
  t.start();

  window.await();
}



public void pulseGenerate(int floors, int numSensors, File f)
{

  try {
    BufferedWriter output = new BufferedWriter(new FileWriter(f));
    int topFloor = 0;
    int activatedSensors = 0;
    boolean stalling = true;

    long timeBetween = 0;
    long randomTimeBetween = 0;
    int numberToDo = 0;
    int numberDone = 0;
    double chanceOfActivate = 0.0;
    Random r = new Random();

    System.out.println(0 + " " + 0 + " Threshold First Event");
    output.write(0 + " " + 0 + " Threshold First Event");
    output.newLine();

    while(topFloor < floors && numberDone <= numSensors)
    {
      if(activatedSensors >= ((numSensors / floors) / 10) + r.nextInt
         (((numSensors / floors) / 10)))
      {
        activatedSensors = 0;
        topFloor++;
      }

      if(stalling)
      {
        timeBetween = 10000;
        randomTimeBetween = 1000;
        numberToDo = r.nextInt(5); //(numSensors / floors) / 10 + 1;
        chanceOfActivate = 0.7;
      } else {
        timeBetween = 100;
        randomTimeBetween = 1000;
        numberToDo = (numSensors / floors) / 4;
        chanceOfActivate = 0.4;
      }
      //System.out.println("number to write = " + numberToDo);
      for(int i = 0; i < numberToDo; i++)
      {
```

```
        long time = timeBetween + r.nextInt((int)randomTimeBetween);
        int floor = r.nextInt(topFloor+1);
        String message = "";
        if(r.nextDouble() < chanceOfActivate)
        {
          message = "In Alarm";
          activatedSensors++;
        } else {
          message = "In Trouble";
          activatedSensors++;
        }
        //writer.write(time + " " + floor + " Threshold " + message);
        //writer.newLine();
        System.out.println(time + " " + floor + " Threshold " +
            message);
        output.write(time + " " + floor + " Threshold " + message);
        output.newLine();
      }

      stalling = !stalling;
    }


    output.write(2010 + " " + (floors-1) + " System " + "<Signal Lost
        .>");
    output.newLine();

    output.close();
  }catch(Exception e) {
    e.printStackTrace();
  }


}


/**
 * Expected format of a line of the file is:
 * <Number of milliseconds compared to previous event> <Floor> <
    Sensor Type> <Message>
 * @author jwu
 * @param f
 */
private void parseFile(File f) {
  try {
    Scanner s = new Scanner(f);
    String str;
    long timeBetween;
    Sensor sensor;
    Event alert;

    long total = System.currentTimeMillis();

    while(s.hasNextLine()) {
```

```
        //get time
        str = s.next();
        timeBetween = Long.parseLong(str);
        this.timeBetweenEvents.add(timeBetween);
        total += timeBetween;

        str = s.next();
        int floor = Integer.parseInt(str);

        str = s.next();
        String sensorLabel = str;

        //get sensor information
        str = s.next();


        if(str.equals("Threshold")) {
          // New sensor ID
          String sensorID = s.next();
          // Prune parens
          sensorID = sensorID.substring(1, sensorID.length()-1);

          str = s.nextLine();
          sensor = new ThresholdSensor(new Location(null, floor, str.
              trim()), sensorID);
        } else if (str.equals("System")) {
          str = s.nextLine();
          sensor = null;
          //sensor = new SystemSensor(new Location(null, floor, str.
              trim()), sensorLabel);
        }else {
          throw new IllegalArgumentException(); //unrecognized sensor
        }




        //get message
        if (sensor != null) {
          alert = new Event(new Date(total), sensor);
          this.events.add(alert);
        }

        //System.out.println(str);
      }
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }

}

/**
 * @author jwu
```

```
 * @param f - the file from which to get event information for this
     thread
 * @return a thread that will be run for testing that does the events
     described in passed file
 */
private Thread getTestThread(File f) {

  parseFile(f);

  return new Thread(new Runnable(){
    public void run() {

      Iterator<Event> eventIterator = events.iterator();
      Iterator<Long> timeIterator = timeBetweenEvents.iterator();
      while(eventIterator.hasNext()) {
        //theLord.updateIncident(new Alert(new Time(0),
        //    new ThresholdSensor(), Double.toString(Math.random())))
            ;

        try {
          Thread.sleep(timeIterator.next());
        } catch (InterruptedException e) {
          e.printStackTrace();
        }

        theLord.updateIncident(eventIterator.next());

        //Util.printDebug("Updating");
        //incident.update();
      }
    }
  });
}


//runs a test file
public static void main(String [] args) {
  /*FileDialog f = new FileDialog((Frame)null, "Select the annotation
      file");
  f.setMode(FileDialog.LOAD);

  f.setVisible(true);
  File annotation_file = new File(f.getDirectory()+f.getFile());
  */
  FileDialog f = new FileDialog((Frame)null, "Select the ffa file");
  f.setMode(FileDialog.LOAD);

  f.setVisible(true);
  File ffa_file = new File(f.getDirectory()+f.getFile());

  EventTesting e = new EventTesting();
  e.initialize(1);
  Thread t = e.getTestThread(ffa_file);
  t.start();
```

```
    e.window.await();

  }
}
```

```
/*****************************************
*
* ./ffa/testing/FDSPlugin.java
*
*****************************************/
package ffa.testing;

import java.awt.FileDialog;
import java.awt.Frame;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.TreeMap;

import javax.swing.JFileChooser;
import javax.swing.JOptionPane;


public class FDSPlugin {

  public static File fdsInput;
  public static File fdsOutput;
  public static File parseOutput;

  public static TreeMap<String, FDSSensor> sensors = new TreeMap<String
     , FDSSensor>(); //every sensor in the simulation will have a name
      to FDSSensor mapping here

  public static void main(String [] args) {

    //choose to use existing ffa file or make new one
//    int choice = 0;
//    String input = JOptionPane.showInputDialog("Enter choice:\n0 -
   Use existing .ffa file\n1 - Enter FDS files to generate and run a .
   ffa file");
//    try {
//      choice = Integer.parseInt(input);
//      if( choice != 0 && choice != 1)
//        throw new IllegalArgumentException();
//    } catch (Exception e) {
```

```
//      JOptionPane.showMessageDialog(null, "Invalid option.");
//      System.exit(0);
//    }
//
//    if(choice == 1) {

      //get FDS input file
      fdsInput = getFile("Select FDS Input file (.fds file)", true,
          null);
      fdsOutput = getFile("Select FDS Output file (.cvs file)", true,
          null);
      parseOutput = getFile("Save the resulting .ffa file as...", false
          , ".ffa");

      //TODO delete
      System.out.println(fdsInput.toString() + "\n" + fdsOutput.
          toString() + "\n" + parseOutput.toString());

      //parse input file for sensors
      parseForSensors();

      //parse output file for sensor activation times (generate .ffa
          file)
      generateFFA();
//    } else {
//      //just use an existing .ffa file
//      parseOutput = getFile("Select the existing file to simulate (.
    ffa file)", true, null);
//
//      //TODO delete
//      System.out.println(parseOutput.toString());
//    }

//    JOptionPane.showMessageDialog(null, "program termination");
    System.exit(0);
  }

  /*
   * Get's a file to be used for something, if open_save is true,
       displays open dialog, otherwise displays save dialog
   */
  private static File getFile(String titleMessage, boolean open_save,
      String file_ext) {
    FileDialog f = new FileDialog((Frame)null, titleMessage);
    if(open_save)
      f.setMode(FileDialog.LOAD);
    else
      f.setMode(FileDialog.SAVE);

    f.setVisible(true);
    String filename = f.getDirectory() + f.getFile();
    if(file_ext != null) {
      File ret;
      if(filename.endsWith(file_ext))
```

```
      ret = new File(filename);
    else
      ret = new File(filename + file_ext);
    return ret;
  }
  System.out.println(f.getDirectory() + " ... " + f.getFile());
  return new File(f.getDirectory()+f.getFile());
}

/*
 * This is called once we have a mapping of device names to
    associated FDSSensors (which contain threshold information)
 *
 * at every step, get the tokens, and check each token to see if a
    new event has occurred
 *    check if sensor dead (if so skip)
 *    check if in trouble (if so, set to dead, skip)
 *    check if in alarm (if so and not already in alarm, set to alarm
    )
 */
private static final double IN_STATE = -1;
private static void generateFFA() {
  //create array of device names, array of alarmsThresholds, array of
      troubleThresholds
  //whenever a sensor enters a state, the appropriate array will be
     modified to have -1 so that the check isn't made again
  //when all sensors enter trouble, simulation terminates
  String [] deviceNames;
  double [] alarmThresholds;
  double [] troubleThresholds;
  String [] current;
  long lastEventTime = 0;
  boolean firstEventOccurred = false;
  long intervalTime = 0;
  long currentTime = 0;

  try {
    Scanner sc = new Scanner(fdsOutput);
    BufferedWriter out = new BufferedWriter(new FileWriter(
       parseOutput));
    String line;

    sc.nextLine(); //must have the units line
    line = sc.nextLine();

    //init device names, alarmThresholds, troubleThresholds
    deviceNames = line.split("\",\"|,\"|\"");
    alarmThresholds = new double[deviceNames.length];
    troubleThresholds = new double[deviceNames.length];
    for(int i = 1; i < deviceNames.length; i++) {
      String s = deviceNames[i];
      alarmThresholds[i] = sensors.get(s).getAlarmThreshold();
      troubleThresholds[i] = sensors.get(s).getTroubleThreshold();
    }
```

```
//TODO multiple floors for first event
String sensor_name;
while(sc.hasNextLine()) {
  line = sc.nextLine();
  current = line.split(", ");
  int alive_sensors = 0; //counts number of alive sensors in each
      round, if all dead ends parsing
  double cur_sens_val = 0.0;
  current[0] = current[0].trim();

  System.out.println("current[0] = " + current[0]); //TODO remove

  currentTime = (long)(eToDouble(current[0]) * 1000);
  intervalTime = currentTime - lastEventTime; //TODO can be moved
      into a if body for efficiency

  for(int i = 1; i < current.length; i++) {
    //parse current values and compare with threshold values, if
        an event occurs, write it

    if(troubleThresholds[i] == -1) //dead sensor, no processing
        needed
      continue;
    cur_sens_val = eToDouble(current[i]);

    if(cur_sens_val >= troubleThresholds[i]) {
      //sensor has entered trouble state
      if(!firstEventOccurred) { //if first event, write
          firstEvent alert
        out.write(intervalTime + " 0 smk1 System First Event");
        out.newLine();
        intervalTime = 0;
        firstEventOccurred = true;
      }

      sensor_name = sensors.get(deviceNames[i]).getName();
      out.write( "" + intervalTime +
          " 0 " +   //TODO multiple floors
          sensors.get(deviceNames[i]).getType() + sensor_name.
              charAt(sensor_name.length()-1) +
          " Threshold" +
          " (" + sensor_name +
      ") In Trouble");
      out.newLine();
      troubleThresholds[i] = -1;

      lastEventTime = currentTime;

    } else if(alarmThresholds[i] != -1 && cur_sens_val >=
        alarmThresholds[i]) {
      //not already in alarm, and going into alarm
      if(!firstEventOccurred) { //if first event, write
          firstEvent alert
```

```
            out.write(intervalTime + " 0 smk1 System First Event");
            out.newLine();
            intervalTime = 0;
            firstEventOccurred = true;
          }

          sensor_name = sensors.get(deviceNames[i]).getName();
          out.write( "" + intervalTime +
              " 0 " +   //TODO multiple floors
              sensors.get(deviceNames[i]).getType() + sensor_name.
                  charAt(sensor_name.length()-1) +
              " Threshold" +
              " (" + sensor_name +
          ") In Alarm");
          out.newLine();
          alarmThresholds[i] = -1;

          lastEventTime = currentTime;
          alive_sensors++;
        } else
          alive_sensors++; //no change in state
      }
    }


    out.write("0 0 smk1 System Simulation End");
    out.newLine();
    //Close the output stream
    out.close();
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  }

  System.out.println("finishing generateFFA");
}

/*
 * given a parameter of the form d.ddddddddE(+|-)ddd convert to an
    return a double
 */
private static double eToDouble(String str) {
  System.out.print("e2d: " + str); //TODO remove
  String [] toks = str.split("E");
  double num = Double.parseDouble(toks[0]);
  int power = Integer.parseInt(toks[1].substring(1));
  if(toks[1].charAt(0) == '-')
    power *= -1;
  double result = num * Math.pow(10, power);
  System.out.println(" == " + result); //TODO remove
  return result;
}
```

```
private static void parseForSensors() {
  ArrayList<String> devc = new ArrayList<String>();
  TreeMap<String, FDSSensor> propid = new TreeMap<String, FDSSensor
      >();
  StringBuilder str = new StringBuilder();
  try {
    Scanner sc = new Scanner(fdsInput);
    String line;
    String [] tokens;

    while(sc.hasNextLine()) {
      line = sc.nextLine();
      str = new StringBuilder(line);
      while(!line.endsWith("/")) {
        line = sc.nextLine();
        str.append(line);
      }

      System.out.println(str.toString()); //TODO remove

      if(str.toString().startsWith("&DEVC")) {
        System.out.println("devc found"); //TODO remove
        devc.add(str.toString()); //can't be processed until after
            reading whole file

      } else if (str.toString().startsWith("&PROP")) {
        System.out.println("prop found"); //TODO remove
        FDSSensor ps = new FDSSensor();

        //TODO get xyz

        //get sensor type name
        tokens = str.toString().split("'");
        String sensorTypeName = tokens[1];
        System.out.println("name = " + sensorTypeName);

        //get activation
        tokens = str.toString().split(".+ACTIVATION\\D+");
        String activation = tokens[1].substring(0, tokens[1].indexOf
            (" "));
        if(activation.endsWith("."))
          activation = activation.substring(0, activation.length()-1)
              ;
        System.out.println("activation = " + activation); //TODO
            remove
        ps.setAlarmThreshold(Double.parseDouble(activation));

        //get trouble
        ps.setTroubleThreshold(Double.MAX_VALUE);

        //get type
        if(str.toString().contains("'LINK TEMPERATURE'"))
          ps.setType("het");//ps.setType("Heat");
```

```
          else if(str.toString().contains("'SPRINKLER LINK TEMPERATURE
              '"))
            ps.setType("flo");//ps.setType("Sprinkler");
          else if(str.toString().contains("'CHAMBER OBSCURATION'"))
            ps.setType("smk");//ps.setType("Smoke");
          else
            JOptionPane.showMessageDialog(null, "unrecognized sensor
                type: \n" + str.toString());
          System.out.println("sensor type = " + ps.getType()); //TODO
              remove

          propid.put(sensorTypeName, ps);
        }


      }

      //TODO remove
      System.out.println("propid: " + propid.toString());
      System.out.println("devc: " + devc.toString());

      //parse devices now that prop id information known
      for(String s : devc) {
        String [] toks = s.split("'");
        FDSSensor prop = propid.get(toks[3]);
        FDSSensor p = new FDSSensor(prop.getXyz(), toks[1], prop.
            getType(), prop.getAlarmThreshold(), prop.
            getTroubleThreshold());

        System.out.println("new devc = " + p.toString()); //TODO remove
        sensors.put(p.getName(), p);
      }

    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }
  }




}




/*****************************************
*
* ./ffa/ui/MetaInfoLayer.java
*
*****************************************/
```

```
/*
Levon K. Mkrtchyan

This Component displays annotation stuffs and
accepts annotation commands
*/

package ffa.ui;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Polygon;
import java.awt.Rectangle;
import java.awt.Stroke;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.geom.AffineTransform;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.util.Iterator;

import javax.swing.JComponent;

import ffa.model.Building;
import ffa.model.Floor;
import ffa.model.SensorLayer;
import ffa.model.SensorType;
import ffa.ui.MapDisplay;

import ffa.annotation.Zone;

public class MetaInfoLayer extends JComponent{
  static final long serialVersionUID = 1L;

  private ClockPanel clockPanel = null;
  private MapDisplay display = null;

  private Building building;
  private Integer curFloor = 0;

  private ComponentListener cl = new ComponentAdapter(){
    public void componentResized(ComponentEvent e){
      MetaInfoLayer.this.setSize(e.getComponent().getWidth(), e.
          getComponent().getHeight());
    }
```

```
};

public MetaInfoLayer(MapDisplay display,ClockPanel cp){
  clockPanel = cp;
  this.display = display;
  this.setOpaque(false);
  display.add(this,new Integer(1));
  display.addComponentListener(cl);
  this.setBackground(new Color(255,255,255,Color.TRANSLUCENT));
  this.setForeground(new Color(255,255,255,Color.TRANSLUCENT));

  setBuilding(display.building);
}

public void paint(Graphics g){
  Graphics2D g2d = (Graphics2D) g;
  Stroke oldStroke = g2d.getStroke();

  AffineTransform original = g2d.getTransform();

  g2d.transform(display.getTransform());

  //float zoom = display.getZoom();

  Floor floor = getBuilding().getFloor(curFloor);

  for (SensorType layerType : SensorType.values()){ //floor.layers.
     keySet()) {
    Color outline = Color.black;
    Color fill;
    switch(layerType){
    case HEAT:
      outline = new Color(0,0,0,0);
      fill = Color.red;
      break;
    case SMOKE:
      outline = new Color(0,0,0,0);
      fill = new Color(Color.gray.getRed(),Color.gray.getGreen(),
          Color.gray.getBlue(),150);
      break;
    case FLOW:
      g2d.setStroke(new BasicStroke(10f));
      outline = Color.blue;
      fill = new Color(0,0,0,0);
      break;
    default://manual
      outline = Color.green;
      fill = new Color(0,0,0,0);
      break;
    }

    for(Zone z : floor.getLayer(layerType).getZones()){
      int opacity = z.getOpacity(clockPanel.getClockTime());
      if(opacity>0){
```

```
            //Color transparent = new Color(opaque.getRed(),opaque.
                getGreen(),opaque.getBlue(),opacity);
            g.setColor(fill);
            g.fillPolygon(z.getZone());
            g.setColor(outline);
            g.drawPolygon(z.getZone());
          }
        }
        g2d.setStroke(oldStroke);
      }

      g2d.setTransform(original);
    }
    public void setBuilding(Building building) {
      this.building = building;
    }
    public Building getBuilding() {
      return building;
    }
}




/*****************************************
*
* ./ffa/ui/ClockPanel.java
*
*****************************************/
package ffa.ui;

import info.clearthought.layout.TableLayout;

import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Font;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingConstants;

import org.joda.time.DateTime;
import org.joda.time.Interval;
import org.joda.time.Period;
import org.joda.time.PeriodType;

@SuppressWarnings("serial")
public class ClockPanel extends JPanel{
  private DateTime start;
  private boolean running = true;
  private Period duration = null;
```

```
private MapDisplay display = null;

JLabel msgLabel;
JLabel clockLabel;

DateFormat timeFormat = new SimpleDateFormat("kk:mm:ss");

/*
 * Static panel - doesn't do anything
 */
public ClockPanel(Date date, String msg) {
  super();
  Util.checkEDT();
  msgLabel = new JLabel(msg);
  final String text = timeFormat.format(date);
  clockLabel = new JLabel(text);

  layoutPanel();
}
/*
 * Dynamic panel, will update time
 */
public ClockPanel(DateTime start, String msg, MapDisplay disp) {
  super();
  Util.checkEDT();
  this.start = start;

  msgLabel = new JLabel(msg);
  clockLabel = new JLabel("");
  display = disp;

  layoutPanel();

  start();
}

private void layoutPanel() {
  clockLabel.setBackground(new Color(51,51,51));
  clockLabel.setOpaque(true);
  clockLabel.setForeground(Color.white);
  clockLabel.setHorizontalAlignment(SwingConstants.CENTER);

  msgLabel.setFont(msgLabel.getFont().deriveFont(15f));
  clockLabel.setFont(clockLabel.getFont().deriveFont(25f).deriveFont(
      Font.BOLD));

  double border = 15;
  double iborder = 10;
  double[][] size = {{border,TableLayout.FILL,border},{border,.25,
      iborder,TableLayout.FILL,border}};
  setLayout(new TableLayout(size));

  add(msgLabel,"1,1,c,c");
  add(clockLabel,"1,3,f,f");
```

```
  }

  public void start() {
    Thread t = new Thread(new Runnable(){
      public void run() {
        while(running) {
          try {
            Thread.sleep(1000);
          } catch (InterruptedException e) {
            e.printStackTrace();
          }
          ClockPanel.this.tick();
        }
      }
    });
    t.start();
  }

  private void tick() {
    DateTime current = new DateTime();

    duration = new Interval(start, current).toPeriod(PeriodType.time())
        ;

    final String text = String.format("<HTML><CENTER>%02d:%02d:%02d",
        duration.getHours(),duration.getMinutes(),duration.getSeconds()
        );
    EventQueue.invokeLater(new Runnable(){
      public void run() {
        clockLabel.setText(text);
      }
    });

    display.repaint(); //repaint the map every timestep
  }

  public DateTime getClockTime(){
    return start.plus(duration);
  }
}
```

```
/*****************************************
*
* ./ffa/ui/DynamicDisplay.java
*
*****************************************/
package ffa.ui;

import java.awt.Color;
import java.awt.GridLayout;
```

```
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.concurrent.ConcurrentSkipListSet;

import javax.swing.BorderFactory;
import javax.swing.JLabel;
import javax.swing.JPanel;

import ffa.model.Event;

@SuppressWarnings("serial")
public class DynamicDisplay extends JPanel {
  private ComponentList eventDisplayList;
  private MapDisplay mapDisplay;

  DateFormat timeFormat = new SimpleDateFormat("kk:mm:ss");

  public DynamicDisplay() {
    Util.checkEDT();

    eventDisplayList = new ComponentList();

    setBorder(BorderFactory.createLineBorder(Color.GREEN));

    setLayout(new GridLayout(1,1));

    add(eventDisplayList.getScrollPane());
  }

  public void addMapDisplay(MapDisplay md){
    mapDisplay = md;
  }

  /**
   * @author jwu
   * @param events - the list of events to reflect changes with
   * Modified to take a list of Events since testing was moved to ffa.
      testing
   */
  public void changeMade(ConcurrentSkipListSet<Event> events) {
    Util.checkEDT();

    eventDisplayList.clear();
    for(Event e : events) {
      eventDisplayList.add(getComponent(e));
      mapDisplay.building.addEvent(e);
    }
    eventDisplayList.refresh();
    mapDisplay.repaint();
  }

  /**
   * @author jwu
   * @param e
```

```
   * @return panel
   * Modified to take Events as opposed to EventStubs
   */
  public JPanel getComponent(Event e) {
    JPanel panel = new JPanel();
    panel.setLayout(new GridLayout(1,1));
    panel.add(new JLabel(e.getText()));
    panel.setBorder(BorderFactory.createEtchedBorder());
    return panel;
  }
}
```

```
/*****************************************
*
* ./ffa/ui/MapDisplay.java
*
*****************************************/
package ffa.ui;

import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.RenderingHints;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseWheelEvent;
import java.awt.geom.AffineTransform;
import java.awt.geom.Point2D;
import java.awt.image.BufferedImage;
import java.io.File;

import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.JLayeredPane;
import javax.swing.JPanel;

import ffa.annotation.AnnotationIO;
import ffa.model.Building;

@SuppressWarnings("serial")
public class MapDisplay extends JLayeredPane{
  private java.awt.Window window;

  private JPanel mapPanel;
  private ControlPanel controlPanel;
```

```
private final String floorPlanRootPath = "staticInfo/floor";
private final String floorPlanExt = ".png";
private final String annotationPath = "staticInfo/annotation";

private AffineTransform inverse = null;

public Building building = null;

public MouseAdapter mapMouseListener = new MouseAdapter(){
  Point2D prev=null;
  boolean exited=true;

  public void mouseEntered(MouseEvent e){
    window.setCursor(new Cursor(Cursor.HAND_CURSOR));
    exited=false;
  }

  public void mouseExited(MouseEvent e){
    window.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    prev=null;
    exited=true;
  }

  public void mousePressed(MouseEvent e){
    window.setCursor(new Cursor(Cursor.MOVE_CURSOR));
    prev=screenToMap(e.getPoint());
  }

  public void mouseDragged(MouseEvent e){
    if(prev!=null){
      Point2D cur=screenToMap(e.getPoint());
      controlPanel.pan((int) (cur.getX()-prev.getX()),(int) (cur.getY
          ()-prev.getY()));
      prev=screenToMap(e.getPoint());
    }
  }

  public void mouseReleased(MouseEvent e){
    if(!exited){
      window.setCursor(new Cursor(Cursor.HAND_CURSOR));
      prev=null;
    }
  }

  public void mouseWheelMoved(MouseWheelEvent e){
    controlPanel.zoomBy(Math.pow(.9,e.getWheelRotation()));
    repaint();
  }

  public void mouseClicked(MouseEvent e){
    if(e.getButton()==MouseEvent.BUTTON2){
      controlPanel.zoomDefault();
      repaint();
```

```
    }
  }
};

public void setCursor(Cursor cur){
  window.setCursor(cur);
}

public void recomputeInverse(){
  inverse = getTransform();
  try{
    inverse = inverse.createInverse();
  }catch(Exception ex){
    ex.printStackTrace();
    inverse = null;
  }
}

public AffineTransform getTransform(){
  AffineTransform pannedZoomed = AffineTransform.getScaleInstance(
      controlPanel.getZoom(), controlPanel.getZoom());
  pannedZoomed.translate(controlPanel.getPanX(),controlPanel.getPanY
      ());
  return pannedZoomed;
}

public float getZoom(){
  return controlPanel.getZoom();
}

public Point2D screenToMap(Point pt){
  return inverse.transform(pt, null);
}

public int getCurrentFloor(){
  return controlPanel.getCurrentFloor();
}

public float computeScale(int floor){
  BufferedImage temp = building.getFloor(floor).image;
  return Math.min(mapPanel.getWidth()/(float) temp.getWidth(),
      mapPanel.getHeight()/(float) temp.getHeight());
}

public BufferedImage getFloorImage(int i){
  return building.getFloor(i).image;
}

public MapDisplay(java.awt.Window parentWindow, Building building) {
  this.window = parentWindow;
  Util.checkEDT();

  this.building = building;
```

```
    setBorder(BorderFactory.createLineBorder(Color.RED));

    //TODO decide on a size for the map
    setPreferredSize(new Dimension(600, 600));

    //setLayout(new GridLayout(2,1));
    mapPanel = new JPanel(){
      public void paint(Graphics g){
        Graphics2D g2d = (Graphics2D) g;
        g.setColor(Color.white);
        g.fillRect(0, 0, this.getWidth(), this.getHeight());
        g2d.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
            RenderingHints.VALUE_INTERPOLATION_BICUBIC);

        AffineTransform originalTransform = g2d.getTransform();

        AffineTransform pannedZoomed = getTransform();
        Point2D temp = pannedZoomed.transform(new Point(0,0), null);
        if(controlPanel.getWorkingImage()!=null)
          g.drawImage(controlPanel.getWorkingImage(),(int) temp.getX(),
              (int) temp.getY(), null);
        g2d.transform(pannedZoomed);

        //drawing state of fire goes here

        g2d.setTransform(originalTransform);

        g2d.transform(AffineTransform.getTranslateInstance(10, 10));
      }//end method draw
    };//end mapPanel anonymous class definition
    mapPanel.addComponentListener(new ComponentAdapter(){
      public void componentResized(ComponentEvent e){
        controlPanel.recomputeScale();
      }
    });

    mapPanel.addMouseListener(mapMouseListener);
    mapPanel.addMouseMotionListener(mapMouseListener);
    mapPanel.setSize(getWidth(), getHeight());

    this.addComponentListener(new ComponentAdapter(){
      public void componentResized(ComponentEvent e){
        mapPanel.setSize(getWidth(),getHeight());
      }
    });

    add(mapPanel,new Integer(0));

    controlPanel = new ControlPanel(this, building.floors.size());

    recomputeInverse();

    //add(controlPanel);
  }
```

```
}



/*****************************************
*
* ./ffa/ui/ComponentList.java
*
*****************************************/
package ffa.ui;

import info.clearthought.layout.TableLayout;

import java.awt.Component;
import java.awt.Dimension;

import javax.swing.JPanel;
import javax.swing.JScrollPane;

public class ComponentList {
  private JScrollPane scrollPane;

  private JPanel panel;

  private TableLayout layout;

  private static final double size[][] = {{TableLayout.FILL},{}};

  public ComponentList() {
    Util.checkEDT();

    panel = new JPanel();

    scrollPane = new JScrollPane(panel, JScrollPane.
        VERTICAL_SCROLLBAR_AS_NEEDED,
         JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

    layout = new TableLayout(size);
    panel.setLayout(layout);
  }

  public JScrollPane getScrollPane() {
    return scrollPane;
  }

  public void clear() {
    layout = new TableLayout(size);
    panel.setLayout(layout);

    panel.removeAll();
  }
```

```
  // Does not re-render
  public void add(Component c) {
    layout.insertRow(0, TableLayout.PREFERRED);
    panel.add(c, "0,0,f,t");
  }

  public void addLast(Component c) {
    int row = layout.getNumRow();
    layout.insertRow(row, TableLayout.PREFERRED);
    panel.add(c, String.format("0,%d,f,t", row));
  }

  public void refresh() {
    Util.checkEDT();
    panel.revalidate();
    panel.repaint();

  }

  public void setPreferredSize(Dimension d) {
    Util.checkEDT();
    panel.setPreferredSize(d);
    refresh();
  }

}
```

```
/*****************************************
*
* ./ffa/ui/Window.java
*
*****************************************/
package ffa.ui;

import info.clearthought.layout.TableLayout;

import java.awt.EventQueue;
import java.io.File;
import java.lang.reflect.InvocationTargetException;
import java.util.Date;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

import javax.swing.JFrame;

import org.joda.time.DateTime;

import ffa.annotation.AnnotationIO;
```

```
import ffa.model.Building;
import ffa.model.Incident;

@SuppressWarnings("serial")
public class Window extends JFrame {
  private ExecutorService eventExecutor;
  private StaticDisplay staticDisplay;
  private DynamicDisplay dynamicDisplay;
  private MapDisplay mapDisplay;

  private Future<?> eventThread;

  private Incident incident = new Incident();

  //Three columns, left and right are 25%, one row, fills screen
  private static final double size[][] = {{.25, TableLayout.FILL,
     .25},{TableLayout.FILL,.15}};

  public Window() {}


  /**
   * @author jwu
   * @param incident
   */
  public Window(Incident incident){
    this.incident = incident;
  }

  public void initGUI() {
    Util.checkEDT();

    /******** Load building file *********/
    File annotationFolder = new File("staticInfo");
    Building building = AnnotationIO.importAnnotation(annotationFolder)
       ;

    /******** Initialize Components **********/
    staticDisplay = new StaticDisplay(building);
    dynamicDisplay = new DynamicDisplay();
    mapDisplay = new MapDisplay(this, building);
    dynamicDisplay.addMapDisplay(mapDisplay);

    ClockPanel startTime = new ClockPanel(new Date(), "Time of First
       Alarm");
    ClockPanel elapsedTime = new ClockPanel(new DateTime(), "Elapsed
       Time", mapDisplay);
    new MetaInfoLayer(mapDisplay,elapsedTime);

    /******** Initialize Frame **********/
    // I believe this is the standard screen size we're working with
    this.setSize(1024, 768);

    //Centers the window
```

```
    this.setLocationRelativeTo(null);

    // This is the main app window
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    setLayout(new TableLayout(size));

    add(staticDisplay, "0,0");

    add(mapDisplay, "1,0,1,1");

    add(dynamicDisplay, "2,0");

    add(startTime,"0,1");

    add(elapsedTime,"2,1");


    /******** Initialize Event Thread **********/
    eventExecutor = Executors.newSingleThreadExecutor();
    eventThread = eventExecutor.submit(new Runnable() {
      @Override
      public void run() {
        eventThread();
      }
    });

    // TODO just for debug purposes, remove later

    /*
    Thread t = new Thread(new Runnable(){
      public void run() {
        while(true) {
          dynamicDisplay.tempAddStuff();
          try {
            Thread.sleep(2000);
          } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
          }
          Util.printDebug("Updating");
          incident.update();
        }
      }
    });
    t.start();  */


}

private void eventThread() {
  while(true) {
    incident.waitForUpdate();
    Util.printDebug("Updated");
```

```
      EventQueue.invokeLater(new Runnable(){
        public void run() {
          dynamicDisplay.changeMade(incident.getEvents());
        }
      });

      //TODO notify the various panels that there's something new going
          on
    }
  }

  public void await() {
    try {
      eventThread.get();
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (ExecutionException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    final Window window = new Window();
    try {
      EventQueue.invokeAndWait(new Runnable(){
        public void run() {
          window.initGUI();
          window.setVisible(true);

        }
      });
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (InvocationTargetException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    window.await();
  }

}




/*****************************************
*
* ./ffa/ui/Util.java
*
```

```
*****************************************/
package ffa.ui;

import javax.swing.SwingUtilities;

public class Util {
  // Allow compiler to remove debugging code
  public static final boolean DEBUG = true;

  public static void checkEDT() {
    if(!DEBUG)
      return;
    if(!SwingUtilities.isEventDispatchThread())
      throw new IllegalStateException();
  }

  public static void printDebug(String s) {
    if(!DEBUG)
      return;
    System.err.println(s);
  }
}
```

```
/*****************************************
*
* ./ffa/ui/StaticDisplay.java
*
*****************************************/
package ffa.ui;

import java.awt.Color;
import java.awt.GridLayout;

import javax.swing.BorderFactory;
import javax.swing.JLabel;
import javax.swing.JPanel;

import ffa.model.Building;

@SuppressWarnings("serial")
public class StaticDisplay extends JPanel {
  private ComponentList staticDisplayList;

  public StaticDisplay(Building building) {
    Util.checkEDT();

    setBorder(BorderFactory.createLineBorder(Color.BLACK));

    setLayout(new GridLayout(1,1));
```

```
    staticDisplayList = new ComponentList();

    add(staticDisplayList.getScrollPane());

    for (String info : building.staticInfo) {
      addEntry(info);
    }
  }

  private void addEntry(String text) {
    JPanel panel = new JPanel();
    panel.setLayout(new GridLayout(1,1));
    panel.add(new JLabel("<HTML>"+text));
    panel.setBorder(BorderFactory.createEtchedBorder());
    staticDisplayList.addLast(panel);
  }
}
```

```
/*****************************************
*
* ./ffa/ui/ControlPanel.java
*
*****************************************/
package ffa.ui;

import java.util.ArrayList;
import java.util.SortedSet;
import java.util.TreeSet;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.*;
import java.awt.Image;
import java.awt.image.BufferedImage;
import java.awt.GridLayout;
import java.awt.Point;

import javax.swing.BorderFactory;
import javax.swing.border.*;
import javax.swing.BoxLayout;
import javax.swing.ImageIcon;
import javax.swing.JButton;
//import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSlider;

//@SuppressWarnings("serial")
public class ControlPanel/* extends JPanel*/ {
  private MapDisplay display;
```

```
private JButton mapLeft;
private JButton mapRight;
private JButton mapUp;
private JButton mapDown;

private JButton zoomIn;
private JButton zoomOut;
private JButton zoomDefault;

private JPanel replayPanel;
private JButton replayRun;
private JSlider replayProgress;

private ArrayList<FloorButton> floorList = new ArrayList<FloorButton
   >();
private JPanel floorSelectPanel;
private SortedSet<FloorButton> troubleFloorList = new TreeSet<
   FloorButton>();
private JPanel troublePanel;

private ComponentList troubleFloorDisplayList;
private ComponentList floorDisplayList;

private float scale = 1.0f;
private float zoom = 1.0f;
private Point pan = new Point(0,0);
private int currentFloor = 0;
private Image workingImage = null;

private ImageIcon arrowIcon = new ImageIcon("images/arrow.png");
private ImageIcon blankIcon = new ImageIcon("images/blank10.png");

ActionListener mapActionListener = new ActionListener(){
  public void actionPerformed(ActionEvent e){
    String command = e.getActionCommand();
    if(command.equals("mapLeft")){
      pan.x+=50/zoom;
    } else if(command.equals("mapRight")){
      pan.x-=50/zoom;
    } else if(command.equals("mapUp")){
      pan.y+=50/zoom;
    } else if(command.equals("mapDown")){
      pan.y-=50/zoom;
    } else if(command.equals("zoomIn")){
      zoomBy(1.25f);
    } else if(command.equals("zoomOut")){
      zoomBy(.8f);
    } else if(command.equals("zoomDefault")){
      zoomDefault();
    } else if(command.matches("^f[0-9]+$")){
      int floor = Integer.parseInt(command.substring(1));

      JButton newButton = floorList.get(floor);
```

```
        newButton.setBackground(Color.green);
        newButton.setIcon(arrowIcon);

        JButton oldButton = floorList.get(currentFloor);
        oldButton.setBackground(new Color(0xAAFFFF));
        oldButton.setIcon(blankIcon);

        currentFloor=floor;
        recomputeScale();
      }
      display.recomputeInverse();
      display.repaint();
    }//end method actionPerformed
};

private void recomputeWorkingImage(){
    BufferedImage unscaled = display.getFloorImage(currentFloor);
    workingImage = unscaled.getScaledInstance(
        new Float(unscaled.getWidth()*zoom*scale).intValue(),
        new Float(unscaled.getHeight()*zoom*scale).intValue(),
        BufferedImage.SCALE_AREA_AVERAGING);
}

public void recomputeScale(){
    replayPanel.setLocation(display.getWidth()-replayPanel.getWidth()
        -15,display.getHeight()-replayPanel.getHeight()-15);

    floorSelectPanel.setLocation(display.getWidth()-floorSelectPanel.
        getWidth()-15,display.getHeight()/2-floorSelectPanel.getHeight
        ()/2);

    scale = display.computeScale(currentFloor);
    recomputeWorkingImage();
    display.recomputeInverse();
    display.repaint();
}

public void zoomBy(double zoomBy){
    zoom*=zoomBy;
    recomputeWorkingImage();
    display.recomputeInverse();
}

public void zoomDefault(){
    zoom=1.0f;
    recomputeWorkingImage();
    display.recomputeInverse();
}

public float getZoom(){
    return zoom*scale;
}

public int getPanX(){
```

```
  return pan.x;
}

public int getPanY(){
  return pan.y;
}

public void pan(int x, int y){
  pan.x+=x;
  pan.y+=y;
  display.recomputeInverse();
  display.repaint();
}

public Image getWorkingImage(){
  return workingImage;
}

public int getCurrentFloor(){
  return currentFloor;
}

public ControlPanel(MapDisplay display, int numFloors) {
  this.display = display;
  Util.checkEDT();

  Border emptyBorder = BorderFactory.createEmptyBorder();
  Color transparent = new Color(255,255,255,Color.TRANSLUCENT);

  mapLeft = new JButton(new ImageIcon("images/arrowLeft.png"));
  mapLeft.setActionCommand("mapLeft");
  mapLeft.addActionListener(mapActionListener);
  mapLeft.setBorder(emptyBorder);
  mapLeft.setBackground(Color.white);
  mapLeft.setPreferredSize(new Dimension(10,10));

  mapRight = new JButton(new ImageIcon("images/arrowRight.png"));
  mapRight.setActionCommand("mapRight");
  mapRight.addActionListener(mapActionListener);
  mapRight.setBorder(emptyBorder);
  mapRight.setBackground(Color.white);
  mapRight.setPreferredSize(new Dimension(10,10));

  mapUp = new JButton(new ImageIcon("images/arrowUp.png"));
  mapUp.setActionCommand("mapUp");
  mapUp.addActionListener(mapActionListener);
  mapUp.setBorder(emptyBorder);
  mapUp.setBackground(Color.white);
  mapUp.setPreferredSize(new Dimension(10,10));

  mapDown = new JButton(new ImageIcon("images/arrowDown.png"));
  mapDown.setActionCommand("mapDown");
  mapDown.addActionListener(mapActionListener);
  mapDown.setBorder(emptyBorder);
```

```
mapDown.setBackground(Color.white);
mapDown.setPreferredSize(new Dimension(10,10));

zoomIn = new JButton(new ImageIcon("images/zoomIn.png"));
zoomIn.setActionCommand("zoomIn");
zoomIn.addActionListener(mapActionListener);
zoomIn.setBorder(emptyBorder);
zoomIn.setBackground(Color.white);
zoomIn.setPreferredSize(new Dimension(20,20));

zoomOut = new JButton(new ImageIcon("images/zoomOut.png"));
zoomOut.setActionCommand("zoomOut");
zoomOut.addActionListener(mapActionListener);
zoomOut.setBorder(emptyBorder);
zoomOut.setBackground(Color.white);
zoomOut.setPreferredSize(new Dimension(20,20));

zoomDefault = new JButton(new ImageIcon("images/zoomDefault.png"));
zoomDefault.setActionCommand("zoomDefault");
zoomDefault.addActionListener(mapActionListener);
zoomDefault.setBorder(emptyBorder);
zoomDefault.setBackground(Color.white);
zoomDefault.setPreferredSize(new Dimension(20,20));

replayRun = new JButton(new ImageIcon("images/run.png"));
replayRun.setBorder(emptyBorder);
replayRun.setBackground(Color.white);
replayRun.setPreferredSize(new Dimension(20,20));

replayProgress = new JSlider();
replayProgress.setValue(0);
replayProgress.setBorder(emptyBorder);
replayProgress.setBackground(Color.white);
replayProgress.setPreferredSize(new Dimension(150,20));

floorDisplayList = new ComponentList();
troubleFloorDisplayList = new ComponentList();

genFloors(numFloors);
for(FloorButton f : floorList) {
  floorDisplayList.add(f);
}
JButton currentFloorButton = floorList.get(currentFloor);
currentFloorButton.setBackground(Color.green);
currentFloorButton.setIcon(arrowIcon);

floorDisplayList.refresh();

/*troubleFloorList.add(floorList.get(2));
//troubleFloorList.add(floorList.get(4));
troubleFloorList.add(floorList.get(3));
for(FloorButton f : troubleFloorList) {
  troubleFloorDisplayList.add(getComponent(f));
}
```

```
troubleFloorDisplayList.refresh();*/

JPanel movePanel = new JPanel(new GridLayout(3,3,10,10));
JPanel filler = new JPanel();
filler.setBackground(transparent);
movePanel.add(filler);
movePanel.add(mapUp);
filler = new JPanel();
filler.setBackground(transparent);
movePanel.add(filler);
movePanel.add(mapLeft);
filler = new JPanel();
filler.setBackground(transparent);
movePanel.add(filler);
movePanel.add(mapRight);
filler = new JPanel();
filler.setBackground(transparent);
movePanel.add(filler);
movePanel.add(mapDown);
filler = new JPanel();
filler.setBackground(transparent);
movePanel.add(filler);
movePanel.setBounds(15,15,50,50);
movePanel.setBackground(transparent);
display.add(movePanel,new Integer(1));

JPanel zoomPanel = new JPanel(new GridLayout(3,1,5,5));
zoomPanel.add(zoomIn);
zoomPanel.add(zoomDefault);
zoomPanel.add(zoomOut);
zoomPanel.setBackground(transparent);
zoomPanel.setBounds(30,100,20,70);
display.add(zoomPanel,new Integer(1));

replayPanel = new JPanel();
replayPanel.setLayout(new BoxLayout(replayPanel,BoxLayout.X_AXIS));
replayPanel.add(replayRun);
replayPanel.add(replayProgress);
replayPanel.setBackground(transparent);
replayPanel.setBounds(-200,-200,
    replayRun.getPreferredSize().width + replayProgress.
        getPreferredSize().width,
    Math.max(replayRun.getPreferredSize().height,replayProgress.
        getPreferredSize().height));
display.add(replayPanel,new Integer(1));

JLabel floorSelectLabel = new JLabel("Floors");
//floorSelectLabel.setHorizontalTextPosition(JLabel.LEADING);

JScrollPane floorScroll = floorDisplayList.getScrollPane();
floorScroll.setBorder(emptyBorder);
floorScroll.setBackground(transparent);
floorScroll.setPreferredSize(new Dimension(floorScroll.
   getPreferredSize().width+10,Math.min(100,floorScroll.
```

```
        getPreferredSize().height)));

    floorSelectPanel = new JPanel();
    floorSelectPanel.setLayout(new BoxLayout(floorSelectPanel,BoxLayout
        .Y_AXIS));
    floorSelectPanel.add(floorSelectLabel);
    floorSelectPanel.add(floorScroll);
    floorSelectPanel.setBackground(Color.white);
    floorSelectPanel.setBounds(-200,-200,
        55,
        //Math.max(floorScroll.getPreferredSize().width,
            floorSelectLabel.getPreferredSize().width),
        floorScroll.getPreferredSize().height+floorSelectLabel.
            getPreferredSize().height);
    display.add(floorSelectPanel,new Integer(1));

    /*add(mapLeft);
    add(mapRight);
    add(mapUp);
    add(mapDown);
    add(zoomIn);
    add(zoomDefault);
    add(zoomOut);
    add(replayRun);
    add(replayProgress);
    add(floorDisplayList.getScrollPane());
    add(troubleFloorDisplayList.getScrollPane());*/

  }

  private void genFloors(int maxFloor) {
    Color fColor = new Color(0xAAFFFF);
    Border fBorder = BorderFactory.createEtchedBorder(EtchedBorder.
        RAISED);
    for (int i = 0; i < maxFloor; i++) {
      FloorButton fButton = new FloorButton(i);
      fButton.setBackground(fColor);
      fButton.setBorder(fBorder);
      fButton.setIcon(blankIcon);
      fButton.setIconTextGap(5);
      fButton.setHorizontalTextPosition(JButton.TRAILING);
      fButton.setActionCommand("f"+i);
      fButton.addActionListener(mapActionListener);
      floorList.add(fButton);
    }
  }

  @SuppressWarnings("serial")
  private class FloorButton extends JButton implements Comparable<
      FloorButton> {
    int floor;

    public FloorButton(int floor) {
      super(""+(floor+1));
```

```
      this.floor = floor;
    }

    @Override
    public int compareTo(FloorButton arg0) {
      return Integer.valueOf(floor).compareTo(arg0.floor);
    }
  }
}
```

References

Addressable fire alarm panels. (2011). Online. Retrieved from http://www.simplexgrinnell.com/Solutions/FireDetectionAndAlarm/Products/ControlPanels/AddressableFireAlarmPanels/Pages/default.aspx

Ahlstrom, V., & Kudrick, B. (2007, May). *Human factors criteria for displays: A human factors design standard update of chapter 5* (Tech. Rep. No. DOT/FAA/TC-07/11). Federal Aviation Administration.

BACPAC BACnet products. (2010). Online. Retrieved from http://www.simplexgrinnell.com/SOLUTIONS/FIREDETECTIONANDALARM/PRODUCTS/SYSTEMACCESSORIES/COMMUNICATIONSDEVICES/Pages/BACpacBACnetProducts.aspx

Berry, D., Usmani, A., Torero, J. L., Tate, A., McLaughlin, S., Potter, S., et al. (2005, September). Firegrid: Integrated emergency response and fire safety engineering for the future built environment. In *Uk e-science programme all hands meeting (ahm-2005).* Nottinham, UK.

Bisantz, A. M., Marsiglio, S. S., & Munch, J. (2005). Displaying uncertainty: Investigating the effects of display format and specificity. *Human Factors*, *47*(4), 777-796.

Breton, R., Paradis, S., & Roy, J. (2002). Command decision support interface (CODSI) for human factors and display concept validation. *Information Fusion, 2002. Proceedings of the Fifth International Conference on*, *2*, 1284 - 1291.

Bryner, N. (2008, February). Fire fighter locator. Retrieved from http://www.fire.gov/locator/index.htm

Bushby, S. T. (1996). *Testing performance and interoperability of BACnet building automation products.* Retrieved from http://www.fire.nist.gov/bfrlpubs/build96/art043.html

Compass, I. (2010). *Features for fire.* Retrieved from http://www.ironcompass.com/index.php?option=com_content&view=article&id=33&Itemid=50

Corporation, K. (2008). *DMP703 universal alarm monitor.* Retrieved from http://www.keltroncorp.com/pdfs/DMP703_Data_Sheet.pdf

Corporation, K. (2009). *Keltron RF774F wireless transceiver.* Retrieved from http://www.keltroncorp.com/pdfs/Keltron_RF774F_Data_Sheet.pdf

Culbert, K. (2010). *Ibisworld industry report 56162: Security, burglar fire alarm services in the us* (Tech. Rep.).

Dai, J., Wang, S., & Yang, X. (1994). Computerized support systems for emergency decision making. *Annals of Operations Research*, *51*, 313-325.

Davis, W. D., Holmberg, D., Reneke, P., Brassell, L., & Vettori, R. (2007, August). *Demonstration of real-time tactical decision aid displays* (Tech. Rep. No. NISTIR 7437). NIST.

Design criteria/facilities standard manual (Tech. Rep.). (2005). University of Maryland Facilities Management. Retrieved from http://www.facilities.umd.edu/DCFS2005/

Drury, J., Klein, G., Pfaff, M., & More, L. (2009, May). Dynamic decision sup-

port for emergency. In *Technologies for homeland security, 2009. hst '09. ieee conference on.* Boston, MA.

FDM. (2010). *Fdm rms - modules - properties.* Retrieved from http://www.fdmsoft.com/FDM_RMS_Properties.html

Guerlain, S., Brown, D., & Mastrangelo, C. (2000, October). Intelligent decision support systems. In *Systems, man, and cybernetics, 2000 ieee international conference on.* Nashville, TN, USA.

Health, S., & Safety. (2010). *Pak-tracker locator.* Retrieved from http://www.scotthealthsafety.com/americas/en/products/accountability/Locator/paktracker.aspx

Holmberg, D. G., Treado, S. J., & Reed, K. A. (2006, January). *Building tactical information system for public safety officials: Intelligent building response (ibr)* (Tech. Rep. No. NISTIR 7314). NIST.

Honeywell International, I. (2004). *Noti-fire-net.* Retrieved from http://www.notifier.com/products/datasheets/DN_6791.pdf

Honeywell International, I. (2009). *BACnet-GW-3: BACnet gateway.* Retrieved from http://www.notifier.com/products/datasheets/DN_6877.pdf

Honeywell International, I. (2010a). *Fire alarm control panels.* Retrieved from http://www.notifier.com/products/controlpanels.htm

Honeywell International, I. (2010b). *Fire alarm peripherals.* Retrieved from http://www.notifier.com/products/peripherals.htm

Honeywell International, I. (2010c). *NFS2-3030.* Retrieved from http://www.notifier.com/products/datasheets/DN_7070.pdf

Honeywell International, I. (2010d). *NFS2-640.* Retrieved from http://www.notifier.com/products/datasheets/DN_7111.pdf

Honeywell International, I. (2010e). *ONYX FirstVision: Interactive firefighters' display.* Retrieved from http://www.notifier.com/products/datasheets/DN_7051.pdf

Honeywell International, I. (2010f). *ONYXWorks: Integrated facilities monitoring network.* Retrieved from http://www.notifier.com/products/datasheets/DN_7048.pdf

John, M. S., Smallman, H. S., Manes, D. I., Feher, B. A., & Morrison, J. G. (2005). Heuristic automation for decluttering tactical displays. *Human Factors*, *47*(3), 509-525.

Jones, W. W., Holmberg, D. G., Davis, W. D., Evans, D. D., Busby, S. T., & Reed, K. A. (2005, January). *Workshop to define information needed by emergency responders during building emergencies* (Tech. Rep. No. NISTIR 7193). NIST.

LLC, R. (2010). *Realview -mobile property inspection tools.* Retrieved from http://www.realviewllc.com/fire-departments.aspx

National Fire Protection Association. (2010). *About NFPA: Overview.* Retrieved from http://www.nfpa.org/itemDetail.asp?categoryID=495&itemID=17991&URL=About\%20Us/Overview

National Volunteer Fire Council. (n.d.). *NVFC: NFPA standards.* Retrieved from http://nvfc.org/index.php?id=764

Newman, M. H. (2000). *Bacnet tutorial overview.* Retrieved from http://www.bacnet.org/Tutorial/HMN-Overview/sld001.htm

NFPA 101: Life safety code. (2009).

NFPA 1620: Standard for pre-incident planning. (2010).

NFPA 170: Standard for fire safety and emergency symbols. (2009).

NFPA 72: National fire alarm and signaling code. (2010).

Norman, J. (2005). *Fire officer's handbook of tactics* (Third ed.). PennWell.

Parasuraman, R., & Wickens, C. D. (2008). Humans: Still vital after all these years of automation. *Human Factors*, *50*(3), 511-520.

Potter, S., & Wickler, G. (2008, May). Model-based query systems for emergency response. In *Proceedings of the 5th international iscram conference.* Washington, DC, USA.

Priegnitz, D., Frashier, D., Marci, T., O'Bannon, T., Smith, S., Steadham, R., et al. (1997, July). Human factors method in the design of the graphical user interface for the open systems radar product generation (orpg) component of the wsr-88d. In *Aerospace and electronics conference, 1997. naecon 1997., proceedings of the ieee 1997 national.* Dayton, OH, USA.

Puit, G. (2000). MGM Grand fire: The deadliest day. *Las Vegas Review-Journal*. Retrieved from http://www.reviewjournal.com/lvrj_home/2000/Nov-19-Sun-2000/news/

Roberts, M. (2010, August). Harris debuts firefighter locator. *Fire Chief*. Retrieved from http://firechief.com/technology/ar/harris-gr100-firefighter-location-20100826/

Robinson, J. N. (2009). Solving the system: Integrated fire alarm monitoring. *College Planning and Management*. Retrieved from http://www.keltroncorp.com/pdfs/University_of_Maryland_Case_Study.pdf

Routley, J. (1995). *Four firefighters die in seattle warehouse fire* (Tech. Rep.). U.S. Fire Administration. Retrieved from http://www.usfa.dhs.gov/downloads/pdf/publications/tr-077.pdf

Routley, J. (1998). *Interstate building bank fire* (Tech. Rep.). U.S. Fire Administration. Retrieved from http://www.usfa.dhs.gov/downloads/pdf/publications/tr-022.pdf

Rovira, E., McGarry, K., & Parasuraman, R. (2007). Effects of imperfect automation on decision making in a simulated command and control task. *Human Factors*, *49*(1), 76-87.

SafeLINC fire panel internet interface. (2010). Online. Retrieved from http://xtra.simplexnet.com/a_e/FA/4100-0028.pdf

Sarter, N. B., & Schroeder, B. (2001). Supporting decision making and action selection under time pressure and uncertainty: The case of in-flight icing. *Human Factors*, *43*(4), 573-583.

Siemens Industry, Inc. (2002). *Thermal fire detectors: Explosion proof models.* Retrieved from http://www.us.sbt.siemens.com/FIS/productdoc/catalogs/6128.pdf

Siemens Industry, Inc. (2003). *VESDA LaserPlus.* Retrieved from http://www.us.sbt.siemens.com/FIS/productdoc/catalogs/1173.pdf

Siemens Industry, Inc. (2005). *MXLV: Multiplex emergency voice alarm/communication system.* Retrieved from http://www.buildingtechnologies.siemens.com/bt/us/SiteCollectionDocuments/sbt_internet_us/2691_726.pdf

Siemens Industry, Inc. (2006). *MXL advanced protection system.* Retrieved from http://www.buildingtechnologies.siemens.com/bt/us/SiteCollectionDocuments/sbt_internet_us/2702_737.pdf

Siemens Industry, Inc. (2010a). *FireFinder XLS fire alarm control panel.* Retrieved from http://www.us.sbt.siemens.com/FIS/productdoc/catalogs/6300.pdf

Siemens Industry, Inc. (2010b). *Intelligent FirePrint detector: Model FP-11.* Retrieved from http://www.us.sbt.siemens.com/FIS/productdoc/catalogs/6175.pdf

Simplex 4100U fire alarm control panel. (2010). Online. Retrieved from http://www.simplexgrinnell.com/Solutions/FireDetectionAndAlarm/Products/ControlPanels/AddressableFireAlarmPanels/Pages/4100UFireAlarmControlPanel.aspx

SimplexGrinnell initiating devices. (2011). Online. Retrieved from http://www.simplexgrinnell.com/Solutions/FireDetectionAndAlarm/Products/InitiatingDevices/Pages/default.aspx

SimplexGrinnell specialized detection devices. (2011). Online. Retrieved from http://www.simplexgrinnell.com/Solutions/FireDetectionAndAlarm/Products/InitiatingDevices/SpecializedDetectionDevices/Pages/default.aspx

Smokeview (version 5) - a tool for visualizing fire dynamics simulation data volume i: Users guide [Computer software manual]. (n.d.).

Talcott, C. P., Bennett, K. B., Martinez, S. G., Shattuck, L. G., & Stansifer, C. (2007). Perception-action icons: An interface design strategy for intermediate domains. *Human Factors*, *49*(1), 120-135.

Thiel, A. (1999). *Special report: Improving firefighter communications* (Tech. Rep.). U.S. Fire Administration. Retrieved from http://www.usfa.dhs.gov/downloads/pdf/publications/tr-099.pdf

Thompson, J. (2009). *Physiological monitoring system checks firefighter vital signs.* Retrieved from http://www.firerehab.com/fire-products/fire-rehab/articles/587030-Physiological-monitoring-system-checks-firefighter-vital-signs/

Treado, S., Vinh, A., Holmberg, D., & Galler, M. (2007, July). Building information for emergency responders. In *Systemics, cybernetics and informatics, 11th world multi-conference (wmsci 2007). proceedings. volume 3.* Orlando, FL.

TrueSite workstation with multi-client capability. (2010). Online. Retrieved from http://xtra.simplexnet.com/a_e/FA/4190-0016.pdf

VESDA VLC-600 TrueAlarm Laser COMPACT. (2010). Online. Retrieved from http://www.simplexgrinnell.com/SOLUTIONS/FIREDETECTIONANDALARM/PRODUCTS/INITIATINGDEVICES/SPECIALIZEDDETECTIONDEVICES/Pages/VESDA.aspx

Vitals vest – physiologists create undergarment to measure vital signs of firefighters. (2008, February). *Science Daily*. Retrieved from http://www.sciencedaily.com/videos/2008/0212-vitals_vest.htm

Wickens, C., Lee, J., & Liu, Y. (1997). *An introduction to human factors engineering*

(2nd ed.). Prentice Hall.

Ye, X., Wang, Y., Li, H., & Dai, Z. (2008, December). An emergency decision support system based on the general decision process. In *Web intelligence and intelligent agent technology, 2008. WI-IAT '08. IEEE/WIC/ACM international conference on.* Sydney, NSW.

Zone, C. (2010a). *Fire zone basic features.* Retrieved from http://www.cadzone.com/index.php?option=com_content&view=article&id=119\%3Afire-zone-basic-features&catid=46&Itemid=143

Zone, C. (2010b). *First look pro.* Retrieved from http://www.cadzone.com/index.php?option=com_content&view=article&id=75&Itemid=158