Modeling Human Workload in Unmanned Aerial Systems

TJ Gledhill

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Michael A. Goodrich, Chair
Kevin Seppi
Eric M. Mercer

Department of Computer Science

Brigham Young University

April 2014

Copyright © 2014 TJ Gledhill

All Rights Reserved

ABSTRACT

Modeling Human Workload in Unmanned Aerial Systems

TJ Gledhill
Department of Computer Science, BYU
Master of Science

Unmanned aerial systems (UASs) often require multiple human operators fulfilling diverse roles for safe correct operation [18, 22, 35]. Although some dispute the utility of minimizing the number of humans needed to administer a UAS [36], minimization remains a long-standing objective for many designers. Reliably designing the human interaction, autonomy, and decision making aspects of these systems requires the use of modeling. We propose a conceptual model which models human machine interaction systems as a group of actors connected by a network of communication channels. We also propose a workload taxonomy derived from a review of the relevant literature which we then apply to the conceptual model. We present a simulation framework implemented in Java, with an optional XML model parser, which can be analyzed using the Java Pathfinder (JPF) model checker. The simulator produces a workload profile over time for each human actor in the system. We conducted case studies by modeling two different UAS. Wilderness search and rescue using a UAV (WiSAR) and UAS integration into the national air space (NAS). The results of these case studies are consistent with known workload events and the simple workload metric presented by Wickens [43].

Keywords: human workload, unmanned aerial system, uas, national air space, unmanned aerial vehicle, modeling human machine interaction

ACKNOWLEDGMENTS

The authors would like to thank the NSF IUCRC Center for Unmanned Aerial Systems, and the participating industries and labs, for funding the work. The authors would like to thank Neha Rungta of NASA Ames Intelligent Systems Division for her help with JPF and Brahms. The authors would also like to thank the NSF IUCRC Center for Unmanned Aerial Systems, and the participating industries and labs, for funding the work. Further thanks go to Jared Moore and Robert Ivie for their help coding the model and editing this paper.

Table of Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction and Overview

Most existing Unmanned Aerial Systems (UASs) require two or more human operators[22, 35]. Standard UAS practice is to have one human to control the aerial vehicle and another to control the camera or other payloads. In addition to this a third human is often responsible for overseeing task completion and interfacing with the command structure. Although some argue persuasively that this is a desirable organization [36], there is considerable interest in reducing the required number of humans and reducing human workload using improved autonomy and enhanced user interfaces [18, 26, 32].

Our initial proposal was to move directly into software development. Given our prior experience with UAS-enabled Wilderness Search and Rescue (WiSAR) [26] we proposed to construct a UAS for that domain. During the requirement gathering and design steps of this project it became clear just how complex the system was. While we had prototypes of almost all the functionality we had no way of measuring if the system itself would meet the requirements. On top of that the limited time and resources meant that we would only get one shot at creating this system. Because of this we decided to take a more conservative approach. Instead of blindly pressing forward with the software development we decided that it would be more beneficial to validate our designs through modeling.

System modeling is not a new approach. There are many different modeling languages each of which is designed to perform specific types of validation [3]. While it was possible to extend an existing modeling language to support our goals much like Bolton and Bass have done with EOFM []. We chose to create our own modeling language from scratch. We

chose this direction for a few reasons not the least of which being our lack of experience with other modeling languages. Instead of learning a new language we desired to use a language and model checker we were already familiar with, Java and Java Pathfinder. Also, a common denominator among system modeling languages is the focus on tasks. This is ideal for modeling an existing system, however, for new systems these detailed tasks are vague or undefined [**?**]. We needed to be able to model and validate these tasks without needing to understand their details. We also desired to measure human workload as a consequence of the system design, something which is relatively new to human machine interface validation [3].

Our modeling language allows models to be implemented in Java simply by implementing a core set of Java interfaces which comply with our conceptual model. These models are then processed in a simulation framework which is run inside JPF for the model checking and metric gathering. The conceptual model underneath the modeling framework consists of Directed Role Graphs (DiRGs) and Directed Team Graphs (DiTGs) which focus on the key Actors within the system and the communication channels which they use to perform their work. The model itself is defined as a state machine. This common approach to modeling allows us a flexible approach to abstraction while still allowing us to gather workload data and lends itself well to model checking.

We chose to base our workload measurements off of multiple resource theory [**?**] with ties to queing theory [**?**] and operator fan-out theory []. By relating these theories to the different operational components of the model we can obtain a quantitative measure of an Actors workload for each time-step in the system.

We performed two case studies, one for WiSAR and another representing the introduction of a UAS into the National Air Space (NAS). We chose WiSAR because of the host of modeling information available to us [9]. We chose to model a UAS operating within the NAS because of the current interest in the subject [**?**] and the decided lack of modeling information available to us which required us to use high levels of abstraction. The results of the case studies show that the modeling language we developed is capable of accurately

modeling UASs. They also demonstrate the ability to model systems using varying degrees of abstraction. While the workload metrics are still unverified initial results appear very promising and trend well with known high workload areas.

1.1 Overview and Papers

Chapters 2 and 3 of this thesis consist of two published papers. Chapter 2 introduces the DiRG and simulation framework. Chapter 3 extends chapter 2 by adding the DiTG and workload metrics.

Chapter 2 presents our core conceptual model along with our implementation of this model in Java. It also presents how our simulation framework is able to validate the model using JPF.

Chapter 3 presents an extension of our conceptual model in the form of the DiTG. It also presents a formal taxonomy of workload metrics and how that taxonomy applies to our conceptual model. Lastly it reports the results of adding the workload metrics into the simulation framework.

Chapter 4 includes the changes made to the conceptual model and simulation framework as a result of the work performed to obtain the results presented in chapter 3. It also presents an XML modeling format which is meant to simplify the modeling process.

Chapter 5 presents a case study which involves modeling the introduction of a UAS into the NAS. This case study uses the new XML modeling format to produce several versions of the Java model. A detailed analysis and comparison of the results of these models is also given.

Chapter 6 contains the conclusions we have made about this research along with the avenues of future work which we find most interesting.

We also include the initial WiSAR proposal along with actual java code and raw metric gathering data to help the reader fully understand the work we have done.

Chapter 2

Modeling UASs for Role Fusion and Human Machine Interface Optimization

TJ Gledhill, Eric Mercer and Michael A. Goodrich. Modeling UASs for Role Fusion and Human Machine Interface Optimization. In Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, Manchester, England, 2013.

Abstract

Currently, a single Unmanned Aerial System (UAS) requires several humans managing different aspects of the problem. Human roles often include vehicle operators, payload experts, and mission managers [18, 22, 35]. As a step toward reducing the number of humans required, it is desirable to reduce operator workload through effective distributed control, augmented autonomy, and intelligent user interfaces. Reliably doing this requires various roles in the system to be modeled. These roles naturally include the roles of the humans, but they also include roles delegated to autonomy and software decision-making algorithms, meaning the GUI and the unmanned aerial vehicle. This paper presents a conceptual model which models the roles of complex systems as a collection of actors, running in parallel. Results from applying this model to the UAS-enabled Wilderness Search and Rescue (WiSAR) domain indicate (a) it is possible to model the entire WiSAR system at varying degrees of abstraction (b) that building and evaluating the model provides insight into the best practices of WiSAR teams and (c) a way to model human machine interactions that works directly with the Java Pathfinder model checker to detect errors.

2.1 Introduction

Most existing Unmanned Aerial Systems (UASs) require two or more human operators[22, 35]. Standard UAS practice is to have one human to control the aerial vehicle and another to control the camera or other payloads. In addition to this a third human is often responsible for overseeing task completion and interfacing with the command structure. Although some argue persuasively that this is a desirable organization [36], there is considerable interest in reducing the required number of humans and reducing human workload using improved autonomy and enhanced user interfaces [18, 26, 32].

The broad research context driving this paper takes a multistep approach: (a) model the roles for a specific UAS and a well-defined set of tasks, (b) delimit assumptions and abstractions used in the model, (c) verify properties of the model, (d) use the model to explore ways of combining roles in such a way that operator workload and the number of humans is minimized, and (e) design vehicle autonomy and user interface support to allow a real UAS team to operate more efficiently. The focus of this paper is on the important lessons learned in the first two steps.

The modeling used in this paper could be applied to a number of tasks, but we focus on Wilderness Search and Rescue for two reasons. First, the authors have done prior work on UAS-enabled Wilderness Search and Rescue (WiSAR) [26]; and second, there is a host of modeling information about how WiSAR is currently performed [9]. The UAS-enabled WiSAR systems produced by this research requires three humans, two GUIs, and a single UAV.

To gain insight into WiSAR we have chosen to model the system as a group of directed role graphs (DiRGs). Each human, GUI, and UAV represent a DiRG, allowing evaluation of potential conflicts between and opportunities for unification of the various roles. These DiRGs are referred to as *Actors* in the models below.

Modeling is essentially a process of abstraction, choosing which elements of a system are essential and which are not [12]. Since one of the goals of this paper is to use model-

checking to evaluate models, we choose a model class that is simple enough that it allows us to clearly delineate between what is modeled and what is not. Thus, we use DiRGs, which can be expressed as Mealy state machines, to model different WiSAR roles. These models explicitly encode key aspects of the various actors, and collectively form a group of Mealy machines that run in parallel. The model is encoded in Java using a custom set of interfaces designed to simulate a discrete time environment, facilitate input/output between roles, and provide non-deterministic event handling.

Model checking is performed using Java Pathfinder (JPF). This is convenient because JPF runs the model checking on the compiled Java code generated by the modeling exercise.

The results of this modeling exercise indicate (a) it is possible to model the entire WiSAR system using varying degrees of abstraction and (b) that building and evaluating the model provides insight into the best practices of WiSAR teams and (c) a way to model human machine interactions that works directly with the Java Pathfinder model checker to detect errors.

2.2 Related Work

NASA Ames Research Center (NASA ARC) is using Brahms, a complex and robust language, to model interactions between operators and their aerial equipment [6]. To study the Uberlingen collision, an in air collision of two commercial passenger planes, Rungta and her colleagues produced a model entirely in the Brahms language. This model correctly predicts the collision and also reveals some of the difficulties intrinsic to this type of system.

One critical aspect of NASA ARCs work carries over into our own: variable task duration [5]. Task duration directly influences whether the situation ends safely, barely avoids a crash, or crashes. The biggest advantage Brahms has over Java comes from its strict grammar. However, Brahms must be translated into Java before using JPF, a step we avoid by implementing our model in Java directly.

Bolton and Bass used the Enhanced Operator Function Model (EOFM) language to create a model consisting of the Air Traffic Controller, the pilot flying the plane, and the pilot monitoring the equipment, as well as the interfaces they used [1]. As they increased the number of allowable miscommunications, their system had an exponential increase of errors. EOFM facilitates the division of goals into multiple levels of activities. These activities can then be broken into atomic actions [2]. The main difference between this model and our own is that EOFM is expressed in XML while ours is expressed in Java. This allows us to perform model checking directly using JPF.

Wilderness Search and Rescue is primarily concerned with finding people who have become lost in rugged terrain. Research has shown that UASs could potentially be used to facilitate this work. Goodrich et al. tested the effectiveness of these types of operations [22]. A key outcome of these field tests is the speculation that effectiveness could be enhanced if the roles of the UAV operator and video operator were combined.

In prior work, a goal-directed task analysis, a work domain analysis, and a control task analysis were performed. [9]. These analyses modeled WiSAR as a collection of goals, work domains, and tasks. While these studies proved valuable for understanding the WiSAR processes they were less helpful in suggesting improvements to the WiSAR processes. Indeed, the limitations of such tools for informing the design of technology to support existing processes has lead to new methods for performing such analyses [28]; the work in this paper complements such work, using model-checking to perform analyses on problems that do not lend themselves to answers using other approaches.

2.3 WiSAR UAS Domain

Wilderness search and rescue often occurs in remote, varying, and dangerous terrains. According to [39], there are four core elements of a WiSAR operation: *Locate*, *Reach*, *Stabilize*, and *Evacuate*. The WiSAR UAS operates within this first element so it is the focus of this paper.

During the *Locate* element, the incident commander (IC) develops a strategy to obtain information. This strategy makes use of the available tactics to obtain this information. The WiSAR UAS is one of the tactics that the IC may choose to use. A WiSAR UAS technical search team consists of three humans: Mission Manager, Vehicle Operator, and Video Operator. These constitute the three human roles in the team. Supporting these human roles are two intelligent user interfaces, the Vehicle Operator GUI and the Video Operator GUI; these constitute two other roles that must be modeled. The final role is the aerial vehicle itself, which is equipped with sensors and controllers that enable it to make decisions. Since the WiSAR UAS technical search team must coordinate its efforts with other members of the search team via the IC, we embed the five UAS roles within a Parent Search model. The parent search model represents the entire command structure for the search and rescue operation.

In the next section, we model the WiSAR roles and the interactions between these roles. Naturally, these roles will need to take input from the environment, so we present a simple model of the environment that emphasizes the key environmental elements, probabilistic events and varying task durations. Note that this model of the environment exists at a higher level of abstraction than what is typically considered an environment model in literature; typical models tend to focus on environmental realism, encoding things like terrain, wind, etc, but our model emphasizes events that affect the behavior of the WiSAR roles.

2.4 Conceptual Model

We have chosen to conceptualize the WiSAR UAS as a group of DiRGs. A DiRG represents a sequence of tasks for a single role. By conceptualizing WiSAR as a collection of DiRGs running in parallel we hope to gain more insight into the WiSAR processes with a goal of improving these processes. In prior work, a goal-directed task analysis, a work domain analysis, and a control task analysis were performed. [9]. These analyses modeled WiSAR as a collection of goals, work domains, and tasks. While these studies proved valuable for

understanding the WiSAR processes they were less helpful in suggesting improvements to the WiSAR processes. Indeed, the limitations of such tools for informing the design of technology to support existing processes occurs because they discover conditions which may result in problems rather than discovering system problems themselves. Performing system-level task modeling, such as we are, is capable of discovering such problems [3].

While we are using this technique to find such problems we expand on it in several ways. First, this technique is most commonly used for analyzing a single human using an interface. Our models involve a team of humans simultaneously using multiple interfaces which naturally increases the state space of the model, decreasing scalability. Second, we are using this technique to analyze the workload of the system. Our goal is the combining of human roles and interfaces. We hope to gain insight into decreasing the system workload, and possibly combining roles, by establishing metrics associated with the task model and model simulation. These metrics can then be used to determine if changes to the model represent a decrease in operator workload.

As is common when modeling human-automation interaction we have decided to model the DiRGs usings Mealy state machines [3] This allows us to abstract the different WiSAR roles into individual state machines that we call Actors. Actors do not correspond to a single aspect of the WiSAR domain, anything can be an Actor, thus providing the freedom to flexibly model as many aspects of the domain as necessary and at various levels of resolution. This freedom is important and represents our primary method of reducing the state space to manage scalability. Actors transition between states by receiving inputs generated by other Actors and Events. Events are also modeled as Mealy machines whose transitions are triggered by a combination of simulation and Actor inputs. Because Actors and Events may receive input from other Actors and Events the combination of their transition matrices define the wiring of the different DiRGs. A single wire is when an output from one Actor is an input on another Actor. This implies that the set of all inputs is the same as the set of all outputs, however, in practice we do not treat these sets as the same since unhandled input

represents transitions returning to the current state. We ignore these looping transitions except when their behavior tells us something interesting.

Formally, the models are the following mathematical structures:

$$Actor = (S, s_0, \Sigma_A \cup \Sigma, \Lambda_A, T) \tag{2.1}$$

$$Event = (S, S_0, \Sigma_A \cup \Sigma_S, \Lambda_A, T) \tag{2.2}$$

$$T : S \times \Sigma \Rightarrow S \times \Lambda_A \tag{2.3}$$

where S is a set of states, s_0 the start state, Σ_A the set of all Actor inputs, Σ_S the set of all Simulator inputs, Λ_A the set of all Actor outputs, and T a transition matrix which specifies the outputs for any state transition. T may have multiple inputs and multiple outputs.

At this point our conceptual model is implementation agnostic. Indeed, the abstraction allows us to group Actors, break Actors into sub-Actors, and use Actors to validate specific behaviors. The model also allows for complex transitions between Actor states and the ability to enter normally unreachable state spaces using Events. The model is also easily adapted to code which can be verified using model checking tools such as Java Pathfinder (JPF) which we will show in the following sections.

2.5 Simulating the WiSAR UAS

Real WiSAR environments and UAV dynamics are complex so full models of the environment and UAV can also become extremely complex. However, many of the complexities are not relevant to the decisions made by the various WiSAR actors. Consequently, we propose a model that "abstracts away" many unessential details and encodes key aspects of the

environment. In order to simulate critical aspects of the WiSAR UAS model it is necessary to represent communication between Actors, concurrency, and task duration, concepts which are outside the scope of a standard state machine. To do this we constructed a basic simulation framework. The simulation framework is encapsulated into a single Java class, called *Simulator*. This section discusses the key components of this simulation framework.

2.5.1 Core Simulator Objects

The Simulator is made up of the following objects: Team, Actors, Events, States, Transitions, and Unique Data Objects (UDO). We organize these objects in the following way. A Team is a wrapper class representing the entire model which contains a collection of Actors, Events, and shared UDOs. Each Actor contains a set of private States. Each of these States contains a set of Transitions. Each Transition contains a set of input UDOs, a set of output UDOs, the outgoing Actor State, and a reference to the Actors current State. This structure is convenient because it naturally encapsulates the different aspects of a Mealy machine.

Each UDO represents a unique piece of data, input or output, and is flagged as active or inactive. A UDO is temporarily set to active after it is sent as output. Each Transition can easily determine if it is possible by checking to see that all of its input UDOs are active. Each State can then return a list of possible transitions. An Actor evaluates the list of possible Transitions to determine how it should transition. If it is empty then the Actor does not transition, otherwise the Actor chooses a single Transition to occur. An Actor chooses a Transition at the Simulators request. The Simulator tracks when this Transition should occur, at which point the Transition will set each output UDO to active and change the Actors current state to the Transitions outgoing State. Because the UDOs must exist before Actors can be initialized we have created the Team class. The Team class wraps all of the Actors, Events, and UDOs into single entity. This class first initializes each UDO, afterwards each Actor and Event is initialized with a list of input and output UDO references which it will use in its transitions. This structure offers a few benefits. Code wise it allows us to simulate

the transfer of data without actually transferring data which drastically simplifies the code. It also forces us to explicitly define our model wiring in two places, first at the Team level and second at the Actor Transition level as mentioned above. Although the Transition set of inputs is not limited to the set of global UDOs we can still compare the set of global inputs and outputs an Actor receives with the set of inputs and outputs defined by its transitions. Through this we can validate that the set of Actor transitions is complete, as defined in our Team, which is an important step in validating the model. A simple example of the initialization of the Team and its components is shown in the following example:

```
Team {
    UDO A1Output = new UDO()
    UDO E1Output = new UDO()

    UDO Inputs[] = [E1Output]
    UDO Outputs[] = [A1Output]
    Actor A1 = new Actor(Inputs, Outputs)

    UDO Inputs[] = [A1Output]
    Actor A2 = new Actor(Inputs, [])

    UDO Outputs[] = [E1Output]
    Event E1 = new Event([], Outputs)
}

A1(Inputs, Outputs) {
    State S1 = new State()
    State S2 = new State()
```

```
    S1.addTransition(this,
        Inputs.E1_Output,
        Outputs.A1Output,
        S2)
}

A2(Inputs, Outputs) {
    State S1 = new State()
    State S2 = new State()

    S1.addTransition(this,
         Inputs.A1Output,
         null,
         S2)
}

E1(Inputs, Outputs) {
    State S1 = new State()
    State S2 = new State()

    S1.addTransition(this,
        null,
        Outputs.E1Output,
        S2)
}
```

While this is only a basic example it clearly shows how the models have been implemented as code. The example also illustrates the Event class. Events differ from Actors in

that Event transitions require an express command from the Simulator before processing. This allows the Simulator to non-deterministically trigger events which, when run in JPF, can be setup to process events at different intervals or when the system changes state. From this we can determine the effects events have on the system in a very robust manner.

2.5.2 Communication Between Actors

To simulate a team of Actors working together it is necessary to establish a communication medium within the model and simulation which can represent the different forms of communication. In the model this communication medium is represented as inputs, outputs, and transitions.

Our initial attempts to simulate this communication resulted in a PostOffice class attached to the Simulator. Actors sent data along with the name of the recipient to the PostOffice. Actors could then retrieve their input from the PostOffice, much the way PO boxes work. Actors also had the ability to make certain output observable through the PostOffice. This meant that an Actor could place data into the PostOffice for other Actors to observe, a public PO box. When we added sub-Actors to the model code it became necessary to allow Actors and sub-Actors to share both private and public PO boxes. It was also necessary to store current and future output separately to achieve concurrency which we explain in the next section. Although this achieved the desired goal the results were less than satisfactory. In addition to the added complexity the design used implicit input and output connections making it much harder to validate that the code represented the desired model.

Our next iteration of the Simulator simplified this communication medium with the use of the previously defined UDOs. By initializing these UDOs and passing them as references to the Actors and Events we greatly simplified inter-Actor communication. This new design also requires explicit declarations for each UDO connection allowing us to validate the model code with the model and again with the transition matrixes. The UDO is also capable of representing both direct and observable communications which naturally allow sub-Actors to

link inputs with parents. Indeed, Actor relationships are now irrelevant in regard to sharing input and output since Actors only depend on the status of the UDOs. In the case where it does matter which Actor generated the output then a new UDO can be created representing that relationship thus preserving Actor independence through explicit connections.

The Team initialization example above demonstrates the use of UDOs. The example defines two UDOs, A1Output and E1Output. Once defined these UDOs are passed by reference to specific Actors and Events as inputs, explicitly defining the connectivity implied by the UDOs. The Actors use the UDO references for constructing their transition matrices which results in the connecting of A2 to A1 and A1 to E1. If we desired to change our model and allow A2 to transition on E1Output the UDO would be added to the A2 input and A2 would declare a transition for that input resulting in the connectivity of A2 to E1.

2.5.3 Simulating Time

To simulate task duration the simulator uses the delta time algorithm. Each Transition has a specified duration range defined by the Actor or Event. The simulator has five different duration settings for choosing a value within a range: MIN, the minimum; MAX, the maximum; MIN_OR_MAX, a random choice between one of these settings; $MIN\_MEAN\_OR\_MAX$, another random choice.. When an Actor begins a Transition the Simulator chooses a duration, the value of that duration is then converted to the Simulator delta time. Basically if a transition is to finish in 30 time steps but another Transition finishes at 25 time steps then the first Transition is placed after the second Transition and is given a count of 5 which means it happens 5 time steps after the prior Transition. Thus as the simulation progresses time remains relative.

Slightly different from the use of transition durations is the notion of simulating Events. The time range over which an Event may occur is often much larger than the ranges defined by tasks, also, Events are only possible in specific state spaces thus preventing us from predicting the available time range of the Event. This prevents us from simulating

Event timing in the same way as task durations because such large time ranges cannot be accurately represented with with only 2 to 3 choices and we cannot select a min, max, or mean if the range is unknown. We solve this problem in two ways. One method is to trigger the Event at regular time intervals while the Event is possible. Depending on the interval size this can cause a dramatic increase in state space. While this increases the possibility of exploring the possible effects an Event can generate on the system it offers no guarantees. Another method for triggering Events is to trigger the Event on each state change within the system while the Event is possible. This guarantees that the Event will be explored in each state space that is presented during the simulation, however, this is also likely dramatically increase the state space and is much more difficult to implement. Since triggering an Event represents a transition within the Event it is included in the delta time algorithm used for progressing simulation time.

2.5.4 The Simulation Loop

It is now possible to describe the actual simulation. After initialization we enter the simulation phase. This phase is used to transition the Actors and trigger Events. One challenge with transitioning the Actors is the need for concurrency. The simulation must allow multiple Actors to transition without interfering with one another. Previous versions of our Simulator placed each Actor within its own thread. We found that threads complicate the conversion into JPF so instead we chose to use transactions. In each transaction we allow each Actor to make the changes required by its transition. These transitions only modify a temporary value on the UDOs. After all transitions are completed we finish the transaction by moving each temporary UDO value into the actual value. The entire simulation phase can be described thus:

```
Begin Transaction:
    Foreach Actor
        if ( Transition duration reached )
```

```
            Process Transition

    End Foreach
End Transaction

Process Transaction

Foreach Actor
    Transition = Get Next Transition
    If Transition is not null
        Convert Transition duration to delta time.
        Update necessary Actor delta times.
End Foreach
```

When there are no longer any pending Transitions the loop ends and the simulation is terminated.

2.6 WiSAR UAS Model

This section describes the models produced for the UAS-enabled WiSAR process. We first discuss Actors and Events, followed by a brief discussion of Java asserts and a case study drawn from WiSAR.

This conceptual model uses Mealy state machines. This allows us to abstract the different WiSAR roles into individual state machines that we call Actors. Actor states do not correspond to a single aspect of the WiSAR domain, thus providing the freedom to flexibly model as many aspects of the domain as necessary and at various levels of resolution. Actors transition between states are triggered by inputs generated by Events and by other Actors. Events are also modeled as Mealy machines whose transitions are triggered by a combination of simulator and Actor inputs. A benefit of this state machine-based conceptual model is the

ability to convert the model into code. The coded model can be verified using model checking tools such as Java Pathfinder (JPF) to gain further insight into the model. In the interest of space we will not describe the Java simulation framework developed for JPF model checking.

2.6.1 Actors

Choosing the core Actors is critical since modeling UAS-enabled WiSAR requires a level of abstraction that gives useful results without adding unnecessary complexity. After exploring several levels of abstraction, we selected a model that treats as an Actor any core decision-making element of the team, yielding the following Actors: parent search (PS), mission manager (MM), UAV operator (VeOp), video operator (VidOp), UAV operator GUI (VeGUI), video operator GUI (VidGUI), and the UAV. We deliberately chose to not model ground searchers, leaving this to future work.

The models of the human roles use specific states for communication. We describe these states once and then refer to them as a single communication state. Typically before a human communicates he or she receives some signal that the communication is being received. We model this as a POKE state. When communicating, an Actor model of a human enters the POKE state where it waits until it receives an acknowledgement. If the acknowledgement is not received then the communication does not occur. After the acknowledgement the human moves into a transmit (TX) state whose duration is based on the data being transferred. At the end of this transfer the human enters an end (END) state and outputs the transferred data to the receiver.

If the Actor model of the human receives a poke, then it responds with a busy or an acknowledge. If the human acknowledges the poke, then it enters the receive (RX) state. The human will not leave this state until the end communication input is received or until it decides to leave on its own. If one of these communications is interrupted before completion, then we consider that the data was not transferred. To better facilitate communication interruptions we only transfer a single piece of information per communication.

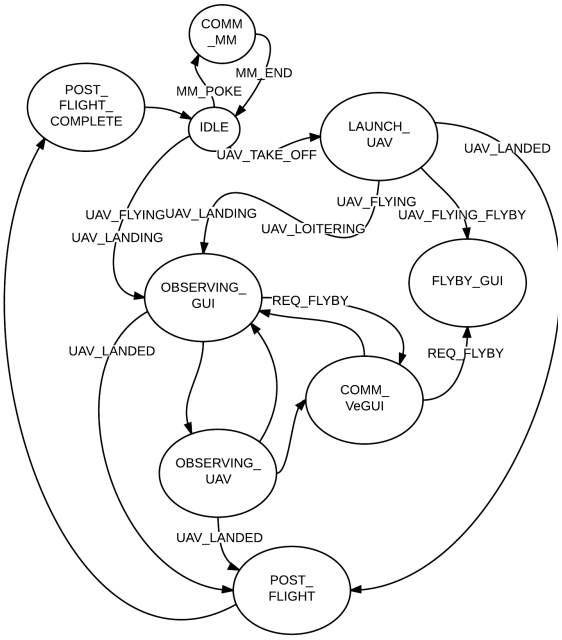

Figure 2.1: UAV Operator DiRG. Excludes transition output.

The next sections are dedicated to describing the Actor state machines with a generalized description of their relative transition matrixes. We omit several of the previously mentioned Actors in the interest of space.

UAV Operator (VeOp)

As illustrated in Figure 2.1, the VeOp Actor has the following states: *IDLE*, *OBSERVING_VeGUI*, *FLYBY_VeGUI*, *OBSERVING_UAV*, *LAUNCH_UAV*, *POST_FLIGHT*, and three communication states: *MM*, *VeGUI*, and *VidOp*. Initially the VeOp is idle. After receiving a new search command from the MM the VeOp constructs a flight plan using the VeGUI. When this is complete the VeOp will then launch the UAV. While in the launch state the VeOp is observing the UAV, when the UAV completes its take off the VeOp moves to observing the VeGUI. While the UAV is airborne the VeOp will continually move between observing the UAV and observing the VeGUI. The VeOp will respond to any problems that are noticed while observing the VeGUI or UAV.

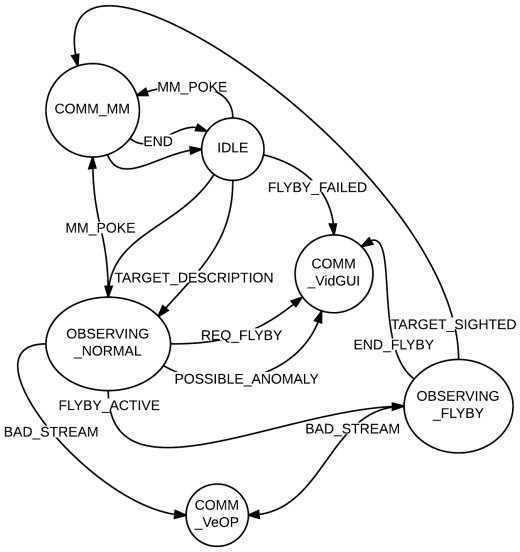

Figure 2.2: Video Operator DiRG. Excludes transition output.

During the flight the VeOp listens for input from the PS and VidOp. If there are flyby requests on the VeGUI, then the VeOp may choose to enter a flyby mode. This implies a high cognitive load on the VeOp while positioning the UAV over the specified anomaly. The VeOp will remain in flyby mode until the VidOp specifies, through the GUI, that the flyby is finished. During an operation the UAV will often land and take off multiple times. The post flight state represents the work necessary to get the UAV ready for flight, such as changing the battery.

Video Operator (VidOp)

As illustrated in Figue 2.2, the VidOp Actor has the following states: *IDLE*, *OBSERVING_VidGUI_NORMAL*, *OBSERVING_VidGUI_FLYBY*, and three communication states: *MM*, *VeOp*, and *VidGUI*. Initially the VidOp is idle. After receiving a target description and the search information the VidOp moves to normal GUI observation. While observing the VidGUI the VidOp watches for anomalies, each time an anomaly is visible the VidOp decides if the anomaly is seen. If it is seen the VidOp decides if it is an unlikely, possible, or likely sighting. This is done with probabilities related to the type of anomaly, true positive or false

positive. If the anomaly is classified as possible then the VidOp makes a validate sighting request for the MM. If the anomaly is a likely sighting then the VidOp requests a flyby from the VeOp.

When the VeOp begins a flyby request the VidGUI signals the VidOp to enter the flyby state. In this state the VidOp watches for the anomaly. Due to the nature of the flyby the VidOp can now make an informed decision about the nature of the anomaly, gaining a much higher probability of being correct. After deciding if it is the target the VidOp signals through the VidGUI that the flyby is finished. If the sighting is confirmed the VidOp reports to the MM, otherwise the VidOp returns to normal GUI observation.

Operator GUI (VeGUI)

The VeGUI Actor has two states: *NORMAL* and *ALARM*. The VeGUI communicates directly with the UAV and VidGUI Actors. The default function of the VeGUI is to observe the UAV. The VeGUI keeps internal variables of all the UAV and VidGUI data that it tracks, all of this data is available through observation of the VeGUI. If it detects an error with the UAV outputs such as low battery, no flight plan, low height above ground, or lost signal the VeGUI will enter the alarm state. This state indicates that there are visible warnings on the screen to alert the VeOp of the problem. The VeGUI listens for VeOp input or changes in the UAV output to signal that the problem has been dealt with before moving into the normal state.

UAV

The UAV Actor has the following states: *READY*, *TAKE_OFF*, *FLYING*, *LOITERING*, *LANDING*, *LANDED* and *CRASHED*. Initially the UAV is in the ready state. Upon command the UAV moves to take off for a specific duration and then to flying or loitering. The flying state is when the UAV is following a flight plan. The loitering state is when the UAV is circling a specific location. The UAV will automatically enter the loitering state after completing its flight plans. While airborne the UAV, upon command, moves to the landing state for a

specific duration before moving into the landed state. Once landed the UAV must be moved into the ready state before it can take off again.

The Actors in this model, thus far, represent a fairly high level of abstraction. Fortunately, the DiRG conceptual framework allows incremental extension of the models by adding lower levels of abstraction. This is accomplished by introducing *sub-Actors* into the model. We illustrate how the Actor/sub-Actor hierarchy can be used by describing two UAV sub-Actors. In these examples the sub-Actors receive all the same input as the parent Actor and all sub-Actor output is sent from the parent Actor.

The first UAV sub-Actor is the UAVBattery. It contains the following states: *INACTIVE*, *ACTIVE*, *LOW*, and *DEAD*. Initially the battery is inactive. The battery is assigned a duration and a low battery threshold. When the UAV receives the take off command the battery enters the active state. The batteries next state is set to low at time $current\_time + battery\_duration - low\_battery\_threshold$. When the battery enters the low state its next state is set to dead at time $current\_time + low\_battery\_threshold$.

A second sub-Actor is the UAVFlightPlan. This represents the flight plan flown by the UAV. The flight plan requires a specific amount of time to complete. The UAVFlightPlan has the following states: *NONE*, *ACTIVE*, *PAUSED*, and *COMPLETE*. Initially the flight plan is set to none. After the operator creates a flight plan using the VeGUI then the flight plan moves to active. During a flight the UAV may loiter, land, or flyby; this causes the flight plan to move to paused. When the UAV begins following the flight plan again it returns to active. After the UAV has flown the flight plan for the specified duration the flight plan enters the complete state.

The results discussed below include four other UAV sub-Actors: UAVHeightAboveGround, UAVSignal, UAVVidFeed, and FlybyAnomaly. Details are omitted in the interest of space.

2.6.2 Events

We used the Event Abstraction for several different elements of the UAS-enabled WiSAR problem. Adding this abstraction allows humans analyzing the system to give a set of inputs to the model and observe the consequences of the inputs.

The following Events were encountered in various UAS-enabled WiSAR field trials and represent a sample of interesting possible operating conditions for the Actors. In the interest of space we list these events while omitting their descriptions. NewSearchAOIEvent, TargetDescriptionEvent, TerminateSearchEvent, LowHAGEvent, LostSignalEvent, TruePositiveAnomalyEvent, FalsePositiveAnomalyEvent, and BadVideoFeedEvent.[1]

2.6.3 Asserts

As a general rule in model-checking, the more complex the model the more that can go wrong. Detecting flaws in the model is extremely valuable because such flaws trigger further evaluation. We present a case study in the next section that illustrates how the evaluation can identify things that need to change in the WiSAR process to avoid serious errors and perhaps failure to find the missing person.

Unfortunately, it can be challenging to differentiate between important flaws and coding bugs. To catch all errors, both flaws and bugs, we use Java Asserts. JPF automatically halts processing when it encounters a false assertion, allowing us to determine if the error is a bug or a flaw.

The model uses asserts in two ways. The first is detection of an undesired state. If an actor enters an undesirable state then an assertion halts the simulation. An example of this is the *UAV_CRASHED* state. The second deals with inputs. Many operations are sequential. They require a specific state and input before the next task can be performed. By looking at an Actors received inputs we are able to tell if an Actor is out of sync with the other Actors. An example of this is the *VeOp_TAKE_OFF* input for the UAV. If the UAV is

[1] HAG = Height Above Ground, AOI = Area of Interest

already airborne and it receives this input we know that the operator is out of sync with the UAV.

Asserts are critical to debugging and verifying of the model. We found that having too many asserts is preferable to having too few.

2.6.4 Case Study: Anomaly Detection

The scenario illustrated in figure 2.3 represents a portion of what should occur when the video operator believes a target has appeared on the video GUI. Each vertical swimlane represents an Actor/ DiRG. Periodically during a flight the UAV will fly over an anomaly. An anomaly can be either a false positive or a true positive, meaning that it is either the desired target or it is not. If the video operator believes that it is the target then a flyby request is made through the video GUI. This request is then made visible to the operator through the operator GUI. When the operator decides to perform the flyby request he signals this through the operator GUI and begins to manually direct the UAV to the location of the anomaly. While the operator is directing the UAV the video operator closely examines the video stream until the anomaly is visible again. The video operator then decides if it is a true target sighting or a false positive. The video operator communicates this to the operator through the video GUI. If it was a target sighting then the video operator passes the information to the mission manager who then passes it to the parent search. This high level view communicates the basic structure of the communication between the different actors.

2.7 Results

In this section, we first discuss model-checking results and then present insights from the modeling process.

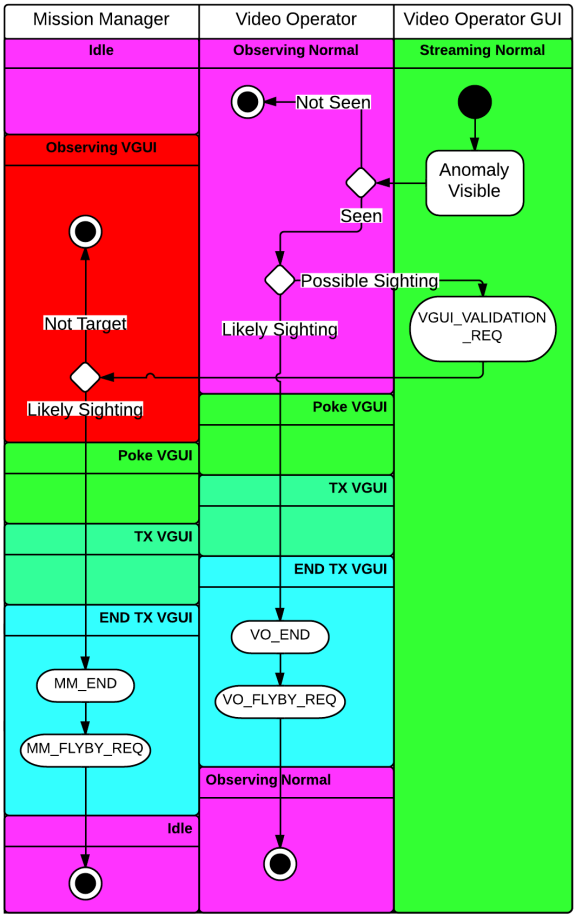

Figure 2.3: Anomaly Detection Model: Swim lanes represent actors. Arrows represent input/output. Colored sections represent actor states.

2.7.1 JPF Results

Model checking constructs an exhaustive proof, by enumerating every reachable state of a system, to establish a property. The JPF model checker uses native Java as the modeling language to describe a system, and implements a custom virtual machine to systematically explore the reachable state space of the compiled Java program. The reachable state space of a sequential model is trivial to enumerate but the reachable state space grows exponentially with the level of non-determinism (i.e., random choice).

The source of non-determinism in the WiSAR model is in timing. The model defines time bounds for different events and tasks in the system. When the model is run, it non-deterministically chooses delays from within those bounds. For example, the time to land the UAV is a uniform random variable bounded between 60 and 1,800 time units. In the WiSAR model, there are 69 different tasks or events with non-deterministic durations. As it is not feasible to enumerate the entire reachable state space of such a large model, this work implements several standard heuristics to limit the non-determinism: minimum(maximum) time only; minimum and maximum times only; and minimum, maximum, and mean times only. As expected, the first heuristic using only the minimum(maximum) time bound results in a sequential model with no non-determinism.

The total number of meaningful lines of code, code within methods, for the WiSAR model is 3,804. JPF is able to analyze the model using the minimum heuristic in negligible time only enumerating 124 states on a MacBook Air with 4GB of memory and an Intel Core I7 1.6 Ghz processor. The maximum heuristic also has negligible time but fewer behaviors with only 16 states. The difference in states is due to the UAV not needing to land several times to recharge its battery.

The minimum and maximum bounds heuristic in JPF generates an important and interesting result. JPF finds a combination of task durations that result in an infinite loop using the heuristic. The same infinite loop can be recreated outside of JPF by repeatedly running the model over and over again as random trials until one of the trails goes into an

infinite loop. The power of using JPF is that it finds the infinite loop every time without the need for the random trials. The root cause of the infinite loop was a flawed communication protocol that implicitly relied on specific delays in the interaction. The protocol has since been corrected and JPF verifies the model to now terminate under all combinations of minimum or maximum delays: 3,911 states in 6 seconds of running time.

The greatest number of states enumerated by JPF comes from using the minimum, maximum, and mean heuristic. For all 69 points of non-determinism in the WiSAR model, JPF exhaustively considers 3 distinct values for each point, and checks every possible combination. JPF enumerates 51,344 states in around 52 seconds using the heuristic. None of the states violate the current set of assertions in the model and the model terminates under all the duration combinations. The jump in the number of states between the minimum and maximum bounds heuristic and this heuristic illustrates the exponential state explosion inherent in model checking.

The JPF model checking does not prove the WiSAR model is the desired model or even a correct working model. There is considerable research yet to be completed in writing the system level requirements of WiSAR and then having JPF verify each of those requirements. If JPF finds a violation on any given requirement, there is still considerable work to determine if the requirement is correct (i.e., really what is desired from the system), if the model has a bug, or if the protocols in the model are fundamentally flawed and not able to implement the requirement. These topics are future work for the WiSAR model.

2.7.2 Lessons from Modeling

One of the goals of using model-checking with UAS-enabled WiSAR is to discover problems and opportunities with the structure of the organization. This section presents several important lessons for UAS-enabled WiSAR that were obtained through the modeling exercise.

First, while modeling the VidOp it became apparent that there was a problem with the organization. This problem occurs because, while the VidOp is marking an anomaly,

the video feed continues to run. This means that it is possible that the VidOp may miss detecting the target. If the VidOp pauses the video, the feed falls behind the live video feed which makes flyby requests more expensive because the UAV will have to backtrack to the anomaly sighting. We analyzed why this problem was not discovered in the WiSAR field trials. The answer is that the field trials included multiple video feeds with multiple observers, a condition that is not likely to occur in a resource-limited search. A lesson from this observation for WiSAR is that technology needs to be developed that allows the WiSAR team to manage this problem. A more general lesson is that the modeling and model-checking process uncovered a potential problem before it appeared in practice.

A second lesson was learned when performing model-checking of a flyby. Our model showed two problems, resuming a flight plan after a flyby and needing to keep a list of flyby requests. We solved this in the model by adding visible queues to the VeGUI and VidGUI and allowing the VeGUI to store multiple flight plans. As before, we analyzed why these problems were not discovered in the WiSAR field trials. The answer is that these problems did occur but were not documented. The lesson for WiSAR is that the VeGUI and VidGUI need new features to support real searches. A broader lesson is that the modeling exercise can be used to not only detect problems but specify the requirements for fixing them.

2.8 Conclusions and Future Work

We have presented DiRGs expressed as Mealy state machines for the purpose of modeling WiSAR in a way that will give insight into improving the WiSAR processes. In addition we have coded these Mealy machines in Java and performed model-checking using the JPF tool for the purpose of gaining even more insight into our model by running it. This contrasts with previous modeling attempts. Results show that additional insight was gained, and that it was possible to introduce new processes into the model and see the effect of those changes.

The result of the modeling and model checking processes was the detection of problems within WiSAR that were not seen during other analysis, or were seen but not documented.

The processes also gave insight, and in some cases specifications, for fixing the encountered problems.

Future work will use explicit declarations to formalize Actor states and transition matrixes. By formalizing these properties it will be possible to find transition errors immediately; it will also make it possible to compare the model with the model documentation for accuracy. This may also make it possible to export the state machine into other model checkers.

We also plan to add sequential constraints to the model using Actors. These Actor will embody the desired sequence of tasks and transitions and will throw assertions if a sequence is not executed in the desired order. This will help us verify that the model is following the desired behaviors.

Acknowledgment

The authors would like to thank Neha Rungta of NASA Ames Intelligent Systems Division for her help with JPF and Brahms. The authors would also like to thank the NSF IUCRC Center for Unmanned Aerial Systems, and the participating industries and labs, for funding the work. Further thanks go to Jared Moore and Robert Ivie for their help coding the model and editing this paper.

Chapter 3

Modeling Human Workload in Unmanned Aerial Systems

J. J. Moore, R. Ivie, T.J. Gledhill, E. Mercer and M. A. Goodrich. Modeling Human Workload in Unmanned Aerial Systems. In Proceedings of AAAI 2014 Spring Symposium on Formal Verification & Modeling in Human-Machine Systems.

Abstract

Unmanned aerial systems (UASs) often require multiple human operators fulfilling diverse roles for safe correct operation. Although some dispute the utility of minimizing the number of humans needed to administer a UAS [36], minimization remains a long-standing objective for many designers. This paper presents work toward understanding how workload is distributed between multiple human operators and multiple autonomous system elements in a UAS across time, with an ultimate goal to reduce the number of humans in the system. The approach formally models the *actors* in a UAS as a set of communicating finite state machines, modified to include a simple form of external memory. The interactions among actors are then modeled as a directed graph. The individual machines, one for each actor in the UAS, and the directed graph are augmented with workload metrics derived from a review of the relevant literature. The model is implemented as a Java program, which is analyzed by the Java Pathfinder (JPF) model checker, which generates workload profiles over time. To demonstrate the utility of the approach, this paper presents a case study on a wilderness search and rescue (WiSAR) UAS analyzing two different mission outcomes. The generated workload profiles are shown to be consistent with known features of actual workload events in the WiSAR system.

3.1 Introduction

Unmanned aerial systems (UASs), ranging from large military-style Predators to small civilian-use hovercraft, usually require more than one human to operate. It is perhaps ironic that a so-called "unmanned" system requires multiple human operators, but when a UAS is part of a mission that requires more than moving from point A to point B, there are many different tasks that rely on human input including: operating the UAS, managing a payload (i.e., camera), managing mission objectives etc. Some argue that this is desirable because different aspects of a mission are handled by humans trained for those aspects [36]. As human resources are expensive, others argue that it is desirable to reduce the number of humans involved.

This paper explores the open question of *how* to reduce the number of humans while maintaining a high level of robustness. Some progress has been made by improving autonomy using, for example, automatic path-planning [**?** **?** **?** **?** **?**], and automated target recognition [**?** **?** **?**]. However, careful human factors suggest that the impact of changes in autonomy are often subtle and difficult to predict, and this decreases confidence that the combined human-machine system will be robust across a wide range of mission parameters [**?** **?** **?**].

The research in this paper argues that one reason for the limitations of prior work in measuring workload is that the level of resolution is too detailed. For example, although the NASA TLX dimensions include various contributing factors to workload (e.g., physical effort and mental effort), the temporal distribution of workload tends to be "chunked" across a period of time. Secondary task measures can provide a more detailed albeit indirect breakdown of available cognitive resources as a function of time [**?**], but with insufficient explanatory power for what in the task causes workload peaks and abatement. Cognitive workload measures, including those that derive from Wickens' multiple resource theory [43], provide useful information about the causes of workload spikes, but these measures have not been widely adopted; one way to interpret the research in this paper is as a step toward robust implementations of elements of these measures. Finally, measures derived from cognitive

models such as ACT-R are providing more low-level descriptions of workload which potentially include a temporal history [**?**], but these approaches may require a modeling effort that is too time-consuming to be practical for some systems.

This paper presents a model of four human roles for a UAS-enabled wilderness search and rescue (WiSAR) task, and is based on prior work on designing systems through field work and cognitive task analyses [9, 22]. The paper first identifies a suite of possible workload measures based on a review of the literature. It then considers seven *actors* in the team: the UAV, the operator and the operator's GUI, the video analyst and the analyst's GUI, the mission manager, and a role called the "parent search" which serves to connect the UAS technical search team to the other components of the search enterprise. The paper then presents the formal model of each of these actors using finite state machines, and then discuss how the connections between these state machines defines what is called a *Directed Team Graph* (DiTG) that describes who communicates with whom and under what conditions. The paper then describes how to augment the model to be able to encode specific metrics based on a subset of the measures identified in the literature. Using the Java PathFinder model checker, a temporal profile is created for each of the workload metrics. These profiles are checked for consistency by associating workload peaks and abatement with likely causes.

3.2 Workload Categories

Workload is restricted to three general categories of metrics in this work: cognitive, temporal, and algorithmic.[1]

Cognitive workload describes the difficulties associated with managing various signals, decisions, and actions relevant to a particular task or goal [**? ? ? ?**]. We adopt a simple form of Wickens' multiple resource theory [43], and make the simplifying assumption that cognitive workload can be divided into two categories: parallel sensing and sequential decision making. We further restrict the sensing channels to visual and auditory modalities, ignoring

[1] A fourth relevant workload category is the cost of maintaining team constructs like shared situation awareness is future work [**?**].

haptic. Parallel sensing means that it is possible for a human to perceive complementary stimuli over different channels. An example of this would be an individual hearing their call sign on the radio while analyzing video. However, when multiple signals may occur over the same channel at the same time, this induces attentional workload for the human. Sequential decision making occurs when a decision must be made, where we have adopted the assumption made by Wickens and supported by work in the psychology of attention [**?**] that a "bottle-neck" occurs when multiple channels either (a) require a decision to be generated or (b) exceed the limits of working memory.

Algorithmic workload results from the difficulty of bringing a task to completion. Adopting a common model from artificial intelligence [**?**], and consistent with Wickens' three stage multiple resource model, we assume that this is comprised of three phases: sense, plan, and act. During the sensing phase, the actor takes all active inputs, interprets them, and generates a set of relevant decision-making parameters. In the planning phase the actor reviews the breadth of choices available and selects one. The actor might use search or a more naturalistic decision-making process like recognition-primed decision-making [**?**] or a cognitive heuristic [**?**]. In the acting phase the actor carries out the decision. Before concluding, we note that workload is highly dependent on the experience of the actor [**?**], but we leave a careful treatment of this to future work.

Temporal workload deals with the scheduling of prioritized, infrequent, and/or repetitious tasks [**?** **?**]. Various measures have been proposed, but we are most interested in those related to so-called "fan-out", meaning the number of tasks that a single actor can manage [18, 26**?** **?**]. There are two particularly important aspects of temporal workload. First, when a task is constrained by (a) the time by which the task must be completed or (b) the need to complete other tasks before or after the given task then (c) it causes scheduling pressure and workload [**?**]. The second aspect is operational tempo, which represents how frequently new tasks arrive. From a scheduling or queuing theory perspective, operational tempo impacts

workload by causing pressure to manage the rate of arrival and the response time of the decision.

3.3 Actor Model

In previous work [4], we represented each *actor*, human or autonomous component, of the WiSAR search team as a Mealy state machines as has been done in other models [3]. This work uses Moore machines where output is determined only by present state.

Actors represent the human decision-makers and autonomous elements of the WiSAR team. An actor is composed of a set of states S, an initial state s_0, a set of inputs from the environment Σ_{env}, a set of inputs from other actors on the team Σ_{team}, a set of outputs Λ_{out}, a simple form of memory Ω_{men}, and a transition function that determines the next state from the inputs and memory δ. Formally, we denote an actor as:

$$Actor = (S, s_0, \Sigma_{\text{env}}, \Sigma_{\text{team}}, \Lambda_{\text{out}}, \Omega_{\text{mem}}, \delta) \tag{3.1}$$

An actor's output has two components: signals to other actors on a team and a *duration* parameter that represents the time required for the actor to complete its transition to the next state. Thus, $\Lambda_{\text{out}} = \Sigma_{\text{team}} \times N^+$. Relative task difficulty is expressed by the duration of the transition. We justify this by assuming that all tasks are performed at a constant rate, thus more difficult tasks take longer. A transition is a relation on the cross product of inputs with outputs where the brackets separate inputs from outputs.

$$\delta : [S \times \Sigma_{\text{env}} \times \Sigma_{\text{team}} \times \Omega_{\text{mem}}] \times [S \times \Sigma_{\text{team}} \times \Omega_{\text{mem}}], \tag{3.2}$$

Given an actor's current state, set of input signals, and memory, it is possible for multiple transitions to be possible. This occurs because we assume that multiple environment signals or inter-actor signals may be occurring at the same time, which are all perceived by the actor since we assume perception is a parallel operation. Thus, it is useful to explicitly

note the number of transitions that are possible from a given state. A transition is considered *enabled* when all of its input requirements are met and *disabled* otherwise.

When an actor is in a given state, it is useful to explicitly denote the set of enabled and disabled transitions. This information enables an estimate of algorithmic workload, where we assume that algorithmic workload is a function of the number of choices available to the actor. Thus, we allow the current state to give a workload signal

$$s_0^{\text{work sig}} = (T_{enabled}, T_{disabled}) : T_{enabled} \cap T_{disabled} = \emptyset \tag{3.3}$$

Internal variables within an actor are comprised of facts stored by an actor and used in decision making. A good example of this can be seen in the mission manager actor of our simulation. As the search begins, the mission manager receives a number of data items from the parent search, e.g. search area and target description. These items of information need to be communicated separately to different actors, so they must be stored internally in the mean time.

3.4 Directed Team Graph (DiTG)

A key element of the actors is that inputs to one actor can be outputs from another actor. We can therefore create a directed graph from actor to actor, with an edge from actor A to actor B existing if the output from actor A is a possible input to actor B. We call this graph a *Directed Team Graph* (DiTG); Figure 3.1 illustrates the DiTG for the WiSAR team used in this paper.

Using the fact that multiple resource theory indicates that visual and auditory channels can be perceived in parallel [43], it is useful to label the edges in the graph with the *channel type* as in Figure 3.2. These labels allow the model-checker to identify when multiple signals are given to a single actor over the same channel. When this occurs, we expect actor workload to be high.

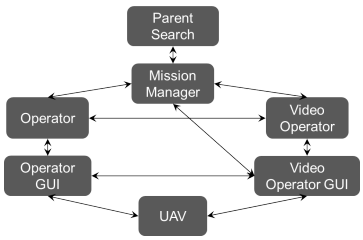

Figure 3.1: High Level DiTG

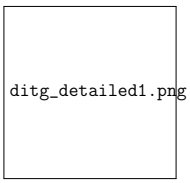

Figure 3.2: Detail view of DiTG: V is a Visual channel and A is an Audio channel

We represent a system as a DiTG, a collection of actors connected to one another by a set of channels. Whenever the state of the system changes, an actor will petition from its current state, a list of enabled transitions thus defining what decisions can be made. The actor may then activate one of these transitions.

Since transitions from one state to another take time, as encoded in the *Duration* element of the output, it is useful to label transitions as either *active* or *fired*. Transitions in the actor model are labeled as *enabled* and *disabled*. When we consider the workload of an actor as part of the overall team, workload depends on what is going on with other team members, so we add the active and fired labels in order to determine when an enabled transition (meaning a possible choice available to an actor) is chosen by an actor (making it active) and when the actor completes the work required to enter the next state (the transition fires).

From an implementation perspective, when a transition becomes active it creates temporary output values for declarative memory and channels. These temporary values are then applied to the actual declarative memory and channel values once the transition fires.

For our model we never explicitly define a single task. Instead we define actors, states, and transitions. Each transition defines its own perceptual, cognitive, response, and

declarative resources [38], allowing the model to represent multiple possible tasks.[2] In this way, an actor's state determines what task(s) are being performed, achieving multi-tasking without explicitly defining tasks.

To simplify the modeling process and ensure rigorous model creation we developed a transition language, similar to a Kripke structure,[3] which allows models to be expressed as a list of Actor transitions. A parser then automatically generates the classes required to run the model simulation. The transition language uses the following structure.

$$\begin{aligned}(s_{current}, [\phi_{input} = value, \ldots], [\omega_{input} = value, \ldots], \\ duration) \times \\ (s_{next}, [\phi_{output} = value, \ldots], [\omega_{output} = value, \ldots])\end{aligned} \tag{3.4}$$

The language is compiled to a Java program suitable to run standalone as a simulation or analyzed by the JPF model checker to create workload profiles.

3.5 Workload Metrics

We are now in a position to combine the three categories of workload (cognitive, algorithmic, and temporal) with the formal model of the actors and team to generate a set of workload metrics. Because the categories include many possible measurements that are beyond the scope of this paper, we use labels for the workload metrics that are slightly different from the workload categories. As shown in Figure 3.3, cognitive workload or resource workload as it is termed in this work, is measured using metrics under the *resource* workload label, algorithmic under *decision*, and temporal workload is labeled the same.

[2]Grant, Kraus, and Perlis have done exceptional work compiling formal approaches to teamwork. While the formalisms are not explicit in our modeling, our informed modelers apply these approaches. For example, joint intentions are represented in actors containing complement transitions.

[3]Since our purpose behind building a model was to allow us to examine workload we decided against using standard model checkers such as Spin since they did not provide the same level of flexibility that we obtained via java and JPF.

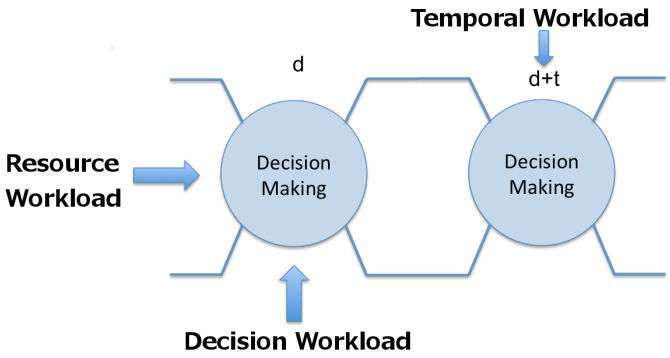

Figure 3.3: Workload in the model.

Resource workload is separated into both inter-actor communication and actor memory load. Decision workload can be broken down into timing, algorithm complexity, and complexity of the solution. Temporal workload includes operations tempo, arrival rate, and response time.

Java Pathfinder (JPF) is a tool used to explore all points of nondeterminism. It does this by compiling the source code as a JPF binary and running it in a virtual machine. Sections of the program are then dynamically altered

The virtual machine can dynamically alter sections of the program and concurrently generate and run copies of the program based off of these changes. This allows us to include alterable values in our model and simulate an array of different actors without the need to modify the model.

We have set up a JPF listener to record workload. A JPF listener is a tool that follows the Listener Design Pattern and acts in the expected fashion. We have focused our listeners on three pieces of the model: the active inputs, enabled transitions, and the time taken to perform a transition. We then use these pieces of the model to represent resource, decision, and temporal workload respectively.

3.6 Results

In the interest of consolidating operators it is critical to find an accurate measurement that detects situations that exceed the capacity of a given human. One way to detect this is by building a map of each actor's workload as a function of time. JPF explores all possible paths the model can take and returns the ones that violate the model's criteria. By augmenting our model with the workload metrics we can identify all possible areas of high workload.

We propose three levels of increasing validity for evaluating the approach in the paper. The first level, the one used in this paper, is to check for *consistency*. We say that the approach is *consistent* if the workload peaks, valleys, and trends match what we know about a small set of given situations; in other words, the approach is consistent if it matches our expectations on tasks that we know a lot about.

The second level, which is an area for future work, is to check for *sensitivity*. We say that the approach is sufficiently *sensitive* if we can use JPF to find new scenarios that have very high or very low workloads, and we can then generate a satisfactory explanation for the levels of workload by evaluating the new scenarios. The third level, another area of future work, is to *substantiate* workload levels using experiments with human participants by comparing the perceived workload of humans with those predicted by the model.

In this paper, we restrict attention to finding areas of high and low workload, and then checking these areas for consistency. We evaluated consistency using two scenarios. The first is when the Video Operator was able to identify the target during a flight without any complications occurring (see Figure 3.4). As the workload measure currently does not have units, the plots are normalized in each category. For the first 40 time steps everything behaves as expected with low to moderate workload. An initial workload bump occurs as actors exchange information necessary to start a search. At time step forty we see a dramatic deviation from the norm. This deviation is a result of constant information passing between the GUIs and the operators making it only logical that the workload would increase substantially.

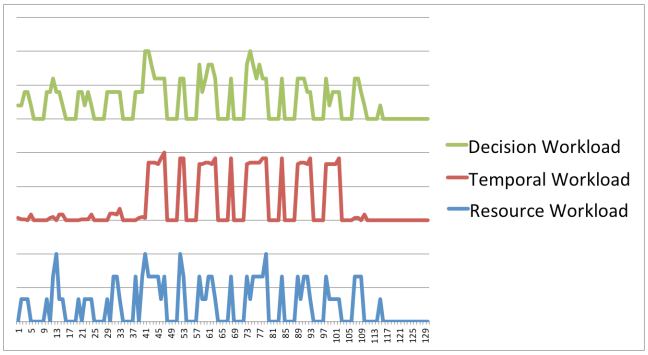

Figure 3.4: Workload over an uneventful flight.

The second simulation is a scenario where after a short period of flight the battery rapidly fails. In this particular situation, the operator was unable to respond quickly enough to land the UAV before it crashed (see Figure 3.5). There is an immediate spike in the temporal workload (middle plot), but surprisingly, the workload then decreases back to normal levels in just five time steps. The second spike revealed an unexpected fluctuation workload leading us to reexamine how information was reported. We found that there was an error in the simulator that has since been repaired. Finally, as would be expected, when the UAV crashed there was a small spike in the workload before everything came to a halt.

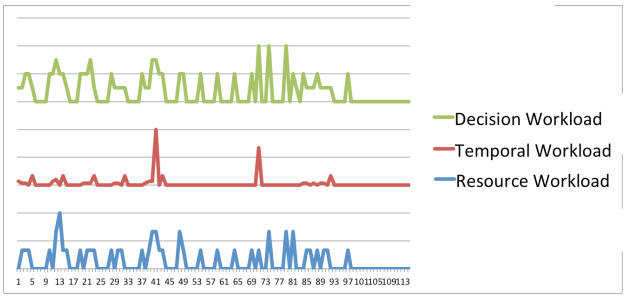

Figure 3.5: Emergency battery failure simulation

3.7 Related Work

This work is an extension of previous work which focused on modeling human machine systems, specifically WiSAR. This work extends this model to incorporate the measurement of workload [4].

Multiple resource theory plays a key role in how we are measuring workload [43]. The multiple resource model defines four categorical dimensions that account for variations in human task performance. A task can be represented as a vector of these dimensions. Tasks interfere when they share resource dimensions. Using these vectors, Wickens defined a basic workload measure consisting of the task difficulty (0,1,2) and the number of shared dimensions. Using this metric it is possible to predict task interference by looking at tasks which use the same resource dimensions. Our model differs in that we do not explicitly define tasks, instead we use Actor state transitions which may imply any number of concurrent tasks. The transition then informs us of which resources are being used and for how long.

Threaded cognition theory states that humans can perform multiple concurrent tasks that do not require executive processes [38]. By making a broad list of resource assumptions about humans, threaded cognition is able to detect the resource conflicts of multiple concurrent tasks. Our model differs from threaded cognition theory in that it does not allow learning nor does our model distinguish between perceptual and motor resources. In almost all other aspects our model behaves in a similar fashion.

Related work on temporal workload has attempted to predict the number of UAVs an operator can control, otherwise known as *fan-out* [19**?** **?**]. This work used queuing theory to model how a human responds in a time sensitive multi-task environment. Queuing theory is helpful in determining the temporal effects of task performance by measuring the difference between when a task was received and when it was executed. Actors can only perform a single transition at a time, similar to queuing theory, but it is possible for each state to take input from multiple concurrent tasks which differs from standard models of queuing theory.

ACT-R is a cognitive architecture which attempts to model human cognition and has been successful in human-computer interaction applications [11**?**]. The framework for this architecture consists of modules, buffers, and a pattern-matcher which in many ways are very similar to our own framework. The major difference is that ACT-R includes higher levels of

modeling detail, such as memory access time, task learning, and motor vs perceptual resource differences. Our model exists at a higher level of abstraction.

Complementary work has been done using Brahms. Brahms is a powerful language that allows for far more detail than we found our research required. In addition at the time we started developing our model Brahms lacked some of the tools we needed for extracting workload data from the model. Because of this we found that Java in conjunction with JPF made a better match for our needs.

3.8 Summary and Future Work

This paper proposes a model-checking approach to analyzing human workload in an UAS. Humans and other autonomous actors are modeled as modified Moore machines, yielding a directed graph representing team communication. Workload categories are distilled from the literature, and the models of the actors and team are augmented so that specific workload metrics can be obtained using model-checking. Preliminary analysis demonstrates a weak level of validity, namely, that the temporal workload profile is consistent with expected behavior for a set of well-understood situations. For these scenarios, inter-actor communication is a primary cause of spikes in workload.

A sensitivity study, followed by an experiment with real human users is needed to understand and justify the workload measures. Once we have verified that our system analyzes workload correctly, it will be useful to design a GUI optimized to managing workload and to formulate a generalized model that will have application to other human-machine systems.

Acknowledgment

The authors would like to thank the NSF IUCRC Center for Unmanned Aerial Systems, and the participating industries and labs, for funding the work.

Chapter 4

Refactoring the Modeling Framework

By far the most challenging aspect of this work was the creation of the models. By constructing the simulation framework as a set of Java interfaces and abstract classes it meant that the framework was almost entirely extensible in any direction. We found this very refreshing during the first few development iterations of the framework and model. We created different Actor types and different sub-Actors within those types. Each transition used an anonymous method containing logic for enabling the transition. Eventually as our model grew in size and complexity the freedom of the Java language became a dual edged sword. Our model was full of implicit declarations, inconsistencies, duplicated code, and anonymous methods. It became extremely difficult to maintain the model let alone add to it. This also made it difficult to rely on the metrics we were gathering as each Actor and Transition could implement metric reporting differently. This chapter covers some of the changes which were made to the simulation framework to help address these issues.

4.1 XML Model Parser

To help reduce duplicated code, inconsistencies, and the number of anonymous methods as well as making the models easier to work with we decided to implement an XML model parser. The model parser would allow the modeler to focus specifically on the model without needing to worry about code. Generic XML classes for the Team, Actor, Transition, and Predicates were created. This meant that every Actor and Transition within the system used

the exact same code. Instead of using an anonymous method for transitions the predicate class allows the modeler to define when a transition is enabled or disabled.

4.1.1 Attaching to the Simulator

The actual XMLModelParser class parses the XML, see appendix C, which generates the generic xml team, actor, and transition objects. Each of these generic XML classes lies separate from the core simulation code interacting through interfaces or by extending abstract classes. This means that the addition of the XML parser does not prevent the use of custom Actors or Transition classes, use of a different model parser, or extension of our existing XML parser. In fact our XML parser is meant to be extended to handle a more robust set of models. As we developed a model using this XMLModelParser we had several occasions to extend its capabilities which we discuss later. Unsurprisingly it was not difficult to extend the XML to accommodate the changes, often requiring less than 100 lines of code (not including changes to the model XML).

4.1.2 Validation

The use of an XMLModelParser also allowed us to add model validation. Initial attempts at modeling WiSAR have shown us that model validation time increases dramatically as model complexity increases. By restricting the model to XML, which is parsed into Java classes, we are able to catch many of the basic modeling errors such as typos, an incomplete DiTG, invalid transitions, and more. The parser tells the modeler exactly where and what the problem was helping the modeler fix the problem. While this does not prevent all modeling errors the time to model was drastically reduced. Although not related to the XML as part of this re-factor we have added several layers of debugging output which is very useful for fixing the more complicated modeling errors.

4.2 Channel Layers

While constructing the UAS integrated into NAS model we noticed that a lot of state information was being passed simultaneously across the different channels. The simulation framework only allowed a single object to be sent across a channel at a time. While this had the potential to send as much data as needed across the channel we had no way of determining how much data was going across the channel. With the creation of the XML model parser we limited channel objects to strings, ints, and booleans. This left us with very limited options for sending lots of data, such as a list of GUI alerts, over a channel.

On examining this problem in detail we decided that our channels were fundamentally flawed. In earlier development cycles we had attempted to quantify the data being passed over a channel as we felt it may be important aspect of the workload. Arriving at this problem again but from a different perspective we decided to add channel layers. See figure 4.1. Each channel can have any number of layers. For example, a visual channel from one person to another has two channels, one which analyzes the face and another which analyzes the body. Their may be more accurate channel layering for this scenario but these layers satisfy this example. When communication occurs on this visual channel the recipient can examine as many layers as are needed for the given state. If the person is in a distracted state then maybe they are not looking at the face layer which may in turn reduce the probability of comprehending the corresponding audio channel. Another good example where this is useful is in GUIs. Each item that a GUI represents to the user can be represented as a different layer. The more layers a GUI presents to a user the more complex it becomes which potentially makes it more difficult for a person to analyze.

As part of the metric analysis we can measure how many layers are being output and read on a particular channel and incorporate this into the workload measurements. We believe that this concept of channel layers creates a more natural flow of information across the DiTG and it does so in a measurable fashion. We also believe that these types of measurements will be more effective in improving human machine interfaces because it allows the specific

outputs of those interfaces to be directly modeled in a way which directly affects the human workload measurements.

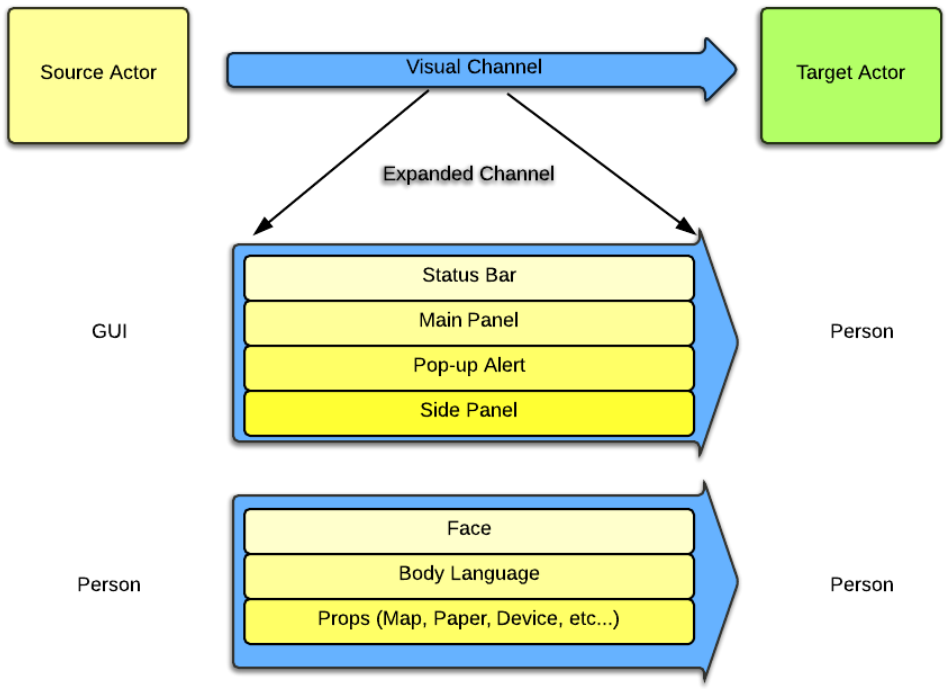

Figure 4.1: Communication Channel Layers

4.3 Metric Improvements

In examining how to extract data from the simulation we came up with two approaches. The first approach was to capture all of the raw data between each cycle and then construct metrics from these raw values such as the number of active channels, number of enabled transitions, total transition count, total output profile, etc After a brief examination of mounds of raw data it was determined that a better approach would be to simply collect specific data points that we cared about when they changed. This made it easier to collect

specific metrics, however, it created disjunct data sets which made it hard to compare the data in a time-line. Also, as mentioned above, the metric dissemination was inconsistent. This add-hoc approach to metric gathering also made it difficult to verify that the metrics being reported were accurate in relation to the other metrics. Without clear accurate metrics it is impossible to realize the true value of the simulation framework we developed.

To address these issues we used a mixed approach to gathering metrics. See figure 4.2 Each cycle the simulator asks the team for metrics, the team then asks each actor which then asks its sub-components. This gives back all of the raw metric data contained in the simulation but as it is being passed back up the chain each component can perform calculations using the raw values it knows about to return specific metric values. This maintains the timeline flow of the simulation and allows us to customize what metrics are actually calculated and displayed from the different levels. An example of this is the Actor. The Actor knows how many input channels he can listen on, his current state, etc The Actor does not directly know what transitions are enabled, how many inputs are in those transitions, etc.. Instead of trying to calculate all of this with multiple queries into the state the Actor simply asks the State object for its metrics. The State then obtains/calculates metric values, obtaining metrics from sub-components if needed, before passing this values back to the Actor and eventually back to the simulator which displays the metrics to the user. These metrics are shown in a timeline paired with the Actor state. The timeline itself represents each time a transition was fired, but more importantly it corresponds to the simultaneous state of each Actor, allowing the user to see which states have the highest workload and how an action by one Actor effects another.

4.3.1 Wickens Workload Model

Gathering the raw values from the model was the easy part. The harder part is to take these raw values and convert them into a value which reflects Actor workload. Since our model relies on Wickens Multiple Resource Theory [43]. We decided to try and replicate

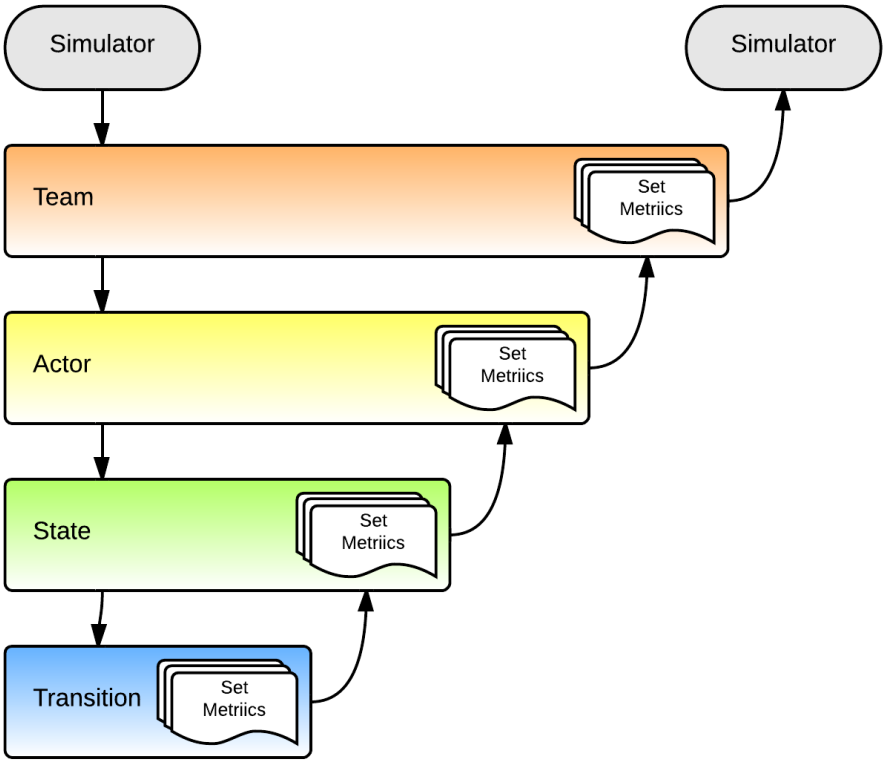

Figure 4.2: Metric Gathering

the simple computational model for predicting workload presented with the theory. Wickens computational model takes the task difficulty, ranging from 0 to 2 where 0 is automated and 2 is difficult, of two tasks and adds them together. He then adds the number of dimensional conflicts between the tasks, max of 4, which gives a result between 0 and 8 [43]. The difficulty of applying this model to our metrics is that we have removed the concept of tasks and replaced it with the notion of state. In order to replicate Wickens computational model we needed a way to represent task difficulty (resource demand) in a similar fashion. We first looked at transition duration. The longer a task takes the more difficult it is. While this might

work it prevents the modeler from placing an Actor into a long running simple task which we feel would degrade the usefulness of this framework. Next we looked at resources. The problem is that our simulation framework only tracks what resources are being used and when. Good for finding conflicts but not for determining how much load those resources are under. It is possible that Wickens ran into the same problem because his simple computational model relies on the modelers experience and intuition to set the task difficulty, which is likely more accurate than implicitly constructing task difficulty based on resource consumption. This led to the realization that we had no good way of explicitly defining an Actors task difficulty. To address this we defined a new term called Actor load.

4.3.2 Actor Load

Actor load represents an abstraction of the load an Actor is under for any given state. Similar to Wickens model we will use values from 0-4. Each Actor state will define its own load. A load of 0 represents little to no load on the actor. These are automated or transitional states. A load of 4 represents simultaneously performing multiple high difficulty tasks. Any value between 1 and 3 is some combination of task difficulty and the number of tasks being performed.

4.3.3 Applying Wickens Computational Model

By placing these values into the State portion of our models we can now replicate the simple computational model Wickens used in his measurements. Using the State load as the task difficulty the next step is to find the dimensionality of the resource conflicts. Since a state represents 1 or more tasks this also presents a challenge. To best approximate Wickens model we needed metrics which could represent one or more tasks. We accomplish this by making the assumption that if an Actor has input from multiple sources in a state then they are performing multiple tasks. It should be noted that we check the input channels for each transition in the current state but only the outputs of the current transition. With these

assumptions we are now ready to calculate dimensional conflicts, Figure 4.3. For the Stage dimension (perception, cognition, response) we check to see if there are multiple sources of input or multiple targets for output. If there are then we increment the dimensionality. For the Modality dimension (Audio, Visual) we check if there are more than a single active channel, input or output, of type audio or visual. If there is then we increment the dimensionality. For the Focus dimension we check if there is more than one source for tactile outputs, if there are then we increment the dimensionality. We do not check inputs as we have no way of distinguishing if a visual channel has focus. This may need to be addressed in future work. For the Codes dimension (Spacial, Verbal) we check that the total number of audio inputs and outputs is greater than 1 or that the total number of visual inputs, visual outputs, and tactile outputs is greater than 1. If either value is greater than 1 then we increment the dimensionality.

By adding the task difficulty (state load) and the dimensionality together we obtain a modified Wickens metric which we can then compare with our own metrics.

4.3.4 Changes to our Metrics

For the resource workload category we now generate Actor metrics the following way. Inputs are gathered from each transition that is part of the current state. The outputs are collected from the active transition. We also use the Actor load from the current state. There are 2 main metrics the channel conflicts and the resource load. Channel conflicts occur whenever more than one active channel share a type, such as multiple audio channels. Since we currently only allow visual and audio input for human Actors this value ranges from 0 to 2. The other metric is the resource load. This metric attempts to quantify the load being placed on the Actors resources. We break the resource load into two parts, input and output, the final result being the sum of both parts. Each part is calculated by adding the number of active channels, number of layers read, number of memory objects read, and the number of active channel types.

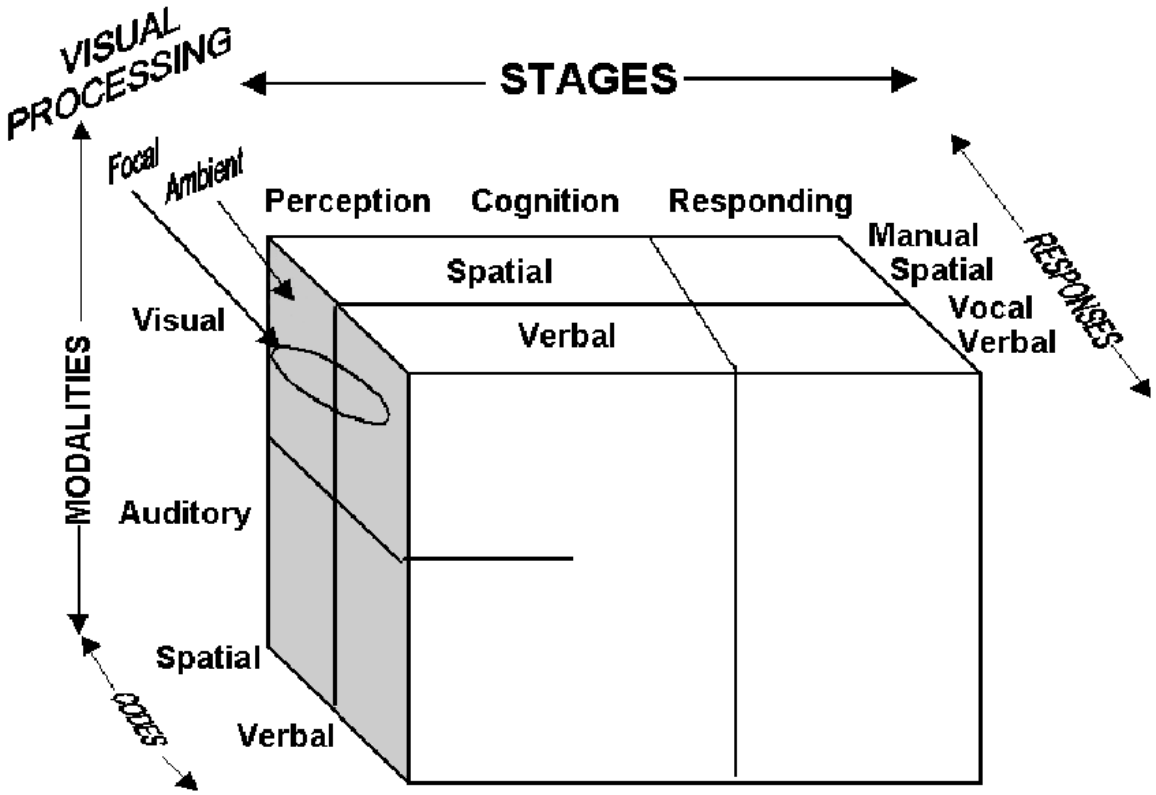

Figure 4.3: Multiple Resource Theory Dimensions

For decision workload we have added input complexity and output complexity. The input complexity is the total number of active inputs plus the number of memory inputs. The output complexity is the number of output channels plus the number of memory outputs. While there is overlap between these values and the resource workload metrics we leave it up to future work to analyze this relationship. We have also modified the duration complexity metric. Before adding Actor load we relied on the duration complexity to inform us of task difficulty. We no longer apply the same weight to the durations. Instead we are now using a logarithmic scale. By assuming that durations are in seconds we classify any transition under a minute as 0 complexity and move up from there. Our reasoning for this normalization is two fold. First it is reasonable to assume that the more time a transition takes the more workload it requires, however, it is also reasonable to assume that there are diminishing returns associated with increasing the workload. We would also like to compare the metrics

together. By normalizing this metric to a value between 0 and 6, for our model, we can show this metric side by side with the others.

4.3.5 Other Changes

We also performed other refactorings to the simulation framework which facilitate the previously described changes and more. As part of this re-factoring the connections to JPF were temporarily disabled in order to simplify the debugging process. The results described in the next chapter were obtained by running the simulation framework as a stand alone application outside of JPF. While this does prevent a deeper evaluation of the models state space the core model behavior still remains the same. Unfortunately it prevented us from collecting the temporal workload metrics from the model.

Chapter 5

Case Study: UAS operating in the NAS

According to the UAS integration into the NAS roadmap [41] the FAA is working with other government agencies and industry to develop a collaborative UAS modeling and simulation environment to explore key challenges to UAS integration. The near-term modeling goals are to:

- Validate current mitigation proposals
- Establish a baseline of end-to-end UAS performance measures
- Establish thresholds for safe and efficient introduction of UAS into the NAS
- And develop NextGen concepts, including 4-dimensional trajectory utilizing UAS technology

We believe that our modeling language accomplishes each of these goals. Our modeling language is extremely flexible allowing it to model a large variety of systems. We have the ability to detect critical failures. We have the ability to examine multiple paths through the model and to randomly perturb the model in different ways to explore these paths. We have the ability to model specific performance measures as extensions to the base model. While this is left to future work the concept of a set of Actors which can be attached to a third party model to perform performance measuring is very attractive when attempting to standardize model behavior. We also have the ability to generate human workload metrics is valuable for setting safety thresholds and analyzing new designs. We also have the ability to use different levels of abstraction for each portion of our model. This is ideal while attempting

to develop new concepts as it allows us to predict the results of model changes without an exact understanding of how the changes will be implemented.

The document [41] also identifies several interrelated research challenges:

- Effective human-automation interaction (level; trust; and mode awareness)
- Pilot-centric ground control station design (displays; sensory deficit and remediation; and sterile cockpit)
- Display of traffic/airspace information (separation assurance interface)
- Predictability and contingency management (lost link status; lost ATC communication; and ATC workload)
- Definition of roles and responsibilities (communication flow among crew, ATC, and flight dispatcher)
- System-level issues (NAS-wide human performance requirements)
- And airspace users and providers qualification and training (crew/ATC skill set, training, certification, and currency)

Our modeling language was specifically designed to target challenges 1, 4, 5, and 6. We hope that our work may be effective in overcoming the other challenges as well but we leave this to future work.

To demonstrate the recent changes and additions to our modeling language we chose to model a basic UAS integrating with the NAS. Due to our lack of domain knowledge we have had to make a fair number of assumptions in order to achieve a high level of abstraction. Despite the high level of abstraction we believe that the results are still quite impressive.

5.1 Model Scenario and Assumptions

A UAS plans to operate within the NAS. The UAV will take off and land at an airport serving both manned and unmanned aerial vehicles. The UAV Operator is located at this

airport and visually monitors the takeoff and landing of the UAV. UAV state is continuously shown on the UAS GUI.

There is an FAA system which provides real-time notice to airmen (NOTAM) information. The UAS connects to this system and displays these NOTAMs on a GUI. The FAA system also allows UASs to file flight plans. Filed flight plans are automatically checked for simple conflicts such as crossing NOTAMS or duplicate takeoff/landing times. If there is a conflict the FAA system flags the flight plan for an ATC (Air Traffic Controller) and displays these requests on its GUI. The ATC then approves or denies flight plans, using the FAA System GUI, at their leisure. This approval/denial then becomes available to the UAS which displays it on its GUI.

The FAA system also provides radar information to the ATC through its GUI. The ATC uses this information to spot potential collisions with UAVs, it is assumed that the UAS also sends near real time position information to the FAA system although that is not required for this to work. If a potential UAV collision is detected the ATC creates an emergency NOTAM in the region of conflict. This emergency NOTAM is seen by the UAS. If the UAV is in or near the emergency NOTAM it must change course to immediately evacuate/avoid the NOTAM. This can be done automatically by the UAS or manually by the UAV Operator. Once the UAV has finished avoiding the emergency NOTAM it enters a loiter state. The UAV Operator is then required to change the flight plan before the UAV will leave the loiter state.

The UAS also has a radar for the UAV which can detect nearby objects. This information is displayed on its GUI and if the UAV Operator detects a potential collision they will begin the deconfliction procedure which requires changing the current flight plan. Once the UAV has landed the scenario is considered complete.

5.1.1 Assumptions

The number of assumptions made in this scenario is too great to fully list. Instead we have only attempted to list the major assumptions which are required for the model to perform as designed.

- The UAV has an unlimited flight time, never loses contact with the UAS, can takeoff and land without incident, and has accurate GPS data, is non line-of-sight.
- The UAS never loses connection to the FAA System, all communication with the UAV and FAA System is instant, no bugs, can create flight plans, can detect NOTAMs on the flight plan, can automatically direct the UAV out of an emergency NOTAM, displays radar information from the UAV.
- The UAV Operator detects all warnings displayed on the UAS GUI, generates flight plans which do not touch NOTAMS, can always deconflict the UAV, never gets fatigued.
- The FAA System distributes NOTAMS and automatically detects if a flight plan needs to be approved by the ATC.
- The ATC detects all information displayed on the FAA System GUI, can add NOTAMS to the FAA System, always adds NOTAMS correctly, approves all flight plans.

5.2 Building the Model

Building the model using XML was a very different experience. In retrospect it would have been valuable to have WiSAR modeled in Java and XML so a direct comparison would be available. Initially working with a single xml file instead of multiple Java class files was very convenient. Additionally the ability to validate the model each time we added a new transition led to very rapid development. Unfortunately as the model complexity increased the added readability, see figure 5.1, was not enough to counter the complexity as the number of transitions and inter-Actor communications increased.

```
// (LOITERING,[D=OGUI_FLYBY_START_F_UAV],[],1,NEXT,1.0)X(FLYING,[V=UAV_FLYING_OP,D=UAV_FLYING_OGUI,D=UAV_FLYING_VGUI],[FLYBY=TRUE])
LOITERING.add(new Transition(_internal_vars, inputs, outputs, FLYING, Duration.NEXT.getRange(), 1, 1.0) {
    @Override
    public boolean isEnabled() {
        if(!OperatorGui.DATA_OGUI_UAV_COMM.OGUI_FLYBY_START_F_UAV.equals(_inputs.get(Channels.DATA_OGUI_UAV_COMM.name()).value())) {
            return false;
        }
        setTempOutput(Channels.VIDEO_UAV_OP_COMM.name(), UAV.VIDEO_UAV_OP_COMM.UAV_FLYING_OP);
        setTempOutput(Channels.DATA_UAV_OGUI_COMM.name(), UAV.DATA_UAV_OGUI_COMM.UAV_FLYING_OGUI);
        setTempOutput(Channels.DATA_UAV_VGUI_COMM.name(), UAV.DATA_UAV_VGUI_COMM.UAV_FLYING_VGUI);
        setTempInternalVar("FLYBY", true);
        return true;
    }
});
```

(a) Java Transition

```
<transition priority="0" durationMax="900" durationMin="900">
    <description>Monitoring the UAVGUI</description>
  - <inputs>
      - <channel name="VISUAL_UAS_UAVOP">
            <layer name="UAV" predicate="eq">FLYING</layer>
            <layer name="RADAR" predicate="ne">COLLISION</layer>
            <layer name="EMERGENCY_NOTAM" predicate="ne">EMERGENCY_NOTAM</layer>
            <layer name="NEW_NOTAM_ON_FLIGHT_PATH" predicate="ne">NEW_NOTAM_ON_FLIGHT_PATH</layer>
        </channel>
    </inputs>
  - <outputs>
      - <channel name="TACTILE_UAVOP_UAS">
            <layer name="MOUSE">MONITORING_UAVGUI</layer>
        </channel>
    </outputs>
    <endState>MONITOR_UAVGUI</endState>
</transition>
```

(b) XML Transition

Figure 5.1: Transition Readability Comparison

The Enhanced Operator Function Model (EOFM), a similar modeling language which is XML based, is able to be seen visually as a tree-like graph due to the modeling structure [**?**]. This visualization allows the modeler to see the hierarchical ordering of actions belonging to a task. Our xml structure does not lend itself naturally to a visualization which grants additional insight into the model behavior.

5.2.1 Modeling Approach and Common Errors

Past experience has shown us that modeling these complex systems can become very difficult. In an effort to reduce the complexity we used the following modeling approach. We first defined each Actor in the system and gave them a starting state. From here we broke the work into task sequences. Each task sequence represents the flow of data through the system, initiating from an external event. For example our model begins with a start mission event

which is received by the UAV Operator. We then implement transitions in sequence until no new transitions are needed. Each time a new transition is added to the model we run the model to check that the XML is correct and to verify that the new transition is reached. The full model can be seen in appendix C.

While this approach was effective it was not a trivial task. By design the transition sequence moves between Actors requiring the modeler to constantly jump around in the model XML. Transition sequences will often fork when the action of one Actor causes multiple Actors to respond. This often results in truncated branches within the model which get forgotten. One reason for running the model after adding a new transition is that when the complexity causes modeling errors they typically appear in two fashions. The first is when an Actor becomes stuck in a state. This is the more desirable behavior as it is easier to track down. The second typical error is an infinite loop. These can be much more difficult to track down. Infinite loops are often caused by missing transitions, which is ok while constructing the model. If the loop is not caused by a missing transition then more in-depth analysis is needed to find the error. The Java debugger along with our different debug levels and outputs makes tracking down these errors rather straight forward although the process can be time consuming. While there are many ways to get into an infinite loop we will only mention the two most common, they are improper transition overriding and uncleared output channels. Transition overriding happens when you have one transition which is overridden by another before it can fire. While transition overriding is a desirable behavior, gaps in the transition logic can cause an Actor to continuously cycle between transitions without ever firing a transition and moving to the next state. This can be thought of as undefined behavior. To fix this, define the true behavior for each possible scenario by creating transitions for all combinations of inputs that the model cares about. The simulator will prevent a transition from replacing itself preventing the infinite loop. Uncleared output channels can cause an Actor to believe that a new value has been sent causing the Actor to transition into an infinite

cycle. It is often best practice to clear an Actors outputs once those outputs are no longer needed.

While the use of XML to create the model was much more efficient we still feel that a more effective approach to modeling is needed which helps standardize the modeling formats and prevent common errors. Whether this approach is a different XML structure, an entirely new format, or a GUI interface which helps visualize the data flow and transition sequences we leave to future work.

5.3 Model Results

We were very impressed with the results generated with this new case study which can be seen in figures 5.2, 5.3, 5.4, 5.5, 5.6,5.7, and 5.8 We created three variations of the model. The first variation involved an uneventful flight. The second variation involved the UAV Operator manually avoiding an emergency NOTAM. The last variation involved the UAS automatically avoiding an emergency NOTAM. The last two variations also involve a possible UAV collision and the ATC adding a NOTAM onto the UAV flight path. We currently are only showing the metrics for the human Actors. For each variation we have four charts depicting the UAV Operator workload and four charts depicting the ATC workload. The first three charts represent the resource input, resource output, and decision workload. The last chart shows the combined total workload next to the adapted Wickens workload. For each chart the y axis represents the workload value while the x axis represents actor state for the given delta time. What this means is that the x axis cannot be seen as a normal timeline since each time step can be an arbitrary value. What this does provide is the Actors workload each time something in the system changed.

5.3.1 Inter Actor Comparison

When comparing figures 5.2 and 5.3 their relationship becomes very clear. In the scenario what is happening is the UAV Operator creates a flight plan for his mission. When he finishes

the flight plan he sends it to the FAA System where it is flagged to be approved by the ATC. The ATC then performs the approval and returns back to normal operations. The UAV Operator receives the approval and begins normal mission operation which involves a slightly higher workload. This is reflected beautifully in the Actor workload. The UAV Operator workload spikes during flight plan creation then flat lines while waiting for a response. Once he receives the response he spikes again as he performs his mission. The ATC workload is just the opposite spiking while he is approving the flight plan and returning to normal afterwards. The comparison of the other variations show the same correlations. We also see from figure 5.5 that the delayed action of the UAV Operator causes the ATC to remain in a high workload state for a long period of time telling us that something is wrong. By looking at the Actor state we see that the UAV Operator was already in the process of deconflicting the UAV and was not able to immediately respond to the ATC emergency NOTAM. The last variation shows an improvement in this regard as the UAS automatically diverts the UAV out of the emergency NOTAM while the UAV Operator is busy. This example demonstrates how the workload modeling allows us to detect problems within the model then analyze fixes to those problems.

5.3.2 Intra Actor Comparison

When comparing figures 5.4 and 5.6 at first glance the metrics appear almost identical. This is good as these models are nearly identical. Upon closer examination we see that when the UAV Operator has to manually avoid the emergency NOTAM there is a prolonged increase in resource input workload, an extra spike of resource output workload, and a noticeable spike in decision workload once the UAV Operator begins the process of avoiding the NOTAM. This results in a significant workload spike compared to the more stable workload seen in the automated emergency NOTAM avoidance. Another difference which is less noticeable is that the manual emergency NOTAM avoidance takes longer to complete than the auto avoid feature something which is critical in these types of situations.

To make this simple model a little more interesting we decided to add an interruption to the auto avoid variation of the model. See figure 5.8. When the UAV Operator begins the first deconfliction procedure he is interrupted by another person, a dummy Actor added to the model just for this interruption. Instead of ignoring this interruption the UAV Operator attempts to communicate with this person while still performing the deconfliction procedure. This results in a visual channel conflict as the UAV Operator is watching both the UAS GUI and the person who interrupted him. In addition to this the UAV Operator is now in a high load/multi tasking state. The result is a workload value of 20 which is almost double the next highest workload value seen in the simulation. Although this spike in workload has yet to be validated it does match our intuition for the situation. We would also like to point out how well these results lend themselves to workload thresholds.

While we are mostly pleased with the workload results there are a few anomalies in the results which are misleading. The most commonly occuring anomaly is a drastic decrease in workload followed immediately be an equally drastic increase in workload. This tends to happen when an Actor is transitioning into the same state it just left. The spike happens because we are collecting metrics twice for each step in the delta clock. Once right before the active transitions are fired and once after they are fired.

5.3.3 Workload Analysis

We found the chart breakdown to be particularly useful in analyzing the results. In the resource input workload chart we get an idea of how many channels, layers, and memory objects the Actor is observing. By comparing this with the resource output workload we can see when the inputs occasioned a response from the Actor. For the most part these charts are well behaved and follow the workload patterns which we know about.

On the other hand the decision workload typically jumps all over the place. Currently we not sure exactly how important the decision workload is to overall workload metric. While we believe the increase in decision workload does in fact increase workload we are unsure how

to weight this value in comparison with the resource and temporal workload values. This is something we will explore in future work.

The last analysis we would like to make is the comparison of our workload metric to that of the adapted Wickens metric. The general flow of the two metrics is very similar, this is expected since everything in the Wickens metric also exists in our workload metric. What is more interesting is were the two metrics differ. Our workload metric is comprised of 9 different values each of which can range from 0 to 2+. This adds a fair amount of detail into the workload measurement which can be seen by comparing the two values for any of the different model variations. Where Wickens shows a flat line our metric shows a heart beat which tells us much more about what is causing the additional workload.

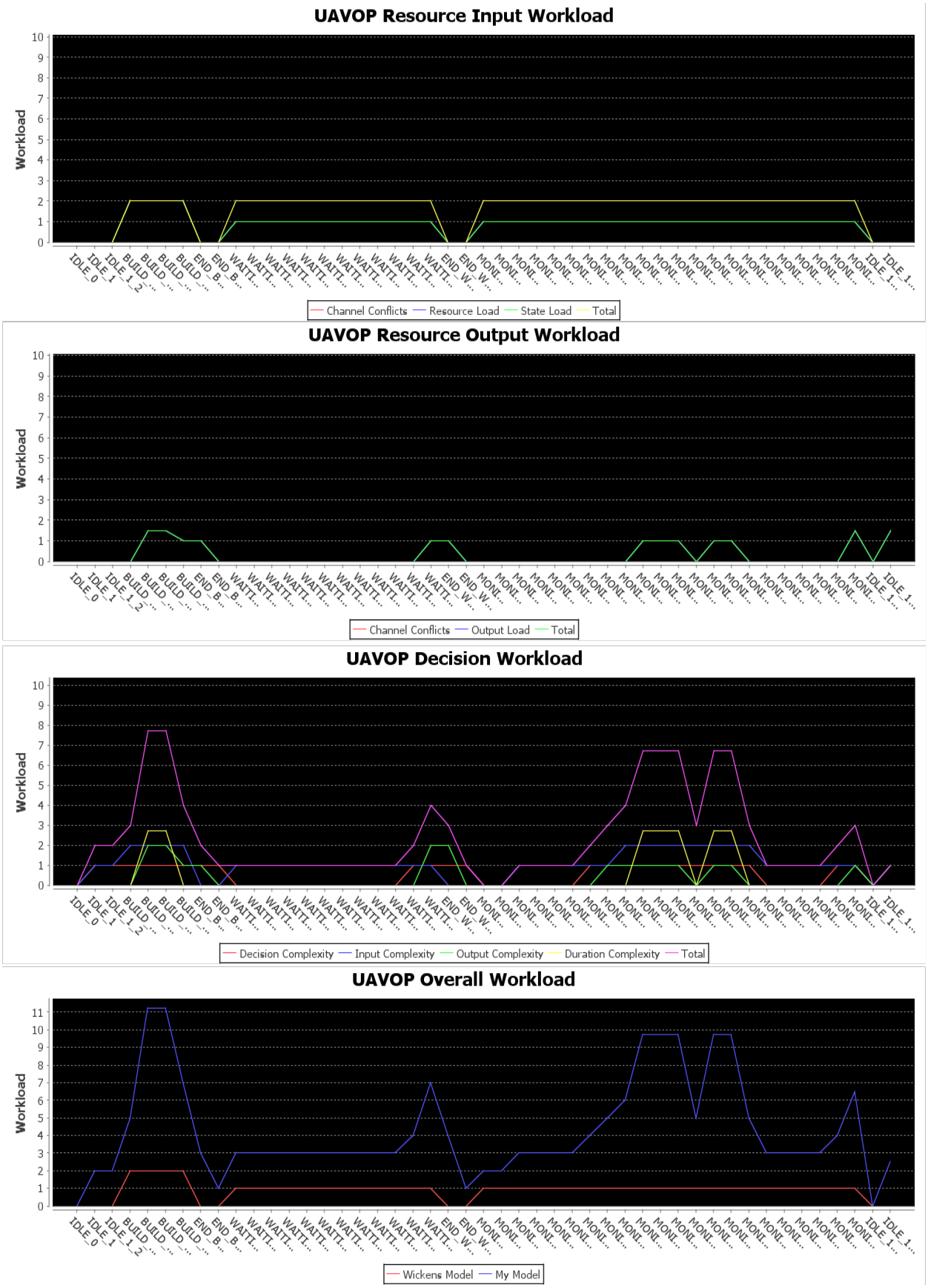

Figure 5.2: UAV Operator: Baseline

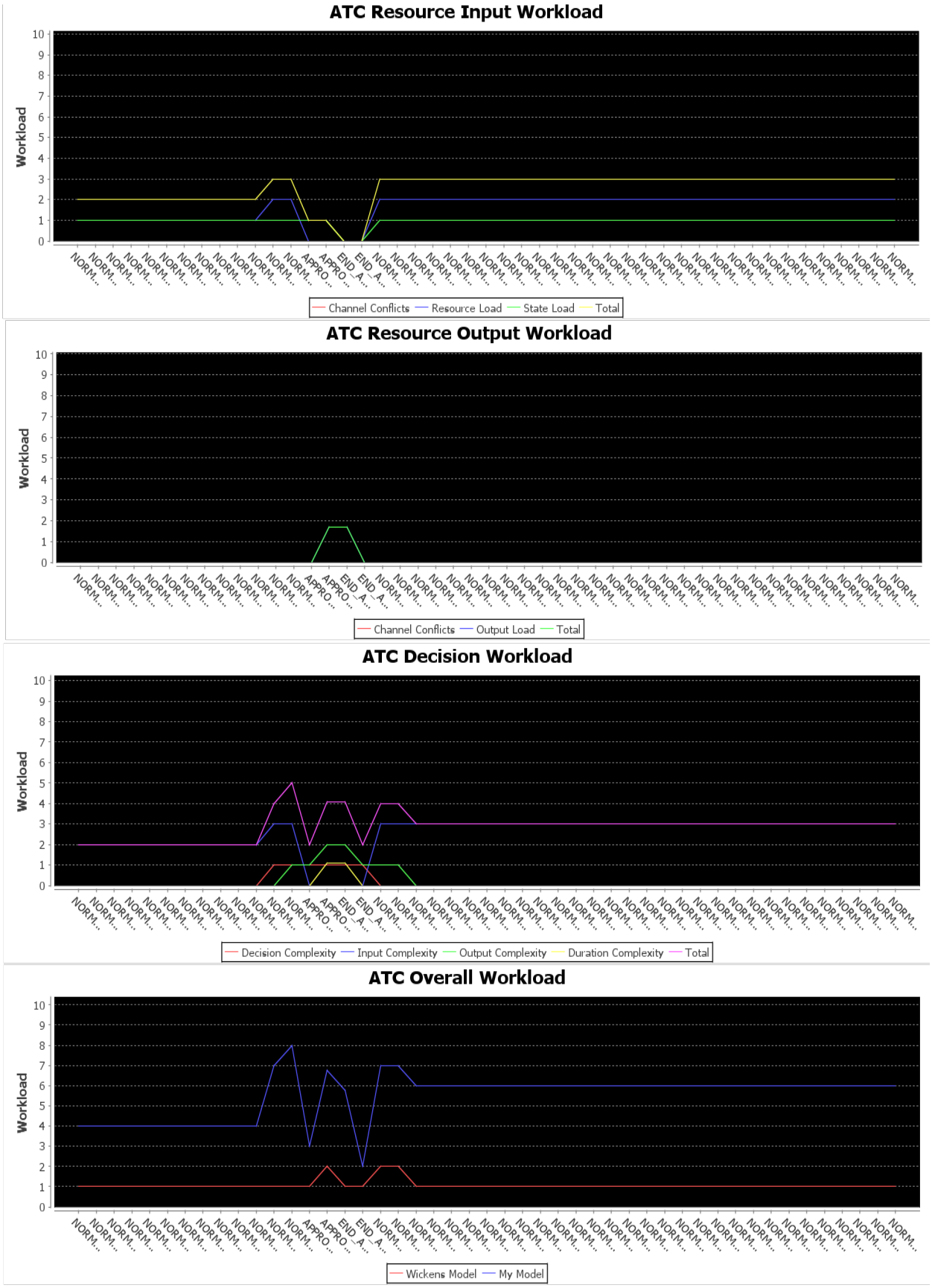

Figure 5.3: ATC: Baseline

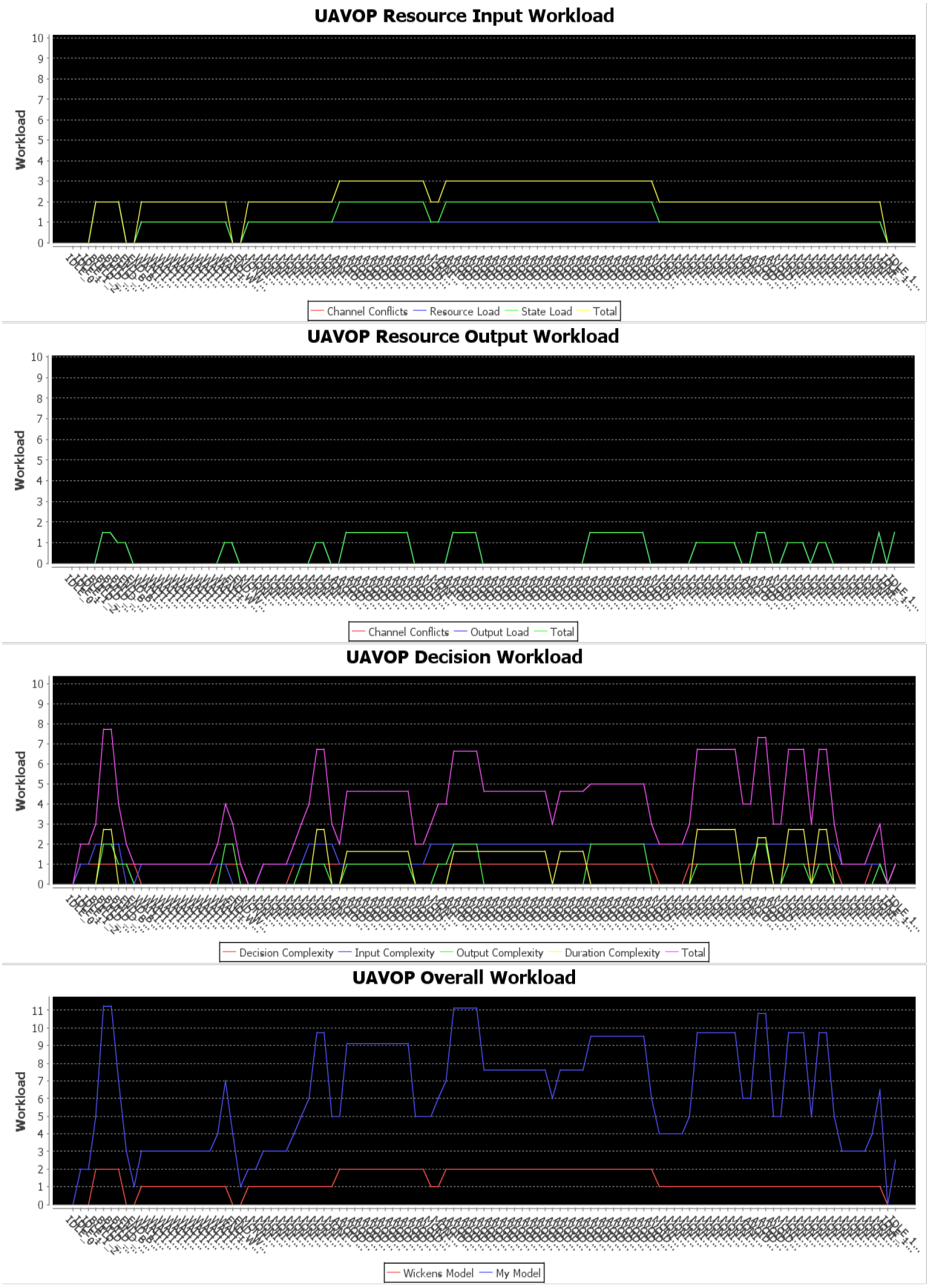

Figure 5.4: UAV Operator: Manually Avoided Emergency NOTAM

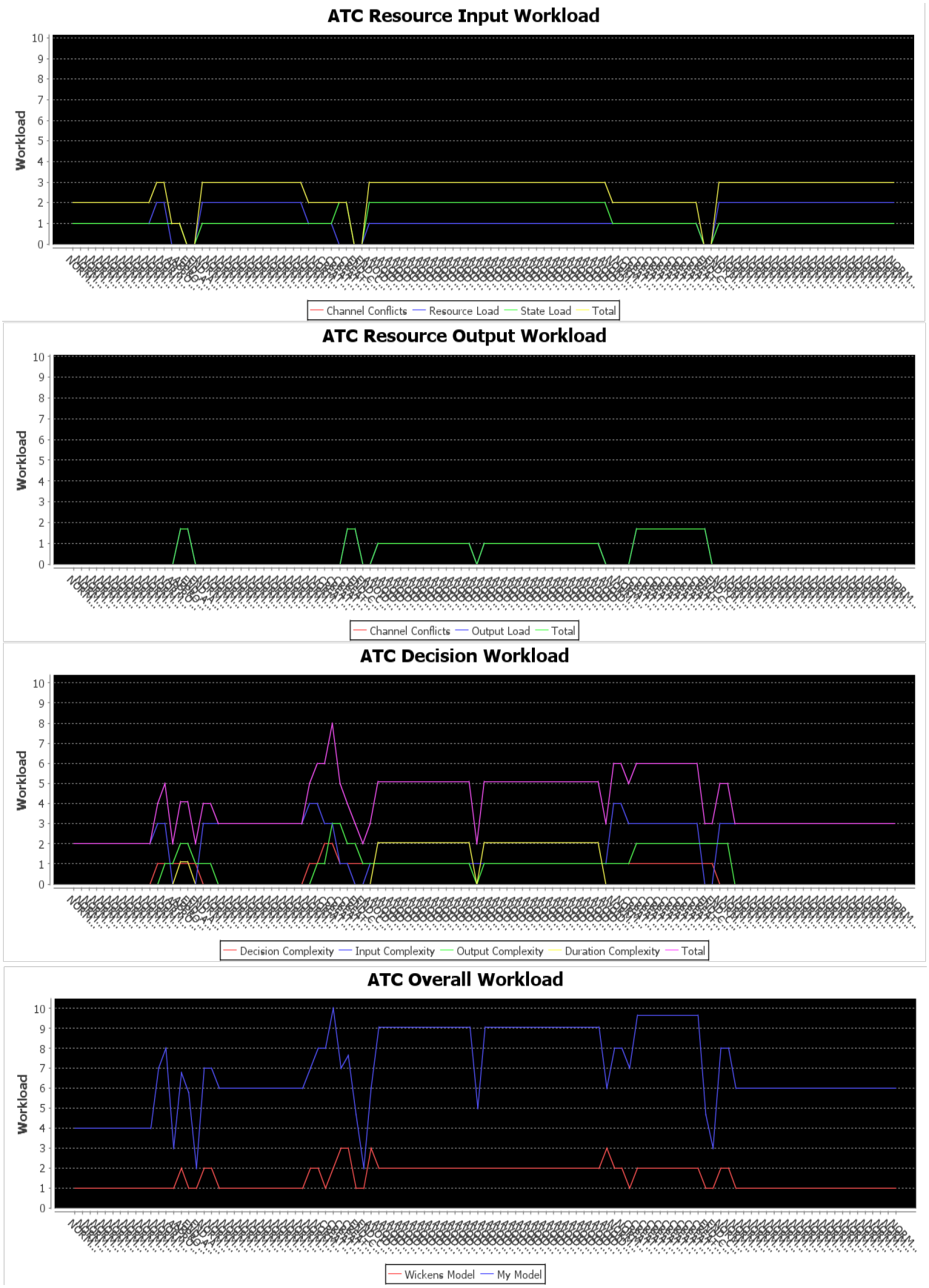

Figure 5.5: ATC: Manually Avoided Emergency NOTAM

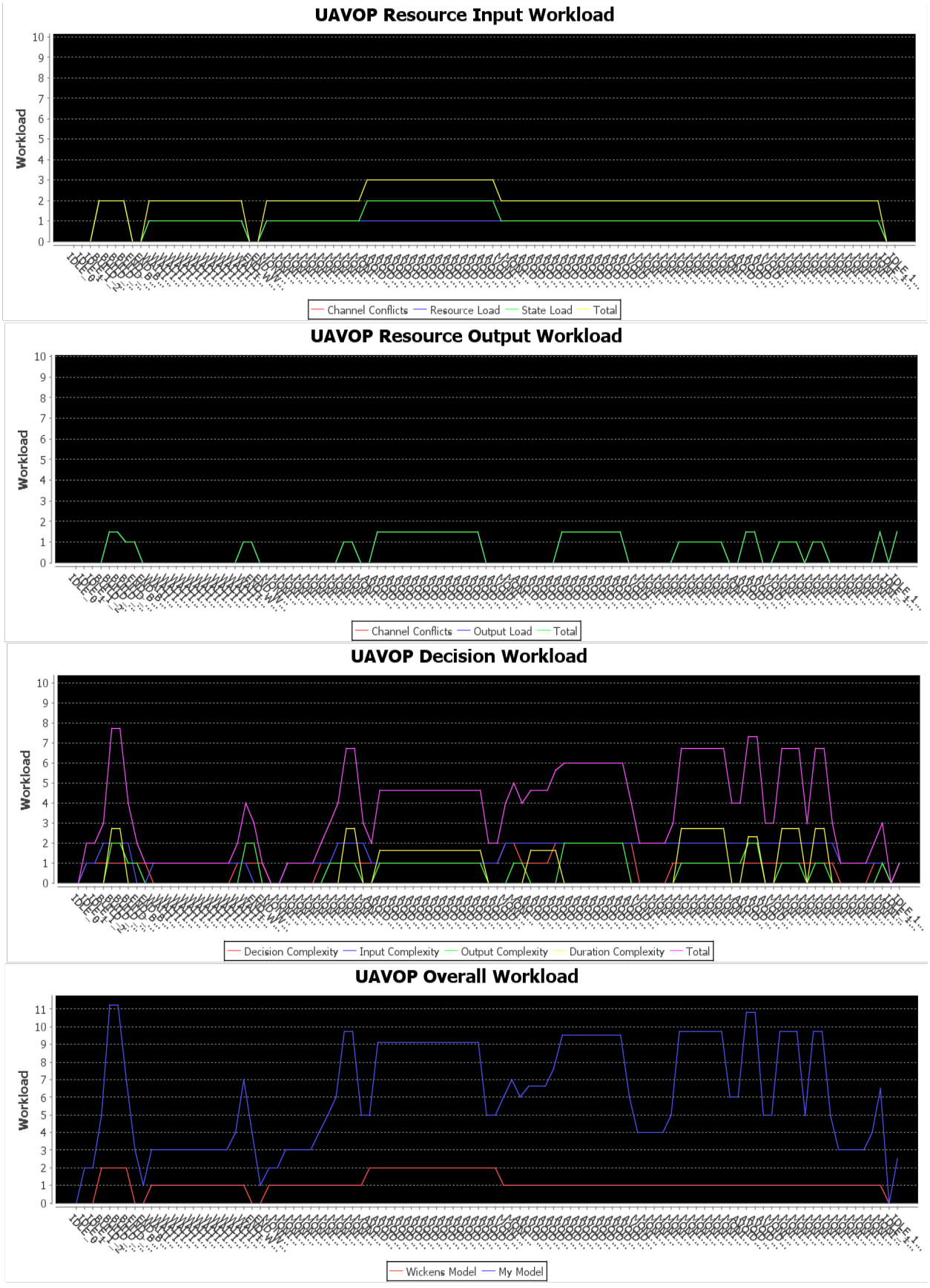

Figure 5.6: UAV Operator: Auto Avoid Emergency NOTAM

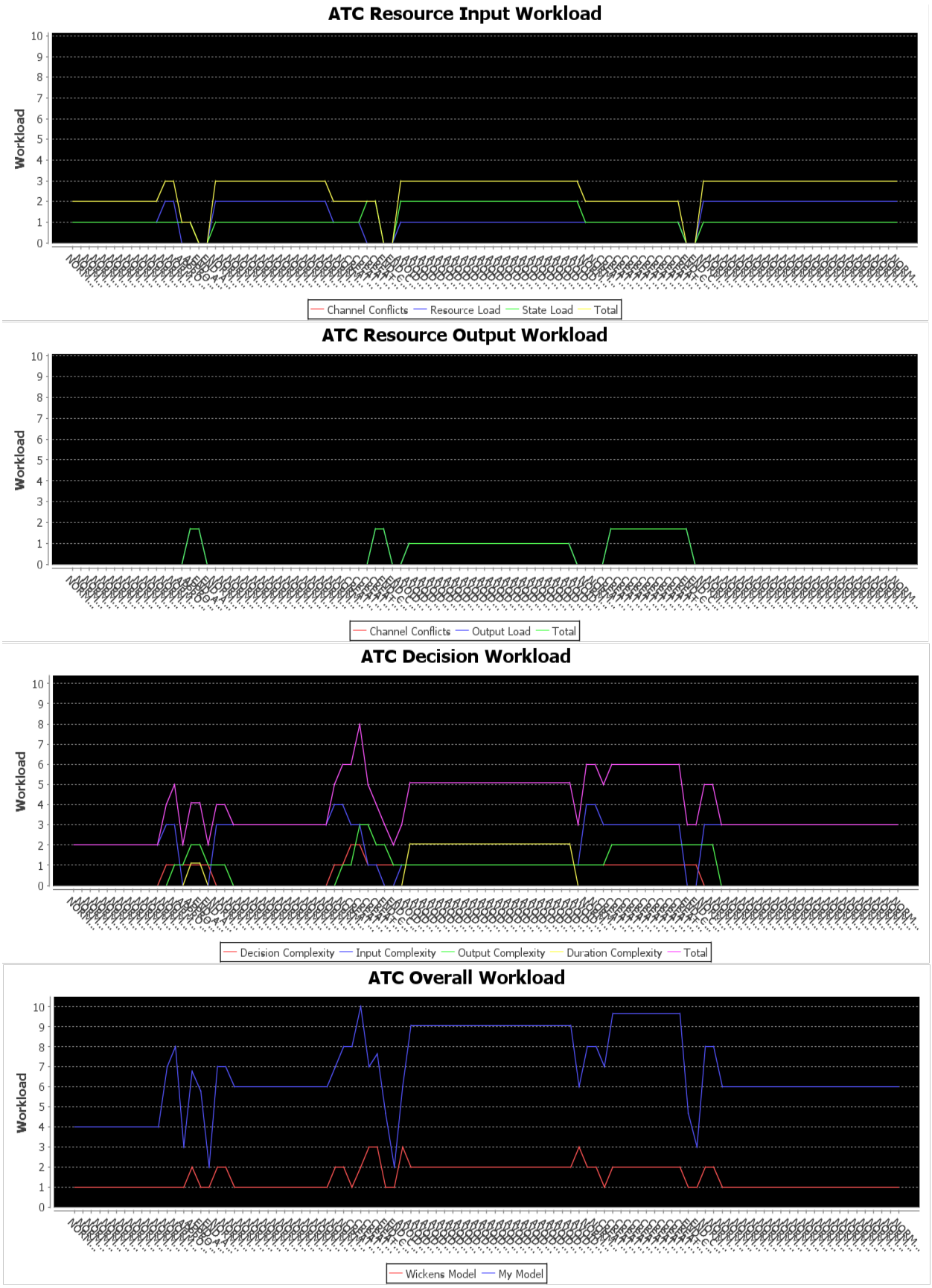

Figure 5.7: ATC: Auto Avoid Emergency NOTAM

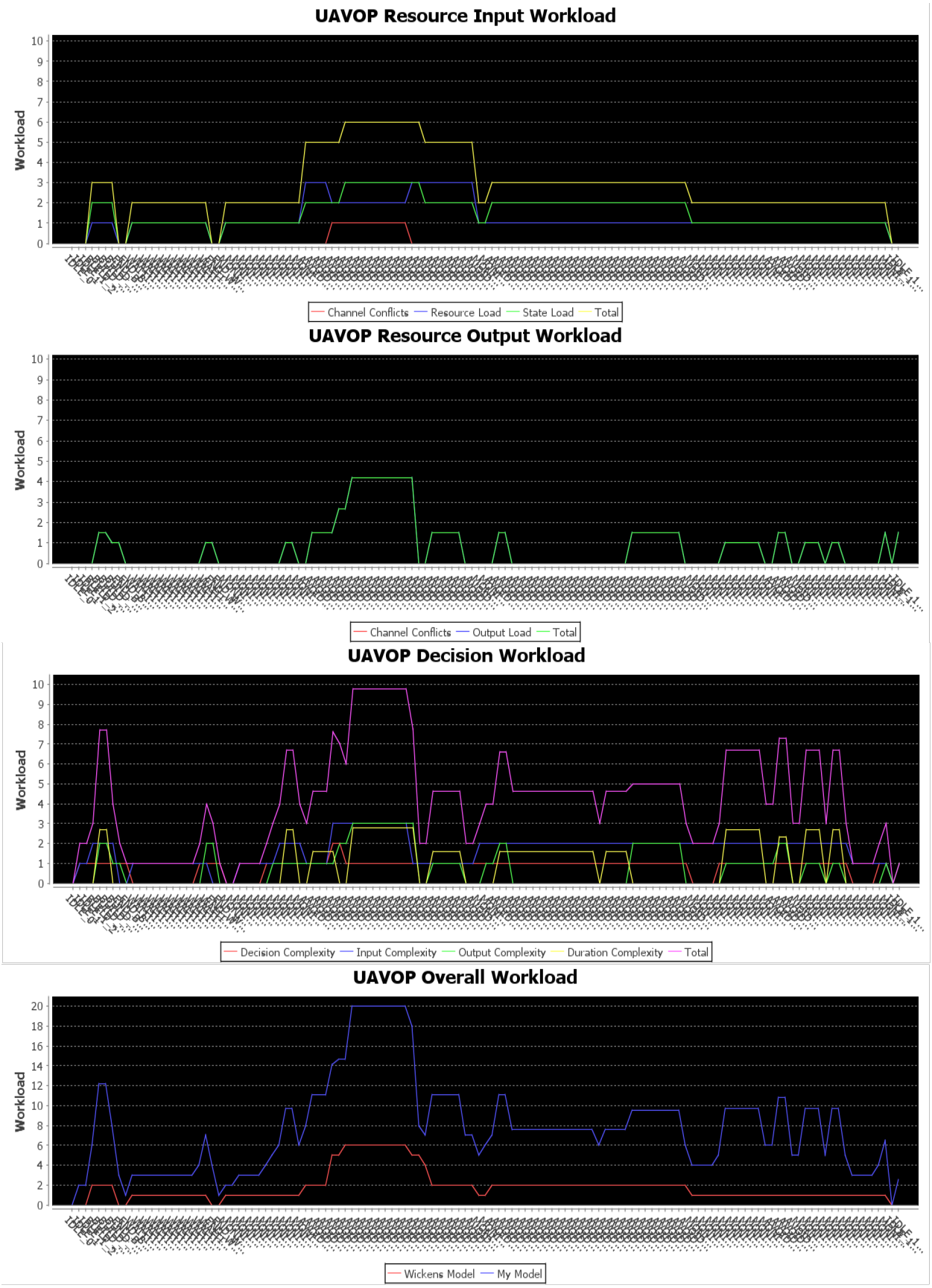

Figure 5.8: UAVOP: Auto Avoid Emergency NOTAM w/ Interuption

Chapter 6

Conclusions and Future Work

As part of this thesis work we have developed a new modeling framework based on the concept of DiRGs and DiTGs which is capable of modeling complex human machine systems as state machines. We have shown that our modeling framework is extremely flexible allowing each portion of a system to be modeled at varying levels of abstraction. This in turn allows the modeler to model unknown or changing portions of the system which we demonstrated in the WiSAR case study when we found flaws in the system which were previously undocumented and again in the UAS integration into the NAS when we demonstrated the effect of automating a portion of the UAS.

Additionally we have demonstrated our ability to explore all possible paths through a model by using JPF. With this tool we were able to find paths in WiSAR which lead to critical failure. By taking the ability to explore all paths through the model and coupling it with the flexibility of using Java as the base modeling language we have the basis of an extremely powerful modeling tool.

In our latest code re-factor we demonstrated the ability to implement an XML model parser to facilitate more efficient modeling. While the improvement the XML provided was not as great as we had hoped this does demonstrate the ability to adapt to different modeling languages. One area of future work is to implement conversion classes between our modeling language and some of the other more popular modeling languages such as Brahms or EOFM which would give us additional validation options.

Finally we have demonstrated our ability to extract workload metrics which are consistent with known high workload events and closely related to human workload theories such as Wickens multiple resource theory. We are encouraged by the predicted workload reduction seen in our second case study which segues into the future of this work.

From the beginning the goal of this work was to create a UAS for WiSAR which allowed a single human to perform all of the necessary tasks. Before this can be accomplished using this modeling framework the workload metrics need to be verified through sensitivity and user studies. We anticipate that machine learning algorithms will be able to take data from the sensitivity and user studies to produce a set of weightings which can be applied to the individual workload metrics. Once the workload metrics have settled the next step is to polish up the WiSAR model, add performance measurement Actors, and systematically modify the model to find a single human design which satisfies the performance measurements and maintains workload below a certain threshold. The next step after completing the WiSAR model is to create a generic UAS model which can then be used to explore the effects of novel UAS GUI designs.

In relation to the future of this modeling framework there are several avenues of future work which we feel would be the most beneficial. The first is improvements to the modeling process. We see this happening in several ways; new language structures which improve the ability to visualize the model, conversion classes which take advantage of existing language structures to generate models for our framework, or extending another modeling language such as Brahms or EOFM to work in a similar fashion.

Another avenue of future work is to establish a verification framework to facilitate the model checking process. One way we see this working is to allow the models to specify metric thresholds, Actor specific performance measurements, and dynamic portions of the model. The verification framework then uses machine learning to iteratively change the dynamic portions of the model to find an optimized design.

The last avenue of future work is to perform an in-depth comparison of this modeling framework to the existing modeling languages. For this modeling framework to be fully accepted into the human machine system modeling community it must be show how it differs from the other languages and what it offers which they do not.

Appendix A

UAS-enabled WiSAR Proposal

A.1 INTRODUCTION

Advances in Micro Unmanned Aerial Vehicle (mUAV) technology has pushed mUAVs into new frontiers. UAV Enabled Wilderness Search and Rescue (UE-WiSAR), one of these frontiers, has been a focus of the Human Centered Machine Intelligence (HCMI), Multiple Agent Intelligent Coordination and Control (MAGICC) and Computer Vision (CV) labs at Brigham Young University since 2005. In that time research has been conducted on human interaction with mUAVs, improving target detection by enhancing video taken from a mUAV, integrating mUAVs into a SAR environment, and improving the mUAVs chance of getting video footage of the target. Over the course of this research many of the ideas for improving UE-WiSAR results have been validated through simple experiments and user studies. Live Field Demo's with actual Search and Rescue personnel have also shown favourable results. These results represent important progress in Human Robot Interaction.

Although the research has proven successful, many of the tools developed for UE-WiSAR are unfit to share with a broader community. This proposal outlines the challenges faced by UE-WiSAR, the solutions discovered for overcoming these challenges and a plan for creating UE-WiSAR software that incorporates said solutions into a stable software package as part of an industrial thesis. As an industrial thesis the emphasis is not on new research but on delivering high quality software

A.2 PREVIOUS WORK

One goal of this project is to take past research and present it in a software application that encourages future researchers to use the software as a framework for continued research. Essentially what this means is that the entire purpose of developing the UE-WiSAR software is to make it available to future researchers and, more importantly, practicing searchers. The majority of the referenced work comes from a combined effort from the HCMI, MAGICC, and CV labs at BYU and focuses on solving the specific problems that arise when bringing a small UAV into the WiSAR arena. This proposal represents the realization of this goal [31].

A.3 WiSAR

A.3.1 The Problem

Wilderness Search and Rescue (WiSAR) is more prevalent today then in any other time in history. While Search and Rescue has been around since the beginning of mankind [40, p. 13], the improved communications and increased accessibility to wilderness areas have caused an increase need for WiSAR operations. Often times these operations have limited resources due to limited funds, remote locations, and dangerous conditions.

A.3.2 Concepts

To help understand how UE-WiSAR will be effective, some WiSAR concepts, defined by T.J. Setnicka [40, p. 35], will be used. The first concept is the four core elements of a WiSAR operation.

$$Locate \Rightarrow Reach \Rightarrow Stabilize \Rightarrow Evacuate$$

Figure A.1: Core SAR Elements

The second concept is the WiSAR plan and its components, specifically Strategy and Tactics. Strategy is the process of gathering information and making an accurate assessment

of the situation. Tactics are outlined solutions for specific situations that can be used as part of a Strategy.

A.3.3 Searching

Locating an individual in the wilderness can be a daunting task and typically represents the majority of time spent on an operation if the person is missing. SAR commanders develop Strategies for locating the person and use the Tactics available to them as part of those Strategies. Each Tactic applied to a search is based on availability, effectiveness, and cost. One Tactic that has proven it's effectiveness is aerial surveillance. One large drawback to this Tactic is the cost and availability. Until recently most aerial surveillance has been done with piloted aircraft. Advances in Unmanned Aerial Vehicles (UAV) have created new options for providing aerial surveillance, but high-end commercial UAVs are still incredibly expensive. This has prompted a deeper look into using mUAVs that are low-cost but by their very nature have a long list of obstacles that need to be overcome before they can be effective.

A.4 WHY DEVELOP UE-WiSAR

A.4.1 New Search Tactic

As mentioned earlier, most WiSAR operations are limited in the search Tactics they can use. Using mUAVs offers a new aerial surveillance Tactic. The mUAVs are small and relatively inexpensive; cost estimates are between $1,000 and $10,000 per mUAV [7, 23, 24]. These platforms represent a small fraction of the possible mUAVs that are capable of performing UE-WiSAR tasks. One model that is currently receiving attention is the multi-rotor mUAVs which can move at slow speeds and remain stationary if needed [10]. The relatively low cost of these mUAVs makes them attractive for aerial surveillance. mUAVs also reduce the risk to search personnel in the event of critical user/equipment failure. Also, future work will allow for multiple simultaneous mUAV surveillance [42].

While very capable, mUAVs are not fit for every situation. Few battery-powered mUAVs can stay aloft with the required camera-equipment for more than 90 min, many for less than half that time. This means that mUAVs are limited in their search range and effective search time. Also, the mUAV is extremely suseptible to high winds, rain, and snow which further limits its use.

While future advances may improve mUAV performance these limitations are important to understand about this search Tactic.

A.4.2 mUAV Surveillance Works

While far from perfect, mUAV surveillance has proven capable of successfully finding search targets in staged settings. User studies using NTSC video showed a probability of detection improvement of 43% over standard footage by using mosaicing on live video feeds [24]. A simplied field trial, close proximit with bright colors, conducted in May 2008 using prototype software was able to locate the simulated missing person in 40 minutes using a lawnmower search pattern [25]. After the field trial a qualitative analysis was performed. Field ready aspects from the analysis were the ability to quickly launch and fly the mUAV and the usefullness of mosaicing for detecting objects from the mUAV. The analysis also identified the need for improved user interfaces and communication. This need for improved user interfaces is an essential component of the UE-WiSAR proposal.

A.4.3 Community Outreach

Although the WiSAR project at BYU is winding down the potential research opportunities in this area have only increased. Improvements can be made in mUAV control, video enhancement, object detection, and more. The problem for those wishing to continue this research or to use the tools in practice is the lack of stable software containing the solutions that have been found. UE-WiSAR is that missing piece. One question that naturally follows deciding to build software is has it been done before. UE-WiSAR fits into four roles that

typically remain independent. The roles are Ground Control Station (GCS), Video Enhancer (VE), Search and Rescue (SAR), and Command and Control (CC). There are many open source software packages for performing tasks related to these roles, but there is no open source software that combines all of these roles into a single framework. UE-WiSAR will do just that making it ideal for continued research in the UE-WiSAR domain.

A.4.4 Thesis Statement

The WiSAR research and prototype software can be refactored into a cohesive and stable software package, UAV Enabled Wilderness Search and Rescue (UE-WiSAR). When finished UE-WiSAR will function as a new Search Tactic capable of performing aerial searches and integrating with real SAR operations. UE-WiSAR will also follow industry design standards with clear documentation making it an idea platform for future research and development in the SAR, UAV, and academic communities.

A.5 PROJECT GOALS

Overcoming the obstacles of using mUAV for WiSAR (Wilderness Search and Resue) operations has been a primary research focus at BYU since 2005. In that time many solutions have been found to overcome these obstacles. These solutions will be outlined in greater detail later in the proposal. In the process of discovering these solutions a plethera of software was created. This software was then used in user tests to determine it's effectiveness. Also, many studies where conducted in understanding SAR, human mUAV interaction, and mUAV-WiSAR integration. These software pieces along with the knowledge of how to use them for WiSAR is incredibly valuable. Unfortunately the software that has been created is disjointed and unstable, as is often the case with research software. Much of the software was written as prototypes for user studies and is not fit for distribution individually or as a whole. **The UE-WiSAR project will combine these prototypes into a cohesive**

and stable software package, designed as a new Tactic, for distribution to the SAR, mUAV, and research communities.

A.5.1 Everyone a Pilot

One goal of the UE-WiSAR project is to simplify mUAV piloting through the use of strategic automation and an intuitive user interface. Manual piloting of mUAVs is a highly cognitive task that takes years to master. Automation in the mUAV for auto take-off and landing, flight stabalization, and user-directed automation such as flight-path generation removes major hurdles for inexperienced pilots making mUAVs more accesible to SAR personnel [16].

A.5.2 Independent Search Operation

Wilderness Search and Rescue can potentially involve hundreds of personnel, many of which are volunteers. This can cause chaos and, unless organized properly, can have harmful effects on the search. To avoid this UE-WiSAR will focus on working as an independent technical search team. This implies an internal organization comprised of a Mission Manager, UAV Pilot, and Video Analysts. This structure is designed to minimize the personnel needed to operate the mUAV search tactic while maximizing its effectiveness. The goal is that this command structure will be able to fold into the main command heirarchy with minimum supervision and maximum effect, communicating the location of the missing person [24].

A.5.3 Improve UAV Video Quality

UE-WiSAR is not useful if human users cannot detect signs of the target. This fact stresses the importance of detecting targets that appear in captured footage. While UAV piloting has a great impact on the content and quality it is not enough. Low resolution, constant movement, and a small detection window make human target detection unacceptably low. To counteract these issues UE-WiSAR will use mosaicing and anomaly detection to improve

detection rates with the goal of generating high object detection rates with mUAV video [13, 34].

A.5.4 SAR Contribution

One of the most important phases of the WiSAR plan is the Critique [40]. The ability to recognize what went wrong and what went well is important for improving future operations. Analysis of multiple past WiSAR operations is also an effective way of gleening insights that can improve future operations. UE-WiSAR is organized to provide spatio-temporal information gathered during the search. This data can then be accessed for review or shared for further analysis. The goal is that the accessibility and organization of the UE-WiSAR data will improve the sharing of search data with SAR repositories such as the International Search & Rescue Incident Database.

A.5.5 Stable R&D Platform

UE-WiSAR has a great deal of untapped potential. The research that has been conducted at BYU represents the initial merging of exciting new technologies. This potential however is difficult to realize without a foundation to build upon. The UE-WiSAR software is that foundation. Without the need to create custom GCS, CC, SAR and VE interfaces, future researchers and developers can pursue new ideas that add-to or improve UE-WiSAR.

A.6 OBSTACLES

UE-WiSAR presents many problems to overcome. This project will deal specifically with those problems that occur in the Human-Robot Interaction, SAR, and Computer Vision domains. In analyzing these problems it is important to realize that these obstacles are built on the assumption that other problems have already been solved. mUAV automation, for example, will prevent undesired crashes during normal flight. To better describe these

obstacles and their relationship to UE-WiSAR each obstacle has been assigned to a solution category. Please see Figure A.2 on page 85.

A.6.1 UAV Piloting

High Cognitive Load. mUAVs operate under multiple degrees of freedom which can be disorienting for pilots. A high level of concentration is required to avoid becoming confused while piloting. This is somewhat mitigated by the low-level autonomy that this project builds upon [31]. However, a fair amount of cognitive load is still required. Path-planning, status monitoring and team communication are tasks that must still be addressed.

Hardware Variety. The variety of hardware that can be used for mUAVs is constantly increasing. For any system to be viable as a perpetual framework for UAV research it must have mechanisms in place to allow for the piloting of any number of unique mUAVs.

Limited Flight Time. Most battery-powered mUAVs are limited to a flight time of under 120 minutes, many are much less. This limitation makes flight planning much more difficult and introduces the potential for critical failure during a flight.

A.6.2 Object Detection

Human Perception Limitations. To be successful human users must be able to detect objects on screen. This implies that video presented to users will only be effective if it accounts for the limitations of the human eye [27]. The eye is only able to discern high detail with the cones located in the center. This means that to detect an object the user must be looking almost directly at the object while it moves across the screen. Another limitation is that the eye has difficulty detecting small changes in intensity, an effect that gets worse as brightness goes down [21].

Low Resolution. The video resolution is a result of current hardware. The mUAVs at this time broadcast NTSC 640x480 resolution. This resolution makes it extremely difficult to locate small objects which may be represented by only a few pixels [25].

Figure A.2: UE-WiSAR Obstacles

Small Detection Window. When using a fixed wing mUAV the video captured is in constant motion. This motion means that a potential target will only remain visible for a short time. Minimum mUAV flight speeds only slightly improve the detection window and cannot fully correct this problem [25].

Jittery Video. For stable flight, the mUAV is constantly correcting course through small adjustments. These adjustments occasionally have the undesired effect of making video appear jittery. This problem is magnified in adverse weather conditions [25].

Object Localization. Assuming an object has been detected in surveillance video the next step in a WiSAR operation is to send ground searchers to the object. In a May 2008 field trial, video analysts where unable to accurately communicate the location of an object and had to observe the searchers from the mUAV to give them directions relative to the object [25]. This example illustrates a different challenge which is determining the exact location of a detected object.

False Positive Detections. Due to the cost associated with missed detection, a human life, a high false alarm rate is considered tolerable. If the false alarm rate is too high, however, it degrades the practicality of the tactic. On top of that each false alarm requires effort that may bog down the search or leach resources from other tactics [24].

A.6.3 WiSAR Integration

The next group of obstacles fall under the WiSAR Integration category and represent the challenges from introducing mUAVs into a WiSAR operation. A cognitive task analysis was performed to provide insights for such an integration [8]. The goal directed task analysis and work domain analysis from this effort communicate how complex a WiSAR operation is. To integrate with such a complex endeavor a few obstacles must be overcome.

Multi-Person Team. WiSAR operations are made up of potentially hundreds of people. Additionally, the mUAV requires its own team. Not only must the mUAV team

operate as a team but it must also interact with the overall operation. These multi-person, multi-team environments often generate role confusion, conflict, and inefficiency [24].

Effective Communication. No search can be effective without the relevant data. A typical WiSAR operation uses a hierarchical command structure. The mUAV team must be able to fold into the hierarchy such that it receives relevant search data. The mUAV team must then communicate internally as individual roles are performed. Important information must then be communicated back into the parent command structure.

Representing, Organizing, and Saving Search Data. The data provided to the mUAV team must be interpreted so that it can be understood by the mUAV. Additionally the data provided by the mUAV must then be interpreted so it can be understood by users and commands further up the chain. This implies an internal data organization associated with the mUAV. This organization must facilitate the storing and sharing of said data.

Duplication of Work. This mostly applies to search area coverage. The mUAV team must be able to track what it has searched and how well it was searched. Without this information search planning will be inaccurate which could have fatal consequences.

A.6.4 UAV Piloting & WiSAR Integration

Where to Search. During a WiSAR operation the probability of area (POA) [29] is constantly changing as new information is acquired. For a UAV to integrate into a WiSAR operation it must have the ability to interpret this information, act on information, and contribute information. If information is lacking then it must be able to generate information to act upon. A target can only be spotted by the mUAV if it shows up on the video.

High Human to UAV Ratios. This obstacle is based on practicality. It is not practical to require a large team for a single mUAV. With the variety of tasks that emerge when introducing a mUAV to WiSAR it becomes quite challenging to keep this ratio down. A study by Cooper [15, 25] speculates that it may be possible for a single human to simultaneously navigate an area while localizing objects. His conclusion outlines several requirements he

feels must be met before this can become reality. This becomes even more challenging when considering the Mision Manager role as well.

A.6.5 UAV Piloting & Object Detection

Maintaining Height Above Ground (HAG) [7]. This represents one of the major crash risks associated with user error. In an early test flight the pilot placed two way points of similar HAG a fair distance apart. As the mUAV flew between the way points it crashed into a tall ridge that separated the waypoints. The pilot wasn't aware that his flight path went through the ridge. This example illustrates one reason for maintaining HAG; another reason is related to Object Detection. Goodrich et al. state that the minimal resolution of an image for detecting a human form is 5cm per pixel [24]. This means that an image can cover an area no wider than 32m and 24m tall. They go on to suggest that the maximum HAG be between 60m to 100m.

Disorienting Video. Disorienting video is produced when the mUAV is performing manuevers which change the camera field of view and cause the user to feel disoriented [34]. Even small manuevers can be disorienting when occurring in succession. This is a challenge because the UAV cannot obtain the needed surveillance without turning. Also, the light weight of the mUAV causes it to be susceptible to wind which can cause excess manuevering.

A.6.6 Universal Obstacles

Non-Technical Users. For UE-WiSAR to be practical it must strive to be accessible to the greatest number of users. With this said UE-WiSAR is a technical operation and cannot be divorced completely from its technical aspects. UE-WiSAR requires some knowledge of mUAVs, networking, and radio transmission. The real obstacle is segregating and limiting the technical experience required for UE-WiSAR into as few roles as possible.

Situation Awareness. This represents the user's ability to maintain awareness of the bigger picture while performing tasks. This is broken into two categories. The first

category relates to the mission. The main goal of any UE-WiSAR operation is to find the missing person. If the tasks performed by mUAV team members require a cognitive load that is too high team members will be unable to meet their respective responsibilities which may have tragic consequences to the search. The second category relates to autonomy. Autonomy is a central concept to UE-WiSAR. As autonomy increases certain negative attributes can emerge [7], including:

- Reduced situational awareness
- Difficulty in supervising autonomy
- Increased interaction time
- Increased demands on the human and autonomy

As autonomy decreases the following negative attributes can emerge [17]:

- High cognitive load on operator
- Steep learning curve
- Increase pilot:UAV ratio

The balancing of this dynamic relationship between the autonomous and operator-controlled elements of UE-WiSAR is important because it is directly linked to the usability of the solution.

A.7 SOLUTIONS

The solutions to the above mentioned obstacles are organized into three categories. See Figure A.2 on page 85. This organization is preferred because many of the solutions discussed here solve problems associated with multiple obstacles.

As mentioned earlier, these solutions already exist in different states. The subsequent paragraphs will expand on the work required to add the solution to UE-WiSAR.

Figure A.3: UAV Piloting Solutions

A.7.1 UAV Piloting

This solution category focuses on overcoming obstacles associated with piloting the mUAV. See Figure A.3 on page 90.

Automated Flight Paths (AFP). The first obstacle this addresses is the high cognitive load on the pilot. Detailed flight paths can be generated in a matter of moments with minimal user input. These flight paths can also be adjusted based on the limited flight time. Lin has created an algorithm that when given a probability distribution map, start point, end point and flight time will generate a flight path that maximizes the mUAV cameras coverage of the probability distribution [30]. Another type of flight path is the Generalized Contour Search [24]. This flight path requires a gimbaled camera but also generates optimal search patterns for certain conditions. Automatically generated flight paths can also attempt to limit the number of turns made by the UAV to minimize the amount of disorienting video captured during a flight. Lastly, these flight paths can use HAG information to maintain the optimal HAG for object detection while also avoiding obstacles such as ridges or cliffs.

Pre-Load Terrain Data. Terrain data provided by a number of sources can be downloaded over the internet prior to the search. This data is critical for implementing effective AFPs and POA distributions.

Modular Hardware Interface. To overcome the hardware variety obstacle UE-WiSAR will use piloting interfaces that must be implemented for specific technologies. While the initial UE-WiSAR release will have limited hardware support, namely Procerus and Mikrocopter, more support can be added through implementing a single interface for the specific technology. This approach minimizes the work and knowledge required to adapt the software to new hardware.

UE-WiSAR Pilot UI. This user interface has two main requirements [31]. First is assigning tasks to the mUAV. This implies an ability to make the mUAV an effective part of the search by having the mUAV capture high quality video footage of regions specified

by the incident commander. It does not imply deep understanding of mUAV piloting, flight path automation, or other technical details associated with piloting a mUAV [16].

The second requirement is the ability to monitor the health of the mUAV. This means the UI must communicate the exact position and status of the mUAV at all times and alert the pilot when user input is required. Because this UI is the main focus point for the pilot, it is a primary concern for loss of situation awareness. To avoid this the UI must be able to dynamically adjust the amount of automation needed as dictated by the situation.

A.7.2 Object Detection

This section focuses on detecting objects in video captured by the mUAV. See Figure A.4 on page 93.

Temporally Localized Mosaic [14, 34]. This process allows the user to view the current frame in relation to a history of previous frames. An object that appeared in a single frame may now remain visible for multiple frames. Morse et al. conducted user studies to analyze the effectiveness of this approach. Those studies found a 43% improvement in hit probability when using mosaiced views versus non-mosaiced views. While there was an increase in false positive detections this increase is considered inconsequential along side the improvement to hit probability.

Spectral Anomaly Detection [13, 37]. In typical WiSAR video the majority of colors are varying shades of grey, brown, and green. This process looks for objects that are "out-of-place". This autonomous detection will not replace user detection, instead it aids user detection by suggesting objects to the user for closer inspection.

GPS Frame Referencing [33]. This process uses geometry to map pixels to gps coordinates. The algorithm uses the GPS coordinates of the mUAV, mUAV position, terrain data, and camera specifications to determine the relation of the point to the UAV. While the process is simple, it suffers from the limited precision of mUAV sensors and may provide highly inaccurate locations.

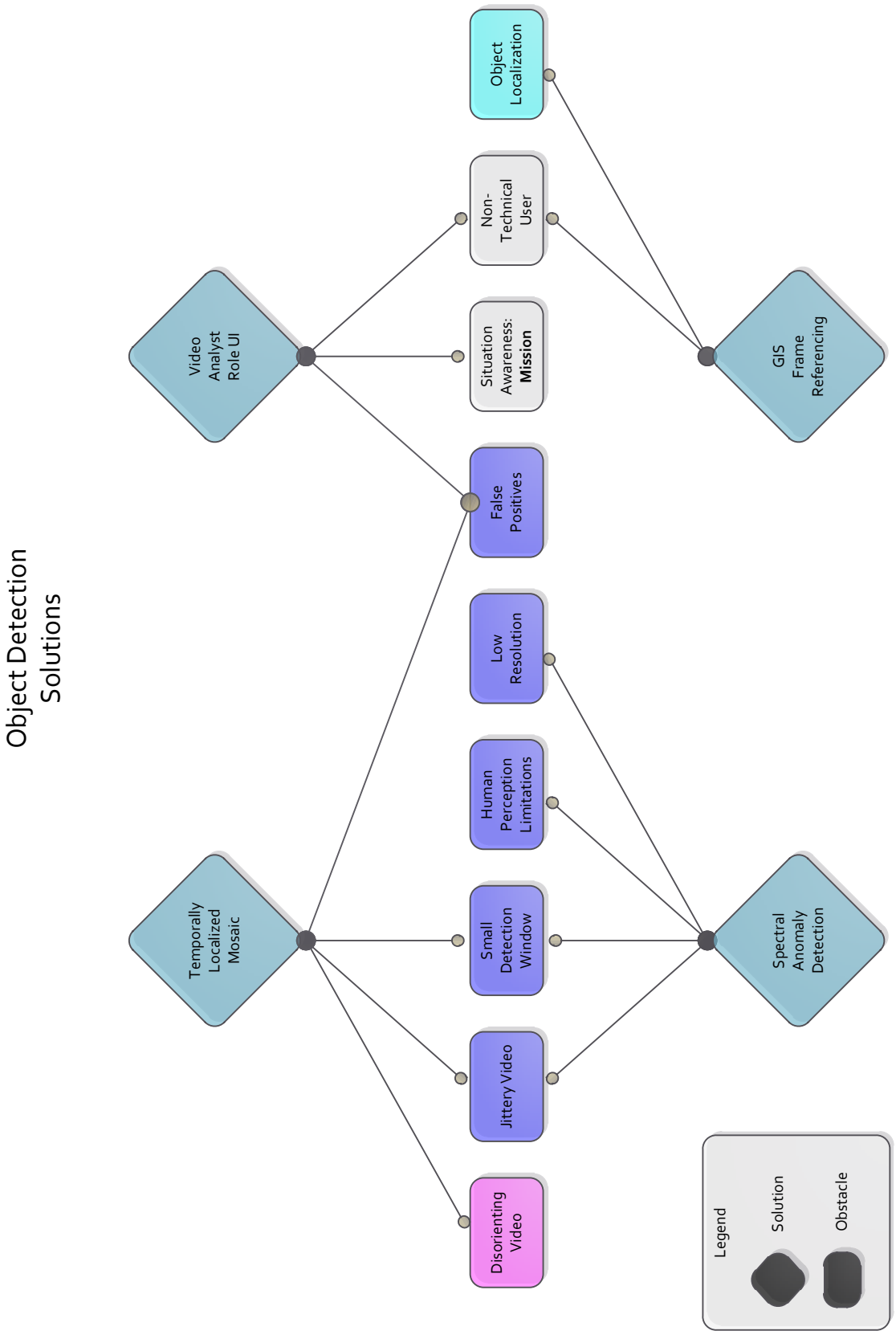

Figure A.4: UAV Piloting Solutions

Video Analyst UI [31]. This UI is meant for users operating under the Video Analyst Role. Its purpose is to help Video Analysts detect objects seen by the mUAV. There are three main requirements associated with accomplishing this purpose. The first is to provide video to the user. As a search progresses Video Analysts may need to analyze live video, video of specific areas, or video from certain times. The UI must make it easy for analysts to find the video that needs to be analyzed. The second requirement is to aid in object detection. The UI must be able to enhance the video as directed by the user. This includes the above mentioned solutions along with other simple enhancements such as brightness, contrast, and rate of playback. The last requirement is that the UI allow the analyst to communicate with the mUAV team. As an analyst works they must be able to communicate findings to the mUAV team.

A.7.3 WiSAR Integration

WiSAR Integration focuses on introducing a mUAV to a WiSAR operation. See Figure A.5 on page 95.

Team Roles [8, 23, 24]. UE-WiSAR will use a hierarchical command structure. The top level role is the *Mission Manager* (MM). The MM is responsible for defining the search in UE-WiSAR. The MM is also responsible for directing the other roles associated with a mUAV search.

The next role is *UAV Pilot*. The Pilot is responsible for capturing video with the mUAV as directed by the MM and communicating important information about the status of the mUAV to the MM.

The last role is *Video Analyst*. The Analyst is responsible for detecting objects in the captured video. The Analyst communicates any findings to the MM.

This role breakdown enables a mUAV team of two or more people to operate effectively through a clean breakdown of work and established communication channels.

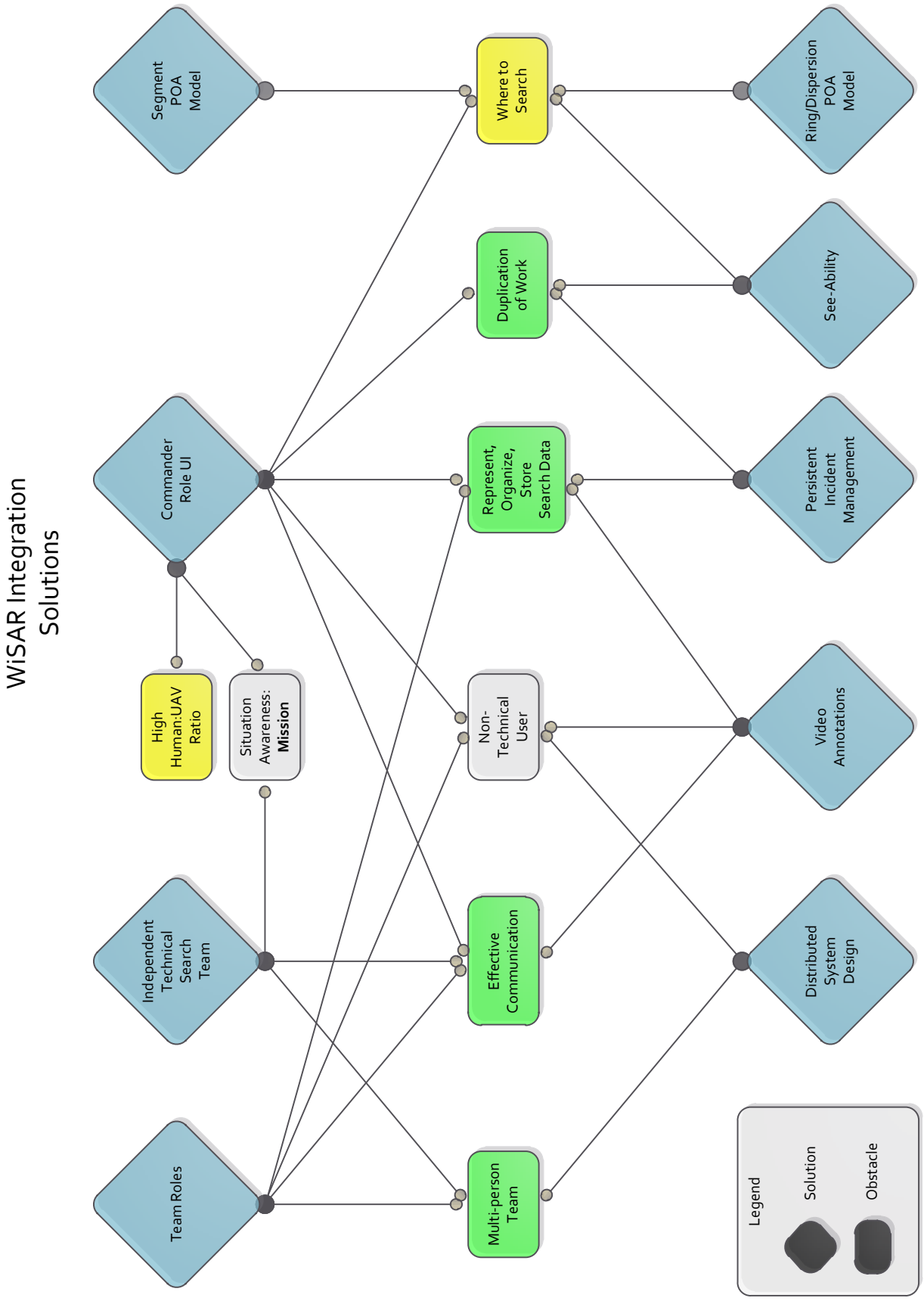

Figure A.5: WiSAR Integration

Independent Technical Search Team [8]. UE-WiSAR represents a single search tactic for locating missing persons. This implies that UE-WiSAR will be used as part of a larger WiSAR operation. This is facilitated through the MM role. The MM is responsible for obtaining the missing person data and entering the data into UE-WiSAR. The MM is then responsible for constructing a mUAV search plan as directed from the chain of command. As the mUAV team follows the search plan, everything is reported back to the MM who then relays relevant information back to the main chain of command.

Command & Control UI This UI is meant for users operating under the Mission Manager Role. Its purpose is to facilitate the MM responsibilities. There are three main requirements for accomplishing this purpose. The first is the ability to build a missing person profile. Information such as the last known location, clothing color, destination, starting point, etc. is essential for the automation and the mUAV team. The UI must allow the MM to enter this data but not overload the MM with tedious data entry. The next requirement is the ability to define a search plan. The UI must show relevant search data to the MM and allow the MM to define a search plan which can then be carried out by the Pilot. The last requirement is the ability to receive communications from mUAV team members. The UI must alert the IC of detected objects, mUAV status, search progress. The UI must also communicate this information in a way that it can be presented to the main chain of command. Additionally the UI must be intuitive, clean, and simple.

Distributed System Design A client-server architecture has been chosen for UE-WiSAR for several reasons. The first reason is the computational load required to enhance video received by the mUAV. The resources needed are much greater than those of a typical desktop computer and require a powerful server. Once the video has been enhanced, however, it can be distributed to clients at little cost. Another reason for this architecture is the unknown team size. Collected video must be analyzed by the human-eye. This architecture allows for any number of video analysts to work symultaneously, assuming there is enough bandwidth. Another reason for this architecture is to simplify the software. Instead of

building an all-in-one software solution, a server and multiple independent client applications can be written. Clients can then choose which application to run according to their roles which in turn makes the client roles less confusing.

Video Annotations These are the primary means of communication for the Analysts. When an Analyst detects an object they can click on the object and create an annotation. The annotation will then carry information about that object such as location, time of discovery, place in video, analyst comments, priority, etc. Once an annotation is created it instantly becomes available to other team members, in the case of the IC, alerting them of the annotation. The annotation can then be modified as needed depending if it was a false alarm or a real sign. If an annotation is a real sign it can be converted to a format acceptable to the external chain of command.

Persistent Incident Management Essentially this refers to a relational data model specifically designed for UE-WiSAR. The root node of this data model will be an incident, all other data will be linked to an incident. The model will hold missing person data, search plans, flight plans, videos, annotations, etc. Storing data in the model will reduce work duplication and make data more accessible. The model will also simplify persistent storage as xml files or a sql database. In concert with the project goals this data model can be shared with other search groups and search databases to help improve WiSAR.

See-Ability [20] This is a method of determining how well an area has been searched by the mUAV. The algorithm uses the position of the camera in relation to the ground to determine the quality of the view. This can later be used to find how many times a location has been viewed, how many unique angles has it been viewed from, and what is the overall quality of the viewings. This can be a great help to the MM and Pilot in avoiding over-viewed areas, narrowing search parameters, and maintaining situation awareness.

POA Models [29] Because UE-WiSAR must be able to act independently, two probability of area models will be added to the CC interface. The first is the Segment Model. This model allows the IC to strategically breakdown a large search region into smaller search

regions. Efforts can then be focused on high priority regions as directed by the IC. The second model is the Ring/Dispersion Model. This model uses the last known location along with the intended destination to create an ever expanding search corridor with eminating rings that occur at specific distances. The highest POA occurs inside the corridor and inside the first ring, then second ring, etc.

A.8 DELIVERABLES

As an industrial thesis a major goal is to produce high quality software that is on par with industry standards. Quality as it relates to the project goals is the users ability to perform tasks within their roles and to allow future developers to understand UE-WiSAR well enough to modify/enhance the software as needed. To following descriptions of work will be used to validate that the project goals have been met and that the project is a success.

Therefore, the first deliverable will be a detailed list of requirements organized into groups and prioritized. Glass's law [**?** , p. 16] states that "Requirement deficiencies are the prime source of project failures." The list of requirements is meant to be an ideal goal. Not every requirment will be acheived, and some may change, instead it is meant to be a road map. By clarifying the project requirements early the student and committee members can make informed decisions, avoid mistakes, and measure progress. It is also expected that design specifications will improve the design and make prototyping more effective.

The second deliverable will be detailed design documents specifying the architecture, data model, workflows, dataflows, protocols, user manuals and language/framework considerations. These documents will make up a collection of text, UML, and other documents. Their purpose is to clarify how the system works and how it accomplishes project goals. Boehm's first law [**?** , p. 17] states "Errors are most frequent during the requirements and design activities and are the more expensive the later they are removed." This step of the project will help to reduce bugs and development time by allowing for peer review and prototyping before major work is done. Many of these documents are living documents and will change

as the project progresses, some may not exist until after coding has been completed. It is expected that requirements and design will take up to one third of the total project time.

The third deliverable will be the actual UAV Enabled Wilderness Search and Rescue code. The code will be well documented, follow consistent coding practices, and compile on designated operating systems. This represents the majority of work done on the project.

The fourth deliverable will be a demonstration of the software in a live environment with a real mUAV and controlled targets. This demostration will show that core features exist, the software is stable, and users are able to perform tasks by following written instructions. Core features are the set of features decided on by the student and committee that the software must have to be considered functional. The users will be individuals, preferably familiar with WiSAR, unfamiliar with operating the UE-WiSAR software. At least one user per role will be involved. An experienced mUAV pilot will also be involved to reduce risk to equipment. Basic tasks will be assigned that require use of the core features. This will not be an exercise in validating prior research.

A.9 DELIMITATIONS

Due to the nature of software development and the size of this project there will be a large list of things to do that can be done in a reasonable amount of time. The goal of this project is not to implement the maximum amount of features, instead the goal is to create a stable foundation for others to implement the features they choose. This is particularly relevent in regards to user interfaces.

This project will not create the ideal user interfaces as described in the solution section as those are subjective and become much too time consuming. Instead the project will focus on functionality and creating basic user interfaces that can be easily replaced by future developers.

The project also doesn't account for the High Stress that is associated with SAR operations. It is known that High Stress has a negative impact on an individuals cognitive load capacity, however, it is too complicated to include in this proposal.

There are several learning curves that are associated with different aspects of this project. Because learning curves are unavoidable and vary with the individual it is enough for this proposal that the software is targeted at as large a user group as possible.

There are too many unknowns to accurately predict the amount of time a project this size will take to complete. The focus will be on the deliverables and working closely with advisors during the development process to adjust the requirements to fit with time constraints.

A.10 CONCLUSION

UE-WiSAR represents an opportunity for the research done at BYU to serve a greater community. As Thomas Edison once said "The value of an idea lies in the using of it." UE-WiSAR will not only provide a new search Tactic for SAR operations, it also provides a framework for mUAV enthusiasts and researchers to build upon for continued research in the field. What now exists as a collection of interesting ideas will become the do-it-yourself manual for performing aerial surveillance using mUAVs.

Unlike many open source solutions, the focus of creating software at current industry standards makes UE-WiSAR even more valuable. With good design and documentation the project is much more likely to take hold in the open source community further increasing its ability to serve those communities it is meant to serve.

Appendix B

Java Classes

Appendix C

XML Model Sample

```
      6pt
<team name="Basic UAS in NAS">
  <channels>
    <channel name="DATA_UAV_UAS" type="DATA" source="UAV" target="
   UAS" />
    <channel name="VISUAL_UAV_UAVOP" type="VISUAL" source="UAV"
   target="UAVOP" />
    <channel name="VISUAL_UAS_UAVOP" type="VISUAL" source="UAS"
   target="UAVOP" />
    <channel name="AUDIO_UAS_UAVOP" type="AUDIO" source="UAS"
   target="UAVOP" />
    <channel name="TACTILE_UAVOP_UAS" type="TACTILE" source="UAVOP
   " target="UAS" />
    <channel name="DATA_UAS_UAV" type="DATA" source="UAS" target="
   UAV" />
    <channel name="DATA_UAS_FAA" type="DATA" source="UAS" target="
   FAA" />
    <channel name="DATA_FAA_UAS" type="DATA" source="FAA" target="
   UAS" />
    <channel name="VISUAL_FAA_ATC" type="VISUAL" source="FAA"
   target="ATC" />
    <channel name="TACTILE_ATC_FAA" type="TACTILE" source="ATC"
   target="FAA" />

    <channel name="DATA_ATC_STATE" type="DATA" source="ATC" target
   ="ATC" />

    <!-- Event Channels -->
    <channel name="NEW_MISSION" type="EVENT" source="" target="
   UAVOP" dataType="String" />
    <channel name="RADAR_COLLISION" type="EVENT" source="" target=
   "UAVOP" dataType="String" />
    <channel name="FAA_COLLISION" type="EVENT" source="" target="
   FAA" dataType="String" />
    <channel name="NEW_NOTAM" type="EVENT" source="" target="ATC"
   dataType="String" />
    <channel name="END_UAV_COLLISION" type="EVENT" source=""
   target="FAA" dataType="String" />
  </channels>

  <!-- Model Actors -->
```

```
  <actors>
    <!-- ACTOR UAVOP
    ////////////////////////////////////////////////////////////-->
    <actor name="UAVOP" showMetrics="true">
      <inputchannels>
        <channel>VISUAL_UAS_UAVOP</channel>
        <channel>AUDIO_UAS_UAVOP</channel>
        <channel>VISUAL_UAV_UAVOP</channel>
        <channel>NEW_MISSION</channel>
      </inputchannels>
      <outputchannels>
        <channel>TACTILE_UAVOP_UAS</channel>
      </outputchannels>

      <!-- Define actor memory -->
      <memory name="FAA_APPROVED" dataType="Boolean">false</memory
    >
      <memory name="FLIGHT_PLAN_READY" dataType="Boolean">false</
    memory>
<!--      <memory name="CHANGED_FLIGHT_PLAN" dataType="Boolean">
    true</memory-->
      <memory name="FLIGHT_STATE" dataType="String">NONE</memory>

      <!-- Start state -->
      <startState>IDLE</startState>

      <!-- States -->
      <states>
        <state name="IDLE" load="0">
          <!-- transitions -->
          <transition durationMin="1" durationMax="1" priority="10
    ">
            <description>Received new mission, begin building
    flight plan.</description>
            <inputs>
              <channel name="NEW_MISSION" predicate="eq">
    NEW_MISSION</channel>
            </inputs>
            <outputs>
              <channel name="NEW_MISSION"><null/></channel>
            </outputs>
            <endState>BUILD_FP</endState>
          </transition>
        </state>

        <state name="BUILD_FP" load="2">
          <!-- transitions -->
          <transition durationMin="900" durationMax="1200"
    priority="10">
            <description>Build a flight plan</description>
            <inputs>
              <memory name="FLIGHT_PLAN_READY" predicate="eq">
    false</memory>
```

```
            <channel name="VISUAL_UAS_UAVOP" predicate="ne">BLAH
</channel>
          </inputs>
          <outputs>
            <channel name="TACTILE_UAVOP_UAS">
              <layer name="MOUSE" dataType="String">FLIGHT_PLAN<
/layer>
              <layer name="KEYBOARD" dataType="String">
FLIGHT_PLAN</layer>
            </channel>
            <memory name="FLIGHT_PLAN_READY" dataType="Boolean">
true</memory>
          </outputs>
          <endState>BUILD_FP</endState>
        </transition>

        <transition durationMin="1" durationMax="1" priority="10
">
          <description>Click the Send flight plan button</
description>
          <inputs>
            <memory name="FLIGHT_PLAN_READY" predicate="eq">true
</memory>
            <memory name="FAA_APPROVED" predicate="eq">false</
memory>
            <channel name="VISUAL_UAS_UAVOP" predicate="ne">BLAH
</channel>
          </inputs>
          <outputs>
            <channel name="TACTILE_UAVOP_UAS">
              <layer name="MOUSE" dataType="String">
SEND_FLIGHT_PLAN</layer>
            </channel>
          </outputs>
          <endState>END_BUILD_FP</endState>
        </transition>
      </state>

      <state name="END_BUILD_FP" load="0">
        <transition durationMin="1" durationMax="1" priority="1"
>
          <description>Clear mouse and keyboard output layers</
description>
          <inputs>
          </inputs>
          <outputs>
            <channel name="TACTILE_UAVOP_UAS">
              <layer name="MOUSE"><null/></layer>
              <layer name="KEYBOARD"><null/></layer>
            </channel>
          </outputs>
          <endState>WAITING_ON_FAA</endState>
        </transition>
```

```
    </state>

    <state name="WAITING_ON_FAA" load="1">
      <transition durationMin="30" durationMax="60" priority="
10">
        <description>FAA approved the flight</description>
        <inputs>
          <channel name="VISUAL_UAS_UAVOP">
            <layer name="FLIGHT_PLAN" predicate="eq">
FLIGHT_PLAN_APPROVED</layer>
          </channel>
        </inputs>
        <outputs>
          <channel name="TACTILE_UAVOP_UAS">
            <layer name="MOUSE">TAKEOFF</layer>
          </channel>
          <memory name="FAA_APPROVED" dataType="Boolean">true<
/memory>
        </outputs>
        <endState>END_WAITING_ON_FAA</endState>
      </transition>

    </state>

    <state name="END_WAITING_ON_FAA" load="0">
      <transition durationMin="1" durationMax="1" priority="1"
>
        <description>Clear user output</description>
        <inputs>
        </inputs>
        <outputs>
          <channel name="TACTILE_UAVOP_UAS">
            <layer name="MOUSE"><null/></layer>
          </channel>
        </outputs>
        <endState>MONITOR_TAKEOFF</endState>
      </transition>
    </state>

    <state name="MONITOR_TAKEOFF" load="1">
      <transition durationMin="1" durationMax="1" priority="1"
>
        <description>UAV is airborne, move to monitor GUI</
description>
        <inputs>
          <channel name="VISUAL_UAV_UAVOP" predicate="eq">
FLYING</channel>
        </inputs>
        <outputs>
          <memory name="FLIGHT_STATE">NORMAL</memory>
        </outputs>
        <endState>MONITOR_UAVGUI</endState>
      </transition>
```

```
    </state>

    <state name="MONITOR_UAVGUI" load="1">
      <transition durationMin="900" durationMax="900" priority
="0">
        <description>Monitoring the UAVGUI</description>
        <inputs>
          <channel name="VISUAL_UAS_UAVOP">
            <layer name="UAV" predicate="eq">FLYING</layer>
            <layer name="RADAR" predicate="ne">COLLISION</
layer>
            <layer name="EMERGENCY_NOTAM" predicate="ne">
EMERGENCY_NOTAM</layer>
            <layer name="NEW_NOTAM_ON_FLIGHT_PATH" predicate="
ne">NEW_NOTAM_ON_FLIGHT_PATH</layer>
          </channel>
        </inputs>
        <outputs>
          <channel name="TACTILE_UAVOP_UAS">
            <layer name="MOUSE">MONITORING_UAVGUI</layer>
          </channel>
        </outputs>
        <endState>MONITOR_UAVGUI</endState>
      </transition>

      <transition durationMin="1" durationMax="1" priority="10
">
        <description>Operator sees that the UAV is beginning
the landing approach</description>
        <inputs>
          <channel name="VISUAL_UAS_UAVOP">
            <layer name="UAV" predicate="eq">LANDING</layer>
          </channel>
        </inputs>
        <outputs>

        </outputs>
        <endState>MONITOR_LANDING</endState>
      </transition>

      <transition durationMin="1" durationMax="1" priority="20
">
        <description>Operator sees a potential conflict and
trys to avoid it</description>
        <inputs>
          <channel name="VISUAL_UAS_UAVOP">
            <layer name="RADAR" predicate="eq">COLLISION</
layer>
          </channel>
        </inputs>
        <outputs></outputs>
        <endState>AVOID_COLLISION</endState>
      </transition>
```

```
        <transition durationMin="1" durationMax="1" priority="19
">
          <description>Operator notices an emergency notam on
the UAV and attempts to assist</description>
          <inputs>
            <channel name="VISUAL_UAS_UAVOP">
              <layer name="EMERGENCY_NOTAM" predicate="eq">
EMERGENCY_NOTAM</layer>
            </channel>
          </inputs>
          <outputs>
            <memory name="FLIGHT_STATE">EMERGENCY_NOTAM</memory>
          </outputs>
          <endState>AVOID_EMERGENCY_NOTAM</endState>
        </transition>

        <transition durationMin="1" durationMax="1" priority="18
">
          <description>Operator notices that the UAV is avoiding
 a NOTAM and attempts to assist</description>
          <inputs>
            <channel name="VISUAL_UAS_UAVOP">
              <layer name="UAV" predicate="eq">AVOID_NOTAM</
layer>
            </channel>
          </inputs>
          <outputs>
            <memory name="FLIGHT_STATE">EMERGENCY_NOTAM</memory>
          </outputs>
          <endState>AVOID_EMERGENCY_NOTAM</endState>
        </transition>

        <transition durationMin="1" durationMax="1" priority="15
">
          <description>Operator notices new NOTAM(s) on the
flight plan, begin changing the flight plan.</description>
          <inputs>
            <channel name="VISUAL_UAS_UAVOP">
              <layer name="NEW_NOTAM_ON_FLIGHT_PATH" predicate="
eq">NEW_NOTAM_ON_FLIGHT_PATH</layer>
            </channel>
            <memory name="FLIGHT_STATE" predicate="eq">NORMAL</
memory>
          </inputs>
          <outputs>
            <memory name="FLIGHT_STATE">NEW_NOTAMS</memory>
          </outputs>
          <endState>AVOID_NOTAM</endState>
        </transition>
      </state>

      <state name="MONITOR_LANDING" load="1">
        <transition durationMin="30" durationMax="60" priority="
10">
```

```
          <description>Operator sees that the UAV has landed and
 sets the mission to complete.</description>
          <inputs>
            <channel name="VISUAL_UAS_UAVOP">
              <layer name="UAV" predicate="eq">GROUNDED</layer>
            </channel>
          </inputs>
          <outputs>
            <channel name="TACTILE_UAVOP_UAS">
              <layer name="KEYBOARD">MISSION_COMPLETE</layer>
              <layer name="MOUSE">MISSION_COMPLETE</layer>
            </channel>
          </outputs>
          <endState>IDLE</endState>
        </transition>
      </state>

      <state name="AVOID_COLLISION" load="2">
        <transition durationMin="300" durationMax="600" priority
="100">
          <description>Operator is in the act of deconflicing
the UAV</description>
          <inputs>
            <channel name="VISUAL_UAS_UAVOP">
              <layer name="RADAR" predicate="eq">COLLISION</
layer>
            </channel>
          </inputs>
          <outputs>
            <channel name="TACTILE_UAVOP_UAS">
              <layer name="KEYBOARD">AVOID_COLLISION</layer>
              <layer name="MOUSE">AVOID_COLLISION</layer>
            </channel>
          </outputs>
          <endState>AVOID_COLLISION</endState>
        </transition>

        <transition durationMin="1" durationMax="1" priority="10
">
          <description>Collision has been avoided, return to
watching the UAVGUI</description>
          <inputs>
            <channel name="VISUAL_UAS_UAVOP">
              <layer name="RADAR" predicate="ne">COLLISION</
layer>
            </channel>
          </inputs>
          <outputs>
            <channel name="TACTILE_UAVOP_UAS">
              <layer name="KEYBOARD"><null/></layer>
              <layer name="MOUSE"><null/></layer>
            </channel>
          </outputs>
```

```
            <endState>MONITOR_UAVGUI</endState>
          </transition>
        </state>

        <state name="AVOID_EMERGENCY_NOTAM" load="1">
          <transition durationMin="300" durationMax="300" priority
    ="1">
            <description>Operator is waiting for the UAV to auto
    remove itself from the Emergency Notam</description>
            <inputs>
<!--              <channel name="VISUAL_UAS_UAVOP">-->
<!--                <layer name="EMERGENCY_NOTAM" predicate="eq">
    EMERGENCY_NOTAM</layer>-->
<!--                <layer name="UAV" predicate="ne">LOITER</layer
    >-->
<!--              </channel>-->
<!--              <memory name="FLIGHT_STATE" predicate="eq">
    EMERGENCY_NOTAM</memory>-->
            </inputs>
            <outputs>
              <!-- DO Nothing the UAV will fix itself, then we
    will continue -->
<!--              <channel name="TACTILE_UAVOP_UAS">-->
<!--                <layer name="KEYBOARD">CHANGE_FLIGHT_PLAN</
    layer>-->
<!--                <layer name="MOUSE">CHANGE_FLIGHT_PLAN</layer>
    -->
<!--              </channel>-->
<!--              <memory name="CHANGED_FLIGHT_PLAN">true</memory>
    -->
            </outputs>
            <endState>AVOID_EMERGENCY_NOTAM</endState>
          </transition>

          <transition durationMin="5" durationMax="5" priority="
    100">
            <description>Operator sees that the UAV is loitering,
    have it resume flight now.</description>
            <inputs>
              <channel name="VISUAL_UAS_UAVOP">
                <layer name="EMERGENCY_NOTAM" predicate="ne">
    EMERGENCY_NOTAM</layer>
                <layer name="UAV" predicate="eq">LOITER</layer>
              </channel>
              <memory name="FLIGHT_STATE" predicate="eq">
    EMERGENCY_NOTAM</memory>
            </inputs>
            <outputs>
              <channel name="TACTILE_UAVOP_UAS">
                <layer name="KEYBOARD">RESUME_FLIGHT</layer>
                <layer name="MOUSE">RESUME_FLIGHT</layer>
              </channel>
              <memory name="FLIGHT_STATE">NORMAL</memory>
            </outputs>
```

```
          <endState>AVOID_EMERGENCY_NOTAM</endState>
        </transition>

        <transition durationMin="1" durationMax="1" priority="10
">
          <description>NOTAM has been avoided, return to
watching the UAVGUI</description>
          <inputs>
            <memory name="FLIGHT_STATE" predicate="eq">NORMAL</
memory>
          </inputs>
          <outputs>
            <channel name="TACTILE_UAVOP_UAS">
              <layer name="KEYBOARD"><null/></layer>
              <layer name="MOUSE"><null/></layer>
            </channel>
          </outputs>
          <endState>MONITOR_UAVGUI</endState>
        </transition>
      </state>

      <state name="AVOID_NOTAM" load="1">
        <transition durationMin="600" durationMax="1200"
priority="10">
          <description>Operator is changing the flight plan for
a normal NOTAM</description>
          <inputs>
            <channel name="VISUAL_UAS_UAVOP">
              <layer name="NEW_NOTAM_ON_FLIGHT_PATH" predicate="
eq">NEW_NOTAM_ON_FLIGHT_PATH</layer>
            </channel>
            <memory name="FLIGHT_STATE" predicate="eq">
NEW_NOTAMS</memory>
          </inputs>
          <outputs>
            <channel name="TACTILE_UAVOP_UAS">
              <layer name="KEYBOARD">CHANGE_FLIGHT_PLAN</layer>
              <layer name="MOUSE">CHANGE_FLIGHT_PLAN</layer>
            </channel>
            <memory name="FLIGHT_STATE">NORMAL</memory>
          </outputs>
          <endState>AVOID_NOTAM</endState>
        </transition>

        <transition durationMin="1" durationMax="1" priority="10
">
          <description>NOTAM has been avoided, return to
watching the UAVGUI</description>
          <inputs>
            <memory name="FLIGHT_STATE" predicate="eq">NORMAL</
memory>
          </inputs>
          <outputs>
            <channel name="TACTILE_UAVOP_UAS">
```

```
                <layer name="KEYBOARD"><null/></layer>
                <layer name="MOUSE"><null/></layer>
              </channel>
            </outputs>
            <endState>MONITOR_UAVGUI</endState>
          </transition>

        </state>

      </states>
    </actor>
</actors>

<!-- System Events -->
<events>
  <event name="NEW_MISSION" count="1">
    <transition>
      <description> Start a new UAV mission</description>
      <inputs>
      </inputs>
      <outputs>
        <channel name="NEW_MISSION">NEW_MISSION</channel>
      </outputs>
    </transition>
  </event>

  <event name="RADAR_COLLISION" count="1">
    <transition>
      <description>Potential collision on UAS radar</description
>
      <inputs>
        <channel name="DATA_UAV_UAS" predicate="eq">FLYING</
channel>
        <channel name="RADAR_COLLISION" predicate="ne">
RADAR_COLLISION</channel>
      </inputs>
      <outputs>
        <channel name="RADAR_COLLISION">RADAR_COLLISION</channel
>
      </outputs>
    </transition>
  </event>

  <event name="FAA_COLLISION" count="1">
    <transition>
      <description>FAA radar shows a potential collision with a
UAV</description>
      <inputs>
        <channel name="DATA_UAV_UAS" predicate="eq">FLYING</
channel>
        <channel name="FAA_COLLISION" predicate="ne">COLLISION</
channel>
        <channel name="RADAR_COLLISION" predicate="ne">
RADAR_COLLISION</channel>
```

```
        </inputs>
        <outputs>
          <channel name="FAA_COLLISION">COLLISION</channel>
        </outputs>
      </transition>
    </event>
  </events>
</team>
```

UAS_in_NAS_auto_avoid_notam.xml

Appendix D

Data & Logs Generated by the Model

References

[1] *Enhanced operator function model: a generic human task behavior modeling language*, SMC'09, 2009. IEEE Press.

[2] *Evaluating Human-human Communication Protocols with Miscommunication Generation and Model Checking*, 2013.

[3] *Using Formal Verification to Evaluate Human-Automation Interaction: A Review*, 2013.

[4] *Modeling UASs for Role Fusion and Human Machine Interface Optimization*, SMC'13, 2013.

[5] *A Synergistic and Extensible Framework for Multi-Agent System Verification*, AAMAS, 2013.

[6] *Aviation Safety: Modeling and Analyzing Complex Interactions between Humans and Automated Systems*, ATACCS, 2013.

[7] J.A. Adams, J.L. Cooper, M.A. Goodrich, C. Humphrey, M. Quigley, B.G. Buss, and B.S. Morse. Camera-equipped mini UAVs for wilderness search support: Task analysis and lessons from field trials. *Journal of Field Robotics*, 25(1-2), 2007.

[8] J.A. Adams, C.M. Humphrey, M.A. Goodrich, J.L. Cooper, B.S. Morse, C. Engh, and N. Rasmussen. Cognitive task analysis for developing unmanned aerial vehicle wilderness search support. *Journal of cognitive engineering and decision making*, 3(1):1, 2009.

[9] Julie A Adams, Curtis M Humphrey, Michael A Goodrich, Joseph L Cooper, Bryan S Morse, Cameron Engh, and Nathan Rasmussen. Cognitive task analysis for developing unmanned aerial vehicle wilderness search support. *Journal of cognitive engineering and decision making*, 3(1):1–26, 2009.

[10] H.A.F. Almurib, P.T. Nathan, and T.N. Kumar. Control and path planning of quadrotor aerial vehicles for search and rescue. In *SICE Annual Conference (SICE), 2011 Proceedings of*, pages 700–705. IEEE, 2011.

[11] John R Anderson, Daniel Bothell, Michael D Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological review*, 111(4):1036, 2004.

[12] G. E. P. Box. Science and statistics. *Journal of the American Statistical Association*, 71 (356):791–799, 1976.

[13] Michael A. Goodrich Bryan S. Morse, Daniel Thornton. Color anomaly detection and suggestion for wilderness search and rescue. 2011.

[14] S.T. Cluff. *A Unified Approach to GPU-Accelerated Aerial Video Enhancement Techniques*. PhD thesis, Brigham Young University, 2009.

[15] J. Cooper and M.A. Goodrich. Towards combining UAV and sensor operator roles in UAV-enabled visual search. In *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, pages 351–358. ACM, 2008.

[16] J.L. Cooper. Supporting flight control for UAV-assisted wilderness search and rescue through human centered interface design. Technical report, DTIC Document, 2007.

[17] J.L. Cooper and M.A. Goodrich. Integrating critical interface elements for intuitive single-display aviation control of UAVs. In *Proceedings of SPIE*, volume 6226, page 62260B. Citeseer, 2006.

[18] M. L. Cummings, C. E. Nehme, J. Crandall, and P. Mitchell. *Developing Operator Capacity Estimates for Supervisory Control of Autonomous Vehicles*, volume 70 of *Studies in Computational Intelligence*, pages 11–37. Springer, 2007.

[19] Mary L Cummings, Carl E Nehme, Jacob Crandall, and Paul Mitchell. Predicting operator capacity for supervisory control of multiple uavs. In *Innovations in Intelligent Machines-1*, pages 11–37. Springer, 2007.

[20] C.H. Engh. *A see-ability metric to improve mini unmanned aerial vehicle operator awareness using video georegistered to terrain models*. PhD thesis, Brigham Young University, 2008.

[21] R. Gonzalez and R. Woods. *Digital Image Processing*. Prentice Hall, Upper Saddle River, NJ, 2002.

[22] M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, M. Quigley, J. A. Adams, and C. Humphrey. Supporting wilderness search and rescue using a camera-equipped mini UAV. *Journal of Field Robotics*, 25(1-2):89–110, 2008.

[23] M.A. Goodrich, J.L. Cooper, J.A. Adams, C. Humphrey, R. Zeeman, and B.G. Buss. Using a mini-UAV to support wilderness search and rescue: Practices for human-robot teaming. In *Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on*, pages 1–6. IEEE, 2007.

[24] M.A. Goodrich, B.S. Morse, D. Gerhardt, J.L. Cooper, M. Quigley, J.A. Adams, and C. Humphrey. Supporting wilderness search and rescue using a camera-equipped mini UAV. *Journal of Field Robotics*, 25(1-2):89–110, 2008.

[25] M.A. Goodrich, B.S. Morse, C. Engh, J.L. Cooper, and J.A. Adams. Towards using unmanned aerial vehicles (UAVs) in wilderness search and rescue: Lessons from field trials. *Interaction Studies*, 10(3):453–478, 2009.

[26] Michael A. Goodrich. On maximizing fan-out: Towards controlling multiple unmanned vehicles. In M. Barnes and F. Jentsch, editors, *Human-Robot Interactions in Future Military Operations*. Ashgate Publishing, Surrey, England, 2010.

[27] C.M. Humphrey, S.R. Motter, J.A. Adams, and M. Gonyea. A human eye like perspective for remote vision. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 1650–1655. IEEE, 2009.

[28] Curtis Michael Humphrey. *Information abstraction visualization for human-robot interaction*. PhD thesis, 2009.

[29] Robert J. Koester. *Lost Person Behavior: A Search and Rescue Guide on Where to Look - for Land, Air and Water*. dbS Productions Charlottesville, Virginia, 2008.

[30] L. Lin and M.A. Goodrich. UAV intelligent path planning for wilderness search and rescue. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 709–714. IEEE, 2009.

[31] L. Lin, M. Roscheck, M.A. Goodrich, and B.S. Morse. Supporting wilderness search and rescue with integrated intelligence: Autonomy and information at the right time and the right place. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[32] P. M. Mitchell and M. L. Cummings. Management of multiple dynamic human supervisory control tasks. In *10th International Command and Control Research And Technology Symposium*, 2005.

[33] Bryan S. Morse, Cameron H. Engh, and Michael A. Goodrich. UAV video coverage quality maps and prioritized indexing for wilderness search and rescue. In *Proceedings of*

the 5th ACM/IEEE international conference on Human-robot interaction, HRI '10, pages 227–234, New York, NY, USA, 2010. ACM. ISBN 978-1-4244-4893-7. doi: http://doi.acm.org/10.1145/1734454.1734548. URL `http://doi.acm.org/10.1145/1734454.1734548`.

[34] B.S. Morse, D. Gerhardt, C. Engh, M.A. Goodrich, N. Rasmussen, D. Thornton, and D. Eggett. Application and evaluation of spatiotemporal enhancement of live aerial video using temporally local mosaics. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[35] R. Murphy, S. Stover, K. Pratt, and C. Griffin. Cooperative damage inspection with unmanned surface vehicle and micro unmanned aerial vehicle at hurrican Wilma. IROS 2006 Video Session, October 2006.

[36] R. R. Murphy and J. L. Burke. The safe human-robot ratio. In M. Barnes and F. Jentsch, editors, *Human-Robot Interaction in Future Military Operations*, chapter 3, pages 31–49. Ashgate Publishing, 2010.

[37] N.D. Rasmussen, D.R. Thornton, and B.S. Morse. Enhancement of unusual color in aerial video sequences for assisting wilderness search and rescue. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 1356–1359. IEEE, 2008.

[38] Dario D Salvucci and Niels A Taatgen. Threaded cognition: an integrated theory of concurrent multitasking. *Psychological review*, 115(1):101, 2008.

[39] T. J. Setnicka. *Wilderness Search and Rescue*. Appalachian Mountain Club, 1980.

[40] Tim J. Setnicka. *Wilderness Search and Rescue*. Apalachian Mountain Club, Boston, MA, 1980.

[41] *Integration of Civil Unmanned Aerial Systems (UAS) into the National Airspace System (NAS) Roadmap*. U.S. Department of Transporation Federal Aviation Administration, first edition edition, 2013.

[42] S. Waharte, N. Trigoni, and S. Julier. Coordinated search with a swarm of UAVs. In *Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2009. SECON Workshops' 09. 6th Annual IEEE Communications Society Conference on*, pages 1–3. IEEE, 2009.

[43] Christopher D Wickens. Multiple resources and performance prediction. *Theoretical issues in ergonomics science*, 3(2):159–177, 2002.