Modeling UASs for Role Fusion and Human Machine Interface Optimization

TJ Gledhill, Jared Joseph Moore, Robert Ivie, Eric Mercer and Michael A. Goodrich

***Abstract*—Recent research shows that wilderness search and rescue (WiSAR) can be aided through the use of unmanned aerial systems (UASs). A single UAS, however, requires several human operators to manage the interface between the UAS vehicle and the larger search and rescue operation. For UASs to scale to real-world wilderness search and rescue scenarios, it is important to reduce operator workload and mitigate the effects of stress and fatigue through effective distributed control and augmented autonomy. A primary challenge in any effort to understand distributed control is effectively modeling the various roles in the system such as the human, GUI, and physical enviroment. This paper discusses a Java model that explicitly formalizes the individual roles of the WiSAR UAS that can be model checked by Java Pathfinder to establish its intended behavior. The model is the basis for research on human machine interfaces to support combined human roles that reduce operator workload. In essence, by modeling each individual role in WiSAR, it is possible to perform role fusion and show that the new UAS is a correct implementation of the original system with the addition of combined roles, increased autonomy, and new interfaces. The experience of this modeling activity suggests that modeling WiSAR or any system will be at least as hard as any solution to distributed control or role fusion.**

***Index Terms*—UAS Modeling, HMI Optimization**

I. Introduction

Problem Statement: UASs require several human operators to monitor and administer.

Unmanned Aerial Systems (UASs) most often require two human operators to control the Unmanned Aerial Vehicle (UAV). One operator controls flight while the other controls the payloads such as sensors, cameras, or weapons. In addition to this a third human is responsible for overseeing task completion and interfacing with the command structure. It is preferable to relieve a majority of this workload using augmented autonomy and enhanced user interfaces.

A viable plan to relieve workload requires specific steps. First, model a specific UAS. Second, change the model to minimize operator workload. Third, verify the effect of those changes. Last, introduce those changes into a graphical user interface. This paper focuses on the completion of the first objective, modeling a specific UAS.

The specific UAS is the wilderness search and rescue (WiSAR) UAS developed at Brigham Young University. The WiSAR UAS is fairly standard consisting of three humans, two GUIs, and a single UAV.

The model explicitly represents the various roles, which characterize the WiSAR UAS, as a group of Mealy machines running in parallel. A custom set of Java interfaces constitute the model. These interfaces simulate a discrete time environment, facilitate input/output between roles, and provide non-deterministic event handling.

Java Pathfinder (JPF) checks the model. JPFs model checking is convenient because JPF runs on the compiled Java code without a complicated conversion process. You can find more information about how JPF functions, as well as a thorough explanation of the model later in this paper.

II. Related Work

From video game creators to NASA researchers more people are obsessed with finding, producing, and examining digital models of the real world. Most want to better understand and verify real world events. Some people have chosen to use Brahms as their medium. Brahms is a robust modeling language that involves agent, geography, and object classes. These classes are used to represent the people involved, their environment, and the tools they have to work with.

NASA Ames Research Center is using Brahms to model interactions between operators and their aerial equipment. These complex models have given new understanding to both the instances studied and the language itself. In their study of the Uberlingen collision the model, produced using the Brahms language, Neha Rungta and her colleagues were able to correctly predict the collision. Such a model could have forewarned the air traffic controller of the collision.

Wilderness Search and Rescue is primarily concerned with finding people who have become lost in rugged terrain. To further this goal they use unmanned aerial vehicles (UAVs), which facilitate the work this group is doing. In principle the vehicle searches for victims in areas that would be difficult for ground teams to reach. Michael Goodrich and his colleagues tested the effectiveness of these types of operations. They find that altitude and video clarity determined the success of the mission. Furthermore they suppose that these factors could be enhanced if the roles of the UAV operator and video operator were combined. By modeling this role fusion the group can verify new user interfaces before risking expensive equipment.

III. WiSAR UAS Domain

WiSAR represents search and rescue efforts in remote, varying, and dangerous terrains. According to T.J. Setnicka **??** there are four core elements of a Wilderness Search and Rescue operation.

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

Fig. 1. Core SAR Elements

The WiSAR UAS operates within this first element. During the locate phase the incident commander (IC) develops a strategy to obtain information, and coordinates the overlying Parent Search (PS). The information obtaining strategy makes use of the available tactics to be successful. One tactic may be using search dogs in a specific region or using a trained tracker on the target's trail.

The WiSAR UAS is one of these tactics. It consists of three humans: Mission Manager (MM), Operator (OP), and Video Operator(VO) working with the Operator GUI (OGUI), Video Operator GUI (VGUI) and the UAV. This team coordinates its efforts with the PS which represents the entire command structure for the search and rescue operation. The key roles, interfaces, and objects of the WiSAR UAS emanate from human to human interactions, human to UAV interactions, human to GUI interactions, unpredictable outside stimulus, and varying task durations.

IV. Simulating the WiSAR UAS

In order to simulate critical aspects of the WiSAR UAS it becomes necessary to model the different roles, interfaces, and objects in a way that facilitates the desired interactions, problems and task durations. We simulate roles using state machines. Actor is the chosen alias for this state machine. Actors specifically symbolize Mealy machines, a type of state machine, represented by equations 1, 2. Where S is a set of states, S_0 the start state, Σ_A the set of all Actor inputs, λ_A the set of all Actor outputs, and T a transition matrix which specifies the outputs for any state transition.

$$Actor = (S, S_0, \Sigma_A, \Lambda_A, T) \tag{1}$$

$$T : S \times \Sigma_A \Rightarrow S \times \Lambda_A \tag{2}$$

Events are Moore State machines that spontaneously change state provided the appropriate conditions of the simulation are met.

Simulating the passage of time provides a metric whereby we can monitor the interactions between Actors in our model. Each action an Actor can take is assigned a range spanning the minimum and maximum time required to complete thereby inserting non-determinism into the system.

Each Actor implements the IActor interface with the methods: processNextState and processInputs. The process-NextState method moves the actor into its next state, generating outputs and setting the next state to a default value and duration. The processInputs method then looks at its inputs and determines what the next state should be and when it should occur. This method also generates additional outputs.

Each Event implements the IEvent interface. The IEvent interface is fairly similar to the IActor interface except it has the method getCount. This makes it possible to insert multiple events of the same type without having to instantiate entirely new objects. Events are also given higher priority than Actors ensuring that all applicable events execute prior to any actors on a given time-step.

Each actor has a method getNextStateTime. This returns the global time-step when the actor will change state. The simulator takes the minimum value of all the nextStateTimes and advances the global clock to that time. If however the time outputted is 0 then the simulator treats that as a signal that the model is done processing and terminates. This eliminates unnecessary processing by ensuring the simulator runs when the system is changing.

Figure 2 portrays the execution of a time step. The simulator first processes the next state of any events that undergo a state change in that time. The outputs of those events are stored in the Post Office to be accessed by the Actors when applicable. The Post Office stores the current inputs to each Actor as well as the inputs that are to be processed on the next time step. Next the Simulator calls processNextState on each of the actors, updating their outputs in the Post Office. The Post Office will then load all the next state inputs into the current inputs making them visible to the actors. This method of passing information makes it possible to provide the illusion of concurrency to each of the actors in the model. The Actors then each call processInputs. Once each Actor has evaluated its inputs and made any changes the simulator checks if there are any future time-steps to evaluate and if not it terminates the imulation.

The main weakness imposed by the methods we used here is that there is no framework already in place. This also is our main strength. Languages such as Brahms are highly structured and so there is a significant amount of work already done for any system that uses that language but by the same point if a slight change in the execution framework becomes necessary it is virtually impossible to implement. In our system such changes are fairly easily adapted.

V. WiSAR UAS Model

This section describes how we modeled the WiSAR UAS using the framework that was previously mentioned. Due to the size of the model this section does not contain the full detail of the work that was done. Instead we have condensed the process into a few key concepts which are explained with real examples from the actual modeling process.

A. Actors

Choosing the core actors was not a trivial task, because we were looking for a level of abstraction that gave results without adding unwanted complexity to the model. The final model simulates the following team of actors: the parent search (PS), mission manager (MM), UAV operator (OP), video operator (VO), operator GUI (OGUI), video operator GUI (VGUI), and the UAV. An example of the preferred level of abstraction is found in our modeling of weather. We decided to model the effects of weather as events on the UAV instead of creating a weather actor. This reduces the number of actors, but limits our analysis of weather effects on the entire system which we found acceptable. Another example was the addition of sub-actors to the UAV actor. Sub-actors are parts of a more complex actor, in this case the UAV. The sub-actors for UAV are the battery, flight plan, height above ground, and the signal. Their inputs and outputs are handled differently than a normal actor. These subactors inputs are linked within the simulator to the UAV. The UAV also has specific modifications that allow it to process all sub-actors before completing its own processing. This level of abstraction reduced the very complex UAV actor into multiple, simpler actors. This gave us confidence that this modeling framework is robust enough to model anything we may need. It also gave us a pattern for modeling the different levels of abstraction that we desire.

B. Events

We used events for several different abstractions. The first class of abstraction is the common event. These events are common and expected within the WiSAR UAS. Examples are receiving new search areas from the parent search or viewing an anomaly on the video GUI. Before anomalies appear on the video GUI the event class randomly determines, within a range, how long the anomaly is visible. The video GUI receives input when the anomaly becomes visible and when it is no longer visible. This allowed us to add non-determinism to the core of the model without complicating the actors. The second class of abstraction is the uncommon event. These are events which are outside normal operation. They represent unforeseen problems that can occur at any time. In essence these types of events push the actors into state spaces that they would never reach during normal operation. Examples are low height above ground, loss of signal, and early low battery. If a low height above ground event is triggered the operator must modify the UAV flight plan before the UAV crashes. The sub-actor UAVHeightAboveGound receives the event input and changes its internal state. It then watches the UAV input to see which occurs first, a flight plan modification or low height above ground complete. If the flight plan modification is received, then the events input is ignored. Otherwise the UAV crashes.

C. Asserts

The more complicated the model the more things that can go wrong. Some errors are caused by coding bugs and some are flaws in the model. Flaws in the model are extremely valuable, but it can be challenging to tell them apart. To catch these errors 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. In our model we have used asserts in two ways. The first is detection of 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 way we use asserts deal with inputs. Many operations are sequential requiring an existing state before they can be performed. By looking at inputs received we are able to tell if actors are out of sync with one another. An example of this is the OP_TAKE_OFF input for the UAV. If the UAV is 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 the model and we found that having too many asserts is preferable to having too few.

VI. Results

In this section we present empirical results to demonstrate that WiSAR UAS is modeled effectively using the approach we have taken. We will demonstrate this using a brief case study extracted from WiSAR UAS. We will then give an analysis of the JPF results we obtained when we verified the model.

A. Case Study: Anomaly Detection

This scenario represents what should occur when the video operator believes they see the target on the video GUI. 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.

B. Model Construction

As mentioned in section V we use seven core actors in our model. The anomaly detection effects every single one of these actors. Because we already had the core actor classessetup before we defined this part of the model it was a simple process of adding additional states and transitions to our actors. For the anomalies themselves we created an anomaly event which sent output directly to the video GUI. We wanted to the operator to non-deterministically choose if an anomaly was not seen, seen as a possible target, seen as a likely target. We were able to acheive this by generating a random probability within a designated range. This means that when run by JPF each of those probabilities will be explored. A small portion of the model can be seen in figure 3. While constructing this portion of our model we used mock actors to represent some of the logic which belonged to the core classes. This allowed us to get the model working before we had finished coding. This proved to be very useful and we can see how it can be an effective tool for future verification. If we are not interested in the non-deterministic states of one actor, or a portion of that actors state space, that portion of the model can be replaced with a mock actor. We did run into a few issues while coding the model. One problem we found was inconsistency in our human memory modeling. Without a strict grammar it is possible to do the same thing in many different ways which leads to flaws in the model. We also found that if we did not move logic into sub-actors the complex actors began to be unmanageable. In some cases moving logic into a sub-actor can be a difficult task. Another problem we found was the lack of asserts for invalid states and inputs. These asserts are a crucial part of determining if the model is performing correctly yet we had no way of enforcing that these asserts were placed into the code.

C. *JPF Results*

After we had coded this portion of our model we began testing the model using JPF. We found that using JPF made it much easier to find bugs in the model. When we began to verify the model one of the first occurences discovered was infinite loops. This told us that something was wrong. Typically these bug are caused by a failure to send the correct output, in one case the infinite loop occurred when the actor would change states too early. We discovered that the default maximum duration for that state was set too low. This represented a flaw in our model. After we had fixed our bugs and changed some of our default maximum durations we found that JPF

VII. Future Work

As stated previously this work is the basis for research on human machine interfaces to support combined human roles that reduce operator workload. To acheive this end we feel that the modeling framework we have presented would be more beneficial if it formalized some of the practices we found to be useful while modeling. We would also like the simulation to check the connectivity of the model before running the simulation to ensure that model is complete.

Acknowledgment

The authors would like to thank Neha Rungta of NASA Ames Intelligent Systems Division for her help with JPF and Brahms.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.