Modeling UASs for Role Fusion and Human Machine Interface Optimization

Michael Shell*, Homer Simpson†, James Kirk‡, Montgomery Scott‡ and Eldon Tyrell§
*School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, Georgia 30332–0250
Email: mshell@ece.gatech.edu
†Twentieth Century Fox, Springfield, USA
Email: homer@thesimpsons.com
‡Starfleet Academy, San Francisco, California 96678-2391
Telephone: (800) 555–1212, Fax: (888) 555–1212
§Tyrell Inc., 123 Replicant Street, Los Angeles, California 90210–4321

Abstract—**Recent research shows that wilderness search and rescue (WiSAR) can be aided through the use of unmanned arial 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.

In Unmanned Aerial Systems (UASs) the standard practice requires 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. A preferred model would off-load a majority of this workload onto the UAS using augmented autonomy and enhanced user interfaces.

The initial objectives for accomplishing this goal are to model a specific UAS, change the model to minimize operator workload, verify the effect of those changes, and introduce those changes into a graphical user interface. The focus of this paper is on the completion of the first objective, modeling a specific UAS.

The specific UAS modeled in this paper 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 explicity represents the various roles, within the WiSAR UAS, as a group of Mealy machines running in parallel. The model is constructed 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.

The model checking is performed using Java Pathfinder (JPF). This is convenient because JPF runs the model checking on the compiled Java code. This means that the model can be run directly by JPF without a complicated conversion process.

II. Related Work

Creating digital models of the real world is important to many disciplines. From video games creators to NASA researchers, more people are obsessed with finding, producing, and examining these digital models in order to better understand and verify real world events. Brahms is a robust modeling language that involves agents, geographies, and objects. These can also be thought of as the people involved, their environment, and the objects or 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 the finding people who have become lost in rugged terrain. Research has shown that UAVs, unmanned aerial vehicles, facilitate the work this group is doing. In principle the vehicle is used to search 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 found that altitude and video clarity determined the success of the mission. Furthermore they supposed that these factors could be enhanced if the roles of the UAV operator and video operator were combined.

III. WiSAR UAS Domain

Wilderness search and rescue represents search and rescue efforts in remote, varying, and dangerous terrains. According to T.J. Setnicka **??** there are four core elements of a WiSAR 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) developes a strategy to obtain information. This strategy makes use of the available tactics to obtain this information. 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 Parent Search (PS) which represents the entire command structure for the search and rescue operation. These key roles, interfaces, and objects of the WiSAR UAS represent human to human interactions, human to UAV interactions, human GUI interactions, unpredictable problems, and varying task durations.

IV. Simulating the WiSAR UAS

In order to simulate critical aspects of the WiSAR UAS it became necessary to model all of the different roles, interfaces, and objects in a simplified way that would facilitate the desired interations. We chose to do this using state machines. We refer to these state machines as Actors. Actors can be thought of as Mealy machines 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.

$$Actor = (S, S_0, \Sigma_A, \Lambda_A, T) \quad (1)$$

$$T : S \times \Sigma_A \Rightarrow S \times \Lambda_A \quad (2)$$

it is necessary to simulate the passage of time. Each actions duration is assigned a range spanning the minimum and maximum time required to complete. This inserts non-determinism into the system. The simulator uses these durations to control the interaction between Actors and Events. An Event is representative of an outside stimulus that gives inputs to Actors. Actors are representative of humans or objects such as the UAV or the GUI, anything that can change state or process information within the system.

Events and Actors are both implemented as Mealy state machines. Each Actor implements the IActor interface with the methods: processNextState and processInputs. These two methods implement the state change within the Actor, generate outputs and process its inputs. The two methods are separated to provide the abstraction necessary to make the simulator act as though it were composed of multiple Mealy state machines running in parallel. There are two main differences between an Actor and an Event. First, Events have higher priority than actors. Second, events have a method to determine if it can be processed given the conditions of the system.

To eliminate pointless processing the simulator will only process times for which there is a change in the state of an actor or an event. The lowest valid next time for a change is 1 so if the next time is 0 then the simulator knows that nothing is left to process and will terminate the simulation. Otherwise it will jump to the next time step.

The diagram below 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. Between processNextState and processInputs the Post Office will 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.

After processing the events, the Actors process their next state and update their outputs in the Post Office. 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 simulation.

Both the main weakness and the main strength imposed by the methods used here is that there is no pre-existing framework. 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. However, highly structured languages lack the flexibility to permit slight changes in the execution framework. In the system implemented here, such changes are fairly easily incorporated.

V. WiSAR UAS Model

We simplified the modeling process by using a custom simulator to represent the UAV-enabled WiSAR environment as parallel mealy machines. Within the simulator important aspects from the real environment are treated as either an actor or an event. This approach simplifies the model because there is no distinction between a human, a UAV, or the weather. Internal states, which change based on the input received by the other actors, represent all facets of the UAS.

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. By treating weather as a core actor any number of conditions could have been represented as internal states. Then the other actors could observe and respond to the different outputs from the weather actor. Instead we decided to only model these weather outputs as events affecting the UAV. This reduces the number of actors, but prevents us from doing detailed analysis of weather effects on the entire system. In this instance we reduced the number of actors by abstracting this facet of the model.

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 sub-actors 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 producing two significant results. Firstly, we verified the simulator will be capable of modeling anything we choose. Secondly, we verified that many of the tasks performed in the system represent a sequence of states.

Depending on the desired level of abstraction sequences can be modeled using a large actor or multiple small actors. This, and our familiarity with java, makes us confident that we will be able to simulate the entire model with the desired abstraction levels.

B. *Events*

We used several different types of events in our model. The first class of event is the common event. These events are common and expected within a UAV-enabled WiSAR operation. 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 event 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

We chose to use JPF, also known as Java Pathfinder, because it can explore all possible paths our simulation could have taken. In other words JPF would check all possible combinations of agent inputs to help us find errors. This tool was an essential part of our verification process. Using unhandled assertion statements in the code enhanced JPFs ability to find instances when our code wasnt acting in the intended way. JPF did its job. Right away it found multiple circumstances that would lead to an infinite loop. One of these loops involved an extremely improbable sequence of successive method calls that would have been nearly impossible to otherwise test for.

Our model consisted of multiple state machines that took a non-deterministic amount of time to form outputs. We choses this implementation because no human would take exactly the same amount of time to do the same thing every time they did it. The events also non-deterministically chose whether or not to occur at any time during a given time frame. This allowed for a more realistic test of the system since events could happen at any time. These two factors, while necessary for the realism of the simulator, would have been incredibly difficult to test without a tool like JPF. Only JPF could have found all possible paths our simulator could have taken. Because of the incredible amount of non-determinism it took a long time for

JPF to traverse every path. The time waited was only a small impediment in light of the amount of work it saved us.
Creation of the model was not easy. The intricacies and nuances of the simulator made each class difficult to write. It was, however, easier than learning a new language, where many more intricacies and nuances would have to be overcome before modeling could begin. Furthermore, we can reuse much of what we created to perform verification of future models, and since we used Java more people can understand our work by simple examination. We aimed for a high level of abstraction. We didnt want to simulate keyboard inputs or specific mouse clicks, but instead chose roles that significantly impacted the likelihood of positive sightings. This level of abstraction was achieved by definition of specific inputs and outputs for each role. The basic initialization of the search was the easiest part to model. It didnt take very long to get everything sorted and the UAV into simulated air. This involved an event that began the search as well as giving each role the ability to process the commands it received to begin the search. Modeling the sub roles of the UAV was difficult. It took a while before we could simulate flights that included crashes, video anomalies, and new search areas. Most of these difficulties came from the complexity each new event gave to the whole system. Simply tracing the expected flow of data between the agents was a huge task.

VII. Conclusion

The conclusion goes here.

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.