Framework for ADAPT Simulated Testing (FAST)

Tyler Alston, Michael Bonds, Ryan Condotta, Richard Garren, Reginald Hayes, Erik Jensen, Jerold Register, Darren Rose, Paul Smith, Spencer Smith, Andrew Sylvia, Brandon Waddell

Department of Modeling, Simulation and Visualization Engineering, Old Dominion University
Norfolk, Virginia 23529

talst007@odu.edu, mbond005@odu.edu, rcond002@odu.edu, rgarr016@odu.edu, rhaye012@odu.edu, ejens005@odu.edu, jregi002@odu.edu, drose012@odu.edu, psmit057@odu.edu, ssmit195@odu.edu, asylv005@odu.edu, bwadd006@odu.edu

ABSTRACT

Emergent air vehicles types, such as unmanned air vehicles (UAVs), have the potential to augment the benefits of aviation by offering new capabilities. However, the current air traffic management (ATM) system, which relies on centralized air traffic control (ATC), cannot accommodate the integration of these new vehicles with preexisting aircraft. Through the Autonomous Departure and Arrival Procedures and Technology (ADAPT) concept, NASA Langley Research Center (NASA-LaRC) will systematically develop a solution that incorporates these new vehicles into a decentralized ATC environment. NASA will construct a set of protocols, or "Rules of the Road" (ROR), which are the logical rules that each air vehicle uses to self-manage operations for travel. To assist NASA with its experimental testing, the 2015-2016 Old Dominion University Modeling and Simulation Engineering Capstone Team will design and develop a simulation software framework, FAST (Framework for ADAPT Simulated Testing), which incorporates agent-based modeling and simulation support components. FAST alone provides functionality for many-vehicle flight simulation and travel mission completion, but requires NASA to complete the simulation through the addition of ROR-related logic to add conflict detection and resolution capability. FAST and the other tools developed by the capstone class will allow NASA to quickly implement, test, and analyze their ROR models. This paper discusses the complete software architecture, which includes and supports FAST, and the capabilities of the system components. It will discuss each component, individually, of the FAST architecture which includes a vehicle repository, input editor, simulator, output analyzer, and visualization tool.

ABOUT THE AUTHORS

All authors of this paper are seniors at Old Dominion University expecting to graduate May, 2016 in the undergraduate program for Modeling, Simulation, and Visualization Engineering.
Tyler Alston: is minoring in Computer Science, and is interested in entertainment and educational applications of modeling and simulation.
Michael Bonds: is minoring in Engineering Management and currently assists with navigation simulation research.
Ryan Condotta: is minoring in Computer Science and Web Programming, with interests in military and aerospace simulations.
Richard Garren: is a US Navy Submarine Veteran, and is also enrolled in the accelerated graduate MSVE program while minoring in Biomedical Engineering and Computer Science.
Reginald Hayes: intends to pursue a Masters in Systems Engineering before pursuing a career in simulation design.
Erik Jensen: is minoring in Physics and Computer Science, and will pursue graduate study in Modeling and Simulation Engineering.
Jerold Register: is minoring in Actuarial Mathematics, interested in modeling transportation networks and systems.
Darren Rose: is minoring in Computer Science and Engineering Management, and intends to pursue a Master's in Modeling & Simulation Engineering while seeking out a career as a research scientist.
Paul Smith: enjoys coding and is currently assisting with foramen atlas research.
Spencer Smith: currently works for Skanska USA Civil. He is also the President of The Society for Modeling and Simulation International ODU Chapter. He has a minor is Computer Science and is interested in consulting.
Andrew Sylvia: has a strong background in robotics, with an interest in aerospace.
Brandon Waddell: intends to pursue a Masters in Modeling and Simulation with a focus on collaborative autonomous systems and distributed simulation.

Framework for ADAPT Simulated Testing (FAST)

Tyler Alston, Michael Bonds, Ryan Condotta, Richard Garren, Reginald Hayes, Erik Jensen, Jerold Register, Darren Rose, Paul Smith, Spencer Smith, Andrew Sylvia, Brandon Waddell

Department of Modeling, Simulation and Visualization Engineering, Old Dominion University
Norfolk, Virginia 23529

talst007@odu.edu, mbond005@odu.edu, rcond002@odu.edu, rgarr016@odu.edu, rhaye012@odu.edu, ejens005@odu.edu, jregi002@odu.edu, drose012@odu.edu, psmit057@odu.edu, ssmit195@odu.edu, asylv005@odu.edu, bwadd006@odu.edu

INTRODUCTION

Current airspace operations include thousands of concurrent flights departing from and arriving at airport locations across the globe. To operate safely and efficiently, these flights depend on various functions of an airspace system, including surveillance, navigation, communications, conflict management, hazard avoidance, and priority management in a centralized control structure.

The introduction of unmanned air vehicles (UAVs) and personal air vehicles (PAVs) with vertical take-off and landing capabilities will be incompatible with the current airspace system. The number of vehicles likely to populate the national airspace is projected to grow several times over the current number, and many of these vehicles will not originate at conventional departure or arrival points, such as at airports. The current system for managing aircraft will not be able to support the increased number of air vehicles entering and exiting the airspace system from non-airport locations without significant infrastructure improvements, increased staffing, and increased delays.

NASA-LaRC seeks to study the feasibility of integrating new vehicle types with traditional aircraft in shared airspace with limited requirements for new infrastructure. Through the Autonomous Departure and Arrival Procedures and Technology (ADAPT) concept, NASA will systematically develop a functional airspace system that incorporates these new vehicles into a decentralized air traffic control environment.

Acting autonomously, vehicles will need to be able to communicate with one another, resolve airspace conflict, and avoid restricted areas. NASA seeks to develop a set of protocols, or “Rules of the Road”, hereafter referred to as "ROR" in this document. ROR are the logical rules that each air vehicle will use to self-manage flight operations for safe and efficient air travel. The simulation framework will test how multiple air vehicles of different types that have different missions will accomplish their missions while traveling through shared airspace. The vehicles will abide by a set of air traffic rules that NASA programmers will apply to the simulation. Analysts will evaluate simulation output metrics that quantify system observations such as safety and efficiency in order to gauge the effectiveness of the proposed ROR.

An agent-based modeling (ABM) approach is being implemented in the simulation architecture to enable autonomous aircraft decision-making ability. FAST alone provides functionality for multiple-vehicle flight simulation and flight mission completion, but requires NASA to complete the simulation through the addition of ROR-related logic to add conflict detection and resolution capability.

This paper addresses the Capstone Team’s proposed solution for NASA: a high level software architectural design of the proposed solution, a discussion of how the design components will function toward meeting the system goals and objectives, and how the system design will support NASA’s ADAPT testing requirements. It will discuss each software component of the FAST architecture, which includes repositories, input editor, simulator, output analyzer, and visualizer. The remainder of this paper is structured as follows:

A Background section will survey some of the previous work in the domain of automated collision avoidance, providing context to the project undertaken by the Capstone Team. A section on the Conceptual Design of the system will provide the reader with a general understanding of what the system is capable of, in terms of functionality and

behavior. A section on the System Architecture will explain succinctly how each of the major components – including input, simulation, and output – function together to realize the conceptual design. A section on Agent Architecture will provide a high-level overview of the agent design and its role in the simulation. A section on Agent Framework will attempt to persuade the reader that the framework approach is best fitted to NASA's development needs. And finally, a Conclusion section will summarize the contents of the paper, followed by a brief statement of Acknowledgements to those who have aided the Capstone Team in their efforts, and a list of References.

BACKGROUND

The shift towards a distributed, autonomous air traffic system, described as free-flight, is already underway [Schultz et al., 1997]. In response to the growing demand for a safer and more efficient air traffic system, satellite-based navigation is taking over as the standard in surveillance and broadcasting avionics technology over ground-based navigational aids [FAA, 2015]. The promise of improved accuracy of information that will be made available – to aircraft, manned or unmanned, as well as to air traffic controllers – has itself spawned numerous research efforts directed towards the development of algorithms for automated collision avoidance in shared conflict scenarios.

Simulation offers safe and low-cost means of comparing algorithms in a multitude of scenarios, which would otherwise be infeasible using real aircraft or other physical models. However, the difficulty in developing and testing these algorithms cannot be overstated. Various approaches have been taken, from procedural rules which resolve conflict between two aircraft (Albaker & Rahim, 2010), to convex optimization (Frazzoli et al., 2000) and mixed-integer linear programming (Richards & How, 2002) formulations handling multiple conflicted aircraft.

Researchers at the Czech Technical Institute have developed AgentFly, a simulation tool with visualization capabilities that tests various collision avoidance algorithms against, as well as in concert with, one another (Pechoucek & Sislak, 2009). AgentFly is built on an agent-based architecture, and its focus is on the free-flight paradigm of decentralized air traffic control. Each aircraft is modeled as an agent capable of independent, as well as peer-to-peer, decision-making capabilities. Aircraft can be placed in deliberate conflict scenarios such as the "superconflict" setup, in which several aircraft are oriented in a circle, each heading towards a multi-collision in the center, which they must collectively negotiate safe passage around, without significant time penalties.

The AgentFly simulation tool represents a significant achievement in the ability to test automated conflict avoidance strategies against one another in numerous scenarios. However, the Capstone Team have recognized the lack of a suitable framework for the development of new ROR which fits the needs of NASA-LaRC. The previous works mentioned serve as a contextual basis for the algorithmic techniques that the Capstone Team anticipates having to accommodate in a framework designed to facilitate the development of ROR.

CONCEPTUAL DESIGN

The framework is a powerful concept in software development. A framework encapsulates many of the base functionalities in a wider class of systems, while providing numerous plug-ins and extension points to modify or extend some higher or lower level behavior. A successful framework design should consist of reusable components, which relieves the application developer from having to develop many of the lower level algorithms that are consistent within the underlying problem domain. The FAST framework is integrated into the agent-aircraft design with the intent of facilitating the rapid implementation and testing of new ROR. NASA developers will be able to build upon the provided framework with additional code for the ROR, allowing them to simulate a variety of autonomous flight procedures. In addition to this unique design capability, a comprehensive air traffic modeling, simulation, analysis and visualization environment will also be provided to NASA.

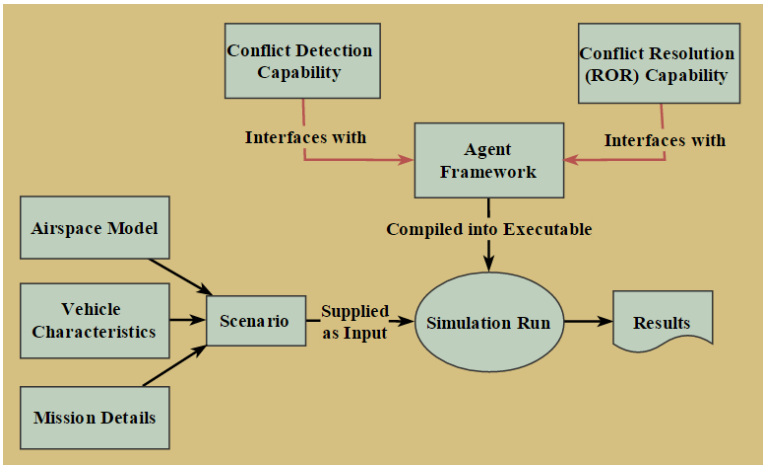

Figure 1: System Conceptual Design

Figure 1 illustrates the conceptual design behind the system under development. The system will be capable of simulating air vehicles interacting in an airspace – a bounded, rectangular cuboidal region with known takeoff and landing points. Each vehicle is equipped with a set of characteristics whose specification defines certain flight behaviors, such as rate of ascent from takeoff or preferred cruising speed. Each vehicle is also provided a mission which defines its intended flight path from a starting location. A scenario is described as a specific instance of vehicles, containing missions, interacting in a shared airspace.

In the absence of any autonomous decision-making capabilities, an aircraft in the simulation will simply fly its intended trajectory, ignoring any potential "loss of separation" with surrounding aircraft (SKYbrary Aviation Safety, 2016). However, an agent with enhanced capabilities could successfully maneuver through a conflicted region of airspace, possibly communicating with other agents to coordinate safe and efficient travel. The agent framework design provides interfaces where conflict detection and resolution capabilities can be selectively modified without breaking any of the code dealing with aircraft kinematics. After the agent framework code is compiled into the supporting project, any number of scenarios can be simulated. The results gathered from a simulation run can be used to assess the performance of a specific set of ROR.

SYSTEM ARCHITECTURE

In order to realize the conceptual design, the system architecture is composed of three modules: Input, Simulator, and Output. These modules are further composed of smaller subcomponents in order to achieve an even further modular architectural design. The Input module is made up of the Input Editor and three file repositories: the Airspace Repository, Vehicle Repository, and Scenario Repository. These four components are responsible for the development and creation of a scenario. After the creation of a scenario, the Simulator module is responsible for enacting the scenario, and recording relevant aircraft data while the simulation is running. Data from the simulation is stored inside of the Output Repository, which is accessed by the components of the Output module. These components consist of: the Data Analyzer, Visualizer, Simulator Repository, and Data Analyzer Repository. The Data Analyzer is capable of accepting a pair of associated scenario and simulation output files to assess the effectiveness of the ROR under testing. The Visualizer is capable of accepting an associated set of scenario, output, and analysis files for displaying data metrics graphs as well as an interactive recording of the flight simulation. A high-level overview of the system architecture is illustrated in Figure 2.

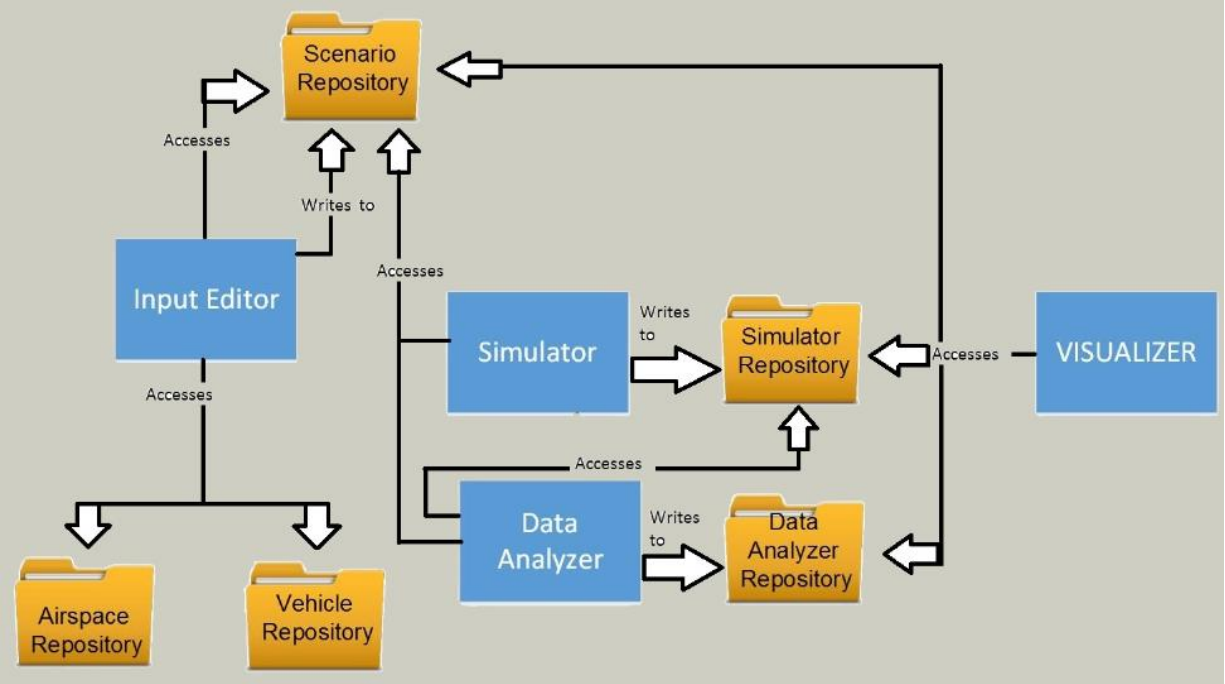

Figure 2: High-level System Architecture

Input Components

In order to facilitate the scenario development process, three repositories are set up to interact with the Input Editor. These repositories are: 1) Airspace Repository, containing a list of pre-defined airspace models to choose from when defining a new scenario; 2) Vehicle Repository, which contains a list of available vehicle models to choose from when defining a scenario; and, 3) Scenario Repository, which will store fully-defined scenario models, including vehicle mission specifications, for execution.

The Input Editor component is responsible for providing the user with valid scenario files for the Simulator to run. The user has the option of creating a new scenario file or loading an existing one into the editor via graphical user interface. Once a file is loaded into the viewing window, the user is able to easily edit the scenario, and then save that information into the Scenario Repository for later execution. The Vehicle Repository and Airspace Repository aid the user in the scenario development process by providing references to pre-defined airspace and vehicle models to include in the scenario model definition.

Simulator Components

The Simulator component of the system is responsible for simulating a vehicle flight scenario. A valid scenario file, such as one generated from the Input Editor, is required by the Simulator; otherwise, there is no guarantee that the simulation will behave correctly. The scenario file is used to initialize the environment, which consists of the airspace model specification, including dimensions, takeoff and landing points, and any static features such as buildings. The scenario file is also used to initialize the number and type of vehicles in the environment, along with their unique flight plans. The Simulator is designed to execute a time-stepped, agent-based simulation, producing individual vehicle state changes. Each time a vehicle changes its state, a time-stamped recording of that vehicle's state information is logged into a common output file for that simulation run. This output file is stored into the Simulator Repository for later analysis and visualization.

In alignment with the ADAPT concept, the Simulator component provides an interface to the framework into which NASA can insert code to further define an agent model, in terms of both conflict detection and resolution capabilities. The Capstone Team will integrate a simple kinematic model to provide each agent with motion capability for proof of concept, although the agent model design will not preclude a more realistic kinematic or dynamic model from being

integrated later on. In the future, NASA may also want to impart the airspace model with environmental effectors, such as weather. However, this capability is beyond the scope of the current project. An overview of the Simulator architecture is shown in Figure 3.

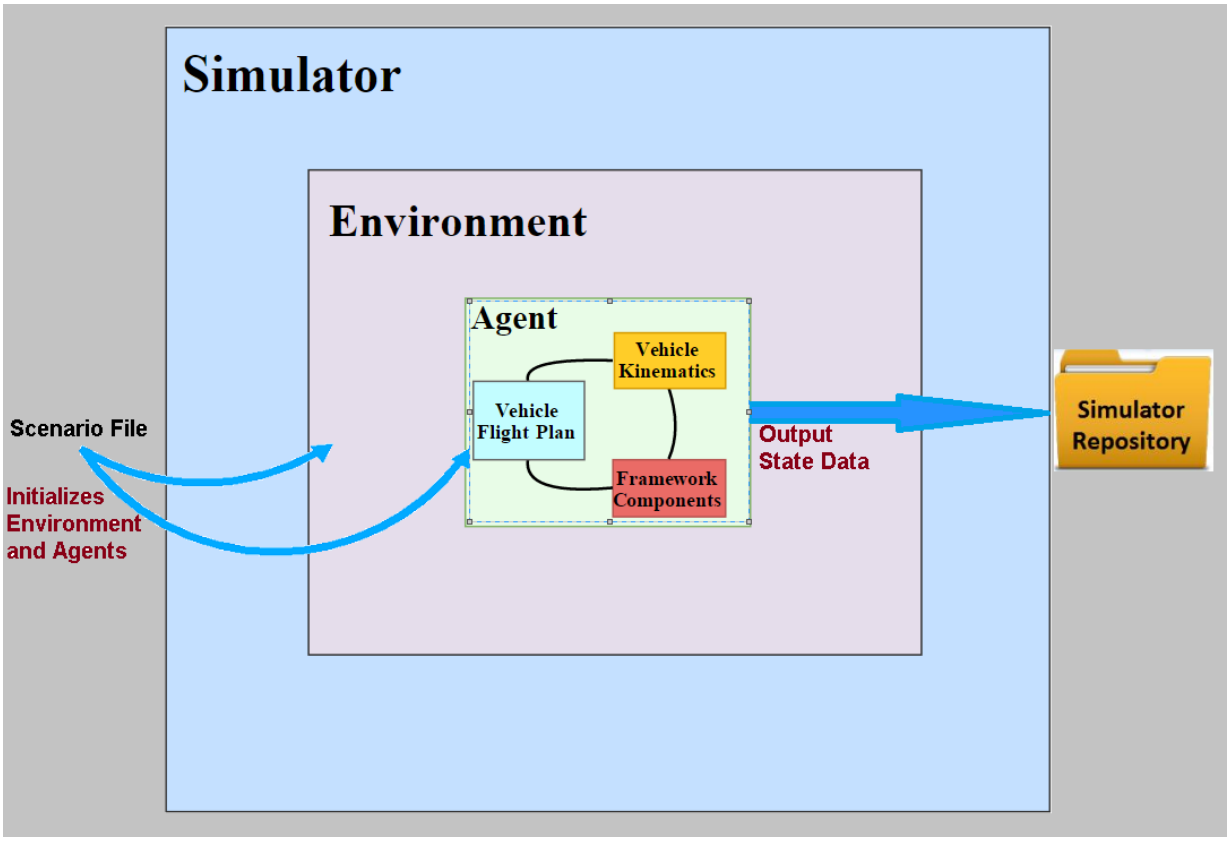

Figure 3: Simulator Architecture

The underlying paradigm for this simulation system was chosen as an ABM, due to several abstractions which can be drawn from the physical model of autonomous flight systems. The physical model containing autonomous vehicles maps logically to autonomous agents in the ABM. The physical model consists of an airspace, and any additional 3-dimensional structures, which also maps logically to the environment component of an ABM. Vehicle behaviors, including communication, navigation, conflict detection and resolution algorithms map to the autonomous behavioral component of an agent.

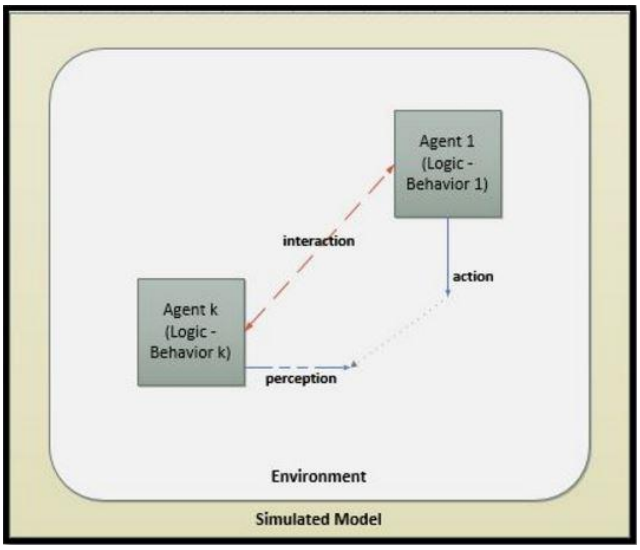

Figure 4: Agent Interaction

With this mapping established between physical and simulated systems, the environment is comprised of all of the agents in the airspace, as well as any other non-agent structures that influence the agents' behavior. As the agents move about the environment, their global positions are reflected onto the environment. Agents are capable of sensing and perceiving the relative distances to other objects in their local surroundings. In addition, agents are capable of communicating with one another directly, using the information they sense from the environment to establish communication channels. These two methods of agent interaction are illustrated in Figure 4.

Output Components

The Simulator Repository contains a set of files that the Simulator produces. Simulation output will be in the form of time-stamped state changes by individual vehicles. Output files are provided to the Data Analyzer for processing useful data metrics, the results of which are stored into the Data Analyzer Repository. Output files are also provided to the Visualizer to show the flight paths of individual vehicles.

The Data Analyzer accesses the Simulator Repository for the output files containing vehicle state change data produced by the Simulator. This component calculates aggregate data metrics through any of three subcomponents: a system-wide analyzer, a vehicle subset analyzer, and an individual vehicle analyzer. The system-wide analyzer produces aggregate data metrics for the entire system, the vehicle subset analyzer computes aggregate metrics for specific vehicle subsets specified by the user, and the individual vehicle analyzer produces metrics for individual vehicles. An information collector subcomponent existing within the analyzer will store performance metrics from each data evaluation module into appropriately labeled files within the Data Analyzer Repository.

The Visualizer is responsible for displaying graphical information about a completed simulation run. The Visualizer contains three subcomponents: an input file reader, a flight animator, and a performance metric data display. The input file reader accepts a pair of files from the Scenario Repository and Simulation Output Repository. Data is fed into the flight animator subcomponent to visualize the desired and actual trajectories for individual vehicles. The data display subcomponent is capable of visualizing data metrics computed by the Data Analyzer. The Visualizer outputs no files; visualizations can be replayed by reusing the input files.

AGENT ARCHITECTURE

The agent architecture is a central focus in the design of the system, as it largely governs the design of the simulation architecture, and in several ways is coupled with the design of the supporting architectural components. The agent architecture is constructed to organize the flow of logic throughout the agent for each time step in the simulation.

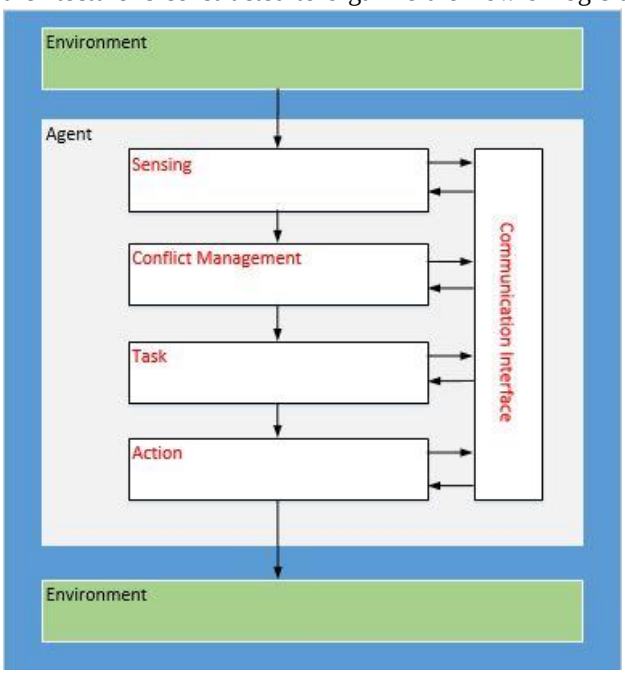

Figure 5: Agent Architecture

Agents are composed of four vertical layers which contain unique, yet interdependent capabilities in self-managing flight. These capabilities are: sensing information from the environment; conflict management, for finding the solution to the safest and most efficient flight path; task planning, to map out a series of basic actions to achieve the desired flight path; and executing the next action for a given task, eliciting a change to the state of the environment on each iteration of the simulation. Communication between agents is enabled horizontally across each layer via interfaces on each layer, and communication is only supported across layers of the same type. The agent architecture is illustrated in Figure 4, along with arrows indicating the flow of information.

The Sensing Layer is responsible for pulling relevant information about agents from the environment, in addition to establishing communication pathways between nearby agents who may potentially be in conflict via the Communication Interface. The identified set of local agents is then passed to the Conflict Management Layer for processing, which will determine if any agents in the set actually are in conflict. While the agent has not attempted to resolve all known conflicts, this layer will invoke its ROR with relevant input in an attempt to find an optimal trajectory. So long as there are no conflicts, the layer's secondary responsibility is to maintain the agent's current course and correct any deviation from the optimal trajectory. Resolving conflict, correcting course, and maintaining course each constitutes a decision which is then sent

to the Task Layer. The Task Layer is then prompted for a set of instructions to gather from the Action Layer once a decision has been made.

AGENT FRAMEWORK

NASA-LaRC have identified that the primary objective of the system should be to allow for quick and easy use, in order to best complement their efforts in the development and testing of ROR. The Capstone Team have decided that a full-stack framework, in which the ROR can be easily modified or replaced, is the best design alternative to meet this request. Where the user of a full-stack framework will incur some penalty is in the flexibility and scope of potential application design choices. However, since the application domain that this framework is intended to be used for is relatively self-contained, some limitation of flexibility is considered acceptable. Most importantly for efficient utilization of the system, there will exist a structure with well-defined interfaces for NASA code to access the necessary low-level features of the agent model, including state information, motion, and communication.

The agent framework is designed to accommodate two plug-in subcomponents: the Conflict Resolution subcomponent and the Conflict Detection subcomponent. The agent is incapable of avoiding conflict unless both detection and resolution capabilities are specified. However, the agent architecture is functional whether or not these capabilities are provided. As a demonstration of the agent framework's positive design characteristics, the Capstone Team will provide a basic conflict detection model of their own design, even though it is expected that NASA will overwrite this piece of software with their own functionality. In summary, the proposed framework solution is designed to curtail many of the issues NASA would otherwise face when integrating key conflict management features into the broader simulation system.

CONCLUSION

The Capstone Design Team understands that NASA's systematic approach to testing ROR requires an architecture that allows for flexibility, reusability, and efficiency of use. The system architecture captures the work flow pattern that will allow for NASA to operate concurrently in each stage of the ROR development process, so that no time is wasted creating scenarios, programming ROR capabilities, or analyzing their effectiveness. The strength of the framework approach is that it provides the user a reusable piece of software which already has implemented in it many of the lower level algorithms that are consistent within the problem domain. The framework within the FAST architecture will allow NASA to focus their efforts on developing the algorithms for managing conflicts between vehicles. NASA will be able to simulate numerous conflict scenarios under reasonable time constraints, and will be able to tightly control every aspect of the ROR development process with the provided suite of modeling, simulation, analysis and visualization tools which spans the full system architecture.

ACKNOWLEDGEMENTS

The Capstone Team would like to thank Drs. Jim Leathrum and Roland Mielke of the Department of Modeling, Simulation & Visualization Engineering at Old Dominion University for their continual guidance and support. Special thanks also go to Dr. Bryan Barmore and Steve Velotas of the National Aeronautics and Space Administration Langley Research Center (NASA-LaRC), Aeronautics Research Directorate for providing this unique opportunity to the 2015-2016 undergraduate class, as well as for their support as subject-matter experts.

REFERENCES

Albaker, B. M., & Rahim, N. A. (2010, September). Unmanned Aircraft Collision Avoidance System Using Cooperative Agent-Based Negotiation Approach [Electronic version]. International Journal of Simulation: Systems, Science & Technology, 11(5), 1-7.

Federal Aviation Administration. (2015, January 12). Navigation Programs - History - The Transition from Ground-Based Navigation Aids to Satellite-Based Navigation. In Federal Aviation Administration. Retrieved October 8, 2015.

Frazzoli, E., Mao, Z., Oh, J., & Feron, E. (2000, March 31). Resolution of Conflicts Involving Many Aircraft via Semidefinite Programming [Electronic version]. Journal of Guidance, Control, and Dynamics, 24(1), 79-86. doi:10.2514/2.4678

"Loss of Separation." - *SKYbrary Aviation Safety*. N.p., n.d. Web. 05 Feb. 2016.

Pechoucek, M., & Sislak, D. (2009, January 27). Agent-Based Approach to Free-Flight Planning, Control, and Simulation [Electronic version]. Intelligent Systems, IEEE, 24(1), 14-17. doi:10.1109/MIS.2009.1

Richards, A., & How, J. P. (2002). Proceedings of the 2002 American Control Conference (Vol. 6, pp. 1936-1941). Danvers, MA: American Automatic Control Council.

Schulz, R., Shaner, D., & Zhao, Y. "Free-Flight Concept," *Proc. AIAA Guidance, Navigation and Control Conf.*, AIAA Press, 1997.