Proceedings of the 5th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, Madrid, Spain, February 15-17, 2006 (pp453-457)

An Agent-based Heterogeneous UAV Simulator Design

MARTIN LUNDELL[1], JINGPENG TANG[1], THADDEUS HOGAN[1], KENDALL NYGARD[2]

[1]Math, Science and Technology
University of Minnesota Crookston
Crookston, MN56716
UNITED STATES

[2]Department of Computer Science and Operations Research
North Dakota State University
Fargo, ND 58105
UNITED STATES

http://www.cs.ndsu.nodak.edu/~nygard

Abstract: To facilitate the evaluation of alternative mission planning models for teams of Unmanned Air Vehicles (UAVs), a multi-agent simulation system was designed and developed. The system, referred to as Ether, is designed to be more generally applicable to a wide variety of agent-oriented systems. The simulator design has the flexibility to support a wide variety of mission planning models, yet provides infrastructure that supports rapid prototyping.

Key-Words: - Simulation, Agents, UAV, Distributed systems

1 Introduction

To facilitate the evaluation of alternative mission planning models for teams of Unmanned Air Vehicles (UAVs), a multi-agent simulation system was designed and developed. The system, referred to as Ether, is designed to be more generally applicable to a wide variety of agent-oriented systems. The simulator design has the flexibility to support a wide variety of mission planning models, yet provides infrastructure that supports rapid prototyping. We describe the requirements of the simulator from the perspective of the application to UAVs. The design is described in detail, including principles followed to develop a simulator that is easy to use, yet provides great modeling expressive power and flexibility.

2 Simulator Requirements

2.1 UAV research

Scientific and Engineering advances in wireless communication, sensors, propulsion and other areas are rapidly making it possible to develop Unmanned Air Vehicles (UAVs) with sophisticated capabilities. Deployed individually, UAVs offer the potential to search for, detect, and destroy enemy targets in relatively complex environments. They potentially reduce risk to human life, are cost effective, and superior to manned aircraft for certain types of missions. It is desirable for UAVs to have a high level of intelligent autonomy, to carry out mission tasks with little external supervision and control. This raises important issues involving tradeoffs between centralized control and the associated potential to optimize mission plans, and decentralized control with great robustness and potential to adapt to changing conditions. We report on a simulator designed to provide flexibility for exploring a wide variety of cooperative control models at the mission planning level for heterogeneous UAVs. We focus particularly on designs that hold the potential to simulate near real-time systems.

2.2 The UAV simulator need

An agent-based simulation system is essential for evaluating alternative control models for UAV missions. The simulator provides the capability of comparing the control algorithms with common metrics that measure efficiency, effectiveness, and cost. We wish to evaluate the alternatives based on the dynamic battlefield in real-time [1] [2]. The

system also provides visualization for the simulation of various UAV missions. The system uses a multi-level approach to modeling a UAV in order to support approaches to mission planning. This is accomplished using an agent system through which different levels of control can communicate [2,3,4].

2.3 Modeling challenges

A mission planning simulator should be general enough and flexible enough to incorporate multiple control algorithms. Established software engineering practices should be followed in the implementation. More specifically, the simulator must have an algorithm control layer which can easily incorporate different decision support algorithms for task allocation, such as Partially Observable Markovian Decision Process (POMDP), Fuzzy Logic, Bayesian Decision Analysis, and Rough Set Theory for UAVs. Another challenge is the desirability of decoupling the simulation from the visualization system. Visualization is typically an integral part of a simulator, and in many cases is tightly coupled. Such designs significantly complicate the design, and can easily result in unjustified time spent on visualization rather than the control algorithms themselves.

2.4 Purpose of the simulator

The Computer Science Department at North Dakota State University has a long history of work in mission planning for cooperating UAVs. Models developed include parallel sweep search for reconnaissance; synchronized, multi-point strike; layered, multiple-perimeter asset patrolling with threat elimination; mobile target tracking and surveillance; and forward air controller with available strike vehicles. In conjunction with these models, several algorithms have been developed to model decision-making by autonomous UAV's including route planning, target assignment, and decisions to strike or not strike a target.

These areas of investigation have led to the development of several simulators, each designed to demonstrate and evaluate the performance of a particular UAV model. There are several problems that arise. First, there is considerable duplication of effort as many individuals invest time in simulator development. Second, it makes it difficult to compare the model against each another in an appropriate way.

The project that we describe is focused on developing a general purpose simulator for to comparing and contrasting different UAV models. This strategy will result in a simulator which can also be used for a variety of models outside the realm of UAV research.

3 Simulator Features

3.1 Functions of the simulator

- Allow the user to configure their simulation by defining agents, their supported interactions, and termination conditions.
- Operate in batch mode over a range of parameters specified by the user.
- Run in a visualization mode where the user can see the simulation in real-time or review prior simulations.
- Support distributed computing in a way that scales with the complexities of the mission model.

3.2 Architecture overview

Figure 1 illustrates the architecture of the simulator that is named Ether. Components in the architecture are detailed in the following sub-sections.

The Core is responsible for starting up and initializing components to be used in support of simulations. This includes loading configuration files, initializing the User Interface and logging capabilities. The most interesting thing the Core does from a design perspective is create the EmodHandler. Emods (Ether Modules) are discussed in greater detail later, but they are essentially the mechanism through which Ether gains its flexibility, facilitated by the EmodHandler in the Core. Interfaces for Agents, InteractionLogic, and the user interface (EtherUI) provide the mechanisms to extend the simulator to meet individual simulation needs.

The Environment in the simulator is implemented as an Emod and contains references to all agents and interaction logic. An Observer of the environment records state and pushes that to the database for analysis and/or visualization.

For more complex simulations there may be a need to run more than one Ether process. This capability is also implemented as an Emod.

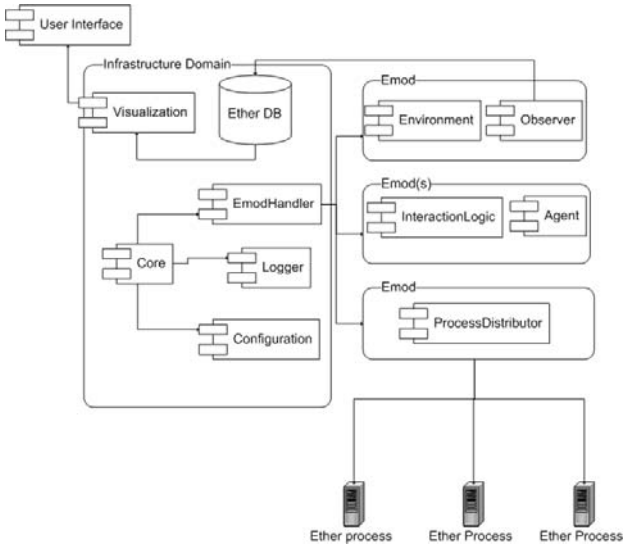

Fig. 1 Ether Simulator Architecture

3.2.1 Characteristics of the Core

3.2.1.1 Initialization

The simulator core is responsible for initializing the users interface, logging capabilities, persistent storage, and the simulation component manager (EmodHandler). When the simulator core is started, it reads an XML file describing the simulation to run, the properties of the environment, the condition upon which the simulation is to end, the ways in which agents can interact with one another, and specifying which agents will be resident in the simulation. All agents are created through reflection, where the class names of the objects to be instantiated are loaded from the simulation configuration file. In this way, the simulator is not dependant on all agents being defined at runtime. Additional agents, restrictions, logic patterns, and interactive components can be built outside of a running simulator and inserted at runtime.

3.2.1.2 Persistence

One of the major functionalities of the simulator core is to provide the simulation environment with access to persistent storage. The principal method of data storage on disk is via a DBMS. Any information that should be stored as XML for transmission or analysis by other tools is to be extracted from the DBMS and rendered into XML after a simulation has run. The use of a DBMS for persisting data to disk at runtime was chosen because it is much faster than rendering XML during the simulation.

3.2.2 Environment

The environment is the realm in which agents execute. It handles all agents and interactions among agents as specified in their corresponding Emod. It provides five principle services:

1. Creates/manages the flow of simulated time.
2. Enforces the laws of physics.
3. Provides methods for underlying agents to interact with each other and the environment.
4. Provides the logging interface.
5. Determines when the simulation is to be terminated.

3.2.2.1 Flow of time

The purpose of the environment is to enforce restrictions which apply to all agents of any type. One of these restrictions is the flow of time. All of the agents must act in the same time frame and cannot be allowed to perform more or less work than any other agent in one cycle of the clock. For example, if two objects are moving at identical velocities, both must travel the same distance in a single clock cycle.

Time is to be measured in seconds, and is iterated at a specified rate. The rate specifies what fraction or multiple of a second will be used for each execution step. The clock allows each agent to simulate that fraction or multiple of a second in a sequential manner, allowing one agent to simulate, then the next, and so on. For the purposes of this document, the use of a small fraction of a second per iteration is considered a high rate, while the use of a large faction of a second per iteration shall be considered a low rate. The higher the rate, the smoother and more accurate the simulation will be. A lower rate will allow the simulator to complete quickly, but will lower the accuracy achieved.

The simulator contains a collection of agents that are currently running under the environment. With each iteration of the clock, each agent is instructed to simulate its actions over time increment issued by the clock. For example, if an agent is traveling at 10m/s, and the rate is 1 millisecond, that agent will travel 10 millimeters during a single iteration. After an agent is processed, it returns any interactions that it performed to the scheduler. These interactions are verified and immediately executed. In each execution cycle of the clock, all Agents have an opportunity to set their state before they interact. This enforces indifference to the ordering of the Agents in terms of the behavior of the simulation.

3.2.2.2 Maintaining the laws of physics

The environment simulation is responsible for imposing the laws of physics on the underlying agents. In turn it is the responsibility of each agent to make use of the simulated physics in order to control its position in the environment. An agent's InteractionLogic components include acceleration

for any agent which is able to move. (Velocity is not an InteractionLogic component, as it can only be modified through acceleration.) If gravity at 9.8m/s^2 is applied to an environment in a downward direction, an agent must be able to report to the environment the production of a 9.8m/s^2 acceleration in an upward direction in order to make no change to its movement.

Some agents are not affected by gravity, but may be affected by other laws of physics. For example, a radio transmission creates an agent, which is the transmission itself. This agent represents the presence of radio waves in the environment and likewise is not affected by gravity. Accordingly, a radio transmission may not have an acceleration InteractionLogic component. However, the environment chooses which InteractionLogic components of an agent are affected by the simulated laws of physics. Thus, any InteractionLogic component could be affected by the laws of physics, and any agent that wishes not to be affected by laws of physics must not implement the related InteractionLogic component. For example, a brick may be built without an acceleration InteractionLogic component, so that the brick is not affected by gravity. However the brick will then not be able to move, since the environment is control of the ability of all agents to change their location. Similarly, a balloon that implements an InteractionLogic component of "size" will not necessarily be affected by the laws of physics unless a law is implemented which affects "size," such as pressure.

3.2.2.3 Agent interaction

The environment is responsible for all interactions between agents. Fig. 2 illustrates the communication among agents, the environment, and the interaction logic in a typical situation. Agents cannot know what other agents exist at their build time. Instead, agents are aware of all InteractionLogic components that they can affect at their build time.

When the environment issues an execution cycle to an agent, that agent determines if it will interact with another agent during that cycle. If it does interact with another agent, it will return to the environment the InteractionLogic component it will change and which agent that change applies to. (Multiple interactions can be reported from a single execution cycle.) The environment then imposes that change on the affected agent.

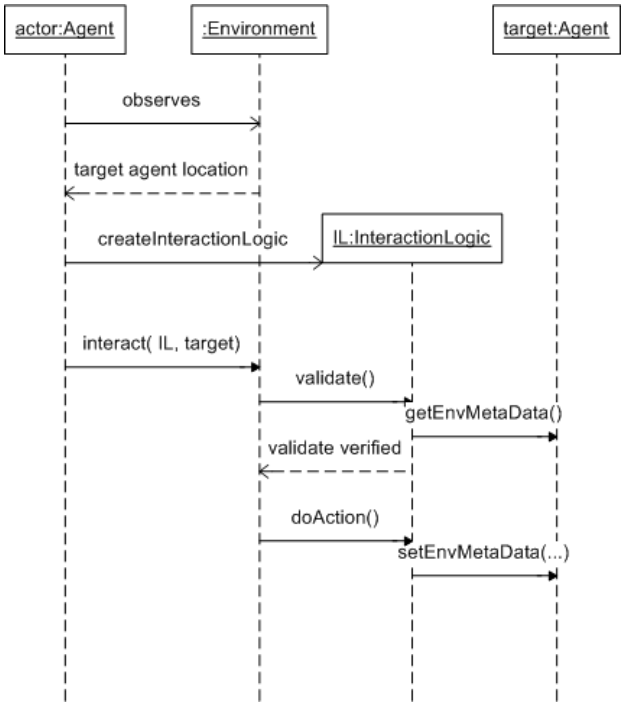

Fig. 2 Agent Interaction Sequence Diagram

3.2.2.4 Logging

The environment provides a standard mechanism by which agents are able to log or record what has changed during each of their executions. Also, the environment supports logging rules under which it can extract data from agents for logging. This was developed during design so that the environment and agents are not logging the same information.

3.2.2.5 Termination condition

The termination of a simulation can be a result of simulation objective being realized and/or a specified time expiring.

3.2.3 Emods

For the purpose of establishing a flexible development environment that is highly decoupled, agents and the environment are separately developed and interlaced at runtime through a simple pluggable components called Emods (Ether Modules).

When the simulator is run it has an Emods directory where Emods reside. When added, or touched, the simulator loads the class files and descriptors and adds the defined component to the simulation environment. The simulator implements a special classloader that allows Emods to be refreshed without restarting the simulator, even when the simulation is running.

An Emod is simply a folder with the extension .Emod, and has the following structure:

- Example.emod/
- Example.emod/emod.xml -> Descriptor file

- Example.emod/icon.png -> Graphical representation to be used within the simulator (optional)
- Example.emod/classes/ -> classpath root for Emod classes

The descriptor file contains information regarding the Emod, such as what type of agent it is, the name of the class to invoke, whether it is renderable, and its possible startup parameters.

3.2.4 Visualization

The simulator supports drawing of agents done with standard graphics primitives. A future version with Scalable Vector Graphics is planned. The visualization component supports full pan and zoom of the environment independent of the visualization data. The visualization component inputs the visual representation of the environment in physical units (e.g., meters) and converts it into a visual representation in logical units (pixels).

The visualization system has two components. A renderer is contained within each agent and provides instructions for drawing that agent. A visualizer which inputs the data generated by the agents and presents it on the screen. The rendering instructions for an agent are provided by the Environment as an XML stream. Potential components that can process this XML stream include a visualizer to display the data, a file writer to store the data, and a network socket to transmit the data to remote visualizers.

3.2.5 Distributed processing

The simulator supports distributed processing. When running simulation sets with many combinations of the available parameters, a large number of simulations may be necessary. To complete these simulation sets in a reasonable amount of time, the simulator can send simulations to remote machines for processing. A control simulator creates the jobs and distributes them to remote nodes. Remote machines are transient, so that if one or more is disconnected the entire set of simulations is not lost. Simulation result data are be collected from the nodes by the control simulator and processed as if the jobs were run on the local simulator.

4 Conclusions

The Ether simulator allow for comparison of alternative agent-based algorithms. For example, we can provide the same environment between multiple simulation runs and change only the decision making algorithm in the agent's Emod. It is possible to isolate individual parameters for investigation. A consequence of simulator flexibility is the burden placed on the users of the simulator to develop their own Emods. Over time an Emod library of agents, agent behaviors, and interaction logic to promote faster simulation builds will be developed and made available.

While the Ether simulator was motivated UAV research, it is not UAV specific. Ether can be used as a general purpose agent-based simulation platform. The users would only need to create their own agents and interactions and package them in the Emod structure.

The first major experiments with the Ether simulator include comparisons among Fuzzy Logic, Bayesian Decision Analysis, and Rough Set theory algorithms for task allocation and decision making for missions with multiple UAVs.

References:

[1] Altenburg K, Schlecht J, Nygard KE. (2002). An Agent-based Simulation for Modeling Intelligent Munitions", Athens, Greece: *Advances in Communications and Software Technologies*, pp. 60-65.

[2] Brooks, R.A., "A Layered Control System for a Mobile Robot," IEEE Journal of Robotics and Automation, Vol. 2:1, pp. 14-23, 1986.

[3] N. Huff, A. Kamel, and K. Nygard, An Agent Based Framework for Modeling UAVs. In Proc. of the 16th International Conference on Computer Applications in Industry and Engineering (CAINE03), 2003.

[4] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. MASON: A Multi-Agent Simulation Environment. Forthcoming, *Simulation Journal*.