AN ACTOR-BASED SIMULATION FOR STUDYING UAV COORDINATION

Myeong-Wuk Jang, Smitha Reddy, Predrag Tosic, Liping Chen, Gul Agha
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
E-mail: { mjang, sreddy1, p-tosic, lchen2, agha } @cs.uiuc.edu

KEYWORDS

Actor, Simulation, Unmanned Aerial Vehicle (UAV), Coordination.

ABSTRACT

The effectiveness of Unmanned Aerial Vehicles (UAVs) is being increased to reduce the cost and risk of a mission [Doherty et al. 2000]. Since the advent of small sized but high performance UAVs, the use of a group of UAVs for performing a joint mission is of major interest. However, the development of a UAV is expensive, and a small error in automatic control results in a crash. Therefore, it is useful to develop and verify the coordination behavior of UAVs through software simulation prior to real testing. We describe how an actor-based simulation platform supports distributed simulators, and present three cooperation strategies: self-interested UAVs, sharing-based cooperation, and team-based coordination. Our experimental results show how communication among UAVs improves the overall performance of a collection of UAVs on a joint mission.

1. INTRODUCTION

The effectiveness of Unmanned Aerial Vehicles (UAVs) is being increased to reduce the cost and risk of a mission [Doherty et al. 2000]. Some military UAVs, such as the Predator and the Global Hawk, were already used during the wars in Afghanistan and Iraq. Decreasing size of the UAVs and increased demand for more intelligent and autonomous behavior of UAVs are paving the way for consideration of a group of UAVs performing a joint mission. While the cost of UAVs is lower than that of real planes, the development cost of a UAV is still very high, and a small error in automatic control may result in a crash. Therefore, when we consider a large number of UAVs working together, it is necessary to design and verify the behavior of UAVs through software simulation prior to real testing.

Many simulators have been developed as single process simulators. However, a single process simulator has several limitations. First, the performance of a simulation depends on the computational power of one computer. Second, a single process simulator has an extensibility issue when a special component requires its own specific process. For example, if we want to simulate the coordination behavior of many virtual UAVs with a few real UAVs, each real UAV is working as an independent process. In this kind of simulation, a single process simulator cannot work well. Therefore, a concurrent object-based distributed simulator provides a better simulation environment.

It is commonplace to say that human beings are disposed to cooperate. Biology and ethology show that "kin-altruism" and "reciprocal-altruism" can ground cooperative behavior in animals, such as wolves surrounding prey, termites nest building, and birds flocking. Drawing a parallel, intelligent UAVs that cooperate with one another are of high interest for their ability to search, detect, identify, and handle targets together. The old age tenets of pre-planning and central control have to be reexamined, giving way to the idea of coordinated execution. In this paper, we describe and analyze three different strategies to coordinate tasks among UAVs in a dynamic environment to achieve their goals.

The outline of this paper is as follows. Section 2 sketches a simulation scenario and explains basic concepts about the actor model and the metrics in our simulation. Section 3 describes architecture for our simulation, and three cooperation strategies for a joint mission are presented in Section 4. Section 5 explains our implementation and experimental results. Then, in Section 6 and 7, we discuss related work and our future work. Finally, we conclude this paper with a summary of our simulation framework and our major contributions.

2. TERMINOLOGY

2.1 UAV Simulation Scenario

Prior to embarking on the architecture of our UAV simulator, we present a simple scenario in order to explain the meaning of basic terms. The application of our simulation is a UAV surveillance mission. For example, 50 UAVs might be launched into a certain area by *Ground Control System* (GCS) to detect targets in the area. For example, *targets* may be civilians to be rescued. In the simulation, UAVs have the autonomy to perform their mission without interaction with the GCS, except during the initial stage when message exchange is necessary to get each UAV started by sending them some default air routes. When UAVs are launched, the UAVs do not have any information about locations of targets. However, each

UAV is equipped with some sensors which can detect objects within the certain range. We assume that all UAVs start from the same location, called an *air base*. Controlling the sequence of takeoffs and landings of UAVs is managed by the control center, called *Air Base System* (ABS). The main task of a UAV is to detect locations of targets in a mission area and investigate them. Therefore, even though they navigate according to the given air routes, they can change their trajectories to handle targets once they detect those targets. In addition, when UAVs encounter *obstacles*, such as tall towers or airplanes, they should change their air routes to avoid a collision. Therefore, in our UAV simulation, there are five types of important components: Ground Control System (GCS), Air Base System (ABS), Unmanned Aerial Vehicles (UAVs), targets, and obstacles.

2.2 Actor

Our UAV simulator is based on the *Actor system*, a concurrent object-based distributed system, and hence, we use the actor model to describe each component in the simulation. An *actor* is a self-contained active object which has its own control thread and communicates with other actors through asynchronous message passing [Agha 1986; Agha et al. 1997]. In addition, an actor can create other actors, just as an object can create other objects. In our UAV simulator, each component, such as a UAV or a target, is implemented as an actor. Since these components in real situations operate concurrently and communicate with one another, their behavior can be captured very well by the actor model. Each software component in the simulation progresses its state independently of the progress of others in response to the environment information gathered either through its own sensor or by communicating with others.

2.3 Attractive Force Value and Utility Value

In our UAV simulation, each target has its own value. This value could be interpreted in several different ways. The value might correspond to the number of soldiers or the importance of a building. Also, we can consider this value as the time required to investigate a target by a UAV. For the simplicity of our simulation, we use a single numeric value instead of symbolic information or time information about a target.

In our simulation, we make the following assumptions. A UAV handles only one target at a time, although the UAV holds and manages information about several targets. In the advent of multiple targets to be handled, the UAV should select one of them. For this purpose, a UAV uses the attractiveness function to decide on a target. The *attractiveness function* maps the value of a target to the *attractive force value*, which represents a UAV's preference. This function depends on the value of the target and the distance between itself and the UAV, and is used to select the best target as follows:

$$\Theta_i(t) = \arg \max_j \left\{ \frac{\Pi_j(t)}{\|x_i(t) - \psi_j(t)\|} \right\}$$

where $\Pi_j(t)$ denotes the value of target j at time t, $x_i(t)$ is the location of UAV i at time t, and $\psi_j(t)$ is the location of target j at time t. If target j is stationary, $\psi_j(t)$ is always the same regardless of time. The value between braces is called the attractive force value of target j, and $\Theta_i(t)$ returns the index of the target that has the maximum attractive force value to UAV i at time t.

As a UAV approaches a target, the UAV starts consuming the value of the target once the UAV is within a certain distance of the target. We call the value consumed by the UAV the *utility value*. The *utility value function* and the *target value function* of the target i at time step $t+1$ are defined as follows:

$$U_i(t+1) = \Pi i(0) - \Pi_i(t+1)$$

$$\Pi_i(t+1) = \max\{\Pi_i(t) - d \cdot n_i(t), 0\}$$

where $U_i(t)$ means the utility value of the target i at time t, d is a discount factor, and $n_i(t)$ is the number of UAVs which are near to the target i at time t. Therefore, in our simulation, when several UAVs are within the range of a target, the value of the target is consumed more quickly.

After a UAV reaches a target, it will fly around the target until the whole value of the target is consumed, either by the UAV alone or in conjunction with a group of UAVs. In our UAV simulation, one purpose of collective behavior of UAVs is to maximize the accumulated utility value within as short a time as possible. Here, the *accumulated utility value* means the whole value of targets consumed by all the UAVs.

3. SIMULATION ARCHITECTURE

Our distributed simulation is comprised of three layers: user interface, UAV simulator, and actor-based distributed platform (Figure 1). The *user interface layer* consists of two programs: Configuration Interface Program and Simulation Viewer. *Configuration Interface Program* provides an easy means of defining important attributes for the simulation. *Simulation Viewer* is a tool to check and verify the simulation results. All task oriented components, such as UAVs and targets, and simulation oriented components, such as Simulation Control Manager (SCM) and Active Broker (AB), are implemented as actors in the *UAV simulator layer*, which will be further explained in section 3.2.2. Each actor has its own thread to progress its state. The thread execution and communication of actors are

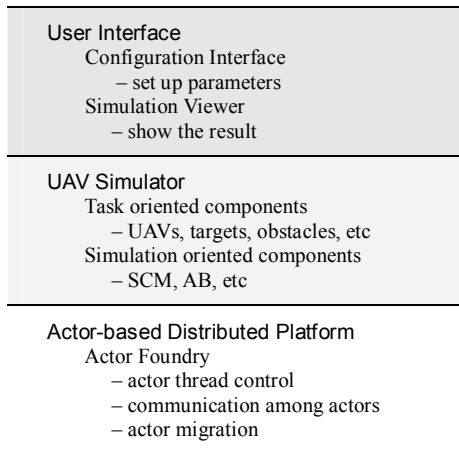controlled by the Actor Foundry, an *actor-based distributed platform*.

User Interface
    Configuration Interface
       – set up parameters
    Simulation Viewer
       – show the result

UAV Simulator
    Task oriented components
       – UAVs, targets, obstacles, etc
    Simulation oriented components
       – SCM, AB, etc

Actor-based Distributed Platform
    Actor Foundry
       – actor thread control
       – communication among actors
       – actor migration

Figure 1: Three-layered Architecture for Distributed Simulation

The actor-based distributed platform is a middleware to support several distributed applications and is not tailor made for a specific simulation, such as a UAV simulation. The UAV simulator defines specific behaviors of UAVs, but does not include all the parameters to test and verify a behavior. These parameters are defined in user interface programs by a user and used in the UAV simulator. The functions of each layer are explained in detail below.

3.1 Actor-based Distributed Platform

The Actor Foundry is implemented in the Java programming language, and supports actor execution, communication between actors, and actor migration [Astlery 1999; Clausen 1998].

In the Actor Foundry, an actor is created by another actor or by a user. When an actor is created, the actor name of the new actor is returned. This name would be used to refer to the receiver actor in message passing or deliver the reference of another actor to the receiver actor. The actor name is unique in the actor world. Therefore, even though an actor migrates from one host to another, the name is always transparent to other actors, and hence, other actors can continuously use the same name to refer to the given actor irrespective of that actor's current location, thereby providing a means for location transparency.

An actor in the Actor Foundry is running as a Java thread, and an actor communicates with other actors through asynchronous message passing. This is the main difference between the Actor Foundry and other object-based distributed platforms, such as CORBA and DCOM [Grimes 1997; OMG 2002]. In other object-based distributed platforms, one thread control is assumed: when an object is called by another object, the caller object is blocked until the called object returns the thread control. In the Actor Foundry, since every actor has its own control thread

to perform its operation and communicates with others through the asynchronous communication, the execution of an actor does not depend on those of others. Due to these features, we can easily use the power of distributed systems. Simulation components implemented as actors run on different computers independently, and they can communicate with others through the unique actor name, even though the distributed platform migrates some components from one host to another.

When distributed components interact with each other through asynchronous communication, analyzing the delivery sequence of communication messages is burdensome because asynchronous communication does not guarantee the message delivery order requirements, such as FIFO order, causal order, or total order [Hadzilacos and Toueg 1993]. Our distributed platform makes a log for message passing among actors, so that users can easily analyze the delivery sequence of messages.

3.2 UAV Simulator

All simulation components in our UAV Simulator can be classified into two categories: task oriented components and simulation oriented components (Figure 2). Task oriented components simulate objects in real situations. For example, a UAV component maps to a UAV object in a real situation while a target component maps to a target object. For the purpose of simulation, we need some virtual components, such as Simulation Control Manager and Active Broker. The following sub-sections explain these two categories of components in detail.
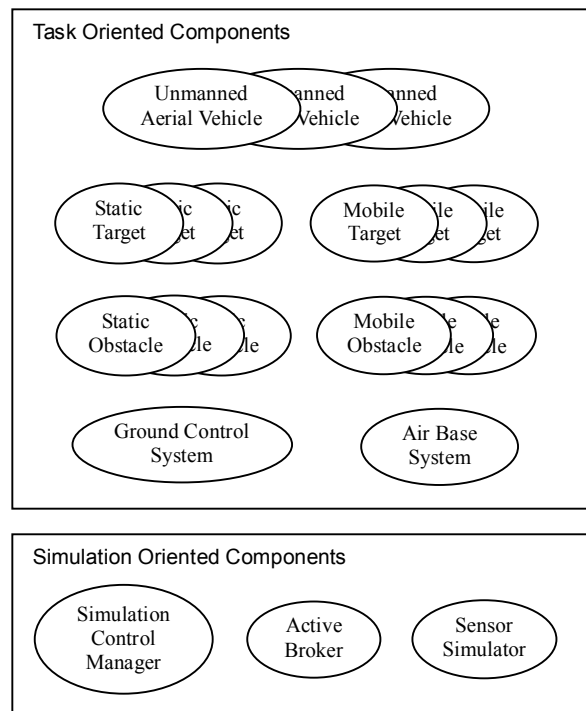
Figure 2: Simulation Components in UAV Simulator

3.2.1 Task Oriented Components

Task oriented components in our UAV simulator consist of five types: Ground Control System (GCS), Air Base System (ABS), Unmanned Aerial Vehicles (UAVs), obstacles, and targets. *GCS* is a central manager of UAVs and is aware of the mission area so as to indicate each UAV its air route in the area. However, GCS may not communicate continuously with UAVs to decide behavior of the UAVs at each time step because UAVs are supposed to perform their mission autonomously. *ABS* represents a control center of an air base and controls the sequence of take-offs and landings of UAVs. *UAVs* perform a given mission autonomously within certain restrictions, such as their kinematics and communication capability. *Obstacles* represent objects in which UAVs are not interested and with which a collision can happen. According to whether an obstacle can move or not, they are classified into two classes: *a mobile obstacle*, such as an airplane, and *a static obstacle*, such as a tall tower or a building. *Targets* represent objects of interest for the UAVs, such as, civilians to be rescued. According to its mobility characteristics, there are mobile targets and static targets.

3.2.2 Simulation Oriented Components

3.2.2.1 Simulation Control Manager.
Each component manages its virtual time because each actor has its own control thread. However, this situation can cause inconsistency in virtual times of components. To maintain consistency between virtual times, *Simulation Control Manager* (SCM) manages local virtual times of the simulation components. When every component starts its execution, the initial value of each local virtual time is set to 0. After every component starts, SCM broadcasts a *virtual time clock message* to the other components. When a component receives the message, the component increases its local time and performs a small portion of its task that should be completed during the predefined time slice unit. For example, when a UAV receives the message, it updates its location and direction vector, and also checks whether or not new objects, such as other UAVs, targets, or obstacles, are detected. If a new neighboring UAV is detected, the UAV might exchange some information with the new neighboring UAV. After a component finishes its computation, it sends a reply message to SCM. When SCM receives reply messages from all the other components, SCM increases its virtual time, and rebroadcasts another virtual time clock message.

3.2.2.2 Active Broker
In order for a UAV to perform a group mission, the UAV needs to communicate with its neighboring UAVs through local broadcasting. *Active Broker* simulates a local broadcasting mechanism. In general, the brokering service supports attribute-based communication. For example, if every UAV registers information about its current flying area with its actor name on a shared space, then when a UAV requests a broker for a message passing with a template that describes a certain area, the broker delivers the message to other UAVs which are in the area. However, this approach is not very accurate for finding the neighboring UAVs. Therefore, we have extended the function of the brokering service. In the active brokering service, every UAV registers information about its current location with its actor name on the shared space, and a UAV sends a special object instead of the template to request a message delivery to Active Broker. The object includes a specific method to be called by Active Broker. The method computes the distance between the location of the sender UAV and other UAVs and selects some which are within the local communication range. When the method returns actor names of neighboring UAVs, Active Broker delivers to them the message received from the sender UAV.

3.2.2.3 Sensor Simulator
Although each real UAV is supposed to be equipped with its own radar sensor, the radar sensors of all UAVs is simulated by a single simulation oriented component, *Sensor Simulator*. In the simulation, UAVs, targets, and obstacles register their current locations on a shared space every second in virtual time. Sensor Simulator periodically computes the distance between any two objects. If some components are within the sensor range of a UAV, Sensor Simulator reports information about these components to the UAV. Each UAV regards this information as its sensor input.

3.2.3 UAV Architecture

The most important simulation component is a UAV component. Therefore, we explain the architecture and the main behavior of a UAV in this subsection. A UAV is comprised of four modules: the physical process module, the trajectory planning module, the cooperative module, and the global control module (Figure 3).
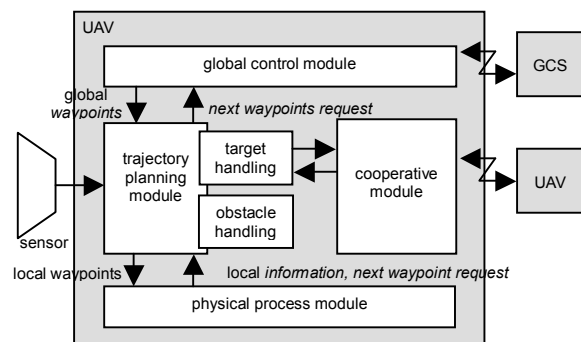

Figure 3: The Architecture of the Unmanned Aerial Vehicle Actor

When a UAV starts its mission, it does not have any information about its air route or the mission area. In our simulation, an air route is defined as a set of *waypoints* that need to be traversed by the UAV. Therefore, the first task of a UAV is to request the waypoints from GCS. The *global control module* of a UAV takes charge in communicating with GCS and managing the waypoints received. We call these waypoints *global waypoints*. When a UAV detects targets or/and obstacles, this information is delivered to the trajectory planning module from Sensor Simulator. The *trajectory planning module* handles them according to the predefined rules. For example, when a UAV detects several targets, it selects one target which has the best attractive force value, and then modifies its air route to reach the target. This function is performed by adding a waypoint to the list of UAV's current waypoints. The set of waypoints used inclusive of the additional waypoints are called *local waypoints*. The *cooperative module* is used when several UAVs want to handle a set of targets. To decide which UAV handles which target, the UAVs communicate with each other through the cooperative module. The kinematics of a UAV is implemented in the *physical process module*. Therefore, whenever this module receives a virtual time clock message, the physical process module computes the next location and the next direction of the UAV. When a UAV reaches the current waypoint, this module starts a turn toward the next waypoint according to the predefined kinematics.

3.3 User Interface

If we have to modify the UAV simulator whenever we execute it with different parameters, it is quite burdensome. Besides, modification at the code level requires comprehension making it hard for novice users to modify the code. In our architecture of UAV simulation, we separate the parameter modification part from the UAV simulator code as the user interface layer. Moreover, we separate the simulation checking part from simulator code. Therefore, the user interface layer consists of two programs: Configuration Interface Program and Simulation Viewer.

3.3.1 Configuration Interface Program

For the convenience of novice users, we have separated the configuration for UAV simulation parameters from the simulator code as a configuration file. This file can be modified by the Configuration Interface Program (Figure 4). Therefore, although a user does not look at and understand the source code for UAV simulation, they can change important parameters of simulation and run it without recompiling the source code. With this program a user can set up the number of UAVs, the size of mission area, the attributes of targets and obstacles, maximum simulation time, and the size of simulation time slice unit.
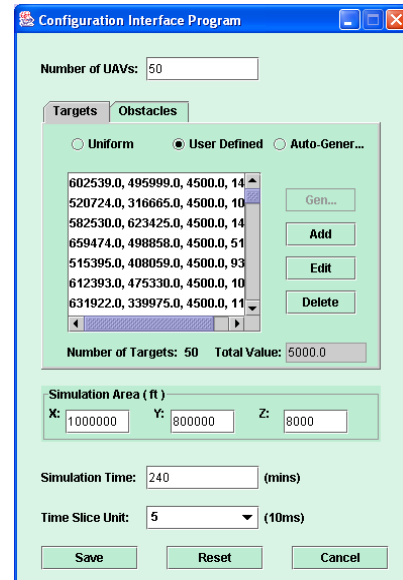

Figure 4: Configuration Interface Program

3.3.2 Simulation Viewer

Because of the characteristics of large scale simulations whose durations may sometimes be so long that we cannot monitor the simulation results continuously, we have separated the simulation checking from the simulation execution. Therefore, we look at and check the simulation results through Simulation Viewer (Figure 5). Another advantage of this approach is that the simulation results can be viewed back and forth with respect to the simulation virtual time.

While our UAV simulator is running according to the given parameters, the simulator generates simulation results on data files. The data files contain the locations and directions of UAVs, targets, and obstacles at every simulation virtual time step. The Simulation Viewer is used to check and verify the simulation results.
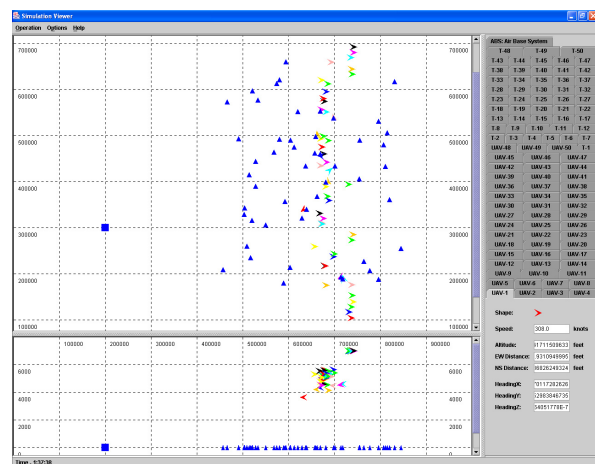

Figure 5: Simulation Viewer

4. COOPERATION AND COORDINATION AMONG UAVS

Cooperation among the UAVs is essential in directing the adjustment of policies in the globally most beneficial direction. In addition to cooperative dissemination of information, coordination of actions in larger teams is essential. With elements of uncertainty existing in the environment, coordination among UAVs has to be adaptive. The UAVs need to dynamically allocate responsibilities for different subtasks depending on the changing circumstances of the overall situation. For example, if additional targets are detected during a group mission, a team of UAVs needs to be able to handle them either by recruiting new member UAVs or changing the previous assignment of targets. In our UAV simulation, we use three strategies: the self-interested UAV strategy, the sharing-based cooperation strategy, and the team-based coordination strategy.

4.1 Self-interested UAVs

In the self-interested UAV strategy, a UAV senses a target and approaches it with the intention of consuming its entire value. When another UAV detects the same target, it also proceeds to consume the value of the target, irrespectively of what other UAVs do. Incessant polling of the target value till such time it is consumed completely serves as a means of interaction among the UAVs. It is not unusual to have more than one UAV concentrated on a target resulting in quicker consumption of its value, but also possibly in duplication of service.

4.2 Sharing-based Cooperation

In this strategy, once a UAV has discovered and located a target, it broadcasts this information so that other UAVs could direct their attention to the remaining targets. Reception of such information will result in the UAVs purging the targets that were advertised. This approach allows for a larger set of targets to be visited in a given time interval and is thus expected to be faster in accomplishing the mission goal. Exchange of information between UAVs referring to the same target will result in a UAV with a lower identification number to determine the UAV that would be responsible for this target based on parameters such as the distance from the target.

4.3 Team-based Coordination

In the team-based coordination strategy, certain UAV takes on the mantle of the leader of its team and dictates course of action to the other UAVs about the targets they need to visit. A team is dynamically formed and changed according to the set of targets detected; i.e. when a UAV detects more than one target, the UAV tries to handle the targets together with its neighboring UAVs. At this time, the main concern is how to select an optimum UAV and decide the number of UAVs required to accomplish a task, when there are a sufficient number of neighboring UAVs. As the basic coordination protocol, we use the Contract Net protocol [Smith 1980; Smith and Davis 1981]. The UAV initiating the group mission works as the *group leader UAV*, and the other participant UAVs are called *member UAVs*. When a member UAV detects another target, the UAV delivers information about the new target to the leader UAV, and the leader UAV will add the target to the set of targets to be handled. The leader UAV considers the distance between a target detected and neighboring UAVs to assign the target. When a member UAV consumes the entire value of a target the UAV secedes from its group.

5. EXPERIMENTAL RESULT

We have developed the UAV simulator and two interface programs in Java programming language. Our UAV simulator is running on the Actor Foundry, but interface programs do not require the Actor Foundry. In order to simulate the flying and turning behavior of UAVs, we use the basic kinematics model of airplanes, but we abstract away the detailed dynamics and kinetics of aircraft.

For the UAV simulation, the size of the simulation area is set to $1,000,000 \times 800,000 \times 8,000$ cubic feet (length $\times$ width $\times$ altitude), size of the mission area to $400,000 \times 500,000 \times 8,000$ cubic feet, the radius of local broadcast communication of a UAV to 50,000 feet, and the radius of radar sensor to 25,000 feet. There are 50 targets in the mission area, and they are normally distributed. Half of the targets are static and the others are dynamic targets. When a UAV is within 1,000 feet from a target, the UAV consumes the value of the target. The initial value of each target is 100, and the discount factor d in the target value function is 5 per second.

To investigate how different cooperation strategies influence the performance of a joint mission, we use Average Service Cost (ASC) defined as follows:

$$ASC = \frac{\sum_{i}^{n}(NT_i - MNT)}{n}$$

where n is the number of UAVs, NT_i means navigation time of UAV i, MNT (Minimum Navigation Time) means average navigation time of all UAVs required for a mission when there are no targets.

Figure 6 shows Average Service Cost for three different cooperation strategies. When the number of UAVs is increased, ASC is decreased in every case. However, the sharing-based cooperation strategy and the team-based coordination strategy are better than the self-contained UAV strategy. From this result, we conclude that communication of UAVs is useful to handle targets, even though UAVs in the self-

contained UAV strategy consumes quickly the value of a target when they handle the target together. Another interesting result is the performance of the team-based coordination strategy is similar to that of the sharing-based cooperation strategy, even though the algorithm of the sharing-based cooperation strategy is much simpler. The overall ASC of the team-based coordination strategy is 3 or 5 seconds faster than that of the sharing-based cooperation strategy. When $n_i(t)$ in the target value function is not used, the performances of the sharing-based cooperation strategy and the team-based coordination strategy are not changed very much while that of the self-interested UAV strategy is decreased (Figure 7).
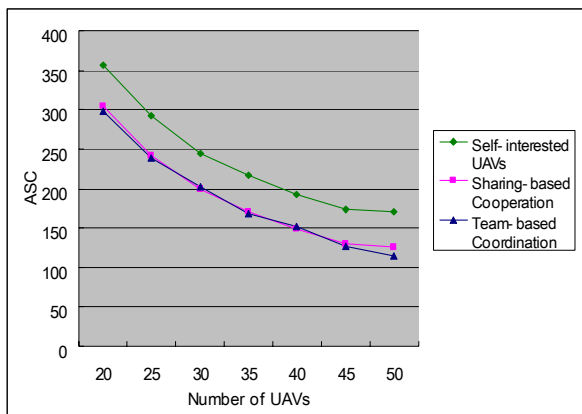

Figure 6: Average Service Cost (ASC) for three different coordination strategies.
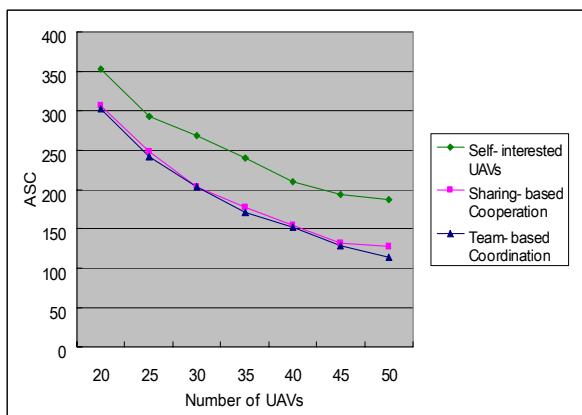

Figure 7: Average Service Cost when $n_i(t)$ is not used.

6. RELATED WORK

Johnson and Mishra present a flight simulation tool for GTMax (Georgia Tech R-Max VTOL UAV) [Johnson and Mishra 2002]. Barney Pell and his colleagues describe the NMRA (New Millennium Remote Agent), architecture for a UAV. The NMRA integrates real-time monitoring and control with planning and scheduling, handles fault recovery and reconfiguration of component models, and simulates the autonomy of a UAV [Pell et al. 1997]. However, the type of the

GTMax UAV is a helicopter, and both papers do not handle cooperation among UAVs.

Altenburg and his colleagues present an agent based simulator to simulate UAV cooperative control [Altenburg et al. 2002]. In their approach, agents are reactive agents while UAV components in our simulation are deliberative agents. Therefore, their agents directly respond to signals from environment, while our agents change their intention about targets and automatically and proactively select a different action. Also, their agents communicate with others indirectly through the environment while our agents communicate with each others directly. Kolek and his colleagues present a simulation framework to evaluate the performance of real time tactical radio networks with a UAV [Kolek et al. 1998]. In this paper, the authors explain how much distributed simulation could be applied to solve military problems, but they do not handle the autonomy of UAVs and coordination among UAVs.

7. FUTURE WORK

The Actor system supports distributed computational environment and actor mobility. In the current platform, it is the programmer's role to determine actor placement. However, this is hard to do when we do not know the CPU speed and the communication speed among different machines. Specifically, when the communication pattern among actors is changed, the initial placement of actors might prove to be a deterrent to cross boundary communication. For this, we are developing dynamic actor reconfiguration algorithm. In the new actor platform, the communication pattern among actors will be monitored, and actors will be dynamically reallocated by the platform.

Another problem of the current actor system is the existence of Simulation Control Manager (SCM) to control the virtual times of UAVs globally. This component may be a bottleneck of the distributed simulation, and if this component were to fail, the simulation would collapse completely. To counter this, the Jefferson's virtual time [Jefferson 1985] based actor platform can be used. In this actor platform, each actor maintains its own virtual time, and when an actor communicates with another actor and the time difference is more than the given threshold, the platform performs the rollback.

As another extension, we are looking to merge a few real UAVs into UAV simulation. That is, we are going to build a UAV simulator with the possibility of real time input from real UAVs and virtual UAVs. In this simulation, a real UAV can communicate with other real UAVs and virtual UAVs to perform a virtual task. This approach can overcome the problem of computer simulation, such as the inaccuracy of UAV kinematics and the communication delay defined by programmers.

In our simulation, we use Contract Net Protocol. It means if a UAV accepts the order from a leader UAV,

the UAV must handle the target. However, the belief about environment changes when UAVs detects more targets or additional UAVs become available after having consumed value of their respective targets. Therefore, when any change in the environment is detected or any UAV becomes available, this information is delivered to the leader UAV, and the leader UAV may reconsider and change the target assignment. Also, a member UAV may secede from its team to handle a new target with a more attractive force value. This idea is motivated from the leveled commitment in Contract Net Protocol [Sandholm and Lesser 1995].

8. CONCLUSIONS

In this paper, we have described the design and development of a distributed UAV simulator using an actor-based platform, a utility function, and Contract Net Protocol. The three layered architecture for our UAV simulation is presented: the actor-based distributed platform, the UAV simulator, and the user interface layer. We have described three strategies to perform a joint mission: the self-interested UAVs strategy, the sharing-based coordination strategy, and team-based cooperation strategy. This has been supplemented by our experimental results and outline of the future work.

Our UAV simulator is working on an actor-based distributed platform, and hence, it naturally adapts to the behavior of a distributed and concurrent situation. We can easily improvise the execution environment without changing the UAV simulator by separating the distributed platform from the simulator. For example, we can migrate some actors from a computer to another during the execution time. Other possible means for improvising the working environment have been presented in the future work section. When we consider multiple UAVs working together, their cooperation mechanisms are of utmost importance. In this paper, we have presented three different approaches, and compared and contrasted them. The experimental results suggest that cooperation and coordination strategies are better than the self-interested UAV strategy. Last but not least, we have introduced the active brokering service to support application oriented searching.

ACKNOWLEDGEMENT

REFERENCES

Agha, G.A. 1986. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, Mass.

Agha G.A.; I.A. Mason; S.F. Smith; and C.L. Talcott. 1997. "A Foundation for Actor Computation." *Journal of Functional Programming*, Vol. 7, No. 1, 1-69.

Altenburg K.; J. Schlecht; and K. Nygard. 2002. "An Agent-based Simulation for Modeling Intelligent Munitions." In *Proceedings of the Second WSEAS International Conference on Simulation, Modeling and Optimization*, Skiathos, Greece (Sep). Available at http://www.cs.ndsu.nodak.edu/~nygard/research/munitions.pdf

Astlery M. 1999. *Actor Foundry*. Department of Computer Science, University of Illinois at Urbana-Champaign, IL (Feb. 9). Available at http://yangtze.cs.uiuc.edu/foundry

Clausen T.H. 1998. *Actor Foundry – a QuickStart*. Department of Computer Science, Institute of Electronic Systems, Denmark (Nov. 9). Available at http://yangtze.cs.uiuc.edu/foundry

Doherty P.; G. Granlund; K. Kuchcinski; E. Sandewall; K. Nordberg; E. Skarman; and J. Wiklund. 2000. "The WITAS Unmanned Aerial Vehicle Project." In *Proceedings of the 14th European Conference on Artificial Intelligence* (*ECAI 2000*), Berlin, Germany (Aug), 747-755.

Grimes R. 1997. *Professional DCOM Programming*. Olton, Birmingham, Canada, Wrox Press.

Hadzilacos V. and S. Toueg. 1993. "Fault-Tolerant Broadcasting and Related Problems." In *Distributed Systems*, S. Mullender (Ed.). ACM Press, New York, 97-145.

Jefferson D. 1995. "Virtual Time." *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3 (Jul), 404-425.

Johnson E.N and S. Mishra. 2002. "Flight Simulation for the Development of an Experimental UAV." In *Proceeding of the AIAA Modeling and Simulation Technologies Conference and Exhibit*, Monterey California, CA (Aug), 5-8.

Kolek S.R.; S.J. Rak; and P.J. Christensen. 1998. "Battlefield Communication Network Modeling." *The DIS Workshop on Simulation Standards*. Available at http://dss.ll.mit.edu/dss.web/98F-SIW-143.html

OMG. 2002. *The Common Object Request Broker Architecture: Core Specification*. Version 3.0.2 (Dec).

Pell B.; D.E. Bernard; S.A. Chien; E. Gat; N. Muscettola; P.P. Nayak; M.D. Wagner; and B.C. Williams. 1997. "An Autonomous Spacecraft Agent Prototype." In *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA, 253-261.

Sandholm T. and V. Lesser. 1995. "Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework." In *Proceedings of the 1st International Conference on Multiagent Systems*, San Francisco, CA, 328-335.

Smith R.G. 1980. "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver." *IEEE Transactions on Computers*, Vol. 29, No. 12, 1104-1113.

Smith R.G. and R. Davis. 1980. "Frameworks for Cooperation in Distributed Problem Solving." *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 11, No. 1, 61-70.

AUTHOR BIOGRAPHIES

MYEONG-WUK JANG is a doctoral candidate and research assistant in the Open Systems Laboratory at the University of Illinois at Urbana-Champaign. His research interests include multi-agent system and task allocation in open distributed computing. He received a BS in Computer Science from Korea University in 1990 and an MS in Computer Science from KAIST (Korea Advanced Institute of Science and Technology) in 1992. He worked for ETRI (Electronics and Telecommunications Research Institute), Korea, until 1998. His web page can be found at `http://www.uiuc.edu/~mjang/`.

SMITHA REDDY is a Master/PhD student and research assistant in the Open Systems Laboratory at the University of Illinois at Urbana-Champaign. Her research interests include distributed systems, high speed networks, and dynamic resource sharing. She received a BE in Computer Science from University of Pune in 1999.

PREDRAG TOSIC is a doctoral candidate and research assistant in the Open Systems Laboratory at the University of Illinois at Urbana-Champaign. He received a BS in Mathematics and Physics and an MS in Applied Mathematics, both at University of Maryland Baltimore County, UMBC, in 1994 and 1995, respectively, and also holds an MS in pure Mathematics from University of Illinois at Urbana-Champaign in 1997.

LIPING CHEN is a doctoral candidate and research assistant in the Open Systems Laboratory at the University of Illinois at Urbana-Champaign.

GUL A. AGHA is Director of the Open Systems Laboratory at the University of Illinois at Urbana-Champaign and Professor in the Department of Computer Science. His research interests include models, languages, and tools for parallel computing and open distributed systems. He received a BS in an interdisciplinary program from the California Institute of Technology, an MA in Psychology from the University of Michigan, Ann Arbor, and an MS and PhD in Computer and Communication Science, from the University of Michigan, Ann Arbor.