Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 1177 – 1186

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

Applying DDDAS Principles to Command, Control and Mission Planning for UAV Swarms

Gregory R. Madey[a*], M. Brian Blake[a], Christian Poellabauer[a], Hongsheng Lu[a], R. Ryan McCune[a], Yi Wei[a]

[a]*Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA*

Abstract

Government agencies predict ever-increasing inventories of Unmanned Aerial Vehicles (UAVs). Sizes will vary from current manned aircraft scales to miniature, micro, millimeter scales or smaller. Missions for UAVs will increase, especially for the 3-D missions: dull, dirty, and dangerous. As their numbers and missions increase, three important challenges will emerge for these large swarms of sensor and surveillance UAVs: (1) the need for near real-time dynamic command & control of the swarms, (2) efficient mission planning and dynamic real-time re-tasking of the swarms, and 3) the need for improved automation of swarm mission planning and command & control. We describe an investigation with the primary objectives to design, develop, and evaluate: (i) a proof-of-concept simulation test-bed that investigates the benefits of using DDDAS (Dynamic Data Driven Applications Systems) for UAV swarm control, and (ii) engineering guidelines that will enable the use of DDDAS principles in such actual systems.

Keywords: DDDAS, UAV Swarms, Agent-based simulation, MultiUAV2, SOA Workflows, Sensor-based processing

1. Introduction

Swarms of low-cost autonomous UAVs (unmanned aerial vehicles) may provide effective capabilities in situations where government operations, disaster response, surveillance or search/rescue missions must take place over large-area terrains [1, 2]. UAVs communicate directly with each other (single-hop) and directly to a ground station (possibly through a satellite link). The reach of the local communication is limited. Their local task is to continuously report sensor data to a central location and to respond to the detection of potentially relevant events by focusing the sensor activity on the event location and adapting sensor collection (e.g., focused and increased sensing rate and quality, etc.). They may need to self-organize and coordinate with all other reachable UAVs to avoid collision, and increase quality and quantity of sensor data (e.g., by collecting video from different angles). UAVs can accommodate a large set of individual tasks or services for search, surveillance, sensing (e.g., with respect to weather), and deployment of supplies or materials. The coordinated operations of a group of UAVs represent interrelated tasks that realize higher-level mission objectives. A challenge in UAV operations occurs when a subset

* Corresponding author. Tel.: 574-631-8752 ; fax: 574-631-9260
E-mail address: gmadey@nd.edu

1877-0509 © 2012 Published by Elsevier Ltd.
doi:10.1016/j.procs.2012.04.127

of UAVs must be requisitioned from the larger mission to address a mission-critical task perhaps in a smaller, more isolated area. As their numbers and missions increase, three important challenges will emerge for these large swarms of sensor and surveillance UAVs: (1) the need for near real-time dynamic command & control of the swarms, (2) efficient mission planning and dynamic real-time re-tasking of the swarms, and (3) the need for improved automation of swarm mission planning and command & control [3-5].

This project is investigating the application of Dynamic Data Driven Applications Systems (DDDAS) principles [3] to Command & Control and Mission Planning/Replanning for future generations of UAVs. Those UAVs will be tasked with supporting evolving requirements for Intelligence, Surveillance, Reconnaissance, and Situational Awareness. UAV swarms may be composed of heterogeneous air vehicles with a wide range of capabilities. Those UAVs may range in size from nano-, micro-, to fighter-, or tanker-sized, and to vehicles with special capabilities. Future UAV missions will be complex, involve cooperative sensing, mixed platforms and capabilities, perhaps requiring cooperation with air and ground forces. UAV mission will require dynamic adaptive workflows, with the possible reallocation of communication, computation, and sensing tasks.

1.1. Goals

Software that addresses the operator overload problem associated with a large UAV swarm can be viewed as a DDDAS [3] that receives sensor data and flight status data from the UAV swarm and updates models that predict future performance of the swarm (e.g., minimum time to any point in area of interest). Insights obtained from those predictive models can then be used to update control rules/parameters for the swarm, indirectly steering the sensor swarm toward improved mission performance. One type of predicted performance will be the global metrics/goals, e.g., the average time required by the swarm to respond to reported events. Such a goal is made difficult by constraints such as the fuel/energy limits of UAVs and the need to replace/recharge/refuel UAVs periodically. Another type of performance prediction is measured by local metrics/goals of individual UAVs of the swarm. Such metrics/goals may include the ability to accurately and quickly detect objects in areas of interest and to respond to them appropriately (e.g., by focusing sensor collection on the area of interest) and in coordination with other UAVs (e.g., to avoid collisions and to increase usefulness of collected sensor data). Based on this scenario, the specific goals of this project are:

- Build a centralized simulator (i.e., the Application System in DDDAS) to be used for DDDAS inspired operator automation for mission planning, re-planning, and command and control. This DDDAS updated simulation will serve as a “look ahead” or “what-if” mission evaluation tool to support UAV swarm operators. In our prototype test bed, this simulator is implemented using Java and the MASON agent library [4].
- Build or adapt a UAV swarm flight simulator that will be a source of synthetic heterogeneous sensor data, unexpected events, and new requirements (i.e., the Dynamic Data in DDDAS). In our prototype test bed, this simulator is implemented using MultiUAV2 [5]. The interaction between MultiUAV2 and the Java/MASON DDDAS simulator is shown in the Figure 1 below (and Figure 4 later in this document).
- Evaluate the system software needed for the communications between the UAV swarm simulator and the centralized DDDAS simulator, both for dynamically updating the running DDDAS simulator and for communicating back to the UAV swarm for the purpose of steering the sensor swarm.
- Perform measurements, studying the simulator/UAV behavior using the following parameters and simulation scenarios:
  - Varying the mix between global metrics and local metrics and study impact on quality of assessment versus time-to-respond;
  - Varying different combinations of UAV reach, number of UAVs, size of area of interest, etc.
  - Varying event occurrences (number of simultaneous events, types of events);
  - Study simulator behavior (accuracy, delay) to changes in any of the above metrics instantaneously (failure of UAV, communication failure, event occurrences, change in mission objective, etc.);
- Design the various building blocks (agent deployment/management, sensing, networking, etc.) for actual UAV deployment using low-cost autonomous vehicles and sensors. Employ state-of-the-art techniques for energy/QoS-aware service composition of these building blocks.

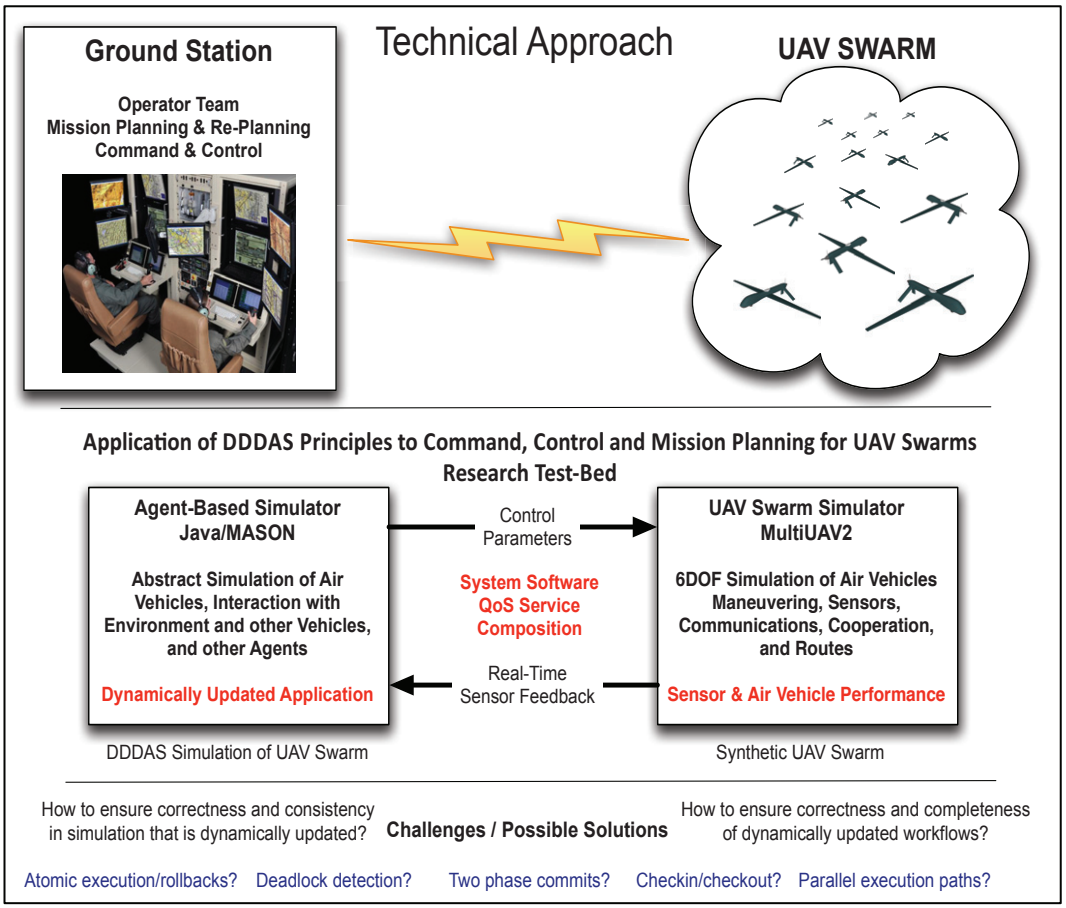

Figure 1: Design of a test bed for evaluating the application of DDDAS principles to UAV swarm Command & Control and Mission Planning. DDDAS research questions about how to ensure the correctness and consistency of dynamically updated simulations will be examined with the Java/MASON simulation. The synthetic UAV swarm simulator will be used to research questions about steering the remote sensors for improved system performance.

1.2. Approach

In the DDDAS concept, data are dynamically injected into an executing simulation to increase the performance of the application; also the executing simulation steers the sensor and mission processes of the application system. This DDDAS control loop can be used to dynamically develop and especially re-task and reconfigure UAV swarm missions. Of the four primary areas of research into DDDAS, we are focusing on three areas: 1) Application Modeling, 2) Sensor Control and 3) System Software. This research into the application of DDDAS concepts has the potential to address multiple UAV-swarm needs. A test bed is emulating a swarm UAV operator console for Dynamic Mission Planning and Command & Control of the swarm. Multiple UAV swarm missions with be evaluated, but we are primarily focusing on the intelligence, surveillance and reconnaissance (ISR) mission. We are developing and embedding in the test-bed: 1) swarm UAV service oriented computing workflow composition, 2) a programmable sensor architecture for re-tasking and remote programming of sensors, and 3) sensor-based pre-processing enabled by an ontology-based subscription language. Using both *in silico* simulation studies and potential physical simulation studies, we will test and evaluate our DDDAS-inspired solutions to the command & control and the mission planning applications.

We are employing two simulators: 1) a synthetic simulation modeling the actual UAV Swarm, and 2) a custom “test-bed” agent-based simulator capable of running many-times faster than real-time, that can be used to predict

and evaluate the behavior and performance of the UAV swarm (See Figure 1). The first is a higher fidelity model of a synthetic UAV swarm, while the second will investigate the challenges of injecting streaming sensor data into a running simulation. The model used to represent the synthetic UAV swarm is the MultiUAV2 simulator. The MultiUAV2 simulator provides for six-degree-of-freedom (6DOF) vehicle dynamics and user-defined cooperative control algorithms. It includes components for: Tactical Maneuvering, Sensor, Object Detection, UAV Cooperation, and Routes. This simulator, because of its high fidelity, was not designed to run in real-time, i.e., "wall-clock" time. It communicates back using our web services middleware to the UAV agent model. That agent-based model is being developed in Java using the agent-library MASON developed at George Mason University.

The agent-based simulator will be dynamically updated, based on feedback from the MultiUAV2 simulator, and used in faster-than-real-time, to predict the performance of the UAV Swarm. Scenarios where mission performance degrades will be demonstrated, with the agent-based simulation used to suggest improvements to the control algorithms, control parameters or the cooperative UAV workflows. We will demonstrate the closed loop multi-model prototype by the end of the first year of the project. The research challenges include questions about how to dynamically update a running simulation and mission workflows. We will address the problems of maintaining internal consistency, data integrity, program correctness, workflow completeness. These problems have been solved in other computing domains and those solutions will be applied here, e.g., from database engines the maintenance of data integrity through atomic execution of transactions rolling back if needed (i.e., the all or nothing principle), isolation of execution until complete and then only committing to the new simulation state. Other solution will come from the orchestration of complex workflows where component failures are possible, e.g., deadlock detection, checkin/checkout, or parallel execution paths. The prototype will inform us on how such a system could work, and indicate the applicability of DDDAS principles for command & control and mission re-planning for UAV swarms.

The remainder of the paper will survey work underway on 1) the application of web service composition to the design and development of the system software supporting the communication between the sensor swarm and the agent-based simulator, 2) sensor re-tasking and sensor-based pre-processing that will be added to the synthetic simulator, 3) the development of the synthetic UAV Swarm simulator using MultiUAV2, 4) methods for employing web services from within MultiUAV2, and 5) the development of an agent-based simulator using Java/MASON.

2. Discovery of Efficient Naturally Occurring Workflows within Predictive Agent Simulations and Operational Translation into Real-Life UAV Swarms

Generally, a predictive simulation exploiting DDDAS principles executes concurrently along with the real-life system while emulating the behaviors of the real-life system. One goal of our DDDAS test-bed is the discovery of naturally occurring workflows based on agent capabilities (or services) that may be more effective than the on-going workflows within the real-life system. In these situations, the predictive simulation must generate queries into the real-life system in order to discover the equivalent agent-based services and, using that information, compose new workflows for the real-life agents. Optimally, these new workflows would be equivalent to those discovered in the predictive simulation.

In order to translate these optimized workflows to the operational system, a new query-language and framework is being developed that will generate queries from the predictive simulations, discover state and capability information from the real-life system, and compose equivalent workflows from real-life operational data. Service-oriented computing (SOC) is a recent software engineering paradigm with the ability to identify and acquire system capabilities on-demand using universal protocols for communication and a standard grammar for querying available services based on user strategies. We are building on this body of work for communicating and correlating capabilities from predictive systems to operational systems.

Discovering workflows within operational systems is similar to the web service composition problem (underlying SOC) that has been addressed with numerous techniques: 1) *uninformed search*, such as breadth-first and depth-first, 2) *heuristic search*, such as greedy search or A* search, 3) *artificial planning*, such as order planning or metric planning, and 4) *metaheuristic search* and 5) *dynamic programming*. These solutions assume an environment where capabilities information is articulated in a homogeneous way, generally through semi-structured data that is semantically marked with user context and nonfunctional characteristics. In particular, we are basing our strategies on developing evolvable *semantic-based* domain models and representation. What is new here is enabling simulation-to-real-life interactions, and these type of DDDAS inspired interactions will require new tools and

techniques. The translation of simulated operations into real-life situations requires several corresponding investigations.

2.1. Modularized, generic representation of agent tasks

Information describing tasks and data shared between autonomous agents can vary greatly in format and representation. Is there a set of representations that can facilitate semi-automated composition of agent tasks/capabilities? The concept of queries in which a user requests information from an automated system usually has a well-documented standard for information exchange. For predictive simulations interacting with real-world systems, such a standard does not exist. As such, we are constructing a domain model and the corresponding knowledge-based applications with a focus on agent-based tasks. This includes characteristics such as timing, dependencies between tasks, and classification of needed external and internal information.

2.2. Composing enhanced workflows from on-going agent capabilities

Leveraging the generic, evolvable representation of agent tasks, computational approaches are being devised to weave together new operations from guidance provided from predictive simulations. New research will evolve the assessment/customization of existing computational techniques for adaptive service composition.

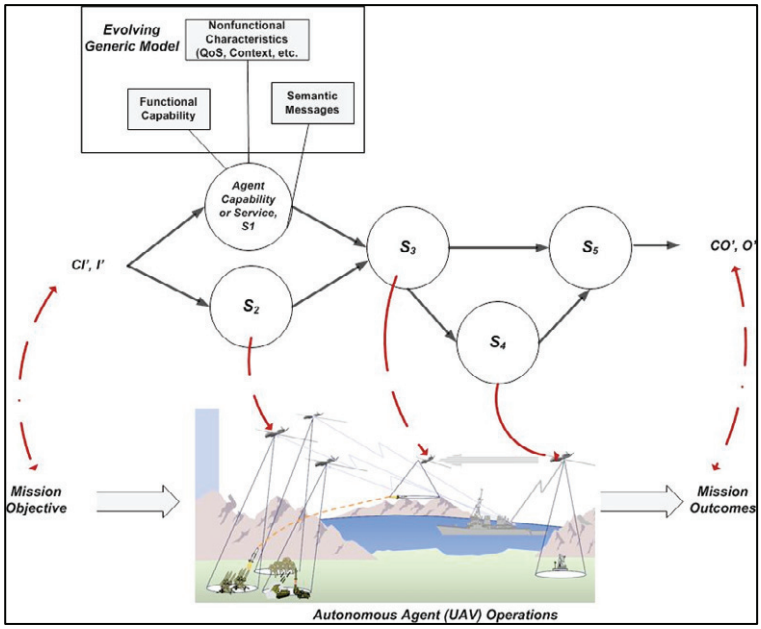

Figure 2: UAV Swarm workflows are modeled as a directed acyclic graph to achieve dynamic UAV swarm re-tasking

2.3. Graph Analysis

As shown in Figure 2, graphs can represent agent capabilities equally in operational environments and in simulated environments. A system, R, can be represented as the set of agent capabilities or services, S_n, where R = < S_1 … S_n>. A service is a 6-tuple of its preconditions, inputs, side-effect, affected object, outputs and post-conditions. S = (CI, I, A, AO, O, CO) is the representation of a service where CI is the list of pre-conditions, I is the input list, A is the service's side-effect, AO is the affected object, O is the output list, and CO is the list of post-conditions. The pre- and post-conditions are ground logical predicates. The generalized Composition problem can be defined as automatically finding a directed acyclic graph G = (V, E) of services from repository R, given query Q = (CI', I',

A', AO',O', CO'), where V is the set of vertices and E is the set of edges of the graph. Each vertex in the graph either represents a service involved in the composition or post-condition of the immediate predecessor service in the graph, whose outcome can be determined only after the execution of the service. Each outgoing edge of a node (service) represents the outputs and post-conditions produced by the service.

3. Sensor Re-Tasking and Sensor-Based Pre-Processing

The quality, reliability, and timeliness of the feedback loop between the remote UAV swarm and the DDDAS control and control simulation depends on the collection and dissemination of the relevant information in a timely fashion. Inherent to the goal of DDDAS is that data collection (i.e., sensing and UAV swarm status) should be adapted frequently (i.e., the steering of the sensors) to improve the performance of the DDDAS control and control simulation. As a consequence, this thrust of the research is concerned with the following challenges: (1) dynamic and efficient re-tasking of sensors and (2) sensor-based pre-processing of collected data. Both of these tasks are closely related, since adaptations in the sensing behavior and the installation of pre-processing routines can be done simultaneously.

3.1. Sensor Re-Tasking

Re-tasking and remote programming of sensors is an essential functionality in dynamic data-driven systems, where sensor output is used to drive simulations and simulation output is used to re-configure the sensors and the devices they are embedded in. Toward this end, a programmable sensor architecture is required that transforms simulation output into sensor re-tasking events.

Based on the requirements of the simulation, we consider three primary types of sensor re-tasking requests that can be made: *instantaneous*, *periodic*, and *event-based*. Instantaneous requests are made when a specific sensor reading is required at the current time, e.g., in order to ensure that the simulation has all its needed information or to confirm the accuracy of some other, previously obtained data. In the simplest scenario, this can be a request for an "alive" message from a sensor or UAV that has not been heard of for a certain amount of time. The format for such requests is:

synchronousRequest (SENSOR, TIMEOUT)

where SENSOR specifies the queried sensor and TIMEOUT indicates the latency requirements of this request. If no sensor reading within TIMEOUT can be obtained, the system may also return an older (cached) sensor value. For sensors that require periodic monitoring, an asynchronous monitor can be registered. The format for such a request is:

asynchronousRequest (SENSOR, PERIOD, EXPIRATION)

which will periodically (with a rate of 1/PERIOD) collect a reading from SENSOR until a certain time has expired (EXPIRATION), which can also be set to infinity. Finally, the third type allows a device to continuously monitor a sensor value, but only report this value if it meets a certain condition (e.g., when a GPS reading indicates that a certain geographic region has been reached). The format of such requests is:

registerEvent (CONDITION, PERIOD, EXPIRATION).

3.2. Sensor-Based Pre-Processing

While the re-tasking approach described above provides the ability to push certain pre-processing activities close to the sensor (e.g., using the event manager), our goal is to provide the tools necessary to describe and install complex sensing activities, typically involving multiple sensors, thereby further improving the performance and efficiency of the sensing activities.

This approach allows a simulator or controller to specify concrete conditions and attributes, integrated multiple sensor types, so as to collect the data only when it is needed and in the form it is needed. Toward this end, we are relying on an ontology-based subscription language for describing services in a software-interpretable format that maps sensing requirements to concrete sensor specifications. This subscription language, which is based on the

DARPA Agent Markup Language for Services (DAML-S), uses a service profile to describe the sensing requirements.

As an example consider the service profile shown below. In this case, a request for images from a camera (with a certain resolution, format, and dpi level) is made, where only cameras that reside in the circular region described by <Center> and <Radius> are requested to transmit images. Further, images are only to be transmitted between 9am and 10am and if certain conditions (described in the serviceConstraint clause) hold true. In this specific case, these conditions are specific to CPU utilization and battery charge level, thereby allowing for sensing adaptations based on the resource utilizations of a device.

```
<serviceProfile>
        <sensorResource>Camera</sensorResource>
        <serviceInputs>(resolution, format, dpiLevel)</serviceInputs>
        <serviceOutput>Image</serviceOutput>
        <serviceAttributes>
                <geographicRange>
                        <Center>Longitude, Latitude</Center>
                        <Radius>1000 meters</Radius>
        </geographicRange>
                <temporalRange>
                        <beginsAt>9am</beginsAt>
                        <endsAt>10am</endsAt>
                </temporalRange>
        </serviceAttributes>
        <serviceConstraint>
                <ifCond>cpuUsage < 80% && batteryLevel > 30%</ifCond>
                <Then>executeService=TRUE</Then>
                <Else>executeService=FALSE</Else>
        </serviceConstraint>
</serviceProfile>
```

In this example, sensing constraints and conditions are expressed as rules using one of the control constructs defined by a subscription language, i.e., using *ifCond-then-Else*. Other currently supported constructs are iterate and *repeatWhile*. Besides these simple requests, complex (composite) requests are also possible; these requests are decomposable into one or multiple simple sub-services. Composite requests can be of two types: (1) periodic and (2) sequential. We intend to extend the subscription language to define additional types and data structures to support highly powerful and complex sensor pre-processing services.

4. Synthetic UAV Swarm Simulator Using MultiUAV2

MultiUAV2 is software capable of simulating multiple unmanned aerospace vehicles that cooperate to finish specific workflows and tasks. It presents many features making it an ideal tool to implement our synthetic UAV swarm simulation for our test bed (see Figure 3). First, it is built using Matlab, Simulink and C++. Using the built-in functions from Matlab and flexible modeling solutions in Simulink, MultiUAV2 can be easily modified, extended and debugged. For time-costly codes (e.g., simulation of vehicle dynamics), MultiUAV2 implements them by using C++ to improve execution efficiency. Second, MultiUAV2 provides an accurate non-linear six-degree-of-freedom vehicle dynamics model. This model computes dynamics of vehicles based on aerodynamic data and other physical and control system parameters (e.g., forces and moments from the air vehicle's propulsion). Third, the simulation is implemented in a hierarchical manner with the capability to model goal identification, goal classification, task assignment, route planning, payload management and inter-UAV communications. Finally, MultiUAV2 offers tools and interfaces to conduct post simulation processing. The MultiUAV2 development environment is simple and easy to work with.

While MultiUAV2 offers the above capabilities, we are adding several new features to support our investigation of DDDAS for UAV swarms. First, MultiUAV2 by design is not a real-time simulation platform, meaning that when the number of vehicles in the simulation increases, the time needed to complete the simulation may be longer than

that of the real UAV swarm. Table 1 shows the relationship between simulation time and simulated time using MultiUAV2. There are ways to increase the speed of simulation such as turning off options (e.g., *g_OptionSaveDataAVDS* and *g_OptionSaveDataPlot*) and increasing the sampling step size. However, as the sampling size increases, the probability that the UAVs miss targets increases. Second, in the baseline version of MultiUAV2, the communications among air vehicles is assumed to be perfect (i.e., with no packet/message loss). To add packet/message loss, a wireless channel model, a physical layer model and a scheduling algorithm will need to be selected. With the structure of MultiUAV2, the wireless channel and physical layer models can be appended to "receive message" block which is a S-function serving as an interface between MultiUAV2 and an end-user component. For the scheduling algorithm, we have to add code before the vehicles call *CreateStructure* to create and send messages.

Table 1: Sample MultiUAV2 simulation times vs. simulated time as the number of UAVs increases.

Number of vehicles	10	15	20	25	30	40
Simulation time (seconds)	153.278	273.528	422.136	731.703	1128.07	2890.38
Simulated time (seconds)	200	200	200	200	200	200

In particular, when messages are generated, senders (i.e., UAVs) call our scheduling algorithm to analyze the channel status (busy or idle as in IEEE 802.11 standard to conduct backoff), or to check whether their transmission slots arrive (if TDMA-based solutions are used). If channel is available, *CreateScturcture* is called and messages are put into a global structure called *g_CommunicationMemory* as implemented in MultiUAV2. For receiving function, upon the arrival of updating cycles, “receive message” block of a receiving vehicle will check messages in *g_CommunicationMemory*, process them and construct an output list of signals to be fed to the SIMULINK simulation. So, when processing the messages, we implement the simulation of wireless environment by judging whether the message has enough power to noise ratio to be received. Third, we need to model the malfunction of sensors. We can take advantage of the Sensor Manager in MultiUAV2 where the function of object detection is implemented. We can add a noise matrix to the produced data to emulate the malfunctioning sensors.

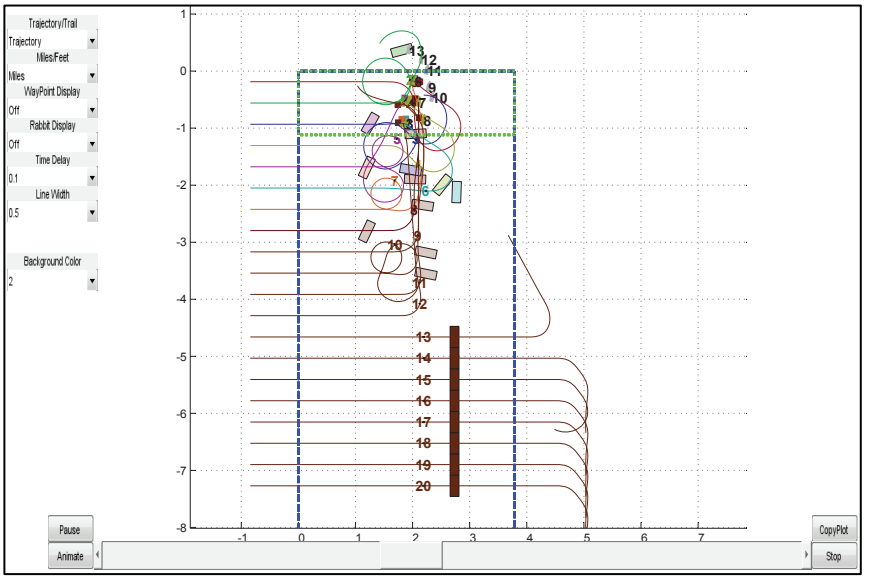

Figure 3: A MultiUAV2 simulation of a 20 UAV swarm.

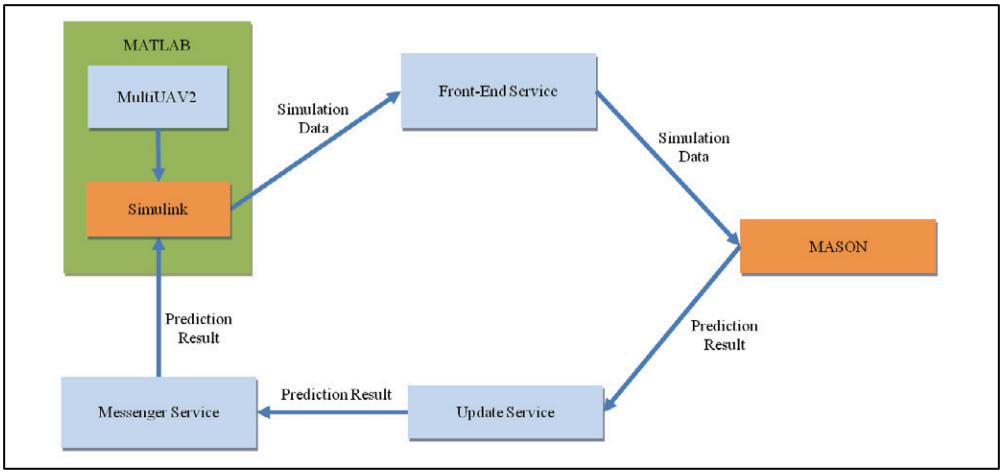

Figure 4: Conceptual architecture of the UAV Swarm test bed. MultiUAV2 is used to simulate a synthetic UAV swarm, while a Java/MASON simulation is the DDDAS application system that is dynamically updated and used to predict and identify new command and control goals, UAV workflows, and individual UAV flight rules and parameters.

5. Conceptual Architecture for the UAV Swarm Test Bed

Figure 4 illustrates the architecture of the Swarm-UAV DDDAS test bed. The MultiUAV2 software is used as a substitute for a real UAV swarm (in our test bed, it simulates the synthetic UAV swarm) and its operational environment, while the Java/Mason simulation is the DDDAS application system that is dynamically updated and, in turn, used to send new control commands back to the UAV swarm. During the simulation of the synthetic UAV swarm, MultiUAV2 data is extracted and fed to a front-end web service. The *front-end service* invokes the Java/MASON simulation software, to conduct UAV swarm behavior prediction tasks. After the tasks are completed, the Java/MASON simulation returns new control parameters, new UAV workflow specifications, or new goals to an *update service*. The service sends update messages to a messenger service putting the messages into a queue. MultiUAV2 periodically polls the messenger service using the update to dynamically adjust simulation parameters.

Also shown in Figure 4 is the relationship between MultiUAV2 and its execution environment. MultiUAV2 is implemented as a MATLAB simulation suite. The majority of the software is written in MATLAB language but it also leverages external C++ codes to accelerate the simulation process. We used MATLAB R2011b to run the simulation. MultiUAV2 uses an important component of MATLAB, the Simulink platform, to model and simulate the UAV flight dynamics and other behaviors. Simulink is widely used by developers from different areas as a platform for model-based design for dynamic and embedded systems. As in MultiUAV2, the embedded flight software, including the various kinds of component managers, is implemented as a Simulink system. We have implemented the link between MultiUAV2 and the front-end service leveraging the web service invocation mechanism provided by MATLAB, and the front-end service is implemented as an Axis2 POJO web service. To extend MultiUAV2, a new block called the S-Function block is added to the Simulink model. The S-Function block is a feature to let developers incorporate functionalities implemented in MATLAB language, C, C++ or Fortran into Simulink. Developers can specify the name of the S-Function in the block property, then provide the S-Function file so that Simulink can automatically calls it during the simulation.

6. Agent-Based Simulator Using Java/MASON

The Java/MASON simulation is our application system that is dynamically updated with sensor and UAV location data from the synthetic UAV swarm running on MultiUAV2. MASON is a multiagent simulation toolkit developed by George Mason, designed to support large numbers of agents relatively efficiently on a single machine. Programmed in Java, MASON runs in heterogeneous computing environments and is capable of interfacing with

many common software packages. The dynamic adaptive simulation component is designed to receive input parameters from the target environment, execute, and provide predictive models in faster-than-real time. Our proof-of-concept simulator, developed using the MASON toolkit, portrays UAV's as agents in a Continuous-2D Field representing the target operating environment. The simulation is capable of taking in the number of agents in the target environment, an agents' sight representing radar distance, and agents' speed, corresponding to the distance travelled in each step of the simulation. Using a sight parameter, agents can detect the number of nearby agents. Each UAV can sense neighboring UAVs within the radius of sight. Using MASON's Inspector property, we can observe the number of nearby UAV characteristics, such as nearby agents or unique coverage, during runtime. MASON includes tools to observe how parameters change over the course of the simulation.

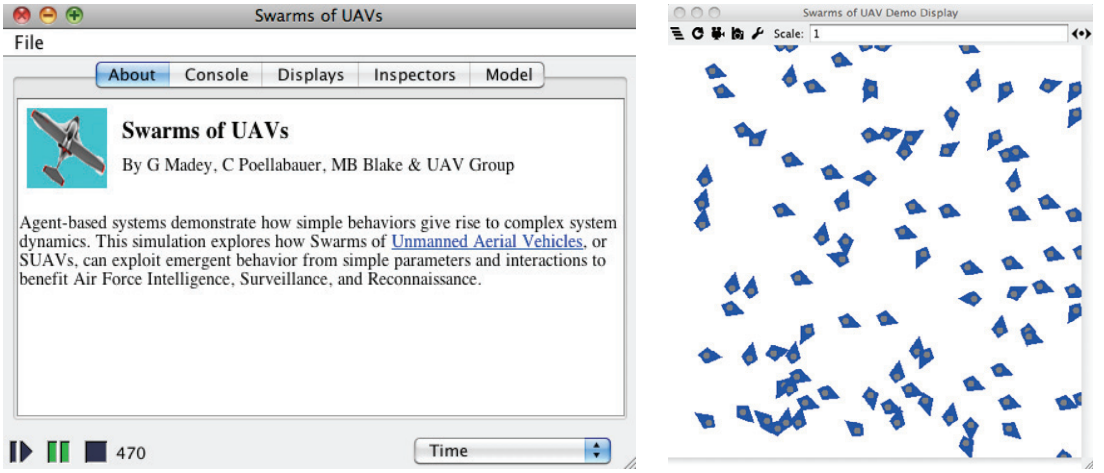

Figure 5: A prototype Java/MASON agent-based simulation of the synthetic swarm of UAVs (emulated using MultiUAV2) is under development. This is the application system that is dynamically updated with simulated synthetic data describing sensor and location information.

Acknowledgements

This research was supported in part under a grant from the AFOSR Award # FA9550-11-1-0351.

References

[1] Library of Congress, "Mini, Micro, And Swarming Unmanned Aerial Vehicles: A Baseline Study," Washington, D.C.2006.

[2] Library of Congress, "Unmanned Aerial Vehicles: Background and Issues for Congress," Washington, D.C.2005.

[3] F. Darema, C. Douglas and A. Patra, "Report of the August 2010 Multi-Agency Workshop on InfoSymbiotics/DDDAS - The Power of Dynamic Data Driven Applications Systems," 2010.

[4] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "MASON: A Multi-Agent Simulation Environment," *Simulation: Transactions of the Society for Modeling and Simulation International,* vol. 82, pp. 517-527, 2005.

[5] S. J. Rasmussen, M. W. Orr, D. Carlos, A. Deglopper, F., and B. R. Griffith, "Simulating Multiple Micro-Aerial Vehicles and a Small Unmanned Aerial Vehicle in Urban Terrain Using MultiUAV2," presented at the 2005 AIAA Modeling and Simulation Technologies Conference, San Francisco, 2005.