Using Integrative Modeling for Advanced Heterogeneous System Simulation

Jacob Gulotta, Diyang Chu, Ximing Yu, Hussain Al-Helal
Tapasya Patki, Jason Hansen, Maribel Hudson, and Jonathan Sprinkle
Department of Electrical and Computer Engineering
University of Arizona, Tucson, AZ 85721-0104

Abstract

This paper is an academic experience report describing the use by researchers at the University of Arizona of a domain-specific language developed by the Institute for Software Integrated Systems (at Vanderbilt University). The domain in question is heterogeneous, distributed simulation of quad-rotor unmanned aerial vehicles (UAVs) as they respond to command and control requests from a human operator. We describe in detail how our individual designs of the controller and guidance laws for the UAV, its rendering and position updates, on-board sensors, and the various commands to delegate mission-critical behaviors, all interact using the ISIS-developed modeling language. We then discuss the outlook for this domain (heterogeneous system simulation and integration) for domain-specific languages and models, specifically for unmanned vehicle control and interaction.[1]

1 Introduction

Large projects with decentralized development face a critical issue in holistic system simulations. Maintaining a single simulation strategy, which may even include the use of proprietary tools and/or shared network drives, is quite difficult to achieve, and can lead to poor software engineering practices where elements are developed outside the simulation toolchain. These elements must be rewritten or adapted to fit inside the tools used by the project. Such practices are prone to problems that are subtle, such as mismatched models of computation, as well as problems that are widespread, such as software bugs while porting.

Many systems require development and design in proprietary tools (e.g., MATLAB/Simulink for the domain of control systems), and may take advantage of sophisticated models of computation available in such tools. Other portions of the system may depend on logic that is best expressed as a switch statement in C/Java, or may be run as an applet (e.g., human control through a command and control interface). How to integrate these portions of the system with various components written in other languages is best done through middleware, and many standard middlewares exist for such applications. However, for an expert in control, or discrete event simulation, middleware programming can be a treacherous and confusing addition to their own algorithms.

The simulation of these systems built with heterogenous tools, components, models of computation, and operating systems is a nontrivial task that is best tackled by a middleware expert. However, there exists the bootstrapping issue of confirming that all programmers for each component follow a styleguide, or include standard header files with standard object definitions. Enforcing such a styleguide early in the process can often lead to the 'chicken-and-egg' problem where experts cannot start working on their algorithms because they do not have a testing infrastructure, while infrastructure developers cannot develop the middleware because they do not have a set of algorithms to design around.

To address this issue for the specific domain of multi-vehicle command and control (C2), Balogh and others [1] developed the HLA paradigm. Starting with a suite of tools that could utilize the infrastructure, and with a few examples, we began an experiment to continue leading-edge implementation of interactions between components, with the intention of integrating the components into an advanced demo that drew from many different simulation, design, and visualization tools. Importantly, we were able to do our component designs and simulations *independently* of the anticipated middleware, infrastructure, and global sim-

[1] A preliminary version of this paper was presented at the 8th OOPSLA Workshop on Domain-Specific Modeling [8].

ulation strategy. Although it was known *a priori* that HLA was the likely candidate, this strategy enabled users to operate without that assumption[2]. Integration of these various components was somewhat trivial, which is a great result for the domain-specific modeling language, as it reduced the complexity of the expert developers significantly.

The scope of this paper does not include motivating the development of this HLA modeling language, nor a detailed description of the HLA middleware used. Readers interested in these details can refer to [1][3]. In fact, there were many design and application domain choices made by the authors of the domain-specific language we use in this paper which we do not justify. We instead present this application example which shows the tremendous amount of heterogeneous simulation, design, and rendering that the use of this domain permitted in the period of just three months. For this paper's scope, we are most interested in the following qualities of a modeling language:

- the ability to specify tool-independent data structures;
- the ability to compose data structures with other data structures;
- the ability to synthesize "glue code" between various tools and software architectures;
- the ability to prototype component behaviors without running middleware as part of the test;
- the ability to use existing domain-specific tools and environments for design of models, and re-use those models in those tools at runtime; and
- the ability to have a single, unifying modeling language that permits all of the above.

In this paper, we describe our experience with using this domain-specific modeling language, specifically with its advancement of our design and simulation agenda from the perspective of "what would we have to do if we did not have the modeling language to help?" We first describe the tools which were at our disposal for design, simulation, and visualization. We next discuss the various implementations of component functionality. Finally, we describe the integrated demonstration, and how we envision our future work based on the capabilities of this modeling environment.

[2]In fact, early application domain choices utilized the ICE middleware by ZeroC.

[3]The maturity of the project and the short timeline for this workshop do not permit an in-print citation of the work.

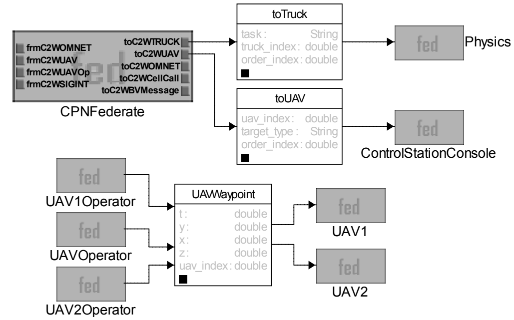


Figure 1. GME Model in the HLA paradigm, which describes an interaction between the `UAV` and the `UAVGSOp`. Note that `New Waypoint` information is passed to the `UAV` from the ground station, and the `UAV` publishes its position for interested parties. The object-oriented structure of those messages are present in the UML class diagram representation.

2 Modeling Language Description

The High Level Architecture (HLA) [3] is a DoD/IEEE standard that will be utilized to facilitate the reuse and interoperation among various simulation tools. Through HLA, it is possible for distributed collaborative development of complex simulation application across platforms. In the concept of HLA, a *federation* is the set of interacting simulations involved, while a single HLA-compliant simulation program is called a *federate*. In common applications, a federation consists of several functional components including:

- simulations, which are generally federates, that contain all object representation;
- Runtime Infrastructure (RTI), which acts as a distributed operating system for federation. The services implemented in RTI enable the exchange of data among simulations. There are various implementations of RTI available, either free or licensed. The one applied in the project is the Portico 8.0 (see `http://www.porticoproject.org/`); and
- the interface to RTI, which is consistent with HLA runtime interface specification. It standardizes the communication between federates and the RTI implementation.

A federation may also contain components like *Data Collector and Passive Viewer* and *Interfaces to Live Participants*, which may further extend the functional-

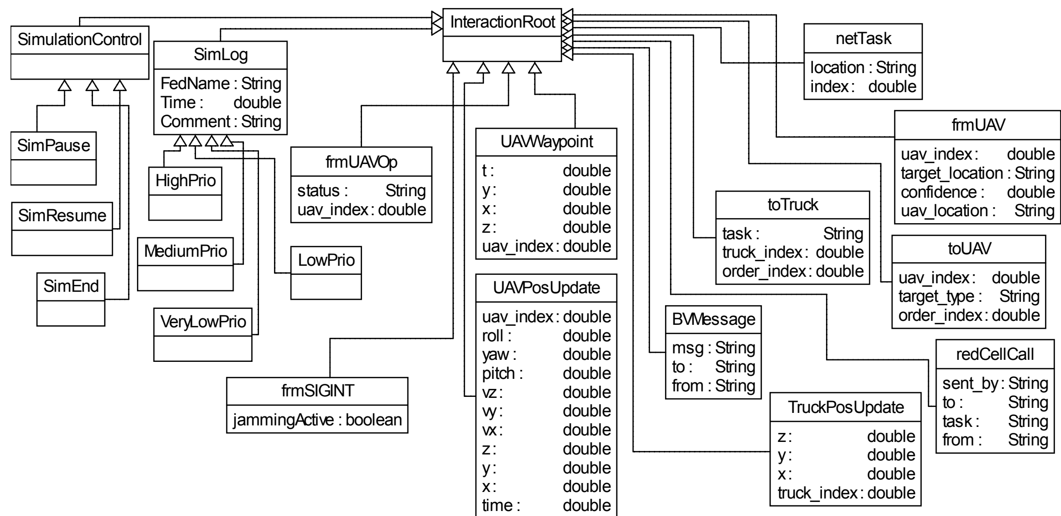


Figure 3. The model of interactions, in the HLA paradigm. Models describing the runtime parameters for simulation, logging, and network interactions are in the left-hand side of the figure. The right hand side concentrates on messages sent back and forth between various components (`UAV`, `Ground Station`, etc.), including the commands used by various components. Finally, the rightmost portion of the figure shows the various objects that are passed by the RTI, including the sending/receiving streams of the network simulator.

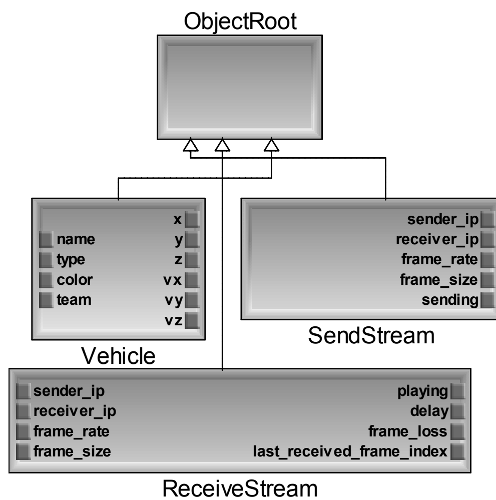


Figure 2. Object Hierarchy

ity of the whole system. The definition of HLA includes three parts: `HLA Interface Specification`, `HLA Object Models` and `HLA Rules`. In the project concerned, various heterogeneous tools are intended to be integrated under the HLA. Specifically, we are to use following tools collaboratively:

- MATLAB/Simulink
- OMNeT++ [9]
- CPN Tools [4]
- DEVSJAVA [7]
- 3D-Viewers.

To simplify and unify the simulation of a heterogeneous system, the Generic Modeling Environment (GME) [5] is applied. GME is a configurable modeling environment developed by the Institute for Software Integrated Systems (ISIS) at Vanderbilt University. Through its Model-Integrated Computing (MIC) philosophy [6] it provides a GUI environment for creating Domain-Specific Modeling Languages (DSML).

Utilizing the HLA modeling paradigm, users are able to create federates of different simulations, while the realization of each federate may vary significantly. The domain model created using HLA paradigm specifies the structure and interconnection of all its components

in the federate. The model interpreter, transforms this structure into configuration files used by the HLA implementation (RTI).

In brief, the HLA paradigm provides the following abilities:

- layout of interaction models of various components;
- middleware-independent specification of data structures for messaging; and
- specification of runtime parameters for the overall simulation.

Basic interactions among federates within the project can be shown in Figure 1. `CPNFederate`, `ControlStationConsole`, `UAVOperator`, `UAV`, etc. are heterogeneously implemented federates which can interact within HLA. For example, the `UAV` federate is developed by MATLAB/Simulink and `CPNFederate` is done by CPNTools. In Figure 1, data transferred among certain federates include `toTruck`, `toUAV`, `UAVWaypoint`. The structure of these data are specified within a larger diagram shown in Figure 3. Figure 3 is the Object and the Interaction Diagram, denoting all the structure of all allowed interconnection in the whole federation.

3 Rendering

The aim of a simulation is to accurately replicate a real-life or hypothetical scenario, including its visualization. Tools that permit the high-fidelity design and simulation of a complex vehicle do not always provide an equivalent high-fidelity rendering of that vehicle, so there exists a need to enhance this visualization through external tools.

The visualization is performed through Google Earth, where our UAV's position is indicated by scrolling the map (i.e., we do not see the UAV, but we see what the UAV sees). The object we are tracking (a truck) is visible only when it is in our field of view. Thus, changes in altitude (and attitude) affect whether or not we can see the truck. The HLA implementation pushes the data values for the UAV's state to Google Earth in accordance with the progression of simulated time.

4 Controller

We use the STARMAC as the UAV model in this project. It is a quadrotor UAV developed by a group at Stanford University. [2] provides both the description of its dynamics and a demonstration of its abilities. For easy visualization and considering the feature of Mathwork design tools (such as a graphical block diagramming tool and a customizable set of block libraries), Simulink was widely used as the modeling language for designing the controllers.

In order to respond to the assorted command and control messages of the `Ground Station`, the Simulink controller can switch between various control laws. A top level view of the modified Simulink block diagram can be found in Figure 5. Both the spiral search and tracking controllers can be seen on the left, with an input flag that specifies which one should be active.

If we want to search some area to locate the truck, we can set the UAV to the spiral search mode. When we choose "spiral" as the search pattern for the UAV, it spirals outward from the location where it received the `Ground Station` message to switch to spiral search mode. In order to let the UAV follow the spiral ideally, the algorithm takes the UAV position as an input, and returns the velocity desired. A typical response can be found in Figure 4(a).

The tracking controller for the STARMAC was achieved using a proportional controller across the motor voltage command and setting up feedback loops around translational acceleration, velocity, and position. As sometimes the detected object's position updates will come in a relatively large time interval, such as one second or more, the controller should always try to use the newest waypoint to calculate the desired direction. The tracking velocity is decided by the horizontal distance between the UAV and the truck. If the distance is too large, we will set the speed of UAV to the maximum value; If the distance is small enough for us to see the truck in the camera, the UAV speed will slow down to a value similar to the truck speed; in this way once the truck changes its direction, the overshoot of UAV will be tolerable. The worst case is when we reached the newest truck waypoint, we still can't see anything in the camera. In this case, there will be two choices: the first one is that the `Ground Station` should set the UAV mode back to spiral search mode; the second is that let the UAV will climb in altitude, so the camera will be able to have a larger view. The response of tracking a truck which is running in a step to the north east can be found in Figure 4(b).

Those plots above are important because the design and simulation was done *inside* the Mathworks toolbox, it contains the function to generate plots, utilizing summing and add feedback blocks, and the whole design is based on the UAV model. If we want to do the design the system in Matlab Simulink and then export them to C/C++, some error will occur because

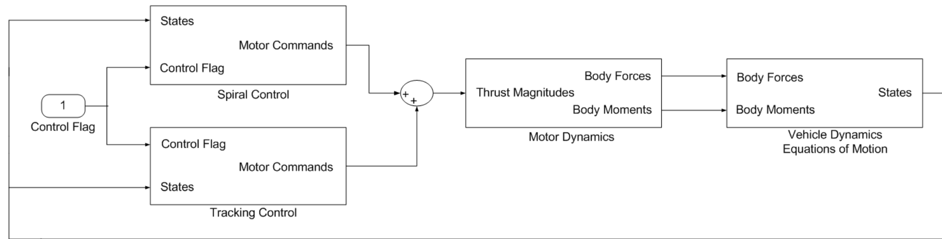


Figure 5. An overview diagram of controller model. Note that the switching controller on the left side of the figure represents the hybrid feature of the controller, whether the vehicle is flying the spiral search pattern, or the tracking pattern.

of some sutle change in modeling environment. With the integration of MATLAB/Simulink into the HLA infrastructure, the same model generated from the design and analysis can also be used as executable models.

5 Ground Feature Detection

For unmanned vehicles outfitted with a camera sensor, one potential application is unmanned, autonomous surveillance. The UAV could be given an instruction as, for example, "find the blue trucks in this area." Analyzing the output image from a camera is a crucial operation for the UAV to complete this task. It would have to determine, first, whether or not there was a blue truck in its field of view and then report the rough location of that truck in some meaningful way based on the image and its present state.

In order to closely emulate an actual implementation it is necessary to have a picture. However, simulation has a major drawback with respect to the sensor: there is no image to be directly analyzed. Therefore a software workaround is required to determine whether an object of interest is contained within the image. Taking advantage of the fact that knowledge of the simulation world is absolute, it is possible to bypass the image analysis phase. Instead of locating an object via the camera's image, an algorithm (given the known position of an object) can report where that object would appear in the image, if at all. This is done with a simple coordinate transformation from one reference frame to another, then scaling and approximating the new coordinates. From there the final task of meaningfully reporting the "detected" location is identical as if the pixel location came from an actual picture. Such an approach allows easy transition from canned simulation data to data obtained from analysis.

The process starts by feeding the algorithm the position of an object from the RTI in x,y,z coordinates. Additional required data are the x,y,z coordinates of the UAV, its roll-pitch-yaw orientation, and the intrinsic parameters of the camera. The most important properties of the camera are its focal length, the size of the CCD, and the resolution. The parameters are set such that the area the camera sees is roughly equivalent to the area displayed in the Google Earth plugin used for visualization. That is to say if the object can be seen using the Google Earth plugin then it can be detected by the camera. Given these settings, the algorithm produces the i,j pixel coordinates that represent where in the picture the specified object lies. Finally the pixel coordinates are reverse transformed, using the state data of the UAV at the time the picture was taken, and reported as an approximate location for the object. In practice only the latter portion is necessary because there will be an image actually available for analysis.

To accomplish this, the RTI and MATLAB/Simulink must communicate with one another. The RTI provides MATLAB the x,y,z coordinates of the object and UAV while Simulink provides the state and orientation data directly to the camera (i.e., not published through the RTI). The camera parameters are fixed and so are simple constants. All the calculations are carried out in MATLAB. An overview of the process can be seen in Figure 6.

6 Integration

Section 2 describes how the GME Environment was used to develop an application model with the help of the HLA paradigm. The `UAV` and the `Ground Station` federates and the associated interactions were discussed, including how the UAV is controlled through Simulink. Target detection was discussed in Section 5, and requires state information of the `UAV` as well as information regarding the target's location.

Now with each of these pieces developed, simulated, and tested individually, we integrated them into

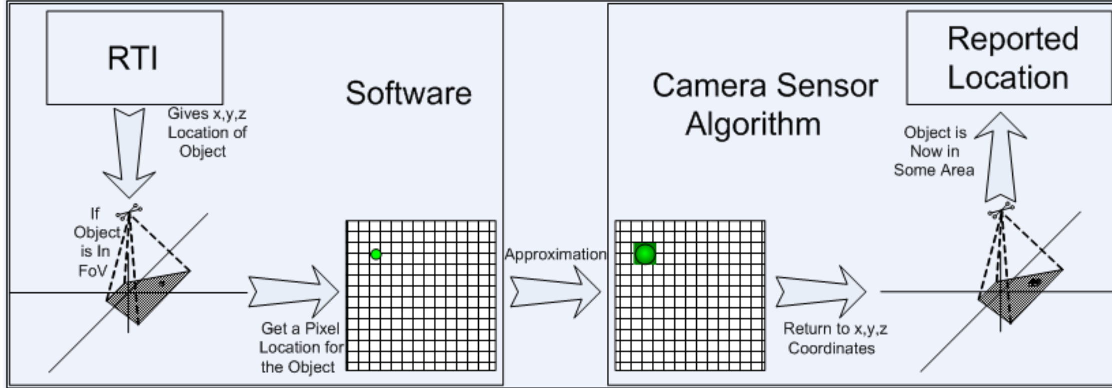


Figure 6. Overview of GPS target identification scheme, given that no image processing is available. If the object is in the field of view (FoV), it is passed along to the targeting component to report location.

a demonstration. To do this, we followed the overall structure as described in Figure 7. With Portico as the RTI infrastructure, we used a Java implementation of a `Ground Station` which allows human command input into the simulation. These commands include direction for the `UAV` to search for a target, fly to a waypoint, track a target, etc.

A separate Java application allows the user to create a path for the target by drawing arbitrary lines on a screen, that are then translated into latitude and longitude. The application keeps track of the latitude and longitude of each point on the path. Importantly, each point on the path represents a particular time at which the truck is at that point. The time is determined by assuming a new point is sent every second. This information about the location of the target is then published, for the MATLAB component discussed in Section 5 to relay information about a target being acquired.

These Java-based components express basic control-flow, and also have their own GUI, utilizing Java's user-interface libraries. For the physical-system simulation, the Simulink models discussed in Section 4 are called, which publish updated position information. This positional information, as well as location of the target, are read by the Google Earth visualization component, and visualized for the benefit of the `Ground Station` human operator. The final result is an integrated demo that can be run from a Windows `.bat` file, reducing the possibility of human error in starting up components in the wrong order, or forgetting to pass in parameters. Such an automation for running the demonstration also reduces the effort required to run tests to confirm that certain tools (e.g., MATLAB) are properly integrated into the demonstrator's machine.

7 Results and Analysis

We successfully integrated several demonstrations that showed our various technical contributions. Depending on the number of interactions that we utilized in each demonstration model, about 5000 lines of code were generated for the entire set of federates available. This included the standard "getter" and "setter" methods for various objects, but more importantly the "publish" and "subscribe" methods were provided, reducing the complexity of programming for domain experts. For the Simulink interaction, some hand-editing of the model is required to integrate, i.e., replacing the state reading and writing blocks with HLA reading and writing blocks. This is important not just for information exchange, but also to prevent Simulink from advancing more rapidly than other portions of the simulation, and thus not synchronizing data with other components.

Based on the amount of generated code discussed in Section 7, it would require a *significant* amount of human effort to code the various integration points for each tool. The HLA modeling language provided an integration point for *each* software tool we needed, as well as many others that we did not need. This not only provides a late-stage integration freedom, but also gives a design freedom, where alternative tools can be explored in parallel tested upon integration for selection of the optimal behavior. In addition to the raw effort of programming the interaction points, there is significant effort required to understand *how* the tools could interact with the middleware. Thankfully, this task has already been done by the HLA modeling lan-

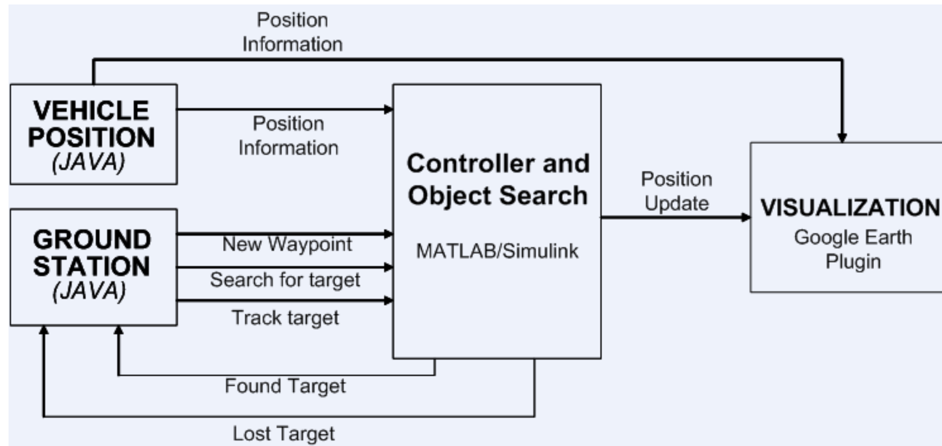


Figure 7. Integrated Schema of the Federation Execution

guage designers.

There are, however, several areas in which the tool can be improved. As of now, the integration of components running on different machines is performed through shared drives. This could be improved to use TCP/IP across a network. To mitigate this shortcoming, such integration is currently performed through code generation, so a better integration solution will be transparent to the users.

Another area for improvement is the integration with MATLAB/Simulink, which currently requires some user editing the MATLAB/Simulink model to include the generated interfaces to HLA. We leave this solution up to the language designers, though one possible approach is to generate a library of blocks that can be used, and then updates to models in these blocks will automatically update any simulation models.

8 Conclusions and Future Work

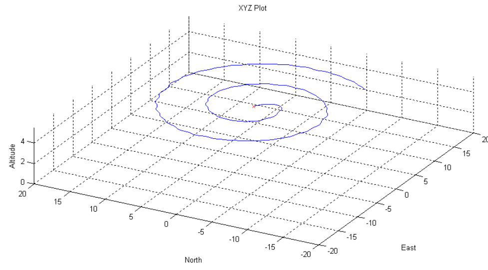
In under three months, the authors were able to integrate a new demonstration of C2 behaviors, including new controllers for the quad-rotor vehicle, new commands sent to the vehicles, new models of the demonstration, and summary simulations that verify behavior on a new installation of the infrastructure. In three months following that initial proof of concept, advanced capabilities such as camera-in-the-loop tracking and search patterns could be tested and debugged as standalone applications, and integrated in the space of a few days. These summary simulations are important for a distributed team, as they confirm to other team members that various functional components are behaving correctly, and also confirm to those teams that they can run the simulation tools required.

Our future work includes utilizing this structure in the development of high-level control algorithms for managing a group of vehicles that co-operatively search for some target(s) over some area. This would be an implementation of mixed-initiative control. The key issues would involve ensuring a stable formation and generating optimal search algorithms. The UAVs would depart as a group in response to a command, and would separate mid-way to perform individual search operations spanning the entire search area, as in Figure 8. Dividing the search space optimally, avoiding collisions and reporting back appropriate information would require the inclusion of intelligent real-time algorithms in the controller. Mesh stability is a good model to obtain a stable formation, as it attenuates disturbances acting on one vehicle as they propagate to other vehicles. Thus the UAVs travel in a mesh. This calls for decentralized control laws and intelligent search strategies.

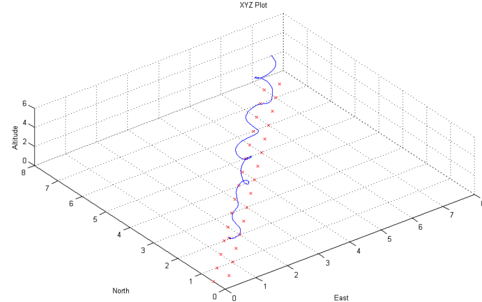
DSMs present a significant advantage in the high-level specification of system interaction, especially when the generation of the software that produces their interaction (i.e., the “glue-code” that holds an interaction together) is computational, and not a case-by-case design. We believe that future uses of domain-specific modeling environments in this domain will further enable experts in control, visualization, computer vision, etc., to put experiments of system-level simulations together more easily than a brute-force integration strategy.

Acknowledgments

This work would not have been possible without the efforts of Gyorgy Balogh, Himanshu Neema, Harmon Nine, Gabor Karsai, and Janos Sztipanovits of



(a) Dynamic simulation of STARMAC spiral search mode in simulink.



(b) Dynamic simulation of STARMAC following a truck in Simulink.

Figure 4. The controller design in Simulink, is able to do component-scale simulation and analysis. In (a) a simulation of the UAV spiral searching mode is shown. In (b) a simulation of the controller from an initial condition near the truck is shown.

the Institute for Software Integrated Systems. Special thanks are due to Gabe Hoffman, Claire Tomlin, and Hal Tharp for their generous advice in the development of the quad-rotor controllers. This work is supported by the Air Force Office of Scientific Research, under award #FA9550-06-1-0267, titled “Human Centric Design Environments for Command and Control Systems: The C2 Wind Tunnel”.

References

[1] G. Balogh, H. Neema, G. Hemingway, J. Green, B. W. Williams, J. Sztipanovits, and G. Karsai. Rapid Synthesis of HLA-Based Heterogeneous Simulation: A Model-Based Integration Approach. *Unpublished manuscript*, 2008.

[2] G. M. Hoffmann, H. Huang, S. L. Wasl, and C. J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proc. AIAA Guidance, Navigation, and Control*, 2007.

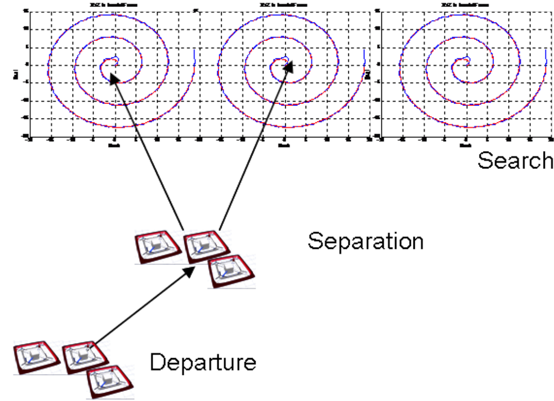


Figure 8. Co-operative Search Operations

[3] IEEE-HLA-1516. IEEE standard for modeling and simulation high level architecture (HLA)-framework and rules. *IEEE Std 1516-2000*, pages i–22, Sep 2000.

[4] K. Jensen, L. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 9(3–4):213–254, 2007.

[5] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing domain-specific design environments. *IEEE Computer*, 34(11):44–51, 2001.

[6] A. Ledeczi, G. Balogh, Z. Molnar, P. Volgyesi, and M. Maroti. Model integrated computing in the large. *Aerospace Conference, 2005 IEEE*, pages 1–8, March 2005.

[7] S. Palaniappan, A. Sawhney, and H. S. Sarjoughian. Application of the DEVS framework in construction simulation. In *WSC ’06: Proceedings of the 38th conference on Winter simulation*, pages 2077–2086. Winter Simulation Conference, 2006.

[8] T. Patki, H. Al-Helal, J. Gulotta, J. Hansen, and J. Sprinkle. Using integrative modeling for advanced heterogeneous system simulation. In *The 8th OOPSLA Workshop on Domain-Specific Modeling*, pages 80–85, October 19-20 2008.

[9] A. Varga. The OMNeT++ Discrete Event Simulation System. In *Proceedings of the European Simulation Multiconference*, June 2001.