Proceedings of the International Workshop on Multi Robot Systems, Washington, DC, March 18 - 20, 2002.

ARCHITECTING A SIMULATION AND DEVELOPMENT ENVIRONMENT FOR MULTI-ROBOT TEAMS

Stephen Balakirsky, Elena Messina, James Albus
National Institute of Standards and Technology
Intelligent Systems Division
Gaithersburg, MD 20899-8230
stephen@cme.nist.gov, elena.messina@nist.gov, albus@cme.nist.gov

Abstract

Collaborative multi-robot teams have great potential to add capabilities and minimize risk within the military domain. The composition of these teams may range from multiple copies of the same model to heterogeneous ground, air, and water vehicles operating in concert. The novel and extremely complex nature of these autonomous systems requires a large up-front investment in design, modeling, simulation, and experimentation. Myriad design decisions must be made regarding the control architecture and general information flow for commands and status exchanged between robots and humans and among robot teams. In addition, decisions must be made on how to best assemble and deploy these teams. To assist in making these design decisions, we are developing an integrated environment that we hope will greatly facilitate the design, development, and understanding of how to configure and use multi-robot teams for military applications and to accelerate the robots' deployment.

Keywords:

simulation, architectures, 4-D/RCS, autonomous vehicles

1. Introduction

Military operations are being redesigned to incorporate robotic vehicles. The recent successful missions flown by Predator and Global Hawk in Afghanistan have increased the visibility of, and confidence in, unmanned vehicles. The United States Army is transforming its combat forces to be lighter, more agile, and network centric. The Future Combat Systems (FCS) Program, run out of the Defense Advanced Research Projects Agency (DARPA) and jointly conducted with the U. S. Army will distribute the sensing, weapon delivery, and command and control elements (Gourley, 2000). A single manned vehicle will control a

series of robotic weapons and sensors. The Army Research Laboratory's Demo III Program culminated in November 2001 with a demonstration of robotic scout vehicles performing missions in concert with soldiers (Murphy et al., 2002). Key technologies – both hardware and software – are becoming available to be exploited by autonomous vehicles.

Despite the early successes of unmanned aerial vehicles and of Demo III, there is still a tremendous amount of research, design, development, and integration work to be done. There are several layers of difficulty that must be tackled. First of all, building and validating software that works in concert with complex hardware and that achieves robust autonomy is a major challenge. Integration of multiple vehicles that cooperatively execute a mission is another level of difficulty. Cooperation among heterogeneous vehicles (ground, air, water) is yet a third level of challenge. Besides the technological challenges, there exists the need for the military doctrine to evolve to be able to fully exploit these new assets. These challenges cannot wait until there are live experiments with the new autonomous platforms. Not only would it be unrealistically costly, but it would delay evaluation and design decisions until it is too late to revoke them. In order to meet the revolutionary needs of the new armed forces, simulation facilities that are integrated with software development capabilities will be crucial elements at every stage of this process.

2. An Architecture for Intelligent Autonomous Vehicles

One of the key decisions to be made in building any kind of complex system is how to organize the hardware and software. The Demo III Program and some of the teams competing for the Future Combat Systems contract have selected the 4-D/RCS reference architecture for their autonomous vehicles (Albus, 1999). The 4-D/RCS architecture consists of a hierarchy of computational nodes, each of which contains the same elements. Based on control theory principles, 4-D/RCS partitions the problem of control into four basic elements that together comprise a computational node: behavior generation (BG), sensory processing (SP), world modeling (WM), and value judgement (VJ). Figure 1 shows the 4D/RCS control node and the connections between its constituent components. Figure 2 shows a sample 4-D/RCS hierarchy for military scout vehicles.

3. Requirements for a Simulation, Modeling, and Development Environment

An architecture is a first step towards guiding and facilitating the construction of complex multi-vehicle autonomous systems. Tools that help automate the software development are another important element. NIST has been working with industry, other government agencies, and academia to investigate tools to facilitate construction of the types of large and complex systems needed by

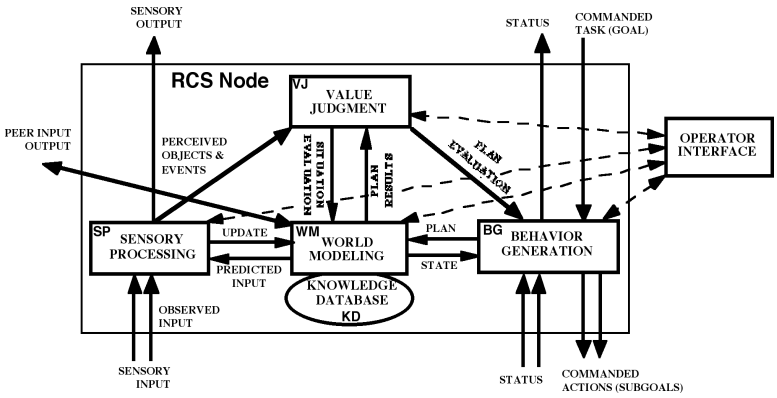

Figure 1. Model for RCS Control Node.

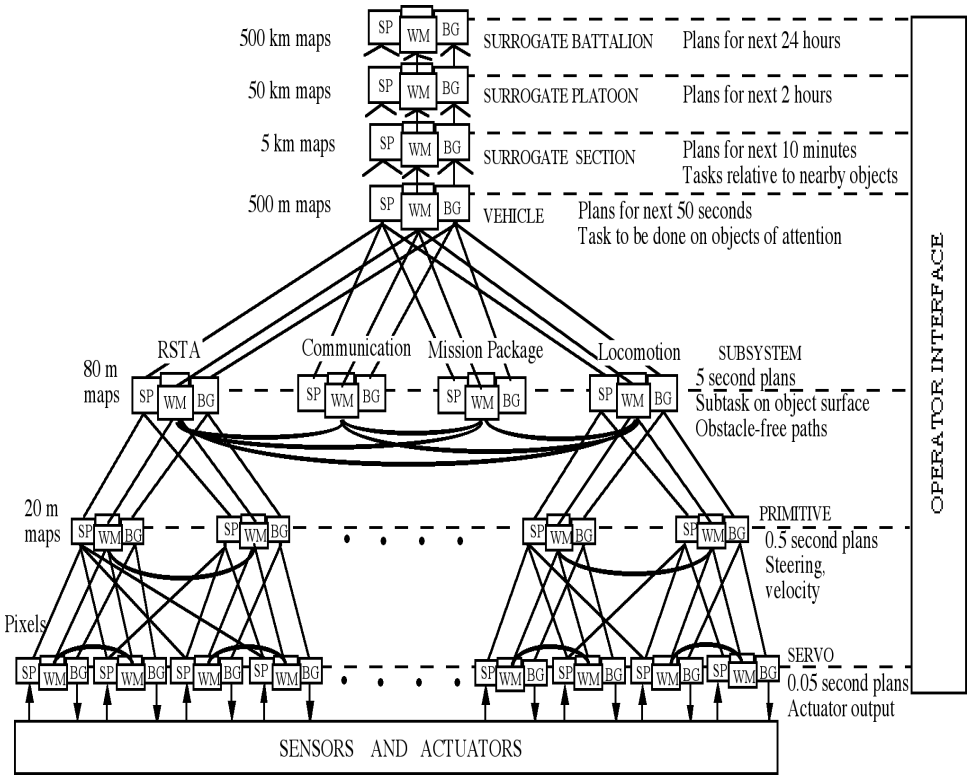

Figure 2. 4-D/RCS Reference Model Architecture for an individual vehicle.

Demo III and FCS. We are developing a large-scale simulation environment that will enable us, along with others, to design the control hierarchy, populate the control nodes, run the system in simulation, debug it, and generate code for the target host. The development and simulation environments are closely tied to the eventual deployment platforms and are intended to be able to operate with a combination of real and simulated entities. The ability to enable human-in-the loop testing and execution is also crucial, given the novel aspects of human-robot interactions.

A high-level list of the requirements for such a development and simulation environment has been developed to help guide its creation. The requirements are as follows:

- Full support of 4-D/RCS architecture
- Graphical user interface for building, testing, debugging the system under development
- Reuse support:
  - Architecture elements
  - Component templates
  - Algorithms
  - Code
  - Subsystems
- Intuitive visualizations of the control system to support design and use and provide an understanding of what the system is doing and why, and what it plans to do next. Examples of visualizations include:
  - Display of control hierarchy as it executes, including commands flowing down and status flowing up
  - Ability to "zoom in" on a particular node and view states as the system executes
  - Ability to view world models within the system
- Execution controls, including
  - Single step through execution
  - Breakpoints and watch windows
  - Logging
- Simulation infrastructure supporting realistic execution scenarios, visualization, and debugging/evaluation experiments. This includes
  - Population of the environment external to the vehicle with relevant features (such as roads, other pieces of equipment, humans, etc.)
  - Dynamic environment (with moving entities)
- Modification capabilities so that the designer and user can perform "what if" experiments. The tools should allow interactive and intuitive modifi-

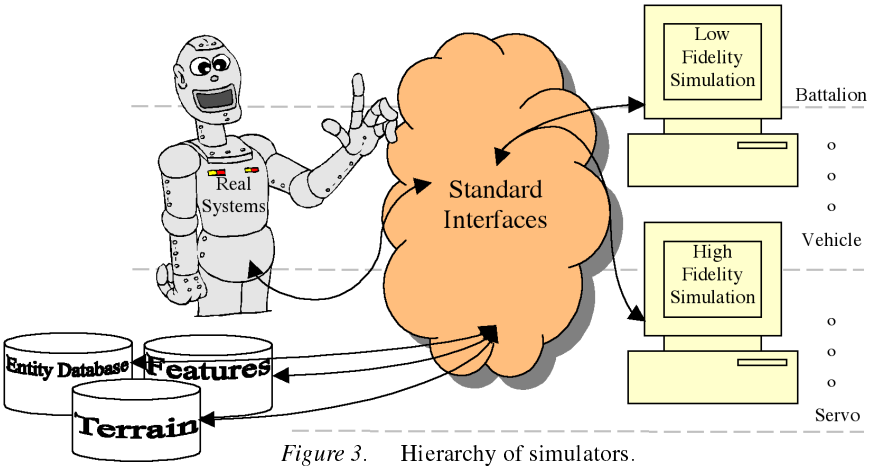

Figure 3. Hierarchy of simulators.

cation of situations in the environment or within the system. The modification capabilities should work seamlessly with the visualization, simulation, and execution features. Examples of types of modifications that should be allowed include:

- Changing world model data
- Importing datasets that represent what the system's sensors would receive
- Changing environmental conditions

4. Proposed System of Systems

We are seeking to create an *integrated* environment that provides capabilities typically associated with software development tools and those associated with simulation environments. Whereas several commercial tools exist to help design and construct software, these tool-sets are disconnected from rich execution environments (real or simulated). There also exist many sophisticated simulation systems, but they work at either a very broad scale at low resolution, or at high resolution. What we are attempting is a totally coherent environment for designing, developing, and validating a team of vehicles. The software design development support aspects include being able to work in a graphical environment to sketch the control hierarchy, bring up partially-filled in 4-D/RCS control nodes, easily create connections between nodes (or components within them), and automatically generate executable code. The software support will also encompass capabilities typically found under run-time debug tools, including single stepping and setting break points.

The software development tools will segue smoothly into the simulation environment. Under this concept, a virtual world is being created that brings

together existing multi-platform and single platform simulation systems into a system of systems. Through the use of well defined interfaces that are supported on a wide variety of computer platforms, the simulator's internal command and data flows will be able to be interrupted and modified. This will allow researchers to "plug-in" their individual technology components and override the default methods that the simulators normally employ. Through the use of these standard interfaces, we are attempting to provide researchers with a low cost technique for evaluating performance changes due to their algorithm's implementation. As shown in Figure 3, interfaces will be provided that range through the entire spectrum of the 4-D/RCS hierarchy; from a low-fidelity multi-platform configuration to a high-fidelity single platform configuration, to the ability to add real platforms into the virtual world. In addition, global variable resolution database resources will be offered. These include a terrain database that contains elevation data, a feature database that contains vector data for roads, buildings, rivers, etc., and an entity database that contains information on all of the platforms participating in the simulation. Filters will be available to tune the database outputs to the specific needs of each algorithm. For example, a low-level mobility planner may require a 40 cm cell size in an elevation array while a high-level planner may require a 40 m cell size. In addition to serving a priori data, these databases will be able to be modified in real-time. Any modifications made to the databases will be viewable by all participants (both real and virtual) in the exercise.

At the top of the hierarchy, a low-fidelity, long temporal and spatial duration, multi-platform simulator will be used. As designed, this class of simulator is capable of simulating the interaction, coordination, and movement of large groups of platforms. While these simulators do simulate down to the level of the individual platforms moving across the terrain, the terrain and mobility models are typically low resolution. Therefore, this class of simulator is best utilized in developing algorithms for group behaviors where precise platform mobility and sensing modeling is beyond the scope of the experiment. A second class of simulator will be employed for situations where precise modeling is required. These simulators will share interfaces with the low-fidelity simulator, and in fact may take commands from the low-fidelity simulators in order to precisely model one or more of the platforms involved in a particular exercise. The high-fidelity simulators will be able to read the shared databases and construct simulated sensor output (or pass real sensor output) that may be used by external sensor processing algorithms. Complex, dynamically correct platform motion models and high resolution terrain information will also be available at this level.

Interfaces will be inserted into each simulator that will enable the export and import of both world model and behavior generation information at each level of the 4-D/RCS hierarchy. This will enable researchers to implement a particular group behavior or algorithm at a particular level of 4-D/RCS. For example, a

cooperative search algorithm could be implemented at the "section" level of 4-D/RCS. The algorithm would receive its command input from the platoon level of the low-fidelity simulator and construct a plan based on information read from the terrain, entity, and feature databases. The planned course of action (individual plans for several platforms) would then be passed back into the simulator for execution. In this particular case, the plans could be passed either back to the low-fidelity simulator or to the high-fidelity simulator. In addition, one or more of the platform's plans could be passed to real systems. As designed, the source and destination of these plans and the data utilized to construct them is transparent to the planning system and totally controlled by the user. This will facilitate an environment where a researcher can simulate as many or as few subsystems and levels as desired.

5. Current Implementation

While the entire software development and simulation system has not yet been implemented, progress has been made on developing prototypes and designs for the overall system. A simulation testbed system has been built with emphasis placed on developing interfaces into the shared databases and into a low-fidelity simulator. These interfaces will allow our individual real robots to interact in simulated group behaviors, and our group behavior planning systems to plan for mixed groups of simulated/real robots. For the initial system, we have chosen the US Army STRICOM's OneSAF Testbed Baseline[1] for both the low-fidelity simulation and shared database server. We have worked closely with the Army Research Laboratory and Science and Engineering Services Inc. to install the standard interfaces. All of the interfaces communicate over NIST's NML communication channels (Shackleford et al., 2000) which provide a multi-platform solution to inter-process communication.

Distributed, shared databases are implemented as part of the standard implementation of OneSAF. We have added interfaces into the simulation system that allow for simple outside access of this information. These interfaces include hooks into the terrain elevation database, feature database, and entity database. For the terrain elevation database, both all-knowing (what is the elevation in this area, to this resolution) and modeled (what is the terrain map as modeled by vehicle x with its sensors) are available. Feature vector data is currently available only on an all-knowing basis. For entity data, filtered information (all friendly, enemy, detected, etc.) reports are available as well as event detections. Events currently supported include line crossings and anticipated line crossings with more to be added shortly.

[1] The identification of certain commercial products does not imply recommendation or endorsement by NIST.

In addition to the database access interfaces, we are able to interrupt the standard OneSAF command flow to inject our own plans. This has been demonstrated by having OneSAF section level plans sent out over an NML channel to a stand-alone vehicle level planner. The results of the vehicle level planner can then be executed on real robotic hardware, sent to a high-fidelity simulator, or sent back into the OneSAF simulator for execution.

Work is continuing on developing further interfaces. These new interfaces will include the ability for a real robotic platform to influence OneSAF databases by continuously updating their own location as well as adding detected features and entities. In addition, further breaks in the OneSAF command flow will be implemented to allow for planning systems that compute group plans to be implemented and evaluated.

In terms of the software development support, we have been experimenting with various representation techniques and development tools. These range from commercial packages, such as Real-Time Innovation Incorporated's ControlShell to novel formal languages, such as Stanford's Rapide (Messina et al., 1999). Recent work has focused on the use of the Unified Modeling Language to support 4-D/RCS control system development (Huang et al., 2001). A commercial development and execution tool for building simpler versions of RCS-style controllers has been developed by a small company (Advanced Technology and Research), but it is targeted at manufacturing systems that have minimal sensing requirements. We are currently working with outside partners to develop a prototype control system development tool that permits the types of visualizations, modifications, and execution controls as were listed above.

References

Albus, J. (1999). 4-D/RCS reference model architecture for unmanned ground vehicles. In Gerhart, G., Gunderson, R., and Shoemaker, C., editors, *Proceedings of the SPIE AeroSense Session on Unmanned Ground Vehicle Technology*, volume 3693, pages 11–20, Orlando, FL

Gourley, S. (2000) Future combat systems: A revolutionary approach to combat victory*Army*

Huang, H., Messina, E., Scott, H., Albus, J., Proctor, F., and Shackleford, W. (2001) Open system architecture for real-time control using an UML-based approach. In*Proceedings of the 1st ICSE Workshop on Describing Software Architecture with UML*

Messina, E., Dabrowski, C., Huang, H., and Horst, J. (1999) Representation of the rcs reference model architecture using an architectural description language. In*Lecture Notes in Computer Science EUROCAST 99*, volume 1798 of *Lecture Notes in Computer Science*. Springer Verlag

Murphy, K., Abrams,M., Balakirsky, S., Chang, T., Lacaze, A., and Legowik, S. (2002) Intelligent Control For Off-Road Driving. In *Proceedings of the First International NAISO Congress on Autonomous Intelligent Systems*

Shackleford, W. P., Proctor, F. M., and Michaloski, J. L. (2000) The neutral message language: A model and method for message passing in heterogeneous environments. In*Proceedings of the 2000 World Automation Conference*