Monitoring and Analysis of Multiple Agent Systems

Stephanie J. Thomas, Joseph B. Mueller, Christopher G. Harvey, Derek M. Surka

Princeton Satellite Systems
33 Witherspoon St., Princeton, NJ 08542
{sjthomas, jmueller, charvey, dmsurka}@psatellite.com
http://www.psatellite.com

Abstract. As agent-based software is more extensively deployed for the control of real-time systems, it is imperative to have tools available for the monitoring and analysis of these systems. Princeton Satellite Systems' (PSS') current ObjectAgent tools enable users to easily create and initialize agents in a multiple spacecraft system. Advanced tools do not presently exist, however, for the monitoring and commanding of agent actions in real-time. To address this issue, PSS is developing the AgentCommand Control and Monitoring System for distributed agent applications under NASA GSFC SBIR funding. A set of user-configurable Matlab software tools has been created that enables users to easily record and post-process the communications of agent-based, multiple satellite systems. Future work will build upon these Matlab playback and analysis tools to enable users to perform real-time commanding and analysis of distributed agent systems.

1. Introduction

As agent-based software is more extensively deployed for the control of real-time systems, it is imperative to have tools available for the monitoring and analysis of these systems. Princeton Satellite Systems is developing the AgentCommand Control and Monitoring System to address this issue for distributed agent applications under NASA GSFC SBIR funding.

AgentCommand is originally being developed for use with the ObjectAgent (OA) software architecture, and its first application will be on the TeamAgent formation flying testbed. The ObjectAgent system is an agent-based real-time architecture for distributed, autonomous control and TeamAgent applies OA to the problem of controlling multiple cooperative satellites. These systems were originally developed under AFRL SBIR funding in support of its TechSat 21 satellite mission, a technology demonstration program that will involve three satellites flying in formation and acting as a "virtual" satellite. The ObjectAgent cluster management software will enable the three TechSat 21 spacecraft to perform high precision formation flying to form a single virtual instrument.

The current ObjectAgent tools enable users to easily create and initialize agents in a multiple spacecraft system. Advanced tools do not presently exist, however, for the monitoring and commanding of agent actions in real-time. Users can either view predetermined agent parameters through the provided interface windows or query and

command agents in a predetermined, script-like fashion. Neither of these methods is capable of providing as robust or as varied human/agent interaction as is necessary to take full advantage of agent autonomy. Furthermore, neither tool is able to provide an overall view of the distributed system. It is desirable to have a system that is able to provide visualization of the interactions among multiple satellites, to summarize the overall status of the system, and to facilitate ease of fault detection, identification and recovery. In the future, it will even be desirable to have an environment in which agents and humans can ask questions of each other and can work together to solve problems, if not as peers then at least as master and student.

There are three specific areas of human/agent interaction that must be improved:

1. *Human understanding of agent actions*. The workings of the agent system must be made more transparent to both users and developers.
2. *Human direction of agent actions*. Humans must be able to easily and accurately command agent actions.
3. *Human/agent collaboration*. In the short term, interest lies in being able to control and understand agent actions, but in the long term, people and agents will work together to solve problems, just as people work together as teams now. Tools must be created to enable this collaboration.

The currently funded AgentCommand research only addresses the first two areas with human understanding of agent actions the primary research area. The work to date has focussed on a set of user configurable software tools that will enable users to monitor agent-based, multiple satellite systems in Matlab.

The first part of this paper describes the software tools that have been developed under the GSFC SBIR Phase I funding. This software includes:

- Agents to record important information generated by both Matlab and C++ OA agent simulations;
- A Matlab-based playback tool to replay agent-based simulations, either Matlab- or C++-based, including simulation outputs and agent activity;
- A basic set of tools to post-process and analyze these simulations; and
- An integrated agent development environment in Matlab that will enable predictive agent-network analysis tools to be incorporated in the future.

The second part of the paper describes plans for Phase II development and beyond. This work will build upon the current Matlab playback and analysis tools to enable users to perform real-time analysis of distributed agent systems. The resulting commanding tool will enable users to send messages/commands to C++ OA agents in real-time. Particular emphasis will be placed on the visualization of these systems. PSS will also begin research and development of tools for the analysis and possible prediction of multi-agent behavior prior to deployment or simulation.

2. Background on ObjectAgent

The ObjectAgent Software Architecture uses agents to implement *all* software functionality, instead of as a top layer only, and this is a key feature that distinguishes ObjectAgent from other agent architectures. This architecture allows decision-making,

including fault detection and recovery capabilities, to be built in at all levels of the software. This alleviates the need for extremely intelligent high-level agents and simplifies the software interfaces.

Each agent is multi-threaded and composed of skills. These skills are user-defined and determine the functionality of the agent. Generally, each skill corresponds to one basic function, has inputs and outputs, and triggers one or more actions. An agent is aware of its skills, inputs, and outputs, and built-in skills enable it to hunt for inputs and automatically configure itself upon launch. In this sense, an ObjectAgent system is self-organizing.

A fundamental component of ObjectAgent is the flexible messaging architecture that provides a reliable method for agent-to-agent communication both on a single processor and across networks, thus naturally supporting distributed systems. A proprietary message format is currently employed with the following characteristics:

- The format is based on simplified natural language;
- Associated with each message is a verb that signifies the type of message;
- Each message has a content field that is parsed and understood by the receiving agent;
- Each message has an optional data field that can be used to send any type of information from one agent to another;
- Each message is time-tagged; and
- Each message has fields indicating to and from whom the message is being sent.

This message format can easily be adapted to other industry formats.

During the first phase of development, ObjectAgent was prototyped in Matlab. A complete, GUI-based environment was developed for the creation, simulation, and analysis of multi-agent, multi-satellite systems. The use of Matlab enabled the proposed agent architecture to be rapidly prototyped and tested. The current Matlab software allows users to quickly verify their agent and algorithm designs prior to deployment in a real-time environment.

ObjectAgent has been ported to C++ for demonstration on a real-time, distributed testbed and eventual deployment as part of the Cluster Manager on AFRL's TechSat 21 distributed satellite mission. Tools are currently being developed to enable OA users to automatically convert their Matlab agents into C++ agents that can be deployed in real-time systems. More detailed information about the ObjectAgent software architecture can be found in [Ref. 2].

3. AgentCommand Software Tools

A primary objective in developing AgentCommand was to enable users to gain better insight and understanding into the operations of multi-agent systems. To this end, emphasis was placed on maximizing the user configurability and extensibility of the software. Many of the tools allow users to choose and create their own plots and views of the data and to store these preferences as templates. To enable post-simulation analyses, the ability to record all agent messages has been added to both the C++ and Matlab OA architectures.

The new monitoring tools allow the following types of analysis of the recorded agent communications:

- Search and sort the stored messages,
- Plot statistics of the messages as a function of any subset of agents or time, and
- View plots of message statistics and spacecraft telemetry together in one integrated playback window.

For example, you can create a plot of the message traffic between a set of selected agents, as shown in the network plot in Figure 3.1.

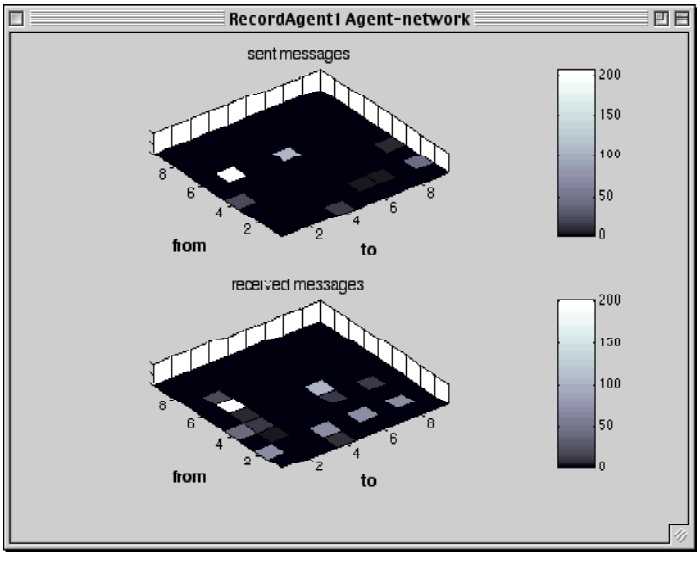

Figure 3.1: Network Plot

3.1 Agent Development Environment

The agent development environment has a palette which manages the various development windows: AgentDeveloper, HealthMonitor, AgentRelationships, and AgentInterfaces. This palette presents a clean interface to the user during development.

As part of AgentCommand, an additional development window has been created in which users can define agent communities and societies. An agent community consists of all the agents collocated on one processor. An agent society consists of a number of communities, for example three spacecraft represented by three processors, and all the agents loaded therein.

This tool incorporates porting the Matlab agents to C++, as some data, such as IP address, is specific to the agent community which is being ported. In the future, this tool will also allow users to perform pre-deployment analysis of a defined society. Users will be able to explore the agent interfaces and estimate message loads, among other analyses.

3.2 Recording Agents

In both the Matlab and C++ architectures, recording of the agent messages was implemented with a special recording agent. This allows message recording with minimal adjustments to the overall agent architecture. Agents copy both their sent and received messages to the RecordAgent. This will eventually allow matching of the messages and analysis of the time lags in the system, but now serves only as a debugging tool.

In systems with multiple processors, which are emulated in the Matlab architecture as "message centers", one RecordAgent must be placed on each processor to be monitored. Each RecordAgent only records message traffic on its home processor. The recorded messages are stored in a text file. Each message is on a separate line and the fields of the message are comma delimited. The names of the fields are stored in a header. The file format produced is the same for both C++ and Matlab simulations, allowing one set of Monitoring tools to be used.

Matlab Recorder

The Matlab `RecordAgent` has two skills, `StartMonitoringSkill` and `RecordMessagesSkill`, and uses the verb, `monitor`. The RecordAgent must ask registered agents to send it copies of their messages, which it does with the message `monitor on`. The agents that are sending their copied messages to the RecordAgent use the following message:

```
monitor messages for RecordAgent(StoreMessagesSkill)
```

This use of messages allows the RecordAgent to be consistent with the ObjectAgent philosophy that all functionality is implemented by agents and all data is transmitted through messages.

Real-Time C++ Recorder

The Real-Time RecorderAgent has only one skill, RecorderSkill. The RecorderSkill creates an outgoing socket connection to an application running on an NT workstation. This application creates a new file to record all of the message data from the RecorderAgent.

All C++ agents have an internal flag to indicate whether or not a local RecorderAgent is active. When the RecorderAgent is active, all agents send copies of outgoing messages to the RecorderAgent. This also happens with all incoming messages. When the RecorderAgent receives one of these carbon copied messages, it composes a TCP/IP packet with the appropriate fields from the message and sends them to the application mentioned above.

3.3 Monitoring Tools

The MonitorWindow, shown in Figure 3.2, is the centerpiece of the Monitoring tools. It loads recorded messages and allows a user to peruse them and plot various statistics. Note that on the right the user can decide which agents to include in a plot.

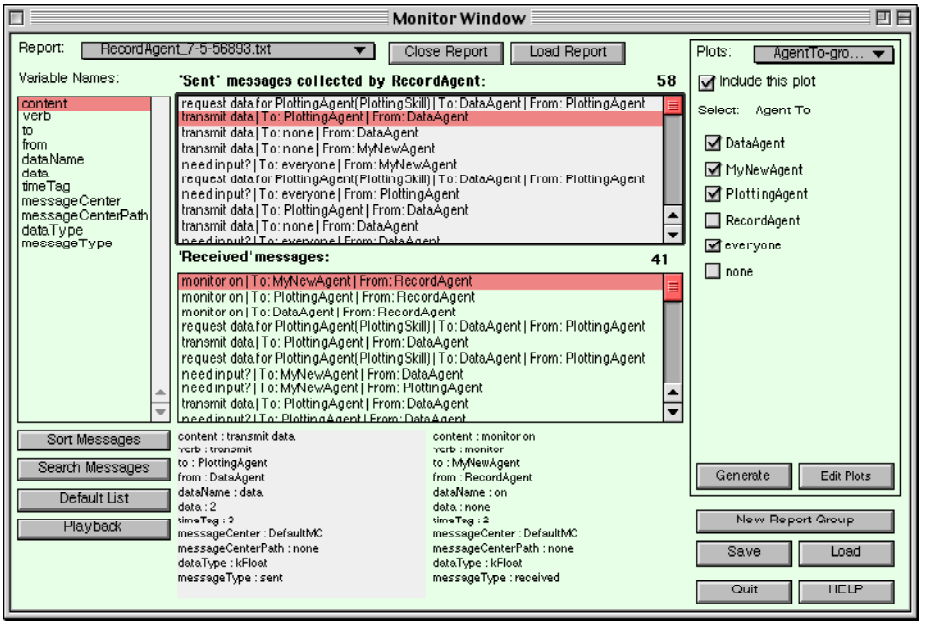

Figure 3.2: Monitor Window

The MonitorWindow is designed to load the text files created by record agents and to allow user-configurable analysis of the recorded messages. The user can define plots of the messages and include various agents or processors in each plot. Message properties can be plotted by time, agent, or processor, also known as the “community”. In addition to plotting various statistics, the user can also sort and search through the stored messages. The messages are displayed in two separate categories: those recorded while being sent, and those recorded upon receipt.

Many possible combinations exist for plot type, grouping, and variable, and these choices are left up to the user. Currently, the allowed plot types are pie chart, grouped bar plot, stacked bar plot, stair plot, and a special network plot (see Figure 3.1 on page 4). The variables will include all message fields but for now are *verb*, *to agent*, *from agent*, *time tag*, and *messageCenterPath*.

The search capability in MonitorWindow allows the user to find messages using the same variables that are currently enabled for plotting. The other message fields, including the values of the data attached to messages, will be added in future versions of the MonitorWindow tool. Once the data field has been added users will also be able to plot the values of the data passed between various agents, as the plots are generated using the search function.

Up to four criteria can be used in a search. Each criteria can be multi-valued. For example, the search to find all the messages *to* or *from* ClusterHealthMonitor with *timetags* between 10 and 20 involves three criteria, with the third criteria, *timetag*, being multi-valued.

Results

In the OA Tutorial [Ref. 4], a PlottingAgent is collecting the time from a DataAgent. To learn about the fault detection system, the user fails the DataAgent to see how the new agent MyNewAgent takes over the task of providing the time to PlottingAgent. Consider the messages generated from this tutorial example when a RecordAgent is added. Messages from DataAgent are forcibly cut off using the FailCommand feature built into OA. This can be viewed in Figure 3.3 produced by the MonitorWindow.

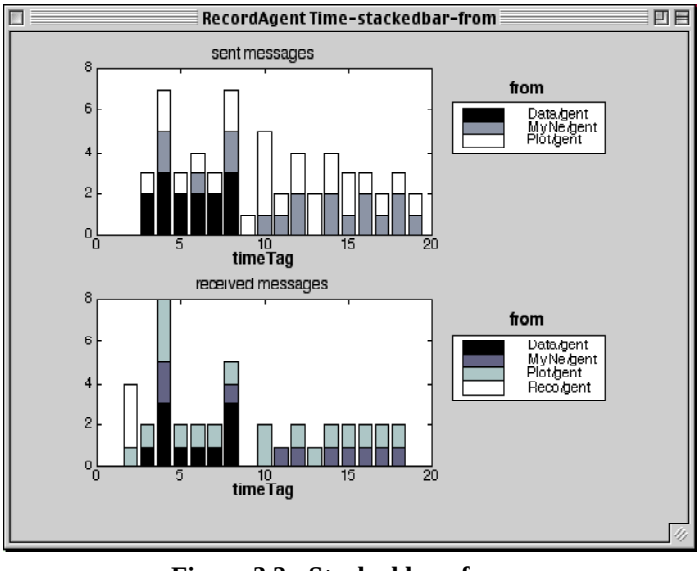

Figure 3.3: Stacked bar: from

This plot was created using the "from" variable. It is not possible to plot messages just by "Agent", as each message is associated with two agents, both a source and a destination. These plots elucidate the agent behavior during the tutorial simulation, especially the switch from DataAgent to MyNewAgent as the source of the time for PlottingAgent. For example, messages received from DataAgent abruptly end when it is failed at time step 9. A few time steps later, messages are seen to be regularly received from MyNewAgent instead.

These stacked bar plots only show the total number of messages being sent and received by the agents. One message field which is very useful to use in plots is the message verb. The plot in Figure 3.4 is a good example.

The Tutorial example only deals with one processor. A multiple-spacecraft TechSat 21 example included with OA shows monitoring of multiple processors. A ten minute simulation was run with the software for two of the four available spacecraft, SC #1 and SC #3. The total message traffic onboard spacecraft #1 is shown in Figure 3.5. Note the large amount of messages at start-up, which level off after a few minutes.

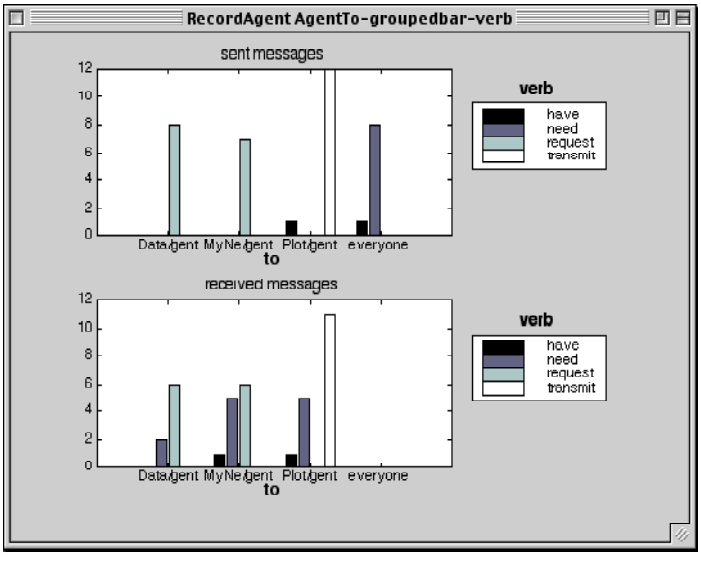

Figure 3.4: Grouped bar plot with verb information

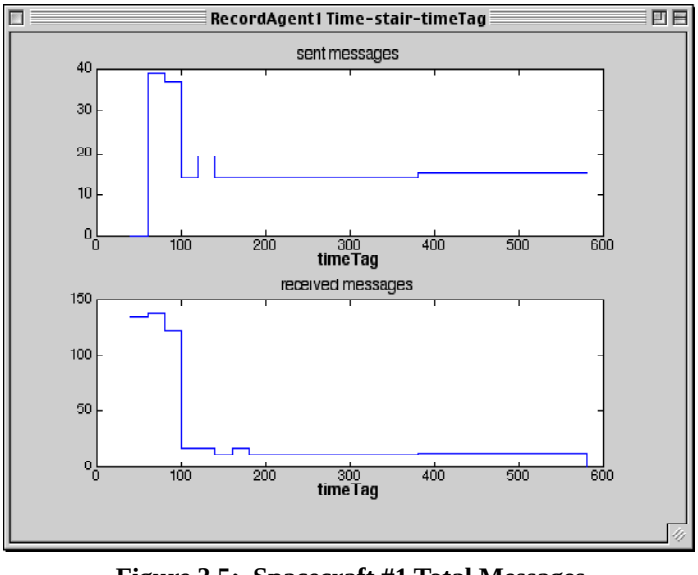

Figure 3.5: Spacecraft #1 Total Messages

3.4 Telemetry Plotting

In addition to reviewing the agents' messages, the ability to manipulate the spacecraft telemetry is also crucial to the agent monitoring functions. The agent behavior must be related to the performance of the spacecraft. The Plotting Tool, shown in Figure 3.6,

was designed to provide a uniform yet flexible way to view the results of the spacecraft simulations. Both raw and derived data can be visualized with a highly configurable plotting capability. Three-dimensional variables can be animated. Plotting preferences are saved in a template.

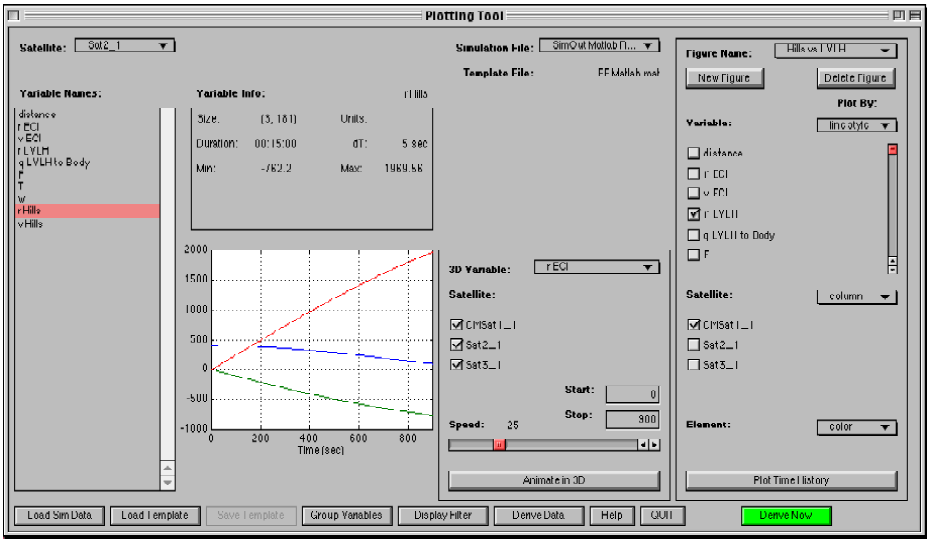

Figure 3.6: Plotting Tool

Templates act as filters on the raw data, changing how it appears to the user. When a simulation file is loaded into the Plotting Tool, the template is blank. At this point a user may load a previously defined template, or simply begin working with the tool to build up a new template. Four types of user preferences are stored in a template:

- Grouped variables
- Excluded variables
- Figures
- Derived data

Each of these categories impacts how the simulation data appears to the user.

In general, the data has three descriptive components to it: variable, satellite, and element. A given variable may have one or multiple elements, and a unique copy of each variable exists for each satellite. For a given figure, the user may choose which variables and which satellites to include in the plot. The user may also define the manner in which these three components are organized within the figure.

Each component may be assigned one of the following four plotting characteristics: row, column, color, and linestyle. Organizing by row or column causes separate subplots to be created within the figure, while the color and linestyle attributes distinguish the components within a single plot.

Figure 3.7 provides an example of the type of figures that can be generated with the Plotting Tool. It shows the relative position and velocity of three spacecraft. The *x*, *y*, and *z* elements are separated by row, the **r** and **v** variables separated by column, and the

satellites distinguished by linestyle. The figure could be easily generated again in several different formats. For each figure created, the title, variables, satellites and all plotting characteristics are stored in the template.

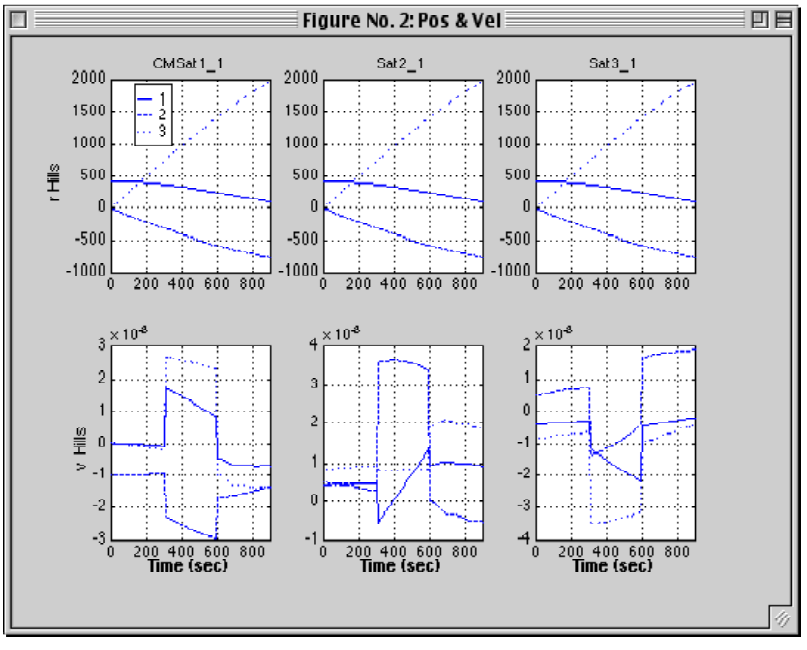

Figure 3.7: Plotting Tool Generated Figure

An additional feature of this tool is that it automates the process of deriving new data from raw simulation telemetry. The two variables plotted in Figure 3.7, "**r Hills**" and "**v Hills**", were not included in the simulation output — they were derived automatically upon applying the template. In this case, a derivation function was used which takes the position and velocity of the satellites in the ECI frame as inputs, performs a coordinate transformation, and returns the relative states in the local Hills frame.

The Plotting Tool also allows users to view animations of any 3-dimensional variables. Upon choosing a variable to animate, the user may then choose which satellites to view, the start and stop times of the animation, and the speed.

A key application of the 3-D animation feature is viewing the relative motion of satellites in a local reference frame. In formation flying, the objective is to control the relative position and velocity of the satellites such that a particular shape or configuration is maintained. The manner in which these relative states evolve over time can be viewed nicely through animation. The "**r Hills**" variable displayed in Figure 3.7 could also be animated to illustrate this relative motion.

Spacecraft telemetry plots that are defined in the Plotting Tool and saved in a template can be viewed in playback alongside agent message analyses from the MonitorWindow.

3.5 Simulation Playback

The playback window can be opened from the MonitorWindow. This tool, which will continue to be useful for post-simulation analysis even after real-time monitoring tools are developed, allows the recorded simulation to be stepped through using any time step. The user can select up to nine plots from the MonitorWindow or from the spacecraft telemetry PlottingTool for display in playback. The tool as shown in Figure 3.8 includes two telemetry plots and two plots of agent message data.

The least effective plots to view in Playback from the MonitorWindow are bar plot with time on the x-axis, as no new information is gained during the playback. However pie and grouped bar plots, such as those shown, provide additional information about the message composition in a specific time interval. Network plots are also a good choice for playback.

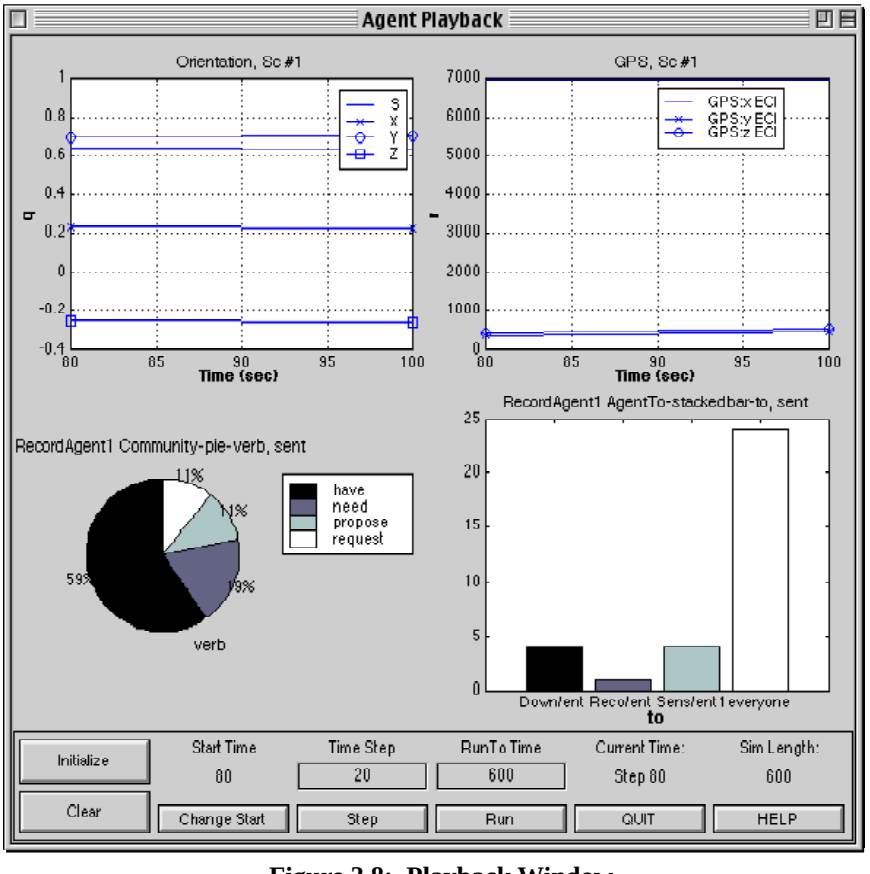

Figure 3.8: Playback Window

The window was designed as a console with the idea that in real-time systems, operators would be looking at various pre-defined plots in addition to viewing the mes-

sage traffic as text. Additionally viewing the plots in one window is logistically easier than clicking through numerous plot windows.

4. Future Plans

A proposal for Phase II of AgentCommand is currently under consideration. In Phase II, the prototype monitoring tools would be upgraded from Matlab playback-type monitoring to real-time monitoring. An advanced control center which allows commands to be sent to the agents in real-time would be fully developed. It is not possible to develop such a control center in Matlab due to inherent limitations in the Matlab environment. Lastly, research will be conducted into various types of pre-deployment agent analysis and verification.

After the Phase II work is complete, AgentCommand will be commercially available both as part of PSS' ObjectAgent system and also as a stand-alone product. Although the commanding and monitoring tools will be developed with ObjectAgent in mind, interfaces to other agent packages will also be considered. The different aspects of AgentCommand - post-simulation monitoring, real-time commanding and monitoring, and pre-deployment analysis of agent societies - may be applicable to various existing software packages.

The following sections describe the work planned for the Phase II effort.

4.1 Real-Time Commanding and Monitoring

The Matlab tools will be finalized, and the requirements specification completed, before the transfer to Java real-time tools begins. The following functionality is expected to be added:

- Match up sent and received messages to determine time delays, which may require adding a unique identifier to each message.
- Expand plotting and searching to include all messages fields, including the data.
- Add advanced pre-deployment analysis techniques to agent society definition.
- Add user preferences (i.e. different logins) if desired.

The command center will use TCP/IP processes to send commands and receive replies. The agent dispatcher will have to be modified to handle the case of messages being sent to the command center.

The command center GUI will consist of a message building window which will check the grammar and allow data to be attached to the commands, plus some windows to view incoming messages, which may be replies to commands or requests for information or help by an agent. It is expected that minimally, at the end of Phase II, a user will be able to command agents to perform tasks or send telemetry and monitor the results of these commands. Agents must have the capability to respond to commanded tasks written into their skills, for example a ReconfigureAgent might have skills which allow it to respond to the command, "`calculate new formation`".

4.2 Demonstrations

Several types of demonstrations will be performed. Some possibilities are listed below.

- PSS OA testbed - this real-time testbed uses the OSE operating system. This will be the primary demonstration for AgentCommand and will include a detailed formation flying demonstration involving multiple spacecraft.
- Softkernel - this is the emulator for OSE. A demo created in this environment will be portable and independent of the expensive board systems. This demo is complimentary to using the actual PSS testbed.
- GSFC Formation Flying Testbed - With sufficient time, it may be possible to incorporate GSFC algorithms into OA agents and to integrate these with the GSFC testbed. Alternatively, pre-existing algorithms, such as from our CETDP formation flying contract, may be used.

4.3 Pre-Deployment Analysis of Agents

Research will be a key part of this task. Prototype analysis tools will be developed and user-tested from the results of study into this area. The pre-deployment analysis would occur on a user-defined agent society. Examples of analysis that will be considered are latency predictions, stability analyses, and deadlock prediction.

5. Conclusions

The new prototype tools developed under AgentCommand enable the post-simulation monitoring and analysis of the ObjectAgent message traffic. These tools are a necessary prerequisite to developing an advanced agent control center; before these tools were developed, there would have been no insight into the results of commands. The prototype tools have already proved very useful in debugging and understanding agent-based software. Taking the time to develop these tools first has the added advantage of providing ideas for desirable pre-deployment analysis.

The next step will be the transition from playback-type analysis to real-time commanding and monitoring. Although developing the prototype monitoring tools took only six months, this next design phase is expected to take two years. The resulting AgentCommand Control and Monitoring Center will provide a sophisticated interface to the software agents with advanced visualization techniques.

PSS knows that in order for ObjectAgent to be accepted into mainstream software applications, it is crucial that users be able to more fully understand the behavior of agents. The ability to interact with agents in real-time has been a cornerstone of the envisioned OA architecture. Together, the addition of these commanding and monitoring tools to the real-time version of OA will greatly enhance its applicability to critical real-time systems.

References

1. Mueller, J. B., D. M. Surka, and B. Udrea, “Agent-Based Control of Multiple Satellite Formation Flying,” The 6th International Symposium on Artificial Intelligence Robotics and Automation in Space, Montreal, Canada, June 2001.
2. Surka, D. M., M. C. Brito, and C. G. Harvey, “The Real-Time ObjectAgent Software Architecture for Distributed Satellite Systems,” 2001 IEEE Aerospace Conference Proceedings, Big Sky, Montana, March 2001.
3. PSS Staff, *AgentCommand: A Control Center for Autonomous Distributed Satellite Systems, Phase 1 Final Report*, Princeton, NJ, August 17, 2001.
4. PSS Staff, *ObjectAgent Tutorial*, Version 2.1, Princeton, NJ. Available from http://www.psatellite.com/product.tpl?sku=10014