ObjectAgent for Robust Autonomous Control

Derek M. Surka

Princeton Satellite Systems
150 S. Washington St., Suite 201
Falls Church, VA 22046
dmsurka@psatellite.com

Abstract

The ObjectAgent system is being developed to create a robust software architecture for autonomous control of complex systems. Agents are used to implement all of the software functionality and communicate through simplified natural language messages. These agents have a set of basic survival skills that monitor for internal software faults, providing low-level fault detection and recovery. Higher-level fault detection and recovery capabilities, including modern artificial intelligence techniques, can easily be incorporated into an ObjectAgent-based system.

Introduction

Princeton Satellite Systems is developing the ObjectAgent (OA) system to create an agent-based software architecture that is designed for autonomous, distributed systems. The system is being designed to deal with traditional real-time software issues, such as memory management and deadlock, and to provide a robust architecture that can be used to implement modern fault detection techniques.

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. Collision avoidance and reconfiguration simulations were performed for a cluster of four satellites. ObjectAgent is now being ported to C++ and the present architecture runs on a PowerPC 750 running Enea's OSE operating system.

ObjectAgent is scheduled to fly on the Air Force's TechSat 21 satellite program in 2003. TechSat 21 is a mission that will involve three satellites flying in formation and acting as a "virtual" satellite. ObjectAgent will be used to build two elements of the flight software, the Cluster Manager and the Spacecraft Manager.

The Cluster Manager is a flight software package that controls all spacecraft operations that require the coordination of multiple spacecraft. It also provides complete fault detection of all cluster operation related systems. One of the primary functions of the Cluster Manager is to perform relative control of the satellites in the cluster. This will include relative stationkeeping and estimation of the cluster center-of-mass and the relative positions of each satellite.

The Spacecraft Manager is a flight software package that provides an autonomous replacement for the ground operations team. It will control all aspects of spacecraft operation including fault detection and redundancy management. The Spacecraft Manager provides an interface between the Cluster Manager and the rest of the TechSat 21 flight software.

Although ObjectAgent is being originally developed for distributed satellite systems, its applicability extends to other distributed and complex systems. These include systems of air, surface, and submersible vehicles, process control, and network control and administration, to name a few.

Previous papers have addressed the basic Matlab architecture of ObjectAgent and have described the research into agent organizations for distributed satellite control (Schetter, Campbell and Surka 2000ab). Papers have also described the basic C++ architecture (Surka, Brito and Harvey 2001) as well as the application of ObjectAgent to the TechSat 21 program (Zetocha et al. 2000). This paper focuses on the built-in features of ObjectAgent that improve system robustness and autonomy and describes some of the innovative fault detection and recovery techniques currently under development.

The first section provides a general overview of the core ObjectAgent system. This includes a discussion of the basic survival skills possessed by agents. The second section describes the approach being taken to address traditional real-time software issues. The third section describes the recently added layer of background health monitoring and error handling. (See (Mueller, Surka and Lin 2001) for more detailed information.) The final section describes the new fault detection techniques currently under development.

Overview of ObjectAgent

ObjectAgent is an agent-based, message-passing software architecture that uses agents to implement all software functionality. Agents are the basis of the system rather than just a top layer. This is a key feature that distinguishes ObjectAgent from other agent architectures. Each agent is a

multi-threaded process and this architecture allows decision-making, including fault detection and recovery capabilities, to be built in at all levels of the software. This in turn alleviates the need for extremely intelligent high-level agents and simplifies the software interfaces.

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. Each message has a content field written in natural language that is used to identify the purpose of the message and its contents. Natural language was selected so that users could easily send messages or commands to the agents as well as understand the messages being sent between agents. The latter is important for debugging purposes. Agent communication takes place solely through messages; there is no shared memory between agents. This ensures that agents can work together even when they are not located on the same processor.

Two additional advantages of using agents for all software functionality are increased flexibility and robustness. Robustness is improved in ObjectAgent because all agents are endowed with a set of basic survival skills. Each agent has knowledge of its skills, inputs, and outputs, and is capable of automatically configuring itself upon launch. It will automatically seek out other agents who can provide it with the inputs it needs as well as other agents who need its outputs. In this sense, an ObjectAgent system is self-organizing.

These same survival skills enable agents to be dynamically added to a system to improve the system capabilities or recover from a failure. The flexible and reconfigurable messaging architecture provides a common software interface that is vital to this ability to dynamically add or change software. Since all software is implemented as agents with the common messaging interface, all software can be easily replaced or updated. This messaging architecture also helps the system to recover from failures.

Another key feature of ObjectAgent is it allows the user to specify the complexity of the agents and agent organizations and does not constrain users to a predefined notion of an agent. The user performs the decomposition of the system into agents. This allows greater flexibility, extensibility, upgradability, and compatibility with existing systems.

Although artificial intelligence techniques are not built in to the ObjectAgent core, the OA system architecture allows AI techniques to be incorporated at any or all levels of the software. Many tools are available to create agent skills. For example, the system includes fuzzy logic, neural net, system identification, learning control, expert system and fault detection tools that the user can employ to solve his or her distributed control problems. These techniques can even be added after the system is in operation, which is not possible with today's flight systems. In addition, tools for enabling agent organizations are included. This permits agents to control the behavior of other agents.

Finally, special attention has been paid to developing a system that is easy-to-use and simplifies the flight software creation process. ObjectAgent is an integrated approach to agent and flight software design, making extensive use of simplified natural language and graphical user interfaces (GUIs). This design environment not only simplifies the agent creation process but also provides a common interface to a number of advanced control and estimation techniques.

Real-Time Software Issues

Since ObjectAgent is being implemented in real-time control systems, the OA architecture must address traditional real-time software issues such as process and memory management and deadlock. It is not possible to address these issues in the single-threaded prototype Matlab environment and work on these issues has just begun. This section describes how we propose to initially address these issues in the C++ implementation of ObjectAgent.

The initial C++ version of ObjectAgent will rely on the real-time operating system (RTOS) to address some of these issues. Enea's OSE was selected as the first operating system for ObjectAgent because OSE is a message based RTOS designed for distributed systems and has many features that lend itself to the ObjectAgent architecture. These features include multi-threading, very good process and memory management, and dynamic process loading.

Unlike traditional embedded operating systems, which utilize lightweight tasks to partition complex activity and semaphores to establish communications, a messaging RTOS uses memory-protected processes and message-based communications. This approach makes it easier to conceptualize complex applications and distribute programming responsibilities across large development teams. The messaging RTOS model also makes it easier to compartmentalize critical operations and data, thereby enhancing reliability and security.

The OSE kernel provides very good process and memory management. Processes may be grouped into blocks, each with its own memory pool. While other kernels may schedule tasks running in a shared memory environment, OSE knows what resources each owns, including such things as file descriptors, sockets, as well as all memory resources, and supervises to avoid conflicts. If tasks die, the kernel can reclaim the resources automatically.

OSE is a true message passing designed for distributed processing applications and features full central processor unit (CPU) or destination transparent messaging. The messaging schema naturally supports fault tolerant and/or high availability designs, in the following fashion:

Processes send messages to other processes;

Processes dynamically bind to other processes; and

OSE supervises all communications between processes; if delivery fails, or a process dies (or is killed), all connected processes are notified so they may take corrective action. One corrective action could be to establish a connection with a backup process (or board)

or messages may be dynamically re-routed to alternate destinations.

Beyond the built-in protection provided by the operating system, the ObjectAgent architecture will provide additional features to address traditional real-time software issues. One such feature is the inclusion in every OA-based system of a "hall monitor" agent on each processor. This agent will monitor all other agents running on the processor to detect run-away processes and deadlock. The hall monitor will then have the ability to shut down any such problem agent. The details of the hall monitor agent are still being worked out.

Health Monitoring and Error Handling

To create a robust, autonomous system, it is not enough to only address the issues associated with developing real-time software. The final system must also be able to detect and recover from other subsystem faults. A background layer of health monitoring and error handling has been prototyped in the ObjectAgent Matlab environment to enable systems to detect and recover from these faults

One of the most important tasks involved with controlling a complex system is to provide information about the system's health. Satellites are an excellent example—several important parameters, such as temperature, fuel, attitude, and battery charge, are constantly being stored in memory and telemetered down to earth. In an agent-based system, where the computational tasks are distributed, it is especially important to monitor health. The ObjectAgent health monitoring architecture accomplishes two goals. First, it enables the user to easily define how agents conduct health monitoring. Second, it carries out health monitoring in the background for an agent network of any size.

The information provided by the health monitoring architecture can be used by agents to detect existing or potential problems in the system. Each agent has its own health report, which is defined by the user. A health report consists of three types of information: an overall *health number*, a set of *important parameters*, and a descriptive *list of errors*.

The health number of an agent is an instantaneous measure of "how well" the agent is doing. The current method of measuring the health number is to begin with a nominal value of 100. If the agent experiences an error, its health is reduced by the severity associated with that error. When the agent recovers, its health is restored by the same amount.

A more detailed set of information is included in the important parameters list. All skill outputs of an agent are available to be selected as important parameters. They are measured at a specified rate, and their name, value, and time of measurement are included in the health report.

The final element of the health report is a list of errors. Each time an agent detects an error, a packet of relevant information is stored in the agent memory. When it is time to send the next health report, any new error information is included.

Once a network of agents is established, an easy-to-use HealthMonitor GUI enables the designer to specify the method of monitoring health among agents. It should be noted that both external parameters (e.g. temperature measurements) and internal parameters (e.g. time to run an algorithm) may be monitored. What the designer decides to do with this information is, of course, case specific.

The fault detection architecture for ObjectAgent was designed with two objectives in mind: 1) to provide a flexible framework to detect, report, and recover from errors in a distributed agent environment; and 2) to minimize the amount of required user-intervention. The user is required to supply the necessary algorithms for detection and recovery. The ObjectAgent software then uses this information to implement the error handling.

The main element of the fault detection architecture is an error. In ObjectAgent, an error is defined as follows:

> *A specific occurrence, with a unique name, that may be detected and recovered from in a distinct manner.*

There are three primary actions surrounding the occurrence of an error: detection, recovery, and reporting. In an agent-based framework, once an error is detected, that knowledge is initially isolated to the original agent. The recovery may be performed by the original agent or by an outside agent, providing it has been informed. ObjectAgent uses its message passing architecture to distribute error information to all appropriate agents, thereby allowing a distributed recovery approach to be used.

The types of errors which may occur are divided into 2 categories: *Input Errors* and *Skill Errors.* An Input Error is any error that has to do with a specific input to a skill, while a Skill Error is an error that occurs while a skill updates. For example, "No Input" is a generic Input Error that occurs when an input is expected, but is not received. The default action is to seek for a new source. The detection and recovery algorithms for this error have been written into an FDIR function, which may be applied to any input of any skill in any agent of the system.

Errors may be either environmental or software oriented in nature. An error such as "Bad Signal" or "High Temp" would be environmental, whereas "No Input" is software oriented because it refers to a skill not receiving an expected input from another software agent.

Finally, the user may select which agents are informed when an error occurs. It is useful to report errors to other agents for two reasons. First, other agents are likely to be affected by an error. If they are informed, then they have a chance to minimize the negative impact that the error may have on them. Second, it may be useful for other agents to assist in correcting the error.

The following example, which is used in the ObjectAgent Tutorial, demonstrates the error handling features of ObjectAgent. A simple network of three agents is simulated. The structure is illustrated in Figure 1.

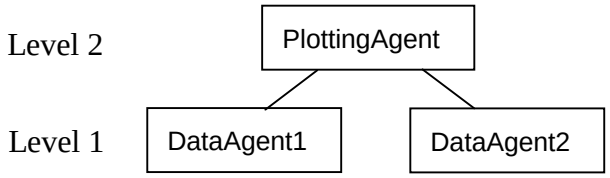

Figure 1: Simulated Network

DataAgent1 and DataAgent2 are identical; each possesses the skill "DataSkill", which simply calculates the system time and sends it as an output called "data". PlottingAgent has the skill "PlottingSkill", which plots the input "data" as a function of time. Prior to running the simulation, DataAgent1 is defined as PlottingAgent's source of "data".

The results of the simulation are shown in Figure 2. The goal is to illustrate that the error handling architecture works properly by using the "No Input" FDIR function. If the "No Input" error is detected by PlottingAgent, it should seek for a new source, and find that source in DataAgent2. PlottingSkill is coded such that, when the source of "data" changes, it plots "o" instead of "x".

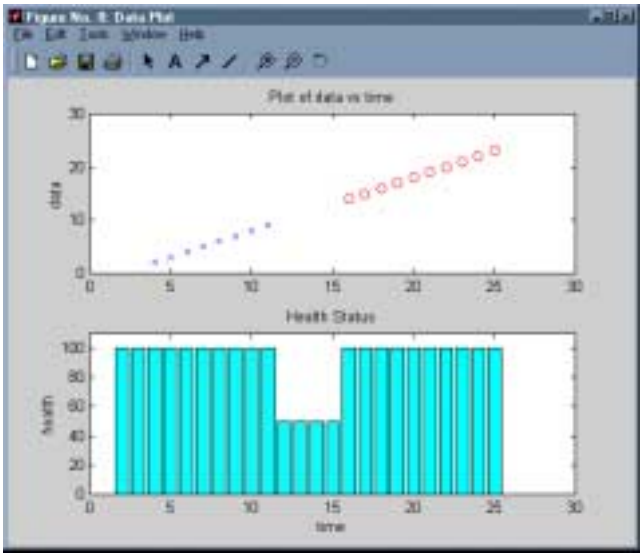

Figure 2: Health and "data" of PlottingAgent when

At 10 seconds DataAgent1 is failed, preventing PlottingAgent from receiving "data". PlottingAgent detects the "No Input" error at 12 seconds (due to a 2 second delay), and reduces its health by the severity of the error, which was set to 50. After 4 seconds pass, the recovery algorithm is shown to be successful when "data" is received from DataAgent2.

Although this is an extremely simple example, it illustrates the ease with which error handling may be executed. It also demonstrates the dynamic reconfigurability of agent communications that is built-in to the core ObjectAgent architecture. The detection and recovery scheme of this particular error may be applied to any input and more advanced schemes can easily be implemented. Advanced fault detection techniques currently under development at PSS are described in the following section.

Advanced Fault Detection Techniques

All fault detection in ObjectAgent is performed by agents and an agent's functionality is determined by the skills that it possesses. Therefore, fault detection is incorporated into ObjectAgent by creating special fault detection skills. Both traditional (e.g. if…then…else constructs) and advanced fault detection techniques can easily be incorporated into agent skills. The system currently includes fuzzy logic, neural net, system identification, and expert system tools that the user can employ for fault detection. A number of additional advanced techniques are also under development.

Decision making is centralized using the Autonomous Dynamic Algorithmic Expert System (ADAXS). ADAXS is an expert system that is designed to handle time-varying data and to apply algorithms (for example an FFT) as part of its decision making process. It can be thought of as a combination of expert system and mathematical language such as Mathematica or Matlab. ADAXS is designed to mimic the behavior of people when they are trying to solve problems.

Data is loaded into ADAXS using standard OA messages. The datafield may contain a string object or Matrix object plus all standard C/C++ data types. New data can be appended to datafields using messages. Decision making involves processing the data using filters and other transforms and then making a decision based on the transformed data. The decision making process uses production rules and the rules make take any one of several forms including if...then...else and Fuzzy logic constructs.

Additional tools exist for clarifying input data. For example, for fault detection, Detection Filters are available. Detection filters are fixed-gain Kalman Filters with the gain matrix designed to make the filter residuals sensitive to actuator and sensor failures and plant model changes. The residuals are normally zero unless a failure occurs.

Another technique under development at Dartmouth University is Prof. Minh Phan's Interaction Matrix Formulation for actuator failure detection. This formulation allows the software to detect actuator failure immediately, based upon past and current input/output data and an efficient filter. This software is robust with respect to measurement noise, outside disturbances, and errors in system modeling. Furthermore, it uses a minimum of computational resources and applies to single input, single output systems as well as multi-input, multi-output systems.

Conclusions

In summary, the ObjectAgent software architecture is designed to increase the autonomy of complex distributed systems. It does so by providing a robust, agent-based software architecture that enables advanced fault detection techniques to easily be added to real-time systems. The message-passing operating system addresses many of the traditional real-time software issues such as process and memory management. Each agent has a set of basic survival skills that creates a robust communications network. A basic fault detection architecture has been created for ObjectAgent that facilitates both health monitoring and error handling for an agent network of any size. Advanced fault detection techniques, such as the ADAXS and Interaction Matrix Formulation systems under current development, can easily be incorporated into this architecture.

Acknowledgments

This work is supported by two United States Air Force SBIR Phase II contracts from the Surveillance and Control Division of the Air Force Research Laboratory's Space Vehicles Directorate. The contract numbers are F29601-99-C-0029 and F29601-00-C-0025 and the program manager is Paul Zetocha.

References

Mueller, J. B., D. M. Surka, and J. J. Lin. 2001. A Background Layer of Health Monitoring and Error Handling for ObjectAgent. To be Presented at FLAIRS 2001. Key West, Florida.

Schetter, T. P., M. E. Campbell, and D. M. Surka. 2000a. Comparison of Multiple Agent-based Organizations for Satellite Constellations. In Proceedings of FLAIRS 2000. Orlando, Florida.

Schetter, T. P., M. E. Campbell, and D. M. Surka. 2000b. Multiple Agent-Based Autonomy for Satellite Constellations. In Proceedings of the Second International Symposium on Agent Systems and Applications. Zurich, Switzerland.

Surka, D. M., M. C. Brito, and C. G. Harvey. 2001. Development of the Real-Time ObjectAgent Flight Software Architecture for Distributed Satellite Systems. To be Presented at IEEE Aerospace Conference 2001. Big Sky, Montana.

Zetocha, P., L. Self, R. Wainwright, R. Burns, M. Brito, and D. Surka. 2000. Command and Control of a Cluster of Satellites. *IEEE Intelligent Systems*. November/December 2000.