4 AN AGENT-OS FOR AUTONOMOUS COOPERATIVE SYSTEMS

Francisco Maturana[1], Raymond Staron[1],
Pavel Tichý[2], Pavel Vrba[2], Charles Rischar[1],
Vladimír Mařík[2] and Kenwood Hall[1]
Rockwell Automation
[1]1 Allen-Bradley Drive, Mayfield Heights, OH 44124-6118, USA
[2]Pekarska 10a, 15500 Prague 5, Czech Republic
{fpmaturana, rjstaron, ptichy, pvrba, cmrischar, vmarik and khhall}@ra.rockwell.com

We are interested in providing an agent infrastructure for truly distributed control. Requirements include multiple language implementations so that this agent host environment can exist in real-time controllers. Our first infrastructure, the Autonomous Cooperative System (ACS) was built as a low priority control task within the operating system of the automation controllers. The environment added the functionality to host a set of agents. This ACS kernel evolved for a long time almost independently of the controller operating system. New releases of the controllers will cause complications in the integration of the software and adoption of new functionality. We need to evolve ACS to be seamless integrated with the controllers. These changes will affect the infrastructure. In this document, we discuss the implications and results of the transformation.

1. INTRODUCTION

The use of intelligent agent technology in industrial control has revealed very important discoveries for building a new breed of automation systems. It has been shown that intelligent agent systems can cope with complex requirements for managing distributed information because agents integrate information dynamically as a normal aspect of their operation. In particular, systems with physical and logical redundancy are excellent candidates to implement agents.

Knowledge from each automation component impacts the overall process locally and globally. System engineers need to oversee these impacts at design time to program the rules for controlling the process and for mitigating the negative impacts on the system when unforeseen scenarios occur. To execute successful production, the information infrastructure needs to be built with a lot of contingencies to prevent malfunctioning. Here is where we find the primary obstacle to flexible and balanced automation. In classical control terms, the integration of the system is subordinated to the maladies of islands-of-automation. However, there are cases where islands-of-automation are still a necessary structure.

Intelligent agent systems, however, will not solve all problems in automation and information systems, but they do provide a reference for distributing knowledge and information. There have been many contributions from the research community

to characterize agents and their frameworks. According to those studies (Brennan, 2003), (Shen, 2001), (Wooldridge, 1995), agents can exist at opposite ends of the distribution spectrum. At one end, agents can be highly granular to act on behalf of a micro-process such as the opening and closing of a valve. At the other, they can be larger decision-making actors such as the type required in process planning and scheduling. Agents can be found anywhere within that spectrum. Each side of the spectrum imposes requirements on the communication and computing.

Some interesting aspects in building agent systems have been the design of agents and the encapsulation of the agents within adequate computing hardware. These considerations can be very critical in large heterogeneous systems in which the hardware and processes are dramatically different but interdependent.

The Foundation for Intelligent Physical Agents (FIPA) (FIPA, 1996) established the need for agent technology to link to the physical world. FIPA now has working groups involved in the manufacturing production and scheduling areas for solving the real-time control interface problem. The FIPA specification is a good start in the direction of a common framework for building agents. However, there is no available specification to make agents transit to the production environment without breaking legacy rules. Thus, what are the essential characteristics of the hosting devices to enable the operation of agents in different contexts?

To make a computing unit agent enabled, it is necessary to identify the middleware as a set of functions to be integrated with the device's operating system (OS). Moreover, the middleware need to be integrated with the hosting OS seamlessly to avoid interruptions during the execution of the control tasks. Agent computing is demanding on the CPU and the communications because of the agent-to-agent messaging and reasoning. Hence, to integrate agents with controllers, we need to consider the agent functionality as another part of the OS.

Our original agent environment, the Autonomous Cooperative System (ACS) (Maturana, 2002), (Staron, 2004), (Tichy, 2001), was implemented in C++ since it was intended for controllers that currently support this type of programming language. The agent environment was successfully tested on multiple industrial applications such as steel rolling mill factory, material handling, and shipboard automation. Most recently, the infrastructure has been used to model complex networks for power distribution, beverage distribution matrix, and water/waste water systems in simulation. Although ACS has been successfully deployed, the system presents overheads in computation time and memory consumption, two resources that have other critical demands on them in a real-time control system. Thus two goals for improving ACS are 1) reducing agent demand on these resources without sacrificing functionality or flexibility, and 2) minimizing the work effort needed to create new versions of ACS for future revisions of the controller firmware.

2. BACKGROUND WORK IN THE AREA OF AGENT CONTROL DEVICES

The concept of a real-time interface between the soft world of agents and the physical world of machinery is tied closely to some fundamental ideas in Holonic Control Devices (Brennan, 2003), (McFarlane, 2000), (Marik, 2001), (Vrba, 2005). This consortium defined some properties of such control devices, but not from the perspective of changes to existing real-time operating systems used in

manufacturing. The agents occupying the lowest layer of an automation system require features necessary for enabling the agents to enter into negotiations about the performance of manufacturing tasks and to mutually coordinate the performance of those tasks. The ability to locate, join, leave and participate *in cooperation domains* is tied to the ability of the operating system to coordinate such tasks without interrupting time critical operation as well as providing an efficient message delivery, redirection and prioritization.

At the machine level, agents have temporal restrictions, i.e., hard and soft constraints, and as a result, agents must use their time to accomplish real-time activities. This condition restricts the time availability to perform other operations such as message parsing and queuing. For example, real-time agent activities can be divided into three main activities: (1) domain, (2) control, and (3) meta-level control. Domain activities include execution actions that achieve high-level tasks. The combined effect of execution actions produces a domain-level action (a process step). Control activities are classified as high-level goals and constraints on how to achieve them (e.g., scheduling) or those activities that facilitate cooperation with other agents. Meta-level control relates to learning patterns. Meta-level control tends to use patterns to optimize the agent search by reducing converge time.

Other definitions of agents consider them as purely responsive entities. In current research (Discenzo, 2002), (Maturana, 2002), (Shen, 2001), proactive capabilities have also been part of agents to learn and predict behaviors of the physical device. Real-time distributed control conditions affecting the agents is a new area which has only recently been investigated (Balasubramanian, 2000).

3. REQUIREMENTS FOR AN AGENT-OS

In recent years, we have been researching how to architect agent infrastructures for control. In this investigation, we have learned important characteristics of agent systems that are tied to the information processing requirements. Agents that are intended to process large amounts of information are better off operating in intermediate and upper levels of the enterprise, where larger blocks of memory are available and no time-critical operations need to be used. On the other hand, agents that are intended to interact with the physical device and whose responsibilities are to steer control and handle events (e.g., pump, breaker, conveyor, etc.) are better off operating in the controllers. There has been important work on agent systems for the upper levels of control (e.g., scheduling) (Leitão, 2002), (McFarlane, 2000). Little work has been done to apply these techniques to the control level. To integrate these areas, a simple and straightforward interface between the agents and the control functionality is needed. As a result, a multi-tier technique has been explored.

Agents have been integrated with controllers that are based on IEC 61131-3 programming language (IEC, 2001). Control systems based on the IEC-611499 specification are also very compatible with agents (Lewis, 2001). The role of any interface is to provide accessibility. The interface between the agent infrastructure and the controller OS allows agents to monitor the status of the controller and to perform reasoning about the process. This is the case in the original ACS (Maturana and Tichy, 2002). Figure 1 shows the agent functionality within the controller which was established as a layer on top of the controller's OS. The ACS layer interacts with the controller's firmware to access control functionality such as data table,

programs, and communication. In this controller architecture, agent processes execute within the OS at the lowest priority.

To establish a path for a formal architecture, we will extract the lessons learned from ACS toward the formalization. ACS permits the downloading of agent classes into the controllers. The agent classes are instantiated to create specific agents. Each agent has a control program associated with it. The agents change the control behavior through an events and the data table of the controller. ACS is a low priority task initialized during the device's power up cycle. The agents are threads within the ACS task. These threads are assigned local priorities below the ACS task to avoid interferences with the device's priority. The agent threads are coordinated by a local scheduler. The agent processes are slow compared to the device's tasks since the ACS task executes when the controller is not executing higher priority activities.

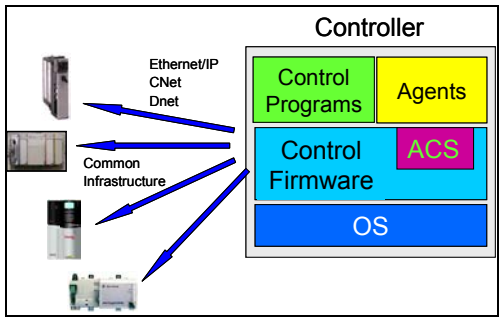

Figure 1: ACS Automation control device

The ACS infrastructure bears larger communication latency since its messages move through intermediate queues into the controller's queues during delivery and reception. The ACS infrastructure depends on ad-hoc anchors into the controller firmware, i.e., the code that connects ACS to the standard controller firmware has never been formalized. When the firmware is changed, the ACS infrastructure is forced to change to follow those changes. More often than not, updating the ACS system is not easy due to changes in the event handling and interrupt configuration.

ACS has obstacles in expandability and upgradeability. To considering it as a commercial product, we need to combine it with the controller's firmware using a leaner approach to avoid the need for continual reengineering.

4. AUGMENTING CONTROL FIRMWARE WITH AGENTS

There is a need for a specification relative to enabling agents natively in the controllers. The extensions to the controller to support ACS were a good starting point and it helped to develop the infrastructure to its full extent. Now, we need to learn how to incorporate the agent functionality into the controller's core.

The architecture of the agent-based control device is simplified by blending the agent functionality into the controller firmware. This change means a significant transformation and enhancements in relation to the style of information processing for the agent decision making process, handling of messages and events, agent composition and creation, and software maintenance and upgradeability.

In the new system, there is no subordinate OS to handle the agent functionality. Instead, the agent functionality follows controller firmware rules. There is no need for special anchors in the firmware to enable agents since each agent process is created as a task using normal priorities. The agent task shares its CPU time with other tasks and it is not segregated to the tail of the list. The controller firmware now schedules all tasks in the controller. The control firmware gains several abilities:

- The ability to *reason* about manufacturing tasks and their relationships to distributed control applications, and to acquire and share *knowledge* related to such reasoning: This ability relates to a planning engine capability. The planning engine needs message parsing and the ability to create concurrent contexts to coordinate the agents. Advanced hardware can bring more efficient capabilities (e.g., hyperthreading in *Itanium* or *XScale* processors).
- The ability to issue appropriate *management commands* to dynamically modify existing applications to perform new tasks or to recover from abnormal operations: This refers to the reactive and proactive behaviors that emerge from the control and/or agent level. The OS must permit the inclusion of auto-generative event-based communication. The ability to transmit events must be allowed between control and the agents.
- The ability to handle both intra- and inter-agent communication of events: Agents communicate with each other via an inter-task or inter-threading message delivery. Across devices, agents need agent location services provided by the system to find addresses and communication routes. Messages are encoded according to FIPA message representation specifications. Also, directory facilitator and agent management system agents must understand and use Semantic Language for agent registration, deregistration, and searching.

In the new architecture, there is no need to have preallocated memory to create the agents. This aspect was limiting in the previous architecture since the controller's image was preconfigured with a block of memory for ACS, with no possibility for adjustment during runtime. The memory block was partitioned to support executive actions, agent threads, and messaging queues. It was a limitation, because in embedded systems, memory is a precious resource. Without knowing the actual size to be used during operation, fixed allocations are inefficient. In the new architecture, a stack is allocated for each task from general memory; thus, memory is consumed only upon need as agents are created.

The subordinate threading model was replaced with tasks (a task is a thread but in the controller firmware context). Each agent is a task which can be created at different priorities level depending on the type of operation, e.g., to act swiftly on complex operations. Support functions such as message parsing and FIPA encapsulation of messages remain the same, but these are also created as tasks. Thus, the communication interface between the agents remains the same. Agents interact with control via events and the data table.

The agents are created in an Agent Development Environment (ADE) as binary objects (i.e., the agent class). The binary object is downloaded to the controllers using a backend loader, as shown in Figure 2. Each object has two entry points, one for the creation of the object instance and other for the initialization of the object's task. The task takes the instance as a parameter. After the task initialization, the object becomes an agent and immediately starts its activity. Multiple objects can be

downloaded into one or many controllers. Multiple agents can be created from a single object type.

With the features above, there is no need to create a special firmware to run agents. It is only necessary to proceed with the downloading of the agent infrastructure (which has been decoupled from the controller's firmware) prior to downloading agents. Thus enabling a controller to run as an agent-base controller is a two step operation: (i) agent infrastructure downloading and (ii) agents downloading.

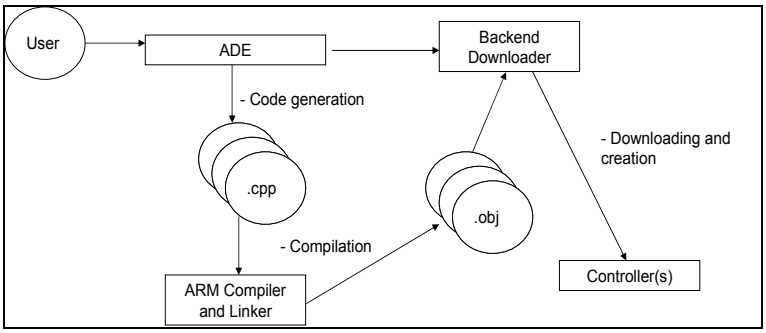

Figure 2: Agent Creation and Downloading

The agent infrastructure is stored on a computer as a library which is delivered to the target controllers by the ADE. The agent infrastructure is common to all controllers. The agents are application specific and these need to be compiled before downloading. There is a prototype system built and tested on two commercial controllers from Rockwell Automation (CompactLogix and ControlLogix L63/A).

5. COMMUNICATION ATTRIBUTES

The Agent-OS requires a unifying communication language, syntax and semantics, and communication transport stack. The message encapsulation is accomplished according to the FIPA specification. The transport encapsulation is based on the Common Industrial Protocol (CIP, 2001).

Figure 3 shows the evolution of a message. An agent emits a message to express some knowledge, desire, or intention. The FIPA layer transforms the message into a stream of data (a compact binary object or a string with XML or Lisp style of encoding). The CIP encapsulation fragments this stream into packets for transmission and attaches transport headers to it. The process of discovering a suitable destination to which to send the message is a linear association that matches the message request with agent capabilities (capability-to-agent). The receiving controller(s) applies packet redirection by looking into the CIP header destination field. The packets are reassembled and delivered to the FIPA layer for decompression and parsing, establishing the communication stack.

Agents can receive and send messages from/to agents or objects on either the same controller or another controller. Messages are delivered directly to the controller's communication queue via an interface for sending messages (*SendMsg()* and *Send()*, as shown in Figure 4). If the destination (Automation Controller ID,

ACID) is local, the message is posted with no fragmentation as a non-blocking action. If the destination is remote, the message goes through CIP encapsulation and it is staged into the communication task for remote delivery.

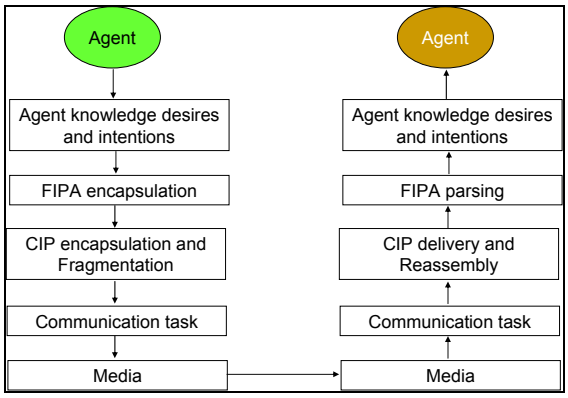

Figure 3: Message Transition

The CIP encapsulation learns the pattern of communication by establishing the frequency of communication between the originator and target controller. With this dynamic knowledge, the CIP encapsulation stimulates the communication task to create a dedicated channel for frequent communications. The less frequent communications retain their unconnected status. This is another important enhancement added to the controller firmware that helps to optimize bandwidth.

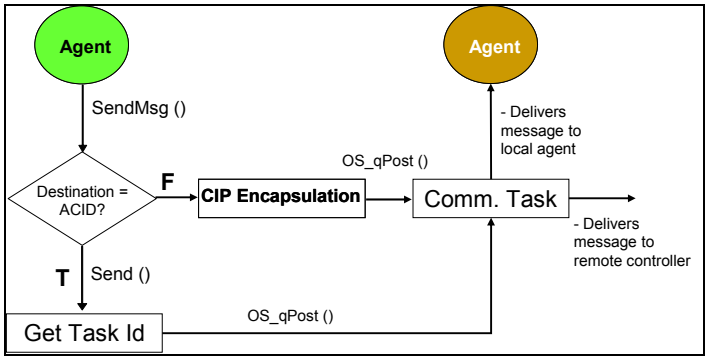

Figure 4: Agent Messaging

Each agent-based task blocks its activity until a new message is posted to its queue. The controller firmware provides a *OS_qPend()* function that is used by the agents to block for an infinite time until a message arrives. The agents access this OS function via an interface *ReceiveMsg()*. This implementation has proven to be efficient in absorbing messages as well as releasing the controller from cycling inside useless messaging loops. Also, an agent sometimes needs to send a message to itself but with delayed delivery. This functionality combines queues and delays.

The agents are retained until explicit deletion occurs by the user. Agents and control programs survive power cycles. The control firmware now has the capability

to detect the departure of a controller using heartbeats. The controllers that depend on the missing capability carry out organizational reconfiguration to bypass the missing agents. This action triggers a communication spike until the controllers learn the new organizational knowledge.

6. REMARKS

Intelligent agent control is a good alternative to monolithic control and information because it is distributed. In this paper, we discussed the evolution of the ACS architecture into a formal approach for the creation of agent-based controllers. We identified the characteristics of the new system and how these will benefit and advance the state-of-the-art of controllers and distributed control. By absorbing the agent functionality into the controller's firmware, we open a path into a new realm in agent control systems.

7. REFERENCES

1. Balasubramanian, Sivaram, Brennan, Robert, Norrie, Douglas H. Requirements for Holonic Manufacturing Systems Control. DEXA Workshop 2000: 214-218.
2. Brennan R, Hall K, Marik V, Maturana F, Norrie D, "A real-time interface for holonic control devices", In: Marik V, McFarlane D, Valckenaers P: *Holonic and Multi-agent Systems for Manufacturing, Lecture Notes in Artificial Intelligence*, No.2744, Springer-Verlag, Berlin, 2003:25-34.
3. CIP: Common Industrial Protocol, 2001, http://www.ab.com/networks/cip_pop.html.
4. Discenzo F, Maturana F, Chang D. Managed Complexity in an Agent-Based Vent Fan Control system Based on Dynamic Reconfiguration, ICCS, Nashua, NH, June 9-14, 2002.
5. FIPA: The Foundation for Intelligent Physical Agents, 1996: http://www.fipa.org.
6. IEC (International Electrotechnical Commission), TC65/WG6, 61131-3, 2nd Ed., Programmable Controllers - Programming Languages, April 16, 2001.
7. Leitão P, Restivo F. A Holonic Control Approach for Distributed Manufacturing, In: Knowledge and Technology Integration in Production and Services: Balancing Knowledge and Technology in Product and Service Life Cycle, V. Marik, L.M. Camarinha-Matos and Hamideh Afsarmanesh (eds), Kluwer Academic Publishers, 2002: 263-270.
8. McFarlane DC, Bussman S. "Developments in Holonic Production Planning and Control", *International Journal of Production Planning and Control*, Vol. 11, No. 6, pp. 522-536, 2000.
9. Lewis RW. *Modelling Control Systems Using IEC 61499*, Institute of Electrical Engineers. 2001.
10. Marik V, Pechoucek M. Holons and agents: recent developments and mutual impacts, *12th International Workshop on Database and Expert Systems Applications*, IEEE Computer Society, 2001: 605-607.
11. Maturana F, Tichý P, Šlechta P, Staron R. "Using Dynamically Created Decision-Making Organizations to Plan, Commit, and Execute Control Tasks in a Chilled Water System". In Proceedings of the 13th International Workshop on Database and Expert Systems Applications DEXA 2002, HoloMAS 2002, Aix-en-Provence, France, 2002: 613-622.
12. Shen W, Norrie D, and Barthès JP. "Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing". Taylor & Francis, London, 2001.
13. Staron R, Maturana P, Tichý P, Šlechta P, "Use of an Agent Type Library for the Design and Implementation of Highly Flexible Control Systems". 8th World Multiconference on Systemics, Cybernetics, and Informatics, SCI2004, Orlando, FL., July 18-21, 2004.
14. Tichý P, Šlechta P, Maturana F, Balasubramanian S. "Industrial MAS for Planning and Control". In (Mařík V., Štěpánková O., Krautwurmová H., Luck M., eds.) Proceedings of Multi-Agent Systems and Applications II: 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001, LNAI 2322, Springer-Verlag, Berlin, 2002: 280-295.
15. Vrba P, Marik V. From Holonic Control to Virtual Enterprises: The Multi-Agent Approach. The Industrial Information Technology Handbook, 2005.
16. Wooldridge M, Jennings N. "Intelligent agents: theory and practice", Knowledge Engineering Review, Vol. 10, No. 2, pp. 115-152, 1995.