Chilled Water System Automation

Rockwell Automation
A Case Study

1. Introduction

This document presents a case study based on the development of a control application using agent technology developed in Rockwell Automation, the Autonomous Cooperative System (ACS) agent platform [1]. The application presented involves intelligent distributed control of shipboard Chilled Water System (CWS) for vessels of the US-Navy [4].

The document is structured as follows. Section 2 provides a description of the CWS application with focus on requirements and utilization of agent technology. Section 3 gives an overview of ACS agent platform that has been designed and implemented in parallel with the development of CWS application. The last section describes lessons that we have learned during this development process.

2. Application Description

The Chilled Water System (CWS) pilot system is based on the Reduced Scale Advanced Demonstrator (RSAD) model, a reconfigurable fluid system test platform. The RSAD has an integrated control architecture that includes Rockwell Automation technology for control and visualization. The RSAD model is currently configured as a chilled water system.

The physical layout of the RSAD chilled water system is a scaled-down version from a real ship. There is one chiller per zone, i.e., currently two plants. There are 16 heat sources on board of the ship (for example, combat systems, communication systems, and radar and sonar equipment) that must be kept cool enough, i.e., below the equipment's shutdown temperature, in order to operate. Each water cooling plant is represented by an agent, as well as each heat

source, each valve, and some parts of the piping system. Within the RSAD, immersion heaters provide the energy to increase the temperature of the heat sources. A temperature sensor at each source provides temperature input to the control system. The main circulation piping is looped to provide alternative paths from any chilled water plant to any heat source.

2.1 System Requirements

To design a highly distributed shipboard automation system, the following four fundamental requirements are considered:

- Reduced manning: This is intended to create a system with less human intervention and more intelligent components capable of making decisions on behalf of the equipment.

- Flexible distributed control: This is understood as the capability of the system to adapt its components operations to respond to dynamically changing conditions without using predefined recipes. To achieve flexible distributed control, extensions to the control and network software is required to enable creation of component-level intelligence that increases robustness of the automation system.

- Commercial-off-the-shelf (COTS): This aspect addresses the cost reduction and system life cycle requirements of new shipboard systems. Under the COTS scope, a ship can be maintained at any friendly location in the world.

- Reliable and survivable operation: As the system becomes more autonomous and self-determined, it is required to augment the level of diagnosability of the components in a distributed manner. The system must be decentralized to avoid a single point of failure. Decision-making process should be placed as close as possible to the component level so that a disconnected system should be able to provide local decisions in the case of failure.

2.2 Agent Technology Deployment

For this type of application, classical or distributed control methodology is applied using programmable logical controllers (PLCs). Nevertheless, in this case there were requirements for survivability and flexibility and these are more or less features of multi-agent systems. Although, it is possible to use backup of PLCs, the control application has to be designed so that all possible combination of failures that might occur are avoided. The control application can be physically distributed over several controllers, but presence of any centralized point in this structure becomes a single point of failure that has to be avoided.

Agents are distributed according to the physical location of the hardware equipment. The hardware partitioning follows the wiring configuration of the input/output (I/O) signals in such a way that the components are independent of one another. Single-point-of-failure nodes, from the hardware distribution perspective, are avoided. Other rules need to be satisfied to be free from single-point-of-failure nodes (for example, use no remote I/O, deploy the smallest possible number of agents per component and controller, and use no mapped inter-controller data). With these rules in mind, the 68 agents for this application are deployed within 23 industrial controllers.

3 Autonomous Cooperative System Description

There was a requirement to use COTS PLCs. Since none of the existing multi agent system platforms were applicable for PLCs at that time, design and implementation of a custom multi agent system platform was needed – this became the Autonomous Cooperative System (ACS) [1]. Standard Rockwell Automation ControlLogix and Flexlogix controllers were augmented with multi-agent system capabilities. Each controller is able to host a collection of various agents, where agents of the same type are downloaded only once and instances differ only by their name and configuration.

3.1 Main Features of Agent Platform

There are two implementations of ACS available: C++ and Java. The first one is able to run in

programmable logical controllers and also in their software emulations running on PCs. The java version is able to run in any Java SE 1.5.

A structure of middle-agents (DF and AMS agents in this case) called dynamic hierarchical teams (DHT) were designed and implemented [3]. This structure has a user-defined level of fault tolerance and is moreover fixed scalable, i.e., the structure can be extended by fixed known cost. This is very important aspect for the fault tolerance of the whole system since these middle-agents are used to find suitable cooperation partners among agents.

There are two levels of agent programming and execution within one agent: high-level and low-level.

- High-level – written in a high-level programming language (such as C++ or Java). This is the core of the agent, the part that is able to communicate with other agents, form plans, undertake reasoning, etc.

- Low-level – written in relay ladder logic, a standard control language. This part is responsible for real-time control to guarantee response times and safety of the control system. This part is also responsible for generating events to the high-level part of the agent as notifications of some state or condition requiring the agent's attention.

The presence of these two levels is essential for fault tolerance. The low-level ensures safe functionality of a control system even when the high-level part is not working properly or when there is a need for fast reaction to some critical situation, perhaps before the high-level part is able to make a more accurate decision.

So far, a declarative style of programming agent behavior has been used and successful implementation of several agent-based control systems undertaken. Nevertheless, this approach has been reevaluated and it became clear that the declarative style has low flexibility since any additional feature has to be included into all parts of the system. Thus, a procedural style of programming has been designed and a procedural engine has been created. This engine enables

use of the full power of a programming language (Java or C++ in this case) together with a set of functions and attributes to interact with other agents, trigger planning processes, and so on.

3.2 Agent Development Environment

The Development Environment (DE) is a software tool that assists the user in programming and deployment of the distributed application [2], e.g., creation and deployment of agents, I/O connections, and automatic code generation. The development environment introduces the following dimensions into the development phase:

- allows the user to specify the physical and behavioral aspects of the application in a manner completely independent of the control system;
- enables the user to specify a multi-processor control system in a manner completely independent of the application that is to run on it;
- assists the user in combining an application with a control system;
- generates the control code and behavior descriptions for each agent in the system;
- combines the code for all agents assigned to each processor in the system;
- augments each controller automatically to handle the communications to other controllers as a result of the program distribution; and
- communicates with all the controllers involved in an application, for their programming and configuration, and for subsequent monitoring and editing.

DE is based on a library of components called the template library (TL). The library is editable by the user, and each template can contain both low-level control behavior (written in ladder diagram) and higher-level intelligent behavior. The model for the control behavior supports an "object" view of the components in that, for example, inheritance is supported. The TL author can express, for example, that "A Radar is a type of CombatSystem". Each instance of Radar inherits all ladder data definitions and the logic from CombatSystem template. Each library is targeted to a specific application domain, e.g., material handling systems or shipboard chilled water systems, and so can be used to create control systems for multiple similar facilities (F) in the same domain. In this way, the effort to build the library is amortized over all the facilities

built with the library, and each facility reaps the benefits of having control system software that is structured, well defined, predictable, and tested.

The user creates a facility from components of the template library and customizes their parameters. Next, the user establishes a Control System (CS) that describes all the controllers, I/O cards, and networks, plus their interconnections. After the TL, F, and CS parts are completed, the user generates and compiles the code. We currently support relay ladder logic (IEC 1131-3) for low-level agent part and C++ code for high-level agent part, plus we are prepared to support also Java code. After the agent assignment, the user downloads the software into the controllers.

Since TL, F, and CS editors are independent, it is possible to change only the necessary parts when the system needs to incorporate changes. For example, a new controller can be added by the CS editor and subsequently have some agents assigned to it. The system is regenerated in a manner consistent with all modifications.

4 Discussion

There were several key lessons from this project. The first is that for actual deployment of such an agent system it is very important to simulate its behavior prior to running it in a real environment. Both a software simulation environment (using Matlab) and a small-scale hardware simulation environment were created for testing purposes. Simulation is especially important for a multi-agent system since an emergent behavior may appear in this case and its identification and study without simulation can be very hard.

The second lesson concerns the issue of standards. During the development of the multi agent system platform close attention was paid the FIPA agent system standards. The FIPA standards were given low priority in the beginning since they bring with them a huge overhead of resources. Subsequently, as development proceeded, the system has been made increasingly FIPA-compliant. Standardization is very important when two or more systems (multi agent

system platforms in this case) need to be interconnected and this was the case for other applications.

Nevertheless, this standards-compliance effort is not for free. The first issue is the overhead that it brings. For example, the FIPA Agent Communication Language (ACL) has to be adopted on top of messages. These are then embedded as a content of this ACL message structure. Agents have to be able to use the FIPA ACL Semantic Language (SL) for mutual understanding of communication, which represents a further overhead. The overhead is not only in computation time and memory consumption, but also in the effort of the people developing the system. Moreover, SL is very impractical since its notation is not suitable for extensive parsing and working with the message in a form of the collections of objects. In summary, all these factors have to be carefully evaluated before adopting the FIPA ACL standards. One of the possible compromises is to use standards for external communication only and avoid them for internal purposes.

A third lesson was that, as the system becomes more complex, there is a need for a visualization and debugging tool to observe the internal communication among the agents and behavior of the system and to discover potential problems. This need led to the development of a tool that receives messages from all agents in the system, reasons about the information, and presents it from different points of view and from different levels of perspective. All these parts of the visualization screen are interconnected. For example, it is possible to identify some problem in the workflow window, select the appropriate part of the conversation among the agents and receive the list of messages involved in the conversation. Through the tool, the user can also configure running agents by sending service messages to them.

On the issue of simulation, the Manufacturing Agent Simulation Tool (MAST) [5] was developed separately from ACS system. MAST represents a new generation of simulation systems with embedded multi-agent systems principles aimed at the manufacturing domain. It runs on top of JADE system and it has been intended as an agent-based demonstration application that would illustrate, for some typical manufacturing task, the major benefits of the deployment of agent technology. To show the robustness and flexibility of the agent solution, attention was

paid to failure detection and recovery. A failure of any component can be emulated (e.g., a failure of the conveyor belt) causing the agents to start negotiations on alternative transportation paths while avoiding the broken component. An easy-to-understand visualization helps the user to observe the overall behavior of the system during the simulation.

Since the project began, an enormous amount of time was spent on the design of the generic planning engine. This planning engine was originally based on a declarative programming language, which imposed many limitations. Any addition to this language resulted in changes required for all related tools. Thus a lesson learnt was that moving to a procedural language gave the benefit of overcoming these limitations and stabilizing development of the related tools. During the deployment of the multi-agent system, a key lesson was that a fault tolerant multi-agent system must satisfy the following characteristics:

- Reliable communication. The system has to guarantee that no messages are lost or duplicated. Every message must be delivered, or else the sender must be notified if a message cannot be delivered.

- Fault tolerant agent platform. Since the agent platform is an environment where agents live, the possible failure of some agent should affect neither this environment nor other agents in this system.

- Fault tolerant knowledge. Knowledge about other agents is a fundamental part of a multi-agent system. Thus this knowledge has to be present at configuration time or dynamically obtained in a fault tolerant manner. For example, having only one directory facilitator that manages this knowledge creates a single point of failure in the system.

- Physical distribution. It should be possible to physically distribute the agents of a multi-agent system and allow communication among them. This distribution increases fault tolerance in the case of a hardware failures, power failures, etc.

Since agent technology is more powerful than classical control, it is important not to limit its power by blindly fulfilling the initial requirements of customers, especially procedures that they are using with classical control. For example, there are standard procedures for performing specific tasks that a person has to follow, usually in a form of recipe. Nevertheless, it could be possible to use more advanced techniques in a multi-agent system that guarantee the same result with increased performance.

What would be the main guidance while considering implementation of an agent system? In the beginning, ensure that agent technology provides some advantage over classical control in considered implementation. The benefit can be not only of improved functionality, fault tolerance, and scalability. The benefit can also be the ability to reuse an already-developed agent solution to solve a similar problem. The first implementation is likely to be harder than before, but subsequent implementations may just be a matter of different parameter configuration.

References

[1] Maturana, F., Šlechta, P., Vrba, P., Tichý, P., Staron, R., Carnahan, D., Discenzo, F., Mařík, V., and Hall, K.: A Specification to Build Distributed Reconfigurable Manufacturing Systems Using Intelligent Agents. The 3rd International Conference on Reconfigurable Manufacturing (CIRP 2005), session C3, C-14, Ann Arbor, MI, 2005.

[2] Staron, R. J., Maturana, F. P., Tichý, P., and Šlechta, P.: Use of an Agent Type Library for the Design and Implementation of Highly Flexible Control Systems. Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004), Orlando, USA, pp. 18-21, 2004.

[3] Tichý, P. Social Knowledge in Multi-Agent Systems. Ph.D. Thesis, Czech Technical University in Prague, Czech Republic, 2003.

[4] Tichý, P., Šlechta, P., Maturana, F. P., Staron, R. J., Hall, K., Mařík, V., and Discenzo, F. M.: Multi-Agent Technology for Robust Control of Shipboard Chilled Water System. In Proceedings of International Federation of Automatic Control (IFAC)

Conference on Control Applications in Marine Systems (CAMS 2004), session TA2, Ancona, Italy, 2004.

[5] Vrba, P., and Mařík, V.: Simulation in Agent-based Manufacturing Control Systems. In Proceedings of the IEEE International Conference on Systems, Men and Cybernetics, Hawaii, USA, pp. 1718-1723, 2005.

Pavel Tichý, Vladimír Mařík and Pavel Vrba
Rockwell Automation Research Center
Pekařská 695/10a
Prague
Czech Republic
{vmarik,pvrba}@ra.rockwell.com

www.rockwell.com

Michal Pěchouček
Gerstner Laboratory
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics
Technicka 2
166 27 Prague 6
Czech Republic
pechouc@labe.felk.cvut.cz