11 INTEGRATING MULTI-AGENT SYSTEMS: A CASE STUDY

Francisco Maturana[1] Raymond Staron[1] Fred Discenzo[1] Kenwood Hall[1]
Pavel Tichý[2] Petr Šlechta[2] Vladimír Mařík[2]
David Scheidt[3] Michael Pekala[3] John Bracy[3]
[1]*Rockwell Automation*
1 Allen-Bradley Drive, Mayfield Heights, OH 44124-6118, USA,
{fpmaturana, rjstaron, fmdiscenzo, khhall}@ra.rockwell.com
[2]*Rockwell Automation Research Center*
Americka 22, 120 00 Prague, CZECH REPUBLIC
{ptichy, pslechta, vmarik }@ra.rockwell.com
[3]*The Johns Hopkins University Applied Physics Laboratory*
11100 Johns Hopkins Road, Laurel Maryland 20723-6099
{David.Scheidt, Mike.Pekala, John.Bracy}@jhuapl.edu

Intelligent Agent technology provides an appropriate framework to integrate knowledge with efficient production actions in distributed organizations. Integration of knowledge depends on balanced information representation within and across heterogeneous organizations. Integrating information within a specific environment can be helped by the deployment of standards and common practices. However, it is harder to attempt such a smooth integration with the information of foreign organizations. It is the challenge of this paper to present an architecture that provides a first step towards successful integration of separate multi-agent systems in a real life control domain.

1. INTRODUCTION

Intelligent autonomous control provides the ability to address large complex systems. However, reliance upon a single intelligent controller presents a survivability problem, as damage to that centralized controller or to the communications infrastructure used to interact with actuators and sensors, can result in a loss in controllability. This is especially applicable when the system in question operates within a hazardous environment. In this paper, we describe two intelligent control systems that have been applied to provide intelligent control for auxiliary systems on capital ships. These platforms are exposed to potential threats throughout the course of normal operations, and survivability is therefore a key concern.

Multi-Agent Systems (MAS), where distributed autonomous intelligent agents collaborate to carry out control activities, provide a compelling means for achieving robust, survivable control. Recognizing this fact, Rockwell Automation (RA) and

the Johns Hopkins University Applied Physics Laboratory (JHU/APL) have independently developed multi-agent architectures, as well as corresponding prototype implementations, to address the control of these auxiliary ship systems. These prototypes target a closed loop fluid distribution system used to regulate the temperature of devices aboard capital ships.

This paper provides some general background on agent-based control as it applies to this particular domain, discusses the architecture of these two MAS, provides some details related to their testing and validation, and concludes by describing a recent new effort to integrate these two agent frameworks in order to form a single collaborative control system.

2. AGENT-BASED CONTROL

In the ship domain, distributing control across multiple agents can be used to improve survivability and to reduce the complexity of the domain for individual controllers. Ideally, device controllers will be co-located with their subservient devices to decrease the likelihood that an intact, serviceable device will be unable to function due to loss of control. In order to satisfy ship-wide goals, distributed controllers must cooperate through either supervisory or peer-to-peer relationships. In this scenario, inter-agent collaboration can be achieved easily within the homogeneous MAS, but past experience indicates that multiple heterogeneous agent systems will need to coexist and collaborate. In our context, heterogeneous agent systems are more than just two different agents; it means different agent systems, syntax and semantics, programming language, and computing platforms and operating systems. Therefore, interoperability is an important dimension of MAS. One intention of this paper is to show how two independent infrastructures can be integrated to produce this type of heterogeneous collaborative control.

Each intelligent agent represents a physical process, machine or device and coordinates its operations with other agents, using standards such as FIPA (FIPA 2003). The MAS architecture is organized according to the following characteristics:

· Autonomy: Each intelligent agent makes its own decisions and is responsible for carrying out its decisions toward successful completion.

· Cooperation: Intelligent agents combine their capabilities into collaboration groups (clusters) to adapt and respond to diverse events and goals.

· Communication: Intelligent agents share a common language to express their beliefs, desires, and intentions.

· Fault tolerance: Intelligent agents possess the capability to detect and isolate equipment failures.

· Interoperability: Intelligent agents use a standard Agent Communication Language (ACL) to extend their collaboration into remote and heterogeneous agent systems.

3. INTELLIGENT AGENT ARCHITECTURES

The following sections describe the agent architectures that were developed by RA and JHU/APL. While these architectures were developed independently, they

nevertheless exhibit many similar attributes, which are common to the domain, as well as intelligent agents in general. To date, implementations of both frameworks have been individually demonstrated on a hardware test bed. This includes demonstrations of standard ship operations under nominal conditions, as well as in the presence of faults. More specific details on the tests that were conducted can be found in (Alger et al. 2002) (Tichý et al. 2002) (Maturana et al. 2002).

3.1 RA Agent Architecture

Industrial automation environments are generally populated by multiple interconnected control devices. These devices include controllers and networks, all working in a synchronized manner to handle production actions and events. Although this technology has proven effective in the last 30 years, the current requirements imposed by larger and increasingly more diverse information volumes have redirected the effort towards more flexible systems. Next generation automation devices will be agent-based. These agents will need to be able to efficiently and effectively coordinate the activities of the controlled hardware. The following sections detail the agents developed by RA to satisfy these criteria.

3.1.1 Automation Architecture

In agent-based control, the controllers have an agent infrastructure for enabling component-level intelligence. With this, it is possible to distribute the intelligence among multiple controllers using different agent sizes and populations. Different network connectivity can be used to exploit the distributed intelligence dimension that is added by the intelligent agents.

3.1.2 Inter-Agent Architecture

The controllers have an agent infrastructure for enabling the component-level intelligence. With this, it is possible to distribute the intelligence among multiple controllers using different agent sizes and populations. The relationship among the agents is loosely coupled. The controller's firmware was modified with two additional infrastructures: (1) Distributed Control Agent (DCA) and (2) Intelligent Agent (IA).

The DCA infrastructure is a set of software interfaces added to the controller's firmware to enable and maintain agent tasks. It uses the base firmware to glue the components via a multithreaded system. Also, it uses the controller's interfaces to communicate with field devices (i.e., sensors and actuators). The IA extension co-exists with user level programs (i.e., ladder logic IEC 1131-3 (IEC 2001)). Here, the user downloads the application specific components, which are the rules and behaviors of the intelligent agents.

For inter- and intra-organization conversations, the intelligent agents emit messages outside their organization by wrapping Job Description Language (JDL) messages inside FIPA envelopes. Communication with remote agent systems (inter-organization) depends on having the listening nodes understanding the language and transport protocol of the sender nodes. FIPA compliance is one requisite but it is insufficient if lower layers of the communication stack do not understand each other.

Hence, our work is important because it also identifies the requirements to build an interoperable agent communication stack.

In JDL, information is encoded as a sequence of hierarchical actions with precedence constraints. These are requests and information messages. When an intelligent agent accepts a request, an instance of a plan template is created to record values that emerge during the planning process. Requests are propagated throughout the organization using the Contract Net protocol (Smith, 1980). The requests visit multiple agents and multiple negotiation clusters are formed.

The FIPA standard uses a Matchmaking mechanism to connect agents. In this implementation, we are FIPA compliant and the architecture includes the functionality of Directory Facilitators (DFs). A DF performs capability registration and matchmaking. For each capability request, a DF provides a list of agents that are able to provide the requested capability. The DF functionality is part of the DCA extension.

3.1.3 Intra-Agent Architecture

The intelligent agents are goal-oriented entities. They organize the system capabilities around system missions. Each mission is intended to satisfy a given set of goals, which are commonly expressed as a single cost or performance metric value.

Group goals emerge dynamically and are agreed upon by the intelligent agents through negotiation. For instance, an intelligent agent that detects leakage in a pipe of the cooling system establishes a goal to isolate the leakage. The intelligent agent then informs adjacent intelligent agents to evaluate the problem according to their local views and knowledge. This is the origin of a group-based goal, which is to isolate the leaking pipes. An intelligent agent has four components:

- Planner: This component is the brain of the intelligent agent. It reasons about plans and events emerging from the physical domain.
- Equipment Model: This component is a decision-making support system. Models of the physical domain are placed here to help the Planner evaluate different configurations. The Equipment Model provides metrics for proposed configurations.
- Execution Control: This component acts as control proxy, which translates committed plans into execution control actions. These actions are synchronized with the control programs. It also monitors events from the control logic and translates them into response-context events to be processed by the Planner component.
- Diagnostics: This component monitors the health of the physical device. It is programmed with a model of the physical device, against which a set of input parameters (e.g., bearing vibration of a pump) is evaluated.

The physical device (e.g., a valve) is operated according to a sequence of actions. These actions are encapsulated inside device drivers and classified according to the type of OEM. The device driver becomes a template library for the physical device. When an intelligent agent is created for a device, an instance of its device driver is copied in the controller's memory. The sensors and actuators associated with the physical device are connected to the automation controller using an industrial network. State variables are contained in the controller's data table.

The intelligent agents perform resource allocation via priority ranking and negotiation. Each intelligent agent provides a set of capabilities to the overall

organization. These capabilities are combined to carry out distributed planning in three main phases: Creation, Commitment, and Execution. During creation, an intelligent agent initiates a collaborative decision making process (e.g., a load that will soon overheat will request cold water from the cooling service). In this process, multiple agents are involved in deciding the local setups. The intelligent agents offer a solution for a specific part of the request. Then, the intelligent agents commit their resources to achieve the task in the future. Finally, the plan is executed.

3.2 JHU/APL Agent Architecture

The original agent framework for autonomous distributed control that JHU/APL developed is called the Open Autonomy Kernel (OAK), and is described more thoroughly in (Alger et. al, 2002). OAK defines flexible inter- and intra- agent architectures, and is targeted towards "hard" control problems that involve complex, partially observable systems.

Internally within OAK agents, system knowledge is comprised of both directly observable knowledge (e.g., sensor readings and memory of past commands) and inferred or "hidden" knowledge (e.g., the actual state or current operational mode of each component). Control is addressed as a three-step process, consisting of *diagnosis*, *planning* and *execution*. See Figure 1. We refer to the process of deriving the inferred knowledge from the directly observable knowledge as diagnosis. Reconfiguration involves identifying a system state that will achieve some desired goal and identifying a sequence of actions that will successfully transition the plant into this state. Issuing and tracking the progress of these action sequences is the role of execution. To perform diagnosis, OAK agents utilize the L2 reasoning engine (Kurien 1999), which is one member of a family of model-based autonomy technologies (Williams, Nayak 1996).

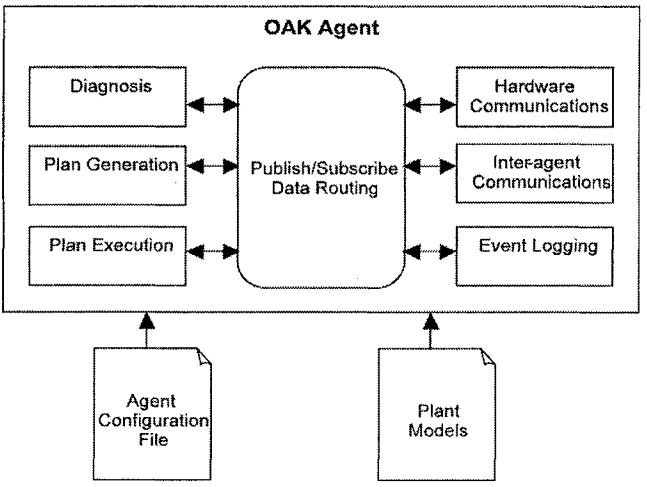

Figure 1– OAK agent architecture

OAK agents classified all external communications into two categories: communications with other OAK agents via the Agent Communication Framework (ACF) and communications with hardware devices. The types of information shared between agents consisted of: facts (assertions about the current state of the world), goals (statements of intent) and information related to agent coordination (e.g., establishing communications paths). The types of information shared between agents and hardware devices consisted of: commands (sent from agents to hardware) and observations (sent from hardware to agents). All inter-agent communications utilized an XML-based Agent Communications Language (ACL), while the individual hardware interfaces each determined the format of information passed between the agent and the hardware.

4. INTEGRATING THE AGENT FRAMEWORKS

The effort to integrate these two agent frameworks stems largely from a desire to leverage the relative strengths of each, while at the same time, demonstrating the effectiveness of common standards. In particular, our goal is to highlight a synthesis of the distributed reconfiguration capabilities of the RA agents and the powerful model-based diagnostic capabilities provided by OAK agents. A fluid control system was selected as the target domain as both organizations have extensive experience in employing agent based control in this capacity.

The first challenge faced by the team was determining the roles and responsibilities for the respective agent systems. One option involved partitioning the fluid control system along functional or spatial boundaries, and assigning a subset of the overall system to each agent network. In this case, however, the highly reconfigurable nature of the system under consideration makes it difficult to identify spatial regions that will tend to remain independent, especially in fault scenarios.

A functional decomposition poses certain difficulties as well, as effective control requires a high level of coordination among the various component types that comprise the fluid system. While both of these approaches remain options for future work, the initial approach selected involves assigning each framework a different portion of the overall control cycle. In this paradigm, RA's agents are responsible for performing reconfiguration, while APL's agents will perform diagnosis.

Adapting these agent frameworks to accommodate situations where a given agent is no longer responsible for the entire control cycle within its domain became one of the challenges of this effort. Previous deployments of these frameworks admitted both hierarchical and peer-to-peer relationship among agents, but assumed that the control cycle itself was largely encapsulated. In this new configuration there is increased motivation to have agents share additional pieces of information. For example, previously APL agents externalized only a succinct summary of their internal state for consumption by peer and parent agents. Now that the corresponding planning capability resides in a different agent, there is increased value in having agents share with the community more detailed diagnostic information, such as multiple hypotheses. In addition to modifying the types of information passed between agents, this task required standardization of the inter-agent communication mechanisms themselves.

4.1 Agent Capabilities

Once agent roles were defined, the next step involved defining the types of information that would be exchanged and the mechanism for exchanging them. The team decided upon a service-based architecture, where both white pages and yellow pages discovery techniques are used. In order to perform yellow pages discovery, a standard set of *agent capabilities* need to be defined. The overall information flow between the two agent systems can be succinctly summarized in terms of these capabilities.

4.1.1 RA Agent Capabilities

The set of services provided by RA agents include:

Register for Data – This capability allows an agent to register for the asynchronous notifications that are produced whenever a command is issued or a new sensor value is observed. Properly diagnosing the current state of the system requires both knowledge of past commands and knowledge of current sensor readings. From the perspective of APL agents, who are consumers of this data, this represents a change from the previous deployment where agents already possessed local knowledge of hardware commands (which were a product of the local planning process).

Control Load – This service provides the mechanism by which external agents can request that a particular load be activated or deactivated. Activating or deactivating a load implicitly invokes the distributed planning engine, which reconfigures the fluid system in order to meet this objective.

Set Priority – This service allows external agents to set the relative importance of loads in the fluid control system. In situations where the full set of desired loads cannot concurrently be activated, this priority is used to determine which loads will be supplied. Thus, external agents that have additional or higher-level information (e.g., knowledge of pending damage events) can use this capability to productively influence the results of the distributed planning algorithm.

4.1.2 APL Agent Capabilities

The set of services provided by JHU agents include:

Get Diagnosis – This capability allows agents to request diagnostic information from the agent. Diagnostic information consists of a set of ranked hypotheses that include each component within the agent's sphere of influence.

Get Diagnostic Model – This capability allows agents to request detailed information about the diagnostic model from the agent.

4.2 Standardization

Successfully implementing these agent capabilities requires a unifying communication language, syntax and semantics, and communication transport stack based on a Common Industrial Protocol (CIP). To this end, we use the FIPA/JDL specification to implement discovery services and to interpret application domain knowledge. Unifying the communication transport stack is a very important activity

towards the standardization of the agent interoperability. In the intended system, we have different agent platforms, programming languages, and radically different agent platform containers, which include Windows XP workstations (C++ domain), Linux workstations (Java domain), and automation controllers (C++ domain), as shown in Figure 2. Our intention is to create a common stack in the form of a library, which will be written in Java and C++ languages to support the workstations and controllers.

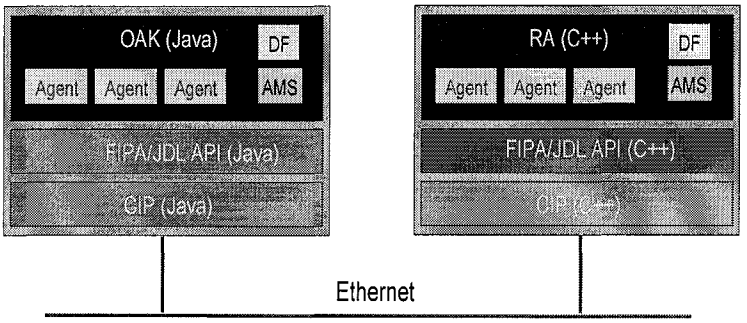

Figure 2– Integrated heterogeneous agent systems

For agent discovery, we use the FIPA specification for Directory Facilitators (DF), which provides matchmaking capabilities and Agent Management Services (AMS). The architecture offers the ability to create multiple global or local DF agents. Redundancy, in conjunction with a heart-beating mechanism, provides fault tolerance. Multiple knowledge propagation and retrieval techniques have been implemented, providing the system designer with the flexibility to trade off efficiency for redundancy.

Communications is defined using Job Description Language (JDL) messages. A common JDL messaging library has been developed and integrated into both MAS. Any non-primitive message content encapsulated within a JDL message is first encoded in XML, using schemas to define the valid syntax. Whenever possible, existing standards were leveraged in order to define this syntax. For example, the syntax adopted for diagnostic model descriptions is the XML Model-based Programming Language (XMPL) (Livingstone2, 2004). Although this specification was developed for the L2 reasoning engine, it has proven sufficiently general to be used with other model-based reasoning engines as well.

4.3 Implementation

In addition to implementing the libraries and specifications described above, this integration effort has also engendered changes to the agent frameworks themselves. Although a detailed description falls outside the scope of this paper, a successor to the OAK agent framework has been developed that strives to provide additional flexibility in defining agents and their capabilities. This new framework greatly simplifies tasks such as developing agents whose capabilities are not prototypical or change at runtime.

4.4 Use Case: Dynamic Modeling

As mentioned previously, fault tolerance is an important aspect of multi-agent systems that are exposed to harsh or dangerous environments. In these situations, the agent network must be robust against the loss of one or more individuals. Redundancy in the form of standbys is one traditional approach often used to address this issue. When the loss of an agent is detected, a new agent is brought online to replace it. As a variation on this concept, JHU/APL has begun exploring agents whose control domains are dynamic and responsive to failures. For example, when a peer agent is lost, neighboring agents can negotiate to determine who will assume responsibility for that agent's resources.

Part of assuming responsibility for an agent's resources involves the ability to perform diagnosis on that portion of the system that was under control of the deceased agent. For model-based agents, this implies obtaining or generating a model for this portion of the system. Thus, the ability to exchange model fragments, as described previously, represents an important prerequisite for many types of dynamic modeling approaches. In order to further the dual objectives of exercising our inter-agent communications standards and developing a framework that can be used to test dynamic modeling algorithms, we have developed a dynamic model management component for APL's next-generation intelligent agents. This component has the ability to maintain multiple model fragments, and to merge these model fragments into the agent's local model at runtime as appropriate.

This feature was exercised in a small scenario involving three child agents and a single parent agent. Each child agent has a local model consisting of a strict subset of the overall fluid control system. At runtime, these models are serialized and sent to the parent agent, using the standards described previously. The parent agent, using the dynamic model management capability, then assembles the overall system model. Subsequently, the parent agent used observations in conjunction with this model to diagnose faults in the system. Although only a small part of the overall solution, this demonstrates some of the utility in providing this type of inter-agent communication.

5. CONCLUSION

Integrating heterogeneous agent systems is not an easy task. Nevertheless, the fundamental pieces necessary for a first integration based on standards such as FIPA, Ethernet/IP, and CIP have been created. Progress made to date indicates that leveraging these standards, in conjunction with a common knowledge representation, will enable a seamless interaction between the two systems.

6. ACKNOWLEDGEMENTS

This work has been conducted as part of the Office of Naval Research "Machinery Systems and Automation to Reduce Manning" program under contracts N00014-00-C-0050 and N00014-02-C-0526.

7. REFERENCES

1. Alger D., McCubbin C., Pekala M., Scheidt D., Vick S. Intelligent Control of Auxiliary Ship Systems. Proc. of Innovative Applications in Artificial Intelligence, AAAI, Edmonton, CA, 2002.
2. Kurien J., Model-based Monitoring, Diagnosis and Control, PhD Thesis Proposal, Brown University Department of Computer Science, 1999.
3. Maturana F., Tichý P., Šlechta P., and Staron R., "Using Dynamically Created Decision-Making Organizations (Holarchies) 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, pp. 613-622, 2002.
4. Tichý P., Šlechta P., Maturana F., and 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, pp. 280-295, 2002.
5. Williams B., and Nayak P. A model-based approach to reactive self-configuring systems. In Proceedings of AAAI-1996.
6. Shen W., Norrie D., and Barthès J.P.: Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing. Taylor & Francis, London, 2001.
7. Smith R.G.: The Contract Net Protocol. High-level Communication and Control in a Distributed Problem Solver. In IEEE Transactions on Computers, C-29(12), pp. 1104-- 1113, 1980.
8. International Electrotechnical Commission (IEC) TC65/WG6, 61131-3, 2nd Ed., Programmable Controllers - Programming Languages, April 16, 2001.
9. The Foundation for Intelligent Physical Agents (FIPA): http://www.fipa.org, 2003.
10. Livingstone2 Documentation, http://ic.arc.nasa/gov/projects/L2/doc, 2004.