An. Şt. Univ. Ovidius Constanţa Vol. 11(1), 2003, 45–54

FLEXIBLE ARCHITECTURE FOR COMMAND INTEGRATION WITH CONTEXT REPRESENTATION

Ozten Chelai

To Professor Silviu Sburlan, at his 60's anniversary

Abstract

The command integration is one of the key issues in today industrial applications. Process control systems are distributed systems with their resources, as data acquisition systems, physically distributed and logically integrated to maintain a stable and trusty climate for process execution and management. Software systems should consist of simple, conceptually clean software components interacting to obtain the needed functionality of the system. This paper proposes an agent system architecture with a special component that represents the context of the system, which offers the necessary support for command integration and intelligent distribution of industrial distributed systems in order to obtain flexibility.

1. Introduction

The command integration is one of the key issues in today industrial applications. Automation systems are real-time industrial systems that integrate information from sensors, boardrooms to data servers and user interfaces. For this, companies must replace incompatible networks and systems with open fieldbuses and distributed control systems.

Flexibility is one of the key features of all modern systems. The control systems have to adapt to the rapidly changing environment and to unexpected situations. A lot of software modules have to be modified when the system

Key Words: Software architectures, distributed systems, control systems, agents systems.

is rebuilt to meet new requirements. To achieve flexibility the concept of autonomous decentralized system can be used.

Why agent systems?

Agent's applications are modular, decentralized, flexible and complex. These are characteristics that suit agents to complex distributed systems. Command applications are integrated in industrial applications that are real-time, distributed complex systems. An industrial application is candidate for agent's technology if it can be decomposed in physical entities with well-defined state variables that are distinct to those of the environment.

In classical systems the decomposition used is "functional". This technique is suited for centralized systems. In distributed systems, the decompositions to be made on distinct entities in the physical world (H.Van Dyke Parunak, 1998, Practical and Industrial Applications of agent-Based Systems, Industrial Technology Institute). The new applications are additions to existing systems, and a suitable way to upgrade and integrate the new techniques is to use agents that encapsulate old systems. This method adds functionality to the systems and offers an easy integration with other agents. Although, agents have different internal structures, they have the same communication rules to understand each other. Agents may communicate through changes in their physical environment, or may exchange messages between them.

An agent based system in control field industrial applications, and others too, has the data and code encapsulated in a software object with its own thread of control and the ability to execute autonomously without external help (H.Van Dyke Parunak, 1998, Practical and Industrial Applications of agent-Based Systems, Industrial Technology Institute). The agents share the state variables of the environment (figure 1).

2. System design

To achieve flexibility is used the concept of autonomous decentralized system (Hodehiko Wada, Yuichi Sakuraba, Masako Negishi, Machinery Control System using Autonomous Agents, Yokogawa Electric Corporation, Japan). There are used components that work autonomously in the system to build an autonomous decentralized type of command system.

Software systems should contain components that are conceptually clean and simple. Components have to communicate to provide system behavior. The problem is that not all the communication is related to the functionality that gives the behavior of the system. The knowledge of the component that is not conceptually required for its behavior is named extraneous embedded

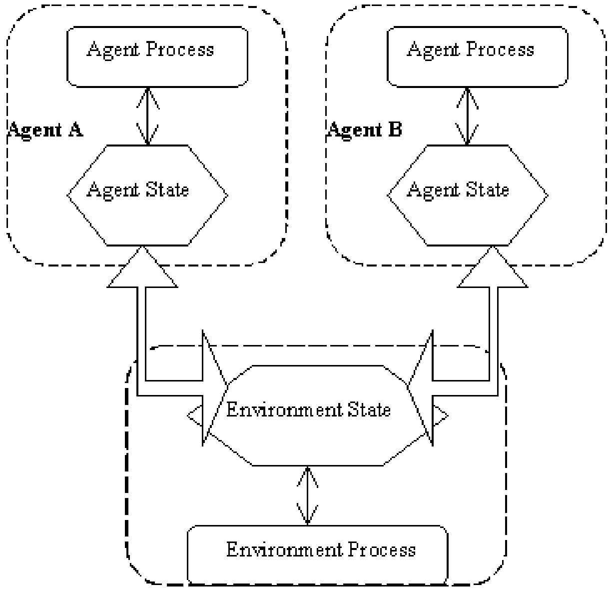

Figure 1. Agents interaction through a shared environment

knowledge (Walker Robert, Murphy Gail, 2000, Implicit Context: Easing Software evolution and reuse, University of British Columbia, ACM). To remove the extraneous embedded knowledge from components, a context component is used. This component shares the state of the system and manages the external and extraneous knowledge so communication is clearer and flexible after actual environment of the system. The context agent represents the context provided by the execution of the system: at any given moment during an execution. The context agent consists of the structure of the system and the history of interactions within the system. To represent the dynamic aspects of the system, the context component must have a temporal part. The environment may be any set of possible states. One state represents "now". The past is a linear sequence of states (there is only one past), and the future is branched (there are many possible futures). The actions are atomic: only one is ever performed at a given time. This model can be implemented easily using existing programming techniques.

Another problem is that the communication support and the rules are different from environment to environment. An open fieldbus architecture is used based on a standard communication protocol and support for standard user layer. There are fieldbuses that make possible to connect control devices with the network and interoperate at all layers of the OSI Model (from physical to application layer). To unify the existing systems and develop new ones, an integrating layer is used that defines how devices will perform data acquisitions and control functions, and how this functions will interoperate across. This architecture must allow real-time operation. The integrating layer can be represented also by the context component. This means that we eliminate a layer, and the overhead of this layer, integrating the context component into the system.

The interaction with the system can be made from:

- An operator that introduces manual commands to the system

- The acquisition system that brings the process data into the system, with local processing integrated into the local system and signaling some abnormal situations

- Other systems (local or external). Their state is shared by a special component, named "Context agent".

According to the operator commands, abnormal situations, environment changing, the actions to be done are executed locally by the local agents. The local agents execute their actions on computers connected to the controllers. The distribution of the actions to be performed is established by a special command component that receives the commands from the operator and knows the state of the system from the shared context component. This special command component has integrated some logic that can plan and distribute the actions made by local components. The command component can store the data, status and process history into a local database system that operates in real-time mode.

3. System architecture

I propose a model for a software system with necessary components for remote commands integration with the existing industrial systems and intelligent command management distribution. The system uses agent technology.

Agent systems must have a strong foundation based on masterful software patterns (Elizabeth A. Kendall, P.V. Murali Krishna, Chirag V. Pathak, C.B. Suresh, Patterns of intelligent and Mobile Agents, Royal Melbourne Institute Of Technology, Australia). The layered pattern is adopted because higher level or more sophisticated behavior depends on lower level capabilities. The leyers are identified the model of the agent's real world. The Layered Agent is a general architecture that addresses simple and sophisticated agents. Agents are composed from one to seven layers. The layer hierarchy is (bottom-up): Sensory, Beliefs, Reasoning, Actions, Collaboration, Translation, and Mobility. Our agent's beliefs are simple and do not change, so the agents don't need sensors, mobility and collaboration.

The architecture of the system use as support the OSI model, and develops, using the application level, an upper layer with needed components for remote command integration.

The system has the command distributed on two layers:

The physical layer with local acquisition and command.

The shared command layer with command logic shared by the physical layer.

The model of the system presents an abstraction of the application at the design level. The model has to define specifically components of the system and the relation between them.

The software agents represents support instruments for model defining of the processes that can be used in application development.

The proposed model use agent technology and has the components represented in figure 2.

The components are:

Local agent - that make the data acquisition, local control and local execution tasks

Context agent - shares the state of the system capturing the changes exposed by the local and command agents.

Command agent - has command logic integrated and shared by the local agents. The list of commands contains:

- External commands - transmitted from the interface agent

- Internal commands - which represent actions, resulted from Belief Desire Intention model implementation after the context changing.

The command agent is capable of independent, autonomous action in order to meet its design objectives. The model implemented for this agent is the belief-desire-intention (BDI) model (Michael Wooldridge, 2000, Reasoning About Rational Agents, The MIT Press, Cambridge, Massachusetts)

Interface agent - is the agent that interacts with the external systems. The interface agent interacts with the external environment as a data server

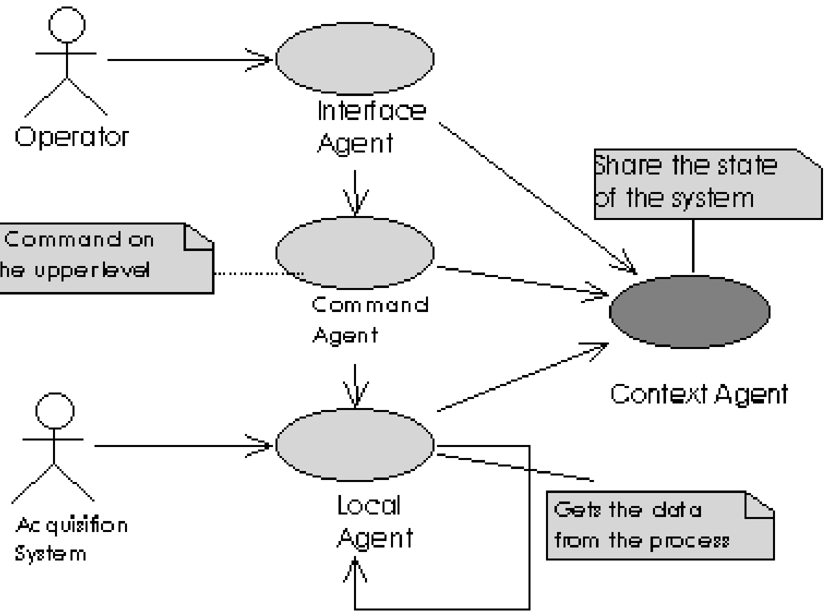

Figure 2: System Architecture

providing a standard interface between the data producers (the system) and users (clients).

The architecture implemented is client-server architecture. Clients make requests to the server that processes the request and returns the response to the clients. The requested information can be locally stored into a database.

The communication between local agents and context agent is implemented with Publisher Subscriber architecture. The data transferred by the publisher (context agent) can be received by one or many subscribers (command agent).

Event notification is the communication method used for the command agent for alarming and trending.

The context agent represents the context provided by the execution of the system: at any given moment during an execution. The context agent consists of the structure of the system and the history of interactions within the system. The messages are interpreted differently depending on the implicit context in which they are being communicated. The context agent share a common world view. The functions of the context agent are:

- Receives the state from the local agents and update the data correspondingly.

- Receives the commands from the command agent

- Expose the general state of the system at each change

An important issue is the security of the application. The Agent Pattern is useful to develop secure applications in a distributed environment (1998, Alberto Silva, Jose Delgado, The Agent Pattern for Mobile Agent Systems, INESC&IST Technical University of Lisbon). With this pattern, client is able to ignore low-level details, such as distribution or security aspects. The client manipulates agents through the AgentView reference, that is an adaptation of Proxy pattern, suitable to support transparent and secure access to different types of objects (figure 3).

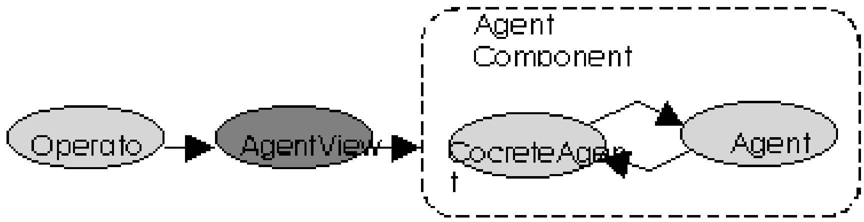

Figure 3: Interface Agent

4. Implementation

The field devices are connected through controllers to a PC computer or a process computer. The local agents are running on these computers. The PCs are connected at a supervisory computer used by an operator. On this computer runs the command agent, context agent and the interface agent. The computer is bound through a LAN adapter into a network.

To generate and test system modules, an agent framework was used: AgentTool_1.8.3 and we propose to use JACK framework to include the intelligent component that implements BDI model.

The programming language used is Java, and the Java execution environment Sun Microsystems JDK 1.3.1.

We propose to integrate the system into existing platforms for process control, as OPC (OLE for Process Control). For this we'll use data servers that manages the acquisition data. Data server is a standard OPC interface between data produces (field devices) and data users (clients) (Thomas Hadlich, OPC Command Interface, Command Interface Subgroup, Data Access 3.0 WG). Clients make requests to the server over the network for the information. The server processes the request and returns the result to the client. The requested information can be stored locally in a database or can be remote stored into the field devices. Client/server is typically used for communications between hosts and field devices.

5. Conclusion

The proposed architecture has the advantage given by the agent technologies: modularity needed in decentralized distributed systems. Modularity permits the system to be modified one piece at a time. Decentralization decouples the individual modules from another, so minimizes the impact that changing modules have on the behavior of other modules. These features are very important from industrial perspective. Agent based architectures permit reuse of existing code reducing cost and time.

The easy integration with the existing systems is another important characteristic of the agent systems.

The context agent manages the interaction between components and share the environment state of the system giving internal flexibility.

The command agent gives intelligence to the system to increase the productivity or efficiency of the system.

System flexibility (functionaly) is achieved by the distribution of the command on two layers. A given command being executed by the best suited local agent.

References

[1] Hadlich Thomas, *OPC Command Interface,* Command Interface Subgroup, Data Access 3.0 WG

[2] Kendall Elizabeth A., P.V. Murali Krishna, Chirag V. Pathak, C.B. Suresh (1999), *Patterns of intelligent and Mobile Agents,* Royal Melbourne Institute Of Technology, Australia

[3] Silva Alberto, Delgado Jose (1998), *The Agent Pattern for Mobile Agent Systems,* INESC&IST Technical University of Lisbon, Portugal

[4] Van Dyke Parunak H. (1998), *Practical and Industrial Applications of agent-Based Systems,* Industrial Technology Institute

[5] Wada Hodehiko, Sakuraba Yuichi, Negishi Masako, *Machinery Control System using Autonomous Agents,* Yokogawa Electric Corporation, Japan

[6] Walker Robert, Murphy Gail (2000), *Implicit Context: Easing Software evolution and reuse,* University of British Columbia, ACM

[7] Wooldridge Michael (2000), *Reasoning About Rational Agents,* The MIT Press, Cambridge, Massachusetts, 2000, ISBN 0-262-23213-8

[8] Wooldridge Michael, N.R. Jennings (1998), *Applications of Intelligent Agents,* Qeen Mary & Westfield College, University of London

"Ovidius" University,
Faculty of Mathematics and Informatics,
B-dul Mamaia 124,

8700 Constantza,
Romania
e-mail: ochelai@univ-ovidius.ro