107
From Holonic Control to Virtual Enterprises: The Multi-Agent Approach

107.1 Introduction

Both the complexity of manufacturing environments as well as the complexity of tasks to be solved are growing continuously. In many manufacturing scenarios, traditional centralized and hierarchical approaches applied to production control, planning and scheduling, supply chain management, and manufacturing and business solutions in general are not adequate and can fail because of the insufficient means to cope with the high degree of complexity and practical requirements for generality and reconfigurability.

These issues naturally lead to a development of new manufacturing architectures and solutions based on the consideration of highly distributed, autonomous, and efficiently cooperating units integrated by the plug-and-play approach. This trend of application of multi-agent systems (MAS) techniques is clearly visible at all levels of manufacturing and businesses. On the lowest, real-time level, where these units are
tightly linked with the physical manufacturing hardware, we referred to them as holons or holonic agents [1]. Intelligent agents are also used in solving production planning and scheduling tasks both on the workshop and factory levels. More generic visions of intensive cooperation among enterprises connected via communication network have led to the ideas of virtual enterprises:

A virtual enterprise is a temporary alliance of enterprises that come together to share skills or core competencies and resources in order to better respond to business opportunities, and whose cooperation is supported by computer networks [2].

The philosophical background of all these highly distributed solutions is the same: the community of autonomous, intelligent, and goal-oriented units efficiently cooperating and coordinating their behavior in order to reach the global level goals. The decision-making knowledge stored and exploited locally in the agents/holons invokes the global behavior of the system that is not deterministic, but rather emergent — such a behavior cannot be precisely predicted at the design time of the community. The experimental testing of the global behavior with the physical manufacturing/control environment being involved is not only extremely expensive, but nonrealistic as well. The only possible solution is the simulation, both of the controlled process of the manufacturing facility as well as the simulation of the interagent interactions.

107.2 Technology Overview

The architecture of an agent usually consists of the agent’s body and the agent’s wrapper. We can also say that the body, the functional core of an agent, is encapsulated by the wrapper to create an agent [3]. The wrapper accounts for the interagent communication and real-time reactivity. The body is an agent’s reasoning component, responsible for carrying out the main functionality of the agent. It is usually not aware of the other members of the community, their capabilities, duties, etc. This is the wrapper, which is responsible for communicating with the other agents, for collecting information about the intents, goals, capabilities, load, reliability, etc., of the other units in the agents’ community.

From the implementation point of view, there are two types of agents: (i) custom-tailored agents, which are implemented in order to provide a specific service to the community (e.g., service brokering) and (ii) integrated agents, which encapsulate a preexisting, “inherited,” or “legacy” piece of software/hardware by the agent’s wrapper into the appropriate agent structure. In this case, the wrapper provides a standardized communication interface enabling to plug the legacy system into the corresponding agents community. From the outside, such a wrapped software system cannot be distinguished from the custom-tailored agents as it communicates in a standard way with the others and understands the predefined language used for the interagent communication (e.g., Agent Communication Language — ACL defined by Foundation for Intelligent Physical Agents [FIPA]). These agentification processes provide an elegant mechanism for system integration — a technique supporting the technology migration from the centralized systems toward the distributed agent-based architectures.

We distinguish three different concepts of agency that we need to explain in more detail.

Mobile Agents

Mobile agents are generally pieces of code traveling freely inside a certain communication network (usually inside the Internet). Such agents, being usually developed and studied in the area of computer science, are stand-alone, executable software modules, which are being sent to different host computers/servers to carry out a specific computational tasks (usually upon the locally stored data). They are expected to report back the results of the computation process. The mobile agents can travel across the network, can be cloned or destroyed by their own decision when fulfilling their specific task, etc.

MAS

MAS can be described as goal-oriented communities of cooperative/self-interested agents in a certain interaction environment. They have been developed within the domain of the distributed artificial intelligence
and they explore the principles of artificial intelligence for reasoning, communication, and cooperation. The important attribute of each agent is its autonomy — the agent resides on a computer platform where it autonomously carries out a particular task/functionality. The agent owns only a part of the global information about the goals of the community that is sufficient for its local decision-making and behavior. However, in some situations, for example, when the agent is not capable of fulfilling the requested operation alone, the agent is capable of cooperation with other agents (asking them for help) usually via message sending. The agents asked for cooperation are still autonomous in their decisions, that is, can either agree on cooperation or can even refuse to cooperate (e.g., due to lack of their own resources). Another important attribute of MAS is the plug-and-play approach — the agents, which can be grouped into different types of communities (such as teams, coalitions, platforms, etc., see Section 107.3), can freely join and leave the communities. Usually, the community offers a yellow-pages-like mechanism that is used by agents to offer their services to the other agents as well as to find out suitable agents for possible cooperation. This allows to dynamically change the overall capabilities and goals of the MAS (according to user requests) by adding new agents with desired functionalities.

The cooperation among the agents supported by their social behavior is the dominant feature of the activities of the agents in the community. The term social behavior means that the agents are able to communicate, to understand the goals, states, capabilities, etc., of the others and to respect the general rules and constraints of behavior valid for each of the community members. Communication (exchange of information usually in the form of messages) plays the crucial and decisive role in the agents’ communities with social behavior of either a cooperative or competitive nature.

Holons

Holons are agents dedicated to real-time manufacturing tasks [4]. These are tightly physically coupled, really “hard-wired” with the manufacturing hardware (devices, machines, workshop cells), thus with a low degree of freedom in their mutual communication. The holons operate in the “hard” real time, and their patterns of behavior are strictly preprogrammed. This means that their reactions in certain situations are predictable and emergent behavior is not highly appreciated (usually not allowed). The Holonic Manufacturing Systems are (HMS) designed mainly to enable a fast and efficient system reconfiguration in the case of any machine failure. One of the pioneering features of holonic systems is a complete separation of the data flow from the control instructions. The communication principles, supporting the holons’ interoperability, are already standardized (the IEC 61499 standard; see Section 107.6).

107.3 Cooperation and Coordination Models

As we have already mentioned, communication among the agents is an important enabler of their social behavior. The agents usually explore specific communication language with standardized types of messages. The set of messages is chosen so that it represents the most typical communicative acts, often called performatives (according to the speech act theory of Searle [5]), used by agents in a particular domain. Examples of such performatives can be request used by an agent to request the other one to perform a particular operation, agree used to confirm the willingness to cooperate, or, for example refuse to deny to perform requested action. Along with the performative, the message bears the information about the sender and receiver, the content of the message, and the identification of the language used for the message’s content. Additional attributes can be included in the message, for example, the reference to the appropriate knowledge ontology describing semantics of the message (see Section 107.5) or the brief description of the negotiation strategy used (the structure of replies expected).

The communication among the agents is usually not just a random exchange of messages, but the message flow is managed by a set of standard communication protocols. These protocols range from a simple “question–reply,” via “subscribe–inform” through to more complex negotiation protocols like “Contract-Net-Protocol” (CNP), different versions of auctions (Dutch, sealed, Vickery, etc.). The communication protocols and communication traffic in general can be represented graphically by means of interaction
diagrams. Figure 107.1 shows the Request and Contract-net protocols from FIPA specifications captured in AUML — Agent-based extensions to the standard UML.

The more knowledge is available locally, which means it is “owned” by individual agents, the smaller the communication traffic needed to achieve cooperative social behavior. One of the crucial questions is how to store and use the knowledge locally. For this purpose, different acquaintance models are located in the wrappers of individual agents. These are used to organize, maintain, and explore knowledge about the other agents (about their addresses, capabilities, load, reliability, etc.). This kind of knowledge, which strongly supports collaboration activities among the agents, is called social knowledge [3]. These models can be used to organize both the long-term as well as temporary or semipermanent knowledge/data concerning cooperation partners. To keep the temporary and semipermanent knowledge fresh, several knowledge maintenance techniques have been developed, namely (i) periodic knowledge revisions — the knowledge is updated periodically by regular “question–answer” processes, (ii) subscription-based update — the knowledge update is preordered by a specific subscription mechanism. The field is strongly influenced and motivated by Rao and Georgeff’s BDI (Beliefs-Desires-Intentions) model [11] used to express and model agents’ beliefs, desires and intentions, and other generic aspects of MAS.

The MAS research community provides various techniques and components for creating the architecture of an agent community. The crucial categories of agents, according to their intra- and inter-community functionalities (e.g., resource agents, order or customer agents, and information agents) have been identified. Services have been defined for specific categories of agents (e.g., white pages, yellow pages, brokerage, etc). In some of the architectures, the agents do not communicate directly among themselves. They send messages via facilitators, which play the role of communication interfaces among collaborating agents [6]. Other architectures are based on utilization of matchmakers, which proactively try to find the best possible collaborator, brokers that act on behalf of the agent [7], or mediators that coordinate the agents by suggesting and promoting new cooperation patterns among them [8]. Increasing attention has recently been paid to the concept of the meta-agent, which independently observes the interagent communication and suggests possible operational improvements [9].

The techniques for organizing long-term alliances and short-term coalitions as well as techniques for planning of their activities (team action planning) have been developed recently [10]. These algorithms can help to solve certain types of tasks more efficiently and to allocate the load among the agents in an

FIGURE 107.1  FIPA’s Request and Contract-net communication protocols.
optimal way. They can be explored with an advantage for automated creation and dissolving of virtual organizations as well as for an optimal load distribution among the individual bodies in a virtual organization. Alliance and coalition formation techniques can be linked with methodologies and techniques for administration and maintenance of private, semiprivate, and public knowledge. This seems to be an important issue to tackle, especially in virtual organizations of temporary nature where units competing in one business project are also contracted to cooperate in the other one. It is also possible to classify and measure the necessary leakage of private/semiprivate knowledge to reflect this fact in the future contracts.

107.4 Agents Interoperability Standardization — FIPA

The FIPA is a nonprofit association registered in Geneva, Switzerland, founded in December 1995. The main goal of FIPA is to maximize interoperability across agent-based applications, services, and equipment. This is done through FIPA specifications.

FIPA provides specifications of basic agent technologies that can be integrated by agent systems developers to make complex systems with a high degree of interoperability. FIPA specifies the set of interfaces, which the agent uses for interaction with various components in the agent’s environment, that is, humans, other agents, nonagent software, and the physical world. It focuses on specifying external communication among agents rather than the internal processing of the communication at the receiver.

The FIPA Abstract Architecture defines a high-level organizational model for agent communication and core support for it. It is neutral with respect to any particular network protocol for message transport or any service implementation. This abstract architecture cannot be directly implemented; it should be viewed as a basis or specification framework for the development of particular architectural specifications.

The FIPA Abstract Architecture contains agent system specifications in the form of both descriptive and formal models. It covers three important areas: (i) agent communication, (ii) agent management, and (iii) agent message transport.

Agent Communication and Agent Communication Language

FIPA provides standards for agent communication languages. The messages exchanged among the agents must comply with a FIPA-ACL specification. FIPA-ACL is based on speech act theory [5] and resembles Knowledge Query Manipulation Language (KQML). Each message is labeled by a performative, denoting a corresponding communicative act (see previous section), such as inform or request. Along with the performative, each message contains information about its sender and receiver, the content of the message, a content language specification, and an ontology identifier. Other important attributes of an message are the information about the conversation protocol that applies to the current message (e.g., FIPA-Request) and the conversation ID that uniquely identifies to which “conversation thread” this message belongs (since there can be more conversations between two agents following the same protocol at the same time).

The core of the FIPA ACL message is its content, which is encoded in language denoted in the language slot of the message. FIPA offers a Semantic Language (FIPA-SL) as a general-purpose knowledge representation formalism for different agent domains. This formalism maps each agent message type (performative) to an SL formula that defines constraints (called feasibility conditions) that the sender has to satisfy, and to another formula that defines the rational effect of the corresponding action. Nevertheless, any other existing or user-defined language can be used as a content language of FIPA messages. One of the most popular languages used today to express the message syntax is the XML.

Bellow is an example of a FIPA message, where the agent “the-sender” requests the agent “the-receiver” to deliver a specific box to a specific location, using the XML content language and the FIPA-Request conversation protocol.

(request
  :sender (agent-identifier :name the-sender)
  :receiver (set (agent-identifier :name the-receiver))
  :content
Agent Management

The FIPA Agent Management Specification provides a specification of how the members of the multi-agent community shall be registered, organized, and managed. According to the FIPA philosophy, the agents are grouped into high-level organizational structures called agent platforms (APs). Members of each platform are usually geographically “close” — for example, they may be run on one computer or be located in a local area network. Each platform must provide its agents with the following two mandatory services: the Agent Management System (AMS) and the Directory Facilitator (DF).

The AMS administers a list of agents registered with the platform. This component implements the creation and deletion of the running agents and provides the agents with a “white-page-list” type of service, for example, a list of all agents accommodated on the given agent platform and their addresses. Unlike the AMS, the DF supplies the community with a “yellow-page-list” type of service. Agents register their services with the DF and can query the DF to find out what services are offered by other agents, or to find all agents that provide a particular service. Thus, agents can find the addresses of others that can assist in accomplishing the desired goals.

Message Transport Service

The Message Transport Service (MTS) is a third mandatory component (besides AMS and DF) that the agent platform has to offer to the agents. The MTS is provided by so-called Agent Communication Channel (ACC), which is responsible for the physical transportation of messages among agents local to a single AP as well as among agents hosted by different APs. For the former case, FIPA does not mandate to use a specific communication protocol or interface — different protocols are being used today in agent platform implementations, such as the TCP/IP sockets, the UDP protocol, or, particularly in JAVA implementations, the JAVA Remote Method Invocation (RMI). In the latter case, the FIPA defines a message transport protocol (MTP) that ensures the interoperability between agents from different agent platforms.

For this purpose, the ACC must implement the MTP for at least one of the following communication protocols specified by FIPA: the IIOP (Internet Inter-Orb Protocol), WAP, or HTTP.

107.5 Ontologies

Ontologies play a significant role not only in the interagent communication, where the content of messages exchanged among agents must conform to some ontology in order to be understood, but also in knowledge capturing, sharing, and reuse. One of the main reasons why ontologies are being used is the semantic interoperability enabling among others:

- to share knowledge — by sharing the understanding of the structure of information exchanged among software agents and people,
- to reuse knowledge — ontology can be reused for other systems operating on a similar domain, and
- to make assumptions about a domain explicit — for example, for easier communication.

Basically, ontology can be referred to as a vocabulary providing the agents with the semantics of symbols, terms, or keywords used in messages [12]. Thus, if an agent sends a message to another agent using particular ontology, it can be sure that the other agent (of course, if it shares the same ontology) will understand the message.
Ontologies for MAS

Two examples can be selected to illustrate multi-agent-oriented ontological efforts in the area of manufacturing. The first one, FIPA Ontology Service Recommendation, is a part of a set of practical recommendations on how to implement agents in a standardized way. The other one, Process Specification Language (PSL) project [13], tries to develop general ontology for representing manufacturing processes. Its aim is to serve as interlingua for translating between process ontologies. The transformation between ontologies for translation using PSL is expected to be defined by humans.

The former approach seems to be much more general and applicable. FIPA uses Open Knowledge Base Connectivity, (OKBC) [14], as a base for expressing ontologies. OKBC is an API for accessing and modifying multiple, heterogeneous knowledge bases. Its knowledge model defines a meta-ontology for expressing ontologies in an object-oriented frame-based manner. OKBC can be mapped to the object-oriented languages, so that classes in programming languages can be built on the underlying ontology and be used for exchanging information. Semantics of OKBC constructs is defined in KIF as a description of what the constructs intuitively mean. However, no reasoning engine that would enable to use this information for, for example, ontology integration, is provided. Moreover, it could be difficult to provide a reasoning support for some of the constructs.

Ontologies in FIPA proposals and related ontologies for practical applications [15] are motivated mainly by the need to have something that would work immediately, because currently more attention is paid to the functional behavior of agents. There is nothing wrong with this approach, if we want to have a working solution in a short time where we do not care about the possibility of reasoning about the ontologies and further interoperability. However, the need for reasoning about ontologies can easily arise, for example, when requiring interoperability in open multiagent systems, that is, systems where new agents with possibly other ontologies can join the community.

107.6 HMS

Over the past 10 years, researches attempted to apply the agent technology to various manufacturing areas such as supply chain management, manufacturing planning, scheduling, and execution control. This effort resulted in the development of a new concept, the HMS, based on the ideas of holons presented by Koestler [17] and strongly influenced by the requirements of industrial control.

Holons are autonomous, cooperative units that can be considered as elementary building blocks of manufacturing systems with decentralized control [16]. They can be organized in hierarchical or heterarchical structures. Holons, especially those for real-time control, are usually directly linked to the physical hardware of the manufacturing facility and are able to physically influence the real world (e.g., they may be linked to a device, tool or other manufacturing unit, or to a transportation belt or a storage manipulator).

Holons for real-time control are expected to provide reactive behavior rather than being capable of deliberative behavior based on complex “mental states” and strongly proactive strategies. They are expected mainly to react to changes in the manufacturing environment (e.g., when a device failure or a change in the global plan occurs). Under “stable circumstances,” during routine operation, they are not required to change the environment proactively. The reason for the prevalence of reactive behavior of real-time control holons is that each of them is linked to a physical manufacturing facility/environment, changes to which are not very simple, cheap, or desirable in a comparatively “stable” manufacturing facility. The physical linkage to physical equipment seems to be a strong limiting factor of the holons’ freedom in decision-making.

The more generic holonic ideas and considerations have led to the vision of a holonic factory [18]. Here, all the operations (starting from product ordering, planning, scheduling, and manufacturing, to invoicing the customer) are based entirely on holonic principles. A holonic factory contains a group of principal system components (holons) that represent physical manufacturing entities such as machines or products as well as virtual entities like orders or invoices. The holons work autonomously and cooperate together in order to achieve the global goals of the factory. Thus, the factory can be managed toward global goals by
the activities of individual autonomous holons operating locally. The community of researchers trying to implement the vision of the holonic factory is well organized around the international HMS consortium.

The vision of the holonic factory covers several levels of information processing for manufacturing. We can distinguish at least three separate levels, namely,

- **real-time control**, which is tightly linked with the physical manufacturing equipment;
- **production planning and scheduling**, both on the workshop and on the factory level; and
- **supply chain management**, integrating a particular plant with external entities (supplier, customers, cooperators, sales network, etc.).

At the lowest RT-control level, the main characteristics of holons is their linkage to the physical manufacturing devices — these holons read data from sensors and send control signals to actuators. Within the HMS activities, the standard IEC-61499 known as *function blocks* has been developed for these RT-control purposes. It is based on function blocks part of the well-known IEC-1131-3 standard for languages in Programmable Logical Controllers (PLCs). The major advantage is the separation between the data flow and the event flow among various function blocks. Multiple function blocks would be logically grouped together, across multiple devices into an application, to perform some process control.

Since the IEC-61499 fits well these RT-control purposes, it does not address the higher level aspects of holons acting as cooperative entities capable of communication, negotiation, and high-level decision making. It is obvious that this is the field where the techniques of multi-agent systems have to be applied. Thus, a general architecture combining function-bocks with agents was presented in [1]. As shown in Figure 107.2, a software agent and function block control application (connected to the physical layer) are encapsulated into a single structure.

In such a holon equipped with a higher-level software component, three communication channels should be considered:

- **Intraholon** communication between the function block part and the software agent component.
- **Interholon** communication that is aimed at communication among the agent-based parts of multiple holons — FIPA standards are used more and more often for this purpose.
- **A direct** communication channel between function block parts of neighboring holons. If we are prepared to break the autonomy of an independent holon, then this communication is standardized by IEC 61499 already; otherwise, a new type of real-time coordination technology is needed to ensure real-time coordination.

![FIGURE 107.2 Holonic agent: combination of function block application and software agent.](image-url)
As a matter of fact, the holons defined in this way behave — on the level of interholon communication — like standard software agents. They can communicate widely among themselves, carry out complex negotiations, cooperate, develop manufacturing scenarios, etc. We can call them holonic agents (or agentified holons), as they consist of both a holonic part connected with the physical layer of the manufacturing system (operating in hard real time) and a software agent for higher-level, soft real-time or non-real-time intelligent decision making. It has already been mentioned that the interholon communication is usually standardized by the FIPA approach, and direct communication could be achieved by IEC standards (not necessarily IEC 61499). Let us stress that the FIPA standards are not applicable for the low-level real-time control purposes as they do not take account of the real-time control aspects.

The attention of system developers is currently directed mainly at the intraholon communication, which is usually both application- and company-specific and is usually connected with the solution of the “migration problem” (the problem of exploring the classical real-time control hardware for holonic control). McFarlane et al. [19] introduced a blackboard system for accomplishing the intra-holon communication, while others [20] proposed using a special management service interface function block.

It is expected that the communication among holonic agents will be standardized for many reasons. One reason is that these holonic agents should be involved in global communities of company agents, where they can directly participate in supply chain management negotiations or contribute to virtual enterprise simulation games, etc. The FIPA communication standards are considered preferable for implementing the inter-holon communication. To develop these standards, the HMS community must declare messages, define their semantics, and develop the appropriate knowledge ontologies (see Section 107.5). This seems to be quite a demanding task, as manufacturing, material-handling, production planning, and supply chain management requirements differ significantly between different industries and between different types of production.

From a wider perspective, for the FIPA standards to be applicable to holonic manufacturing, they should take account of the preexistence and coexistence of other standards. In manufacturing industry, there is STEP (Standard for the Exchange of Product Model Data), which is a comprehensive ISO standard (ISO 10303) that is used for representation and exchange of engineering product data and specifies the EXPRESS language for product data representation in any kind of industry. Integration of these widely accepted concepts with the HMS and FIPA effort seems to be of high importance. On the level of physical interoperability, there are various standards, such as the TCP/IP and UDP protocols, Common Request Broker Architecture (CORBA), Distributed Common Object Model (DCOM), and others. Similarly, the use of higher-level cooperation standards in the area of multi-agent systems such as KQML, FIPA, and JINI is inevitable for dynamic, flexible, and reconfigurable manufacturing enterprises.

107.7 Agent Platforms

A complex nature of the agents (high-level decision-making units capable of mutual collaboration) requires using a high-level programming language such as C++ or JAVA for their implementation. As mentioned in the previous section, for manufacturing purposes the software agents’ parts of holonic agents have to be able to interact with the low-level control layer. In the majority of current holonic testbed implementations, the low-level holonic control (connected to the physical layer) is usually carried out by IEC 1131-3 (mainly ladder logic) or IEC 61499 function block programs that run on industrial PLC-based automation controllers. However, the software agent parts, implemented in C++ or JAVA, are running separately on a standard PC and communicate, for example, via a blackboard system (part of the data storage area in a controller allocated for each holonic agent and shared by the agent- and holonic-subsystem of the holonic agent).

It is obvious that for real industrial deployment, particularly where a high degree of robustness is required, the use of PC(s) for running agent-components of holonic agents is not safe and is also not possibly feasible for certain types of control systems. The only acceptable solution is to run holonic agents as wholes directly within PLC-based controllers. One controller can host one or more holonic agents, but not all of them — they have to be distributed in reasonable groups over several controllers and allowed to communicate with each other either within a single controller or among different controllers.
The major issue of such a solution is to extend the current architecture of a PLC in such a way that it is able to run software agents written in a high-level programming language in parallel with the low-level control code and also provide the interface for interactions between these two layers. The programming language in which the software agents should be implemented can either be C++ or JAVA. However, there are many reasons to prefer the JAVA language to be the target one. One of its advantages is the portability of JAVA programs, which the user develops independent of hardware platforms or operating systems — the same application can run either on a PC with Microsoft Windows or Unix/Linux or on a small device like Personal Digital Assistant (PDA) or a mobile phone with Windows CE, Symbian, or other operating systems with JAVA support. Another reason to choose JAVA is that currently there are a large number of JAVA-based agent development tools available, either as commercial products or open-source projects, that simplify the development of agent systems. Moreover, some of them are fully compliant with the FIPA specifications, which insures the desired interoperability.

Agent Development Tools Characteristics

Basically, the agent development tool, often called an agent platform, provides the user with a set of JAVA libraries for specification of user agent classes with specific attributes and behaviors. A kind of a run-time environment that is provided by the agent platform is then used to actually run the agent application. This run-time environment, implemented in JAVA as well, particularly ensures transport of messages among agents, registration, and deregistration of agents in the community (white pages services) and also registration and lookup for services provided by agents themselves (yellow pages services). Some other optional tools can also be a part of the agent platform runtime, for instance, a graphical viewer of messages sent among agents, etc.

The implementation of JAVA-based agents in the automation controllers obviously requires such an agent platform runtime to be embedded into the controller architecture. Since it is used as a background for the real-time holonic agents, there are specific requirements on the properties of the agent platform, such as speed, memory footprint, reliability, etc. The evaluation of available JAVA agent platforms, presented in the following paragraphs, has been conducted [21] in order to find out to what extent they fulfill these criteria and therefore which ones are best suitable for the purposes of manufacturing control.

FIPA Compliancy

Compliance with the FIPA standards has been recognized as a crucial property ensuring the interoperability of holonic agents not only at the lowest real-time control level (allowing, e.g., communication of different kinds of holonic agents hosted by PLC controllers from different vendors) but also the interoperability between holonic agents and other agents at higher levels of information processing within the company, for example, data-mining agents, ERP agents, supply chain management agents, and so on.

The FIPA specification of the message transport protocol (see the Section Message transport service) defines how the messages should be delivered among agents within the same agent community and particularly between different communities. For the latter case, the protocol based on IIOP or HTTP ensures the full interoperability between different agent platform implementations. It means that the agent running, for example, on the JADE agent platform can easily communicate with the agent hosted by the FIPA-OS platform, etc.

Costs and Maintainability of the Source Code

From the cost point of view, the agent platforms that are currently available can basically be divided into two categories: free and commercial ones. Majority of the free agent platforms are distributed under a kind of an open source license (e.g., GNU Lesser General Public License), which means that you are provided with the source codes and allowed to modify them. This is an important characteristic since the integration of the agent platform into the PLC-based controllers certainly requires some modifications to be made, for example, due to different versions of JAVA virtual machine supported by the controller, the specifics of the TCP/IP communication support, or other possible issues and limitations.
On the other hand, in the case of commercial products the cost in order of thousands USD per each installation, for example, can considerably increase the total cost of the agent-based control solution where a large number of PLC controllers, PCs, and possibly other devices running agents are expected to be deployed. Moreover, the source codes are not available, so that all modifications of the platform that need to be made in order to port it to another device has to be committed to the company developing the agent platform.

Memory Requirements

An issue that has to be taken into account is usually a limited memory available for user applications on the controller. Within the RAM memory of the controller, which can, for example, be about 4 to 8 MB, the agent platform run-time environment, the agents themselves and also the low-level control code (ladder logic or function blocks) have to fit inside. There are also smaller PLC-like devices that can have only 256 KB of memory available, which would be a strong limitation factor for integrating the run-time part of the agent platform. Fortunately, the agent platform developers, especially in the telecommunication area, are seriously interested in deploying agents on small devices like mobile phones or PDAs, that is, on devices with similar memory limitations. Due to this fact, for some of the agent platforms, their lightweight versions have been developed, usually implemented in Java2 Micro Edition (CLDC/MIDP) [22]. It has been documented [23] that the memory footprint of such an agent platform runtime can be less than 100KB, that is, small enough to fit well within the memory capacity limits of majority of small mobile devices and thus the PLC-based automation controllers as well.

Message Sending Speed

The last factor considered in this evaluation is the speed of the message sending between the agents. It has already been argued that the holonic agents are expected to be used for real-time control applications where a fast reaction can be a vital characteristic. In Figure 107.2 (Chapter 6), a direct communication channel between RT control subsystems of neighboring holons is conceded but it obviously breaks the autonomy of holonic agents. If we are not willing to accept such a violation, communication at the agent level is the only allowable way of interaction among holonic agents. Thus, the agent platform runtime, carrying out such interactions, should be fast enough to ensure reasonable message delivery times (i.e., in the order of milliseconds or tens of milliseconds).

We have conducted a series of tests to compare the message-sending speed of different agent platforms. Detailed information about the benchmarking testbed configuration and the speed measuring results can be found in the subsequent section.

Agent Platforms Overview

Table 107.1 gives an overview of majority of currently available agent development tools with respect to the properties discussed in previous paragraphs. A security attribute has been added as a property of the agent platform ensuring secure communication (usually via SSL), authorization, authentication, permissions, etc. The ✓ sign indicates that an agent platform has a particular property; meanwhile, the ✗ sign indicates that such a property is missing. If a ? sign is used, there is no reference to such a property in available sources and it can be assumed that the platform does not have it.

The Java Agent Services (JAS) have also been included in Table 107.1. However, this project is aimed at the development of the standard JAVA APIs (under the javax.agent namespace), that is, a set of classes and interfaces for the development of your own FIPA-compliant agent-based systems. From this perspective, JAS cannot be considered as a classical agent platform, since it does not provide any run-time environment that could be used to run your agents (either on a PC or possibly on an automation controller).

The JINI technology [24] has not been considered in this evaluation either. Similar to JAS, JINI is a set of APIs and network protocols (based on JAVA Remote Method Invocation) that can help you to build and deploy distributed systems. It is based on the idea of services providing useful functions on the network and the lookup service that helps clients to locate these services. Although JINI provides a solid framework for various agent implementations (see, e.g., [25]), it cannot itself be regarded as an agent platform.
TABLE 107.1  Agent Platforms Overview

<table>
<thead>
<tr>
<th>Agent Platform</th>
<th>Developer</th>
<th>FIPA Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Agent Communication</td>
</tr>
<tr>
<td>JADE</td>
<td>CSELT/Emorphia</td>
<td>√</td>
</tr>
<tr>
<td>(Java Agent DEvelopment Framework)</td>
<td><a href="http://jadecselt.it">http://jadecselt.it</a></td>
<td></td>
</tr>
<tr>
<td>FIPA-OS</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>ZEUS</td>
<td>British Telecom</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td><a href="http://www.labs.bt.com/projects/agents/zeus">www.labs.bt.com/projects/agents/zeus</a></td>
<td></td>
</tr>
<tr>
<td>JACK</td>
<td>Agent Oriented Software</td>
<td>√</td>
</tr>
<tr>
<td>(Jack Intelligent Agents)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRASSHOPPER 2</td>
<td>IKV ++ Technologies AG</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td><a href="http://www.grasshopper.de">http://www.grasshopper.de</a></td>
<td></td>
</tr>
<tr>
<td>ADK</td>
<td>Tryllian</td>
<td>√</td>
</tr>
<tr>
<td>(Agent Development Kit)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAS</td>
<td>Fujitsu, HP, IBM, SUN,...</td>
<td>√</td>
</tr>
<tr>
<td>(Java Agent Services API)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AgentBuilder</td>
<td>IntelliOne Technologies</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td><a href="http://www.agentbuilder.com">http://www.agentbuilder.com</a></td>
<td></td>
</tr>
<tr>
<td>MadKit</td>
<td>MadKit Team</td>
<td>x</td>
</tr>
<tr>
<td>(Multi-Agent Development Kit)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comtec Agent Platform</td>
<td>Communication Technologies</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td><a href="http://cas.comtec.co.jp/ap/">http://cas.comtec.co.jp/ap/</a></td>
<td></td>
</tr>
<tr>
<td>Bee-gent</td>
<td>Toshiba</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td><a href="http://www2.toshiba.co.jp/beeagent/index.htm">http://www2.toshiba.co.jp/beeagent/index.htm</a></td>
<td></td>
</tr>
<tr>
<td>Aglets</td>
<td>IBM Japan</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td><a href="http://frl.ibm.com/aglets">http://frl.ibm.com/aglets</a></td>
<td></td>
</tr>
</tbody>
</table>
Message-Sending Speed Benchmarks

It has been discussed earlier that a speed at which the messages are exchanged among agents can be a crucial factor in agent-based real-time manufacturing applications. Thus, we have put selected agent platforms through a series of tests where the message delivery times have been observed under different conditions.

In each test, the so-called average round-trip time (avgRTT) is measured. This is the time period needed for a pair of agents (let say A and B) to send a message (from A to B) and receive a reply (from B to A). We use a JAVA System.currentTimeMillis() method, which returns the current time as the number of milliseconds since midnight, January 1, 1970. The round-trip time is computed by the A-agent when a reply from B is received as a difference between the receive time and the send time. An issue is that a millisecond precision cannot be mostly reached; the time grain is mostly 10 or 15 msec (depending on the hardware configuration and the operating system). However, it can easily be solved by repeating a message exchange several times (1000 times in our testing) and computing the average from all the trials.

As can be seen in Table 107.2, three different numbers of agent pairs have been considered: 1 agent pair (A–B) with 1000 messages exchanged, ten agent pairs (A1–B1, A2–B2, …, A10–B10) with 100 messages exchanged within each pair, and finally 100 agent pairs (A1–B1, A2–B2, …, A100–B100) with ten messages per pair. Moreover, for each of these configurations, two different ways of executing the tests are applied. In the serial test, the A agent from each pair sends one message to its B counterpart and when a reply is received, the round-trip time for this trial is computed. It is repeated in the same manner N-times (N is 1000/100/10 according to number of agents) and after the Nth round-trip is finished, the average response time is computed from all the trials. The parallel test differs in such a way that the A agent from each pair sends all N messages to B at once and then waits until all N replies from B are received. In both the cases, when all the agent pairs are finished, from their results the total average round-trip time is computed.

As the agent-based systems are distributed in their nature, all the agent platforms provide the possibility to distribute agents on several computers (hosts) as well as run agents on several agent platforms (or parts of the same platform) within one computer. Thus, for each platform, three different configurations have been considered: (i) all agents running on one host within one agent platform, (ii) agents running on one host but within two agent platforms (i.e., within two Java Virtual Machines — JVM), and (iii) agents distributed on two hosts. The distribution in the last two cases was obviously done by separation of the A–B agent pairs.

The overall benchmark results are presented in Table 107.2. Recall that the results for serial tests are in milliseconds (msec) while for parallel testing, seconds (sec) have been used. Different protocols used by agent platforms for the interplatform communication are also mentioned: Java RMI for JADE and FIPA-OS, TCP/IP for ZEUS, and UDP for JACK. To give some technical details, two Pentium II processor-based computers running 600 MHz (256 MB memory) with Windows 2000 and Java2 SDK v1.4.1_01 were used. Some of the tests, especially in the case of 100 agents, were not successfully completed mainly because of communication errors or errors connected with the creation of agents. These cases (particularly for FIPA-OS and ZEUS platforms) are marked by a \( \times \) symbol.

Platforms — Conclusion

On the basis of the results of this study, the JADE agent platform seems to be the most suitable open-source candidate for the development tool and the run-time environment for agent-based manufacturing solutions. In comparison with its main competitor, FIPA-OS, the JADE platform offers approximately twice the speed in message sending and, above all, a much more stable environment, especially in the case of larger number of agents deployed.

Among the commercial agent platforms, to date, only JACK can offer full FIPA compliancy and also cross-platform interoperability through a special plug-in — JACK agents can send messages, for example, to JADE or FIPA-OS agents (and vice versa) via the FIPA message transport protocol based on HTTP. In the intraplatform communication, based on the UDP protocol, JACK (unlike other platforms) keeps
pace with JADE in case of one host and even surpasses it in other cases, being approximately 2–3 times faster. Considering the full implementation of the Belief–Desire–Intention (BDI) model, JACK can be regarded as a good alternative to both the open-source JADE and FIPA-OS platforms.

107.8 Role of Agent-Based Simulation

The process of developing and implementing a holonic system relies on several phases and widely explores the simulation principles. The simulation process and its fast development using efficient simulation tools represent the key tasks for any implementation of a real-life holonic/agent-based system. The following stages of the design process based on simulation can be gathered like this:

1. Identification of holons/agents: The design of each holonic system starts from a thorough analysis of:
   a. The system to be controlled or manufacturing facility to be deployed.
   b. The control/manufacturing requirements, constraints, and hardware/software available.

   The result of this analysis is the first specification of holon/agent classes (types) to be introduced. This specification is based on the application and its ontology knowledge. The obvious design principle is that each device, or each segment of the transportation path or each workcell is represented by a holon.

2. Implementation/Instantiation of holon classes from the holon/agent type-library. The holon/agent-type library is either developed (step 1) or reused (if already available). Particular holons/agents
are created as instances of the holonic definitions in the holon/agent-type library. Furthermore, the implementation of communication links among these holon/agent instances is established within the framework of initialization from these generic holon/agent classes (for instance, holons are given the names of their partners for cooperation).

3. **Simulation:** The behavior of a holonic system is not deterministic, but rather emergent — the decision-making knowledge stored locally in the agents/holons invokes the global behavior of the system in a way that cannot be precisely predicted. Yet, the direct experimental testing of the global behavior with the physical manufacturing/control environment being involved is not only extremely expensive, but nonrealistic as well. Simulation is the only way out. For this purpose, it is necessary to have:

(a) A suitable tool used to model and simulate the physical processes in the manufacturing facility. Standard simulation tools like, for example, Arena, Grasp, Silk, or Matlab can be used for these purposes.

(b) A suitable agent run-time environment for modeling the interactions of holonic agent parts. On the basis of the results of agent platforms comparison (Section 107.7), the JADE and the FIPA-OS platforms as open sources or JACK as a commercial tool can be recommended.

(c) A good simulation environment to model the real-time parts of multiple holons. There are function block emulation tools from Rockwell Automation (Holobloc [26]) and the modified 4-control platform from Softing [27]. Yet, these tools do not adequately generate and handle real-time control problems. A more sophisticated real-time solution would be to use embedded firmware systems like JBED, with its time-based scheduler, to run JAVA objects (which simulate function blocks) and manage events in a realistic manner.

(d) Human–Machine Interfaces (HMI) for all the phases of the system design and simulation.

4. **Implementation of the target control/manufacturing system:** In this stage, the target holonic control or manufacturing system is reimplemented into the (real-time) running code. This implementation usually relies on ladder logic, structured text, or function blocks at the lowest level of control. However, some parts of the targeted manufacturing systems (such as resource or operation planning subsystems) are often reused as in the phase 3. For example, in the eXPlanTech production planning MAS [28], there was 70% of the real code reused from the simulation prototype. Therefore, the choice of the multi-agent platform in the phases 3 and 4 is critical (it has been advised to operate with one platform only).

**107.9 Conclusions**

Why does the agent technology seem to be so important for the area of manufacturing? What are the reasons and advantages of applying them? What do they really bring? Let us try to summarize the current experience shared by both the holonic and multiagent communities. The main advantages can be summed up as follows:

1. **Robustness and flexibility of the control/diagnostic systems:**

(a) Robustness is achieved mainly due to the fact that there is *no central element*, no centralized decision making. Any loss of any subsystem cannot cause a fatal failure of any other subsystem.

(b) The agent technology enables to handle the problems of *production technology failures* in a very efficient way. Optimal reconfiguration of the available equipment (which remains in operation) can be carried out in a very fast way. Thus, sustainable continuation of the production task or operation or safety stopping of the manufacturing process can be achieved. (Similarly, accomplishment of the life-critical part of a mission — after an important part of the equipment has been destroyed — can be achieved in the military environment).

(c) *Changes in the production facility* (adding a machine, deleting a transportation path, etc.) can be handled on the fly, without any need to reprogram the software system as a whole. Just a couple of messages are exchanged, and the agents are aware of the change and behave accordingly.
Changes in production plan or schedule can be handled easily, without the need for stopping the process or bringing it back to some of the initial states. The changes in the production plan or schedule can be handled in parallel to solving the tasks connected with changes in the facility equipment and/or failures.

2. The plug-and-play approach is strongly supported. This enables to change/add/delete the hardware equipment as well as software modules on the fly. The migration process from the old to the new technology can be carried out smoothly, on a permanent basis, without any need to stop the operation. This also makes the system maintenance costs significantly cheaper.

3. Control and diagnostics are carried out as near to the physical processes as possible; control and diagnostic subsystems can cooperate on the lowest level (and in a much faster way). Control and diagnostics can be really fully integrated. This fact improves the behavior of control/diagnostic systems in the hard real-time control/diagnostic tasks. Moreover, it is possible to change the principles of behavior centrally, just by changes in the rules or policies known to each of the agents.

4. The same agent-based philosophy can be used on different levels, in different subsystems of the manufacturing facility and company. The same agent-oriented principles and techniques can be, for example, applied on the hard real-time level (holonic control), soft real-time control, strategic decision making for control tasks, for integrated diagnostics or diagnostics running as a separate process aside of control, for production planning and scheduling, for higher-level decision making on the company level, for supply-chain management, as well as for the purposes of virtual enterprises (viewed as coalitions of cooperating companies). Despite the same communication standards and negotiation scenarios being used across all the tasks mentioned above, a very high efficiency resulting from automatic communication and negotiation between units on different levels and located in different subsystems can be obtained.

Besides the advantages of the agent-based solutions, several disadvantages can also be easily identified:

1. The investments needed to implement the agents-based manufacturing system are higher. Unfortunately, the available flexibility, which is the payoff for these expenses, is usually so enormous that the manufacturing process can leverage just a very small portion of it.

2. As there is no central control element present (in an ideal agent-based factory), in the society of mutually communicating agents, unpredictable, emergent behavior can be expected. This causes several obstacles for the agent-based solutions to be easily accepted by the company management. The only way out seems to be a very thorough simulation of the agent-community behavior. From the authors’ own experience, the simulation detects just a limited number of patterns of emergent behavior. The “dangerous” patterns can be avoided by introducing appropriate policies across the system. Thus, the system simulation helps to understand the patterns of emergent behavior and their nature and to find protective measures, if necessary.

3. The current control systems offered by all the important vendors support the centralized control solutions only. The migration toward autonomous, independent controllers, communicating asynchronously (when needed) among themselves in the peer-to-peer way, seems to be the necessary technology enabler for a wider application of the agent-based solutions. Rockwell Automation, as a pioneering company in solving the migration process, is currently extending the classical PLC controller architecture to enable to run JAVA agents as well as a JAVA-based agent run-time environment directly within existing PLCs in parallel with the classical real-time scan-based control. Thus, the concept of holonic agents presented in Section 107.6 shifts from mainly academic considerations to the actual implementation.

4. Nearly the entire community of control engineers has been educated to design, run, and maintain strictly centralized solutions. This is quite a serious obstacle, as the engineers with the “classical” centralization-oriented approach (stressed in the last three decades under the CIM label) are really not ready and able to support the agent-based solutions. Much more educational efforts will be needed to overcome this serious hurdle.
5. Not all the tasks can be solved by the agent-based approach (the estimates talk about 30% of the control tasks and 60% of the diagnostic tasks to be suitable for application of the agent-based techniques). But certain areas with a higher degree of applicability of the agent-based technology have been identified already (see below). In general, applying the agent-based technology in inappropriate tasks can lead to frustration.

The areas suitable for application of the agent-based techniques are as follows:

1. **Transportation of material/material handling.** The transportation paths (conveyors, pipelines, AGVs, etc.) and their sensing and switching elements (diverters, crossings, storages, valves, tag readers, pressure sensors, etc.) can be easily represented by agents; their mutual communication can be defined and organized in a quite natural way. Interesting pioneering testbeds have been built. They document the viability and efficiency of this approach in the given category of tasks.

2. **Intelligent control of highly distributed systems,** namely in the chemical industry and in the area of utility distribution control (electrical energy, gas, waste water treatment, etc.). Many decisions can be made locally, in a very fast way; the communication among the autonomous unit is carried out only if really needed.

3. **Flexible manufacturing in automotive industry.** For this industry (aimed at mass production of individually customized products), very variable customization requirements, changes in the plans and schedules, changes in technology, as well as equipment failures seem to be quite obvious features of everyday operation. All these requirements and emergency situations can be easily handled by the agent technology.

4. **Complex military systems** (like aircraft and their groups, ships, army troops in the battlefield) can be modeled and managed as groups of agents. For instance, very high flexibility of the technical equipment on board a ship enables to accomplish at least a part of its mission if certain subsystems are destroyed or permanently out of operation.

The research in the field of agent-based control and diagnostic systems for manufacturing has been concentrated namely around the HMS consortium within the frame of the international initiative Intelligent Manufacturing Systems. Currently, there can be recognized several leading academic and industrial centers active in this field and bringing important results.

Let us mention the following academic sites: University of Cambridge, Center for Distributed Automation and Control (CDAC), Cambridge, U.K.; University of Calgary, Department of Mechanical Engineering, Canada; Katholieke University of Leuven, Department of PMA, Belgium; Vienna University of Technology, INFA Institute, Vienna, Austria; University of Hannover, IPA, Hannover, Germany; Czech Technical University, Gerstner Lab, Prague, Czech Republic.

Among the industrial leaders in agent-based control and diagnostics, following companies should be mentioned: Rockwell Automation, Milwaukee, WI; Rockwell Scientific comp., Thousand Oaks, CA; Daimler-Chrysler, Central Research Institute, Stuttgart, Germany; Toshiba + Fanuc, Japan; ProFactor, Steyr, Austria; SoftIng, Munich, Germany; CertiCon, a.s., Prague, Czech Republic; CSIRO, Melbourne, Australia.

The agent-based technology for manufacturing is developing in a very fast way. This development trend strictly follows the current trend in MAS research in the field of Artificial Intelligence as well as all the recommendations of the FIPA standardization consortium. But a long way remains in front of us: it is necessary, for example, (i) to change the way of thinking of industrial designers and engineers of control systems, (ii) to document the reliability and manageability of the emergent behavior of the agent-based systems (for this purpose, much more robust simulation tools should be developed), (iii) to support the migration processes from the centralized to agent-based control, which concern both the hardware and software, (iv) to solve the technical problems of interoperability, communication, and negotiation among the agents, and (v) to work toward widely acceptable ontology structures and languages.
References


<table>
<thead>
<tr>
<th>Query Number</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>AQ: 1</td>
<td>Please check Table 1 caption is ok?</td>
</tr>
<tr>
<td>AQ: 2</td>
<td>Please check Table 2 caption is ok?</td>
</tr>
<tr>
<td>AQ: 3</td>
<td>Please provide Editors</td>
</tr>
<tr>
<td>AQ: 4</td>
<td>Please provide Editors</td>
</tr>
<tr>
<td>AQ: 5</td>
<td>Please provide page numbers</td>
</tr>
<tr>
<td>AQ: 6</td>
<td>Please provide Editors</td>
</tr>
</tbody>
</table>