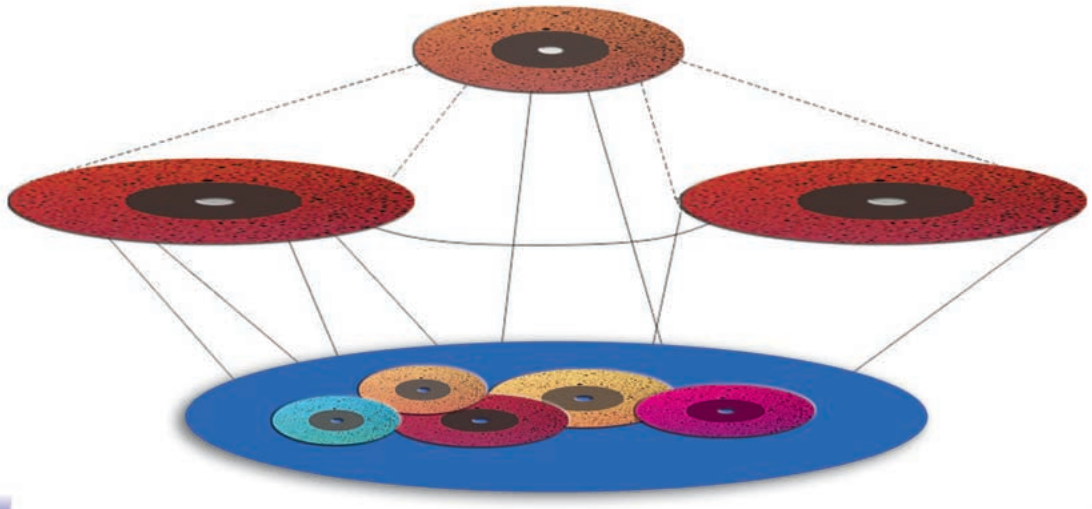
FEATURE

Agent-Based Control Systems

Why Are They Suited to Engineering Complex Systems?



By Nicholas R. Jennings and Stefan Bussmann

Modern control systems must meet increasingly demanding requirements stemming from the need to cope with significant degrees of uncertainty, as well as with more dynamic environments, and to provide greater flexibility. This, in turn, means that control systems software is highly complex in that it invariably has a large number of interacting parts [20]. This complexity requires that state-of-the-art software engineering methods and techniques be employed. In this article, we will argue that analyzing, designing, and implementing such complex software systems as a collection of interacting, autonomous, flexible components (i.e., as agents) affords software engineers several significant advantages over contemporary methods.

In seeking to demonstrate the efficacy of the *agent-oriented software engineering* approach [11], the most compelling argument would be to show quantitatively how its adoption improved the development process in a range of (control system) projects. Although several applications have been deployed (see [15] and [19] for overviews), such data are simply not available (as is the case for other contemporary software engineering approaches such as patterns, application frameworks, and

Jennings (nrj@ecs.soton.ac.uk) and Bussmann are with the Department of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, U.K. Bussmann is also with DaimlerChrysler AG, Alt-Moabit 96a, 10559 Berlin, Germany.

0272-1708/03/$17.00©2003IEEE

component-ware). Given this fact, the best that can be achieved is a qualitative justification for why agent-oriented approaches are well suited to engineering complex control systems. This general premise is then supported with two specific case studies in the domains of industrial process control (in particular, electricity transportation management) and manufacturing control (in particular, manufacturing line control), where the experiences of using an agent-based approach are assessed.

Before making the general case for agent-oriented software engineering, however, we first discuss the characteristics of complex software systems (of which control systems are naturally an instance). We then discuss the methods that software engineers have developed to help manage this complexity.

The role of any new software engineering paradigm is to provide structures and techniques that make complexity easier to handle. Fortunately for designers, this complexity exhibits several important regularities [20]. First, complexity frequently takes the form of a hierarchy; that is, a system composed of interrelated subsystems, each of which is in turn hierarchic in structure, until the lowest level of elementary subsystem is reached. The precise nature of these organizational relationships varies between subsystems; however, some generic forms (such as client-server, peer, etc.) can be identified. These relationships are not static; they often vary over time. Second, the choice of which components in the system are primitive is relatively arbitrary and is defined by the observer's aims and objectives. Third, hierarchical systems evolve more quickly than nonhierarchical ones of comparable size (i.e., complex systems will evolve from simple systems more rapidly if there are clearly identifiable stable intermediate forms than if there are not). Fourth, it is possible to distinguish between the interactions among subsystems and those within subsystems. The latter are both more frequent (typically by at least an order of magnitude) and more predictable than the former. This gives rise to the view that complex systems are nearly decomposable: subsystems can be treated almost as if they are independent, but not quite, since there are some interactions between them. Moreover, although many of these interactions can be predicted at design time, some cannot.

Drawing these insights together, a canonical view of a complex system (Figure 1) can be defined. The system's hierarchical nature is expressed through the "related to" links; components within a subsystem are connected through "frequent interaction" links, and interactions between components are expressed through "infrequent interaction" links.

Given these observations, software engineers have devised several fundamental tools of the trade to help manage this complexity [2], [3].

- *Decomposition:* The most basic technique for tackling large problems is to divide them into smaller, more manageable chunks, each of which can then be addressed in relative isolation (note the nearly decomposable subsystems in Figure 1). Decomposition helps tackle complexity because it limits the designer's scope.
- *Abstraction:* The process of defining a simplified model of the system that emphasizes some of the details or properties while suppressing others. Again, this works because it limits the designer's scope of interest at a given time.
- *Organization:* The process of defining and managing the interrelationships among the various problem-solving components (note the subsystem and interaction links of Figure 1). This covers phenomena such as inheritance (in object-oriented systems) and subroutines (in procedural languages). The ability to specify and enact organizational relationships helps designers tackle complexity by enabling a number of basic components to be grouped together and treated as a higher-level unit of analysis and by providing a means of describing the high-level relationships among various units.

Having characterized complex software systems and identified the fundamental software engineering approaches that help manage this complexity, the case for agent-oriented software engineering can now be made.

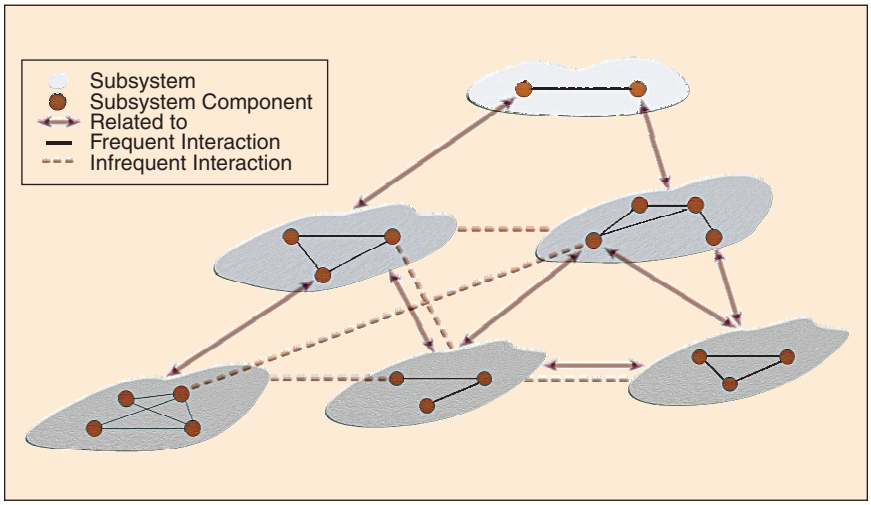


Figure 1. *View of a complex system.*

The Case for Agent-Oriented Software Engineering

The first step in arguing for an agent-oriented approach to software engineering involves identifying the key concepts of agent-based computing. The first such concept is that of an agent: "An agent is an encapsulated computer system that is situated in some environment and can act flexibly and autonomously in that environment to meet its design objectives" [22].

Several points about this definition require elaboration. Agents are i)

clearly identifiable problem-solving entities with well-defined boundaries and interfaces; ii) situated (embedded) in a particular environment over which they have partial control and observability—they receive inputs related to the state of their environment through sensors and they act on the environment through effectors; iii) designed to fulfill a specific role—they have particular objectives to achieve; iv) autonomous—they have control over both their internal state and their own behavior; v) capable of exhibiting flexible problem-solving behavior in pursuit of their design objectives, being both reactive (able to respond in a timely fashion to changes that occur in their environment) and proactive (able to opportunistically adopt goals and take the initiative) [24].

When adopting an agent-oriented view, it soon becomes apparent that most problems require or involve multiple agents: to represent the decentralized nature of the problem, the multiple loci of control, the multiple perspectives, or the competing interests [1]. Moreover, the agents will need to interact with one another, either to achieve their individual objectives or to manage the dependencies that ensue from being situated in a common environment. These interactions can vary from simple semantic interoperation (information passing), through traditional client-server-type interactions, to rich social interactions (the ability to cooperate [12], coordinate [10], and negotiate [13] about a course of action). Whatever the nature of the social process, however, two points qualitatively differentiate agent interactions from those that occur in other software engineering paradigms. First, agent-oriented interactions generally occur through a high-level (declarative) agent communication language (often based on speech act theory [17]). Consequently, interactions are conducted at the knowledge level [16]: in terms of which goals should be followed, at what time, and by whom (compare method invocation or function calls that operate at a purely syntactic level). Second, as agents are flexible problem solvers, operating in an environment over which they have only partial control and observability, interactions need to be handled in a similarly flexible manner. Thus, agents need the ability to make context-dependent decisions about the nature and scope of their interactions and to initiate (and respond to) interactions that were not foreseen at design time.

In the majority of cases, agents act either on behalf of individuals/companies or as part of some wider initiative. Thus, there is typically some underpinning organizational context to the agents' interactions. This context defines the nature of the relationship between the agents and the rules that must be adhered to during the interaction. For example, the agents may be peers working together in a team, or one may be the manager of the others, or the agents must negotiate according to the rules of a particular auction house. To capture such links, agent systems have explicit constructs for modeling organizational relationships (e.g., manager, team member, auctioneer). In many cases, these relationships are subject to ongoing change: social interaction means existing relationships evolve (e.g., a team of peers may elect a leader) and new relationships are created (e.g., a number of unrelated agents band together to deliver a service that no one individual can offer). The temporal extent of these relationships can also vary enormously: from providing a service as a one-off to a permanent bond. To cope with this variety and dynamism, agent researchers have devised protocols that enable organizational groupings to be formed and disbanded [6], specified mechanisms to ensure that groupings act together in a coherent fashion [12], [10], and developed structures to characterize the macro behavior of collectives [15], [24].

Drawing these points together (Figure 2), it can be seen that i) adopting an agent-oriented approach to software engineering means decomposing the problem into multiple autonomous components that can act and interact in flexible

Agents Versus Objects

Although there are certain similarities between object- and agent-oriented approaches (e.g., both adhere to the principle of information hiding and recognize the importance of interactions), there are also several important differences [22]. First, objects are generally passive in nature: they need to be sent a message before they become active. Second, although objects encapsulate state and behavior realization, they do not encapsulate behavior activation (action choice). Thus, any object can invoke any publicly accessible method on any other object. Once the method is invoked, the corresponding actions are performed. Third, object orientation fails to provide an adequate set of concepts and mechanisms for modeling complex systems; for such systems "we find that objects, classes and modules provide an essential yet insufficient means of abstraction" [2, p. 34]. Individual objects represent too fine a granularity of behavior and method invocation is too primitive a mechanism for describing the types of interactions that take place. Recognition of these facts led to the development of more powerful abstraction mechanisms such as design patterns, application frameworks, and component-ware. Although these are undoubtedly a step forward, they fall short of the desiderata for complex system developments. By their very nature, they focus on generic system functions, and the mandated patterns of interaction are rigid and predetermined. Finally, object-oriented approaches provide only minimal support for specifying and managing organizational relationships (basically, relationships are defined by static inheritance hierarchies).

ways to achieve their set objectives; ii) the key abstraction models that define the agent-oriented mindset are agents, interactions, and organizations; and iii) explicit structures and mechanisms are often used to describe and manage the complex and changing web of organizational relationships that exist among the agents.

From a control perspective, this view of software systems has several similarities to work on heterarchical systems in distributed control [8]. The work of [9], for example, avoids the drawbacks of hierarchical control by distributing the decision making into intelligent parts. However, the work on heterarchical control tends to concentrate on the distributed systems nature of these control systems and, to a certain extent, on the autonomy of the individual components, rather than on the flexible, high-level nature of the interactions and the explicit representation of the organizational context.

The Software Engineering Credentials of the Agent-Oriented Approach

Here the argument in favor of an agent-oriented approach to software engineering is composed of the following steps: i) show that agent-oriented decompositions are an effective way of partitioning the problem space of a complex system, ii) show that the key abstractions of the agent-oriented mindset are a natural means of modeling complex systems, and iii) show that the agent-oriented philosophy for modeling and managing organizational relationships is appropriate for dealing with the dependencies and interactions that exist in complex systems. When taken together, these steps form a complete mapping (cross product) between the characteristics of a complex system and the key software engineering abstractions for handling complexity as they apply to agent-based systems. Each step is now discussed in turn.

The Merits of Agent-Oriented Decompositions

Complex systems consist of a number of related subsystems organized in a hierarchical fashion (Figure 1). At any given level, subsystems work together to achieve the functionality of their parent system. Moreover, within a subsystem, the constituent components work together to deliver the overall functionality. Thus, the same basic model of interacting components, working together to achieve particular objectives, occurs throughout the system. Given this fact, it is entirely natural to modularize the components in terms of the objectives they achieve. (Indeed, the view that decompositions based on functions/actions/processes are more intuitive and easier to produce than those based on data/objects is even acknowledged within the object-oriented community (see [18, p. 44])). In other words, each component can be thought of as achieving one or more objectives. A second important observation is that complex systems have multiple loci of control: "real systems have no top" [18, p. 43]. Applying this philosophy to objective-achieving decompositions means the individual components should localize and encapsulate their own control. Thus, entities should have their own thread of control (i.e., they should be active), and they should have control over their own actions (i.e., they should be autonomous).

For the active and autonomous components to fulfill both their individual and collective objectives, they need to interact (recall that complex systems are only nearly decomposable). However, the system's inherent complexity means it is impossible to a priori know about all potential links: interactions will occur at unpredictable times, for unpredictable reasons, between unpredictable components. For this reason, it is futile to try to predict or analyze all the possibilities at design time. It is more realistic to endow the components with the ability to make decisions about the nature and scope of their interactions at runtime. From this, it follows that components need the ability to initiate (and respond to) interactions in a flexible manner.

The policy of deferring to runtime decisions about component interactions facilitates the engineering of complex systems in two ways. First, problems associated with the coupling of components are significantly reduced (by dealing with them in a flexible and declarative manner). Components are specifically designed to respond to unanticipated requests and can spontaneously generate requests for assistance if they find themselves in difficulty. Moreover, because these interactions are enacted through a high-level agent communication language, coupling becomes a knowledge-level issue. This removes syntactic concerns from the types of errors caused by unexpected interactions. Second, the problem of managing control relationships among the software components (a task that bedevils traditional objective-based decompositions) is significantly reduced. All agents are continuously active, and any coordination or synchronization that is required is handled bottom-up through interagent interaction.

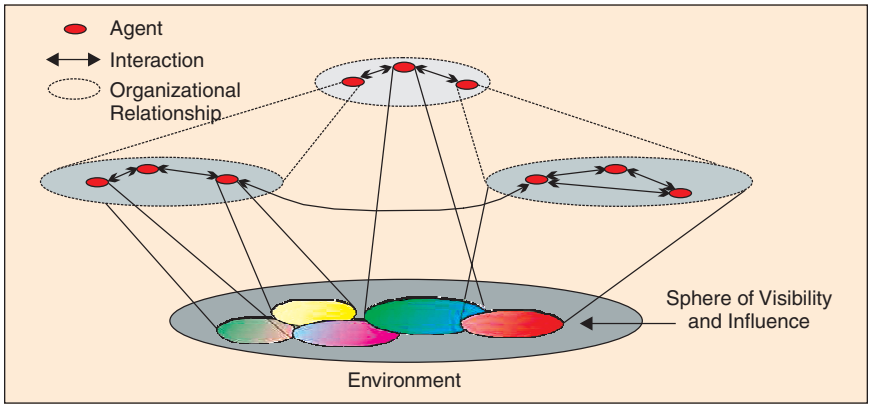


Figure 2. *Canonical view of an agent-based system.*

From this discussion, it is apparent that the natural way to modularize a complex system is in terms of multiple autonomous components that act and interact in flexible ways to achieve their objectives. Given this, the agent-oriented approach is simply the best fit.

The Suitability of Agent-Oriented Abstractions

A significant part of the design process is finding the right models for viewing the problem. Typically, there will be multiple candidates, and the difficult task is choosing the most appropriate one. When designing software, the most powerful abstractions are those that minimize the semantic gap between the units of analysis that are intuitively used to conceptualize the problem and the constructs present in the solution paradigm. In the case of complex systems, the problem to be characterized consists of subsystems, subsystem components, interactions, and organizational relationships.

- Subsystems naturally correspond to agent organizations. They involve several constituent components that act and interact according to their role within the larger enterprise.
- The case for viewing subsystem components as agents has been made above.
- The interplay between the subsystems and between their constituent components is most naturally viewed in terms of high-level social interactions: "In a complex system at any given level of abstraction, we find meaningful collections of objects that collaborate to achieve some higher level view" [2, p. 34]. This view accords precisely with the knowledge-level treatment of interaction afforded by the agent-oriented approach. Agent systems are invariably described in terms of "cooperating to achieve common objectives," "coordinating their actions," or "negotiating to resolve conflicts."
- Complex systems involve changing webs of relationships among their various components. They also require collections of components to be treated as a single conceptual unit when viewed from a different level of abstraction. Here again, the agent-oriented mindset provides suitable abstractions. A rich set of structures is available for explicitly representing organizational relationships [6]. Interaction protocols exist for forming new groupings and disbanding unwanted ones. Finally, structures are available for modeling collectives [12]. The latter point is especially useful in relation to representing subsystems, as they are nothing more than a team of components working together to achieve a collective goal.

The Need for Flexible Management of Changing Organizational Structures

Organizational constructs are first-class entities (in the programming language sense) in agent systems—explicit representations are made of organizational relationships and structures. Moreover, agent-oriented systems have the concomitant computational mechanisms for flexibly forming, maintaining, and disbanding organizations. This representational power enables agent systems to exploit two facets of the nature of complex systems. First, the notion of a primitive component can be varied according to the needs of the observer. Thus, at one level, entire subsystems can be viewed as singletons or collections of agents can be viewed as primitive components, and so on, until the system eventually bottoms out. Second, such structures provide the stable intermediate forms that are essential for the rapid development of complex systems. Their availability means that individual agents or organizational groupings can be developed in relative isolation and then added into the system in an incremental manner. This, in turn, ensures a smooth growth in functionality.

Will Agent-Oriented Techniques Be Widely Adopted?

Two key pragmatic issues will determine whether agent-oriented approaches catch on as a software engineering paradigm: the degree to which agents represent a radical departure from current software engineering thinking and the degree to which existing software can be integrated with agents.

Several trends become evident when examining the evolution of programming models. First, there has been an inexorable shift from languages whose conceptual basis is determined by the underlying machine architecture to languages whose key abstractions are rooted in the problem domain. Here the agent-oriented world view is perhaps the most natural way of characterizing many types of problems. Just as the real world is populated with objects that have operations performed on them, so it is equally full of active, purposeful agents that interact to achieve their objectives. Indeed, many object-oriented analyses start from precisely this perspective: "We view the world as a set of autonomous agents that collaborate to perform some higher level function" [2, p. 17]. Second, the basic building blocks of the programming models exhibit increasing degrees of localization and encapsulation [19]. Agents follow this trend by localizing purpose inside each agent, by giving each agent its own thread of control, and by encapsulating action selection. Third, ever richer mechanisms for promoting reuse are being provided. Here, the agent view also reaches new heights. Rather than stopping at reuse of subsystem components (design patterns and component-ware) and rigidly preordained interactions (application frameworks), agents enable whole subsystems and flexible interactions to be reused. In the former case, agent designs and implementations are reused within and between applications. Consider, for example, the class of agent architectures that have beliefs (what the agent knows), desires (what the agent wants), and intentions (what the agent is doing) at their core [23]. Such architectures have

been used in a wide variety of applications, including air traffic control, process control, fault diagnosis, and transportation [15], [19]. In the latter case, flexible patterns of interaction such as the contract net protocol [21] (an agent with a task to complete advertises this fact to others that it believes are capable of performing it; these agents may submit a bid to perform the task if they are interested, and the originator then delegates the task to the agent that makes the best bid) and various forms of resource-allocation auction (e.g., English, Dutch, first- and second-price sealed bid [25]) have been reused in numerous applications (see the manufacturing line control scenario). In short, agent-oriented techniques represent a natural progression of current software engineering thinking, and, for this reason, the main concepts and tenets of the approach should be readily acceptable to software engineering practitioners.

"An agent is an encapsulated computer system that is situated in some environment and can act flexibly and autonomously in that environment to meet its design objectives."

The second factor in favor of a widespread use of agents is that their adoption does not require a revolution in terms of an organization's existing software systems. Agent-oriented systems are evolutionary and incremental, as legacy (nonagent) software can be incorporated in a relatively straightforward manner (see the electricity transportation case study). The technique used is to place wrapping software around the legacy code to serve as an agent interface to the other software components. Thus, from the outside, the wrapper looks like any other agent; on the inside, it performs a two-way translation function: taking external requests from other agents and mapping them into calls in the legacy code and taking the legacy code's external requests and mapping them into the appropriate set of agent communication commands. This ability to wrap legacy systems means agents may initially be used as an integration technology. As new requirements are placed on the system, however, bespoke agents may be developed and added. This feature enables a complex system to grow in an evolutionary fashion (based on stable intermediate forms) while continually maintaining a working version of the system.

Case Studies

Having discussed the potential benefits of agent-based systems for complex systems in general, we now present two specific agent-based control system applications. The goal in presenting these case studies is twofold: i) to ground the abstract concepts of agent-based computing in specific application contexts and ii) to highlight the practical advantages that can accrue from an agent-based solution. Moreover, the scope and applicability of agent-based solutions is emphasized by discussing two examples that are at fundamentally different levels of the control spectrum: control of an entire network and control of an individual production line.

Electricity Transportation Management

This application was developed and deployed by the Spanish electric utility Iberdrola (more details of the underpinning agent technology can be found in [14], and more details of its application in this domain can be found in [7]). Generally, energy management is the process of monitoring and controlling the cycle of generating, transporting, and distributing electrical energy to industrial and domestic customers. Generation transforms raw energy (hydraulic, thermal, nuclear, and solar) into a more accessible form that then needs to be transported from its generation site to the consumer. To minimize losses during transportation, the electrical voltage is made high (132 kV or above) before it is placed on a transport network and sent over many hundreds of kilometers. Finally, the voltage is lowered and electricity is delivered to consumers using a distribution network involving many kilometers (all below 132 kV) spread over a much smaller area.

To ensure that the transportation network remains within the desired safety and economical constraints, it is equipped with a sophisticated data acquisition system (SCADA) and several conventional application programs that help the operator (a control engineer) analyze it (these programs are primarily designed for normal operating conditions). The network's operation is monitored from a dispatching control room (DCR), and whenever an unexpected event occurs, hundreds of alarms are automatically sent to it by the SCADA system. Under these circumstances, the operator must rely on experiential knowledge to analyze the information, diagnose the situation, and take appropriate remedial actions to return the network to a safe state. To reduce the operators' cognitive load in such circumstances and to help them make better decisions faster, Iberdrola first developed several (stand-alone) decision support systems (e.g., a real-time database that stores information about the state of the network and an alarm analysis expert system that diagnoses faults produced in the network based on the alarm messages received at the DCR). To improve this support, Iberdrola decided that these systems should interoperate to produce a coherent view and that new functionality should be added (to enable the control engineer to actually perform and dynamically monitor the service restoration process and also to exploit the new data sources, such as chronological information and faster rate snapshots, which became available as the SCADA system was im-

proved). What follows is a description of how these goals were achieved using agent technology.

Why Use Agent Techniques for This Application?

This application required the tried and tested decision support tools to be integrated and extended with new functionality. Two means of realizing this system upgrade strategy were considered: extend the existing systems to cover the new features or follow a distributed approach and allow the new functionality to be expressed as distinct computational entities that can interact with the existing systems through a common distribution platform. The second option was chosen because it was considered the most effective means of the following:

- *Permitting reasoning based on information of different granularity.* Two types of alarms, nonchronological and chronological, need to be dealt with. In the former case, the time stamped is coincident with the time of acquisition by the control system (consequently, it is conditioned by the control system's polling mechanism), whereas in the latter case, the time stamped is coincident with the actual occurrence of the event. As chronological alarms represent a more accurate picture of events in the network, they generally lead to a swifter diagnosis; however, they have the disadvantage that chronological information has a low priority in Iberdrola's communication channels. Thus, when the channels are saturated (as often happens during a disturbance), their time of arrival is unpredictable. For these reasons, a new alarm analysis expert system was built that utilized chronological information and could subsequently integrate its results with those of the preexisting system, rather than constructing a monolithic system that received both types of data and had to embody both types of diagnostic knowledge.

  A similar situation occurs when considering service restoration. Two types of information are relevant to this activity: snapshots (which provide a comprehensive picture of the current state of all the network's components) and alarm messages (which show how the state of the components has changed over time). The former can be produced relatively quickly and give a complete picture of the system's state, whereas the latter may take several minutes for a large disturbance but are needed to indicate the type of fault from which the system must be restored. Rather than trying to place both types of information and reasoning in a single system, it seemed more natural to develop a service restoration subsystem that dealt mainly with snapshots and received the necessary high-level information about the equipment at fault from a diagnosis subsystem (rather than trying to deal with the raw alarm messages themselves).
- *Allowing the inclusion of different network models within the same system.* Some of the problem solvers need to work on the SCADA model of the network, while others need the applications network model (a model that permits differential equations to be solved and takes into account the physical characteristics of all its components). Rather than trying to combine and harmonize these complex and disparate models at design time, it was decided that each subsystem should work on whichever model was most appropriate for its task. Then the various components can interact at runtime to resolve any inconsistencies that arise from their use of different network models.
- *Enabling the use of several different problem-solving paradigms.* The diverse range of activities that need to be performed means there is no universally best problem-solving paradigm; procedural techniques are required for algorithmic calculations such as connectivity (to know which component is connected to which other) and load-flow analysis (solution of the differential equations), whereas symbolic reasoning based on heuristic search is best for diagnosis. A distributed approach enabled each component to be encoded in the most appropriate method.
- *Meeting the application's performance criteria.* Transportation management is a time-critical application, and as many different types of information can be processed in parallel, with only a small synchronization overhead, the response time of the overall system can be improved through the use of several interconnected machines.

Having decided on a distributed approach, a choice had to be made between using more conventional distributed processing techniques or agent-based techniques. Here the latter was adopted for the following reasons:

- *Robustness:* As the subsystems have overlapping domains of expertise, the failure of one of them to produce an answer does not necessarily mean that no solution will be forthcoming (because one of the other systems may be able to produce at least a partial solution). However, to achieve this backup functionality flexibly, the different problem-solving components need to be intelligently coordinated in a context-sensitive manner (see the cooperative scenario for more details), a task beyond present-generation distributed processing systems.
- *Reliability:* The solutions of the overlapping systems can be cross referenced so as to present the operator with more reliable information. Again, however, this cross-referencing functionality needs to be properly managed according to the prevailing circumstances and thus requires dynamic and flexible reasoning to take place.
- *Natural representation of the domain:* An agent-based approach accurately represents the way the control engineers work when a large disturbance occurs. They perform specialized roles—one works toward

restoration, another tries to diagnose the problem based on different sources of information, and so on—and they communicate relevant information to one another to ensure they are following a coherent course of action toward the overall objective of restoring the service.

Specification of the Agents

During normal working conditions, management of the network by the operator in the DCR consists mainly of routine and simple tasks. However, during emergency situations, management becomes considerably more difficult because of the large number of constraints that must be taken into consideration and the insufficient quality of the information available to make these decisions. Emergency situations typically originate from a short circuit in a line, bus bar, or transformer. They can be exacerbated by equipment malfunctioning (e.g., a breaker failing to open) or subsequent overloads (a domino effect can cause one line to fail because of an overload, which in turn increases the load on neighboring lines so they become overloaded and subsequently fail, and so on). The situation can become even worse if power stations become disconnected, as this will cause an imbalance in the network's power. Consequently, actions to restore service must be taken rapidly and accurately so that what starts as a relatively minor problem does not escalate into a major disaster. In these circumstances, the actions the operator can perform consist mainly of breaker operations, topology changes, and activation/deactivation of automatisms and protective relays. For larger disturbances, however, actions on power plants may also be required. From this description of the control engineer's job, a top-down analysis determined that a comprehensive decision support system should cover the following activities:

- detect the existence of disturbances; sometimes the operation of protective relays and breakers can be caused by routine maintenance, and this should not be confused with genuine disturbances
- determine the cause, location, and type of the disturbance, including identifying if any equipment is permanently damaged
- analyze the situation of the network once it arrives at a steady state
- prepare a restoration plan to return the network to its original operational state.

Allying this top-down analysis with the bottom-up perspective of examining the extant systems, we decided to encapsulate the following preexisting systems as agents: the alarm analysis expert system and the interface to the control system. As discussed earlier, the availability of chronological alarm messages necessitated a new diagnosis system, which we decided to make available as an agent. Finally, it was always known that information about the initial area out of service (the blackout area) could help constrain the search for the faulty equipment; however, developing a dedicated stand-alone system for this purpose was never deemed cost-effective since the original alarm analysis expert system's performance was considered satisfactory (if somewhat slow). However, through the use of agent technology, much of the basic infrastructure to implement this functionality was now available from other agents, so developing a system capable of producing this information was considered economically viable.

In more detail, the operational system consists of seven agents running on five different machines (Figure 3). This figure shows a small portion of the Iberdrola network, which contains four substations (Sestao, Sodupe, Erandio, and Achuri). Each of these substations has a corresponding remote transmission unit (RTU) that sends information to the DCR in Bilbao about the status of its bus bars, breakers, and other electrical components. In the DCR, this information is collected by the front-end computer and made available to the cooperating agents through the control system's interface functions.

- *BRS (breakers and relays supervisor):* The new alarm analysis expert system detects the occurrence of a disturbance, determines the type of fault and its extent, generates an ordered list of fault hypotheses, validates hypotheses, and identifies malfunctioning equipment. To perform its analysis, it takes two types of inputs: chronological alarm messages and snapshots of the network that give the status of every breaker and switch.
- *AAA (preexisting, nonchronological alarm analysis agent expert system):* This agent pursues similar goals to the BRS, but the quality of information it receives is inferior to that of the BRS. Although the alarm messages received by both systems relate to the same physical operations, those received by the AAA represent ±5 s accuracy, whereas those received by the BRS are precise. This means that if the data are error free, then the BRS performs a better diagnosis than the AAA. However, if some of the chronological information is lost (a distinct possibility when the SCADA system is busy), the BRS may perform worse than the AAA. Therefore, whenever incomplete or erroneous information exists, which is true in most interesting cases, there is a need for cooperation between the two systems to make the overall system more robust and reliable (see below).
- *SRA (service restoration agent):* This agent devises a service restoration plan to return the network to a steady state after a blackout has occurred. To do so, it takes into account the constraints imposed by the damaged equipment as identified by the diagnosis agents.
- *UIA (user interface agent):* This agent implements the interface between the users and the community of agents. It allows the user to inspect the results produced by the diagnosis agents, display the alarms received, and browse through the log of analyzed disturbances. From the standpoint of restoration, the user can see the plan produced, modify it, run it in a

simulated environment to see its predicted effect, and request the development of a new restoration plan that takes into account the actions that are deemed pertinent. Through the use of a distributed windowing system, the UIA presents the appropriate information on the consoles of the various control engineers who are working on the system (Figure 3 shows two such control engineers, one working on restoration activities and one on diagnosis activities).

- *BAI (blackout area identifier):* When a fault occurs, the network's protective relays and breakers automatically try to isolate the minimum amount of equipment possible; in an ideal case, only the element at fault would be isolated. The BAI's objective is to identify which elements of the network are initially out of service, as the actual element at fault must be within this region. It uses nonchronological alarm messages as its information source and cooperates with the BRS and the AAA to increase the efficiency of the overall diagnosis process.
- *CSI (preexisting control system interface):* The CSI acts as the front end to the control system computers. Its objectives are to acquire and distribute network data to the other agents, to interface with the conventional management system application programs, and to monitor the restoration process to detect unexpected deviations. It is split into two physical agents: CSI-D, which detects the occurrence of disturbances and preprocesses the chronological and nonchronological alarm messages that are used by the AAA, BAI, and BRS agents; and CSI-R, which detects and corrects inconsistencies in the snapshot data file of the network, calculates the power flowing through it, and makes this information available to the SRA and the UIA.

This system design ensures that all the tasks identified by the top-down analysis are performed by at least one agent. Robustness is achieved by having multiple agents that are able to provide the same (or at least some) overlapping results. Efficiency is obtained by the parallel activation of tasks. Reliability is increased because even if one of the agents breaks down, the rest of the agents can often produce a result that, although not as good as the one provided by the complete system, is still of use to the operator.

Cooperative Diagnosis and Restoration

An important example of cooperation in this system involves the information interchange between the AAA, BRS, and BAI agents. The AAA and BRS produce the same result from different information sources, whereas BAI applies different knowledge to produce a result that should be coherent with that of the AAA and BRS.

Assume a block of nonchronological alarm messages has been provided by the SCADA system, and these alarm messages have been identified as related to a disturbance by the CSI. Using its model of the other agents, the CSI will realize the alarms are relevant to the AAA and BAI and will voluntarily send them out as unsolicited data. Some time later, the same process will be repeated, and the BRS will receive the corresponding chronological alarm messages. At this point, the AAA, BAI, and BRS are all operating in parallel.

When the AAA receives the alarm messages, it starts its diagnosis process, and a preliminary set of hypotheses is produced. During this time, the BAI would also have received the alarm messages and would have started trying to identify the initial blackout area. Again based on its models of the other agents, the BAI sends this information to the AAA. After a certain delay, the BRS agent starts working on the analysis of the chronological alarm messages. This will also result in a list of initial hypotheses being produced. The BRS checks whether any agents are interested in this information—again the AAA is noted and the hypotheses sent to it. The BRS then continues with its diagnosis to try and validate the cause of the fault.

After producing its tentative list of hypotheses, the AAA proceeds with a detailed analysis to try to ascertain the precise cause of the fault. The following situations may then occur: i) the initial blackout area is available to the AAA, triggering a refinement behavior that may reduce the number of hypotheses to be validated because the BAI has given a focused view of the situation; ii) the initial hypotheses pro-

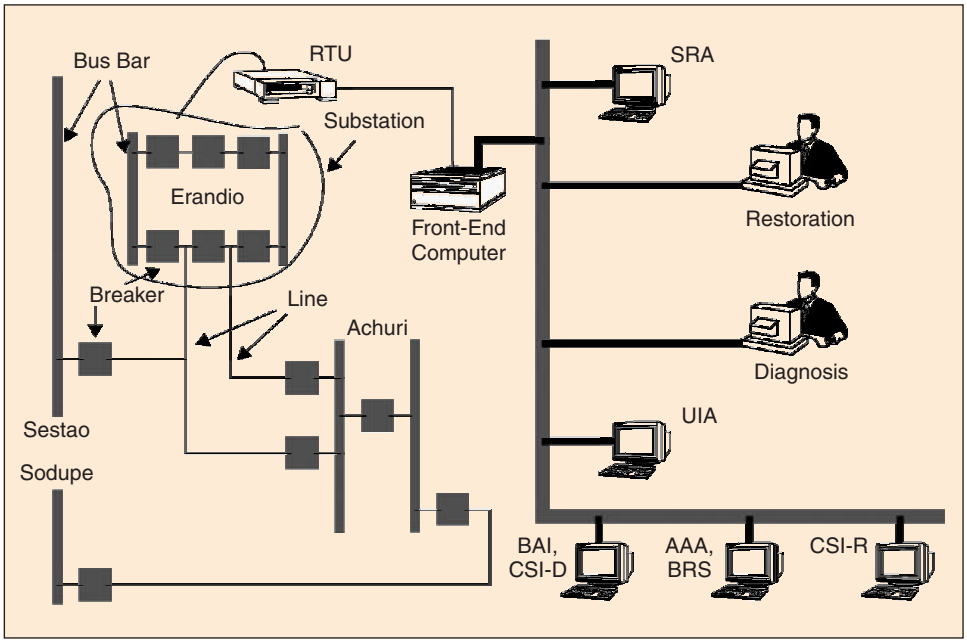


Figure 3. *The transport network and Iberdrola's agents.*

vided by the BRS are available to the AAA, triggering another refinement behavior and obtaining a better reordering of the hypotheses to be validated and a benefit in finding the element at fault; iii) the validated hypotheses provided by the BRS are available to the AAA, triggering yet another refinement behavior that has the same functionality as the previous one, but the reordering is based on validated hypotheses that are more accurate; iv) if no information is available from the BAI or BRS, the AAA proceeds with its hypothesis validation as a stand-alone agent. Therefore, if the other agents are down or are too slow to provide the information, the AAA will continue and find a faulty element, although its diagnosis will be less reliable and will take longer.

The restoration process is activated whenever a disturbance is detected. Once the disturbance is identified, the disturbance identifier is sent to the CSI-R, which acquires the snapshot of the network, corrects any inconsistencies that have arisen in its representation, and calculates the power flow solution of the current state. This information is then passed on to the SRA so that it can prepare for its restoration planning. The SRA waits until the diagnosis agents have informed it of the element suspected of being at fault and then proceeds to prepare a restoration plan. If, during this plan preparation, the SRA is informed that the equipment at fault is different from that originally indicated by either the AAA or the BRS, then it replans the restoration taking this information into account.

The UIA is the interface through which the user accesses the results produced by the agent community. During the diagnosis phase, the user is presented with both the tentative (early) list of suspected hypotheses and the final (validated) list. During the restoration phase, the UIA supports a more participatory interaction between the user and the teams of agents. The user is presented with the restoration plan and can then decide to modify it, run a detailed simulation to see the effects of the plan, or specify new constraints and ask for a new plan to be devised taking them into account.

Observations and Reflections

This application afforded several benefits. First, the agent system gives better results than its stand-alone counterparts because it takes into account multiple types of knowledge and data and then integrates them in a consistent manner. Second, the agent system is more robust because there are overlapping functionalities, meaning partial results can be produced in the case of component (agent) failure. Third, some results can be provided more quickly because cooperation provides a shortcut. Fourth, the functionalities of the different domain systems can be increased independently, which makes them easier to maintain (see, for example, the argument for developing the BAI and the general point about stable intermediate forms). Fifth, the control engineer is provided with an integrated view of the results of interest. Finally, the system has been designed to be open so that new agents can be added in an incremental manner.

One of the key features of this multiagent system is the way it handles fault diagnosis by using two different types of data (the nonchronological alarms used by the AAA and the chronological alarms used by the BRS) and two different points of view (the typical diagnosis approach of hypothesis generation and validation used by the AAA and BRS and the BAI's monitoring approach, which provides a high-level view of the status of the network). With this setup, the solution method that is best suited to the current situation can be dynamically selected. For example, if the BRS is operational but the AAA is not, the solution provided to the control engineer is the one created by the BRS; but if both the BRS and the AAA are running, the solution provided is the one that is mutually agreed on between them. Also, the fact that multiple agents are trying to generate the same results can be exploited to avoid repetition of certain tasks if deemed desirable in a particular context. For example, both the AAA and the BRS can provide initial hypotheses; consequently, if these hypotheses are provided by the BRS and made available to the AAA before it starts its own generation task, then this task need not be executed and the hypotheses provided by the BRS can be used instead. This ability to flexibly manage, at runtime, multiple sources of data and multiple problem-solving perspectives provides enormous robustness to the overall system because if one of the agents crashes, the others will still be able to provide some form of solution.

Manufacturing Line Control

This application was developed by a DaimlerChrysler-led industrial consortium and was deployed on one of DaimlerChrysler's production lines in Stuttgart, Germany (more details can be found in [5]). The overall aim of the system is to provide a flexible and robust system for controlling a manufacturing line. This process has several basic parts (workpieces) that have various operations performed on them by various machines.

The industry-standard approach to manufacturing control is to devise a global schedule, typically covering one day, for the entire manufacturing process. This indicates when the various parts should be released from their stores, which machines they should be routed through, and what operations should be performed at the various machines. The problem with this centralized and preplanned approach, however, is that plan formation is divorced from plan execution. Thus, the schedule can rarely be adhered to in practice: machines and operations fail (sometimes in ways that are difficult to predict), and operations take longer than expected. When such disturbances occur, the plant controller must either initiate a costly rescheduling exercise or use the out-of-date schedule as an approximate guide. Both of these options lead to inefficiencies in the manufacturing process and are therefore undesirable.

Why Use Agent Techniques for This Application?

To overcome the aforementioned problems, the control system must be made responsive to the prevailing situation of the manufacturing process. In this domain, a centralized controller is not a viable option—it would be too time consuming to construct and maintain an up-to-date representation of what is going on in the whole system, it would be a severe bottleneck in the system's performance, and it would represent a single point of failure (meaning the system would not be robust).

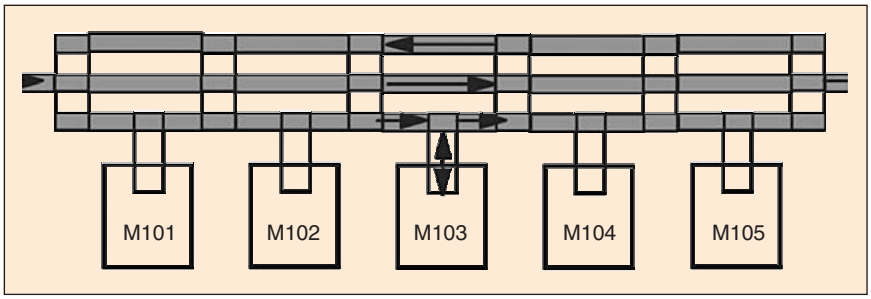


Figure 4. *A flexible manufacturing system.*

Given the requirements for decentralization, responsiveness, and flexible contingency handling, an agent- based approach was adopted. In this system, each manufactured part is represented by an autonomous agent that has the objective of getting itself to the end of the manufacturing line after a specified set of operations has been performed on it. Each machine is also represented by an agent. Such agents have the objective of maximizing their throughput, and they do this by deciding what parts will be accepted in what order and what operations will be performed at what time. Thus, for a given part to have an operation performed on it, its agent must negotiate with a machine agent capable of performing that operation. In short, resources are allocated dynamically on a just-in-time basis through a continuous coordination process among the relevant agents.

Specification of the Agents and Their Interactions

To achieve robustness and flexibility, the machines must have overlapping capacities. This means that there must always be more than one machine that can perform every manufacturing step. Thus, when machine breakdowns occur, the redundancy gives the system the flexibility of diverting the part to another machine. Diverting a part, however, is not possible without the ability to bypass a machine (or machines). To this end, the concept of a modular manufacturing system was developed (Figure 4) in which the entire manufacturing system is composed of standard modules. Each module consists of a machine, three one-way conveyors, and two transportation switches (Figure 5). Every switch can move a part from any of its entry points to any of its exits. In Figure 4, each of the intermediary switches has two entries and one exit on the left-hand side and two exits and only one entry on the right-hand side.

An arrangement of standard modules, as in Figure 4, means a part can either enter a machine through the lowermost conveyor or bypass the machine through the middle one. After bypassing a machine, a part has two options: it can either proceed in a forward direction to a subsequent machine or move backward using the topmost conveyor. If, for instance, the lowermost conveyor is already occupied, preventing a part from entering the target machine, then the part can move backward and forward in a circle until the lowermost conveyor is available again. In this way, the entire transportation system serves as a flexible buffer.

To control this flexible manufacturing system in a decentralized manner, a specific agent is associated with each workpiece, machine, and switch. A workpiece agent manages the state (which operations need to be performed and which operations have already been performed) of the workpiece attached. A machine agent controls the overall material flow through a machine, not just the work in progress. To this end, every machine agent manages what we call a virtual buffer. This buffer includes not only the machine's current work in progress, but also the outgoing flow of material; that is, all those workpieces that have already been processed by the machine but are not yet able to find an appropriate new machine. The switch agent controls a particular switch, deciding which entry to serve first and where to move a part.

All these agents constitute parallel processes. These processes, of course, are not independent; they have to be coordinated. Here coordination is achieved by a negotiation procedure, which also takes place simultaneously. A single workpiece negotiates with the machines about which of them should process it next. In particular, the workpiece auctions off its current due operations by inviting machines to bid for these tasks. Every machine bid includes information about the current state of the machine's virtual buffer. If a workpiece awards one or more operations to a specific machine, then getting to this machine becomes the next goal of the workpiece. The routing of a workpiece is also organized through a sequence of bilateral communications, in each case, between the workpiece and the next switch that the workpiece approaches. This continues until the workpiece eventually reaches its goal.

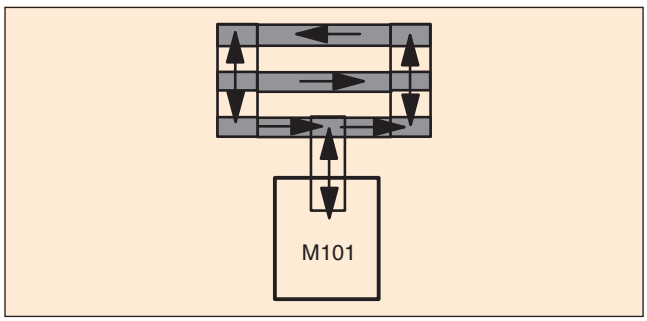


Figure 5. *A standard module.*

In more detail, the allocation of workpiece operations to machines is carried out by a first-price, sealed auction (the organizational structure). Each round involves three steps.

Step 1: The protocol is always initiated by a workpiece agent; in particular, whenever a workpiece first enters the manufacturing system and, thereafter, immediately after it leaves a machine. In any case, the workpiece determines its current task (next and subsequent operations to be performed on it) and all forward successors of the machine it has just left. The system is more efficient if parts move forward (left to right in Figure 4). However, this is not always possible (e.g., due to disturbances), and if workpieces do move backward, an assignment must be enforced to avoid deadlocks (see [5] for more details). The workpiece then sends an invitation to all these machines to bid for its current task.

Step 2: If a machine receives an invitation to bid for a current task, it checks whether it is able to perform it. If it can, it issues a bid; otherwise, it simply ignores the call. Short-term disturbances of some of the machine's operations are ignored here. This is because the subject of the negotiation is a future allocation of a subtask, and the current situation obviously does not tell us much about a machine's state when the workpiece enters the machine. The machine agent issues no bids without making sure that it is actually ready to accept a new workpiece; it therefore checks the capacity of its virtual buffer. If it does not have capacity, it does not answer the call. If the agent does make a bid, it includes the current size of its virtual buffer and the maximum number of the desired operations it is able to perform.

Step 3: The workpiece agent collects all the bids for a specific call. If there are no bids, it issues another invitation to bid, continuing with step 1; otherwise the workpiece selects the best bid. This selection is based on both components (a) and (b) of a bid, with (a) having a higher priority. In this case, the lower the current size of the virtual buffer, the better. The more operations the maximal subtask (b) contains, the better. The chosen machine agent is then informed, and it then includes the relevant workpiece in its (virtual) input buffer.

Once a workpiece agent has selected its next target machine, the workpiece must be moved to its new goal. In a layout like the one depicted in Figure 4, there is usually a vast number of different paths ultimately leading to the same goal. Of course, shorter paths are preferred, but even more important than optimizing the routing is the avoidance of any congestion (since this can have disastrous consequences on the overall system performance). In an unpredictable environment such as a manufacturing system, jams can only be avoided by strictly separating the actual routing from the goal itself. In this system, such dynamic routing is ensured through a sequence of bilateral communications, each time between the workpiece and the next switch it approaches. A switch always tries to move a workpiece directly to its goal, thus trying to optimize the routing. If an exit is not available, then an alternative route is taken. In this case, however, the priority of the workpiece is incremented. These priorities are used to decide which workpiece to prefer if a switch has more than one possibility: the workpiece with the highest priority is always served first so as to avoid indefinitely routing a workpiece along a cycle rather than to its actual goal.

Observations and Reflections

To evaluate the system, DaimlerChrysler conducted a series of simulations, all of which are based on authentic product types and cycle times. The disturbance characteristics have been taken from existing machines. A typical configuration consists of four blocks of identical machines. The number of machines in a block ranges from 5 to 11, with 36 machines in total. The simulations have shown that the agent-based mechanism is extremely robust against disturbances of machines as well as failures of control units. Moreover, its performance is nearly optimal, achieving about 99.7% of the theoretical optimum.

In addition to this simulation work, the control system has been installed as a bypass to an existing large-series manufacturing line for cylinder heads. The bypass, located in a plant in Stuttgart-Untertürkheim, Germany, is shown in Figure 6. The bypass has undergone a series of performance tests which showed that the results of the simulations are valid under real manufacturing conditions. Moreover, the system has now been in routine operation for more than two years and has confirmed its robustness in day-to-day production situations.

Figure 6. *DaimlerChrysler's prototype (BLEICHERT Osterburken, reprinted with permission).*

The success of the system, in terms of both increased throughput and greater robustness to failure, can be attributed to several points. First, representing the components and the machines as agents means the decision making is much more localized. It can therefore be more responsive to prevailing circumstances. If unexpected events occur, agents have the autonomy and proactiveness to try alternatives. Second, because the schedules are built up dynamically through flexible interactions, they can readily be altered in the event of delays or unexpected contingencies.

Conclusions

This article has sought to justify precisely why agent-oriented approaches are well suited to developing complex software systems in general and control systems in particular. These general points are then made concrete by showing how they apply in two very different agent-based control systems: Iberdrola's electricity transportation management system and DaimlerChrysler's manufacturing line control system. In making these arguments, proponents of other software engineering paradigms can claim that the key concepts of agent-oriented computing can be reproduced using their technique. This is undoubtedly true. Agent-oriented systems are, after all, computer programs, and all programs have the same set of computable functions. However, this misses the point. The value of a paradigm is the mindset and the techniques it provides to software engineers. In this respect, agent-oriented concepts and techniques are both well suited to developing complex, distributed systems and an extension of those currently available in other paradigms.

We believe that agent-based systems provide several advantages to the next generation of control systems. They provide a decentralized solution based on local decision making that gives the system a high degree of flexibility and robustness. The downside of devolving the decision making to autonomous components, however, is that it is correspondingly more difficult to predict overall system behavior. To this end, work is progressing on agent-oriented methodologies specifically for control applications [4]. In the systems described here, the agents are under the control of a single organization, which makes it easier to tailor their behavior so that desirable system properties emerge from their interplay. In the more general case of multiple organizations, however, producing predictable system-wide behavior is still an area of active research.

References

[1] A.H. Bond and L. Gasser, Eds., *Readings in Distributed Artificial Intelligence*. New York: Morgan Kaufmann, 1988.

[2] G. Booch, *Object-Oriented Analysis and Design with Applications*. Reading, MA: Addison Wesley, 1994.

[3] F.P. Brooks, *The Mythical Man-Month*. Reading, MA: Addison Wesley, 1995.

[4] S. Bussmann, N.R. Jennings, and M. Wooldridge, "Re-use of interaction protocols for decision-oriented applications," in *Proc. 3rd Int. Workshop Agent-Oriented Software Engineering*, Bologna, Italy, pp. 51-62.

[5] S. Bussmann and K. Schild, "Self-organising manufacturing control: An industrial application of agent technology," in *Proc. 4th Int. Conf. Multi-Agent Systems*, Boston, MA, 2000, pp. 87-94.

[6] K.M. Carley and L. Gasser, "Computational organization theory," in *Multiagent Systems*, G. Weiss, Ed. Cambridge, MA: MIT Press, 1999, pp. 229-330.

[7] J.M. Corera, I. Laresgoiti, and N.R. Jennings, "Using ARCHON, Part 2: Electricity transportation management," *IEEE Expert,* vol. 11, pp. 71-79, 1996.

[8] D.M. Dilts, N.P. Boyd, and H.H. Whorms, "The evolution of control architectures for automated manufacturing systems," *J. Manufact. Syst.*, vol. 10, no. 1, pp. 79-93, 1991.

[9] N. A. Duffie and R.S. Piper, "Non-hierarchical control of a flexible manufacturing cell," *Robot. Comput. Integrat. Manufact.*, vol. 3, no. 2, pp. 175-179, 1987.

[10] E.H. Durfee, "Practically coordinating," *AI Mag.*, vol. 20, no. 1, pp. 99-116, 1999.

[11] N.R. Jennings, "On agent-based software engineering," *Artific. Intell.*, vol. 117, no. 2, pp. 277-296, 2000.

[12] N.R. Jennings, "Controlling cooperative problem solving in industrial multi-agent systems using joint intentions," *Artif. Intell.*, vol. 75, no. 2, pp. 195-240, 1995.

[13] N.R. Jennings, P. Faratin, A.R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge, "Automated negotiation: Prospects, methods and challenges," *Int. J. Group Decision and Negotiation* , vol. 10, no. 2, pp. 199-215, 2001.

[14] N.R. Jennings, E. Mamdani, J. Corera, I. Laresgoiti, F. Perriolat, P. Skarek, and L.Z. Varga, "Using ARCHON to develop real-world DAI applications Part 1," *IEEE Expert* , vol. 11, pp. 64-70, 1996.

[15] N.R. Jennings and M. Wooldridge, Eds., *Agent Technology: Foundations, Applications and Markets*. New York: Springer-Verlag, 1998.

[16] A. Newell, "The knowledge level," *Artif. Intell.*, vol. 18, pp. 87-127, 1982.

[17] J. Mayfield, Y. Labrou, and T. Finin, "Evaluating KQML as an agent communication language," in *Intelligent Agents II,* M. Wooldridge, J.P. Müller, and M. Tambe, Eds. New York: Springer, 1995, pp. 347-360.

[18] B. Meyer, *Object-Oriented Software Construction*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[19] H.V.D. Parunak, "Industrial and practical applications of distributed AI," in *Multi-Agent Systems*, G. Weiss, Ed. Cambridge, MA: MIT Press, 1999, pp. 377-421.

[20] H.A. Simon, *The Sciences of the Artificial*. Cambridge, MA: MIT Press, 1996.

[21] R.G. Smith and R. Davis, "Frameworks for cooperation in distributed problem solving," *IEEE Trans. Syst., Man, Cybern.*, vol. 11, pp. 61-70, 1981.

[22] M. Wooldridge, "Agent-based software engineering," *Proc. Inst. Elec. Eng.*, vol. 144, pp. 26-37, 1997.

[23] M. Wooldridge, *Reasoning About Rational Agents*. Cambridge, MA: MIT Press, 2000.

[24] M. Wooldridge and N.R. Jennings, "Intelligent agents: Theory and practice," *Knowledge Eng. Rev.*, vol. 10, no. 2, pp. 115-152, 1995.

[25] P.R. Wurman, "Dynamic pricing in the virtual marketplace," *IEEE Internet Computing*, vol. 5, pp. 36-42, Mar./Apr. 2001.

Nicholas R. Jennings is a professor of computer science in the Department of Electronics and Computer Science at Southampton University, where he carries out basic and applied research in agent-based computing. He has published some 150 articles on various facets of agent-based computing. He was the recipient of the Computers and Thought Award in 1999 for his contributions to multi-agent systems, the recipient of an IEE Achievement Medal in 2000 for his work on agent-based computing, and the recipient of the ACM Autonomous Agents Research award in 2003.

Stefan Bussmann studied computer science at the University of Kaiserslautern, Germany, and at the Institut National Polytechnique de Grenoble, France. Since 1993 he has been with DaimlerChrysler Research and Technology in Berlin, where he is working as a senior scientist on innovative production planning and control methods. He is also a part-time Ph.D. student at the University of Southampton.