Agents in Overalls: Experiences and Issues in the Development and Deployment of Industrial Agent-Based Systems[1]

H. Van Dyke Parunak
ERIM Center for Electronic Commerce
PO Box 134001
Ann Arbor, MI 48113-4001 USA
+1 734 623 2509
vparunak@erim.org

Abstract

Agent technologies have steadily matured in moving from the research laboratory to industrial application over the last ten years. Numerous systems have been deployed or are under advanced development with strong industrial support. These systems suggest important lessons for both industry and researchers. From an industrial perspective, these examples reflect trends in both business and technology that make agents an increasingly attractive commercial tool. From a research perspective, these examples identify important gaps in agent technology that merit the attention of academicians.

1. Introduction

Academic research in multi-agent systems (or any other technology) and industrial application are in perpetual tension. The researcher focuses on a particular technology (e.g., communication, planning, learning), and seeks practical problems to demonstrate its usefulness (and justify further funding). The industrial practitioner has a problem to solve, and cares much more about the speed and cost-effectiveness of the solution than about its elegance or sophistication. A capable researcher specializing in a specific technology can guarantee the relevance of that technology to any problem at hand. The practitioner has no *a priori* commitment to a particular technology, but stubbornly insists on a problem statement that often inconveniences the researchers.

At ERIM's Center for Electronic Commerce (CEC), we have been in the middle of this tension for the past fifteen years. A review of some applications (both ours and others') that have matured from the laboratory to the workplace holds lessons for both communities. These applications demonstrate in a concrete way the peculiar aptitude of agents for some of the pressing commercial problems faced by the industrial community, and industrial delegates will find these applications important case studies to help them evaluate the suitability of agents for their own problems. Conversely, the difficulties faced in developing these systems highlight knotty questions that can challenge the most innovative researcher, and in addition can attract industrial funding.

Section 2 of this paper offers three definitions that are useful in bridging the gap between industry and the academy: a definition of agenthood, a taxonomy of application types, and a scale on which the maturity of an agent application may be assessed. Section 3 discusses several application areas in which agent technologies have been successfully deployed. Sections 4 and 5

[1] This paper is an expanded version of an invited talk at PAAM'99. Its insights reflect the collective experience of the ERIM CEC team, including Mark Brown, Mitch Fleischer, Olga Gilmore, Jorge Goic, Jim Kindrick, Bob Matthews, Murali Nandula, John Sauter, Jon Schneider, Tee Toth-Fejel, Ray VanderBok, and Jack White.

summarize some of the lessons that these examples offer to industrialists and researchers, respectively.

2. What is a "Mature Agent Application"?

A clear discussion of mature agent applications requires that we understand what counts as an agent, what sorts of applications can be addressed, and what constitutes maturity in the application world. These questions address important business concerns. The definition of agenthood responds to apprehensions about the intrinsic risk of the technology itself. The taxonomy of existing applications demonstrates the applicability of the technology to recognized industrial problems. Attention to the maturity of current applications helps users assess how much work remains to implement agents for their applications.

2.1 What is an Agent?

A sure way to get disagreement among agent researchers is to suggest a definition of what constitutes an agent. The definition promoted by ERIM CEC does not claim to represent a research consensus. However, it has proven useful in communicating some of the main features of agent research to industrialists in a way that has encouraged them to explore these technologies further. Business people are by nature conservative, and we have found them more likely to consider agent-based solutions if they can see them as an incremental step that builds on proven earlier technology. Table 1 portrays agents as the natural next step in a software history of increasing localization and encapsulation.

	Monolithic Program	Structured Programming	Object-Oriented Programming	Agent-Oriented Programming
How does a unit behave? (Code)	External	Local	Local	Local
What does a unit do when it runs? (State)	External	External	Local	Local
When does a unit run?	External	External (called)	External (message)	Local (rules; goals)

Table 1: Software Evolution.—The history of software is one of increasing modularization and localization.

Originally, the basic unit of software was the complete program. Arbitrary jumps of control made it difficult to manipulate any unit other than the individual line of code and the entire program. Data often occupied the same deck of cards as the program, and the entire deck, code and program, was the responsibility of the programmer, who thus determined the behavior of the complete program before it began execution.

The "structured programming" movement designed programs from smaller packages of code, such as structured loops and subroutines, with a high degree of local integrity. Though a subroutine's code was encapsulated, its state had to be supplied externally through arguments, and it gained control only when externally invoked by a call.

The next generation was object-oriented programming, which localized not only a segment of code but also the variables manipulated by that code. Originally, objects were passive, and gained control only when some external entity sent them a message.

Software agents take the logical next step of giving each object its own thread of control and its own internal goals, thus localizing not only code and data, but also invocation. From this perspective, an agent is "an active object with initiative," or more succinctly, a "pro-active object." Similar definitions of the same flavor are "bounded processes" or "objects that can say 'no'" (due to Jim Odell). Our emphasis on an agent's initiative suggests the moniker, "objects that can say 'go'."

This view of agents is minimalistic. It omits many characteristics that one researcher or another might insist are essential for agency (such as a mentalistic model of internal state, or the ability to move over a network, or the function of representing a human). Still, we have found it useful in helping users to understand how agents fit into the larger software picture. Once they have made this step, it is a relatively simple matter to explain to them why one or another accessory is needed for the agents that will address their particular application.

Our advocacy of a minimalist view reflects our orientation to deploying agents in real-world situations. A researcher's success in attracting federal research funding depends on making the case that the ideas to be developed are as advanced and unprecedented as possible. This stance is counterproductive in persuading naturally conservative business people to adopt agents. Such an audience is highly averse to technical risk, and is much more eager to adapt new ideas if those ideas can be shown to be incremental extensions of technology that is already known and proven in the business environment. Object-oriented programming is well understood and widely accepted, and we have found business users receptive to agents as the next logical step in software evolution.

2.2 What kinds of Applications have been addressed?

Agents have been applied in three broad classes of environments: digital, social, and electromechanical. These domains differ in the challenges they pose to system robustness and in the impact of time on system operation.

Problem domains that deal with purely *digital environments* include telecommunications and static optimization problems. These are the simplest problems for agents, since agents are themselves digital creatures, and digital environments do not require them to engage a foreign culture. Robustness in these domains can often be provided by the internal structure of the system. For example, agents need deal only with a symbolic representation of the problem. This representation can be engineered for their convenience, and agents can assume its completeness. Another example of internal robustness is the maturity of the support services offered by digital environments. For instance, agents in such environments typically interact through existing communication services that take care of error checking and retransmission transparently to the agents themselves. Temporally, purely digital applications are the least constrained. Time is defined only by the clock speed of the processors on which the agents and their environment execute, and can be accelerated by buying a faster processor.

Social environments for agents support problem domains such as computer-supported collaborative work (CSCW), electronic commerce, and information retrieval. In these domains, silicon-based agents must interact with the non-digital world in the form of carbon-based humans.

The internal robustness available in digital domains does not apply in social environments. For example, the symbolic representation of a problem that an agent manipulates may differ from the real issues in the social system with which the agent is interacting, leading to the need for agents to be more actively involved in recovering from misunderstandings. However, these systems offer another source of robustness, external to the agents and their computational environment. People are intrinsically robust, and can adapt their behaviors to the capabilities and limitations of their tools. Thus if an agent-based system offers significant benefits, its users are likely to compensate for its shortcomings by modifying their own behavior accordingly. Temporally, social environments impose external deadlines on agent execution. Again, however, these deadlines can be negotiated between the silicon and carbon portions of the system.

Electromechanical environments require agents to manipulate the non-intelligent physical world, and include problem domains such as robotics, factories, and intelligent highway systems. These environments offer the least support for overall system robustness. They are dominated not by adaptable humans, but by Murphy's Law, which asserts that if something can go wrong, it will, and at the most inopportune time. Thus these environments impose the greatest demands on agents in terms of error recovery and contingency management. Temporally, these environments are also the most demanding. Time is not defined internally (by the speed of the agents' own processors), nor by a negotiated convention between carbon and silicon agents, but by the laws of physics in the external world. Agents that cannot keep up with these real-time requirements have no chance of success.

These three environments define not only the kinds of problems to which agents have been applied, but also the kinds of interfaces that an agent-based application must support. Because agents are deployed by people to solve problems in which people are interested, all three classes of system must support a social interface with humans. All agents require digital interfaces to communicate with each other, but these interfaces are taken for granted in social and electromechanical problem domains. Interfaces between agents and the physical world are of interest only in electromechanical domains. Figure 1 shows the overall taxonomy of agent applications that results from considering both the problem domain that the agents manipulate and the dominant interfaces that they support.

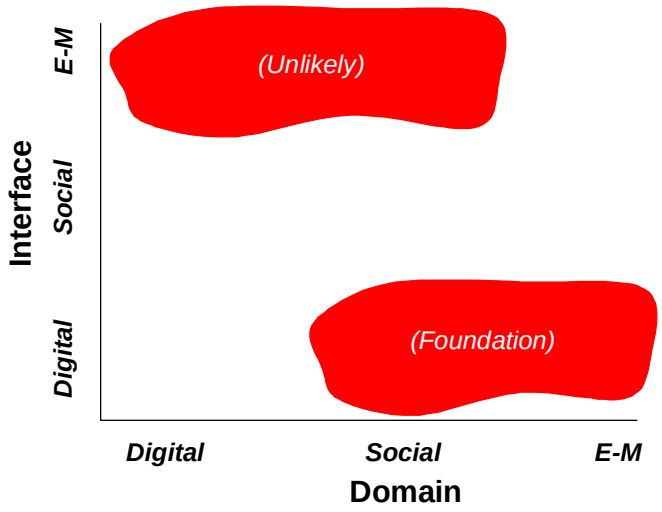

Figure 1: The World of Agent Applications.—Applications can be distinguished by the domains that they manipulate and the interfaces that they must support.

2.3 What constitutes Maturity?

Few business users want to be the first to test a new technology in real applications. But once a technology has shown its mettle, everyone wants to be second. In tracking the state of deployment of agent technology, we have found it useful to rank various applications on a scale that distinguishes them by maturity, summarized in Figure 2.

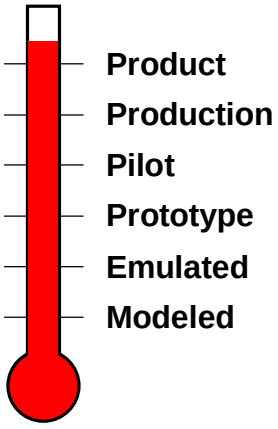

Figure 2: Degrees of Maturity.—The maturity of agent applications ranges from those that have only been modeled to those that are offered as products.

Modeled applications, the least mature, are those that exist only as architectural descriptions or theoretical analyses. It is rare for an industrial user to adopt a technology that is available only at this level.

Emulated applications have been demonstrated in a simulated environment. When the application domain is electromechanical (e.g., manufacturing control), an emulated application is relatively immature. However, when the application domain is digital (e.g., telecommunications routing),

emulation may be very close to the actual deployment environment, and may be adequate to encourage commercial adoption.

A **prototype** application has been demonstrated on real domain "hardware," but under laboratory conditions. In the case of social applications, this "hardware" includes people conducting tasks of the sort that the system is designed to support.

A **pilot** application has been demonstrated in a real commercial environment (as opposed to a laboratory), but in a limited portion of the enterprise where some degree of experimentation and failure can be tolerated. Such pilot tests are often a critical step in building a business case for broader deployment.

A **production** application is being used in daily commercial activities that are critical to the company's success. It has earned the trust of management, and is a part of the routine operations of the firm.

Some early production applications are developed in close conjunction with the user and require a high level of support for their continued operation. A **product** is an application that is robust enough to be offered for sale to multiple users without imposing a crushing support burden on its supplier. Truly widespread deployment of agent technology requires applications at the product level, so that users can treat them as commodities with known acquisition and operation costs and vendors who can be held accountable for their functionality.

3. Examples

Agent technology has matured to the point that we can identify applications across the full range of problem domains and maturity levels. Figure 3 classifies five such applications, from a much larger population that could be cited, in terms both of domain and of maturity.

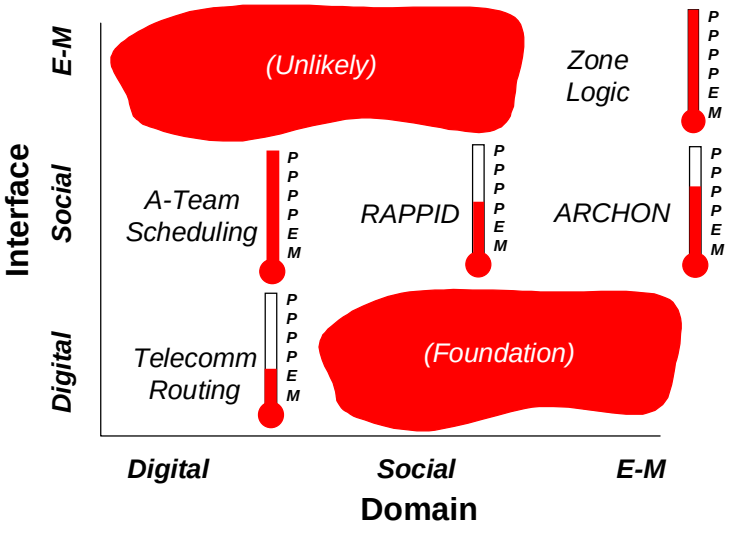

Figure 3: Some Sample Applications.—Agent-based systems exist at all maturity levels and in every application domain.

3.1 Digital Domain, Digital Interface

A particularly promising application of agents in a digital environment with predominantly digital interfaces is dynamic routing in telecommunications networks. In a packet-switched network, packets of information move from one node of the network to another. A typical network offers many alternative routes from a source to a destination, and the alternatives may differ in their capacity, length, and degree of congestion.

Conventionally, each node in the network has a routing table that it uses to direct visiting packets along the next step in their path. For each possible destination, the table recommends a node adjacent to the current node that a packet should visit next. These recommendations are static. They can take account of path lengths and static node capacities, but do not reflect moment-by-moment congestion in the network.

Several researchers are exploring the use of synthetic pheromones to enable packet agents and node agents to maintain routing tables dynamically [2, 8]. Though only at the emulation level, these efforts are typically conducted in close collaboration with telecommunication companies and use emulation platforms that offer a high degree of real-world fidelity, leading to the prospect of a relatively short adoption path.

Instead of recommending a single next step for each possible destination, the row of the table corresponding to a single destination records a pheromone level for each neighbor of the node, estimating the distance to the destination through that neighbor. These pheromones are reinforced by the packets that visit the node. For example, if packet transit times are symmetrical in both directions over network links, a packet originating at node *i* and arriving at node *j* from node *k* can update *j*'s estimate of the distance to *i* via *k* from the packet's memory of how long it has been en route from its origin.

Packets select their next step using a roulette wheel with one segment for each of the node's neighbors. The segment widths are set inversely to the distance pheromones recorded for the desired destination, so that neighbors lying on shorter routes to the destination are more likely to be selected. The random element in the selection ensures that the system explores alternative routes so that it can discover environmental changes and incorporate them in the pheromone levels that guide operational choices.

These telecommunications applications are a specific instance of a broader class of optimization by means of fine-grained agents [1, 5] that are beginning to find application in domains such as factory scheduling and routing of airline freight shipments [6].

3.2 Digital Domain, Social Interface

IBM markets an agent-based scheduling system for process industries such as paper and steel manufacturing, based on the A-Team architecture [15, 18, 27]. In this architecture, a team of functional agents generate, evaluate, or modify candidate solutions that reside on one or more blackboards, or "pools." One agent's task may be to generate new solutions and add them to the pool. Another class of agents test solutions against various evaluation criteria. Yet another class applies various optimization procedures to improve existing solutions. A "destroyer" agent removes old, low-ranking solutions from the pool to make room for new candidates. As a static off-line optimization procedure, this application belongs to the digital domain. However, humans can participate in the role of any of the agents. They can nominate initial solutions, apply evaluation criteria that would be difficult to automate, modify solutions to take account of

requirements that are hard to achieve through automated optimization, and veto solutions that violate management requirements. The agent structure of the system permits humans to participate symbiotically in complex schedule optimization tasks.

3.3 Social Domain, Social Interface

Social domains with social interfaces are perhaps the most common domain for commercial agent application today. For instance, know-bots routinely traverse multiple web sites to collect information, such as the best price for a given commodity or the site most likely to be relevant to a given topic [3]. Most people who have used the web for these functions have been assisted by real-world agents of this type, often without knowing it.

RAPPID (Responsible Agents for Product-Process Integrated Design) [24] illustrates how agents can help human designers coordinate their work more effectively. This system, developed at ERIM CEC, has been prototyped with human designers on problems including simple mechanical transmission systems, an armored personnel carrier, and a container ship.

Figure 4 illustrates how different people are responsible for the components and subsystems that make up a product. Conflicts arise when different teams disagree on the relation between the characteristics of their own functional pieces and the characteristics of the entire product. Some conflicts are within the design team: How much of a mechanism's total power budget should be available to the sensor circuitry, and how much to the actuator? Other conflicts set design against other manufacturing functions: How should one balance the functional desirability of an unusual machined shape against the increased manufacturing expense of creating that shape?

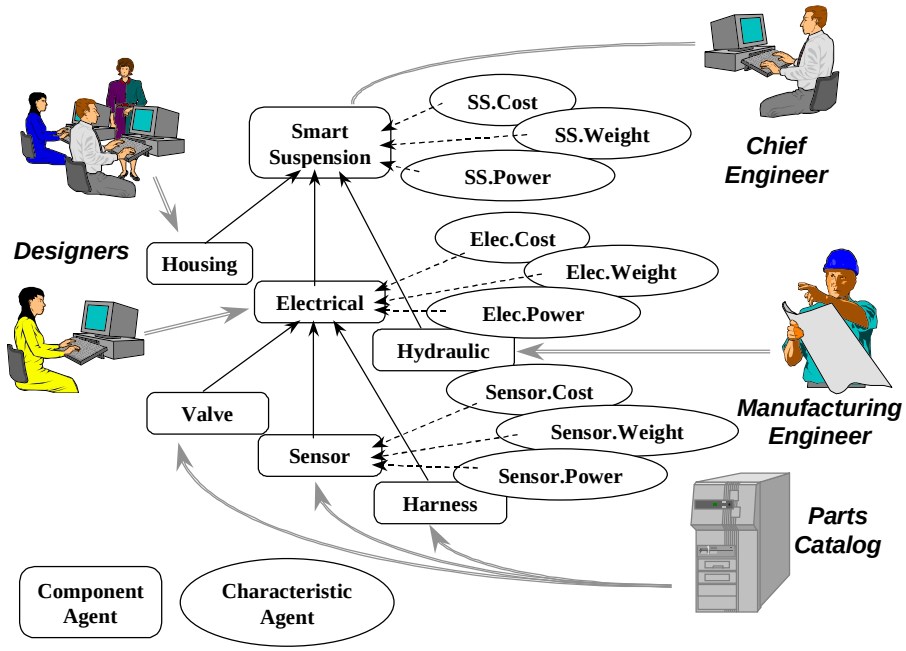

Figure 4: The RAPPID Ecosystem.—Both components and characteristics are agents.

It is easy to represent how much a mechanism weighs or how much power it consumes, but there is seldom a disciplined way to trade off (say) weight and power consumption against one another. The more characteristics are involved in a design compromise, the more difficult the trade-off becomes. The problem is the classic dilemma of multivariate optimization. Analytical solutions are

available only in specialized and limited niches. In current practice such trade-offs are sometimes supported by processes such as Quality Functional Deployment [7] or resolved politically, rather than in a way that optimizes the overall design and its manufacturability. The problem is compounded when design teams are distributed across different companies.

RAPPID explores two innovative techniques for coordinating the actions of different agents (designers): market dynamics and set-based reasoning [23, 25, 26].

In RAPPID, designers buy and sell the various characteristics of a design. Each characteristic agent is a computerized agent that maintains a marketplace in that characteristic. In the current implementation, the agents representing components are interfaces for human designers, who bid in these markets to buy and sell units of the characteristics. A component that needs more latitude in a given characteristic (say, more weight) can purchase increments of that characteristic from another component, but may need to sell another characteristic to raise resources for this purchase. In some cases, analytical models of the dependencies between characteristics help designers estimate their relative costs, but even where such models are clumsy or nonexistent, prices set in the marketplace define the coupling among characteristics.

To drive the design process toward convergence, RAPPID uses set-based reasoning. Most design in industry today follows a point-based approach, in which the participating designers repeatedly propose specific solutions to their component or subsystem. The chief engineer is expected to envision the final product at the outset, specifying to the designers what volume in design space it should occupy and challenging them to fit something into that space. Inevitably, as illustrated in Figure 5, some of the chief engineer's assumptions turn out to be wrong, requiring designers to reconsider previous decisions and compromise the original vision. This approach is analogous to constraint optimization by backtracking. Because mechanisms for disciplined backtracking are not well developed in design methodology, this approach usually terminates through fatigue or the arrival of a critical market deadline, rather than through convergence to an optimal solution.

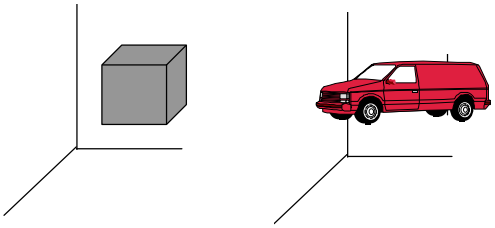

Figure 5: Point-Based Design.—If the chief engineer guesses wrong, designers have to backtrack.

Toyota has pioneered another approach, set-based design [30]. In this approach, illustrated in Figure 6, the chief engineer's task is not to guess the product's location in design space, but to guide the design team in a process of progressively shrinking the design space until it collapses around the product. Each designer shrinks the space of options for one component in concert with the other members of the team, all the while communicating about their common dependencies. This approach directly reflects consistency algorithms for solving constraint problems. If the communications among team members are managed appropriately, the shrinking design space drives the team to convergence.

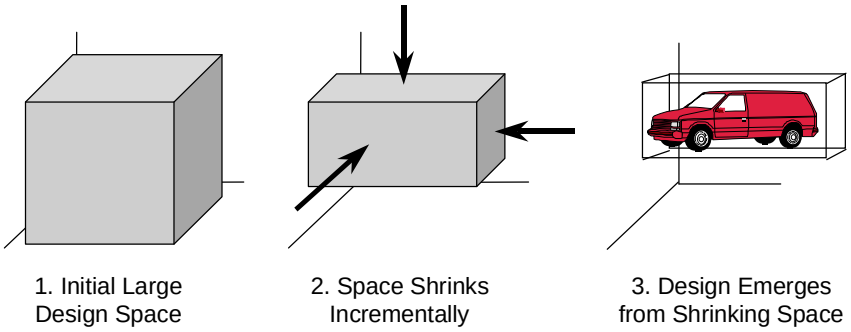

Figure 6: Set-Based Design.—The chief engineer guides the design team in shrinking the design space until it converges on a solution.

3.4 Electromechanical Domain, Social Interface

The problems of interfacing digital agents with real-world sensors and actuators appear to be a "small matter of engineering," but have broken the back of more than one project in the electromechanical domain. A common strategy is to use agents to drive a human interface, and let people make the actual connection. This approach is taken in the ADS-based Shinkansen operator interface [10, 16, 17] and the application of ARCHON to power systems management [12, 13, 31].

ARCHON has been demonstrated in numerous domains, including robotics, cement factory control, and particle accelerator control, but its most mature deployment is in the management of an electrical power grid, for which it was piloted in Iberdrola. Through the years, this utility had developed a series of expert systems and other automated aids to support operators in the control room:

- A control systems interface gave operators access to information from the underlying electromechanical systems.
- An alarms analysis and diagnosis expert system helped operators understand the possible causes of various alarm signals they might receive, while a breakers and relays supervisor interpreted the more precise but slower chronological reports on malfunctions.
- A blackout area identifier enabled determining the set of customers actually affected by a particular system failure.
- A service restoration system was responsible for planning the sequence in which failed systems should be brought back on line to ensure a smooth restoration of service.

As this panoply of support systems grew, operators frequently had to transfer information manually from the output of one system to the input of another. ARCHON wraps each of these systems in an agent shell that provides a common interaction language, maintains local knowedge about the capabilities of its own system and its acquaintances, and reasons about where to send results and requests for assistance. The management of a power grid is a time-critical application in the physical world, and ARCHON's success required it to meet these constraints. However, its interface to that physical world was mediated through human operators. This mediation permits human oversight for safety-critical applications, and also simplifies the deployment problems that arise in interfacing an agent system directly to physical control hardware.

3.5 Electromechanical Domain, Electromechanical Interface

In spite of the difficulties of dealing directly with the physical world, the commercial benefits are great, and numerous examples can be cited, including the AMROSE system for articulated robot arms [20] and the PARC thermal market [4, 9].

The Zone Logic system [28] applies agents to real-time control. Offered as a commercial product by Septor Corporation in the late 1980's and early 1990's, the system was part of regular automotive production at Chrysler, GM, and Saturn.

Complicated manufactured parts such as engine blocks are often manufactured on a machine called a transfer line. Such a machine moves workpieces sequentially through a series of stations. At each position, individual mechanisms perform some specific task. For example, the first station might bore a hole, the second might thread the hole, and the third might screw a hardened insert into the threaded hole.

A transfer line permits higher processing rates than discrete machines served by separate material transport systems. It contains a large number of mechanisms that must be controlled and coordinated. A typical transfer line may be a hundred meters long and contain dozens of stations with hundreds of mechanisms and more than 1500 degrees of freedom in movement overall. Traditional control schemes for such systems require the software engineer to understand the relations among all these mechanisms. When the system fails, identifying the responsible mechanism and the reason for the failure can be very time consuming. As a result, transfer lines often are down for maintenance more than half of the time. When the system is restarted after a failure, the various stations must be reset to a standard initial state, often requiring the scrapping or manual reprocessing of parts in process at the time of the failure. Because of the complex interactions among their mechanisms, transfer lines are notoriously difficult to keep operating. In many environments, 50% productivity is the most that can be achieved. By giving mechanisms autonomy, a Zone Logic-controlled machine can readily achieve 90% productivity.

Table 2: Zone Logic Slide Agent.—Simple rules determine the behavior of the agent for each set of state conditions.

		State Variables						
Zone #	& Name	Advanced Limit Switch	Returned Limit Switch	Advance Motor Power	Return Motor Power	Max Time in Zone (sec.)	Next Zone on Time-Out	Allowed Next Zones
1	Initializing	X	X	0	0	0.1		2, 5, 8
2	Returned	0	1	0	0			3
3	Advancing from Returned	0	1	1	0	1.0	9	4, 2
4	Advancing Between	0	0	1	0	5.0	9	5, 8
5	Advanced	1	0	0	0			6
6	Returning from Advanced	1	0	0	1	1.0	9	7, 5
7	Returned Between	0	0	0	1	5.0	9	4, 8
8	Stopped Between	0	0	0	0			4, 7
9	Error Default	X	X	0	0			1

0 = off; 1 = on; X = ignore

4. Industrial Insights

In the light of the growing success of industrial agent-based applications, several lines of encouragement can be offered to potential users who may be wary of adopting agent-based solutions.

You're almost using agents already.—As Table 1 suggests, agents are only an incremental step beyond object technology, which is widely deployed in industry. They represent the natural evolution of software, and are supported by an increasing array of commercial tools and methods.

Your competition is profiting from agents.—The examples in the previous section are only a brief selection from a growing list of projects, with significant involvement from companies including Advanced Micro Devices, Allied Signal, British Telecom, Daewoo Motors, DaimlerChrysler, General Motors, Hewlett Packard, Iberdrola, IBM, Japan National Railway, Kawasaki Steel, Newport News Shipbuilding, Nortel, Odense Naval Shipyard, Saturn Motors, Southwest Airlines, Unilever, and Xerox, among others. The degree of deployment has grown to the point that the major risk to businesses now is not being first to test whether agents can work, but being last to profit from their capabilities.

Even simple agents are useful.—A glance at the proceedings from a research-oriented agent conference such as the International Conference on Multi-Agent Systems or the International Conference on Autonomous Agents is sometimes intimidating because of the vast array of techniques and agent capabilities being explored. Understandably, researchers emphasize the importance of their own specialties, and at one time or another such capabilities as planning, social modeling, mobility, user modeling, mentalistic state, or the ability to manipulate a sophisticated agent communication language have been promoted as necessary conditions for agent-hood. From an industrial perspective, it is more useful to consider these and other

capabilities as accessories. They are part of the agent builder's toolkit, available for use when the problem demands. But many of the fielded applications of agents use much simpler agents, essentially objects on steroids.

We know when and when not to use agents.—Like any other technology, agents have certain capabilities, and are best used for problems whose characteristics require those capabilities. Five such characteristics are particularly salient in the examples we have reviewed: agents are best suited for applications that are modular, decentralized, changeable, ill-structured, and complex.[2] Industrialists faced with problems that fit these characteristics will find that agents are a natural, proven way to address them.

- Agents are fundamentally a specialization of software objects, and share the benefits of *modularity* that have led to the widespread adoption of object technology. They are best suited to applications that fall into natural modules. An agent has its own set of state variables, distinct from those of the environment. The agent's input and output mechanisms couple its state variables to some subset of the environment's state variables. An industrial entity is a good candidate for agent-hood if it has a well-defined set of state variables that are distinct from those of its environment, and if its interfaces with that environment can be clearly identified.
- As a pro-active object, an agent does not need to be invoked externally by a central authority, but autonomously monitors its own environment and takes action as it deems appropriate. This *decentralized* character of agents makes them particularly suited for applications that can be decomposed into stand-alone processes, each capable of doing useful things without continuous direction by some other process.
- Agents are well suited to modular problems because they are objects. They are well suited to decentralized problems because they are pro-active objects. These two characteristics combine to make them especially valuable when a problem is likely to *change frequently*. Modularity permits the system to be modified one piece at a time. Decentralization minimizes the impact that changing one module has on the behavior of other modules. Modularization alone is not sufficient to permit frequent changes. In a system with a single thread of control, changes to a single module can cause later modules, those it invokes, to malfunction. Decentralization decouples the individual modules from one another, so that errors in one module impact only those modules that interact with it, leaving the rest of the system unaffected.
- An early deliverable in traditional systems design is an architecture of the application, showing which entities interact with which other entities and specifying the interfaces among them. Some applications (including many that support electronic commerce) are intrinsically under-specified and thus *ill-structured*, and agents offer the only realistic approach to managing them. Even where more detailed structural information is available, the wiser course may be to pretend that it is not. A system that is designed to a specific domain structure will require modification if that structure changes. Agent technology permits the analyst to design a system to the classes that generate a given domain structure rather than to that structure itself, thus extending the useful life of the resulting system and reducing the cost of maintenance and reconfiguration.

[2] These are an extension of the categories described in [11].

- Modern competitive requirements greatly increase the *complexity* of the manufacturing problem. Increased product complexity explodes the size of the space that designers must search and the number of machines that must be scheduled and controlled, and makes the task of mapping between design and simulation model more daunting. Supply networks and increased product variability push traditional optimal scheduling against the wall of computational complexity. As is often the case, the complexity in this example is combinatorial in nature, resulting from the fact that the number of different interactions among a set of elements increases much faster than does the number of elements in the set. By mapping individual agents to the interacting elements, agent architectures can replace explicit coding of this large set of interactions with generation of them at run-time. Consider 100 agents, each with ten behaviors, each behavior requiring 10 lines of code. The total amount of software that has to be produced to instantiate this system is 10,000 lines of code, a relatively modest undertaking. But the total number of behaviors in the repertoire of the resulting system is on the order of ten for the first agent, times ten for the second, times ten for the third, and so forth, or 10^{100}, an overwhelmingly large number. Naturally, not all of these will be useful behaviors, and one can imagine pathological agent designs in which none of the generated behaviors will be appropriate. However, appropriately designed agent architectures can move the generation of combinatorial behavior spaces from design-time to run-time, drastically reducing the amount of software that must be generated and thus the cost of the system to be constructed. Just as well-structured systems can become ill-structured when viewed over their entire life span, so a system that appears to require only a few behaviors can become more complex as it is modified in response to changing user requirements. By adopting an agent approach at the outset, systems engineers can provide a much more robust and adaptable solution that will grow naturally to meet business needs.

5. Research Opportunities

The tremendous promise of agent technologies for industrial application has led to impressive instances of their adoption by commercial organizations. However, significant hurdles remain, hurdles that invite the concerted effort of the agent research community. Two prominent and promising areas of research include methods and tools for system design and development, and methods for understanding the emergent dynamics of systems involving multiple agents.

5.1 Design Methods and Development Tools

The increased interest of industrial users in agents highlights important differences between the requirements for developing agents in academic and in industrial circles.

- In a university setting, developers are highly-trained agent researchers with an intrinsic interest in the technology and graduate-level expertise in the latest techniques. They enjoy (and are often rewarded for) innovation on the spot as they encounter shortcomings in their tools. Industrial developers are professional software engineers with a focus on the requirements of the problem to be solved and the time and cost of developing a solution. While they are highly skilled, they cannot be assumed to be *au courant* on the latest agent research results, and the environment discourages them from trying radical new ideas.
- The rank and file of academic researchers are Ph.D. candidates, leading most academic agent projects to assume the scope of a single dissertation. The emphasis is on what one person can accomplish in a few years. The individual researcher can manage much of the coherence of the

project personally, without the need for sophisticated tools. It is not unusual for an industrial development project to have a scope of hundreds of person-years, posing major challenges of coordination among a large team of developers who may be widely separated geographically.

- The success of an academic project is determined by peer review. If a project demonstrates a sufficiently novel and interesting technique, it stands a good chance of being judged a success. Industrial projects must provide real-world functionality that meets or exceeds current technology at a competitive price. Most industrial users are unimpressed by the techniques under the hood of an application. They care about the level of end-user functionality provided.

Because of these contrasts, the tools and methods that suffice for academic development are not adequate for industrial use. An important part of the business justification for a system is an estimate of how much it will cost and when it will be ready for use. The methods most commonly used for designing and building agent-based systems, even in industry, still amount to clever hacking, and generally require the leadership of someone intimately acquainted with the wide range of research underway in multi-agent systems to point out the best practice at each step. Wide-spread deployment of agent technology requires that best practice be packaged so that it can be deployed by practicing engineers rather than requiring Ph.D. scientists. Researchers can play a critical role in demonstrating potential tools to support industrial-strength development.

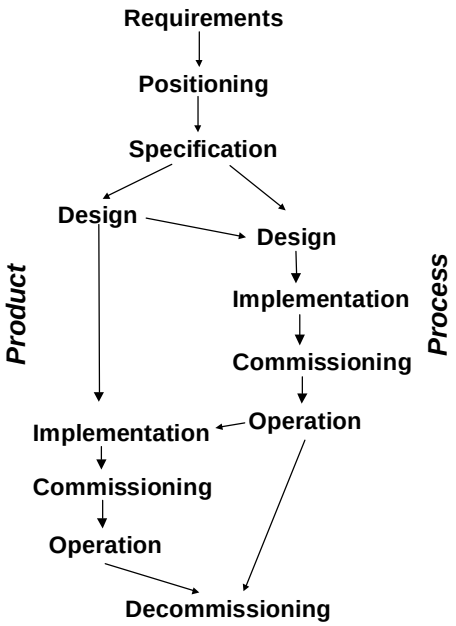

Figure 7: Industrial Life Cycle.—Industrial projects are managed through a well-defined sequence of stages, many of which require special attention to accommodate agents.

- The scope of industrial projects requires that they be managed in the context of a well-defined life cycle such as that outlined in Figure 7 [22]. The research community should become better acquainted with life-cycle-oriented development, and focus their skills on the holes in this process that are not adequately supported by current tools.
- Communication among members of a development team and between stages of the life cycle requires well-defined artifacts for the documentation of a system as it develops. The

development of the Unified Modeling Language as a standard representation, supported by a number of CASE tools, has done much to move the design of an object-oriented system from the realm of science to engineering. This set of conventions needs to be extended to accommodate the additional functionality of agents, such as coordination requirements and conversational structure [21, 29].

- While one can write object-oriented systems in C or Pascal with appropriate conventions, it greatly helps to have languages such as Smalltalk, C++, and Java to enforce appropriate best practice. Encouragingly, a plethora of agent development environments has emerged over the last several years, largely building on previous object-oriented environments (for example, ObjectSpace's Voyager on Java [19], or Gensym's Agent Development Environment on G2). As new techniques are developed in the research world, these tools need to be extended to make them available to industry.

5.2 Managing the Dynamics of Multi-Agent Systems

One of the justifications for agents is that the whole system can be more than the sum of its parts. A collection of relatively simple agents can yield surprisingly rich and complex interactions. This potential represents a two-edged sword. The behavior of the whole system is often not obvious from the outset, and can challenge the viability of the application if it is not managed appropriately. For example, Kephart and colleagues [14] have shown two emergent characteristics of an information economy with information sources, brokers, and users. Brokers tend to specialize spontaneously around particular topics, a powerful and valuable form of self-organization. However, they engage in cyclical price wars, an undesirable instability. Further research is needed on tools and methods to handle these dynamics, including techniques for simulating communities of agents, mathematical methods for the analysis of the behavior of such systems, and evolutionary mechanisms for growing agent populations to satisfy specified requirements.

6. Summary

Agents are already moving out of the laboratory and into the industrial workplace, and both industry and the research world are better off for these early successes. Continued advances in the field depend on a deeper appreciation of each side of what the other has to offer. Enough "success stories" exist to enable users to appreciate the distinctive capabilities of agent technologies and make intelligent decisions about deployment. In turn, the research community can usefully focus its attention on specific issues that have both deep scientific interest and considerable potential for responding to the specific needs of real-world applications.

7. REFERENCES

[1] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. New York, Oxford University Press, 1999.

[2] E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Theraulaz. Routing in Telecommunications Networks with "Smart" Ant-Like Agents. Santa Fe Institute, Santa Fe, NM, 1998.

[3] F.-C. Cheong. *Internet Agents: Spiders, Wanderers, Brokers, and Bots*. Indianapolis, IN, New Riders, 1996.

[4] S. H. Clearwater and B. A. Huberman. Thermal Markets for Controlling Building Environments. Xerox PARC, Palo Alto, CA, 1993.

[5] D. Corne, M. Dorigo, and F. Glover, Editors. *New Ideas in Optimisation*. New York, McGraw-Hill, 1999.

[6] F. Federspiel. Complexity Solutions for Manufacturing and Logistics. In *Proceedings of Seventh Annual Santa Fe Chaos in Manufacturing Conference*, M. Morley, Inc., 1999.

[7] R. Hauser and D. Clausing. The House of Quality. *Harvard Business Review*, 66(May-June):63-73, 1988.

[8] M. Heusse, D. Snyers, S. Guérin, and P. Kuntz. Adaptive Agent-Driven Routing and Load Balancing in Communication Networks. ENST de Bretagne, Brest, FR, 1998.

[9] B. A. Huberman and S. H. Clearwater. A Multi-Agent System for Controlling Building Environments. In *Proceedings of ICMAS'95: First International Conference on Multi-Agent Systems*, pages 171-176, AAAI Press/MIT Press, 1995.

[10] H. Ihara and K. Mori. Autonomous Decentralized Computer Control Systems. *IEEE Computer*, 17(8):57-66, 1984.

[11] N. Jennings. Applying Agent Technology. 1996. Plenary presentation at PAAM'96.

[12] N. R. Jennings. The ARCHON System. 1996. Web Page, http://www.elec.qmw.ac.uk/dai/archon/test_1.html.

[13] N. R. Jennings, E. H. Mamdani, I. Laresgoiti, J. Perez, and J.Corera. GRATE: A General Framework for Co-operative Problem Solving. *IEE-BCS Journal of Intelligent Systems Engineering*, 1(2 (Winter)):102-14, 1992.

[14] J. O. Kephart, J. E. Hanson, D. W. Levine, B. N. Grosof, J. Sairamesh, R. B. Segal, and S. R. White. Dynamics of an Information-Filtering Economy. In M. Klusch and G. Weiss, Editors, *Cooperative Information Agents II*, vol. 1435, *Lecture Notes in Artificial Intelligence*, pages 160-170. Springer, Berlin, 1998.

[15] H. S. Lee, S. Murthy, S. W. Haider, and D. Morse. Primary Production Scheduling at Steel-Making Industries. *IBM Journal of Research and Development*, 40(2 (March)):231-252, 1996.

[16] J. Mori, H. Torikoshi, K. Nakai, K. Mori, and T. Masuda. Computer Control System for Iron and Steel Plants. *Hitachi Review*, 37(4):251-8, 1988.

[17] K. Mori, H. Ihara, Y. Suzuki, K. Kawano, M. Koizumi, M. Orimo, K. Nakai, and H. Nakanishi. Autonomous Decentralized Software Structure and its Application. In *Proceedings of Fall Joint Computer Conference*, pages 1056-63, 1986.

[18] S. Murthy, R. Akkiraju, J. Rachlin, and F. Wu. Agent-Based Cooperative Scheduling. In *Proceedings of AAAI Workshop on Constraints and Agents*, pages 112-117, AAAI Press, 1997.

[19] ObjectSpace. Voyager: Agent-Enhanced Distributed Computing for Java. 1997. Web Page, http://www.objectspace.com/Voyager/voyager.html.

[20] L. Overgaard, H. G. Petersen, and J. W. Perram. Motion Planning for an Articulated Robot: A Multi-Agent Approach. In *Proceedings of Modelling Autonomous Agent in a Multi-Agent World*, pages 171-182, Odense University, 1994.

[21] H. V. D. Parunak. Visualizing Agent Conversations: Using Enhanced Dooley Graphs for Agent Design and Analysis. In *Proceedings of Second International Conference on Multi-Agent Systems (ICMAS'96)*, pages 275-282, 1996.

[22] H. V. D. Parunak. Industrial and Practical Applications of DAI. In G. Weiss, Editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge, MA, 1999.

[23] H. V. D. Parunak, A. C. Ward, M. Fleischer, J. Sauter, and T.-C. Chang. Distributed Component-Centered Design as Agent-Based Distributed Constraint Optimization. In *Proceedings of AAAI Workshop on Constraints and Agents*, pages 93-99, American Association for Artificial Intelligence, 1997.

[24] H. V. D. Parunak, A. C. Ward, M. Fleischer, and J. A. Sauter. The RAPPID Project: Symbiosis between Industrial Requirements and MAS Research. *Autonomous Agents and Multi-Agent Systems*, 2:2 (June):111-140, 1999.

[25] H. V. D. Parunak, A. C. Ward, and J. A. Sauter. A Systematic Market Approach to Distributed Constraint Problems. In *Proceedings of International Conference on Multi-Agent Systems (ICMAS'98)*, American Association for Artificial Intelligence, 1998. Available at http://www.erim.org/cec/rappid/icmas98.pdf.

[26] H. V. D. Parunak, A. C. Ward, and J. A. Sauter. The MarCon Algorithm: A Systematic Market Approach to Distributed Constraint Problems. *AI-EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Forthcoming, 1999.

[27] J. Rachlin, R. Goodwin, S. Murthy, R. Akkiraju, F. Wu, S. Kumaran, and R. Das. A-Teams: An Agent Architecture for Optimization and Decision-Support. In *Proceedings of Fifth International Workshop on Agent Theories, Architectures, and Languages*, pages 17-31, University Pierre et Marie Curie, 1998.

[28] R. Roberts. *Zone Logic: A Unique Method of Practical Artificial Intelligence*. Radnor, PA, Compute! Books, 1989.

[29] M. P. Singh. Developing Formal Specifications to Coordinate Heterogeneous Autonomous Agents. In *Proceedings of Third International Conference on Multi-Agent Systems (ICMAS'98)*, pages 261-268, IEEE Computer Society, 1998.

[30] A. Ward, J. K. Liker, J. J. Cristiano, and D. K. S. II. The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster. *Sloan Management Review*, (Spring):43-61, 1995.

[31] T. Wittig. *ARCHON: An Architecture for Multi-agent Systems*. New York, Ellis Horwood, 1992.

