Case Study: Intelligent Software Supply Chain Agents using ADE

Dr. Anshu Mehra
Gensym Corporation
125 Cambridge Park Dr.
Cambridge, MA 02140
AMehra@gensym.com

Dr. Mark Nissen
Naval Postgraduate School
555 Dyer Rd. Code SM/Ni
Monterey, CA 93943
MNissen@nps.navy.mil

Abstract

This paper reviews extant agent applications and describes the Agent Development Environment (ADE) toolkit. ADE is the integrated development environment to design, develop, debug, simulate and deploy agents. ADE supports the development of multi-agent applications capable of running on a single machine or on a distributed network. ADE has been used to build commercial applications in the area of: (i) manufacturing scheduling, (ii) manufacturing process control, (iii) network information filtering, and (iv) network load balancing.

We present intelligent supply chain agents for the software procurement process using electronic commerce. The paper highlights the use and utility of intelligent agents in electronic commerce using ADE. Supply chain management represents a critical competency in today's fast-paced, global business environment. However, in the current transition from EDI to Web technology, most of the capability for process integration is being lost. The *integration* of buyer and seller supply chain processes is critical for speed and responsiveness in today's hypercompetitive product and service markets. Intelligent agent technology offers the potential to overcome this limitation and effectively integrate buyer and seller processes without the rigid inflexibility of EDI. We use domain knowledge in software procurement and distributed problem solving approaches. The paper concludes with suggestions for future research.

Commerce Through Intelligent Suppy Chain Agents

Supply chain management (see Porter and Millar 1985) represents a critical competency in today's fast-paced, global business environment, and a number of effective practices (e.g., just-in-time deliveries, electronic data interchange (EDI), supplier inventory management) are employed to improve the competitiveness and efficiency of enterprises around the world. With the continuing surge of activities on the Web and corresponding research on electronic commerce, many firms are moving to Web-based support for commercial transactions (e.g., electronic catalogs, storefronts, malls, etc.). In fact, Web-based commercial transactions are beginning to supplant the traditional EDI for some business-to-business commerce, which itself represents a quantum improvement over paper-based processes.

However, most of the capability for business process integration is being lost during the transition from EDI to Web technology. Whereas EDI effectively compels buyers and sellers to integrate their supply chain processes, Web-based supply chain technologies are noticeably one-sided; that is, the latter sites and applications are predominately developed for *either* the buyer or seller, but *not both*. Our two decades of experience with EDI (see Sokol 1996) suggest that *integration* of buyer and seller supply chain processes is critical for speed and responsiveness in today's hypercompetitive business environment (see D'Aveni 1994).

Alternatively, intelligent agent technology offers the potential to effectively integrate buyer and seller processes without the rigid inflexibility of EDI. Using domain knowledge and distributed problem-solving technology to develop a set of intelligent supply chain agents, in this paper we demonstrate this potential through the agent-integrated supply chain process of software procurement. We first provide a high-level overview of extant agent applications and later describe the Agent Development Environment (ADE) that is used to design and implement intelligent supply chain agents. A case example follows to demonstrate the feasibility and to highlight the use and utility of intelligent agents in this commercial domain. The paper closes with a set of conclusions and suggestions for future research along these lines.

Extant Agent Applications

Work in the area of software agents has been ongoing for some time and it addresses a broad array of applications. Indeed, one need not research too far back in the literature to identify a plethora of agent examples--so many that any attempt to review them, even briefly, would constitute a journal-length paper in and of itself. In this section we provide a high-level overview of extant agent applications, with a particular emphasis on a framework to relate them with this present work.

It is informative to group extant agent applications into four classes: 1) information filtering agents, 2) information retrieval agents, 3) advisory agents, and 4) performative agents. Briefly, most information filtering agents are focused on tasks such as filtering user-input preferences for

e-mail (e.g., Maes 1994, Malone et al. 1987), network news groups (Sycara and Zeng 1996), frequently asked questions (Whitehead 1994) and arbitrary text (Verity 1997). Information retrieval agents address problems associated with collecting information pertaining to commodities such as compact disks (Krulwich n.d.) and computer equipment (uVision 1997), in addition to services such as advertising (PriceWatch 1997) and insurance (Insurance 1997). We also include the ubiquitous Web indexing robots in this class (see Etzioni and Weld 1995) along with Web-based agents for report writing (Amulet 1997), publishing (InterAp 1995) and assisted browsing (Burke et al. 1997). Agents for technical information delivery (Bradshaw et al. 1997) and information gathering (Knobloch and Ambite 1997) are not Web-based per se, but they perform a similar function.

A third class of agents is oriented toward providing intelligent advice. Examples include recommendations for CDs (Maes 1997), an electronic concierge (Etzioni and Weld 1995), an agent "host" for college campus visits (Zeng and Sycara 1995) and planning support for manufacturing systems (Maturana and Norrie 1997). Agents for strategic planning support (Pinson et al. 1997), software project coordination (Johar 1997) and computer interface assistance (Ball et al. 1997) are also grouped in this class, along with support for military reconnaissance (Bui et al. n.d.) and financial portfolio management (Sycara et al. 1996). Performative agents in the fourth class are generally oriented toward functions such as business transactions and work performance. Examples include a marketplace for agent-to-agent transactions (Chavez and Maes n.d.) and an agent system for negotiation (Bui n.d.), in addition to the performance of knowledge work such as automated scheduling (Sen 1997, Walsh et al. 1997), cooperative learning (Boy 1997) and automated digital services (Mullen and Wellman 1996).

The intelligent supply chain agents developed through this present research are probably best categorized in the fourth group above (i.e., performative agents), but they have been designed to also exhibit behaviors such as information filtering and retrieval, and their use can be accomplished through simulation (i.e., in an advisory role) as well as enactment (i.e., the performative role). Thus, intelligent supply chain agents have similarities with examples from each of the four classes above. To further describe and differentiate intelligent supply chain agents, we have integrated the agent-taxonomy work of Franklin and Graesser (1996) with a three-dimensional structure from Gilbert et al. (1995) to develop the analytical framework presented in Figure 1.

In this framework we use the same intelligence and mobility dimensions noted in the three-dimensional structure above, but with the substitution of the new dimension *collaboration* in lieu of autonomy/agency. This follows the presumption of agent autonomy stressed by Franklin and Graesser. For purpose of discussion, we have annotated this three-dimensional space with one, relatively "pure" exemplar from each dimension. For example, many expert system applications are quite extensive in terms of formalized, expert-level intelligence, but they traditionally are not designed to operate on foreign hosts nor do they generally collaborate with other expert systems to jointly solve problems. Similarly, remote programming of the sort enabled by Java and Telescript equip programs to execute on foreign machines, but these procedural applications are not generally endowed with the capability for intelligent inference nor are they usually thought of in terms of collaborative processing. Likewise, parallel processing has an explicit focus on collaborative problem solving between multiple, parallel processors, but this problem solving is usually focused more on procedural processing than intelligent reasoning and execution on foreign hosts is rarely envisioned. Clearly exceptions exist for each class (e.g., distributed AI, intelligent Java agents, etc.), but these three exemplars should convey the basic concepts associated with each dimension.

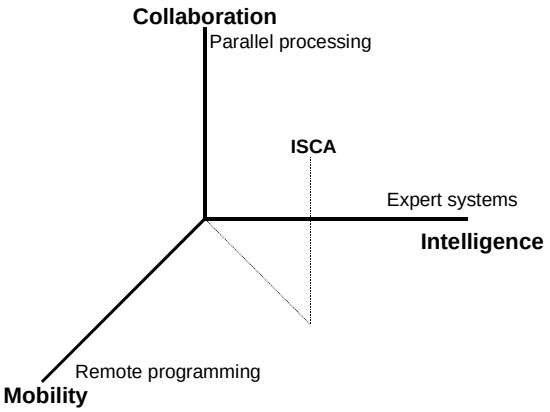

Figure 1 Agent Framework

Notice the annotation for intelligent supply chain agents (labeled "ISCA" in the figure). Although this class of systems is not as extreme as any of the three exemplars from above along any particular dimension, it occupies a position roughly in the middle of this three-dimensional agent space; all three of the exemplars from above are situated along only a single axis. This adds to the challenge of our agent development work, but it serves to enable a new set of capabilities that prove to be quite effective and useful for operational processes such as software supply chain management. With this in mind, we turn now to the agent development environment and architecture for this class of intelligent supply chain agents.

Agent Development Environment and Architecture

Agent Development Environment (ADE) is the integrated development environment to design, develop, debug, simulate and deploy agents. ADE is built on G2, an object-

oriented graphical environment that offers a robust platform for the development of intelligent real time systems. ADE supports the development of multi-agent applications capable of running on a single machine or on a distributed network. The main ADE components are *Agent*, *Message*, *Activity*, *Host* and *Environment*. In this section we briefly outline each in turn, followed by a discussion of agent simulation. Examination of this ADE architecture instantiated for a manufacturing supply chain example is presented to close the section. We begin with a high-level architectural schema that inter-relates each of these ADE components. This is diagrammed in Figure 2.

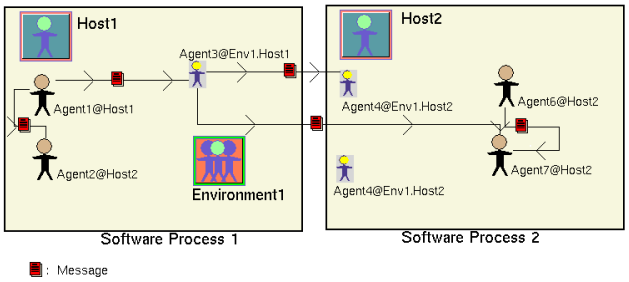

Figure 2 ADE Architectural Schema

Agent. In ADE, agents communicate through messages or events (a subclass of message). ADE provides a basic direct addressing message service, with some optional functionality (e.g., guaranteed delivery, message broadcast, subject-based addressing). ADE uses delegation based event handling similar to the JavaBeans model in which agents use messages to generate and listen for events. Each agent has a network-wide unique name. This enables communication among agents distributed across a network to be independent from an agent's location. Agents refer to each other by their name and the name of an agent cannot be changed during its entire "life." ADE provides a "*Yellow Pages*" lookup capability; that is, specific properties can be defined for agents, enabling other agents to send messages qualified by their properties. Each agent can query the yellow pages to find the names of agents matching a specific Boolean set of properties. Agents can be dynamically created, deleted, cloned and moved across the network. ADE provides a base agent class called *AdeAgent*. AdeAgent can be specialized and augmented by application specific agent types. Example agents include: ResourceMonitoringAgent, ManufacturingCell-Agent and JobBrokerAgent.

Agents in ADE are autonomous, multi-threaded objects with their own state. Each thread of control of an agent is represented by an activity instance. An agent can concurrently perform multiple activities. For example, a *MachineToolAgent* can be concurrently performing two activities: monitoring a machine job and negotiating future jobs with other agents. Messages and other events are sent, and listened for, within the context of a specific activity of an agent. Agent activities are defined either using the *Grafcet* graphical language (discussed below) or directly with methods for activity subclasses.

Message. ADE provides a base level message class of type *AdeMessage*. Agents communicate with each other by sending objects of type AdeMessage or its subclass. A message contains the destination agent name. Messages are handled by agent activities. A message can be sent to a specific activity of an agent. In ADE, no acknowledgment is required for messages. Exchange of messages between agents may be synchronous or asynchronous. A synchronous message blocks the activity of the agent until the reply is received from the agent to which the message was sent. Alternatively, an agent may continue to perform its activity without blocking. It is assumed that messages take a finite amount of time to be delivered. Thus, it is possible for messages to get delayed or lost, and for messages sent in opposite directions by different agents to cross one another (i.e., both be in transit at the same time). ADE supports two major subclasses of AdeMessage: (i) AdeSolication is a message for which the sending agent expects a reply; and, (ii) AdeAssertion is a message for which the sending agent expects no reply. Messages are used for the communication between agents and between the different activities of the same agent. Communication between agents and external devices or processes is also accomplished through messages. A subclass of AdeMessage called AdeEvent is provided in ADE for discrete event simulation.

Activity. An activity defines a specific behavior of an agent. *AdeActivity* class provided in ADE facilitates the development of a multi-thread capability without dealing with threads, stacks and priorities. An agent may be concurrently performing multiple activities of the same type or of different types. Within an activity, multiple threads may be active at the same time. Messages sent to an agent may either initiate a new activity or may continue a dialog with an ongoing activity. In the first case, the agent starts a new thread of activity. During execution of an activity the agent can send and receive synchronous and asynchronous messages. Once an activity is started, the message can be sent directly to it. An activity maintains a queue of received messages. Within an agent, the *AgentHandler* defines the destination activity for each message received. This handler is called when a message does not identify its destination activity, which usually occurs when an agent is initiating communication with other agents.

Activities are defined either as methods or using Grafcets (see Figure 3 for an example Grafcet used to define a supply chain management application). *AdeGrafcet* is a graphical language that shows both parallel and sequential control structures in easy-to-understand pictorial form. AdeGrafcet is an extension of Grafcet, or Sequential Function Charts (SFC), a graphical language that has been accepted as an industrial standard (IEC 848 and IEC 1131-3) for local, PLC-level sequential logic control (David and

Alla 1992). A *Grafcet Chart* contains *Nodes* and the *Links* among them define the flow of control of the activity of an agent. The main types of nodes are *Steps*, *Transitions*, *MacroSteps*, *IterativeSteps* and *ProcessSteps*. The main types of links are *Branches* and *Joins*.

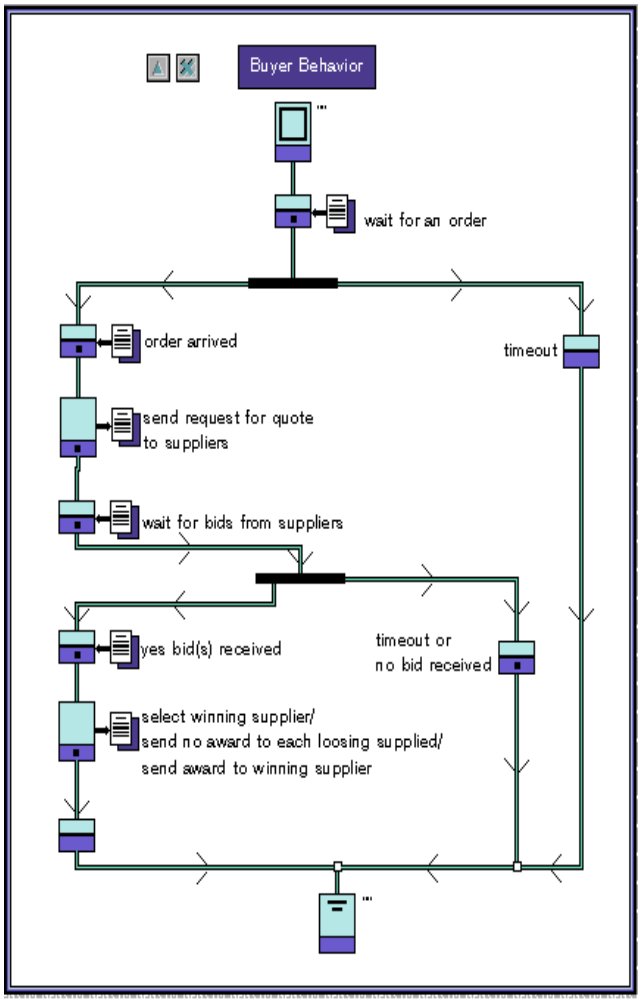

Figure 3 Sample Grafcet Chart for Supply Chain Management

A *Step* represents a state, phase or mode. Associated with a step are actions that are performed when a step is activated. In standard Grafcet the actions that can be done in a step are of a Boolean nature, whereas AdeGrafcet actions in steps are more general; they can be compared with statements of a conventional programming language. Message statements to other agents may be embedded in the action of a step, and actions are internally represented as procedures. The transitions act as gates on the flow of control through the Grafcet Chart. Each transition is associated with a condition that determines whether or not control can pass through the transition. In ADE Grafcet transition conditions are expressed as Boolean expressions written as procedures. Control can pass through a transition when its Boolean control expression evaluates to TRUE. Wait statements for specific messages from other agents may be embedded in condition procedures. AdeGrafcet also provides *MacroStep* as a way to embed one Grafcet chart in another. *IterativeStep* enables the definition of embedded Grafcet Charts whose process is repeated a number of times. *ProcessStep* is a MacroStep executed in more than one Grafcet Chart. They are equivalent to subroutines in standard programming languages. A *Link* connects steps to transitions. Grafcet allows a single step to be followed by more than one transition, and a single transition to be followed by more than one step. Thus, Grafcet allows control to fan-in and fan-out, and Grafcet provides for a choice between synchronous and asynchronous operations through a variety of fan-in and fan-out links. There are five types of links: Asynchronous Branch, SynchronousBranch, First-True Branch, Asynchronous Join and Synchronous Join.

Host. In ADE, every agent registers itself to *AdeHost*. There is one AdeHost for every software process on which a multi-agent application is running. An AdeHost is responsible for delivering messages, as well as dynamically initializing, moving, cloning and destroying agents. When an agent is created, it is assigned to a specific host. The host then installs the agent, registers the agent properties and, if requested, connects the agent to databases, on-line control systems, etc. The "*Locator Service*" of a host enables each agent to locate all the other agents in the application. When an agent moves (e.g., from one machine to another), AdeHost forwards all the future messages to the new address.

Environment. ADE supports agent clusters by providing a special agent called AdeEnvironment. As depicted in the figure above, agents belonging to an environment may reside on different hosts. AdeEnvironment enables hierarchical grouping and encapsulation of agents and provides local "*Yellow Pages*" services. Although agents can move to different hosts across a network, an agent may belong to only one environment. Agents within an environment may be disallowed to communicate with outside agents, and an environment can be a cluster of other environments.

Agent Simulation. Because agent-based systems can exhibit complex emergent dynamics, simulation is an essential component of a multi-agent development environment. ADE supports simulation during development through a *SimulationAgent* that emulates the behavior of external devices or processes. In this way, the interaction of agents with the external physical environment can be simulated during the development phase. When the multi-agent application is deployed, the interface with the Simulation-agents is replaced by the actual interface with the physical devices.

Summary. In summary, ADE provides (i) a predefined class hierarchy of agents and agent components; (ii) an agent communications "middleware"; (iii) a graphical programming language to design and develop agents' behavior based on the Grafcet standard; (iv) a distributed simulation environment to test multi-agent applications built with ADE; (v) a complete debugging and tracing environment; and, (v) a deployment center to deploy agents in the G2 environment or as "JavaBeans" in Java Virtual Machine.

Manufacturing Supply Chain Instantiation

To add context to this discussion, ADE is used to build a generic, three-level supply chain. Figure 4 shows a process with an order agent, a customer agent and four suppliers--one each for the engine, crankshaft, piston and cylinder. The customer agent requires engines to make cars, and the engine agent requires crankshafts, pistons and cylinders. In this example, all agents are on a single process, but they could just as well be distributed on multiple processes instead. To further elaborate, we briefly address the registration of agents in this supply chain instance and the negotiation protocol defined for their purposeful communications.

Registration of Agent. As noted in previous sections, each agent has a unique name and specific properties. For example, an engine supplier provides engines of a certain kind and quality. Each agent and its properties are registered with the host. Agents can find each other by querying the host. An example query by an agent may be: "get the name of all engine suppliers located in Ohio." Agents communicate by sending messages addressed to the name of each destination agent. The agent handler distributes the message to the appropriate Activity (thread) instance inside an agent. The logic of an Activity type of an agent is described using a Grafcet Chart similar to one in Figure 3. Agent sends the message via AdeHost. The AdeHost locates the address of destination agent and delivers the message.

Negotiation Protocol. The five-layer contract negotiation protocol used in this example is delineated in Figure 5. The customer sends a "request for quotation" (RFQ) message to the first-tier supplier. The supplier then sends "RFQ" messages to the second-tier suppliers. The first tier supplier is therefore a consumer of the second tier suppliers' products, so the same rules, protocols and intelligent behaviors can be defined and applied recursively for supply chains of arbitrary depth. Each supplier sends a "bid" message with a due date, quantity and cost, or a "no-bid" message with reason for no-bid. The customer then sends an "award" or a "no-award" to each supplier. The suppliers return either "confirmation" or "non-confirmation" messages. Finally the customer sends either a "proceed" message with purchase order number or a "no-proceed" message.

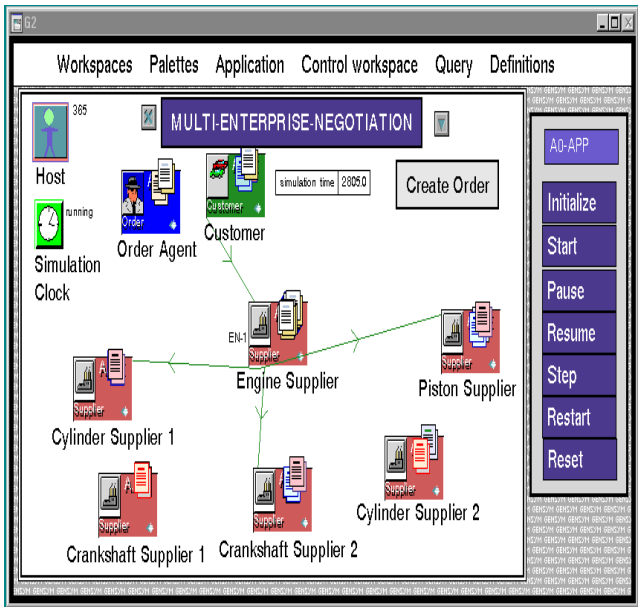

Figure 4 Modeling of Agents in a Supply Chain Management Application

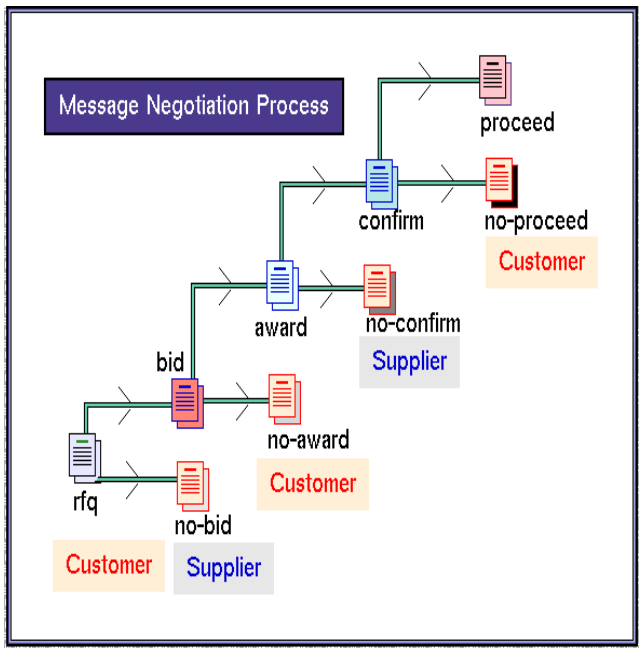

Figure 5 Contract Negotiation Protocol

GOVERNMENT SOFTWARE PROCUREMENT APPLICATION

In this section we discuss the use of ADE and application of intelligent supply chain agents to redesign the government software procurement process. We begin by outlining the two primary processes involved--government

software purchasing and commercial software order fulfillment. We then describe the structure and behavior of the intelligent supply chain agents developed to perform in this environment.

Software Supply Chain Processes

As noted above, two primary processes are involved with this application of intelligent supply chain agents: government software purchasing and commercial software order fulfillment Because process integration is critical to effective supply chain management, we present and discuss two process instances in terms of a single, integrated whole; that is, both purchasing and order fulfillment are modeled as a single process that spans organizational boundaries. Specifically, the government software purchasing process examined through this investigation pertains to work done by the Supply Department at the Naval Postgraduate School (NPS). Although a leading, accredited university like most schools that offers graduate management, engineering and like degrees, NPS is also a government institution. Therefore it is subject to all the same procurement laws and regulations that govern the purchasing activities of any military unit or federal agency.

The commercial software order fulfillment process examined through this investigation pertains to work done by the Product and Licensing Department at Gensym Corporation. Gensym is a leader in software for developing intelligent real-time applications and maintains an active research and development activity that drives frequent product introductions, updates and releases. Therefore it represents the kind of rapid product evolution that has been problematic for government procurement. The high-level process delineated in Figure 6 depicts the integration of the User, NPS Supply Department and software Contractor. Notice that the process flow in this government software procurement instance differs somewhat from its manufacturing supply chain counterpart discussed above. The power of an agent-based solution derives in part from the ability to capture and formalize the knowledge and details that are specific to each domain and application.

The baseline (i.e., before redesign using agent technology) process begins with a user in the organization identifying a need and determining his or her preliminary software requirements. A market survey follows with the market information (e.g., products, capabilities, companies, prices, etc.) used to complete a (paper-based) procurement request form. This form is submitted to the Supply Department for processing, in which a Buyer verifies the form (e.g., in terms of completeness, required documentation such as sole-source justification, adequate budget, etc.) and then researches some potential sources for procurement (e.g., existing contracts, approved-vendor lists, small/ disadvantaged-business lists, etc.) in addition to the sources identified through the market survey. An RFQ is generally issued and quotations are analyzed by the Buyer, who then summarizes the information for review and source selection by the user. A purchase order is then issued and the transaction is completed as the software is delivered to the user and payment is made.

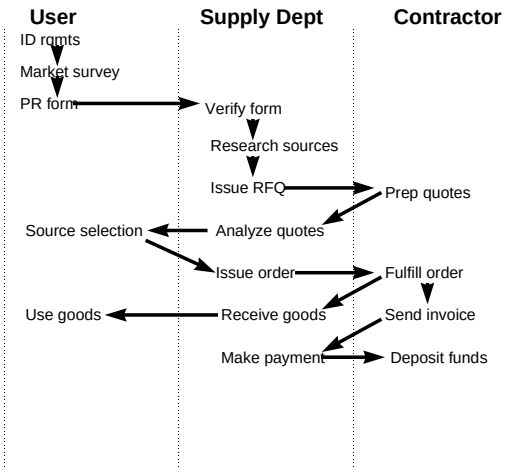

Figure 6 Integrated Process

Not shown in the figure is the underlying knowledge, expertise and information that is required for people to perform the software purchasing and order fulfillment processes depicted above. For example, the user must know how to conduct a market survey and have access to alternative sources of software supply, as well as an understanding of the basic procedures for government procurement and information pertaining to the specific purchase request form used. Similarly, the Buyer must possess thorough knowledge of the Federal Acquisition Regulation and know how to review the purchase request (e.g., what constitutes completeness, when to request additional information, etc.) and have current information pertaining to alternative sources of supply. The Buyer must also have access to one or more suppliers' systems to be able to post the RFQ and requires knowledge of the procedures required for quotation analysis and source selection. Access to and understanding of the receiving and payment systems and procedures is also necessary to complete the transaction, and of course vendor personnel must understand the policies, procedures and systems associated with software product and licensing. These are precisely the kinds of knowledge, expertise and information that we capture in the intelligent software supply chain agents through the integration of rules and methods in the object-oriented, ADE implementation.

Intelligent Software Supply Chain Agent Application

In order to describe the structure and behavior of the intelligent supply chain agents developed to perform in this environment, we draw from the ADE discussion in previous sections and begin with the Grafcets developed to support the NPS-Gensym software supply chain. The first Grafcet, presented in Figure 7, depicts the user behavior and maps homomorphically to the integrated process flow

from Figure 6. For example, the Grafcet flow begins with the user identifying his or her need and determining the preliminary software requirements. The next step involves the market survey. Notice the "market agent" that is identified as the recipient of a task message here. The *supply chain agent* does not care whether this task is accomplished by a human or machine agent, so long as the market survey is completed. Upon receipt of acceptable market survey results, the agent uses its knowledge of NPS purchasing procedures to create the purchase request form which is sent to the Supply Department for processing.

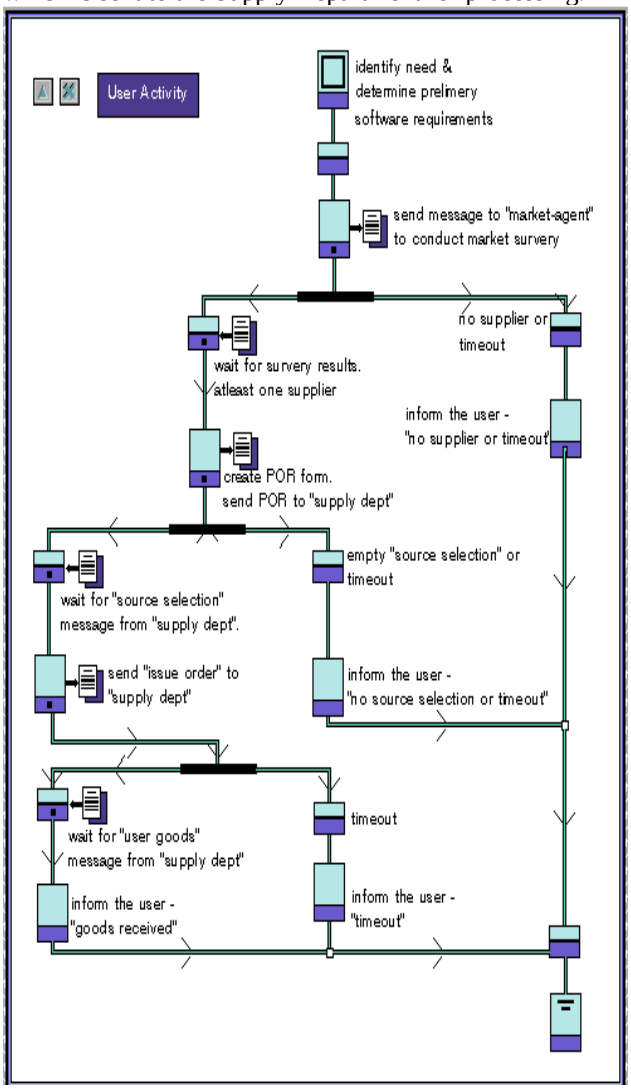

Figure 7 Grafcet for User Behavior

The corresponding Grafcet for the Supply Department is presented in Figure 8, where the supply agent is "listening" for a purchase request. Recall that the agents are multi-threaded, so they can be performing a host of other activities while waiting for such requests. As depicted in Figure 6 above, the supply agent verifies the purchase request, which is either returned for additional information or processed through the subsequent steps (e.g., researching sources, issuing RFQ, etc.) depicted in the figure.

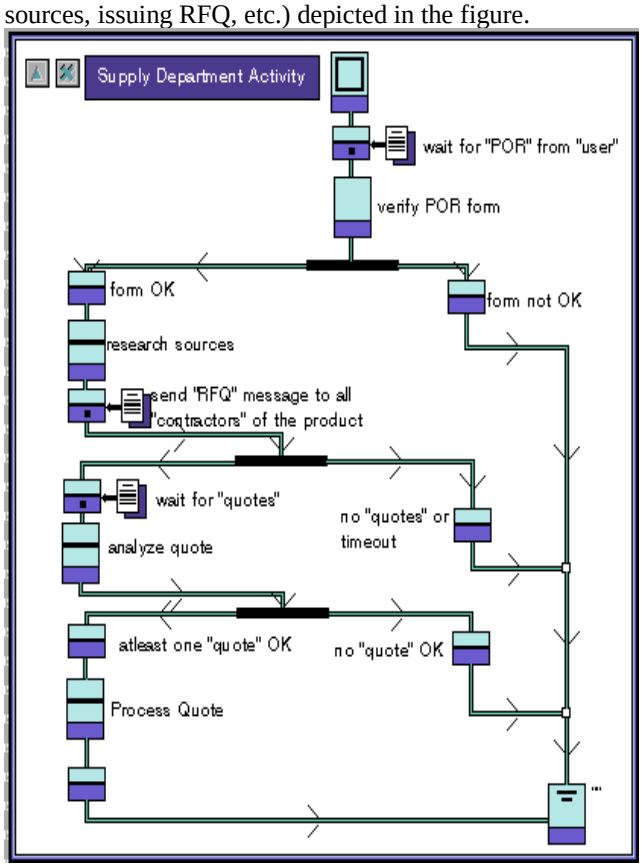

Figure 8 Grafcet for Supply Department Behavior

The software contractor behavior is specified through the Grafcet presented in Figure 9. As above, the contractor agent is multi-threaded, so it can do more than just wait for an incoming order from the NPS Supply Department. Upon receipt of such an order, however, it prepares a quotation and sends it to the requesting agent (i.e., supply). If an order is received, the agent's tasks branch to fulfill the order (i.e., send the software "goods") and invoice the customer.

The agents' activity behaviors are implemented via the Grafcet Charts shown in Figures 7-9. Specific behavior for each step and node of the Grafcet chart is described through a method. The distributed nature of the ADE enables agents to inter-operate on different hosts (i.e., agents can be simultaneously at the customer's and supplier's sites). The distributed agents can communicate with each other via AdeHost. The supply chain application described in this paper involves multiple instances of only a single agent type for the user, supply department and software contractor. A marketspace of multiple agent types can be created by subclassing AdeAgent and describing the agents behavior using Grafcets.

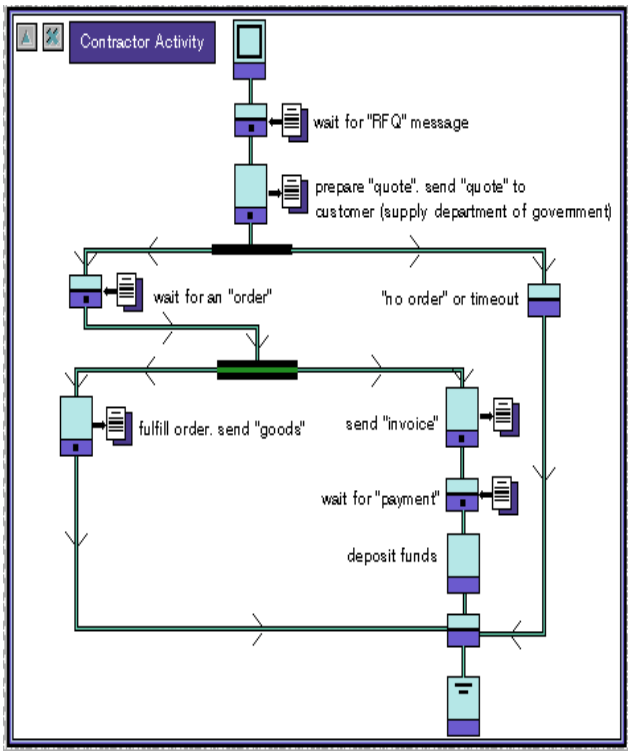

Figure 9 Grafcet for Software Contractor Behavior

OTHER ADE APPLICATIONS

Intelligent Distributed Supply Chain Management Agents

The application uses Agent Development Environment to show distributed decision making of a car manufacturer and its supply chain. Each supplier and consumer in the supply chain is represented by an autonomous intelligent agent. The agent has some internal goals and beliefs. It makes independent decisions based on interaction with other agents and inputs from the environment. The agents are developed in G2 and can be deployed on the remote Java Virtual Machine as Java Applications by a simple click and drag operation. The agent can move between G2s or between G2 and Java Virtual Machine in real time.

Multi-Agent Learning in Adaptive Process Control

This application describes a prototype application of multi-agent architecture and reinforcement learning to adaptive control. The application domain is a Plating Line, which deposits layers of different metals on plates to produce electrical connectors. Plates go through a sequence of baths where different metals are layered by electrolytic process. At the end of the process, a controller checks the thickness of the layers of metals and determines discrepancies between set points and actual values. The process is regulated by two control variables: the Baths Rectifier Current and the Line Speed. The objective of the prototype is to demonstrate: (1) A possibility to distribute the problem solving activities of a complex control problem over multiple interactive components. (2) A capacity to learn in a simulated environment and apply the learning in real time to maintain a stable output thickness of the Plates as close as possible to the set point. The prototype adopts: (1) A multi-agent architecture to represent the decision making of the components of the plating process and their interaction. (2) Reinforcement learning as a method through which the agents learn to adapt their behavior by interacting among themselves and with the environment.

Scheduling & Dispatching using Intelligent Distributed Autonomous Agents

The application shows intelligent scheduling and dispatching decisions being made by small intelligent autonomous agents using the market mechanism. Each job and its sub-jobs, and each resource is represented by an agent. The resources are aggregated into manufacturing cells. There are agents for tracking production, generating reports, alarm management, maintenance management, and process planning. The application displays the current load and number of jobs for each resource, number of jobs early, late, or on time, and various other shop floor statistics. The agents in this application are distributed over multiple CPUs.

Information Filtering Agents

The application shows Intelligent Information Filtering Agents developed using the Agent Development Environment. The information filtering agents can be configured and their filtration rules can be described using graphical and spread sheet displays in ADE & G2. The agents can be deployed on the remote nodes as Java Beans. Once the agents are deployed, they can be dynamically monitored, controlled, and rebuild if necessary by ADE.

Network Load balancing

The application shows real-time load balancing by agents distributed over the network. The system consists of multiple nodes (agents) which manage a subset of the network. The node agents performs localize load balancing in their sub-network especially in cases of a node failure or high network loads. In such cases, the node agent redistributes the network traffic among other nodes in the network by negotiating with other agents. The network traffic is redistributed to the least loaded and the closest node(s). The dynamic reallocation results in a leveling of loads and the most efficient processing of all network traffic in the system.

CONCLUSIONS

In this paper, we described an Agent Development Environment for developing distributed multi-agent applications. ADE is an integrated environment to design, develop, debug, simulate and deploy intelligent agents. The behavior of an agent can be described using a graphical language, Grafcets. We then presented a supply chain management application developed for the government software procurement process. The application is extremely important because the U.S. government procurement lead times are notoriously long and the time required for software purchases often exceeds the product lifecycles themselves. Since the U.S. government represents the largest single buyer in the world with an estimated $40B worth of annual software procurement (STSC 1996), any significant improvement in the federal procurement process can effect tremendous savings for the nation.

We designed, developed and integrated three types of agents for the government software procurement process using ADE: (a) government software user, (b) government supply department, and (c) commercial software vendor. Although the proof-of-concept implementation is far from an "industrial strength" application, it satisfies our feasibility goals and suggests that the agent-based approach and ADE technology has the potential to scale well across multiple users, customers and vendors. This represents our primary objective at this early research stage.

We also constructed a simulation model for the paper-based "as is" process that is used today at the Naval Postgraduate School and Gensym Corp. We plan to use simulation to analyze procurement cost and lead time between the paper-based "as is" process and the agent-based "redesigned" process. The simulation results will also be used to adapt and tailor the supply chain agents.

Also, since software as a product is comprised of digital information, the exchange of software-product information and the goods themselves can be performed electronically. Future work in this area can utilize Electronic Data Interchange to exchange products and products information via intelligent autonomous agents, thus further reducing procurement lead times. The agent-based commercial transactions between buyers and sellers can supplant the traditional EDI for business-to-business commerce and can potentially increase speed and responsiveness in today's hypercompetitive business environment.

References

Amulet. Amulet online description. Internet address: http://www.amulet.com (1997).

Ball, G., Ling, D., Kurlander, D., Miller, J., Pugh, D., Skelly, T., Stankosky, A., Thiel, D., Van Dantzich, M. and Wax T. "Lifelike Computer Characters: The Persona Project at Microsoft," in J. Bradshaw (Ed.). *Software Agents*. AAAI Press: Menlo Park, CA (1997).

Boy, G.A. "Software Agents for Cooperative Learning," in J. Bradshaw (Ed.). *Software Agents*. AAAI Press: Menlo Park, CA (1997).

Bradshaw, J.M., Dutfield. S. Benoit, P. and Woolley, J.D. "KAoS: Toward an Industrial-Strength Open Agent Architecture," in J. Bradshaw (Ed.), *Software Agents*. AAAI Press: Menlo Park, CA (1997).

Bui, T. "Intelligent Negotiation Agents for Supporting Internet-based Competitive Procurement," working paper (n.d.).

Bui, T., Jones, C., Sridar, S. and Ludlow, N. "Decision Support for Reconnaissance Using Intelligent Software Agents," Naval Postgraduate School research proposal (n.d.).

Burke, R.D., Hammond, K.J. and Young, B.C. "The FindMe Approach to Assisted Browsing," *IEEE Expert* (July/August 1997), pp. 32-40.

Chavez, A. and Maes, P. "Kasbah: An Agent Marketplace for Buying and Selling Goods," working paper (n.d.).

D'Aveni, R. "Call for Papers on the Topic of Hypercompetition," *Organization Science* (1994).

David, R. and Alla, H. *Petri Nets and Grafcet: Tools for Modeling Discrete Events Systems*. Prentice-Hall International: UK (1992).

David, R. "Grafcet: A Powerful Toll for Specification of Logic Controllers," *IEEE Transactions on Control Systems Technology* 3:3 (September 1995), pp. 253-268.

Etzioni, O. and Weld, D.S. "Intelligent Agents on the Internet: Fact, Fiction, and Forecast," *IEEE Expert* (August 1995), pp. 44-49.

Franklin, S. and Graesser, A. "Is It an Agent or Just a Program? A Taxonomy for Autonomous Agents," in *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages* Springer-Verlag: New York, NY (1996).

Gilbert, D., Aparicio, M., Atkinson, B., Brady, S., Ciccarino, J., Grosof, B., O'Connor, P., Osisek, D., Pritko, S., Spagna, R., and Wilson, L. "IBM Intelligent Agent Strategy," working paper, IBM Corporation (1995).

Insurance. Insurance online description. Internet address: http://www.dmatters.co.uk (1997).

InterAp. "InterAp Assigns Intelligent Agents to the Web," *PCWeek* (12 June 1995).

Johar, H.V. "SoftCord: an Intelligent Agent for Coordination in Software Development Projects," *Decision Support Systems* 20 (1997), pp. 65-81.

Knobloch, C.A. and Ambite, J.L. "Agents for Information Gathering," in J. Bradshaw (Ed.), *Software Agents*. AAAI Press: Menlo Park, CA (1997).

Krulwich, D. *An Agent of Change*. Andersen Consulting Center for Strategic Technology Research (n.d.).

Maes, P. "Agents that Reduce Work and Information Overload," *Communications of the ACM* 37:7 (July 1994), pp. 30-40.

Maes, P. "Pattie Maes on Software Agents: Humanizing the Global Computer," *Internet Computing* (July-August 1997).

Malone, T.W., Grant, K.R., Turbak, F.A., Brobst, S.A and Cohen, M.D. "Intelligent Information-Sharing Systems," *Communications of the ACM* 30:5 (1987), pp. 390-402.

Maturana, F.P. and Norrie, D.H. "Distributed Decision-making Using the Contract Net Within a Mediator Architecture," *Decision Support Systems* 20 (1997), pp. 53-64.

Mullen, T. and Wellman, M.P. “Market-based negotiation for digital library services,” Second USENIX Workshop on Electronic Commerce (November 1996).

Pinson, S., Louca, J.A. and Moraitis, P. "A Distributed Decision Support System for Strategic Planning," *Decision Support Systems* 20 (1997), pp. 35-51.

Porter, M. and Millar. V. 1985.

PriceWatch. PriceWatch online description. Internet address: http://www.pricewatch.com (1997).

Sen, S. "Developing an Automated Distributed Meeting Scheduler," *IEEE Expert* (July/August 1997), pp. 41-45.

Sokol, P. *From EDI to Electronic Commerce: A Business Initiative* McGraw-Hill: New York, NY (1996).

STSC. *Guidelines for Successful Acquisition and Management of Software Intensive Systems* Software Technology Support Center: Hill AFB, UT (1996).

Sycara, K., Pannu, A., Williamson, M. and Zeng, D. "Distributed Intelligent Agents," *IEEE Expert* (December 1996), pp. 36-46.

Sycara, K. and Zeng, D. "Coordination of Multiple Intelligent Software Agents," to appear in *International Journal of Cooperative Information Systems* (1996).

Verity. Verity online description. Internet address: http://www.verity.com (1997).

Walsh, W.E., Wellman, M.P., Wurman, P.R. and MacKie-Mason, J.K. “Some Economics of Market-based Distributed Scheduling, Submitted for publication (1997).

Whitehead, S.D. "Auto-faq: An Experiment in Cyberspace Leveraging,: *Proceedings of the Second International WWW Conference* 1 (1994), pp. 25-38.

Zeng, D. and Sycara, K. "Cooperative Intelligent Software Agents," Carnegie Mellon University technical report no. CMU-RI-TR-95-14 (March 1995).