Redesigning Software Procurement through Intelligent Agents

Mark Nissen, Naval Postgraduate School, 555 Dyer Rd. Code SM/Ni, Monterey, CA 93943; e-mail: MNissen@nps.navy.mil.

Anshu Mehra, Gensym Corporation, 125 Cambridge Park Dr., Cambridge, MA 02140; e-mail: AMehra@gensym.com.

Abstract

Software procurement represents a process of vital importance to most enterprises in the public and private sectors, but particularly in the U.S. government, the software procurement process is highly pathological. Intelligent agent technology offers the potential to remedy this process pathology and an agent-based redesign transformation is recommended to dramatically improve the performance of the software procurement process. This paper outlines a number of contemporary agent applications and presents the results of a measurement-driven redesign analysis of the government software procurement process. We discuss intelligent software procurement agents that are developed to automate and support this process using a novel tool called the Agent Development Environment (ADE). The paper highlights the behavior and utility of intelligent software procurement agents and presents some preliminary results associated with their application to an operational government organization. We show how this agent-based process redesign offers excellent potential for improvement in both cost and cycle time for the process and offer an agenda for continued research along these lines.

Redesigning Software Procurement

With the passage of each product lifecycle, software becomes increasingly important to efficiency and effectiveness in the public and private sectors alike. Many major corporations rely on the power of information technology (IT) to compete effectively in this fast-paced, hypercompetitive, global business environment of today, and the military has long invested in software-intensive weapon and communication systems. With an expanding mission but declining budget, the U.S. government is fast becoming software-dependent as well, as it represents the largest single buyer in the world with an estimated $40B worth of annual software procurement (STSC 1996).

However, U.S. government procurement lead times are notoriously long and the time required for software purchases often exceeds the product lifecycles themselves. It widely employs linear, bureaucratic, paper-based, regulation-laden procurement processes, even when buying the same commercial off-the-shelf (COTS) software used by consumers, for example. Although the government has made considerable progress in terms of procurement over the last few years (e.g., acquisition streamlining, using EDI, electronic bulletin boards, emphasizing COTS software, establishing a preference for commercial specifications and standards), its procurement process will always be disadvantaged with respect to those found in the corporate sector. The immense size of the government organization, requirements for public trust and accountability, socio-economic legislation and other attributes all contribute to this result. In fact, the government must develop *superior* processes just to achieve parity with the private sector in terms of process efficiency and effectiveness.

In this paper, we employ measurement-driven inference to diagnose a number of pathologies associated with the government procurement process and use the results to redesign software procurement through an intelligent agent application. This agent technology serves to enforce process integration between buyer and seller, takes advantage of global connectivity, satisfies regulatory requirements with even greater consistency than is possible today, and automates most of the software procurement process with attendant savings in terms of cost and cycle time. When implemented to augment the government's suite of legacy, current and emerging IT tools, this agent application can effect the kinds of quantum performance improvements sought through process redesign (Davenport 1993). We provide a brief overview of some exemplary intelligent agent applications in the following section and then outline the key steps and results associated with redesign of the software procurement process. The architecture for our intelligent agent application is discussed subsequently, followed by the case of its initial application in the government acquisition domain. We close the paper with an agenda for continued research along these lines.

Intelligent 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 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 software procurement 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 software procurement 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.

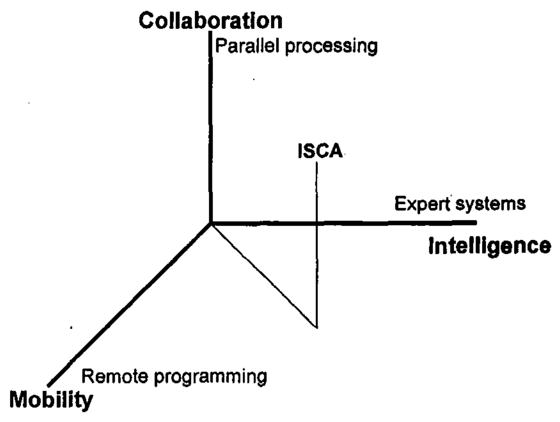

Figure 1 Agent Framework

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.

Notice the annotation for intelligent software procurement 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 software procurement process redesign.

Software Procurement Process Redesign

The government procurement process is costly and time-consuming, particularly with respect to IT. There are many instances of procurements taking so long that purchased software becomes obsolete before it is received by the government user and put into place in the organization, for example. Because the set of processes associated with software procurement appear to be so pathological for the government, we focus our initial redesign activities on this area, and we include the corresponding order-fulfillment processes associated with commercial software vendors to analyze a two-segment supply chain process in this section. We first introduce the integrated supply chain process and then step through the measurement-driven inference associated with its redesign.

Integrated Supply Chain Process

As noted above, two primary processes are involved with this supply chain—government software purchasing and commercial software order fulfillment—and 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 offer 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 2 depicts the integration of the User, NPS Supply Department and software Contractor.

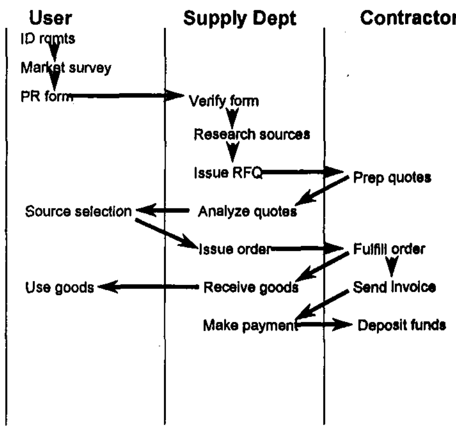

Figure 2 Integrated Supply Chain Process

The 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.

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 kinds of knowledge, expertise and information suggest that an intelligent system may offer good potential to support this integrated process.

Process Redesign

To redesign this integrated process, we employ the measurement-driven method described in Nissen (1996, 1997a, 1997b), which obtains measurements from a graph-based process representation to diagnose pathologies and recommend redesign transformations. The knowledge-based method and technology have been successfully employed to redesign a number of government procurement processes (e.g., see Nissen 1997c), and we highlight each of the key steps below.

Step 1. Model baseline process. In terms of process redesign, the baseline (i.e., "as is" process that exists before redesign) process is the one diagrammed in Figure 2 above. Measurement-driven inference requires a representation that supports automated measurement, so we convert the informal model from above into an attributed directed graph (A-digraph) as shown in Figure 3. With this model, each process activity is represented as a node linked to predecessors and successors by directed edges in the graph. To avoid clutter, only the first seven activities are shown in the figure for this baseline process. For reference, we also list a representative set of attributes associated with the "Research Sources" step. These include, for example, role and cardinality of the agent responsible for performing the activity, along with the corresponding department and organization, inputs to and outputs from the activity, and classes of tools and communications used to support the process and its products. As an A-digraph representation, this model supports automatic process measurement using our graph-based measurement scheme.

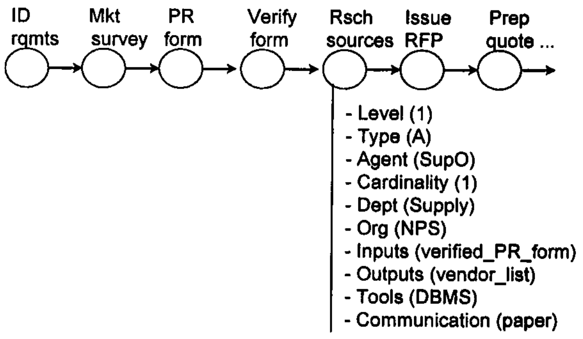

Figure 3 Baseline A-Digraph Representation

Step 2. Measure baseline process. Measurements are obtained by counting certain nodes, edges and attributes that possess heuristic value in terms of process diagnosis. Some of the relevant measures and baseline process values are presented in Table 1 for discussion. Details associated with the measures and measurement process are provided in the references above. From the table, we see that the process is comprised of twelve activities (i.e., size = 12) and, as a sequential process flow, it is twelve steps in length. The parallelism measurement (1.00 is a theoretical minimum for this measure) is calculated as the ratio of size divided by length and quantifies the sequential nature of this baseline process instance. The other measures are expressed in terms of normalized attribute counts. For example, the handoff fraction (0.67) indicates that roughly two thirds of the process activities are associated with time-consuming handoffs from one organizational role to another. Such handoffs are often associated with work sitting in in-boxes and out-boxes, awaiting agent assignment, managerial approval, couriers, mail service and like activities that are well known contributors to cycle time. The three IT fractions are used to quantify the extent to which the process involves IT-based support, communication and automation, respectively. Zero represents a theoretical minimum for these measures.

Step 3. Diagnose process pathologies. The measurements presented in Table 1 offer heuristic value in terms of process diagnosis, and the diagnostic implications of these measures are also shown in the table as they pertain to process pathologies. For example, we noted above that process friction has a well known impact on process cycle time, as do sequential process flows and paper-based communications. Likewise, manual, labor-intensive processes are also well known to suffer from relatively high cost. These diagnosed pathologies are used in turn to match redesign transformations.

Table 1 Baseline Process Measurements

Measure	Value	Diagnosis
Process size	12	
Process length	12	
Parallelism	1.00*	Sequential process
Handoffs fraction	0.67	Process friction
IT-Support fraction	0.25	Manual process
IT-Communication fr.	0.00*	Paper-based process
IT-Automation fr.	0.00*	Labor-intensive process

Step 4. Match redesign transformations. Several redesign transformations are matched with the pathologies diagnosed in the preceding step. For example, the sequential process flow matches with a de-linearize transformation (i.e., performing some process steps in parallel as opposed to serially), but a check of inputs and outputs for each step precludes the activities of this process from being performed with greater concurrency; that is, the output from each step is required as an input to the next, so de-linearization is infeasible. Alternatively, the combination of process friction with a manual, paper-based process flow matches with a workflow-system transformation to support the process activities. Further, the labor-intensive process (i.e., absence of automation) matches with the diagnoses above to suggest that an intelligent agent transformation may augment the workflow system to further enhance process performance. Clearly many other redesign recommendations can be made from these measurements and diagnoses, but these are directly applicable to the present discussion (i.e., intelligent agent transformation of software procurement). We now discuss the architecture associated with our intelligent procurement agent transformation.

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. We begin with a high-level architectural schema that inter-relates each of these ADE components. This is diagrammed in Figure 4.

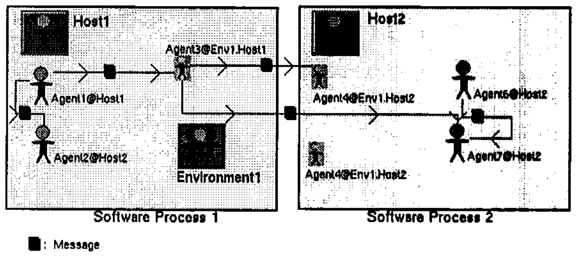

Figure 4 ADE Architectural Schema

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, ManufacturingCellAgent 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. *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*.

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, (vi) a deployment center to deploy agents in the G2 environment or as "JavaBeans" in Java Virtual Machines.

Redesign Agents and Results

In this section we present the preliminary results from our use of ADE and application of intelligent software procurement agents to redesign the government software procurement process. We briefly describe the structure and behavior of the intelligent software procurement agents developed to perform in this environment and discuss the performance implications of this agent-based process transformation.

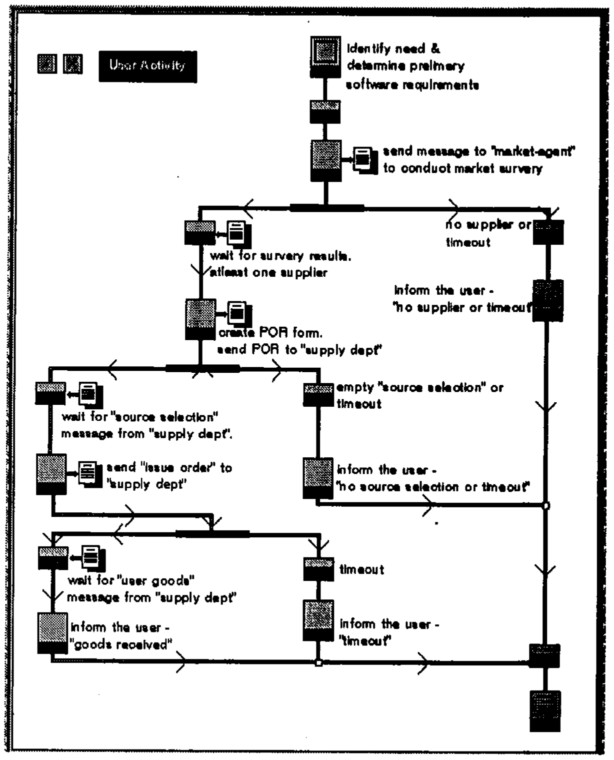
User Activity
Identify need & determine prelimery software requirements
send message to "market-agent" to conduct market survery
no supplier or timeout
wait for survery results. atleast one supplier
inform the user - "no supplier or timeout"
create POR form. send POR to "supply dept"
empty "source selection" or timeout
wait for "source selection" message from "supply dept".
send "issue order" to "supply dept"
inform the user - "no source selection or timeout"
wait for "user goods" message from "supply dept"
timeout
inform the user - "goods received"
inform the user - "timeout"

Figure 5 Grafcet for User Behavior

Agent Structure and Behavior

In order to describe the structure and behavior of the intelligent software procurement 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 5, depicts the user behavior and maps homomorphically to the integrated process flow from Figures 2 and 3. 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.

The corresponding Grafcet for the Supply Department is presented in Figure 6, 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 2 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.

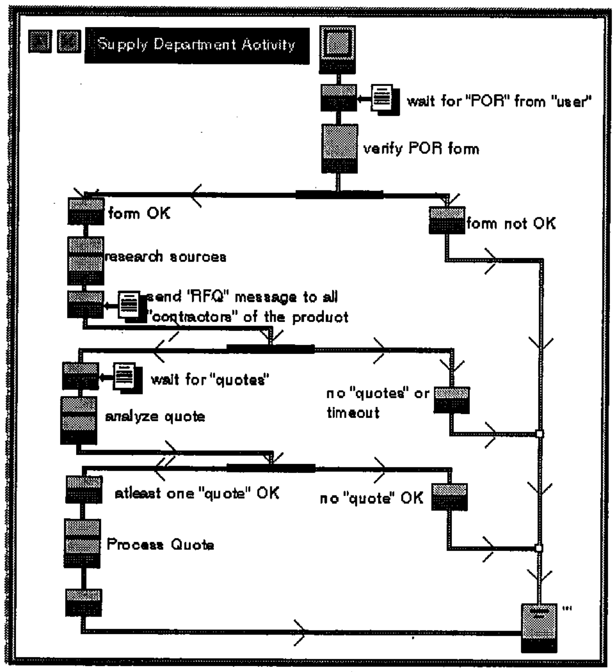

Figure 6 Grafcet for Supply Department Behavior

The software contractor behavior is specified through the Grafcet presented in Figure 7. 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 5-7. 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.

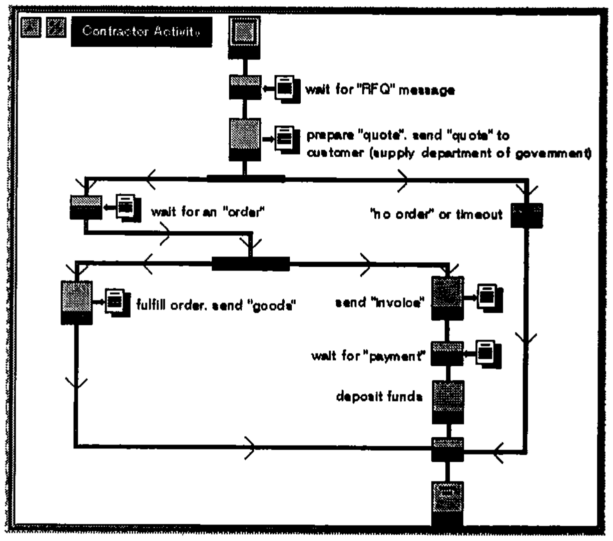

Figure 7 Grafcet for Software Contractor Behavior

Preliminary Redesign Results

We note the preliminary nature of these redesign results, for the work to date has taken us only through a proof-of-concept demonstration of the integrated supply chain agents. Although we have obtained measurements from this agent-based, post-redesign process configuration, we have yet to simulate the process performance or measure the implemented redesign through pilot or like testing. Nonetheless, even these preliminary results are very encouraging and suggest continued work along these lines. The post-redesign measurements are presented in Table 2, along with the corresponding baseline process measurements from above for reference.

From the table one can observe the agent-based redesign transformation has no effect on process size, length or parallelism, nor did we predict that it would. Thus, we see no performance implications with respect to these measures. In contrast, the redesign has a tremendous effect on the remaining four measures—handoffs, IT-Support, IT-Communication and IT-Automation fractions—which are associated with reduced cost and cycle time as shown in the table. Although we have yet to simulate or pilot this agent-based redesign, results such as these (e.g., eliminating process handoffs and dramatically increasing the density of IT employed in the process) offer excellent potential to generate the kinds of performance effects noted in the table. With this, we feel confident that the intelligent agents developed through this research can be employed to dramatically improve performance, not only of this software procurement process, but many software and other procurement processes in like organizations across a wide range of industries and sectors. This leads to our agenda for continued research along these lines.

Table 2 Redesign Process Measurements

Measure	Baseline	Redesign	Performance Implication
Process size	12	12	
Process length	12	12	
Parallelism	1.00*	1.00*	
Handoffs fraction	0.67	0.00*	- cycle time
IT-Support fraction	0.25	0.92	- cost
IT-Comm. fr.	0.00*	0.83	- cycle time
IT-Automation fr.	0.00*	0.67	- cost

Continued Research

In this paper, we redesigned a software procurement process and described the Agent Development Environment for developing distributed multi-agent applications. 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. Although the proof-of-concept implementation is far from an "industrial strength" application, it satisfies our feasibility goals and suggests the agent-based approach and ADE technology have potential to scale well across multiple users, customers and vendors. And our measurements suggest excellent potential for dramatic performance gains through reduced process cost and cycle time. This satisfies our primary objective for this early research stage.

We are now constructing a simulation model for the integrated supply chain process that is used at the Naval Postgraduate School and Gensym Corp. We plan to use simulation to analyze and compare procurement costs and lead times for 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 EDI and the Web to exchange products and product 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.).

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).

Nissen, M.E. "Measurement-Driven Inference for Reengineering Support"; appeared in the Workshop of AI in Business Working Notes from the AAAI 96 Conference (1996).

Nissen, M.E. "Toward Intelligent Web-based Redesign Support," AAAI Technical Report WS-97-02 (1997a).

Nissen, M.E. "Reengineering Support through Measurement-Driven Inference," *Intelligent Systems in Accounting, Finance and Management* Vol 6 (1997b).

Nissen, M.E. "Reengineering the RFP Process through Knowledge-Based Systems," *Acquisition Review Quarterly* (Winter 1997); also published in the Acquisition Review Quarterly World Wide Web electronic library: http://www.dsmc.dsm.mil/pubs/arq/97arq/nissen.pdf (1997c).

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).