Toward an Intelligent Agent Framework for Enterprise Integration

Jeff Y-C Pan and Jay M. Tenenbaum

Enterprise Integration Technologies Corporation
459 Hamilton Ave., Palo Alto, CA 94301
and
Center for Integrated Systems, Stanford University
pan@cis.stanford.edu; marty@cis.stanford.edu

Abstract [1]

We propose a software framework for integrating people and computer systems in large, geographically dispersed manufacturing enterprises. Underlying the framework is an enterprise model that is built by dividing complex business processes into elementary tasks or activities. Each such task is then modeled in cognitive terms (e.g., what to look for, what to do, who to tell), and entrusted to an Intelligent Agent (IA) for execution. The IAs interact with each other directly via a message bus, or through a shared, distributed knowledge base. They can also interact with humans through personal assistants (PAs), a special type of IA that knows how to communicate with people through multi-media interfaces. Preliminary experimental results suggest that this model-based, man-machine approach provides a viable path for applying DAI to real-world enterprises.

1 Introduction

We are creating a software framework for integrating people and computer systems in large, geographically dispersed manufacturing enterprises. It is based on a vision of augmenting human workers with a large number of computerized assistants, known as intelligent agents, or IAs. Each IA supports a clearly discernible task or job function, automating what it can and calling on the services of other IAs, as well as human beings, when necessary. IAs can interact directly via a dedicated message bus, the IA Network, or through a shared knowledge-base, the MKS knowledge service[1,2], as illustrated in Figure 1.

At the core of MKS is a comprehensive object-oriented model of the enterprise and how it functions. The MKS model includes descriptions of personnel, facilities, equipment, inventory, manufacturing processes, and other corporate assets. It also captures the flow of information, decisions and materials through the enterprise – how things get done. The model is wired into the enterprise's information infrastructure (databases, CIM systems, accounting systems etc.) so that it continuously reflects the actual state of the enterprise. Agents interact with these information resources through the model via high-level service protocols that insulate them from details such as where information resides. They can also register with the model their interest in particular events, and be notified when they occur. MKS thus serves agents as a repository for shared knowledge, and a center for information exchange.

IAs model the perceptual, reasoning, action and communication skills involved in performing human job functions, such as those of an equipment operator, production scheduler, engineer, purchasing agent or manager. IA's must therefore know what to look for, what to do when they see it, and who to tell, just like the person they model. Interactions among IAs follow the established corporate lines of communication and procedures. Collectively, IAs and their communication links form an active, operational model of an enterprise.

To participate in this society of agents, humans require the services of personal assistants (or PAs). PAs belong to a special class of IAs that know how to

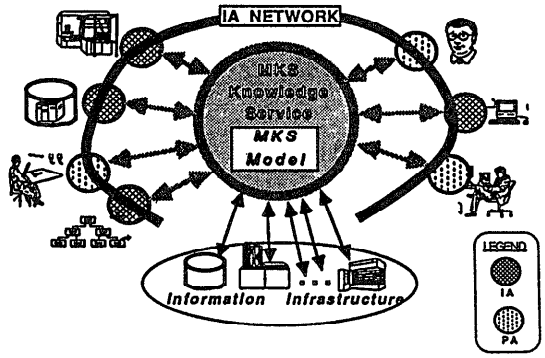

Figure 1: An IA-based Enterprise Integration Framework

[1]This work was supported by the Defense Advanced Research Projects Agency under contracts N00014-87-K-0729 and N00014-90-J-4016.

communicate with humans through E-mail, graphical editors, and other standard modalities (e.g., beepers, faxes, telephone). They also know how to translate between these human modalities and IA network protocols. To facilitate the interaction, one's PA maintains a personal model that includes where they currently are, how they can be reached, what information they would like monitored through the knowledge service, and what actions to take, either in response to messages from IAs or notifications from the knowledge service. For example, reports of minor trouble might be forwarded automatically to a human subordinate or an IA, whereas a report of major trouble might warrant being paged. Moreover, one's PA model can include personal preferences for supporting information to be presented in conjunction with decision requests. Such an architecture supports the notion of man/machine cooperative work, as articulated by Winograd [3]. Indeed, it goes further in allowing tasks gradually to be handed off to IAs so that ultimately they can be performed interchangeably by a person or their agent.

In addition to supporting traditional human activities, IAs can automate a variety of tasks for which people may be too expensive or otherwise limited (e.g., too slow, too inattentive). Think of IAs as an inexhaustible source of "cheap labor" that can provide dedicated human-like services such as watching over each piece of equipment in a factory and shepherding each workpiece and piece of paperwork along their prescribed routes. IAs can also integrate software in interesting ways, such as by automating human job functions that primarily involve transferring information between several disjoint systems, or by serving as intelligent front ends that make existing software more useful and easier to use.

For the framework to succeed in an operational setting, we believe it is essential that the enterprise models underlying IAs, PAs and MKS be created and maintained by the people they serve – the end-users that understand best the tasks to be modeled. Two features of our approach help make this possible. First, partitioning complex activities into simple tasks and modeling them in cognitive terms produces models that are familiar to workers, and therefore easy for them to understand. Second, we are developing simple modeling tools for end-users, that enable them to copy and customize generic activity models (e.g., for monitoring, transactions, brokering) from a library. Customization might require editing a script or fleshing out a decision tree, but such skills can be quickly mastered when the editing tools support representations that are already familiar[4,5].

In the remainder of this paper, we flesh out this vision and describe a prototype implementation, currently under development, that will run the semiconductor fabrication facility at Stanford's Center for Integrated Systems (CIS). Sections 2 and 3 provide technical details on the framework and its implementation. Section 4 reports on preliminary experiments with the prototype at CIS. Finally, Section 5 summarizes our results to date and draws implications for both distributed AI and enterprise integration. For a fuller treatment of these issues, see [6].

2 The IA Framework

2.1 Technical Overview

The IA framework shown in Figure 1 consists of three synergistic technologies: intelligent agents, the MKS knowledge service, and a distributed system infrastructure (not shown). IAs interact with humans through their Personal Assistants (PAs) over an enterprise-wide network using standard service protocols. IAs can also access information resources, control on-line equipment, and trigger other IAs indirectly through the MKS knowledge service[2].

Each computerized agent is an active, autonomous process that models a single discernible task. The task may be either one traditionally performed by a human or one intended specifically for a computerized agent (e.g., tracking a wafer through a process). We model activities in cognitive terms that make the models easy for our users to understand and maintain. Activities are described by corresponding "activity" objects in the MKS model, in terms of the information they consume, process and produce, and the other MKS model objects (people, equipment, wafer lots and so forth) that participate. From such descriptions, IAs can determine what information and events to monitor and how to respond.

An IA framework for an enterprise is built in two stages. First, enterprise activities are modeled in cognitive terms and added to the MKS model as Activity Objects. Second, these activity models are selectively *activated* as IAs, to actually perform the modeled tasks. We shall now cover each of these steps in some detail.

2.2 Activity Modeling

Following the object-oriented methodology of MKS, individual activities are modeled by customizing generic activity models from the MKS library. The library of activity objects is organized as a classification hierarchy according to the nature of the work performed.

Each activity object contains the knowledge necessary for performing a specific job (i.e., the modeled activity). The knowledge is provided in the form of a template that defines the essential elements necessary to construct a cognitive model for performing the task: the players, capabilities, states, and sensory inputs, and the core model consisting of message and sensory input patterns, sets of actions, and *pattern⇒action* rules (i.e., reasoning) that associate them. **Players** describe the parties involved in an activity, and are specified generically where possible (eg., a job role vs. a specific person); **Capabilities** specify the tasks an agent is capable of performing, with applicable "cost" to use when bidding for jobs; **States** are the natural contexts that people use in deciding what sensory patterns and actions are appropriate. (Perceptions and actions can also be conditioned on goals, resources and other cognitive concepts, where appropriate.); **Sensory inputs** designate the sources of data to monitor and the specific patterns to look for (as functions of state); **Core Model** defines the IA's behavior in terms of sets of pattern-action rules, indexed by current state. (Sensory patterns define what to look for, Message patterns define what IA messages are meaningful in the context of a particular activity, and Action Sets detail the sequences of actions to accomplish some desirable results.)

While Pattern⇒Action rules are the main form of reasoning model used in our experiments, other intuitive behavioral representations can be used where they contribute to ease of expression and comprehension. Examples include state transition diagrams, petri nets, flow charts, decision trees, and scripts. The choice depends on what is most natural for the task at hand.

Following object-oriented design practice, models of specific activities can be generalized to create reusable libraries of objects that model generic job roles. Accordingly, we have begun building a comprehensive library of generic activity models for semiconductor manufacturing, together with supporting libraries of basic methods for sensing, reasoning, communicating, and taking action. We are also developing activity modeling tools analogous to the MKS modeling tools for processes, equipment and the like. They will make it possible to select an activity object from a library that has been predefined for common high-level tasks (e.g., equipment control) and customize it by selecting and composing desired sensing, data processing, decision making, and action behaviors from the method libraries. Libraries and CASE tools thus effectively raise the level of vocabulary used for modeling.

2.3 Intelligent Agents

In the following subsections, we first discuss the *agentification* process by which a passive activity object is transformed into an active IA, and then elaborate on the process by which IAs communicate with each other.

2.3.1 Agentification of IAs

Agentification refers to the three steps involved in creating a computational process(es) (i.e. an IA) whose behavior mimics that described in a corresponding activity object. First, a unique instance of an activity object is created from the MKS library – a surrogate for a particular agent performing a particular task. Second, one or more computational processes are created to implement the "autonomous" agent's sensors and behavior. Finally, these processes are activated so that they can begin receiving and responding to sensory data and incoming IA messages. We will now elaborate on each of these steps.

The **Instantiation** step involves customizing an object template copied from the MKS activities library for the specific task at hand. Normally, this involves filling in situation-specific information for Players and other slots in the template. (e.g., The operator of the Tylan Furnace is Mary).

The **Process Creation** step initiates computational processes that efficiently implement the prescribed behavior in a given runtime environment. Think of these processes as interpreters that translate the concise behavioral descriptions provided by activity objects into the best possible runtime implementations. For example, suppose that an activity model calls for periodically monitoring a data source. Under MKS, such a requirement can be translated into a simple registration of interest with the notification mechanism [1]. The process can then remain suspended until a change

to the subject data is reported to the MKS model. If such a mechanism is not available, a process would be created to periodically sample the data source in response to clock interrupts. The intent, in either case, is to insulate the model builder from implementation details. To maximize flexibility, separate processes are established for an IA's reasoning engine and each of its primary sensory and communication inputs. Each such receptor process is then "trigger-wired" to its corresponding data source in the most computationally parsimonious fashion. The IA network is now ready for activation.

The final **Activation** step arms the sensor processes so that they "watch" their assigned data sources. Simultaneously, all the IA-message listeners and transmitters are switched on so that IAs can exchange requests and inquiries through the IA network. The process implementing the reasoning engine is placed in a continual stimulus-response mode. When a "stimulus" arrives, either as a symptom detected by one of the IA's sensory sub-processes, or through a message received from another IA, the reasoning process is invoked to generate the proper responses and actions, following the decision procedure (e.g., production-rule, state transition diagram, decision tree) contained in the activity model.

2.3.2 The IA Network

The IA Network is a *logical* communication bus, designed exclusively for exchanging messages among IAs in a special format, known as the *IA protocol*. An *IA message*, based on the IA protocol, allows an IA to report to or request services from other IAs. Even though IA messages may be broadcast over the same physical network as other logical communication protocols (e.g., the MKS protocol, over an Ethernet), their high-level semantics provide concise, natural, and comprehensive communication among IAs. Describing communications among IAs in this high-level vocabulary also insulates an enterprise's activity model from the implementation details of its network infrastructure.

An IA-message consists of four parts: 1). the message type, specifying the kind of communication pattern to be engaged in; 2). a "target" description, addressing the intended IA(s) either directly by name(s), or indirectly by role, interest, or capability/qualification; 3). the "body" of the message which is to be sent to the targeted IA(s) and interpreted within its context; and 4). an optional list of keyword arguments detailing interactions (e.g., what to do should a message fail to find its targeted IA within a specific time limit).

There are four types of IA messages that are currently adopted for our IA system: REQUEST, INFORM, INQUIRY, and BID. A **REQUEST** message is used to issue commands to an IA, resulting in desired actions. An **INFORM** message is a special type of request, intended primarily for passing textual information to an IA(s), for forwarding to the most appropriate person. The receiving IA gets to decide who is the most appropriate person in its own context (e.g., Mary, the equipment operator on duty). It can then forward the message to that person's PA, which may decide, for example, to send him/her an E-mail with the message as its contents. An **INQUIRY** message is designed to acquire information through another IA. On sending this type of message, an IA will be suspended until the expected information becomes available. A special feature for this type of message is that default behaviors for an aborted situation (e.g., :if timed out or :if-rejected) will terminate the suspension - a precaution to prevent permanent "hanging" of the IA due to an inquiry that cannot be satisfied. Finally, a **BID** message encompasses three stages of behavior: 1). broadcasting messages to a set of targeted IAs inviting them to submit bids to supply service, with associated costs; 2). evaluating all bids received within the :max-response-time and selecting the "winning" bid according to the :cost-function; 3). sending out a request message with the winning bidder as the targeted IA.

The IA message format allows targeted IA(s) to be described by their names, by their roles (including their interests), or by their capabilities. *Call by role* requires that the targeted IA be determined dynamically in the context of the sending IA. *Call by capability*, on the other hand, requires that a generalized pattern specifying a *qualification* be broadcast and interpreted by all IAs within the specified broadcast scope.

2.3.3 Personal Assistants

As discussed earlier, Personal Assistants are a special class of IA distinguished by their ability to communicate with people as well as with other IAs. They *encapsulate* individuals, enabling them to interact with other IAs using the network protocols and to act as their own (i.e., living) activity model.

Each PA maintains a personal activity model of the individual it serves. One's PA model would include the following information about them: where they are; how to contact them *(e.g., E-mail, fax, phone, X-terminal, pager)* their capabilities (a list of tasks

they are qualified to perform); their responsibilities (a list of tasks currently assigned to them); IAs supporting their current tasks; their general and task-specific information needs and interests; their preferences for how information should be presented; an activity model for personal tasks.

Using this information, one's PA can perform a variety of services on their behalf. For example, it can monitor incoming information for them 24 hours a day (e.g., read their E-mail buffer, messages from their IAs, or notifications from the MKS knowledge service). Routine events can be handled autonomously, such as by dispatching them to a human subordinate or an IA, while important issues and information are brought immediately to their attention.

3 Implementation

Our experiments with IAs are being conducted at Stanford's CIS, whose fabrication line provides an accessible real-world manufacturing environment. The IA architecture is being implemented as an extension to the MKS framework [1] , by adding a sub-hierarchy to the MKS model taxonomy that includes the activity library and instantiated activity objects. IAs also use the MKS knowledge service [2] to access real-time, manufacturing data from the CIS fabrication line.

Like MKS, the IA architecture is being prototyped in HyperClass, an object-oriented programming environment [7] implemented on Lucid CommonLisp version 3.0. A distinctive feature of HyperClass is MetaClass, a toolkit for rapidly constructing customized interactive graphical editors. MetaClass is being used extensively to create specialized editors for building IA activity models. Additionally, it helps us prototype the graphical user interfaces by which PAs interact with their human masters. The multi-tasking capability of Lucid CommonLisp 3.0, provides an easy way to implement the multiple autonomous processes required for IAs. While all experiments, to date, have been done on Sun4 workstations, the system can be readily ported to other hardware platforms (e.g., DEC Stations 3100 and 5000) running Lucid 3.0.

In our initial prototype, all IAs are restricted to run within a single workstation, though they can remotely access the MKS model and enterprise-wide information through the distributed MKS knowledge service protocols [2]. However, future generations of the system must be fully distributed, so that IAs can live in a variety of geographically dispersed workstation environments (C++, Lisp, Unix, VMS and so forth). A distributed infrastructure, itself based on low-level agents known as *Proxies*, is being developed for this purpose [8].

4 An IA-run Enterprise

We shall now illustrate, with our ongoing experiments at CIS, how Intelligent Agents can be used to model and run an enterprise. Additional experiments using the framework to coordinate design and manufacturing decisions for concurrent engineering are reported in Brown[9].

The following scenario is a slightly dramatized version of currently running code, focusing on a few generic tasks such as routing wafers through processing steps, assigning equipment, and monitoring for equipment malfunctions. The action begins when the operations manager starts a new wafer lot. At the same time that a wafer lot is started in the fab-line, a "shadowing" wafer-lot *genie* is created in the IA world. This wafer-lot IA is entrusted with moving the lot expeditiously along the routes defined in the process recipe, making sure it is on schedule and receiving its fair share of resources. At each process step, it performs dynamic equipment assignment by selecting the most suitable equipment (e.g., capable and least busy) among all available machines in the fab-line [2]. The wafer-lot IA subsequently sends an IA-message to the chosen equipment's operator IA, requesting that the lot be added to the incoming waiting queue of the equipment. The wafer-lot IA then switches itself to a "holding-in-queue" state where it awaits potential abnormal reports from other IAs (e.g., the equipment's IA reporting that the machine is being shut down). It also wakes up periodically to make sure that its job request is not unfairly stalled in the equipment's queue.

From a production perspective, each piece of equipment has an *operator IA* that maintains a waiting queue for incoming lots and is responsible for keeping its machine running at peak efficiency. Whenever the equipment is in an "idling" state, its operator IA will attempt to select a wafer lot from the waiting queue following a given prioritization rule, and instruct the equipment to load and process the lot.

Each piece of equipment also has a monitor IA that emulates a technician watching for anomalous sensor

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.