98

SAND--98-8542C
CONF-980726--

Please include the following Notice at the bottom of the first page in addition to the ~~as part of the Acknowledgment section.~~ statement of DoE support. JPS

MASTER

DISCLAIMER

The Use of Software Agents and Distributed Objects to Integrate Enterprises: Compatible or Competing Technologies?

Carmen M. Pancerella (carmen@sandia.gov)
P.O. Box 969, Mailstop 9012, Sandia National Laboratories, Livermore, CA 94551-0969†

ABSTRACT

Distributed object and software agent technologies are two integration methods for connecting enterprises. The two technologies have overlapping goals — interoperability and architectural support for integrating software components — though to date little or no integration of the two technologies has been made at the enterprise level. The primary difference between these two technologies is that distributed object technologies focus on the problems inherent in connecting distributed heterogeneous systems whereas software agent technologies focus on the problems involved with coordination and knowledge exchange across domain boundaries.

This paper addresses the integration of these technologies in support of enterprise integration across organizational and geographic boundaries. We discuss enterprise integration issues, review our experiences with both technologies, and make recommendations for future work. Neither technology is a panacea. Good software engineering techniques must be applied to integrate an enterprise because scalability and a distributed software development team are realities.

1 INTRODUCTION

An enterprise encompasses heterogeneous computers communicating on a wide-area network, often across organizational boundaries. An enterprise, whether it be financial, health care, manufacturing, or engineering, must be agile. Agility refers to rapid response to changes in resources, processes, requirements, and technology while at the same time providing cost-effectiveness, reduced cycle times, and high quality and accuracy. A robust software architecture is key to achieving agility.

Given the requirement for agility and the need to connect an extended enterprise, there are demands placed on the enterprise software architecture. Specifically, the software must support a variety of platforms, operating systems, and programming languages. It must support rapid and easy customization, integration, and reconfiguration. The architecture must allow legacy software to be integrated. The software must be easy to deploy throughout geographically distributed facilities. Intelligent software components must be developed to manage the various resources in the enterprise. Finally, knowledge must be represented in a formal and unambiguous manner and exchanged between various domains and organizations.

We first discuss background technologies. In Section 2 we propose enterprise integration issues and strategies. In Sections 3 and 4 we present our experiences with integration using distributed objects and software agents, respectively. In Section 5 we discuss the use of Java to integrate enterprises. In Section 6 we evaluate the approaches for enterprise integration. In Section 7 we discuss future research and development necessary to build enterprise computing systems.

1.1 Distributed Object Technology

Distributed object technology [1] allows computing systems to be integrated such that objects work together across machine and network boundaries. Examples include CORBA[2], OLE[3], and OpenDoc[4]. A distributed object is a reusable, self-contained piece of software that can be combined with other objects in a plug-and-play fashion to build distributed systems. A client object makes requests of a server object, and the operation proceeds unaffected by their respective locations. A distributed object has a well-defined interface, describing the data and functionality it exposes to other objects.

1.2 Common Object Request Broker Architecture (CORBA)

CORBA [2] is an industry standard for building distributed, heterogeneous, object-oriented applications. CORBA is specified by the *Object Management Group* (*OMG*), a consortium. It is the most common standard for the deployment of wide-area distributed objects today. It is open, robust, heterogeneous, interoperable, multi-platform, and multi-vendor supported.

The Interface Definition Language (*IDL*) is used to define interfaces in CORBA. An IDL interface describes the data types and methods that a server provides for an implementation of an object. IDL is not a programming language, but rather a language that describes interfaces. Mappings from IDL to programming languages, *e.g.*, C, C++, and Java, are specified.

---

† This work was supported by Sandia Corporation under Contract No. DE-AC04-94-AL85000 with the U.S. Department of Energy.

The *Object Request Broker (ORB)* is the communication hub for all objects in the system; it provides the basic object interaction capabilities necessary for components to communicate. The *Internet Inter-ORB Protocol (IIOP)* is an open Internet protocol for connecting large distributed applications. Specifically, it provides for ORB-to-ORB communication built on top of TCP/IP. IIOP is scalable from the LAN to the Internet. Using IIOP allows developers to select any ORB that supports this protocol, and the resulting distributed objects can interoperate.

1.3 Software Agents

There are a number of definitions of *software agents* [5]. In the context of this paper, an *agent* is an autonomous, persistent, encapsulated software component that communicates with other agents using an agent communication language in order to accomplish tasks. We expand on each of these properties:

- *Autonomous*: Each agent operates independently and asynchronously and interacts with other agents on a peer-to-peer level, and not a strictly client-server communication structure.
- *Persistent*: Each agent maintains its own state, which is changing over the lifetime of the agent. If the agent goes off-line, there will be some method of storing any agent messages until the agent returns.
- *Encapsulated*: Agents serve as containers for a collection of knowledge representing some functionality. This knowledge is only *accessible via communication in the appropriate agent communication language.*
- *Agent communication language*: An agent communication language possesses formally defined syntax and semantics and can be unambiguously represented in machine-readable format.

1.4 Agent Communication Languages

ARPA has proposed an agent communication language. This language is comprised of three parts: a messaging language and protocol, the *Knowledge Query and Manipulation Language (KQML)* [6], for the transfer of messages; a content language, *Knowledge Interchange Format (KIF)* [7], based on formal logic and predicate calculus; and domain *ontologies* [8], vocabularies and formal relationships among entities in the ontology.

KQML is a performative-based language based on speech act theory. The language specifies three layers: *content*, *message*, and *communication*. The content layer is independent of KQML. The message layer is a speech act performative, which specifies the protocol for agent communication. Examples of KQML performatives include `tell`, `ask`, `reply`, and `subscribe`. The communication layer specifies a set of features describing lower level communication parameters, such as `sender`, `receiver`, and `message ID`. In addition, many agent systems built with KQML specify a content-independent message *router* and a *facilitator*. KQML is extensible, allowing users to define new performatives as the need arises.

KQML has been used independently of KIF to develop multi-agent systems. *PDES/STEP (Product Data Exchange using STEP/Standard for the Exchange of Product model data)* [9-10] has been used as a content language for agent-based engineering [11]. *EXPRESS* is a language specified in the PDES/STEP standard.

The *Foundation for Intelligent Physical Agents (FIPA)* [12] is an organization established to promote the development of specifications of generic agent technologies that maximize interoperability within and across agent applications. FIPA is developing a standard agent communication language. The work in this paper work precedes FIPA.

KQML does not have the low-level technology for enabling distributed collaboration among heterogeneous software components in a wide-area environment, though it can be used on top of a distributed object technology. One of FIPA's goals is to integrate with existing infrastructures, such as CORBA.

1.5 Knowledge Representation and Knowledge Sharing

An important element in any distributed enterprise system is the mechanism used to represent domain information. For example, successful integration of multiple engineers and their tools in a manufacturing enterprise requires the entire enterprise to share a common model of relevant product and domain data. Inconsistencies or discrepancies between the different information models can severely degrade overall system performance and hinder scalability. In the manufacturing domain, ambiguities in the product data exchanged between the designer and manufacturing can result in a non-functioning product and necessitate another development cycle.

Realization of an integrated enterprise requires the exchange of knowledge between the entities and services in the enterprise. Because services may be remotely located from the users, the mechanism used to exchange knowledge must provide reliable, distributed exchange of machine-readable information. Collaborative engineering systems have been developed with autonomous software agents encapsulating the individual system components and communicating with a formal agent-communication language [13-17]. Such agent-based architectures facilitate the integration of engineering enterprises.

2 SOFTWARE DEVELOPMENT FOR ENTERPRISE INTEGRATION

We propose a number of software integration issues that should be considered when developing an enterprise architecture. Once these issues are considered, we advocate the need for an integration strategy or well-defined architecture so that adding, deleting, or updating components and/or agents does not disrupt the entire enterprise.

2.1 Integration Issues

We identify a number of software integration issues. Though all of the following are not relevant to every enterprise, we suggest that this list be reviewed prior to making software development decisions.

- *Object-oriented abstractions*: An object-oriented abstraction of a software component reduces complexity and time-to-market since interfaces are well-defined and new modules can be plugged and played into the architecture. An object-oriented abstraction also provides software reuse of common services.
- *Heterogeneity*: Most enterprises will be heterogeneous with respect to hardware platforms, operating systems, and programming languages.
- *Ease of development*: There is always a need for the enterprise to grow, such that new software can be developed and integrated into the enterprise. It should also be easy for a software development team, working across both geographic and organizational boundaries to work together.
- *Support for distributed applications*: The enterprise is, by definition, distributed. Data and knowledge will be exchanged across the enterprise. It is necessary that support for communicating distributed components is provided and that the computer network itself is hidden from software developers.
- *Interoperability*: Interoperability defines the ability for two software components (objects or agents) on heterogeneous machines to read the data that is exchanged on the network.
- *Extensibility and scalability*: The enterprise is constantly growing to include new software, new partners, and new computer systems. The software integration architecture must be flexible enough to adapt to this. Further, as the size of the enterprise grows, the software architecture must scale and should not collapse.
- *Security*: Since the enterprise is extended to include partners outside of organizations, the Internet will often be used to transport data. Computer security must be present in several forms. First of all, users of enterprise tools may have to be authenticated. Second, data might be proprietary or sensitive in nature, and hence, to transport this data over the Internet, it must be encrypted.
- *Maturity of technology*: The underlying infrastructure should be mature enough so that changes made to the low-level infrastructure have little or no effect on the applications in the enterprise. Maturity of technology is a possible guarantee that the underlying technology will not go away unsupported, at least not until something replaces it. By considering technology maturity the number of problems encountered with upgrades and the likelihood of the enterprise collapsing because of an immature foundation will be reduced.
- *Standards compliance*: By complying to standards or defacto standards at the middleware level, it makes it easy to plug-and-play applications from other sources and to purchase applications that comply to the same standard.
- *Integration with software component software*: It may be important to integrate with component software, e.g., OLE, CORBA, and DCE, so that desktop applications and existing Unix infrastructures are available for use in the enterprise.
- *Cost*: There are many costs associated with enterprise integration: the cost of purchasing software, the cost of developing in-house software, the cost of integrating software, the cost of software maintenance, and the cost of re-doing the enterprise if a poor software design decision is made. These costs must be evaluated.
- *Agent communication protocol*: When there are many agents (or objects) in a distributed system, there should be some agent (or object) protocol for how these distributed components communicate.
- *Coordination*: Coordination refers to the process of multiple agents (or objects) communicating to accomplish a goal.
- *Semantic unification*: Semantic unification means that two agents (or objects) both agree upon the meaning of the data exchanged. Without an agreed upon vocabulary, potential problems arise.

2.2 Integration Strategies

Neither distributed object technologies nor software agent communication languages alleviate the need for an integration strategy or *framework*. CORBA and agent communication languages are only tools, and still the software development in the enterprise must be designed and developed with an integration strategy.

Without an integration strategy, tools, data and applications are integrated into the enterprise in an ad-hoc fashion. Ad-hoc integration leads to the following problems:

- The resulting system is not scalable and can, in fact, collapse at some point.
- There is an N^2 problem, where N is the number of applications in the system. Adding one new application often requires N other applications to change to accommodate the new application. The enterprise itself is very brittle.
- Integration and software problems are solved as they arise, which can result in a duplication of effort since each software developer in the enterprise may have to write code which solves the same problem.
- It becomes nearly impossible to plug in new applications.

There are a number of integration architectures [18] and hybrids of architectures. A *custom integration* is designed in terms of applications-specific APIs, where each application/component has a unique API. Custom integration architectures are often ad-hoc, in that there is no focus on the enterprise. In our agent-based engineering system presented in Section 4, we used a custom integration strategy where the interface to each agent or service in the system was specifically defined. In our the CORBA-based manufacturing cell presented in Section 3, our integration strategy is to provide well-defined interfaces in a *vertical integration* of a manufacturing cell, such that interfaces between components in a hierarchy are defined. Vertical integration architectures are common in formal industry standard specifications. A *horizontal integration architecture* is fully symmetric, where interfaces are common between applications. Integration of an application or component into such an architecture requires only one interface to be constructed.

3 EXPERIENCES WITH INTEGRATION USING DISTRIBUTED OBJECT TECHNOLOGIES

We have implemented a CORBA-integrated manufacturing cell in the *Sandia Agile Manufacturing Testbed (SAMT).* We explain the software integration very briefly here; this work has been published in greater detail in [19-20].

3.1 Sandia Agile Manufacturing Testbed

The SAMT [21] is a manufacturing research facility at Sandia National Laboratories in Livermore, California. The physical component in the SAMT is a networked manufacturing cell containing various machine tools, various storage devices, and transport devices. The computers in the manufacturing cell include both PCs, running Windows NT, and Unix workstations. The SAMT attempts to address the following product realization cycle: design, planning, cell management, and fabrication. During design, engineers use computer-aided design (CAD) tools and analysis tools to design parts that meet customers' requirements. The planning phase includes planning for fabrication, assembly, and inspection. The cell management activities include scheduling, tracking, and job dispatching to the shop floor. The cell management software and machine operators need access to design and planning data. The product realization cycle is by no means sequential, but rather iterative; for example, incomplete designs may be planned to guarantee that a part is manufacturable. Information from fabrication and inspection must be stored, analyzed, and accessible to designers and process engineers at a later time.

3.2 Manufacturing Cell Software

An underlying objective of the cell management software is *information-driven manufacturing*, that is, to first automate the flow of information to facilitate all those processes which precede and follow the actual machining of a part. Where it makes sense, the cell management software also permits the automation of the machining itself.

Each manufacturing device in the SAMT implements the same IDL interface: by manipulating the software interface, a client program can control the corresponding machine. The client software, the cell management software components, controls the manufacturing activities in the SAMT. The cell management software components are responsible for these tasks: entering process plans into the cell from an internal or external source; directing the development of a production plan from a process plan; assigning production plans to be scheduled; maintaining short-term cell schedules; dispatching jobs to machines in the cell; coordination of manufacturing devices in the cell; event logging of all cell activities and storage of all cell data; gathering data and statistics on machining processes (tool utilization, for example); interface to planner for replanning of machining based on sensory input; interface to material handling system; and interface to inventory system.

3.3 Manufacturing Devices

Each of the physical manufacturing objects in the agile manufacturing cell is controlled by a corresponding CORBA software object. In spite of the apparent differences among the various devices (lathe, robot, storage table, *etc.*), these objects all support the same software interface, an IDL interface called `IDevice`. As seen in Figure 1, the "plug-in" jack at the top represents the `IDevice` interface itself. This is the network-visible interface that each manufacturing device in the cell is required to implement.

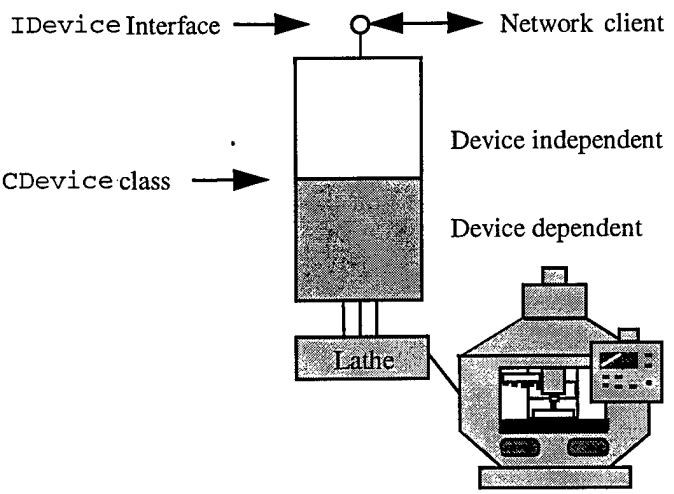

Figure 1. Implementation layers of `IDevice`.

Below this is a largely device-independent layer that is common to all of our `IDevice` implementations. While dealing with issues like presenting the CORBA interface, threads, access-control, version strings, *etc.*, the functionality of this layer does not vary greatly for different manufacturing devices. However, the `IDevice` object is ultimately required to access the hardware level of a device, for example, to open a chuck or execute a block of NC code. The mechanisms for accomplishing this do vary greatly and the functionality required of this machine-dependent layer is captured in our standardized `CDevice` C++ class. We refer the reader to [19] for a more detailed discussion of `Cdevice` and for the complete IDL for `IDevice`.

3.4 IDevice Interface

`IDevice` inherits operations and attributes from the following other interfaces:

- `IBaseDev` — Naming and operational status for the machine.
- `IAllocDev` — Controlling access to the machine.
- `IRunDev` — Running processing activities on the machine (*i.e.*, machining a part).
- `IMovePart` — Transferring material into and out of a machine.

The `GetProgDB()` operation within `IDevice` returns a value of type `IProgDB` , which is an object reference, another IDL interface, to its database of manufacturing numerical control (NC) programs. The returned object reference can be used to access the database of NC programs available for the device. Similarly, the `GetConsole()` function returns a reference to an `IConsole` object, which can be used to access the operator's console for the device, enabling communication with a human operator.

In the CORBA `IRunDev` interface we provide both synchronous and asynchronous methods for running programs. The string return value of an asynchronous method is posted to a passed notification object when the operation is completed. The creation of a notification object for a call-back allows a client to exit or process other jobs without the results being lost.

In the `IMovePart` interface, material movement is accomplished with the operations `TakeFromPartner()` and `GiveToPartner()`. Using these operations, direct device-to-device communications affect exchanging a part without micro-management of a cell manager. A robot device, for example, has an object reference to its partner in the exchange, so the robot object can manipulate the machine tool directly. Thus a call to `GiveToPartner()` on the robot results in its assuming responsibility for the transfer, invoking operations on the machine tool interface as needed. All material transfer is performed as peer-to-peer object interaction, independent of the supervisory control of the task sequencer.

In many cases, the operation of machines in the manufacturing cell is not completely automated. Our CORBA-based software architecture supports both automated and manual activities. Consider, for example, the operation `RunNamedProgram()` in interface `IRunDev`. This operation accepts as an input string the name of the NC program to run in the manufacturing device, and it is the obligation of the `IDevice` object implementation to do whatever is necessary to carry out the requested machining operation. In the fully automated case, the software can carry out the task by itself. It looks up the provided name in the program database associated with the machine, downloads the resulting NC code into the machine, and runs the code on the machine. If the machine does not support automated operation, it still must be a part of the information flow in the cell. In this case, the implementation of `IDevice` uses the `IConsole` interface to perform the operation. The `IConsole` interface provides operations needed to carry on a dialog with a human operator.

3.5 Cell Management Software

The cell sequencer dynamically attaches to devices, hence, there is no need to re-compile when a new machine tool comes on-line or a machine tool disappears. The sequencer accepts jobs, dispatches tasks in the cell, prevents deadlock situations and guards against starvation of any single job. The IDL for the sequencer, `ICellSeq`, can be found in [19].

The IDL contains an attribute `CellName` which holds the name of the manufacturing cell, allowing for several cell sequencers to be coordinated by a shop floor scheduler. A new job can be added to the sequencer with the `AddJob()` operation. This operation takes an `ITraveler` object reference as an argument; the `ITraveler` object will contain all of the necessary information to execute the job in the cell. The `AddJob` operation also takes an `INotify` object reference so that the sequencer's client can be notified of job completion or error conditions encountered. The return value of the `AddJob()` operation is the assigned `JobID` given by the sequencer; this `JobID` can then be used to `Pause()`, `Abort()`, `Resume()`, or `Delete()` a job in the sequencer. Though the cell sequencer coordinates cell activities, many operations, for example, all material movement activities, are accomplished intelligently by the devices.

We have included operations for dynamically adding and removing devices in the cell sequencer, so the sequencer will never have to be recompiled or restarted when a new `IDevice` object is available on the network. The current interface and implementation of `ICellSeq` will remain constant as new cell management components are added.

3.6 Integrating Enterprise Applications to the Vertically Integrated Manufacturing Cell

There are many ways to integrate existing and new applications into our CORBA-based manufacturing environment.

3.6.1 Integrating Commercial Off-the-Shelf Software

A large number of design, analysis, and manufacturing applications are commercial off-the-shelf (COTS) software packages. Any COTS package that has an API or a scripting language can be wrapped with a CORBA interface and made available on the network. Further, any COTS package that has an extensible GUI can be extended to be a CORBA client.

3.6.2 Integrating Legacy Software and Databases

In most enterprises, it will be necessary to integrate legacy codes. We integrated several legacy FORTRAN design and analysis codes. Wrapping these codes as CORBA objects made some legacy codes accessible again, and no software was rewritten. First, we stripped out any I/O from the source code, and created a FORTRAN library. Second, we defined an IDL interface to the analysis code, providing functionality for sending input data, executing any engineering functions, and extracting data from the code. Next we called the FORTRAN library from the object implementation. The analysis code is then a software component that is available over the network to any CORBA client. A new GUI or web interface to the code can be developed easily without changing the engineering functionality in the analysis code.

The same approach can be used to wrap relational databases and make them accessible on the network as CORBA objects. Remote database access is critical to the extended enterprise. New databases can be added to the enterprise by implementing the common IDL database interface.

3.6.3 Supporting a Variety of Software Clients

CORBA allows many different client applications to be integrated into the environment. We have adopted the following software engineering development technique in our integration: separate the GUI from the "compute engine" or application. This allows for greater software reuse since new GUI development tools/languages and browser extensions evolve rapidly.

Figure 2 shows how CORBA objects can easily be accessed by a variety of client technologies. In the left-hand column of the figure are CORBA objects, and in the right-hand column various client technologies are located. Any client can access any CORBA object, by using an "adapter". An "adapter" is the technology (either vendor-provided or developed elsewhere) that allows a CORBA object to be accessed by a client application. The figure illustrates this plug-and-play client technology, by matching adapter jacks and client jacks.

We have developed GUIs and client applications for cell monitoring and cell management using the Tcl/Tk [22] programming language. An extension [23] of Tcl/Tk, TclDii, allows the use of distributed CORBA objects from within Tcl. Tcl/Tk allows X-Windows based user interfaces to be assembled easily, and this extension is very useful for rapid-prototyping of GUIs. The resulting client GUI is portable across several platforms.

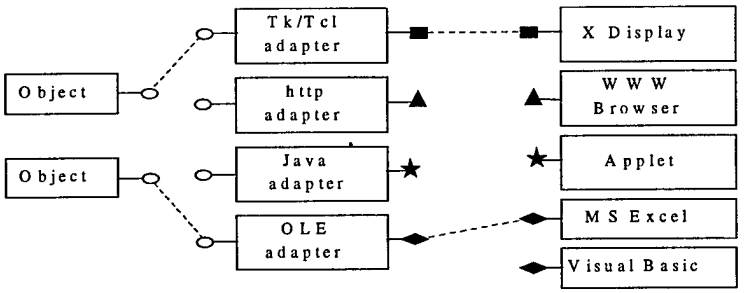

Figure 2. Integrating a variety of software clients into our software architecture.

3.6.4 Integrating to World Wide Web

Today more and more companies have an Internet presence and this is central to their way of doing business with customers, suppliers, and business partners. In addition to the Internet, Intranets (*i.e.*, internal restricted-access networks, possibly connected by a firewall to the Internet) and Extranets (*i.e.*, restricted networks shared between partnering business entities) are becoming commonplace. Web browsers can be used to navigate networks and to execute applications.

There are several ways in which a CORBA-based distributed enterprise is accessible on the web. One powerful way is with the inclusion of Java client applets that communicate with CORBA objects, executing on computers in the enterprise. Java is an object-oriented, platform-independent programming language that has libraries for Internet access. Most web browsers allow Java applets to download and execute. Since the applet is loaded on the fly, new applications and modifications to applications can be deployed instantly. Java clients supporting IIOP will be transportable across ORBs.

Another method of allowing web access to CORBA objects is by using CGI (Common Gateway Interface) scripts which are clients to the CORBA objects. Tcl, C, C++, and Perl are languages for writing CGI programs that are clients to CORBA servers. Other researchers [24-25] discuss the integration of the World Wide Web and distributed objects, and the consensus is that this is a very complimentary and powerful technology integration.

3.6.5 Integrating to PC Applications Software

The OMG specifies internetworking technology between OLE and CORBA. Several vendors have implemented this, allowing access to OLE objects from CORBA objects and vice versa. This enables access to and from many PC and Mac desktop applications. For example, this technology can be used to input numerical data from a Microsoft Excel spreadsheet into a CORBA design code on the network. Also, Visual Basic applications can execute on a PC and communicate with CORBA distributed objects. Visual Basic is ideal for rapidly building GUIs for PCs and for interfacing to OLE objects.

3.7 Strengths and Weaknesses of Our CORBA-Based Manufacturing Integration

We did not explicitly discuss our machining devices and cell sequencer in terms of agents; however, they have properties of agent-based systems, and arguably could be called agents. First, the device interface encapsulates the functionality of a machining, storage, or transport device. Second, a device responds to well-defined method calls from other devices in a peer-to-peer fashion. Third, there is a protocol established for coordination, *e.g.*, material movement from one device to another. Fourth, the devices and cell sequencer are persistent. Finally, the device objects are reactive and have some intelligence.

The distributed object interfaces in our manufacturing cell are robust, allowing for easy addition, deletion, and updating of manufacturing devices in a plug-and-play fashion. CORBA enhances the system integration because it is an industry standard for interoperable, distributed objects. CORBA is scalable and extensible to the enterprise level.

CORBA supports integration with many different information technologies and programming languages. In time, as commercial software vendors provide CORBA interfaces to various software components, it should be easy to integrate them with our developed manufacturing software. Software developers can build up collections of reusable, large-grained services that can be used and customized by other developers to assemble new applications or integrate existing applications.

The manufacturing cell integration is a very specific manufacturing application (cell control), which is very well-suited to the vertical integration framework that we applied. Adding additional tools and databases into the larger manufacturing enterprise can break this framework without careful planning. We believe that a horizontal integration framework, one that allows any software tool to be plugged into the enterprise, is a better integration strategy for the enterprise. The Product Realization Environment (PRE) [26], being developed at Sandia National Laboratories for Defense Programs, is an example of a horizontal enterprise integration framework. Some advantages of PRE include these:

- Elimination of ad-hoc integration
- Provision for simple, non-intrusive application integration
- Provision for simple, dynamic integration of new applications into enterprise
- Standard set of objects shared by all applications
- Use of MIME types for information exchange
- Standard solutions to common problems, reducing the programming effort and redundancy while providing for robust, common systems
- Software development tools and standards
- Shielding application programmers from bugs in CORBA software, compilers, and operating systems
- Common services
- Elimination of difficult and/or repetitive programming

In our manufacturing cell, all software interfaces were designed and implemented at Sandia. The same integration would have been difficult, though not impossible, if the manufacturing cell was spread across several organizations. In this latter case, software developers from each organization would need to agree on the interfaces.

SEMATECH has proposed a vertical integration strategy in the form of IDL for a computer integrated manufacturing (CIM) framework [27-28]. Unlike our integration architecture, the SEMATECH architecture does not explicitly support information-driven manufacturing with human integration. Also, the SEMATECH interfaces have been designed by a consortium committee, hence, the interfaces themselves are rather complex. Furthermore, it is unclear if other enterprise applications can be easily integrated, as in a horizontal framework. Finally, no real implementation of the SEMATECH framework exists, even though this has been proposed for several years now.

CORBA does not address any knowledge exchange, and hence, any exchange between devices and cell management software uses a protocol and ontology developed in-house. Since CORBA is not an agent communication language or protcol, the knowledge exchange problem arises when we scale our manufacturing cell to the enterprise level.

4 EXPERIENCES WITH AGENT-BASED INTEGRATION TECHNIQUES

In joint research with the Center for Design Research (CDR) at Stanford University, we proposed an agent-based concurrent engineering architecture [11]. We focused on the design phase and the mechanisms necessary to exchange process capability data between a manufacturing service and the designer. This architecture provides a mechanism for utilizing dynamic capability information taken from an on-machine inspection process [29] to realize a concurrent design environment. The implementation of this concurrent engineering environment addresses four primary challenges:

1. Process model generation.
2. Acquisition of the model by the designer.
3. Mapping of the model into the design space.
4. Applying model information during design.

4.1 Representation of Process Capability Models

The proposed architecture uses a formal object-oriented conceptual model (written in EXPRESS) to represent the capability information for an on-machine inspection process. This model represents both the objects present in the development cycle and their relationship to one another during execution of the development cycle. In this model, knowledge relating to geometry, topology, materials, tolerances and features is represented using STEP Parts 41,42, 43, 45, 47 and 48. The entire process capability model hierarchy can be found in [11].

4.2 Agent Architecture

In the agent architecture shown in Figure 3, knowledge and functionality are encapsulated inside agents. The primary benefit of this agent-based architecture is that it facilitates the integration of large systems comprised of distributed, heterogeneous components. The integration of multiple designers with multiple manufacturing services represents such a system. Agents logically unify heterogeneous distributed information and knowledge. This particular architecture is geographically distributed, with the designer residing at Stanford and the manufacturing facility at Sandia.

In our implementation, manufacturing process knowledge is represented using portions of the STEP standard and this knowledge is exchanged as agent messages written in EXPRESS. KQML is used for the outer structure of all agent

messages, and messages are sent over the Internet using TCP/IP. This architecture is implemented with the Java Agent Template (JAT) [30], a template for implementing KQML-speaking agents in Java, developed at the CDR.

Information, in the form of STEP schemas, is exchanged from the inspection process to the design processes and is displayed graphically to designers in Java applets. The designer, therefore, does not need to learn or understand STEP, and the design environment requires no additional software beyond a web browser. The features and part information is mapped to STEP as a method of knowledge exchange, and the designer views this information in a format that is useful and readable to him/her.

When the designer requests information and process constraints from manufacturing services, the work is performed by the design agent, interfacing to the CAD system. The design agent serves two functions: first, as incoming agent messages are received, the contents are interpreted and based on the message contents, the CAD tool (and hence designer) is notified accordingly; and second, as design phase events occur, agent messages are constructed and sent to coordinating agents. Similarly, the cell manager agent receives all requests from outside of the manufacturing cell, forwards requests to internal manufacturing agents as appropriate, combines any responses, and returns messages to the sending agent.

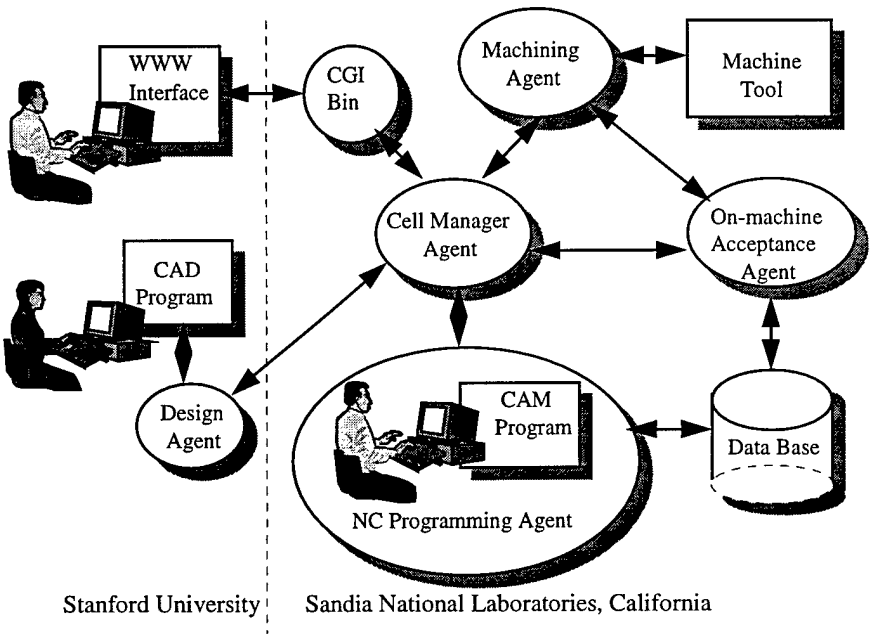

Figure 3. Agent Architecture.

4.3 Application of Capability Information During Design

The concurrent engineering architecture uses a feature-based design system [17,31-32] that enables designers to evaluate the inspectability of each feature as it is added to a design. The system consists of a CAD system, in our case, PRO/ENGINEER by Parametric Technology Corporation, a design agent, and a constraint manager. This architecture can support any commercial or publicly available CAD tool. As shown in Figure 4, the design agent and constraint manager communicate with the CAD system via the prescribed API. The appropriate process capability models and all relevant inspection constraints for the selected manufacturing service and machine are acquired by the design agent, mapped into the local representation format (dependent on the CAD system and constraint manager), and loaded into the both the CAD system and constraint manager. The set of feasible fabrication features and stock parts are loaded into the CAD system. The related process, machine and manufacturing service constraints on the fabrication features are loaded into the constraint manager.

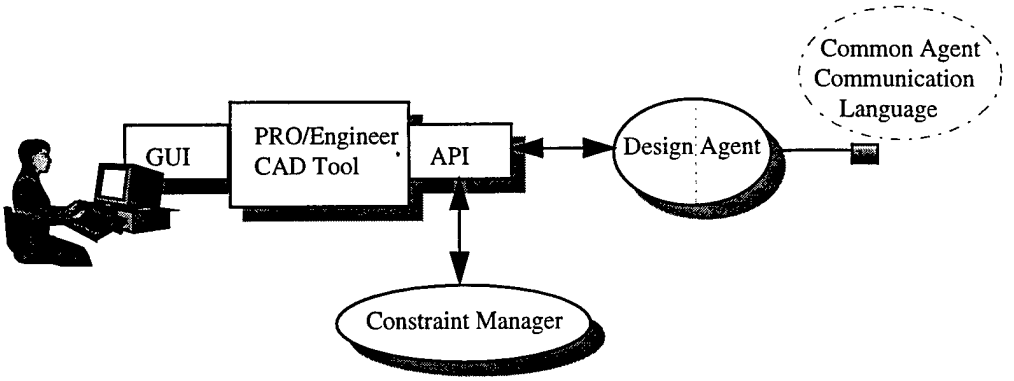

Figure 4. Design Agent: Internal Architecture.

Once the capability information has been acquired and loaded, the designer uses the CAD system to select a feasible stock and uses constructive solid geometry operations with feasible fabrication features to design a part. As each fabrication feature is applied to the part, the designer must identify, for the set of resulting inspection features, the desired tolerance. The specified tolerance, along with the nominal feature geometry and its position relative to the current part will be submitted to the constraint manager to determine constraint satisfaction. The constraint manager will apply all relevant declarative and procedural constraint information. If any constraint violations are found they will be reported to the designer, who may alter the applied feature or renegotiate the OMA constraints to satisfy the constraints or continue with the violating design.

As seen in Figure 5, the machining agent is implemented with an agent wrapper, a KQML interface, around the manufacturing cell software presented in Section 3.

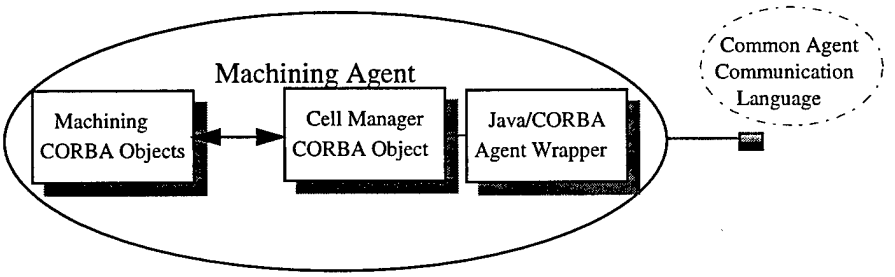

Figure 5. Machining Agent: Internal Architecture.

The JAT makes it easy for prototyping software agents in Java; however, this software does not have the same services as standard distributed object technologies such as CORBA. Hence, software developers still have to implement many agent services.

4.4 Integrating Enterprise Applications in the Agent-Based Architecture

Any COTS software package that has an API or a scripting language can be wrapped as an agent, though the process of creating an agent interface (including determining what knowledge must be defined, exchanged, and interpreted) will be more complex than it would be to make the same software available as a CORBA object. Any legacy software or any database can also be wrapped as an agent. Further, any COTS package that has an extensible GUI can be extended to be an agent which sends messages to other agents.

Though it is possible to create interfaces to other component software technologies, there are no vendor-supplied tools for doing so, and hence, the software development cost is high. Furthermore, there are no vendor-supplied toolkits for building agents that speak KQML (or any other agent communication language); KQML software to date is developed in-house.

4.5 Strengths and Weaknesses of Our Agent-Based Integration

The agent-based architecture and related knowledge representation allows designers to design parts for manufacturability and inspectability. This agent architecture supports a geographically and organizationally distributed system where the design

agent and manufacturing agent reside at different locations. There is an unambiguous agent communication language. This architecture can be adapted easily to other design environments and also to other manufacturing and inspection processes. By using the JAT, or any Java agent development kit, integration with the web comes for free. Java is an excellent programming language for creating agents that are available on the World Wide Web.

This architecture demonstrates the feasibility of the knowledge exchange between a design agent and a manufacturing agent, yet there are still many challenges. First, even though we used EXPRESS to define our knowledge, our particular ontology is not at all standard and it is unlikely that it ever will be. The *ontology problem, i.e.*, defining a complete vocubulary for use within a domain, is a very difficult problem that researchers have been working on for years. Our knowledge exchange is dependent on a simplified and incomplete on-machine inspection process model. The design knowledge only includes simple geometries and features, so in order to be production worthy, more complex geometries and features, costs associated with constraints, and fabrication constraints must be added. This concurrent design architecture will become more powerful as functionality is added to agents, and new manufacturing processes are encapsulated as agents. At this time, there is a significant startup effort for other design and manufacturing entities to plug into this architecture, and many partners may be reluctant to spend the time and energy for the software development.

The software development costs in this project are high because developers are forced to deal with the low-level network and agent messaging requirements. The JAT, a publicly available research software package, hides some of the KQML parsing and TCP/IP message requirements, though this tool is not professionally developed, not supported, not robust, and not industry standard. Finally, there are no development tools to integrate KQML agents with other technologies as seamlessly as CORBA does.

5 USING JAVA TO INTEGRATE MANUFACTURING AND ENGINEERING ENTERPRISES

Many researchers [33] are exploring Java as a way to build distributed computing systems, in essence, an alternative to CORBA. Java has a distributed object model, the Remote Method Invocation (RMI) [34]. Java has these desirable features for enterprise integration: close integration with the web, security, multi-threaded language, absence of common error-prone language semantics and additional useful language features (*e.g.*, exception handling and garbage collection), and portability across hardware and operating system platforms.

Java has the following strengths beyond CORBA:

- *Mobility of code*: Any Java application cab dynamically download the classes of remote objects, their interfaces, stubs, parameter, and return values. The OMG is soliciting requests the implementation of mobile code, so in the future CORBA should address this issue.
- *Pass by value*: Java RMI passes non-remote arguments and results by value and remote objects passed by reference. CORBA passes all objects by reference, leading for inelegant solutions when pass by value is needed. The OMG, however, has plans to address this issue.

CORBA, a more mature technology, and has the following strengths over the Java RMI:

- *Language neutrality*: CORBA specifies mappings from IDL to many programming languages, allowing software developers to select the programming language that is best suited for his/her application.
- *Integration with legacy systems*: CORBA allows an IDL wrapper to be written to support the legacy system. With the Java RMI, you can only wrap applications with C and C++ to integrate to existing systems.
- *Advanced communication patterns*: CORBA provides both synchronous and asynchronous methods. With the Java RMI, all method invocations are synchronous.
- *Associated services*: The OMG has defined a significant number of services, *e.g.*, naming, event, security, and notification. The Java RMI is not as mature at this time, though in time many services should become available.
- *Security*: Even though the Java RMI class loader imposes the same security as those imposed by the applet class loader, the CORBA security service is a low-level framework which support authentication, authorization, encryption, auditing and logging, and credential management.
- *Integration with distributed object technologies*: Unlike CORBA, Java does not integrate with OLE and DCE at this time.
- *Performance*: Because the Java Virtual Machine must interpret byte code at run-time, the speed of a Java server will not be fast as the speed of a CORBA server written in C or C++. The arrival of Java Just-In-Time (JIT) compilers in the future should make this a non-issue.

Our current recommendation is that CORBA provides a more mature infrastructure for building distributed object systems than the Java RMI. The Java RMI and CORBA are merging. The OMG has specified a mapping from IDL to Java, and

hence, any client or server can be written in Java. Java, because of its GUI component classes and integration in web browsers, is a very attractive language for writing client software. In addition, future releases of Netscape will provide IIOP capability in their Java Virtual Machine, and hence, any applet can be an IIOP client to a CORBA server, regardless of the use of the Java RMI. Further, Javasoft plans to support IIOP in a future release of the Java RMI, making many of the CORBA services and CORBA strengths available from the Java RMI.

6 REVIEW OF INTEGRATION TECHNIQUES

We now review the integration techniques with respect to connecting large enterprises. Our recommendations may not be true for single domain problems or non-enterpise applications.

6.1 Problems With Using CORBA to Integrate Enterprises

One current limitation of CORBA is that it does not define a protocol for knowledge exchange thus the support for software agents is not at all easy or standard. Second, CORBA does not enforce standard interfaces nor common terminology for the easy integration of related software components. Thus, integration between domains and organizations is not trivial. Third, CORBA does not yet provide mobility of code, and developing mobile agents (though not discussed in this paper) is not possible. Finally, as we mentioned in Section 2.2, an ad-hoc integration in CORBA, without an integration framework, can lead to scalability problems. The manufacturing cell that we developed had a very well-defined vertical integration framework.

6.2 Problems With Using Software Agent Communication Languages to Integrate Enterprises

There are several problems with using software agent communication languages to integrate manufacturing and engineering enterprises. First of all, there is no standard way for the agents to connect to and use non-agent applications, *e.g.,* databases, in the enterprise. Second, KQML and other agent-based approaches are not widely used; thus, there are no development tools, no applications from which to leverage, and no vendor-supplied software on which to develop. Third, the KQML standard is up-in-the-air, currently there is no real formal protocol for performatives and any number of new performatives can be arbitrarily defined by users. So, even in an agent-based enterprise, the software developers have to cooperate on interfaces prior to new agents being plugged into the architecture. Fourth, the development of ontologies and the knowledge exchange problem are difficult problems, and thus there is no simple solution that you can pick up and integrate into an architecture. Finally, there is no built-in security (authentication or encryption) in KQML. Some of these problems are being addressed by FIPA, though at this time this standard is not being embraced by many multi-agent system developers.

6.3 Evaluating Distributed Object Techniques and Software Agent Approaches to Enterprise Integration

We review the current state of both technologies with respect to the criteria developed in Section 2.

CORBA provides an interface language for creating an object-oriented abstraction of an enterprise software component. Agent communication languages such as KQML do not provide the same abstraction. However, an object-oriented programming language, such as Java, can be used to build agents, thus, agents can be developed with object-oriented abstractions.

Both CORBA and the agent-based approach are integration techniques that can be applied regardless of hardware platform or programming language. CORBA *explicitly* supports a higher level of integration, *i.e.*, at the distributed object level, and it provides a number of services that are useful to distributed object systems. KQML agents can be built on top of CORBA.

In our experience, the software in our CORBA-based manufacturing environment was much easier to develop than the manufacturing agents in the agent-based enterprise. This is primarily because the low-level network interface and marshalling of arguments across distributed components was provided by CORBA, and hence, the software developers did not have to write low-level network software to deal with the heterogeneous, distributed system. In the agent-based approach, we developed software to show a proof-of-concept rather than providing a production quality manufacturing environment. This agent-based software was not built on top of middleware, so the developers had to worry about low-level network details. The JAT relieved the software developers of some low-level details; however, the JAT is research software.

While both the agent approach and the distributed object approach have goals of supporting distributed applications, only the CORBA approach to building enterprise systems has explicit support for developing distributed applications. Likewise, only CORBA, and not KQML, is designed to solve the issue of interoperability.

Both the distributed object approach and the agent approach to enterprise integration are extensible, yet as the number of objects (or agents) increase, there will be problems with scalability. In both approaches, there must be methods for dealing with the scalability of the enterprise design.

One of the necessary missing features in both strategies is security. The OMG has specified a CORBA security standard, yet not many vendors have yet implemented this specification. In our manufacturing cell, the manufacturing devices and cell management software are only accessible on Sandia's restricted Intranet, so the firewall provides some security. The agent software, however, has no security implementation. When a vendor-supplied CORBA security implementation is available, the CORBA-based manufacturing system can be updated to take advantage of security needs.

Regarding maturity of applications, academic researchers have done some prototyping with integrating engineering and manufacturing enterprises with software agents; however, the resulting systems are not robust, not complex, and not distributed in a wide-area geographically distributed environment. Furthermore, the KQML technology is not mature or widely used. On the other hand, industry is using distributed object technologies and standards, such as CORBA, to build enterprise systems, and this technology is mature. CORBA has proven applications, vendor-implemented object services, and some object facilities.

CORBA is an industry standard for building distributed object systems. Though there have been efforts to "standardize" KQML as an agent communication language, to date, this has not happened. Furthermore, because researchers do not agree upon the semantics for the basic set of KQML performatives and KQML is extensible, a standards effort is difficult. FIPA is attempting to standardize an agent communication language. Yet, even if there is a standard agent communication language, only time will tell if these languages are ever widely used.

CORBA is interoperable with other component software. At this time, there is no adapter or facility to allow component software and agent communication languages to interoperate.

Whereas academic researchers are not always willing to pay for vendor-supplied software, industry realizes that the cost to develop software in-house is usually much higher. Most ORB products are supplied by vendors, though some public domain ORBs are planned. KQML is "free", so to speak, though the in-house software development costs to build the underlying infrastructure are high.

CORBA, unlike KQML, is not an agent communication language, and there is no agent facility in CORBA. If an agent architecture were developed on top of CORBA, some agents communication protocol must be used. Some researchers have developed a formal semantic protocol [35] for using KQML in multi-agent systems.

There is no semantic unification as part of the CORBA standard or being developed by the OMG. KIF, PDES/STEP, and ontologies all provide semantic unification. If agents are developed in CORBA or if agents use KQML, there is still a need to add the semantic unification to the architecture. In the agent-based engineering scenario that we developed in Section 4, we used the OMA model presented in EXPRESS format for semantic unification.

Finally, neither CORBA nor KQML directly provides coordination of agents. This is something that the software designers and developers must build into the architecture. Because KQML is an agent communication language, it will be easy to develop coordinating agents based on the message exchange. However, agents can be built with CORBA thus that various methods cause the agents to coordinate.

6.4 Recommendations for Integrating Manufacturing and Engineering Enterprises

The three technologies that we have presented in this section — CORBA, Java/RMI and KQML — are not mutually exclusive. In fact they are complementary in many ways. In the previous section we discussed how CORBA and the Java programming language are compatible. In earlier sections, we mentioned that KQML, or any agent communication language, can be layered on top of CORBA to take advantage of the strengths of CORBA.

Most multi-agent systems to date address problems at the domain level and not at the enterprise level. Those systems which have attempted to address enterprise computing have been "toy" examples and not production quality. Hence, agent coordination has not been tested for enterprise computing. We believe at this time that KQML and agent communication languages are not yet integrated with disributed object technologies to the extent that they should for integrating the enterprise, and the inclusion of software agents into a distributed framework should be explored more fully. We strongly advocate the need for more research and development in agent-based enterprise computing.

Finally, software developers should consider these properties when making a decision about enterprise integration: enterprise integration costs, the size of the enterprise that needs to be integrated, and the extent to which standards are important.

7 CONCLUSIONS AND FUTURE WORK

The abundance of electronic information and the complexity of software systems is a reality, and the Internet is growing at a phenomenal rate worldwide. Therefore, the amount of electronic information available on networked computers continues to increase. Yet people are not necessarily more productive and are at times less productive because of the time and effort

needed to search this information, to use new and legacy software resources, and to combine networked software applications and data. This is an "information age paradox".

Software agents, adaptive and intelligent software components, can respond to users and the state of an environment, act on behalf of users, and can coordinate users and other software components in order to accomplish a goal. They provide a viable solution to the information age paradox.

Though the concept of an agent was proposed at MIT in the 1950's, the need for agents has never been greater. Further, the maturity of technologies that support software agents — robust programming languages, distributed computing technologies, expert systems, and most importantly Internet technologies — makes the time perfect for the development, application, and deployment of software agents to new and existing information systems at the enterprise level.

We have presented two different approaches to integrating enterprises: a CORBA-based distributed object manufacturing enterprise and an agent-based architecture for design and manufacturing. We have also proposed a list of criteria that should be considered prior to building enterprise information architectures. We have evaluated both enterprise integration techniques and have made recommendations regarding which technology is better-suited for each of the enterprise integration issues. Our general belief is that a combination of technologies is needed for achieving advanced, intelligent integration.

8 ACKNOWLEDGMENTS

The author expresses gratitude to a number of people at Sandia National Laboratories. Jim Costa has been the program manager for this work. Bob Whiteside and Paul Klevgard are collaborators on the distributed object manufacturing software. Andy Hazelton provided his on-machine acceptance models for agent knowledge exchange. Finally, Mark Cutkosky and Rob Frost at the CDR collaborated on the design and manufacturing agents in our agent architecture.

9 REFERENCES

1. R. Orfali, Harkey, D., and Edwards, J., *The Essential Distributed Objects Survival Guide*, John Wiley and Sons, Inc., New York, New York, 1996.

2. The Common Object Request Broker: Architecture and Specification, OMG Technical Document PTC/96-03-04, Object Management Group, Framingham, Massachusetts, July 1995.

3. K. Brockschmidt, *Inside OLE 2*, Second Edition, Microsoft Press, Redmond, Washington, 1995.

4. Apple Computer, Inc., *OpenDoc Programmer's Guide for the Mac OS*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

5. J. M. Bradshaw, editor, *Software Agents*, The MIT Press, Cambridge, Massachusetts, 1997.

6. T. Finin, J. Weber, *et. al.*, "Specification of the KQML Agent-Communication Language", The DARPA Knowledge Sharing Initiative External Interfaces Working Group, February 9, 1994.

7. M. R. Genesereth, R. E. Fikes, *et. al.*, "Knowledge Interchange Format Version 3.0 Reference Manual", Computer Science Department, Stanford University, Stanford California, June 1992.

8. T. R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing", in *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer Academic Publishers, 1993.

9. H. Mason, editor, *Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 1: Overview and Fundamental Principles*, Version 9, ISO TC184/SC4/WG PMAG Document N50, December 1991.

10. J. Owen, *STEP: An Introduction*, Information Geometers Ltd, Winchester, UK, 1993.

11. C. M. Pancerella, A. J. Hazelton, and H. R. Frost, "An autonomous agent for on-machine acceptance of machined components", *Proceedings of Modeling, Simulation, and Control Technologies for Manufacturing*, SPIE's International Symposium on Intelligent Systems and Advanced Manufacturing, Philadelphia, Pennsylvania, October 1995.

12. `http://drogo.cselt.stet.it/fipa/.`

13. M. Cutkosky, R. Engelmore, R. Fikes, T. Gruber, M. Genesereth, W. Mark, J. Tenenbaum, and J. Weber, "PACT: An experiment in integrating concurrent engineering systems", *IEEE Computer*, January 1993.

14. J. G. McGuire, D. R. Kuokka, J. C. Weber, J. M. Tenenbaum, T. R. Gruber, and G. R. Olsen, "SHADE: Technology for Knowledge-Based Collaborative Engineering", *Journal of Concurrent Engineering: Applications and Research (CERA)*, 1(2), September 1993.

15. G. R. Olsen, M. Cutkosky, J. M. Tenenbaum, and T. R. Gruber, "Collaborative Engineering based on Knowledge Sharing Agreements", *Proceedings of the 1994 ASME Database Symposium*, September 11-14, 1994, Minneapolis, MN.

16. D. Goldstein, "An Agent-based Architecture for Concurrent Engineering", *Concurrent Engineering: Research and Applications*, 2: 117-123, 1994.

17. M. R. Cutkosky and J. M. Tenenbaum, "Toward a Framework for Concurrent Design", *International Journal of Systems Automation: Research and Applications*, 1(3):239-261, 1992.

18. T. J. Mowbray and R. Zahavi, *The Essential CORBA*, John Wiley and Sons, Inc., New York, New York, 1995.

19. R. A. Whiteside, C. M. Pancerella, and P. A. Klevgard, "A CORBA-Based Manufacturing Environment", *Proceedings of the Hawaii International Conference on Systems Sciences*, Maui, Hawaii, January 1997.

20. C. M. Pancerella, and R. A. Whiteside, "Using CORBA to Integrate Manufacturing Cells to a Virtual Enterprise", *SPIE Proceedings of Integrated Manufacturing – Plug and Play Software for Agile Manufacturing*, Boston, Massachusetts, November 1996.

21. H. Park, J. Stori, and P. Wright, "Rapid Response Manufacturing in Distributed Environments: the important role of process planning and open architectures", *Proceedings of the Atlanta IMECE Symposium on Rapid Response Manufacturing*, Atlanta, Georgia, November 1996.

22. J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

23. G. Almasi, A. Suvaiala, *et. al.*, "TclDii: A TCL Interface to the Orbix Dynamic Invocation Interface", Concurrent Engineering Research Center, West Virginia University, Morgantown, West Virginia, 1994.

24. O. Rees, N. Edward, *et. al.*, "A Web of Distributed Objects", ANSA, Cambridge, United Kingdom, November 1995.

25. A. Vogel, "WWW and Java Threat or Challenge to CORBA?", CRC for Distributed Systems Technology, University of Queensland (Brisbane), Australia, 1996.

26. `http://www-collab.ca.sandia.gov/pre/.`

27. SEMATECH Computer Integrated Manufacturing Application Framework Specification 1.2, SEMATECH Technology Transfer #93061697E-ENG, Austin, Texas, 1995.

28. D. Doscher, and R. Hodges, "SEMATECH'S Experiences with the CIM Framework", *Communications of the ACM*, Vol. 40, No. 10, October 1997.

29. A. J. Hazelton, "On-Machine Acceptance of Machined Components", *Proceedings of the 10th Annual Meeting of the American Society of Precision Engineering*, October 1995.

30. H. R. Frost and M. R. Cutkosky, "Design for Manufacturability Via Agent Interaction", *Proceedings of the 1996 ASME Design for Manufacturing Conference*, Irvine, California, August 1996.

31. P. K. Wright, "Principles of Open-Architecture Manufacturing", Engineering Systems Research Center, ESRC 94-26, University of California at Berkeley, October 1994.

32. S. K. Gupta and D. S. Nau, "A Systematic Approach for Analyzing the Manufacturability of Machined Parts, *Computer Aided Design*, 27(5):323-324, 1995.

33. K. U. Reddy, "Java and Distributed Computing", *Proceedings of the Grace Hopper Celebration of Women in Computing Conference*, San Jose, California, September 1997.

34. Sun Microsystems, Inc., "Java Remote Method Invocation Specification", December 1996.

35. J. Mayfield, Y. Labrou, and T. Finin, "Evaluation of KQML as an Agent Communication Language", in *Intelligent Agents Volume II -- Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages*, M. Wooldridge, J. P. Muller and M. Tambe (eds.), *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 1996.

Report Number (14) SAND--98-8542C
CONF-980726--

Publ. Date (11) 199804
Sponsor Code (18) DOE/ER, XF
UC Category (19) UC-405, DOE/ER

DOE