

SANDIA REPORT

SAND98-8226 • UC-405

Unlimited Release

Printed January 1998

The Use of Agents and Objects to Integrate Virtual Enterprises

C. M. Pancerella

Prepared by

Sandia National Laboratories

Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Approved for public release; distribution is unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A04
Microfiche copy: A01

The Use of Agents and Objects to Integrate Virtual Enterprises

C. M. Pancerella
Distributed Systems Research Department
Sandia National Laboratories

Abstract

The manufacturing complex for the Department of Energy (DOE) is distributed: design laboratories, manufacturing facilities, and industrial partners. Designers must have a concurrent engineering environment to support all aspects of the cradle-to-grave product realization process across the distributed sites. Engineers must be able to analyze and simulate processes, retrieve and process heterogeneous information, both archived and current, and access multiple databases. Manufacturers must be able to coordinate activities of various manufacturing centers, which may involve a negotiation process. Furthermore, Sandia must be able to export manufacturing capabilities, such as on-machine acceptance, to outside suppliers. A key element to making this a reality is a flexible information architecture.

The DOE information architecture must support a wide-area virtual enterprise, with distributed intelligent software components. The architecture must provide for asynchronous communication; multiple programming languages and operating systems; incorporation of geographically distributed manufacturing services; various hardware platforms; and heterogeneous workstations, PC's, machine tool controllers, and special-purpose compute engines. Further, it is critical that manufacturing facilities are not isolated from design, planning, and other business activities and that information flows easily and bidirectionally between these activities. To accomplish this seamlessly, heterogeneous knowledge must be exchanged across both domain and organizational boundaries. Distributed object and software agent technologies are two methods for connecting such engineering and manufacturing systems. 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. 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 knowledge exchange across domain boundaries.

This project addresses the integration of these technologies in support of concurrent engineering, team collaboration, and manufacturing across organizational and geographic boundaries. We discuss our experiences with both technologies, explore both in the context of enterprise integration, and suggest future research in this area.

Table of Contents

1.	Introduction	7
1.1	Distributed Object Technology	7
1.2	Common Object Request Broker Architecture (CORBA)	7
1.3	Software Agents	8
1.4	Agent Communication Languages	8
1.5	Knowledge Representation	9
1.6	Agent Architecture and Knowledge Sharing	9
2.	Experience with Integration Using Distributed Object Technologies	10
2.1	Sandia Agile Manufacturing Testbed	10
2.2	Manufacturing Cell Software	10
2.3	Manufacturing Devices	11
2.4	IDevice Interface	11
2.5	Cell Management Software	13
2.6	Integrating Enterprise Applications Using CORBA	14
2.6.1	Integrating CAD Systems and COTS Software	14
2.6.2	Integrating Legacy Software and Databases	14
2.6.3	Support for a Variety of Software Clients	15
2.6.4	Integrating to World Wide Web	15
2.6.5	Integrating to PC Applications Software	16
2.7	Strengths and Weaknesses of Our CORBA-Based Manufacturing Integration	16
3.	Experiences with Agent-Based Integration Techniques	17
3.1	Representation of Process Capability Models	17
3.2	Agent Architecture	18
3.3	Application of Capability Information During Design	19
3.4	Integrating Enterprise Applications in the Agent-Based Architecture	20
3.5	Strengths and Weaknesses of Our Agent-Based Integration	20
4.	Software Integration Issues	21
4.1	Integration Strategies	22
4.2	Problems With Using CORBA to Integrate Enterprises	23
4.3	Problems With Using Software Agent Communication Languages to Integrate Enterprises	23
4.4	Evaluating Distributed Object Techniques and Software Agent Approaches to Enterprise Integration	23
4.5	Using JAVA to Integrate Manufacturing and Engineering Enterprises	24
4.6	Recommendations for Integrating Manufacturing and Engineering Enterprises	25
5.	Conclusions and Future Work	26
6.	Acknowledgments	26
7.	References	26

1 INTRODUCTION

The DOE manufacturing enterprise must be agile. *Agile manufacturing* [1] refers to rapid response to changes in product mix, batch size, manufacturing processes, customer 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 this agility and to realizing the integration of the extended enterprise. The information architecture must be scalable, interoperable, and reconfigurable. Internet technology, electronic data exchange, and industry standards for interoperability are core to the infrastructure. It comes as no surprise that the February 1996 *Communications of the ACM* devoted an entire issue to the growing role of computer science and software in manufacturing [2] or that entire workshops devoted to the role of plug-and-play software for agile manufacturing [3] are being sponsored by engineering professional organizations.

Given the requirement for agility, the heterogeneous computing resources, and the need to connect to an extended enterprise, there are several demands placed on software in the DOE manufacturing enterprise. 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 software must facilitate information flow through the product realization cycle — from design and analysis to planning to fabrication and inspection. The software must be easy to deploy and usable throughout engineering and manufacturing facilities. Manufacturing cells must be easily integrated into the extended enterprise such that manufacturing process data (for example, inspection reports and on-machine measurements) can be stored in a database for use by designers and process engineers in the future. Finally, knowledge must be represented in a formal and unambiguous manner and exchanged between the various domains and organizations in the enterprise.

In this section, we provide terminology and discuss background technologies related to distributed objects and software agents.

1.1 Distributed Object Technology

Distributed object technology [4] allows computing systems to be integrated such that objects or components work together across machine and network boundaries. Examples of current distributed object or component technologies include CORBA[5], OLE[6], and OpenDoc[7]. A distributed object is not necessarily a complete application but rather 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 distributed object can execute either on the same computer or on another networked computer as other objects. Thus a client object may make a request of a server object and the operation proceeds unaffected by their respective locations. Following the principles of object-oriented design, a distributed object has a well-defined interface, describing the data and functionality it exposes to other objects.

The most common standard for the deployment of wide-area distributed objects today is the Common Object Request Broker Architecture (CORBA). CORBA addresses issues of interoperability in a distributed heterogeneous system.

1.2 Common Object Request Broker Architecture (CORBA)

CORBA [5] is an industry middleware standard for building distributed, heterogeneous, object-oriented applications. CORBA is specified by the *Object Management Group (OMG)*, a non-profit consortium of computer hardware and software vendors. At this time, CORBA provides the best technical solution for integrating distributed enterprises; it is open, robust, heterogeneous, interoperable, multi-platform, and multi-vendor supported.

The OMG Interface Definition Language (*IDL*) is used to define interfaces in CORBA. An IDL interface file describes the data types, and methods or operations that a server provides for an implementation of a given object. IDL is not a programming language, but rather a language that

describes only interfaces: there are no implementation-related constructs in the language. The OMG does specify mappings from IDL to various programming languages including C, C++, Java, and Smalltalk. We will use IDL in Section 2 to show our interfaces to manufacturing devices, the task sequencer, and other CORBA objects in a manufacturing cell. Our vendor-supplied ORB product is Orbix from IONA Technologies.

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. *CORBA object services* are common services needed in distributed object systems; these services add functionality to the ORB. CORBA object services include, for example, standards for object life cycle, naming, persistence, event notification, transactions, and concurrency. *CORBA common facilities* provide a set of general-purpose application capabilities for use by object systems, including accessing databases, printing files, document management, and electronic mail in a distributed system.

The Internet Inter-ORB Protocol (IIOP) is defined in the CORBA 2.0 specification; it is an open Internet protocol for connecting large distributed applications across the Internet. Specifically, it provides for ORB-to-ORB communication built on top of TCP/IP. IIOP can connect applications running on different computers and is scalable from the LAN to the Internet.

1.3 Software Agents

There are a number of definitions of *software agents* [8]. In the context of this research and agent-based engineering, an *agent* is defined as an autonomous, persistent, encapsulated software component that communicates with other agents using an agent communication language. 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 procedural and declarative knowledge representing some engineering 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. Examples of such agent communication languages are KQML [9], KIF [10] and PDES/STEP (Product Data Exchange using STEP/Standard for the Exchange of Product model data) [11]. By exchanging messages, agents act in a community in order to accomplish tasks.

1.4 Agent Communication Languages

The ARPA Knowledge Sharing Effort has proposed an agent communication language, which addresses a protocol for software agents to exchange knowledge. The agent communication language is comprised of three parts: a messaging language and protocol, the *Knowledge Query and Manipulation Language (KQML)* [9], for the transfer of asynchronous messages; a content language, *Knowledge Interchange Format (KIF)* [10], based on formal logic and predicate calculus; and domain *ontologies* [12], 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; currently, the only limitation on this language is that it is ASCII text. The message layer is speech

act performative, which specifies the protocol for agent communication. Examples of KQML performatives include tell, ask, reply, subscribe, and advertise. The communication layer specifies a set of features that describe 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* (specialized agent that maintains information about other agents). KQML supports both synchronous and asynchronous messages. KQML is also an extensible language, allowing users to define their own performatives as the need arises.

This ARPA Knowledge Sharing effort lacks the low-level technology for enabling distributed collaboration among heterogeneous software components in a wide area environment though the KQML protocol can be built on top of a distributed object middleware, such as CORBA.

1.5 Knowledge Representation

An important element in any distributed engineering system is the mechanism used to represent domain information. Both the academic and industrial communities have recognized the need for a semantically unambiguous and machine-readable language for encoding relevant knowledge [13-15]. Successful integration of multiple engineers and their tools requires the entire enterprise to share a common model of relevant product and domain data. Inconsistencies or discrepancies between the different information models severely degrade overall system performance and hinder scalability. In the manufacturing domain, ambiguities in the product data exchanged between the designer and manufacturing will usually result in a non-functioning product and necessitate another development cycle.

The PDES/STEP standardization effort [11,16] represents a major international drive to develop a mechanism for the exchange of product information using a formal, machine-readable syntax, the *EXPRESS* language [17,18], with unambiguous semantic content (provided by integrated resources) [19-24]. In a concurrent engineering system, information concerning part geometry, topology, features, material properties and tolerances (STEP Parts 41, 42, 43, 45, 47 & 48) [19-24] must be represented and exchanged. Several concurrent engineering systems have been developed which utilize portions of the STEP standard for information representation and exchange. [25-28]

1.6 Agent Architecture and Knowledge Sharing

Realization of a concurrent engineering environment which uses machine-specific capability models requires the exchange of detailed process and product models between the designer and the manufacturing service. Because manufacturing services may be remotely located from the designer, the mechanism used to exchange process capability models must provide reliable, distributed exchange of machine-readable information. Previous collaborative engineering systems have satisfied this need with autonomous software agents which encapsulate the individual system components and communicate using a formal agent-communication language [13-15,26,29]. These systems used agent-based architectures to facilitate modularization and integration of distributed heterogeneous engineering systems. By using the Internet for reliable byte-level data connectivity, a truly world-wide distributed system is created [13].

In Section 2 we present our experiences with manufacturing integration using a distributed object environment. In Section 3 we present our experiences with engineering integration using a software agent architecture. In Section 4 we discuss integration goals and evaluate the distributed object approach and the software agent approach to integrating manufacturing and engineering enterprises. In Section 5 we discuss future research needs in this area.

2 EXPERIENCES WITH INTEGRATION USING DISTRIBUTED OBJECT TECHNOLOGIES

We have implemented a CORBA-based object-oriented integration of a manufacturing cell in *the Sandia Agile Manufacturing Testbed (SAMT)*. We explain the development activities briefly here. This work has been published in greater detail in [30-31].

2.1 Sandia Agile Manufacturing Testbed

The SAMT [32] is a manufacturing research facility at Sandia National Laboratories in Livermore, California. The project objective is to develop agile manufacturing processes for various machined and welded products. The physical component in the SAMT is a networked manufacturing cell containing a conventional milling machine (Cincinnati 4-axis mill), a milling machine equipped with an open architecture controller (Haas 4-axis mill), a lathe, a gas tungsten arc welder, a coordinate measuring machine (CMM), various storage devices, and a Stäubli robot which services some of the manufacturing and storage devices. The computers in the manufacturing cell include both PC's, running the Windows NT operating system, and Unix workstations.

The SAMT is an attempt to address the following product realization cycle: design, planning, cell management, and fabrication. During design, engineers use computer-aided design (CAD) tools, simulation tools, and analysis tools to design parts that meet the customers' requirements. The planning phase includes planning for fabrication, assembly and inspection; fixturing, tooling, and analysis tools assist the process engineer during this stage. 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.

2.2 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 OMG 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 the following tasks:

- entering process plans into the cell from the SAMT process planning node or external source
- directing the development of a production plan from a process plan (*i.e.*, producing a physical realization of a process plan)
- assigning production plans to be scheduled
- maintaining cell schedules, both long-term and short-term
- 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 (on-machine inspection [33], CMM inspection, *etc.*)

- 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
- interface to inventory system.

2.3 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.

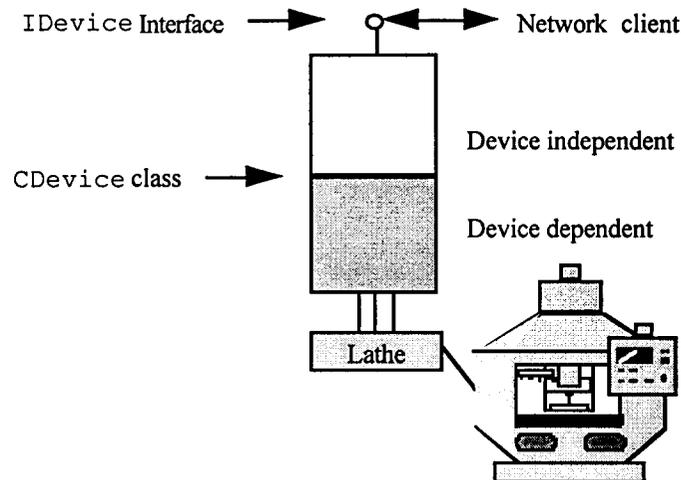


Figure 1. Implementation layers of IDevice.

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 [30] for a more detailed discussion of CDevice.

2.4 IDevice Interface

The IDL for the IDevice interface is as follows:

```
interface IDevice : IBaseDev, IAllocDev, IRunDev, IMovePart
{
    // Locate program database for device.
    IProgDB    GetProgDB();
    // Locate operator console for device.
    IConsole   GetConsole();
};
```

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 to its database of manufacturing numerical control (NC) programs. This type corresponds to another IDL interface, thus, the returned object reference can be used to access the database of NC programs available for the device. IProgDB assumes nothing about the underlying database. Similarly, the GetConsole() function returns a reference to an IConsole object. This object 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. A call to the synchronous RunThisProgram() returns only when the operation is complete, causing the caller to block during processing. An asynchronous call StartThisProgram() immediately returns a boolean success/fail value, indicating whether the operation was successfully initiated. The string output value that would have been returned by RunThisProgram() is instead posted to a passed notification object when the operation is completed. Details can be found in [30]. 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 cell manager. A robot device, for example, has an object reference to its partner in the exchange (set in a previous call to the robot's SetPartner() operation), the robot object can manipulate the lathe directly. Thus a call to the GiveToPartner() operation on the robot results in its assuming responsibility for the transfer, invoking operations on the lathe interface as needed, and awaiting its replies. This kind of peer-to-peer interaction is very natural and easy to implement with distributed objects, and is a real strength of the CORBA technology.

In many cases, the operation of machines in the manufacturing cell is not completely automated. This may be because of limitations in the machine controller, or just because we still rely on the expertise of human machinists in various manufacturing processes. Further, the removal of humans from the manufacturing process is *not* a design goal of our system. Our CORBA-based software architecture in place for the SAMT cell control 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 (using the IProgDB interface) associated with the machine, downloads the resulting NC code into the machine, and runs the code on the machine.

In the case that the machine does not support automated operation, it still must be a part of the information flow in the cell. Thus, we still provide an IDevice object for the machine which implements, for example, the RunNamedProgram() operation. The implementation of this operation is obliged to do whatever is necessary to carry out the task. In this case, the implementation uses an IConsole interface to perform the operation. The IConsole interface provides operations needed to carry on a dialog with a human operator. The IConsole interface gives the machine operator access to an electronic traveler, described in [31]. The IDevice implementation uses these operations to request that the operator run the named NC program on the machine, for example.

As seen in Figure 2, from the cell management software perspective, both automated and manual operations appear the same. This design decision allows us to implement a wider range of manufacturing devices without changing any cell management software.

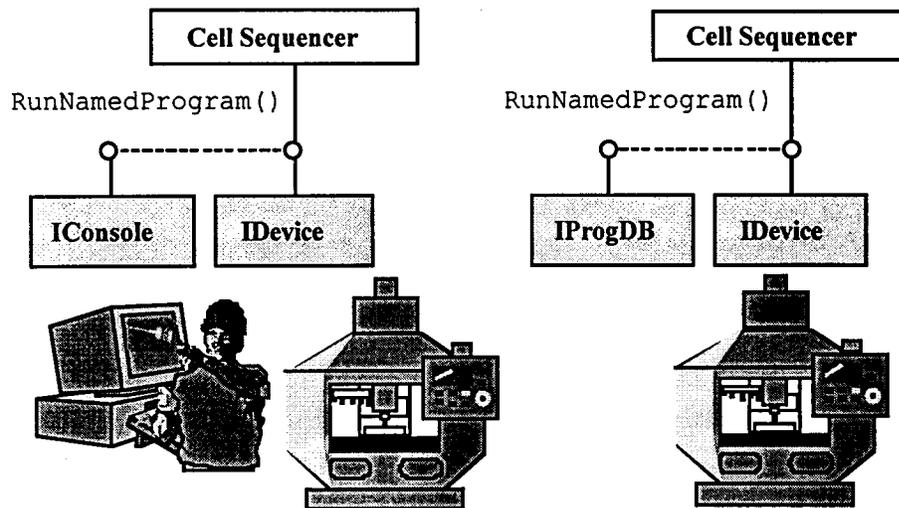


Figure 2. Automated and manual operations.

2.5 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. An abbreviated IDL for the sequencer, ICellSeq, is as follows:

```
interface ICellSeq
{
    readonly attribute string CellName;
    long    AddJob (inout ITraveler, in INotify WhenDone);
    void    Pause (in long JobID);
    void    Resume (in long JobID);
    void    Abort (in long JobID);
    boolean DeleteJob (in long JobID);

    boolean AddDevice (in string DevName, in IDevice Dev);
    boolean AddRobotDevice (in string DevName, in IDevice Dev);
    boolean RemoveDevice (in string DevName);
    IDevice QueryDevice (in string DevName);
};
```

The attribute CellName contains 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. When it comes into the cell, the ITraveler object will contain all of the necessary information to execute the job, including a “script” of high-level instructions to be accomplished 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 of type long, indicating the assigned JobID given by the sequencer; this JobID can then be used to Pause(), Abort(), Resume(), or Delete() a job in the sequencer, even while the task dispatcher is operating. Though the cell sequencer coordinates cell activities, many operations will be accomplished intelligently by the devices. For example, we have mentioned that all material transfer is performed as peer-to-peer object interaction, independent of the supervisory control of the task sequencer.

There are four operations in ICellSeq which allow a client to manipulate devices known to the sequencer: AddDevice(), AddRobotDevice(), RemoveDevice(), and QueryDevice(). Notice that

there are two different operations to add a device to the sequencer: the `AddRobotDevice()` is necessary to distinguish robot and transport vehicles from all other manufacturing and storage devices known by the sequencer. The sequencer must know if a device is a transport device in order to prevent certain deadlock conditions in the cell. By including operations for dynamically adding and removing devices in a cell sequencer, the sequencer will never have to be recompiled or restarted when a new `IDevice` object is available on the network. In theory, this architecture supports a cell sequencer remotely dispatching jobs to any `IDevice` objects.

The current task sequencer is quite simple, with no scheduling optimization criteria. This task dispatcher can be used by a smart scheduling object. This elaborate scheduler will call `ICellSeq` to dispatch jobs once it has optimized on "time", "cost", and "priority" values on jobs.

Another strength of our CORBA implementation is that the current interface and implementation of `ICellSeq` should remain constant as described above even when we add this and other new components to the cell management activities.

2.6 Integrating Enterprise Applications Using CORBA

From our experience, there are many ways to integrate existing and new applications within our CORBA-based manufacturing environment. We discuss these here.

2.6.1 Integrating CAD systems and COTS software

A large number of design, analysis, and manufacturing applications are commercial off-the-shelf (COTS) software packages. For example, at Sandia most mechanical designers use Pro/ENGINEER from Parametric Technology Corporation as the CAD system of choice. Pro/DEVELOP is a software developer's toolkit with a C API for allowing Pro/ENGINEER to access other applications. Through this API, a designer's standard interface can be customized to be a client to any CORBA (or Java) object in the enterprise.

Any commercially available software 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.

2.6.2 Integrating legacy software and databases

In a manufacturing enterprise, it may be necessary to integrate legacy codes into a design or manufacturing environment. In the SAMT environment, we had several legacy design and analysis codes written in FORTRAN. These particular codes were solid engineering codes, yet they did not execute because the I/O required an old graphics terminal. CORBA provided a solution to making this code accessible again, without having to rewrite the code. First, we stripped out any I/O from the source code, and created a FORTRAN library, which could be called from C or C++ code. Second, we defined a CORBA IDL to the code, providing functionality for putting data into the analysis code, executing any engineering functions, and extracting data from the analysis code. This software component then becomes available over the network to any CORBA client. (In Section 2.6.3, we discuss a number of methods for developing client software to CORBA objects.) A new graphical user interface 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 wrapping relational databases as CORBA objects, accessible on the network. Providing CORBA access to databases is critical to the development of software to support a small manufacturing cell or the extended enterprise. New databases can be added to the enterprise by implementing the common IDL database interface. Because CORBA-based applications expose standard interfaces to the network, the application becomes available as a component to other CORBA-based applications and other CORBA clients on the Internet or a restricted corporate Intranet. This functionality allows software developers to 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.

2.6.3 Support for a variety of software clients

While CORBA allows two CORBA objects on two different machines, operating under two different operating systems, and written in two different programming languages to access each other's attributes and methods seamlessly across a network, it also allows many different client applications to be integrated into the environment. We have adopted this software engineering development technique: as much as possible, we attempt to separate the GUI from the "compute engine", so to speak. This philosophy allows for more software reuse, especially given how rapidly new GUI development tools/languages and browser extensions evolve. Examples of the speed of the technology development curve is the decreasing development time between new web browser versions or the increasing growth of the Java programming language in just one year.

Figure 3 shows how CORBA objects can easily be accessed by a variety of client programs on a variety of hardware platforms. In the left-hand column of the figure, CORBA objects are located. In the right-hand column, a number of client technologies are presented. Any client on the right 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 desktop client or a client written in some GUI development language. The figure illustrates this plug-and-play client technology, by matching adapter jacks and client jacks.

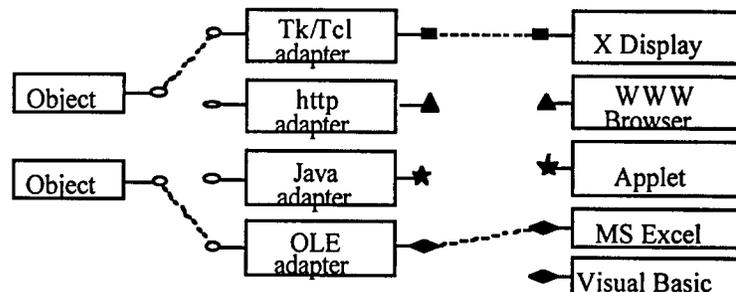


Figure 3. Integrating a variety of software clients into our software architecture.

We have developed GUI's and client applications for cell monitoring and cell management using the Tcl/Tk [34] programming language. The Tcl/Tk applications were developed using an extension [35] of Tcl/Tk, TclDii, which allows the use of distributed CORBA objects from within Tcl. One of the great strengths of Tcl/Tk is the ease with which X-Windows based user interfaces can be assembled, and this CORBA-based extension to Tcl is a very useful tool for rapid-prototyping GUI's and for debugging CORBA components. The resulting client GUI is portable across several platforms.

2.6.4 Integrating to World Wide Web

Today more and more manufacturing 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 in today's corporations and laboratories. Web browsers can be used to navigate Intranets and execute applications, which can be quite sophisticated.

There are several ways in which a CORBA-based distributed system is accessible on an Intranet or the Internet. One powerful way is with the inclusion of Java client applets that communicate with CORBA objects, executing on other computers in the enterprise. Java [36] is an object-oriented, platform-independent programming language that has libraries for Internet access. Most web browsers allow Java applets to execute in the web browser when a web page is downloaded. Since the web browser loads and executes applets on the fly, new applications and modifications to applications can be deployed instantly. Many ORB vendors have solutions for the integration of Java clients (and Java servers); we have used OrbixWeb from IONA Technologies. Java clients supporting IIOP will be transportable across ORBs. Not all Java-CORBA integration strategies

will work through a corporate firewall since some may require an open socket connection from the applet to the ORB daemon; furthermore, some firewalls block Java execution.

Another method of allowing web access to CORBA objects is by using CGI (Common Gateway Interface) scripts which are clients to the CORBA objects. This may be helpful when a firewall is an issue. We have successfully used Tcl, C, C++, and Perl as languages for writing CGI programs that are clients to CORBA servers. These other papers [37,38] discuss the integration of the World Wide Web and distributed objects (*i.e.*, CORBA), and the consensus is that this is a very complimentary and powerful technology integration.

Current releases of Netscape client and server software include Netscape ONE [39], the open network environment, which supports IIOP. Thus any Netscape application can communicate transparently with any CORBA enterprise application.

2.6.5 Integrating to PC applications software

Orbix allows access to OLE [6] objects by providing an OLE adapter provided by IONA Technologies. This adapter allows CORBA objects and OLE objects to interact, thus enabling access to and from many PC and Mac desktop applications. We have used this technology to input numerical data from a Microsoft Excel spreadsheet into a CORBA design code on the network; the resulting output comes into Excel, and the output can be graphed in Excel. We have used the same adapter to build Visual Basic applications that execute on a PC and communicate with CORBA objects across a network. Visual Basic is a good language for rapidly building GUI's and clients for a PC and for interfacing to OLE objects.

2.7 Strengths and Weaknesses of Our CORBA-Based Manufacturing Integration

In our CORBA-based manufacturing environment, the distributed object CORBA interfaces for management of a manufacturing cell are robust, allowing for easy addition, deletion, and updating of manufacturing devices in a plug-and-play fashion. Further, this architecture supports not only manufacturing automation, but human integration by providing console interfaces to manufacturing devices. CORBA enhances the system integration because it is an industry-standard for interoperable, distributed objects across heterogeneous hardware and software platforms. The resulting architecture is scalable and extensible across a wide-area enterprise.

CORBA supports integration with many different information technologies: World Wide Web, OLE, Java, and different programming languages. In time, as commercial software vendors, for example, database vendors, provide CORBA interfaces to various software components, it will be easy to integrate them with our developed manufacturing software. Our experience with CORBA in this environment was quite positive. While it would no doubt have been possible to implement a manufacturing environment using a non object-oriented distributed computing technology, we believe that CORBA substantially eased the implementation of a good, scalable, manufacturing architecture. Specifically, the ease with which object references can be used as return values and calling arguments for operations was central to the architecture we designed. While analogous functionality is presumably possible with other distributed computing technologies, it seems not to be as easy and natural as it is with CORBA.

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. (In a vertical integration, the software components are connected in a vertical fashion, where interfaces between components are well-defined and well understood.) Adding additional tools and databases into the larger manufacturing enterprise can break this type of framework, hence, 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) [40], being developed at Sandia National Laboratories for Defense Programs, is an example of a horizontal enterprise integration framework. Some of the 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
- Standard solutions to common problems, which reduces 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 within a single organization, *i.e.*, Sandia National Laboratories. 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.

As a solution to this problem, SEMATECH has proposed CORBA IDL for a computer integrated manufacturing (CIM) framework [41]; this framework will likely be updated as a result of recommendations made at NIST [42] and experiences at SEMATECH [43]. A key concern we have is the degree to which the SEMATECH architecture supports information-driven manufacturing with human integration and the degree to which outside manufacturing software suppliers adopt these interfaces. A second concern that we have is that because these interfaces have been designed by a consortium committee, the interfaces themselves are rather complex. A third concern of ours is the extent to which other engineering applications can be easily integrated, as in a horizontal framework. A final concern of ours is that no real implementation of the SEMATECH framework exists, even though this has been proposed for several years now.

3 EXPERIENCES WITH AGENT-BASED INTEGRATION TECHNIQUES

In joint research with the Center for Design Research at Stanford University, we proposed an agent-based concurrent engineering architecture [44]. 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 [33] 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.

The goal of our architecture was not to develop “*the solution*” to concurrent engineering, but rather to implement a concurrent engineering system which explores solutions to the challenges listed above, focusing specifically on the need for machine-specific capability information.

3.1 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 [44].

3.2 Agent Architecture

In our agent architecture shown in Figure 4, knowledge and functionality are encapsulated inside agents. The primary benefit of an agent-based architecture is that it facilitates the modularization and integration of large systems comprised of distributed, heterogeneous components. The integration of multiple designers with multiple manufacturing services represents such a system. An agent information infrastructure facilitates the integration of heterogeneous software tools, databases, legacy software, CAD/CAM commercial packages, and newly developed software. Agents logically unify heterogeneous distributed information and knowledge. This particular architecture is geographically distributed, with the designer residing at Stanford University, and the manufacturing facility and on-machine acceptance capability (OMA) residing at Sandia/California.

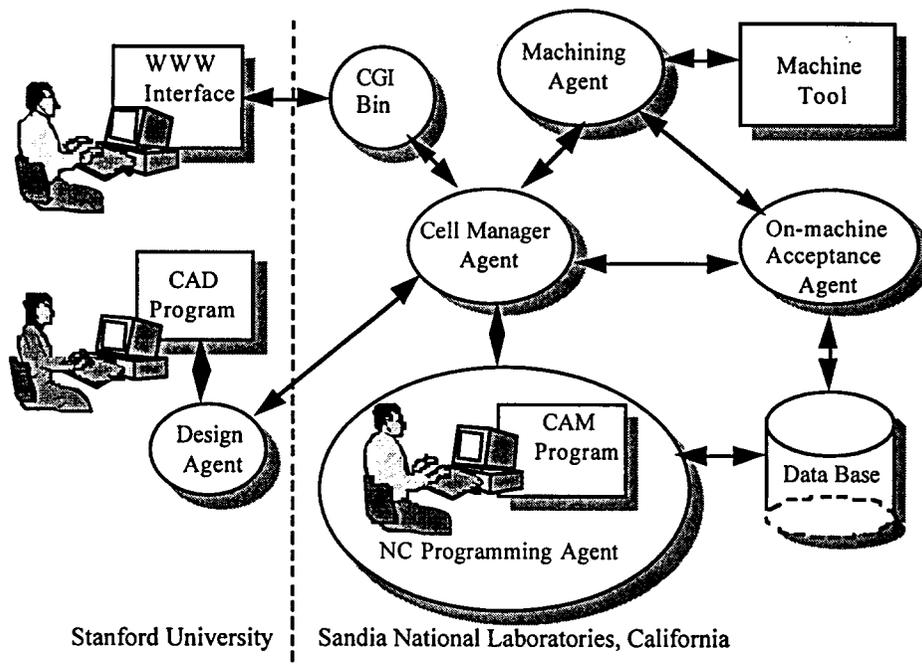


Figure 4. Agent Architecture.

In our implementation, manufacturing process knowledge is represented using portions of the STEP standard and this knowledge will be exchanged as agent messages written in EXPRESS. KQML is used for the outer structure of all agent messages. Agent messages are sent over the Internet using the TCP/IP transport protocol.

Information, in the form of STEP schemas, that is exchanged from the inspection process to the design processes is displayed graphically to designers as Java applets. Java [36] is a platform-independent, interpreted language which allows for the rapid prototyping of GUI's and execution of client programs within a web browser. The designer, therefore, does not need to learn or understand STEP and/or EXPRESS, and the design environment requires no additional software since most web browsers support the transport and display of Java applets. Thus, the features and part information will be mapped to STEP as a method of knowledge exchange, and the designer will view 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 one or more messages to the sending agent.

3.3 Application of Capability Information During Design

The concurrent engineering architecture uses a feature-based design system [29,45,46] which 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. (Note: this architecture can support any commercial or publicly available CAD tool.) As shown in Figure 5, the design agent and constraint manager will 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 will be acquired by the design agent (according to the above protocol) 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 will be loaded into the CAD system. The related process, machine and manufacturing service constraints on the fabrication features will be loaded into the constraint manager.

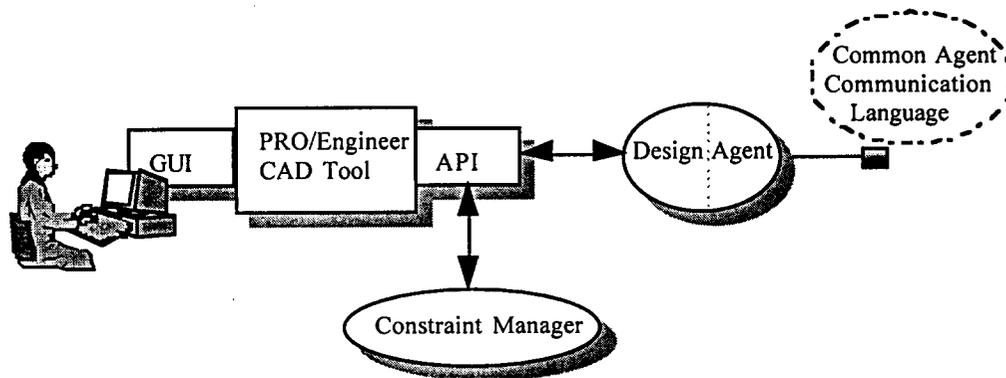


Figure 5. Design Agent: Internal Architecture.

Once the capability information has been acquired and loaded, the designer uses the CAD system to select a feasible stock and uses CSG 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.

The machining agent is implemented with an agent wrapper around the manufacturing cell software presented in Section 2. This wrapper can be seen in Figure 6.

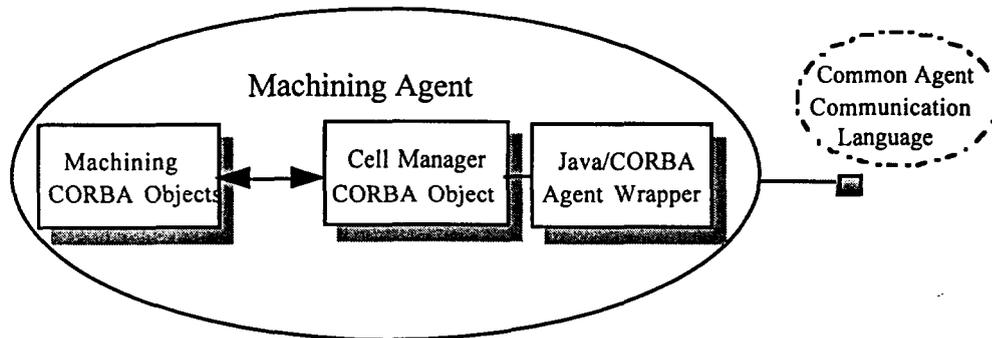


Figure 6. Machining Agent: Internal Architecture.

This architecture is implemented with the Java Agent Template (JAT) [47], a template for creating KQML-speaking agents in Java, designed by Rob Frost. The JAT makes it easy for prototyping software agents in Java; however, this does not have the same services as standard distributed object technologies such as CORBA. Hence, software developers still have to implement many agent services.

3.4 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 is in making this same software available as a CORBA object. Any legacy software and 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.

Java is an excellent programming language for creating agents that are available on the World Wide Web. The Java Agent Template is one example of a KQML agent toolkit written entirely in Java.

Though it is possible to create interfaces to other component software technologies, such as OLE, there are no vendor-supplied tools for doing so, and hence, the software development cost is high.

3.5 Strengths and Weaknesses of Our Agent-Based Integration

We have developed an agent-based architecture and related knowledge representation for using an on-machine inspection process in a concurrent engineering environment. This allows designers to design parts for manufacturability and inspectability, and hopefully reduces the computational cost of process planning. This agent architecture supports a geographically and organizationally distributed system where the design agent resides at one location and the manufacturing agents reside at Sandia National Laboratories in Livermore, California. Agent messages are constructed in an unambiguous agent communication language, and they are sent between a designer and manufacturing services across the Internet. This concurrent engineering approach can be adapted easily to other design environments and also to other manufacturing and inspection processes.

We have shown the feasibility of the knowledge exchange between a design agent and a manufacturing agent, yet there are still many challenges in this area. First of all, 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. Our knowledge exchange is dependent on a simplified on-machine inspection process model, which does not include thermal effects and has a number of simplifying assumptions. Further, the OMA model should be enhanced with the addition of historic on-machine acceptance data. With respect to constraints, we have only proposed to support simple

geometries and features, yet in order to be production worthy, we need to add more complex geometries and features, costs associated with constraints, and fabrication constraints. Currently, fabrication feature constraints are not represented because the same machine is used for both fabrication and inspection; however, the process capability model used to generate the inspection constraints should be largely reusable for creating fabrication constraints and thereby incorporating design-for-manufacturability. This concurrent design architecture will become more powerful as functionality is added to additional agents, and additional manufacturing processes (*e.g.*, process planning and assembly) 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 most industry partners will be reluctant to spend the time and energy for the software development.

The software development costs in this project are high at this time 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 real equivalent development tools such that KQML integrates with other technologies as seamlessly as CORBA does.

4 SOFTWARE INTEGRATION ISSUES

There are a number of software integration issues that should be considered when developing an enterprise architecture. We list them here.

- *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 manufacturing 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 services and applications can be used by other software components. It is necessary that new software can be developed and integrated into the enterprise.
- *Support for distributed applications*: The enterprise is, by definition, distributed. Data and knowledge will be exchanged across the enterprise. It is necessary that software development of distributed components is easy and that the network layer is hidden from the software developer.
- *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*: The enterprise is constantly growing to include new software, new partners, and new computer systems. The software integration architecture must be flexibly enough to adapt to this.
- *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 to engineering and manufacturing tools may have to be authenticated. Second, data might be proprietary or sensitive in nature, and hence, to support the transport of this data over the Internet, it must be encrypted.
- *Maturity of applications*: 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.

- *Standards compliance:* By complying to standards at the middleware level, it makes it easy to plug-and-play applications and to purchase applications that comply to the same standard.
- *Integration with software component software:* It is important to be able to integrate with component software, *e.g.*, OLE, DCOM, CORBA, and DCE, so that desktop applications and existing Unix infrastructures are readily 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 an software design is made.
- *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:* When two or more agents (or objects) talk about a “part”, for instance, both must be referring to the same thing. Without an agreed upon vocabulary, potential problems arise.

We first discuss integration strategies or frameworks. We then revisit the specific integration problems that we encountered in the CORBA-based manufacturing environment and the agent-based concurrent engineering environment. Then we survey the CORBA-based approach and the agent-based approach to integration with respect to the above criteria.

4.1 Integration Strategies

Neither of distributed object approach nor the software agent approach alleviate the need for an integration strategy or framework. CORBA and KQML are only tools, and still the software development in the enterprise must be done with an integration strategy. In the CORBA-based manufacturing cell, our integration strategy was to provide well-defined interfaces in a vertical integration of a manufacturing cell. In the agent-based engineering, our integration strategy was to define specific interfaces for each agent or service in the system. 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.

Hence, with either approach, it is critical to have an enterprise integration framework to support scalability of the enterprise.

4.2 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. Furthermore, 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. Finally, as we mentioned in Section 4.1, 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.

4.3 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.

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

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, using an object-oriented programming language, such as Java, can be used to build KQML-speaking 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 home-grown, academic software, which was perfect for our showing a proof-of-concept rather providing a production quality manufacturing environment. However, the agent-based approach was not built on top of middleware software such as CORBA, and the software developers had to worry about low-level network details. By using the JAT to integrate an agent-based enterprise, some of the development effort is done in this template, relieving the software developer of low-level details. On the other hand, the JAT itself is supported by an academic department and the Java programming language itself is currently undergoing changes.

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 implementations of this OMG security specification at this time. 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.

As discussed in Section 2, 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 usually not 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 ORB's are planned. KQML is "free", so to speak, though the in-house software development costs to build the underlying infrastructure are high.

As we have mentioned in Section 4.2, CORBA is not an agent communication language and there is no agent facility in CORBA, and thus if an agent architecture were developed on top of CORBA, some protocol for how agents communicate must be used. KQML is an agent communication language.

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 3, 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.

4.5 Using Java To Integrate Manufacturing and Engineering Enterprises

Many researchers [48] 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) [49]. Java has the following features which are useful in a distributed object system: close integration with the web, security, multi-threaded language, absence of common error-prone language semantics and addition of useful language features (for example, exception handling and garbage collection), and portability across hardware and operating system platforms.

Java has the following strengths beyond CORBA:

- *Mobility of code*: allowing any Java application to dynamically download the classes of remote objects, their interfaces, stubs, parameter, and return values.
- *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 issued a request for proposal (RFP) to address this issue.

CORBA, however, is still a more mature technology, had 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, among them, naming, event, security, and notification. The Java RMI is not as mature as CORBA, and in time many of these services may 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 other distributed object technologies*: Java does not integrate with OLE and DCE as CORBA does 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. However, there are many useful ways to use Java within a CORBA-based environment (see Section 2.6.4). The Java/RMI and CORBA technologies 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, and applet can be an IIOP client to a CORBA server, regardless of the use of the Java RMI. Further, Javasoftware 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.

4.6 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 another agent communication language, can be layered on top of CORBA to take advantage of the strengths of this distributed object technology. We believe

at this time, however, that KQML and agent communication languages are not yet developed to the extent that they should for integrating into the enterprise, and the inclusion of software agents into a distributed framework should be explored more fully.

Some considerations when making a decision about enterprise integration include enterprise integration costs, the size of the enterprise that needs to be integrated, and the extent to which standards are important.

5 CONCLUSIONS AND FUTURE WORK

We have presented two different approaches to integrating engineering enterprises: a CORBA-based manufacturing environment and an agent-based architecture for design and manufacturing. We have also come up with a list of criteria that are important for 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 the two technologies is needed for achieving advanced, intelligent integration.

6 ACKNOWLEDGMENTS

The author expresses gratitude to a number of people at Sandia National Laboratories. Norm Breazeal, now retired, was a great visionary and supporter for this work. Jim Costa has been the program manager for this work. Bob Whiteside and Paul Klevgard are collaborators on the distributed object manufacturing software in the SAMT. Andy Hazelton provided his on-machine acceptance models as a formal model for intelligent agent exchange. Finally, Mark Cutkosky and Rob Frost at the Center for Design Research at Stanford University collaborated on the design for manufacturability and inspectability agents in our agent architecture.

7 REFERENCES

1. R. Nagel, and R. Dove, editors, *21st Century Manufacturing Enterprise Strategy*, An Industry-Led View, Iacocca Institute, Lehigh University, Bethlehem, Pennsylvania, 1991.
2. M. J. Wozny, and W. C. Regli, guest editors, "Computer Science in Manufacturing", *Communications of the ACM*, Vol. 39, No. 2, February 1996.
3. B. L. Maia Goldstein, editor, *Proceedings of the SPIE Conference on Plug and Play Software for Agile Manufacturing*, Boston, Massachusetts, November 1996.
4. R. Orfali, Harkey, D., and Edwards, J., *The Essential Distributed Objects Survival Guide*, John Wiley and Sons, Inc., New York, New York, 1996.
5. The Common Object Request Broker: Architecture and Specification, OMG Technical Document PTC/96-03-04, Object Management Group, Framingham, Massachusetts, July 1995.
6. K. Brockschmidt, *Inside OLE 2*, Second Edition, Microsoft Press, Redmond, Washington, 1995.
7. Apple Computer, Inc., *OpenDoc Programmer's Guide for the Mac OS*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.
8. J. M. Bradshaw, editor, *Software Agents*, The MIT Press., Cambridge, Massachusetts, 1997.
9. 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.
10. 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.

11. 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.
12. 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.
13. M. Cutkosky, R. Engelmores, 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. J. Owen, *STEP: An Introduction*, Information Geometers Ltd, Winchester, UK, 1993.
17. P. Spiby, editor, *ISO 10303 Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 11: Description Methods: The EXPRESS Language Reference Manual*, ISO DIS 10303-11:1992(E), July 1992.
18. Schenck and Wilson, *Information Modeling: The EXPRESS Way*, Oxford University Press, 1994.
19. ISO 10303-41, *Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 41: Integrated Generic Resources: Fundamentals of Product Description and Support* ISO DIS 10303-41.
20. ISO 10303-42, *Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 42: Integrated Generic Resources: Geometric and Topological Representation*, ISO DIS 10303-42.
21. ISO 10303-43, *Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 43: Integrated Generic Resources: Representation Structure*, ISO DIS 10303-43.
22. ISO 10303-45, *Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 45: Integrated Generic Resources: Materials*, ISO DIS 10303-45.
23. ISO 10303-47, *Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 47: Integrated Generic Resources: Tolerances*, ISO DIS 10303-47.
24. ISO 10303-48, *Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 48: Integrated Generic Resources: Form Features*, ISO DIS 10303-48.
25. C. Feng and A. Kusiak, "Constraint-based Design of Parts", *Computer-Aided Design*, 27(5):343-352, 1995.
26. D. Goldstein, "An Agent-based Architecture for Concurrent Engineering", *Concurrent Engineering: Research and Applications*, 2: 117-123, 1994.
27. P. Gu and K. Chan, "Product Modelling using STEP", *Computer-Aided Design*, 27(3): 163-179, 1995.
28. T.-H. Liu and G. Fischer, "Developing Feature-based Manufacturing Applications Using PDES/STEP", *Concurrent Engineering: Research and Applications*, 1: 39-50, 1993.
29. 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.

30. 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.
31. 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.
32. 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.
33. A. J. Hazelton, "On-Machine Acceptance of Machined Components", *Proceedings of the 10th Annual Meeting of the American Society of Precision Engineering*, October 1995.
34. J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.
35. 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.
36. K. Arnold and J. Gosling, *The Java Programming Language*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1996.
37. O. Rees, N. Edward, *et. al.*, "A Web of Distributed Objects", ANSA, Cambridge, United Kingdom, November 1995.
38. A. Vogel, "WWW and Java Threat or Challenge to CORBA?", CRC for Distributed Systems Technology, University of Queensland (Brisbane), Australia, 1996.
39. Netscape Communication Corporation, "The Netscape ONE Development Environment Vision and Product Roadmap", white paper, Netscape Communications Corporation, Mountain View, California, July 1996.
40. <http://www-collab.ca.sandia.gov/pre>.
41. SEMATECH Computer Integrated Manufacturing Application Framework Specification 1.2, SEMATECH Technology Transfer #93061697E-ENG, Austin, Texas, 1995.
42. S. L. Steward and J. A. St. Pierre, "Experiences with a Manufacturing Framework", *Proceedings of the OOPSLA '95 Business Objects Workshop*, Springer-Verlag, London, 1996.
43. D. Doscher, and R. Hodges, "SEMATECH'S Experiences with the CIM Framework", *Communications of the ACM*, Vol. 40, No. 10, October 1997.
44. 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.
45. P. K. Wright, "Principles of Open-Architecture Manufacturing", Engineering Systems Research Center, ESRC 94-26, University of California at Berkeley, October 1994.
46. 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.
47. 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.
48. K. U. Reddy, "Java and Distributed Computing", *Proceedings of the Grace Hopper Celebration of Women in Computing Conference*, San Jose, California, September 1997.
49. Sun Microsystems, Inc., "Java Remote Method Invocation Specification", December 1996.

UNLIMITED RELEASE

INITIAL DISTRIBUTION:

1	MS 9003	D. L. Crawford, 8900
1	MS 9011	R. E. Palmer, 8901
1	MS 9003	D. L. Lindner, 8902
1	MS 9011	P. W. Dean, 8910
1	MS 9012	J. E. Costa, 8920
1	MS 9214	N. M. Berry, 8920
1	MS 9214	E. J. Friedman-Hill, 8920
1	MS 9012	J. A. Friesen, 8920
10	MS 9012	C. M. Pancerella, 8920
1	MS 9012	R. A. Whiteside, 8920
1	MS 9012	S. C. Gray, 8930
1	MS 9019	B. A. Maxwell, 8940
1	MS 9011	J. Meza, 8950
1	MS 9011	D. B. Hall, 8970
1	MS 9001	T. O. Hunter, 8000
	Attn:	J. B. Wright, 2200
		M. E. John, 8100
		L. A. West, 8200
		W. J. McLean, 8300
		R. C. Wayne, 8400
		P. N. Smith, 8500
		P. E. Brewer, 8600
		T. M. Dyer, 8700
		L. A. Hiles, 8800
		K. C. Olsen, 11600
1	MS 0188	D. Chavez, LDRD Office
3	MS 9018	Central Technical Files, 8940-2
4	MS 0899	Technical Library, 4916
1	MS 9021	Technical Communications Dept. 8815/Technical Library, 4916
2	MS 9021	Technical Communications Dept. 8815 for DOE/OSTI