Carnegie Mellon University
**Software Engineering Institute**

# Distributed Object Technology With CORBA and Java: Key Concepts and Implications

Kurt Wallnau
Nelson Weiderman
Linda Northrop
*June 1997*

TECHNICAL REPORT
CMU/SEI-97-TR-004
ESC-TR-97-004

**Technical Report**
CMU/SEI-97-TR-004
ESC-TR-97-004
June 1997

# Distributed Object Technology With CORBA and Java: Key Concepts and Implications

Kurt Wallnau
Nelson Weiderman
Linda Northrop

Product Lines Systems

**Software Engineering Institute**
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

[illegible]

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

# Table of Contents

## List of Figures

# Distributed Object Technology with CORBA and Java: Key Concepts and Implications

**Abstract:** The purpose of this report is to analyze the potential impact of distributed object technology (DOT) on software engineering practice. The analysis culminates with the conclusion that the technology will have a significant influence on both the design and reengineering of information systems and the processes used to build them. We see a profound impact and fundamental change in both technical thinking and practice as a result of the related technologies we group together as DOT.

# 1 Introduction

Distributed object technology (DOT) is defined quite broadly for the purposes of this report. We consider it to include three technologies that have synergistically merged to provide something quite powerful—something greater than the sum of their parts. Those three technologies, in order of their emergence, are

- object technology
- distribution technology
- Web technology

Object technology (OT) was introduced to the computing mainstream in the late 1970s by Adele Goldberg and Alan Kay with a language called Smalltalk [Goldberg 83]. In the object-oriented model, systems are viewed as cooperating objects that encapsulate structure and behavior and belong to hierarchically-constructed classes. In the last twenty years the benefits of object-oriented technology have been demonstrated. Object-orientation has changed the way today's systems are built and maintained. Substantial experience and tooling exists for object-oriented analysis (OOA), object-oriented design (OOD), and object-oriented programming (OOP). The transition from more traditional structured approaches is not complete, nor has OT been fully exploited, especially in legacy systems that predate objects. But OT is maturing rapidly and is well-accepted as a technology that addresses complexity, improves maintainability, promotes reuse, and reduces life-cycle costs of software.

Distribution technology (DT), which involves autonomous computers that are connected by a network and have no shared physical memory, dates to the middle 1980s. The advent of smaller, more powerful, and less expensive CPUs precipitated an interest in moving applications from mainframes to PCs and workstations, and in distributing functionality over multiple communicating computers. Over time, the notion of distribution has generally migrated from tightly coupled, geographically close, homogeneous machines to more loosely coupled, geographically remote, heterogeneous machines. Traditional DT employed a "client/server" model where computations on one machine (the "client") invoke computations on another machine (the "server") in a manner often viewed as a remote procedure call (RPC). The server process is a provider of services; the consumer is a consumer of services. The client/server model is concerned primarily with the problems of distributing function across local and wide area networks through devices such as pipes and sockets.

More recent developments in distribution technology have involved the use of *middleware* technology. Although middleware is broadly defined (see Rymer [Rymer 96]), for the purposes of this paper, middleware—sometimes known as glue—refers to technologies that facilitate integration of components (objects) in a distributed system. While the details of these technologies vary considerably, middleware products usually provide a run-time infrastructure of services used by components to interact with each other. The focus of DT has been to make the computing environment more and more transparent with respect to the locus of both the computing engines and the objects of computing.

Web technology (WT) was born in the 1990s and has quickly caused an explosion in the use of the Internet. The advent of Web browsers and search engines has caused an excitement about computing reminiscent of that surrounding the introduction of the personal computer in the middle 1970s. Web technology has universalized and globalized computing as never before. We are not concerned here with the most popular aspect of Web technology, namely the ability to create pages of information that can be accessed anywhere in the world with a slick user interface. Rather, this paper is concerned with the use of Web technology to build large application systems or to reengineer old application systems. The advent in 1995 of the object-oriented language Java brought Java "applets" that provide "executable content" to previously static Web technology.

The combined use of these technologies (OT, DT, and WT) changes the fundamental way in which systems are engineered. Objects have been added to networks and integrated with middleware, often called Object Request Brokers (ORBs). Objects are being used in distributed systems to represent units of distribution, movement, and communication. The addition of Web technology provides application portability. There is in effect a new paradigm of computing, at the heart of which are interoperable objects. The level of abstraction is raised yet another notch, thus enabling greater leveraging of developer resources. Just as we have moved up the abstraction scale in programming, databases, and user interfaces, DOT gives us the opportunity to move further and faster in the distribution and globalization of application development using all the previous technologies transparently.

The remainder of this paper is divided into four parts. The first part provides some background on two specific exemplars of DOT. The second part describes the impact of DOT on software systems design. The third part explores the impact of DOT on software development and engineering processes. The fourth part provides some conjectures about future trends for DOT and its use. The last part gives a summary.

# 2 CORBA and Java

We will study two technology exemplars, namely the Common Object Request Broker Architecture (CORBA) for ORB technology and Java applet technology for Web technology. We are using exemplars to make the discussion more concrete. We are using these particular exemplars because they are the most visible and popular representatives of their respective classes. While this decision may somewhat color our discussion of DOT with the peculiarities of CORBA and Java, our intent is to address the broader aspects of DOT.

To provide the necessary perspective, we will give a brief explanation of the origins and current states of these exemplars. They arose from two broad classes of DOT progenitors: Operating systems and distributed systems infrastructures influenced CORBA, and programming languages and the Web gave way to Java.

## 2.1 Operating Systems/Distributed Systems Influence and CORBA

The key DOT concept stemming from operating systems is *interconnection*. Many levels of abstraction can be used to describe the connection between machines on networks. As early as 1980, an Open Systems Interconnection (OSI) architecture [Zimmerman 80] defined a hierarchy of seven layers, each representing a collection of communication functions from bits and bytes at the lowest level to application protocols at the highest level. "Sockets" between processes on the same or different machines became the primary mechanism provided by operating systems to connect distributed systems in networks. Remote procedure calls (RPCs) became a higher-level (more abstract) mechanism for using the sockets.

As object technology became more popular through the 1980s there was more interest in bundling the concept of objects with the concept of transparent distributed computing. Objects, with their inherent combination of data and behavior and their strict separation of interface from implementation, offer an ideal package for distributing data and processes to end-user applications. Objects became an enabling technology for distributed processing. In the early 1990s an international trade association called the Object Management Group (OMG) [OMG 97] defined a standard for the distribution of objects. The OMG defined the Common Object Request Broker Architecture (CORBA), which provided a standard by which OT could be used in distributed computing environments. The latest version of this standard, CORBA 2.0, addresses issues related to interface, registration, databases, communication, and error handling. When combined with other object services defined by the OMG Object Management Architecture (OMA), CORBA becomes a middleware that facilitates full exploitation of object technology in a distributed system. However, if we were to characterize CORBA technology in the simplest possible language, it would be to say it is an object-oriented RPC.

CORBA is concerned with interfaces and does not specify implementations; CORBA is a standard for which there are many (a dozen at present) current products, and each is referred to as an ORB. Among them are ORBIX by IONA Technology, NEO by SunSoft, VisiBroker by Visi-Genic, PowerBroker by Expersoft, SmallTalkBroker by DNS Technologies, Object Director by

Fujitsu, DSOM by IBM, DAIS by ICL, SORBET by Siemens Nixdorf, and NonStop DOM by Tandem. The OMG sponsors a permanent showcase on the Web to demonstrate the interoperability between ORBs from various vendors according to the CORBA 2.0 specification [CORBAnet 97]. The major product entry that is not CORBA-compliant is Microsoft's Distributed Component Object Model (DCOM). DCOM is a competitor to CORBA; it is not discussed in this report (although many of the points made concerning the impact of CORBA apply to DCOM.)

## 2.2 Programming Language/Web Influence and Java

The key concept that arises from combining languages and Web technology is that of "mobile objects" or "executable content," which is possible with Java. Java is a pure object-oriented programming language that is designed to look and feel like C++ but is at its core like Smalltalk. As a language Java is appealing but not revolutionary; Java is strongly typed, provides garbage collection, dynamic linking, interfaces, packages, exception handling, and built-in support for threads at the language level. "Java is a blue collar language" that includes features from C, C++, Smalltalk, Simula, Mesa, and Modula [Gostling 96]. It is much more minimalist than Ada 95 and C++, and is not a hybrid language as are Ada and C++ and many of the other object-oriented languages (CLOS, Visual Basic, Object Pascal, Objective C). It is more traditional in its syntax and semantics than Smalltalk, and because it borrows from so many other languages, Java feels familiar. Java is architecture-neutral, and so Java applications are completely portable. Rather than producing machine-specific instructions, Java is translated into vendor-neutral bytecode. The Java Virtual Machine, which executes as an operating system process or a Web browser, translates the bytecode into the machine-specific instructions.

The real promise of Java is its use and delivery in the context of the Web. With the release of Netscape 3.0 on August 12, 1996, "Java applets" became executable on all major platforms that Netscape supports. Namely, it became possible to execute Java programs on Unix boxes, WinTel machines, and Macintoshes by using a Web browser and by accessing pages on the Web that have executable content (the applets). The Web thus becomes the delivery vehicle for programs that can execute on a client machine and can reach back to the server for customized information retrieval.

This power is not derived from the language per se, but from the architecture-neutral approach used by Java. The Java Virtual Machine, including the Java interpreter, is part of the version of Netscape Navigator delivered for each platform. First, an applet written in Java is transported to the browser and then the applet is executed using the browser's interpreter. While the execution of applets is quite slow at present, there are "just-in-time" (JIT) Java compilers on the near horizon that will compile the bytecode on the client machine before executing them.

Other key components of Java technology are the application programming interfaces (APIs) that are standard, but separate from the language. These include the Abstract Window Toolkit (AWT) that provides a complete set of classes for writing graphical user interface (GUI) programs. In addition there are classes for applets, I/O, networking, and debugging. With these additional classes, Java is currently more “net aware” and “Web aware” than other object-oriented programming languages.

Both CORBA and Java offer substantial support for the construction of distributed systems. Together, CORBA and Java present application developers with the unprecedented ability to build new systems and to reengineer legacy systems. DOT provides architectural choices not previously available and suggests new development and engineering processes. In the next sections we explore these new capabilities and processes.

# 3 Impact of DOT on Software System Design

Software architecture is a topic of considerable interest in the 1990s. While many different perspectives exist, there is general agreement that software architecture deals with design abstractions for system-level structural concerns. By "system-level" we mean something larger than a single computer program. As noted by Garlan and Shaw, such concerns include "gross organization and global control structure; protocols for communication; synchronization, and data access; physical distribution" [Garlan 93].

DOT provides mechanisms that address many of these concerns in a way that builds upon already-proven abstraction mechanisms (objects, messages, inheritance) available in OT. The significance of these distribution extensions to OT is that software designers now have at their disposal the means of expressing abstract system designs and, more importantly, now have tools for quickly fabricating working versions of these designs. That is, there is now a more direct path from abstract architectural concepts to concrete implementations of these concepts.

It is useful at this point to draw a comparison between the use of DOT for architectural patterns and the use of middleware technology. Recall that middleware refers to technologies that facilitate integration of components in a distributed system. In software architecture terms, middleware products provide services corresponding to well-known architectural idioms[1] (e.g., blackboard and implicit invocation). DOT provides more general, but lower-level building blocks upon which a variety of middleware services can be implemented. Thus, one would use a DOT such as CORBA to define a collection of related object types that implement a message-based implicit invocation system. Further, the OT heritage of DOT makes it feasible (and possibly desirable) to define these middleware object types more abstractly in the form of a *design pattern* that can be tailored to specific design problems [Gamma 95]. It is also possible to use DOT to develop architectural patterns that do not correspond to middleware, but instead represent proven design solutions for specific system design problems.

To illustrate the use of DOT to express architectural abstractions and to demonstrate the potential impact of CORBA on software system design, in Section 3.1 we briefly describe three case studies. Each of the case studies used commercially-available implementations of CORBA. In Section 3.2 we shift our attention to the impact of Java on software system design. Although experiences with architectural uses of Java are more limited that those with CORBA, they emphasize different aspects of distribution than CORBA, and even limited experiences with Java reveal the potential impact on design concepts.

1. We reserve the term *pattern* to refer to an object-oriented style of describing architectural idioms and their implementations.

## 3.1 Three Case Studies of CORBA-Based Designs

### CORBA Case Study 1: Middleware for Distributed, Real-Time Simulation

The Defense Modeling and Simulation Organization (DMSO) is leading the development of the High-Level Architecture (HLA), an architecture for modeling and simulation within the US Department of Defense (DoD). One component of the HLA is the Run-Time Infrastructure (RTI), a collection of services supporting (among other things) real-time interoperation among simulations, as for example in joint simulation exercises [Calvin 96]. A prototype RTI was imple-

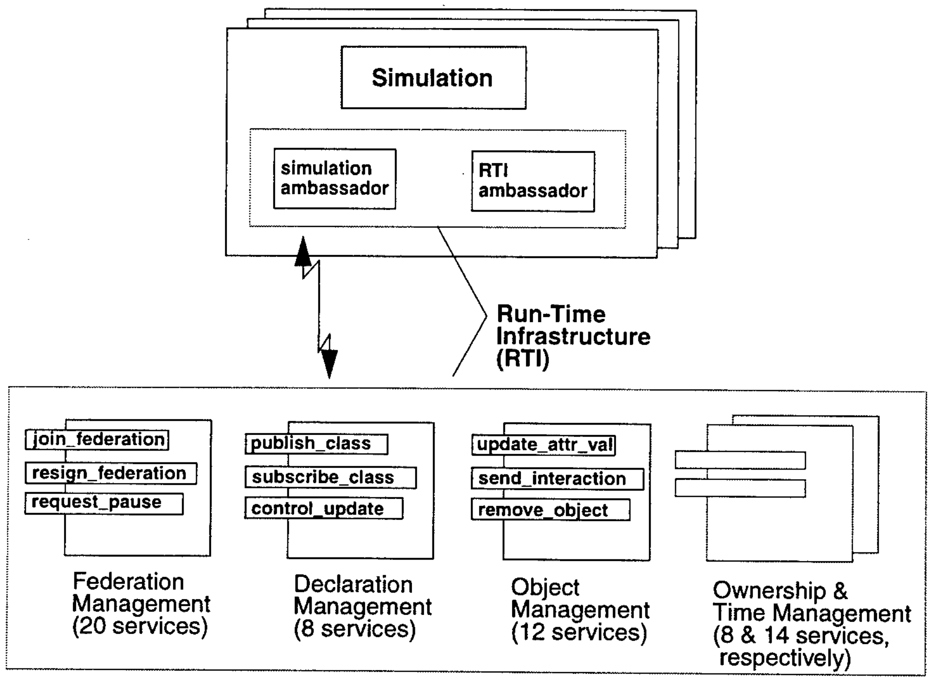
Simulation
simulation ambassador
RTI ambassador
Run-Time Infrastructure (RTI)
join_federation
resign_federation
request_pause
publish_class
subscribe_class
control_update
update_attr_val
send_interaction
remove_object
Federation Management (20 services)
Declaration Management (8 services)
Object Management (12 services)
Ownership & Time Management (8 & 14 services, respectively)

**Figure 1: DMSO Run-Time Infrastructure—A Custom Middleware Solution**

mented by Mitre and Lincoln Labs using Iona Technology's ORBIX, a CORBA-compliant ORB. Distributed joint simulation presents challenging technical problems that require innovative design solutions. Among these difficult design problems are interoperation of heterogeneous simulations (e.g., real-time versus faster than real-time simulations, different simulation technologies, different execution platforms), and management of distributed simulations (e.g., exercise set-up, and run-time coordination).

In order to illustrate the key architectural concepts, we present a simplified illustration of the RTI in Figure 1. The lower box contains the various management services in a custom middleware solution tailored to distributed simulation involving heterogeneous simulation models. Simulations are viewed as "black boxes" that execute remotely from the central middleware services. Interactions between the middleware and the remote simulation are mediated by two objects: a simulation ambassador, which acts as a bridge between the RTI and the simulation, and an RTI ambassador, which provides local implementations of selected RTI services.

CORBA was used in two ways to address some of the design challenges listed above. First, platform and simulation technology heterogeneity were addressed by the use of the CORBA Interface Definition Language (IDL), a language and (platform/network) architecture-neutral interface specification language. IDL interface descriptions conform to the distributed object model that underlies CORBA, a model that describes how object class hierarchies are defined and how object instances are accessed and their methods executed.

The second use of CORBA has already been mentioned: It was used as a basis for designing a custom (domain-specific) middleware system. That is, the RTI middleware expresses abstractions that deal directly with the difficult issues of coordination of simulations—these services go far beyond a more primitive publish/subscribe pattern or even real-time message buses that might otherwise have been used to support interoperation of remote distributed, heterogeneous components.

There is a subtle point worth noting about the nature of the RTI services: Their focus on *coordination* rather than *functionality* represents an important architectural design concept that will be reinforced in the following case study illustrations. The underlying idea is that the difficult design issues in distributed systems are related to coordinating the execution of functionality—e.g., synchronization, currency, consistency—and that design abstractions should therefore focus on these issues, rather than on questions of what functionality is provided by specific components. Abowd provides a discussion of these ideas [Abowd 93]. However, the key point is that DOT provided the means for developing a novel and innovative design solution.

It is important to note that the DMSO/RTI experience with DOT is not completely unblemished. A number of severe problems with the RTI design were discovered, but only after an initial implementation of the prototype was released. Initially the problem was reported to have been attributed to the use of DOT, e.g., deadlocks were arising between objects. A dispassionate (but perhaps unfair) conclusion would be that the designers of the RTI were not sufficiently familiar with either OT or distributed systems design. More likely, however, the designers were not sufficiently familiar with the confluence of OT and distribution. One significant issue is that the CORBA specification does not address real-time issues, and the ORBIX product used did not provide real-time extensions to CORBA. This deficiency of CORBA introduced further complexity in the design and implementation, and no doubt contributed to the initial design problems.

## CORBA Case Study 2: Structural Patterns for Distributed Component Integration

The second case study is a prototype manufacturing engineering design toolkit developed by the SEI, NIST Manufacturing Engineering Laboratory, and Sandia National Laboratory. Wallnau describes details of this case study [Wallnau 96]. The essential design problem was to use CORBA to modernize a system of (legacy) manufacturing engineering design tools developed over many years by Sandia, called the SEACAS toolkit. The modernized toolkit would make these non-distributed, platform-specific tools remotely accessible (including access from a wide-area network) to end users on different platforms. Further, the tools could be integrated at the level of remote objects.

Unlike the DMSO/RTI, the design approach taken in the SEACAS modernization illustrates the use of DOT to develop *design patterns* (albeit simple ones). In contrast, the RMI implemented a specific middleware solution. Patterns are abstract: They do not describe implementations, but rather design solutions to specific design problems that tend to occur repeatedly. The seminal work on OT design patterns describes patterns that address intra-program design issues [Gamma 95]. Design patterns for distributed systems problems are also emerging [Schmidt 96a]. The design problems addressed by the SEACAS patterns can be summarized:

- SEACAS tool services needed to be available through a wide-area (Internet-based) network; however, the tools were not designed to execute in a distributed, let alone wide-area distributed, setting.
- Individual tool runs might require several hours of wall clock time, for example where finite element analysis is applied to three dimensional solid models. In these circumstances it was essential that unreliable Internet connections not interfere with using the tools.
- SEACAS has many tools, developed over many years, that provide a wide variety of manufacturing design services. It was impractical, given available resources, to develop a comprehensive object model that describes this functionality.

Figure 2 illustrates the key concepts of the SEACAS *ConsistencyManager* pattern. A "consistency manager" models an external off-the-shelf component as a state machine that is either in a consistent or inconsistent state. The state machine has two data representations: an external representation, which a client can manipulate, and an internal representation, which is derived by the component from the external representation. The `specification()` operation allows clients to read and write external data. The operation used to make this data consistent is `update()`. The `done()` and `consistent()` operations are probes that the client can use to determine whether an update is in progress, and, if not, whether the object is consistent. Tool output is directed to a buffered output channel that can handle asynchronous data input and output. There are a few other operations not illustrated (for example, operations that allow clients to examine diagnostic data).

The pattern, and its implementation in CORBA, illustrate several aspects of how DOT supports innovative design.

First, we note that the ConsistencyManager pattern had a positive impact on another difficult problem often associated with the integration of off-the-shelf components, namely the question of how to "wrap" components in order to make them work together. To date, component wrapping has been an undisciplined and *ad hoc* procedure (see Wallnau[1] for a discussion of attempts to make this a more rational process). The ConsistencyManager pattern, however,

1. Wallnau, K.; Mooreman-Zaremski, A.; & Clements, P. "Correcting, Identifying, and Avoiding Interface Mismatch: Theory and Practice." Available from Kurt Wallnau, kcw@sei.cmu.edu.

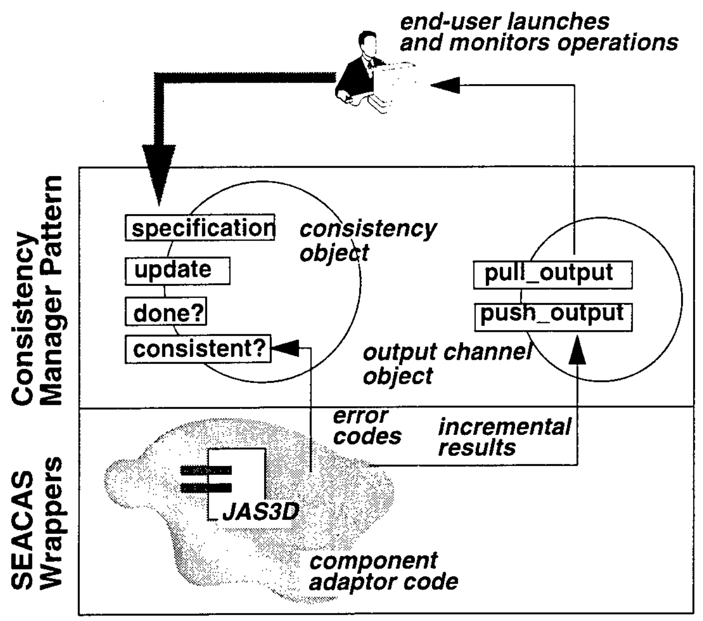

**Figure 2: ConsistencyManager Pattern and Use for Integrating Wide-Area Distributed SEACAS Components**

provided a uniform set of wrapping requirements. Rather than having to view each wrapping problem in terms of unique tool functionality, each tool needs only to be wrapped in such a way as to interact with a very limited repertoire of coordination primitives already described by the patterns. This both simplifies the wrapping process and provides insight into pattern-based techniques for "qualifying" components for fitness for use (i.e., Can they support a particular coordination model described by the design pattern?).

Moreover, this pattern addresses the coordination aspects of the design problem, i.e., how the SEACAS tools will execute and interact with long design sessions and potentially unreliable (wide-area) network connections. For example, `update()` was implemented as a non-blocking operation to minimize the need for synchronous client/object connections; and the object implementation was made concurrent so that the probes could be executed while an update was in progress. These design decisions, reflected in the ConsistencyManager pattern, allow non-distributed tools to behave in predictable ways when executed in a wide-area network setting. Both the CORBA specification and vendor implementations of CORBA provide a wide variety of concurrency and synchronization primitives upon which flexible patterns can be designed and implemented.

Lastly, OMA also provided mechanisms to address another critical design problem associated with long design sessions—namely, the need for objects to persist. By using OMA persistence services (recall that CORBA is just the communication infrastructure of the OMA), the lifespan of individual consistency objects would be independent of the execution of any SEACAS component and independent of the duration of client connections to these objects. In effect, then, the SEACAS prototype used DOT to combine the use of architectural patterns (the ConsistencyManager pattern) within an overarching architectural style referred to as a "repository style," i.e., clients view SEACAS as a repository of interrelated, persistent objects that implement the ConsistencyManager pattern.

## CORBA Case Study 3: Hybrid Structural/Functional Patterns in Embedded Controller

The final CORBA-based case study describes an innovative design of a product line architecture for high-end printers. The product line is being developed by a significant vendor (for our purposes heretofore referred to as PCo) in the document printer industry. There are two significant design problems that were addressed in this case study; the first dealt with performance requirements, and the second dealt with product line reuse requirements.

What distinguishes high-end printers from their less expensive counterparts is speed—simply put, pages printed per minute. Fast printers will outsell slower printers, even if the slower printers have substantially more functionality. Thus, the primary consideration in this design was how to maximize software concurrency. To this end, the hardware architecture consists of (up to) 20 SPARC processors running in a tightly-coupled (shared memory) configuration. SunSoft's Solaris operating system provided basic concurrency (thread management) services, and SunSoft's NEO ORB was used to describe and implement the software architecture. One essential design problem, then, was to design a software architecture that would maximize concurrency within programs (object implementations) and across processors.

The product line reuse requirements were no less consequential on the final design. Note that in the previous case studies a premium was placed on architectural concepts that emphasized coordination, but were silent on issues of functionality (i.e., what functions were being computed by components). In both previous case studies, functionality remained implicit in the design, and was expressed at the protocol level. For PCo, an important requirement was plug-replaceable functionality, to enable new versions of printers, with different combinations of functionality, to be quickly assembled from standard parts—that is, a product line of printers.

Figure 3 depicts the innovative architectural concepts used to satisfy these two competing design objectives (performance and unit replaceability). Once again we see the use of DOT to describe and implement a design pattern based upon coordination, rather than functionality. The top-level components in the design pattern (referred to as "Coordinator objects") are abstractions that are used to channel data to available objects that are executing on particular processors; they represent a design metaphor that can be expressed as "pushing data through logic."

As with the ConsistencyManager pattern described in the previous case study, the "Coordinator pattern" objects provide operations to route data to an object (`data_input()`), a probe for determining the execution state of the object (`con_status()` for consumer status, because the Coordinator object is playing the role of a data consumer in this case), and an operation for passing specific computation instructions to the object (`instruction()`). Parallel operations are defined to allow the same object to act as a supplier of data to other objects (these are depicted on the right side of the structural object). The Coordinator pattern is fo-

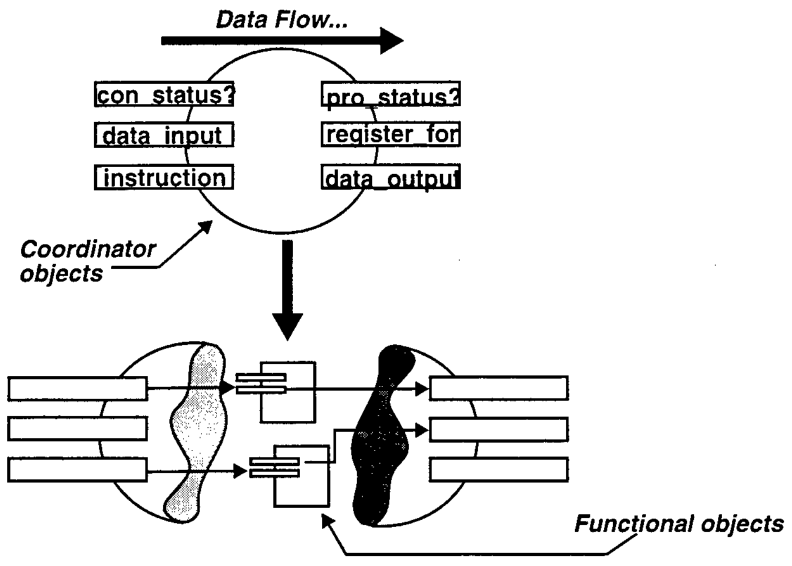

**Figure 3: Hybrid Patterns for Distributed, Concurrent, Embedded Processing**

cused on design issues relating to performance and concurrency, and makes the design both analyzable and run-time tunable.

Reuse of functionality is also addressed by the Coordinator pattern, though at a different level of abstraction. Although functionality is left implicit in the top-level design elements (the pattern), the PCo architecture exploits another feature of DOT that is derived from its OT heritage: the ability to use inheritance to define highly-factored class lattices. Thus, in addition to defining a top-level design pattern that is focused on performance considerations, the PCo architecture also uses CORBA interface definition language (IDL) to model the functionality of the printer domain. This model expresses a deep, rich class hierarchy. Plug-replaceable components implement these classes, and are "plugged into" the overall printer architecture as defined by the Coordinator pattern. These function-oriented interface definitions round out the overall architecture, which now addresses both product-line reuse and performance aspects of the design.

## 3.2 Java-Based Designs

Java is a much more recent technology than CORBA, but it is clear that it has already had, and will continue to have, an enormous impact in the DOT marketplace. The reasons for this are simple: the affinity of Java with the World Wide Web, the fact that the Web and browsers are now part of the standard computing vocabulary, and the integration of Java with Web browsers that run on all popular computing platforms. As already noted, Java as a language is cleaner and leaner than C++. Moreover, the integration of Java with the Web adds a dimension to OT not available in languages such as C++, namely mobile objects. That is, with Java it is possible to write object implementations that are delivered to the end user's execution environment, along with any supporting class libraries that are needed to implement the object.These mobile objects are the essence of the powerful Java applets.

One way of differentiating Java from CORBA is that Java provides mobile objects within a homogeneous computing environment (i.e., all computation occurs within a Java Virtual Machine), while CORBA provides remote objects, assuming a heterogeneous computing environment. With Java, there is the advantage of uniformity and portability across many physical machine architectures. The data types on the Virtual Machine are coerced to be the same on all platforms. While this certainly provides a cleaner global implementation model, there are also some costs. The most obvious is the architectural mismatch between the Virtual Machine and the actual machine that must be resolved by software. This mismatch has until now been resolved by a Java interpreter on each physical architecture. As noted earlier, in the near future the mismatch will be resolved by JIT compilers also resident on each physical architecture. It is important to remember that the Java interpreter and these future compilers are part of the Java Virtual Machine included in the Web browser.

Recent developments in both CORBA and Java, however, have begun to blur this distinction. The newest version of Java, Version 1.1, supports a feature called Remote Method Invocation (RMI). RMI provides a naming service for Java programs to look up object references for objects that are executing on remote Java Virtual Machines, and then to invoke methods on these remote objects. In effect, RMI provides ORB functionality that is fully integrated with the Java language and run-time environment. Unlike CORBA, however, the RMI ORB is fully integrated with the Java language and run-time environment. Thus, while CORBA interfaces are described using an architecture-neutral IDL, interfaces of remote Java objects are described using the Java interface construct.

Just as Java has been adding CORBA-like capabilities, CORBA has been evolving to accommodate the impact of Java. Most notably, ORB vendors are now supporting the development of Java clients for CORBA objects. Java applets that access remote CORBA objects (such applets are called "orblets") have the effect of pulling CORBA into the Web, and thus make it feasible to use Web browsers as delivery vehicles for CORBA-based object services.

We identify four stages in the evolution of the impact of Java technology on system design:

- applet objects on Web pages
- applets communicating directly with hosts and databases
- applets integrated with CORBA technology
- Java end-to-end technology (RMI)

Hamilton describes a number of issues relative to these stages [Hamilton 96]. The overall impact is described by Hamilton and others as an evolution from desktop computing to "net-centric" computing.

## Applet Objects on Web Pages

The first stage of the Java phenomenon probably generated the most interest in Java because it allowed Web page designers to add interaction and animation to their pages. An applet is invoked by an "applet tag" within an HTML page in a browser. It is therefore necessary for the browser to not only understand the applet tag, but to also provide the Java interpreter to execute the applet once it arrives at the client location from the server. Among the first applets were simple animations that were loaded by the browser to a client machine and then executed on the local client machine. Once loaded in the client machine, the Java Virtual Machine executes the Java program that drives the animation. One popular application is a "ticker" program that scrolls text across the screen in the manner of a stock ticker. The message to be scrolled can be provided as a parameter to the applet on the server machine or stored in a file on the server machine. A user interface can be created to allow control of the color, the font, or the speed of the message.

The significant benefit of using Java for user interfaces is that the user interface does not have to be reimplemented for multiple platforms. There is a dynamic presentation capability. Because browsers such as Netscape Navigator and Microsoft's Internet Explorer provide a Java Virtual Machine, a Java applet will run on all machines for which the browser is implemented. Thus, for the price of one implementation on one server, the application developer has provided the user interface functionality on many types of client machines. Moving user interfaces from the client to the server, labeled the "thin client" or "write once/run everywhere," can significantly reduce both maintenance and administration costs. The only downside is the slightly different renderings by the browsers to accommodate the physical characteristics of the display hardware.

## Applets Communicating Directly with Hosts and Databases

In order for a distributed application to accomplish anything of significance, it must connect back to the host system for access to databases. In the simplest scenario, the Java program connects back to the server machine using socket technology that is provided by the networking application programming interfaces (APIs) of Java. The Java networking API provides mechanisms to encode and decode universal resource locators (URLs), connect to URLs, and handle streams. It is available now in version 1.0.2 of the Java Development Kit (JDK). The Database Connectivity Specification (JDBC) API provides relational database operations between the host and the client and will be available with JDK 1.1. These interfaces facilitate the connection between the applet on a client to "live" data on the host. Examples are stock or news tickers. This technology can be considered distributed technology, but it is not true object technology since there is no concept of an object request broker.

One popular application at this level is a "shared whiteboard" wherein two or more people can make annotations on a shared picture in the manner of a sportscaster drawing on a TV picture with a telestrator. One specific application for the shared whiteboard is for meteorologists to collaborate over shared weather maps and satellite imagery as shown in Figure 4 [Veltre 95].

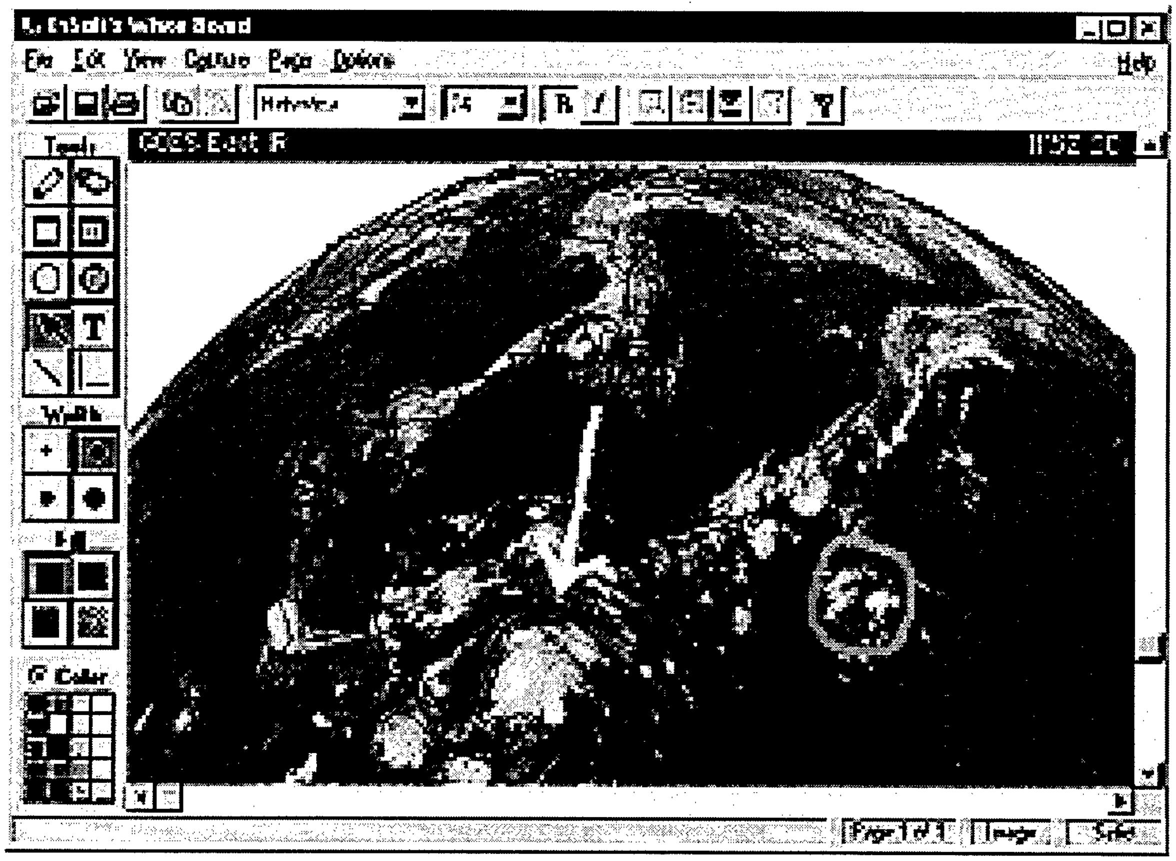

Figure 4: Whiteboard Example

Due to security constraints, the Java programs are not allowed to make arbitrary connections. They can only make connections back to the server from which the Java applet was retrieved. These and other restrictions help prevent untrusted applets from causing unbridled mischief on the client machine. Similarly, Java applets are not permitted to access system software on the client machine.

## Applets Integrated with CORBA Technology

SunSoft has introduced a product called the Java Object Environment (Joe)[1] that integrates Java with its CORBA-compliant product NEO. Similar products from other vendors are IONA's Orbix for Java [IONA 96] and Post Modern Computing's Black Widow. Typically, CORBA systems contain many distributed objects providing services and communicating through object request brokers using standardized object services. Joe and these other integrated tool kits provide an applet interface to a CORBA object environment. Joe is the NEO client-side run-

1. The SunSoft Joe product will actually work with any CORBA objects that support the interoperability protocols defined in the CORBA version 2.0 specification.

time library re-implemented in Java. It also provides the IDL-to-Java mapping required by CORBA implementations. Using CORBA interfaces, Joe will connect a Java client to a server, a RDBMS, or to existing applications running remote NEO objects. Java is an ideal CORBA client.

The following example represents the steps in accessing a mutual fund database [Deshpande 96]. First you would look up "mutual funds" using a search engine such as Lycos or Yahoo. Next, the browser contacts the URL returned by the search engine and returns an HTML page containing an applet tag. The applet is a Joe applet so the Java classes implementing the Java ORB run-time system are downloaded into the Web browser if necessary. When the applet runs, it contacts one or more NEO servers running on the same mutual fund server. On these servers a NEO-based mutual fund application is already running. It may span multiple machines and utilize many databases. The Joe applet presents you with a rich GUI and communicates and initiates transactions with the NEO servers on the back end as necessary. Figure 5 illustrates the steps and the dataflows of the Java/Joe solution after the URL for the mutual fund has been retrieved from the search engine.

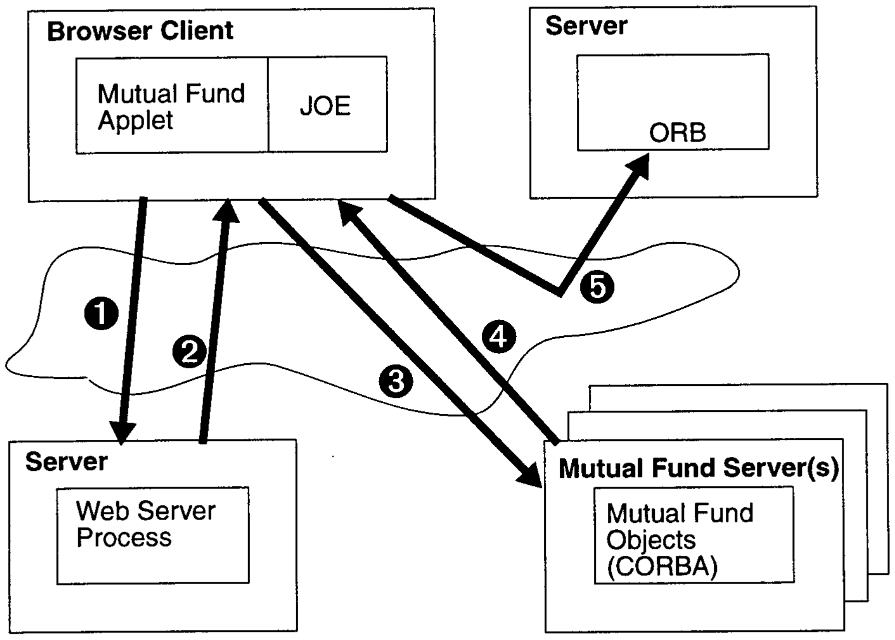

**Figure 5: Mutual Fund Example (Java/CORBA)**

As vendors provide Java interfaces to their own implementations of CORBA technology, there will be more power to be integrated into browsers. For example, Netscape 4.0 will incorporate Visigenic's ORB, which represents another distinct approach for combining CORBA, Java, and the Web. Netscape 4.0 is scheduled for release in 1997.

## Java End-to-End Technology

JavaSoft, a separate division of Sun, offers a competing solution for DOT called Remote Method Invocation (RMI) that is strictly Java-based. As noted, RMI provides a uniform homogeneous solution based on the Java language and the Java Virtual Machine. RMI gives Java a power boost that allows applets to work with and invoke one another seamlessly. RMI is included in Version 1.1 of the Java Development Kit.

RMI takes the place of a CORBA-compliant ORB. Instead of writing the interface specifications in IDL and the server in C++, the RMI solution is to write these in Java. The CORBA servers are replaced by Java servers. It turns out that Java is just as good for writing server side programs as client side applets. If there is little legacy code and the application developer is willing to implement business objects using Java, RMI is certainly a simpler way to build three-tier applications with shared services. "Wrapping" of legacy code with native Java methods is possible, but can be more difficult than wrapping with IDL. Figure 6 shows the same mutual fund example using a pure Java solution.

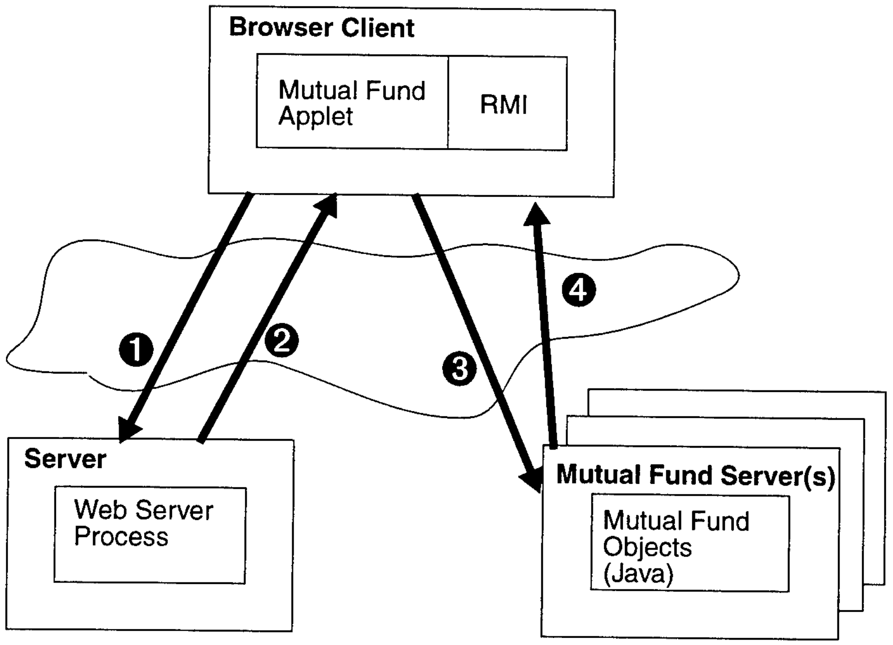

**Figure 6: Mutual Fund Example (Java)**

It is too early to assess completely the design implications of mobile objects, the merging of CORBA and Java, or the association between objects and the World Wide Web. Most of this technology is "work in progress" and there are rough edges that need to be smoothed. Certainly new and novel design patterns are possible (e.g., agent based architectures); new classes of systems with new and unique design problems may emerge (e.g., intranet systems, wide-area information management systems). Clearly, Sun will drive where the technology goes. There is considerable tension between the goals of SunSoft (with NEO/Joe) and Java-

Soft (with RMI). If Sun signals the choice of one of these strategies over the other, it would be a significant event and could change the course of the technology.

The most important point about the impact of Java on the design of software systems bears repeating. Java adds application portability to the client that CORBA gives to the server. Java adds the transportable user interface to a variety of distributed object technologies. Market forces will determine whether this evolves as pure Java or mixed Java and CORBA. What is clear is that Java is maturing as a programming language and a development environment. Initially perceived as a tool for making Web pages, the true potential of Java for serious software development in our networked world has now been recognized. Java is here to stay.

# 4 Impact of DOT on Software Development/Engineering Processes

Raising the level of abstraction is a persistent goal of software engineering that enables us to use more powerful methods and tools to be more productive in our work. However, the move to more powerful tools has associated costs and risks. Each new level of abstraction creates a new middle layer between the high-level abstraction and the low-level implementation. For example, compilers bridge the gap between high-level languages and machine languages. While high-level languages improve the productivity of application developers, they also spawn the need for compiler writers, run-time system developers, and operating system interfaces. Similarly, the impact of DOT makes work easier at the application developer level, but complicates work at the middleware level.

The costs of education and training at both the application developer level and the middleware level are significant. Another cost, at least initially, is the inefficiency and lack of robustness and consistency of the tools. The primary risk is the chance of the misuse of the tools. The use of any tools without the proper educational background undermines any benefit the tools can provide. When powerful tools are misunderstood and misused, the consequences of their misuse can be catastrophic.

Because there are many technologies that constitute DOT, there is a great deal to learn. Moreover, there are a variety of methods and tools that must be introduced to use DOT effectively. The purpose of this section is to elaborate on the methods and tools associated with DOT. This is a complex technology transition problem and we can only provide a general discussion of the more important issues.

For the purpose of our analysis, we will return to the DOT characterization in the introduction of terms of the three technologies: OT, DT, and WT. We will also subdivide the problem in terms of the general knowledge needed (education) versus the specific knowledge needed (training) for each of the technologies.

## 4.1 Object Technology

Object technology is the overarching technology. It is pointless to consider moving to DOT without a firm understanding of OT. This initial step should not be underestimated since it represents a major paradigm shift. Many organizations have already made that step, or partially made the step, but many have not. Adoption of OT cannot be made in a single step—the transition needs to progress through levels of absorption before assimilation into a software development organization actually occurs. This transition period can take considerable time. Education and training are essential, and pilot projects are recommended.

It has been acknowledged for some time that OOP is not a sufficient technology for application development [Northrop 97]. In order to reap the true benefits of OT, object-oriented analysis (OOA) and object-oriented design (OOD) should precede object-oriented programming efforts. Over the years numerous methods have been described for analysis and design of OO systems. The most significant approaches have come from Booch, Rumbaugh (OMT), Jacobson (Use Case Approach/Objectory), Schlaer-Mellor, and Wirfs-Brock (Responsibility Driven Design). Recently, Booch, Rumbaugh, and Jacobson have joined forces and merged their individual approaches into the Unified Approach using the Unified Modeling Language (UML), which consists of a common set of models and notation for object-oriented analysis and design. Though not yet complete, this merging of technology signals a maturation of the field and notable progress toward a common analysis/design approach. Issues relating to the use of each of these methods include defining the appropriate set of objects, determining the appropriate classification and relationship of objects, defining the appropriate set of operations (object behavior), and determining how to best represent them. Among the software engineering issues to be addressed are abstraction, encapsulation, modularity, hierarchy, typing, concurrency, and persistence.

Most recently, the Object Management Group (OMG), has issued an RFP for an OOA and OOD metamodel that describes four views: static, dynamic, usage, and architectural. The goals of the OMG in this endeavor are vendor-neutral common semantics, and abstract syntax for OOA and OOD methodologies. Selection is planned for June 1997 with adoption the following month.

Since OT is quite mature, there is a core of people who have had the necessary education and training in this field, and a good number who have significant OT development experience. Furthermore, there are good books, courses, tutorials, and workshops that are available to people entering the field. Anyone who wants to learn OT can have access to the materials. However, the difficulty of learning OT should not be underestimated. Nothing can substitute for the experience of building several systems using the OT models with the specific tools and methods. Also it should be noted that the testing and management of object-oriented efforts require new approaches and these areas are not documented as well as the analysis and design approaches.

## 4.2 Distributed Technology and CORBA

Distributed technology is concerned with the middle layer between the application and the distributed and networked application hardware. As pointed out cogently by Waldo, it is quite tantalizing, but wrong, to believe that objects can interact in a distributed system in the same way that they interact in a single address space (local computing environment) [Waldo 94]. Waldo and colleagues identify four major differences between local and distributed computing. They are latency, memory access, partial failure, and concurrency.

The most obvious difference is latency. The difference between the time required to invoke a method on a local machine versus the time to invoke a method on a remote (or possibly remote) machine can be up to four or five orders of magnitude. The design of the application must take these differences into account. Memory access refers to the use of pointers, which may be valid in a local address space, but not in a remote address space. Either memory accesses must be controlled by the underlying system or the programmer must be aware of the different types of access. The problem with partial failure is that, unlike local systems, when pieces of the system fail (network or computing resources) it is undetectable by some centralized resource allocator. Partial failure requires that programs deal with indeterminacy. Similarly, concurrency in a distributed system is different from concurrency in a multi-threaded local system. This is because there is no single operating system that can aid in synchronization and in the recovery from failure.

For DT the education problem is to learn about and understand all the issues surrounding distribution. It is necessary to learn about one or more of the different infrastructures for distributed object computing. These include CORBA by OMG, Object Linking and Embedding (OLE) and Component Object Model (COM) by Microsoft, and OpenDoc from CI Labs.[1] Once some framework for DOT is understood, it is necessary to learn some specific implementation of that framework to employ it to build systems. In the case of CORBA there are at least a dozen vendors selling products claimed to be CORBA compliant. Learning how to use these products is non-trivial. For example, one has to learn a great deal about UNIX internals while installing and using Orbix.

Moreover, this technology is rapidly evolving. New products are coming out faster than existing products are dying or being consolidated. There are relatively few good books or courses on CORBA, but the numbers are increasing. While developers are now building systems with the middleware products, there are still frequent complaints about robustness and performance. The products need to mature, but practitioners must also learn how to make most effective use of the technology. Just as Ada programmers of the middle 1980s frequently used tasking inappropriately, the CORBA programmers of the present may be causing performance problems by making inappropriate use of CORBA features.

## 4.3 Web Technology and Java

As already noted, Web technology and Java are the most recent additions to DOT. Both the tool and method development have moved considerably more rapidly than previous technology innovations because Web technology has helped considerably in transitioning itself. The Web has become the ubiquitous teaching and distribution mechanism. Java has been rapidly embraced. As noted in Tribble, a study by Forrester Research of Fortune 1000 companies found that 62% are currently doing some Java development, and that 42% expect Java to play a strategic role within a year [Tribble 96].

The core concepts for Web development have been HTML and the notion of a URL. HTML, the language for arranging text and images on a Web "page," has evolved over its short lifetime to include higher level structures such as tables, forms (for entering data), and frames (inde-

1. There might be some debate as to whether Microsoft's OLE/COM and CI Labs' OpenDoc are object technologies. They are both considered object-based rather than object-oriented since neither includes inheritance.

pendently scrollable subwindows). The URL provides the mechanism for referencing other pages across the global name space. The tools for developing Web pages have evolved from simple editors to smart editors for HTML that understand the syntax of the markup language and that facilitate FTP access, to full blown Web site development tools that claim no knowledge of HTML is required. Other tools are used to manipulate graphics and conversion tools are used to translate documents from word processor formats to HTML.

As noted, the Java programming language is reasonably straightforward and quite similar to other well-known languages. However, as with any object-oriented language, there is more to learn than syntax. The associated class library can only be learned with experience and use. There is a hierarchy of classes with APIs for networking; for example, the abstract window toolkit, IO, math, debugging, and operating system services.

Separate from the Java language, there has been a rapid increase in the number of development tools. In addition to the Java Development Kit from Sun there are platform-specific tools like Cafe from Symantec and Microsoft's J++. Java Workshop 1.0 from JavaSoft boasts a 100% applet-based development system with an integrated set of Web-centric development tools and a graphical user interface (GUI) builder [21]. It is currently available for Solaris and Microsoft Windows, but is targeted for other platforms, including the Macintosh, in early 1997.

Unlike CORBA, and despite its newness (and probably due to its popularity, accessibility, and the unprecedented public relations campaign for Java), Java training materials are widely available. There are already dozens of books on Java and an increasing number of courses, workshops, and tutorials. Java has already been selected by a number of universities as an introductory programming language. Furthermore, much can be retrieved directly from Web sites, including complete examples of applets and programs, as well as tutorials. Since there are relatively few truly new concepts introduced by Java, the educational burden is light. But the training burden is heavy. There is a steep learning curve for the tools, but the means to climb it are available. The main problem is to separate the wheat from the chaff.

The impact of Web technology for application development has been the addition of a new synergism. With proper use of search engines, it is possible to find information—and sources of information—that were previously hard to find. Developers frequently go to the Web to find objects that they need rather than building the objects themselves. The Web is a powerful client. In addition, Java is providing a transportable interface to legacy systems that facilitates the development of applications for which the user interface was previously a bottleneck. Furthermore, Web technology is allowing people to work from their homes and allowing developers in third world countries to contribute to the expansion of knowledge and the creation of systems. Net-centric computing presents opportunities for profound changes in the process of developing application systems by globalizing the access to information.

# 5 Conjectures About Trends in DOT

The speed at which DOT is being developed and adopted is perhaps unprecedented. This is especially true of Java. Therefore it is difficult to predict future trends in DOT with any scientific basis. Nevertheless, some trends are visible, and these are discussed below.

## 5.1 Accelerated Adoption of Design Patterns for Distributed Systems

As noted by Clements and Northrop, the topic of software architecture spans a wide variety of concepts [Clements 96b]. While this diversity of ideas and approaches can be considered a desirable quality by researchers, from a practitioner and industrial perspective this quality represents not diversity but cacophony. Why this diversity? There are many reasons, but among them is the lack of an agreed upon foundation for expressing the fundamental elements of design. For example, should the fundamental design elements be informal boxes and arrows, axiomatic statements, partially-ordered event sets, or domain-specific languages? No one can categorically say which is best, and there are many other notions of what would constitute a reasonable foundation.

Herein lies the major contribution of OT to the topic of software architecture. OT provides a set of fundamental elements (a *lingua franca*) with which to express designs. These elements—classes, objects, inheritance, encapsulation, polymorphism, and others—may not be ideal, but they are reasonable tools for *expressing and reusing abstractions*, and they are increasingly well understood and used in industry. Moreover, the computational model associated with these design elements is realized by readily available, mature OT (e.g., compilers, and databases). We emphasize the phrase *expressing and reusing abstractions* because at a basic level this is what software architecture is all about: understanding what makes a particular design successful and reusing the *essence* of this design in other systems. We express this essence as a *design abstraction*, and provide guidelines on when, and how, to reuse this abstraction to solve *specific, well-bounded problems.*

However, a *lingua franca* based on OT is not enough: Additional structuring is necessary. The notion of the *design pattern* has emerged from the OT community as one means of expressing design abstractions that solve specific, well-bounded problems.[1] We have already cited the important and influential work of Gamma and colleagues [Gamma 95], but we should also note that a large and active community has emerged that is focused on identifying and documenting design patterns. While to some extent the search for design patterns characterizes all *research* in software architecture, what makes OT-based design patterns significant is that their

1. Application frameworks are another means of expressing design abstractions, although in the case of frameworks these abstractions are far more comprehensive and focused on a specific application domain. While frameworks are economically feasible in some settings, they are expensive to develop and maintain. In contrast, design patterns are more abstract and more readily used across application domains.

expression in the OT *lingua franca* makes such patterns more readily implementable using widely available, mature software development technologies.

Figure 7 illustrates how design patterns can be defined using OT conventions by recasting the somewhat informal depiction of the ConsistencyManager pattern depicted in Figure 2 in the more concrete notation used by Gamma and colleagues. Although the pattern illustrated in Figure 7 is not complete, it illustrates some important points. First, it delineates the abstract

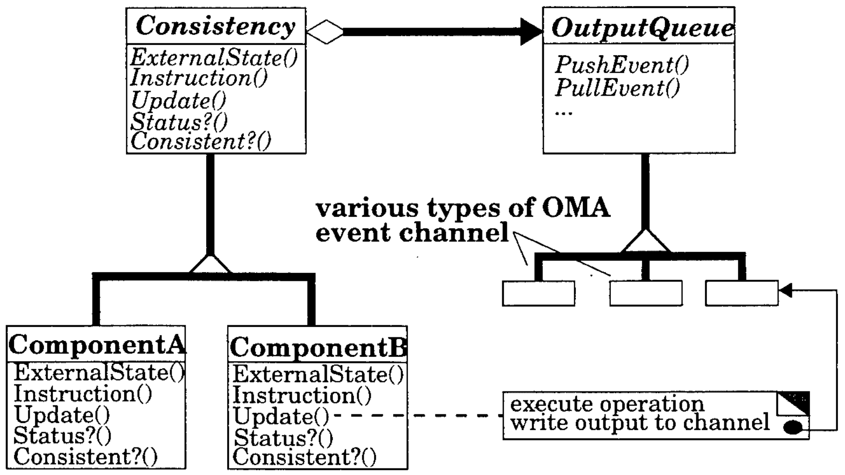

**Figure 7: ConsistencyManager Pattern Defined in OT *Lingua Franca***

part of the pattern from its concrete implementation (the *italicized* text indicates abstract concepts). Second, the use of OT abstractions to define these concepts means there are well-defined semantics for composing these abstractions, e.g., by multiple inheritance "mix-ins," by direct subclassing, or by composing patterns. Third, the pattern description is sufficiently concrete to warrant the inclusion of pseudo-code, thus illustrating the narrowing gap between abstract designs and concrete implementations. Last, patterns can be imported and reused from established DOTs such as OMA; the *OutputQueue* corresponds closely to the *EventChannel* interfaces defined by the OMA.

Development and adoption of design patterns is an ongoing process. From the original work on patterns that focused on lower-level programming concerns, design patterns that directly address issues of scale, for example patterns for distributed systems, are emerging [Schmidt 96]. DOT will both accelerate and enrich this process. Acceleration will occur because, as noted earlier, DOT bridges the gap between architectural concept and prototypical implementation. We can therefore expect the more frequent and more widespread use of DOT to develop distributed systems (large and small). Since design patterns emerge foremost from practice and not from theory, widespread use of DOT will encourage the development and proliferation of design patterns for distributed systems. Enrichment will occur because DOT extends the set of design elements provided by OT with important distribution concepts (e.g., object location, activation, and synchronization).

## 5.2 Penetration of DOT Into Demanding Application Areas

A corollary to the conjecture that there will be more frequent and widespread use of DOT is that at least some of these applications will be in settings that exhibit demanding quality attributes, such as performance, security, and dependability. Besides the obvious implications on the development of design patterns suitable for these kinds of demanding requirements (Simplex, for example, can be thought of as a nascent design pattern for dependable online upgrade of real-time control systems [Sha 95]), these demanding applications will have a reciprocal impact on the specification and implementation of DOT.

To some extent, DOT is already evolving to address some of these issues of scale. For example, the Java Virtual Machine and Java-enabled Web browsers address security issues.[1] The OMG has adopted a specification for security services as recently as March 1996. Performance is also a concern for DOT. On the Java side, substantial work on the JIT compilers is shrinking the performance gap between compiled C++ and Java. The OMG has also established a technical committee for analyzing approaches to extending CORBA and the OMA to support hard real-time applications. These and other enhancements of DOT are being driven by demand, and will ultimately have the effect of generating new demand as DOT passes from innovative to established technology and is employed in demanding settings.

There is already sufficient evidence of this demand. OMG participation in the real-time technical committee (TC) includes representatives from the telecommunications industry and financial management industry; the most recent TC meeting included representatives from 44 OMG member companies (almost 10% of the entire OMG membership).

The SEI has been directly experiencing DOT penetration through customer demands for support. The SEI led a technically-oriented risk assessment of a project employing CORBA technology in embedded printer control systems as described in the third case study in Section 3.1. The SEI was involved in a software architecture evaluation for an architecture using a CORBA-compliant infrastructure. The SEI is also currently funded by the National Institute for Standards and Technology (NIST) Manufacturing Engineering Laboratory (MEL) to investigate the use of CORBA for manufacturing shop-floor automation; there are also ties between this work and semiconductor/wafer manufacturing. The SEI is funded by DMSO to support their use of CORBA as described in the first case study in Section 3.1. Recently, the SEI was requested by the US Navy to submit a joint proposal with them to the Open Systems Joint Task Force (OSJTF) for investigating the suitability of CORBA for use in embedded Navy weapons systems. The FAA is also funding SEI work in investigating CORBA for the Display System Replacement (DSR) upgrade. MITRE has experienced similar demand, and has been involved in the use of CORBA for distributed real-time simulation (also a case study), and for embedded avionics software (AWACS). In response to this widespread and growing demand, the SEI and

1. Note that the adequacy of these solutions is the subject of controversy, and they will undergo continued enhancement.

MITRE are establishing collaborative working relationships to specifically address the topic of real-time CORBA.

The bottom line is that practical demand is driving enhancements to DOT to enable its use in large-scale, mission-critical systems.

## 5.3 Increasing Visibility of "Intranet" Systems

There is a clear trend towards increased use of the Internet and Web browser technology to deliver services to end users, and not just elaborate Web pages. While Internet-based applications for business uses already exist, a more common scenario is the use of *intranet* systems. Intranet systems use the Internet protocols and Web browsers, but limit access to the services to a well-bounded administrative domain, e.g., all of the nodes may be behind a corporate firewall.

There may be compelling economic justifications for moving from current PC-based business applications to a Web-based delivery approach; one approach for calculating return on investment (ROI) is to estimate the cost of maintaining individual PCs versus the cost of maintaining "thin clients," i.e., stripped-down, low-cost remote workstations [Tribble 96]. If this analysis is valid, a general migration to intranet-based corporate information systems can be expected in the coming years. The fact that security considerations in an intranet are more tractable than in the unbounded Internet system also suggests an "intranet first" trend.

As this trend becomes more pronounced, and more visible, demand for the development of intranet solutions for mission-critical information systems will emerge; this further substantiates the earlier prediction that DOT will begin to penetrate into more demanding application areas. For example, the use of DOT in combination with Web browsing technology could radically improve the way data management systems such as the DoD JEDMICS (Joint Engineering Data Management Information and Control System) deliver services to end users, and radically reduce the cost of maintaining and upgrading JEDMICS system functionality.

The arguments in support of these conclusions—support for platform heterogeneity (Internet and Web protocols), ready-made and standard service delivery mechanisms (Web browsers), central management but remote distribution of functionality (mobile code and applets), loose coupling of software components (distributed and remote objects)—are very compelling when viewed collectively. Moreover, similar arguments and conclusions can be reached about most (or all) information management systems that support distributed users, and may themselves be distributed.

## 5.4 Migration of Legacy Systems Using DOT

The vast number of existing legacy systems represents both substantial investment and core business functionality. Organizations can neither afford to redevelop these systems nor afford to keep them isolated from current technology improvements. DOT is now sufficiently mature to support legacy system migration [Klinker 96]. "Object wrapping" can provide an object interface to the legacy systems. Clients can view the legacy system through a simple CORBA API presented by the wrapper. The wrapping layer can communicate with the legacy system via sockets, RPC, or an API. Once wrapped, the legacy system, or its subsystems, becomes highly reusable software components. The Web, as already noted, can be considered simply another client that needs to communicate with the wrappers. Consequently, the legacy applications will be accessible from the Web. "Together, Java and CORBA-based object wrappers can be used as the basis for a sophisticated and dynamic set of applications to other technologies" [Klinker 96].

## 5.5 Use of DOT and the Web to Search for Breakthrough Technologies

The trends just discussed are already visible in industry and within the US DoD—they reflect a widely-occurring phenomena. On a more speculative note, we observe that significant interest in system survivability issues has been triggered by the explosive growth of, and reliance on, the Internet. A recent broad agency announcement (BAA) issued by the Defense Advanced Research Program Agency (DARPA) on the topic of survivable systems explicitly calls out CORBA and Java as key enabling technologies. The concept of "programming the network" represents a potentially radical shift in perspective that may open up opportunities for systems breakthroughs. For example, onboard support for Java in digital switching hardware might allow entire switching systems to be dynamically reconfigured in response to a threat; autonomous, trusted computing agents might search the Internet for suspicious activities; other intelligent agents might be capable of dynamically adapting component interfaces to repair or prevent damage from an external threat.

# 6 Summary

In the earlier sections of this paper we argued that the technical characteristics of DOT have significant importance to software engineering. Specifically, we described architectural implications (Section 3), process implications (Section 4), and the implications of already visible trends (Section 5).

While DOT does not address all of the challenging problems of designing and fielding complex systems, it addresses many important issues; moreover, as the trends discussed earlier in this paper indicate, the issues that are addressed by DOT are becoming more important and at the same time DOT is becoming more widespread. As DOT is assimilated it will undoubtedly form the basis for the next level of abstraction aimed at surmounting the increasing complexity in software development. DOT is a landmark in computing history.

Furthermore, the implications of DOT are not purely technological. DOT and the Web are definitely changing the way the DoD and US industry do business. Failure to keep abreast of (and exploit) these trends represents as much of a *risk* to many organizations as trying to adopt these rapidly changing technologies. The ratcheting-up of the level of abstraction in the design and development of software likewise will have profound implications on software development processes and, ultimately, on US global competitiveness; the days where systems are developed a line of code at a time are quickly fading, and software process and management practices that fail to recognize this are fast becoming obsolete (if they are not already).

The US repeatedly reasserts its dominance in software technology by “changing the rules” by which software is developed and used; DOT is another such change. We see a profound impact and fundamental change in both technical thinking and practice as a result of the related technologies that we have grouped together as DOT.

## References

| | |
|---|---|
| [Abowd 93] | Abowd, Gregory; Bass, Len; Howard, Larry; & Northrop, Linda. *Structural Modeling: An Application Framework and Development Process for Flight Simulators* (CMU/SEI-93-TR-14, ADA271348). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993. |
| [Betz 94] | Betz, Mark. "InterOperable Objects." *Dr. Dobb's Journal 19*, 11 (October 1994): 18-39. |
| [Britton 81] | Britton, K.; Parker, A.; & Parnas, D. "A Procedure for Designing Abstract Interfaces for Device Interface Modules," 195-204. *Proceedings of the International Conference on Software Engineering*. San Diego, CA, March 9-12, 1981. Los Alamitos, CA: IEEE Computer Society Press, 1981. |
| [Calvin 96] | Calvin, J. & Weatherly, R. "An Introduction to the High Level Architecture (HLA) Runtime Infrastructure (RTI)," 705-715. *Proceedings of the 14th Workshop on Standards for the Interoperability of Distributed Simulations*, Volume 2. Orlando, FL, March 11-15, 1996. Orlando, FL: University of Central Florida, Division of Sponsored Research, 1996. |
| [Clements 96a] | Clements, P.; Wallace E.; & Wallnau, K. "Discovering a System Modernization Decision Framework: A Case Study in Migrating to Distributed Object Technology," 185-189. *Proceedings of the International Conference on Software Maintenance (ICSW)*. Monterey, CA, November 4-8, 1996. Los Alamitos, CA: IEEE Computer Society Press, 1996. |
| [Clements 96b] | Clements, P. & Northrop, L. "Software Architecture: An Executive Overview." *Component-Based Software Engineering*. Los Alamitos, CA: IEEE Computer Society Press, 1996. |
| [CORBAnet 97] | *CORBAnet—The ORB Interoperability Showcase* [online]. Available WWW <URL: http://www.corba.net/> (1997). |
| [Deshpande 96] | Deshpande, Salil R. *CORBA/NEO/Joe Tutorial*. Palo Alto, CA: CustomWare Training Publications, 1996. Also see http://www.customware.com/training. |
| [Ewald 96a] | Ewald, Alan & Roy, Mark. "Bringing Objects to You." *Object Magazine 6*, 4 (July 1996): 69-70. |
| [Ewald 96b] | Ewald, Alan & Roy, Mark. "Choosing Between CORBA and DCOM." *Object Magazine 6*, 8 (October 1996): 24-30. |

| | |
|---|---|
| [Gamma 95] | Gamma, E.; Helm, R.; Johnson, R.; & Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley, 1995. |
| [Garlan 93] | Garlan, D. & Shaw, M. "An Introduction to Software Architecture." *Advances in Software Engineering and Knowledge Engineering, Volume 1.* River Edge, NJ: World Scientific Publishing Company, 1993. |
| [Goldberg 83] | Goldberg, A. & Robson, D. *Smalltalk-80: The Language and Its Implementation.* Reading, MA: Addison-Wesley, 1983. |
| [Gosling 96] | Gosling, James. "The Feel of Java" [videotape]. Invited talk at the 1996 Conference on Object Oriented Programming, Systems, Languages, and Applications. Stanford, CA: University Video Communications, 1996. |
| [Hamilton 96] | Hamilton, Marc A. "Java and the Shift to Net-Centric Computing." *Computer 29*, 8 (August 1996): 31-39. |
| [Iona 96] | Iona Technologies, Ltd. *Orbix Web White Paper* [online]. Available WWW <URL: http://www-usa.iona.com/> (1996). |
| [Jul 96] | Jul, Eric. "Introduction to Distributed Computing Using Objects," Tutorial 9. *Tutorial Notes: Conference on Object Oriented Programming, Systems, Languages, and Applications.* San Jose, CA, October 6-10, 1996. US: ACM SIGPLAN, 1996. |
| [Klinker 96] | Klinker, Paul; Marsh, John; Tisaranni, John; & Zahavi, Ron. "How to Avoid Getting Stuck On the Migration Highway." *Object Magazine 6*, 8 (October 1996): 46-51. |
| [Low 96] | Low, G. C.; Rasmussen, G.; & Henderson-Sellers, B. "Incorporation of Distributed Computing Concerns into Object-Oriented Methodologies." *Journal of Object-Oriented Programming 9*, 3 (June 1996): 12-20. |
| [Mowbray 96] | Mowbray, Tom. "Migrating Legacy Systems to Object Technology." *Object Magazine 6, 8* (October 1996): 31. |
| [Northrop 97] | Northrop, Linda M. "Object Oriented Development." *Software Engineering.* Los Alamitos, CA: IEEE Computer Society Press, 1997. |
| [O' Brien 96] | O' Brien, Timothy. "Sun's New Products Leverage Objects Onto the Net." *Object Magazine 6*, 4 (July 1996): 64-65. |
| [OMG 97] | *Welcome to OMG's Home Page* [online]. Available WWW <URL: http://www.omg.org/> (1997). |

|  |  |
|---|---|
| [Rymer 96] | Rymer, J. "The Muddle in the Middle." *Byte Magazine 21*, 4 (April 1996): 67-70. |
| [Schmidt 96a] | Schmidt, Douglas C. *Design Patterns for Concurrent, Parallel, and Distributed Systems* [online]. Available WWW <URL: http://siesta.cs.wustl.edu/~schmidt/patterns-cpd.html/> (1996). |
| [Schmidt 96b] | Schmidt, Douglas C. "Object Oriented Design Patterns for Concurrent, Parallel, and Distributed Systems." Tutorial 50. *Tutorial Notes: Conference on Object Oriented Programming, Systems, Languages, and Applications.* San Jose, CA, October 6-10, 1996. US: ACM SIGPLAN, 1996. |
| [Scotkin 96] | Scotkin, Joel. "How 1.1 Will Affect Business." *Java Report 1*, 6 (November/December 1996): 37-40. |
| [SEMATECH 94] | SEMATECH. *Computer Integrated Manufacturing (CIM) Application Framework Specification 1.1.* (TT# 93061697D-ENG). Austin, TX: SEMATECH, 1994. |
| [Sha 95] | Sha, L.; Rajkumar, R.; & Gagliardi, M. *A Software Architecture for Dependable and Evolvable Industrial Computing Systems* (CMU/SEI-95-TR-05). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1995. |
| [Stal 96] | Stal, Michael & Uwe Steinmueller. "OLE, CORBA—Infrastructures for Distributed Object Computing." Tutorial 35. *Tutorial Notes: Conference on Object Oriented Programming, Systems, Languages, and Applications.* San Jose, CA, October 6-10, 1996. US: ACM SIGPLAN, 1996. |
| [Sun 96] | Sun Microsystems. *Java Workshop Product Information* [online]. Available WWW <URL: http://www.sun.com/developer-products/java/> (1996). |
| [Tribble 96] | Tribble, Bud. *Java Computing in the Enterprise: What it Means for the General Manager and CIO* [online]. Available WWW <URL: http://www.sun.com/javacomputing/whpaper/> (1996). |
| [Veltre 95] | Veltre, R. & Weiderman, N. *The METOC Anchor Desk.* Pittsburgh, PA: Evolutionary Systems Laboratory, Carnegie Mellon University, 1995. |
| [Waldo 94] | Waldo, J.; Wyand, G.; Wollrath, A.; & Kendall, S. *A Note on Distributed Computing* [online]. Available WWW <URL: http://www.sunlabs.com/techrep/1994/abstract-29.html/> (1994). |
| [Wallnau 96] | Wallnau, K. & Wallace, E. "A Situated Evaluation of the Object Management Group's (OMG) Object Management Architecture." *Proceedings of* |

*OOPSLA '96*. San Jose, CA, October 6-10, 1996. New York, NY: ACM Press, 1996.

[Zimmerman 80] Zimmerman, H. "OSI Reference Model—The ISO Model of Architecture for Open Systems Interconnection." *IEEE Transactions on Communication 28*, 4 (April 1980): 425.

| REPORT DOCUMENTATION PAGE | *Form Approved* *OMB No. 0704-0188* |
|---|---|

| | | |
|---|---|---|
| 1. **AGENCY USE ONLY** (leave blank) | 2. **REPORT DATE** June 1997 | 3. **REPORT TYPE AND DATES COVERED** Final |
| 4. **TITLE AND SUBTITLE** Distributed Object Technology with CORBA and Java: Key Concepts and Implications | | 5. **FUNDING NUMBERS** C — F19628-95-C-0003 |
| 6. **AUTHOR(S)** Kurt Wallnau, Nelson Weiderman, and Linda Northrop | | |
| 7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213 | | 8. **PERFORMING ORGANIZATION REPORT NUMBER** CMU/SEI-97-TR-004 |
| 9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** HQ ESC/AXS 5 Eglin Street Hanscom AFB, MA 01731-2116 | | 10. **SPONSORING/MONITORING AGENCY REPORT NUMBER** ESC-TR-97-004 |
| 11. **SUPPLEMENTARY NOTES** | | |
| 12.a **DISTRIBUTION/AVAILABILITY STATEMENT** Unclassified/Unlimited, DTIC, NTIS | | 12.b **DISTRIBUTION CODE** |

13. **ABSTRACT** (maximum 200 words)

The purpose of this report is to analyze the potential impact of distributed object technology (DOT) on software engineering practice. The analysis culminates with the conclusion that the technology will have a significant influence on both the design and reengineering of information systems and the processes used to build them. We see a profound impact and fundamental change in both technical thinking and practice as a result of the related technologies we group together as DOT.

| | | | |
|---|---|---|---|
| 14. **SUBJECT TERMS** distributed object technology, CORBA, middleware, distributed systems, Web technology, Java, Java-based designs | | | 15. **NUMBER OF PAGES** 36 |
| | | | 16. **PRICE CODE** |
| 17. **SECURITY CLASSIFICATION OF REPORT** UNCLASSIFIED | 18. **SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED | 19. **SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED | 20. **LIMITATION OF ABSTRACT** UL |