Manager

Alan W. Brown

# Mastering the Middleware Muddle

For many software-development managers, staying abreast of the many new technologies that affect the systems we build is nearly impossible. One of the most complex and confusing technology areas is *middleware*—the cornerstone required to build enterprise-scale distributed systems. Within this esoteric technology domain, it's all too easy to hold up our hands and declare that the outlook is bleak for any kind of consensus on the direction distributed systems will take. Enterprise application integration issues, the nuances of competing middleware products from lots of vendors, and several overlapping standards activities all contribute to the confusion and uncertainty. To help us make sense of the middleware muddle, I'll take this space to discuss the key concepts and issues involved.

**It's all too easy to declare that the outlook is bleak for any kind of consensus on the direction distributed systems will take.**

To better understand recent progress in this arena, let's focus on current best practices for building enterprise-scale systems. That way, we can assess the current state of the technology and its impact on potential future strategies for distributed-systems developers. I'll begin by generalizing that any successful solution in enterprise-scale software engineering has four key elements: processes, infrastructure, tools, and adoption services. Let's briefly review current best practice in each of these four areas.

## Processes

Many changes are taking place in the way enterprise-scale systems are designed and how those designs are communicated. Within these activities, progress has taken place in three primary areas:

♦ *Modeling notations*. The Unified Modeling Language has become the notation of choice for describing the artifacts of many aspects of distributed-systems development. As the lingua franca in this area, UML is widely taught in colleges and universities, standardized by the Object Management Group, and supported by the vast majority of tool vendors. For improved communication and interoperation across projects, this widespread acceptance represents a great step forward.

♦ *Methods for designing distributed systems*. Because traditional software-development techniques in general are poorly suited to distributed-systems development, a new wave of development approaches is beginning to emerge. These must focus on legacy wrapping, provide visualization of a system's logical and physical architecture, use behavioral modeling via interfaces, package services as components to execute in a component-based infrastructure, and employ the UML notation. Few documented methods are yet available. A notable exception is the Catalysis approach (see the recent book by Desmond D'Sousa and Alan Cameron Wills listed in the boxed text on p. 20).

♦ *Life-cycle approaches*. Individual design methods and techniques must fit within a broader process framework covering a software-intensive system's complete life cycle. Fortunately, experiences in building and deploying large-scale distributed systems are now being documented and form the basis for guidance and heuristics captured in process frameworks such as the Rational Unified Process and Select Perspective. While far from perfect, both efforts direct considerable attention toward key life-cycle issues such as requirements traceability, synchronization of teams producing design

**EDITOR:** Roger Pressman • R.S. Pressman & Associates • pressman@rspa.com



0740-7459/99/$10.00 © 1999 IEEE

artifacts, and configuration management of resulting components and applications.

## INFRASTRUCTURE

The target for deploying distributed systems has evolved significantly over the past few years. In all but the most demanding real-time systems domains, today's enterprise-scale systems target commercial operating systems augmented with supporting technologies to obtain some measure of distribution transparency across multiple (heterogeneous) platforms. For example, various message-oriented middleware technologies are designed to solve problems requiring asynchronous communication between separate processes, and object transaction server technologies bring TPM-style services to object-oriented languages and databases. Within this context, recent developments in component models and component services are particularly noteworthy.

### Component models

To simplify distributed-system design and deployment, developers must restrict some of the architectural choices available in their designs and provide common conventions, structure, and documentation. These are needed to let parts of a distributed system be developed and used by separate groups of people, implemented in multiple languages, and reconfigured at runtime.

Conceptually, it is useful to view each of the separate parts being developed and assembled as offering services to other parts of the system through well-defined interfaces. In this case, the parts are known as *components*, and the conventions and restrictions on their use in building a distributed system are collectively called a *component model*. The component model defines how components expose their functionality as services through interfaces and describes a set of common conventions that all implementations of the component model must employ. This makes component-based development of systems easier because application developers can rely upon them being supported when assembling applications from components.

Many component models have been defined, but two are by far the most dominant. On the Microsoft platform, the component model is called the Component Object Model and its distributed version is DCOM. Building a distributed application on a Microsoft platform is much easier as a result of COM. In fact, most of Microsoft's own applications use COM as the basis for their distribution and integration. On non-Microsoft platforms, the OMG has defined the Common Object Request Broker Architecture as the basis for their component model. Multiple implementations of CORBA exist on different platforms (including Microsoft platforms). It is widely used in distributed systems, particularly for bridging heterogeneous platforms.

### Component services

The component models make it easier to connect a system's pieces. However, on their own, they provide little support for higher-level services that would commonly be needed in many applications for synchronization, security, persistent data management, and so on. In some situations, developers might wish to build these services specifically to meet their application's demands. However, for many applications, generalized services would be much more convenient. So, both common component models have associated services that provide much of the functionality required for building distributed systems.

On Microsoft platforms, the combination of COM and various distributed-system services is known as COM+. This includes support for transaction man-

### TERMS

| | |
|---|---|
| CMM | Capability Maturity Model |
| COM | Component Object Model |
| CORBA | Common Object Request Broker Architecture |
| EAI | Enterprise application integration |
| EJB | Enterprise JavaBeans |
| IDL | Interface definition language |
| MOM | Message-oriented middleware |
| MSMQ | Microsoft message queue server |
| MTS | Microsoft transaction server |
| OMA | Object Management Architecture |
| OMG | Object Management Group |
| OTS | Object transaction server |
| RUP | Rational Unified Process |
| TPM | Transaction processing manager |
| UML | Unified Modeling Language |

## Middleware Resources

### UML


- M. Fowler, *UML Distilled*, Addison Wesley Longman, Reading, Mass., 1997.
- G. Booch, J. Rumbaugh, and I. Jacobson, *The UML User Guide*, Addison Wesley Longman, Reading, Mass., 1999.


### Catalysis


- D. D'Souza and A. Wills, *Objects, Components, and Frameworks with UML: The Catalysis Approach*, Addison Wesley Longman, Reading, Mass., 1999.


### Component Life Cycle


- P. Kruchten, *The Rational Unified Process: An Introduction*, Addison Wesley Longman, 1999.
- P. Allen and S. Frost, *Component-Based Development for Enterprise Systems: Applying the Select Perspective Approach*, Cambridge Univ. Press, New York, 1998.


### Component Models and Services


- R. Sessions, *COM and DCOM: Microsoft's Vision for Distributed Objects*, John Wiley & Sons, New York, 1998.
- R. Orfali and D. Harkey, *Client/Server Programming with Java and CORBA*, John Wiley & Sons, 1998.
- A. Vogel and M. Rangarao, *Programming with Enterprise Java Beans, JTS, and OTS*, John Wiley & Sons, 1999.


### General Sources

- Component strategies: www.componentmag.com
- Distributed computing: www.distributedcomputing.com
- Application development trends: www.ADTmag.com

agement and asynchronous communication. On non-Microsoft platforms, the OMG has defined the Object Management Architecture, which consists of CORBA with a set of services for transaction management, security, persistence, and so on.

More recently, a Java-based component model for enterprise applications has also been developed. The Enterprise JavaBeans specification defines a server-side component model for Java. It aims to simplify the assembly of Java applications by describing a set of services (such as transactions and persistence) that all implementers of the EJB standard must support. Already, more than 25 vendors have stated their intention to support this standard. While the EJB specification is still immature, it has the potential for a strong enterprise-scale Java-based approach to distributed-systems development.

## Tools

Tools are an essential part of making any solution effective, affordable, and repeatable. This is particularly true when developing enterprise-scale distributed systems. In considering the available tools, we can categorize tool support based on the kinds of vendors supplying tools.

- *Smaller vendors offering focused solutions.* The vast market for tools supporting distributed-systems development has resulted in a large number of small companies offering specific solutions. The most successful of these typically provide innovative approaches targeting the latest technologies or help bridge gaps in the technologies to enable integration of disparate systems. Vendors in this category include CrossWinds, Rogue Wave, and SilverStream.
- *Infrastructure vendors making their infrastructure products more usable.* Because the infrastructure for distributed systems plays such a key role, the vendors of infrastructure products form an important part of any solution. One way they attempt to improve their effectiveness is to ensure that they offer a range of technologies supporting the successful introduction, deployment, and management of their products. The larger vendors—Microsoft, IBM, and Sun—can develop many such tools themselves. However, building close relationships with third-party tool vendors also forms a key part of this market. Consequently, we see a bewildering array of tool combinations, announcements of alliances and partnerships, and tool-integration choices. This alliance trend is also the source of many merger and acquisition activities as the larger infrastructure vendors consolidate their hold in their markets by offering more closely coordinated sets of tools. Other vendors in this category include database vendors such as Oracle and Sybase, and middleware vendors such as Iona and BEA.
- *Independent enterprise application-development tool vendors.* Many customers require greater flexibility in their choice of middleware solutions and do not wish to be closely tied to one particular technology. A number of vendors provide tools aimed at offering some level of middleware independence. They offer a robust suite of tools, services, and solutions for developing enterprise-scale systems targeting a range of component infrastructure technologies. Vendors in this category include Rational and Sterling Software.

## Adoption services

Numerous studies have demonstrated that the experience and training of the people involved in a

project are the most influential factors on the productivity and quality of the resulting software-intensive system. So, it is important to assess the current state of education and training services for distributed systems.

As organizations embrace new distributed technologies, they invariably encounter problems in finding the right information about the technology, applying it effectively in a specific organizational context, and measuring the effectiveness of their attempts. Certainly, we have much to gain from general organizational-improvement techniques as defined in efforts such as the Software Engineering Institute's Capability Maturity Model and the International Standards Organization's 9000 series standards. But, in general, these do not particularly focus on the specific needs of organizations developing enterprise-scale distributed systems.

From a commercial perspective, two areas offer hope that a more mature approach to organizational effectiveness is emerging:

♦ *Wide availability of solutions providers and consultants targeting specific distributed-systems technologies*. In the past few years, a number of small organizations have sprung up to provide education and consulting services for larger organizations trying to improve their distributed-systems development practices. These tend to focus on a specific technology solution, with extensive skills in the application and use. Examples include Castek, MTW, and The Theory Center.

♦ *Education and training materials*. A number of high-quality magazines, texts, and reports now address key issues in developing enterprise-scale distributed systems. These provide information on individual success stories, describe the best techniques to apply in different situations, define the underlying concepts of the approaches in detail, and explain the nuances of individual technologies (see the boxed text on p. X for examples). Similarly, a number of influential authors and commentators on these technologies are beginning to be recognized. Examples include David Chappell, Roger Sessions, Peter Coad, and Ann Thomas.

Building distributed systems more effectively and efficiently is an essential goal of software engineering. We are driven by the push toward greater use of commercial packages, the need to improve access to legacy data and services, and the new business opportunities offered by Web-based technologies and electronic commerce.

Unfortunately, the world of enterprise-scale distributed systems is complex and difficult. The problem involves many aspects, many issues to be faced, and lots of overblown rhetoric to sort through. It is unlikely that the industry will achieve the goal of building systems by "snapping together components" in anything other than the most limited, tightly scoped domains. Yet there are many signs of a growing maturity in distributed-systems approaches and technologies. ❖

**It is unlikely that we can build systems by "snapping together components" in any but the most limited, tightly scoped domains.**

**Alan W. Brown** is director of research for Sterling Software's Application Development Group, where he is responsible for advanced technology activities across the organization; e-mail alan. brown@sterling.com.