NAC

Business Services Architecture:

The Integration of Software Components and Common Services Infrastructure

Position Paper

About NAC

The Network Applications Consortium (NAC) is a strategic end-user organization dedicated to improving the interoperability of mission-critical applications in a heterogeneous inter-enterprise computing environment. The Consortium's goal is to influence the strategic direction of vendors developing enterprise application and infrastructure technologies. NAC focuses on providing vendors with input and feedback regarding product development and marketing strategies by:

- publishing papers that state NAC's strategic vision of the industry's direction;
- educating vendors on end-user enterprise-wide computing requirements;
- promoting, facilitating, and documenting collaboration among NAC members and vendors;
- advising distributed computing vendors on marketing and product development strategies.

NAC members include:

American Bureau of Shipping
American Medical Security
Bank of America
Bechtel
Barclays Bank, U.K.
Bell Atlantic
Boeing Company
Carolina Power & Light Co.
Chevron
Compaq Computer Corporation
Continental Grain Company
Exxon
MCI Telecommunications
NIKE, Inc
Pacific Bell
Pacific Gas & Electric
SmithKline Beecham
Stanford University
University of Michigan
University of Wisconsin
World Bank

This paper is the result of NAC's Strategic Interest Group (SIG) process, a collaborative effort of a subset of NAC members whose mission is to provide a cohesive NAC viewpoint on a particular industry sector or technical topic. The following NAC members were instrumental in writing this paper:

Carolina Power & Light	*Harold Albrecht, Jerry Norman*
NIKE, Inc.	*Jordan Ausman, Wallace Box, Bill Morrison, Jon Otto, Curtis White, Paul Woeltje*
Bell Atlantic	*Doug Savary*
NetResults	*Doug Obeid*
Pacific Gas & Electric	*Rob Batey, Dylan Kaufman*
University of Wisconsin	*Keith Hazelton*
AUTHORS:	*Harold Albrecht, Keith Hazelton*
EDITOR:	*Kelli Wiseth*

We welcome your feedback about this paper. For more information contact:

Doug Obeid, Executive Director
Network Applications Consortium
c/o NetResults
5214-F Diamond Heights Blvd., Suite 705
San Francisco, CA 94131

Contents

Executive Summary

Today's global, always-online economy requires business applications that are highly flexible, easy to deploy, maintain, and extend, and the IT organization is scrambling faster than ever to deliver them. Ideally, knowledge workers, who best know the business problem that needs to be solved, should be able to assemble applications independently, from a storehouse of easy-to-use, easy-to-modify "software Legos™" called *components*. Component-based application development promises to enable organizations to leverage investments in applications by allowing them to easily reuse proven software components, whether purchased or developed in-house. The ability to create applications from proven, prefabricated software components or to modify a business rule in an existing component-based application — even by non-technical staff — offers the potential to contain (and even reduce) application life-cycle development, deployment, and maintenance costs over the long term.

But there's much to be done before that promise and potential can become reality. For starters, while components on the desktop have simplified life for thousands of "visual" programmers (who also created a burgeoning market of components for end-user application development), components on the server side of the enterprise are an emerging technology, with just a few products coming to market[1] only recently. Mission-critical, enterprise-class applications that must support thousands of users will still be developed primarily using transaction processing (TP) monitors, and we believe this will likely be the case for some time.

In addition, there are three significant long-term issues that must be addressed before component-based applications can bring the "plug-and-play" quality suggested above to enterprise-class applications.

First, existing directory and security infrastructure is not yet integrated with component models (see *Integration with Common Services Infrastructure* on page 30). Many organizations (such as those comprising the NAC), have worked over the past several years toward achieving an integrated yet flexible foundation of *common network services* that provide directory and security (and many other) services across all applications. This infrastructure is essentially the "plumbing" that holds the distributed computing environment together. The NAC has been a key proponent of standards and protocols that can be implemented by all vendors in products that comprise this infrastructure or need to interact with it — basically, all network applications — with the goal of enabling a "plug and play" metaphor at the network services layer. For the NAC, the lack of seamless integration of component-based applications with existing directory and security services in particular, and with other

[1] IONA OrbixOTM (with Orbix for MVS), Microsoft Transaction Server 2.0 (with Cedar and Microsoft Message Queue (MSMQ, nee "Falcon") for Windows NT Server, and IBM Component Broker (with DB2 adapter) for Windows NT Server.

common network services in general, will be a significant impediment to the successful implementation of component-based applications. Migration to component-based applications can only be beneficial if it *leverages existing and evolving common network services*. Directory and security services are the cornerstones of a global network that can support all the business requirements of today and tomorrow, and component-based application implementations must be able to integrate with these existing services transparently.

Second, the two key distributed component models discussed in this paper, specifically, COM/DCOM/ActiveX and CORBA/IIOP/JavaBeans, are not interoperable. Although they are conceptually similar in many respects, server-based components built for one model cannot simply be moved into the other model, nor can components in one model use components in the other without bridges or other mediating technologies.[2] (See *Interoperability Challenges of Heterogeneous IT Environments* on page 33 for discussion).

Third, the market doesn't yet offer management tools required for successful deployment and scalability for enterprise-class component-based applications. The distributed, decentralized approach intrinsic to component models is unworkable without a choreographer-like function that dynamically coordinates and scales run-time operations. The recent proliferation of distributed transaction processing monitors is a welcome sign of serious vendor attention to this issue (for details, see *Manageability and Distributed Transaction Processing Monitors* on page 35).

Although these issues are far from being solved (see *Technical Issues* on page 27 for discussion), the NAC does believe that organizations can take steps now to ensure that they'll be ready to take advantage of the pre-fabricated components and component frameworks[3] as they emerge in the marketplace. Organizations should plan to move to a multi-tier, service-oriented architecture, in which strategic applications are partitioned between user services, business services, data services, and legacy services. We've put the key concepts together in the NAC's *Business Services Architecture* (BSA), discussed on page 18. Successful migration to a service-oriented architecture requires a fundamental cultural shift toward recognizing infrastructure and business services as long-term capital assets.

Further, it isn't too soon to begin limited development of multi-tier component-based applications based on the Business Services Architecture, as long as the requirements

---

[2]The many bridging mechanisms currently available are focused on the client side, not on the server side. However, IONA's recent licensing of COM technology from Microsoft and subsequent product announcements will soon change that, with an OMG compliant, bi-directional COM to CORBA server.

[3] Component *frameworks* are pre-built, partially assembled component sets that can be customized to fit your organization.

for application robustness and scalability are not too taxing. The NAC also recommends that members:

- Evangelize the concept that business services and data services must be developed and preserved as assets.
- Architect for the future by migrating to a multi-tier business services architecture in which each service partition is isolated and preserved as a long-term capital asset.
- Begin selecting your own interface standards at the corporate or major function level to ensure that services will work in current and future application development environments.
- Start (or continue) application development migration to a network services model, which leverages both new and existing common infrastructure services to achieve a seamless distributed environment with the scalability and robustness required of enterprise applications.
- Collaborate with other NAC member organizations to divide and conquer the information challenge we all face in integrating component services with existing infrastructure and in interoperating across disparate pieces of a heterogeneous IT environment. This would mean identifying areas of expertise within our companies and putting in place effective ways of leveraging that expertise by somehow sharing "latest, best available information on problem X."

The NAC members may need to make many shifts in the culture or organization, including re-engineering the application development process (see *Enabling Component-based Application Development* on page 22). For example, to enable the long-term vision of component-based application development in which business users can implement or modify applications, a technical services organization must be in place to provide the appropriate support.

Some of these steps are going to require supporting products. Organizations will look to vendors to provide interoperable modeling tools, repositories, and other tools that will enable non-technical staff to implement the applications they need. Organizations need robust, scaleable tools for modeling business problems effectively, and we also need enterprise-wide repositories that can store all components that will be available to others, including business users.

The interoperability among tools and repositories is at one end of the development-deployment cycle; at the other end is the requirement for interoperability among the applications and products. The NAC recommends that vendors help define interoperability paths among distributed component-based server products, the object-request brokers, the Web servers and browsers, the components that get downloaded at runtime, and so on. As the NAC has stated since 1991, we have no choice but to work in heterogeneous environments. And we don't always know what that "heterogeneity" is going to comprise.

Thus, in general, the NAC encourages vendors to honor each other's best contributions, whether competitor or business-partner, by *matching features* and *defining interoperability paths among products*. Vendors that don't provide bridging or other needed interoperability technology for their products should cooperate fully with companies that do by providing all necessary information for product development efforts. Microsoft's recent licensing of COM to IONA for use in the Orbix product line is an example: while Microsoft may not do CORBA, that doesn't mean that CORBA vendors can't do COM.

Further, the NAC urges vendors to work together as appropriate to solve mutual technological problems, and share information about solutions to benefit the industry at large. A good example of this type of effort is JavaSoft's adoption of Lotus' InfoBus technology into the JavaBeans component architecture.

Other specific recommendations to vendors that will help NAC member companies move towards a component-based Business Services Architecture are:

- Provide component frameworks that support seamless interfaces to whichever implementation of standards-based common network services match the customer's environment. End-to-end support of role based authorization credentials obtained under a single sign-on, in conjunction with mapping of external users to a particular role, based on certification by some acknowledged certificate authority, and end-to-end component/service location transparency based on common directory services, are just a few examples of key requirements.
- Provide component-based, framework-based enterprise application suites that will interoperate with each other, and with the component-based services and applications we have developed for ourselves. The IBM San Francisco project embodies many of the concepts that NAC would like to see implemented, particularly with the evolution of the foundation layer to a common CORBA/Java based distributed component model. However, at the moment, it appears to fall short in the area of providing seamless interfaces to standards-based common network services of the customer's choice.
- Accept the concept of a "universal thin client," a conservative assumption about the desktop environments of all those to whom we would like to extend our business services. The NAC's concept of the "universal thin client" today includes a web browser, a Java Virtual Machine, and support for distributed component computing. There should be no other requirements, such as platform or operating system dependencies. The universal thin client must be supported by the business services component frameworks, whether inside or outside the enterprise.
- Provide user services component frameworks that make it easy to support a universal thin client, with zero configuration and deployment, because this is becoming a pervasive requirement for access by business partners, customers, and suppliers. Allow the user to choose to perform more processing on the

server (the "thin" client model) or on the client (for ostensibly faster performance).

- Provide server-side support for distributed transaction processing (DTP) through TP monitors that can work with a wide range of underlying technologies. Note that this recommendation is driven more by the need for scalability and manageability for high volume transactions than by the narrower issue of maintaining transactional integrity. As one recent study put it, "In reality, only between five and ten percent of the code in TP monitors is about synchronizing transactions."[4]

Full discussion of the recommendations begins on page 36.

[4] Jeri Edwards with Deborah Devoe, 3-*Tier Client/Server At Work*. John Wiley and Sons, 1997, p. 20

Introduction

Successful organizations implement information technology for one reason, and one reason only: to meet their business goals and enhance their position in the marketplace. Whether the goal is to sell the most shoes, provide the highest-quality education, or deliver the most cost-effective energy that consumers can buy, organizations use information technology (IT), directly and indirectly, to meet or exceed their business goals.

There's nothing new about this premise. However, given today's frenetic pace of change, IT is required to provide new solutions faster than ever as organizations scramble to meet increased customer demands, develop new profit opportunities, and succeed in a highly competitive global marketplace that has been shaped by decades of world-wide political change, including removal of many of the barriers to international and local trade. Within the United States, for example, deregulation over the past decade has changed and continues to change the competitive landscape for the banking, energy, telecommunications, and transportation industries.

In addition, advances in communications and computer technology, as well as widespread use of the Internet, have changed market dynamics dramatically. For example, by enabling organizations of any size to compete on a more-or-less level playing field called cyberspace, the Internet keeps today's consumers just a hot-link away from numerous competitors — regardless of location, time zone, or long-term viability of the company behind the Web site.

Thus, a key focus for organizations today is determining how to attract, retain, and support customers in innovative ways, and IT must develop applications accordingly. Broadly stated, today's application requirements include enabling the organization to conduct more business, with more people, in more places, with more frequency, through every conceivable type of media. For example, an organization may want to provide a single-point-of-contact for customer service that can be accessed at any time of day or night, whether it be through a $299 Network Computer, semi-intelligent occasionally connected Personal Digital Assistant, or via voice through a telephone. Projects like these are at the top of IT's list of deliverables.

Furthermore, the IT organization must provide the applications to support the business strategy faster than the competition — at "Web speed:" IT can no longer take 18, 12, or even 6 months to develop, test, and deploy applications to meet business goals.

Developers are being asked to deliver new business services at a point-and-click pace because time itself is a strategic weapon.[5]

The net result is that applications must be flexible and provide leverage. They must be easy to deploy, maintain, and modify when needed to enable the organization to effectively profit from emerging market opportunities. IT must be able to re-use key aspects of business applications rather than creating anew each time the business requirements change. *Component-based applications,* built using established design principles and combined with multi-tiered architectural models, hold the promise of providing the needed flexibility, development speed, and implementation ease.

In simple terms, *components* are functional software units that can interact with other functional units, typically in the context of a supporting application, such as a Web browser, word processor, or spreadsheet, or on their own. For example, one component from Adobe Systems, an ActiveX Control, extends the functionality of Microsoft Word so that end-users can save documents as Acrobat pdf (portable document format) files for easier information sharing with others. Word automatically "knows about" this ActiveX Control because its functions (services) — saving the content of a Word document in the Acrobat file type — are "published" system-wide through a mechanism known as Automation[6], so automatically the Acrobat file format is added to Microsoft Word's save routines. ActiveX Controls in this context are just one example of the hundreds of components available for the desktop today.

The example highlights just a couple of the key benefits of components and component-based applications. Specifically, incremental functionality — in this case, an additional file format — is added with relative ease because the component approach is *modular*. Rather than installing an entirely new version of Microsoft Word that would include the added functionality, the user has only to install the ActiveX Control — a single 75K file — to gain the additional save routines. In addition, the component has a means of interacting with other software from another vendor without the second vendor knowing the internals of the component — yet the component does what it's supposed to do. In software developers' slang, the component is said to be a "black box"[7], an externally identifiable entity that provides a known service, yet hides its internal mechanisms. This modular encapsulation can simplify the developers' job by providing a means to use existing software without having to learn about its internal design or code in order to understand how to use it.

---

[5] *"Time — The Next Source of Competitive Advantage."* G. Stalk, Jr. The State of Strategy. A Harvard Business Review Paperback. 1991.

[6] Formerly known as "OLE Automation."

[7] A more formal definition of black box: "A process with known inputs, known outputs, and a known function but with an unknown (or irrelevant) internal mechanism." *The Practical Guide to Structured Systems Design. Meilir Page-Hones. Yourdon Press Computing Series. 1988.*

Another frequently cited potential benefit of component-based applications is lower total-cost-of-ownership (TCO). Over the long-term, component-based applications promise to lower the TCO over the full life-cycle of the application, from development, through deployment and maintenance. Components are expected to do this by finally providing the level of re-use that development organizations have been trying to achieve for several generations. However, many changes to application development processes must occur within an organization before component-based applications can be effectively implemented. For example, the application development process itself may need to be re-engineered[8]. Thus, despite all the hype about lower TCO, the Network Applications Consortium (NAC) believes that the component-based application model will increase rather than decrease costs in the short run.

However, it is the ability to extend and leverage applications over the long-term — not decrease short-term costs — that is a primary potential benefit of component-based applications. Flexibility and extensibility are the two chief characteristics that the NAC requires for distributed enterprise-class business applications. Specifically, NAC organizations must be able to apply component technology to the back-end services that comprise their enterprise applications portfolio, not just on the desktop.

For example, say an organization has an existing database system that provides the basis for many of its financial applications, including the company's general ledger. The controller wants the general ledger system modified such that he will be notified automatically, by an email message generated directly from the database, when the accounts receivable total for any single customer is greater than $5,000 and is aged beyond 90 days. Applying the concepts presented in the Adobe ActiveX Control scenario to such a change, the functionality of the general ledger system should be extendible with the addition of the appropriate component to the system, one that generates an email message based on this specific business rule.

Note that this example assumes a great deal of *transparent interoperability* behind the scenes: the email system and the database service work together, just as the Adobe ActiveX Control and Word could work together, to provide the necessary functionality. This foundation upon which all enterprise-wide cross-application interactions depend is referred to by the NAC as the *network services infrastructure*, and it is a basic requirement that applications, component-based or otherwise, make use of this infrastructure in a transparent manner. Unfortunately for organizations today, this is not usually the case.

The need for component-based applications to integrate transparently with the network services infrastructure layer is one key message in this paper, in particular the as-yet unmet need to integrate with existing and evolving directory and security services.

---

[8] See *Enabling Component-based Application Development* beginning on page 21 for a discussion of application development process issues.

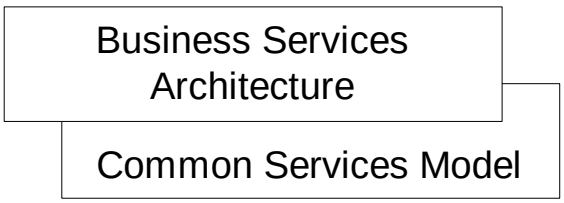
Directory and security services are the cornerstones of a global network that can support all the business requirements of today and tomorrow, and component-based application implementations must be able to integrate with these services transparently.

In this paper, the NAC presents a generic Business Services Architecture (BSA) as the recommended way to design, build, and deploy distributed, enterprise-class component-based applications (see *NAC's Component-based Business Services Architecture* on page 18). The NAC BSA synthesizes key concepts from multi-tier distributed application architecture, service-oriented architecture, component-based application development, and the common network services model, and these are introduced first, in the *Functional Overview: Enterprise-wide Component-based Applications* beginning on page 5.

In addition, NAC presents an overview of the application development process and organizational changes that will be required to support this highly flexible application development paradigm (see *Enabling Component-based Application Development* on page 22). Beyond the organizational and process issues are significant technology challenges that must be addressed, as discussed starting on page 27. Given the technical issues, NAC concludes with recommendations to vendors, member companies, and other organizations about what they can do today to begin realizing the NAC's long-term vision of application assembly by business experts rather than programmers.

Functional Overview: Enterprise-wide Component-based Applications

The NAC's Business Services Architecture (BSA) is a conceptual framework used to highlight key issues relative to component-based applications. Synthesizing key features from multi-tier, service-oriented architectures; component-based application development models; and the Burton Group's Network Services Model, the BSA also provides a flexible foundation for designing, deploying, maintaining, and extending distributed, component-based enterprise-class applications. The key characteristics and benefits of the underlying architectures and models are discussed briefly in this section before presenting the BSA.



Multi-Tier Architectural Characteristics that Enable Distributed Computing

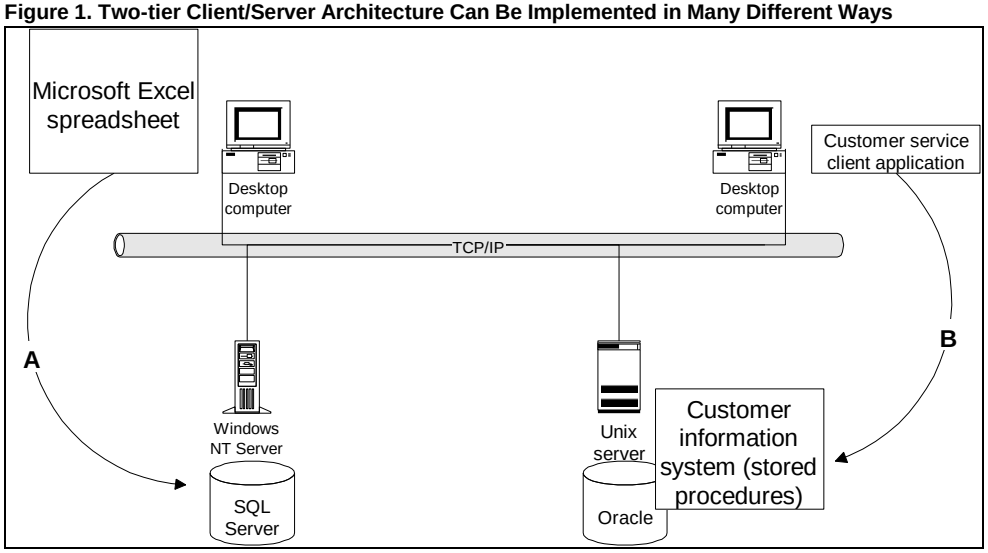
From the highest-level viewpoint an application can be *monolithic* or *multi-tier*. (A tier is a logical, not a physical, construct.) Monolithic applications are those in which client presentation logic, business logic, business-to-database schema mapping, and connectivity logic — in essence, the entire application from one end to the other — is designed as a complete unit. Core business logic is buried deep within the application. When business requirements or rules change, it's difficult and time consuming to "get at" and change the business logic. Many so-called legacy applications were built in this manner, modular design techniques notwithstanding: the elements that comprise the application were compiled and deployed as a unit, not available to be used by other applications.

Unlike monolithic applications, a multi-tier application architecture partitions the programming logic into individual functional units. Early client/server development activities in the late-1980s and early-1990s defined a two-tier model, typically implemented as a client application and a database server. As with monolithic applications, two-tier client/server applications suffer from inflexibility and high maintenance costs, mostly because the business rules are held hostage in either the client, the database, or a mix of both.

For example, in the two-tier client/server implementation below (A), much of the programming logic that defines the business rules is contained as code on the client, as VB for Applications (Visual Basic for Applications) scripts, for example. On the other

hand, in implementation (B), much of the programming logic that defines the business rules is contained in the database itself, as stored procedures[9].

Figure 1. Two-tier Client/Server Architecture Can Be Implemented in Many Different Ways



Regardless of whether the two-tier approach is client-centric (A) or database-centric (B), the two-tier design doesn't scale well. In the client-centric approach, updating an application involves distributing software to every client workstation. In the database-centric approach, additional client applications cannot seamlessly take advantage of the business rules without taking the application apart. For example, witness the difficulties in integrating two different two-tier applications:

> *At the forefront of the client/server movement, Acme Gizmo Enterprises (AGE) first implemented an Oracle database application in 1992 in its customer service department. To ensure fast response time to hundreds of customer service representatives taking orders over the phone, the developers used numerous stored procedures in a two-tier client/server implementation. The application has served AGE well, but now that Global Unified Luxury Products (GULP) has acquired AGE, AGE's customer-related functions must be incorporated into GULP's customer information system.*
>
> *Like AGE, GULP also had taken a two-tier client/server approach to its customer information systems, but its business rules are buried in the client application as well as the server. As an example of the business rules: Whereas AGE allowed customer orders of up to $500 to be processed without a credit check, GULP requires orders over $200 to be approved by a credit manager. Rules such as these are buried within the code of both*

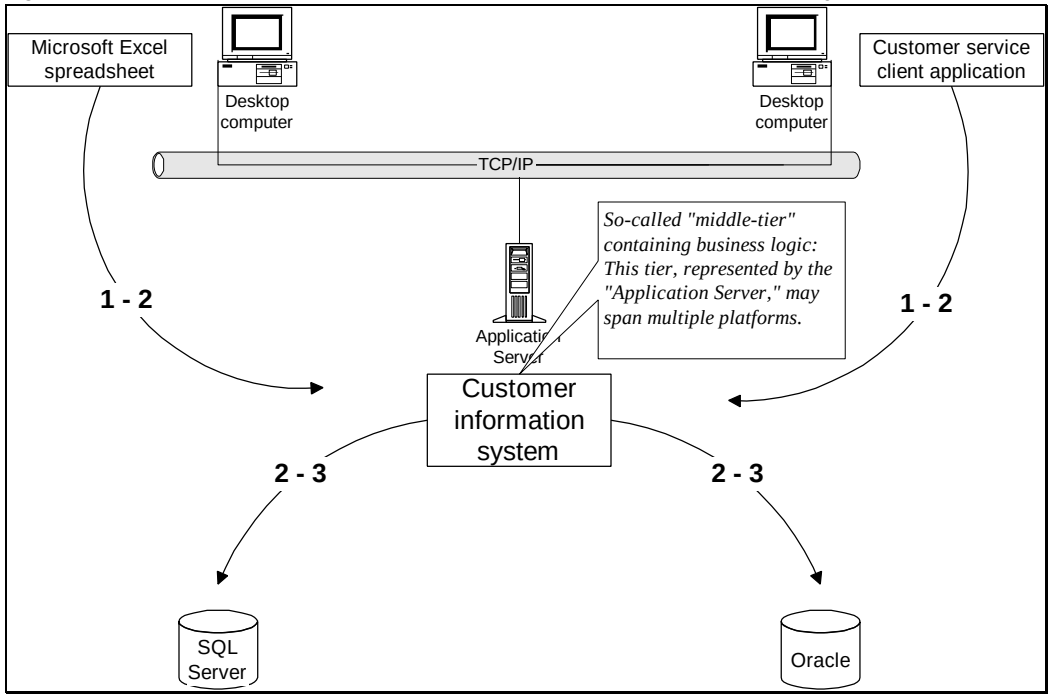
[9] Batch SQL routines stored in the database.

systems, in one case, in the client applications, in the other case, in the database itself as stored procedures.

The bottom line is that unification of the two systems will likely require extensive research, detailed analysis, and re-coding from end-to-end (both client applications and server applications) because of the two-tier architecture.

The three-tier client/server architecture is an improvement over the two-tier model because it goes further to partition the functions comprising the application into truly modular units. *Modularity* is one of the chief characteristics of multi-tier architectures that enhances application flexibility: groups of related functions and data are packaged into units. Presentation logic, business logic, and data mapping and access logic are contained in separate modules.

Figure 2. Three-tier Client/Server Architecture Enables Re-use of Business Rules by Other Clients

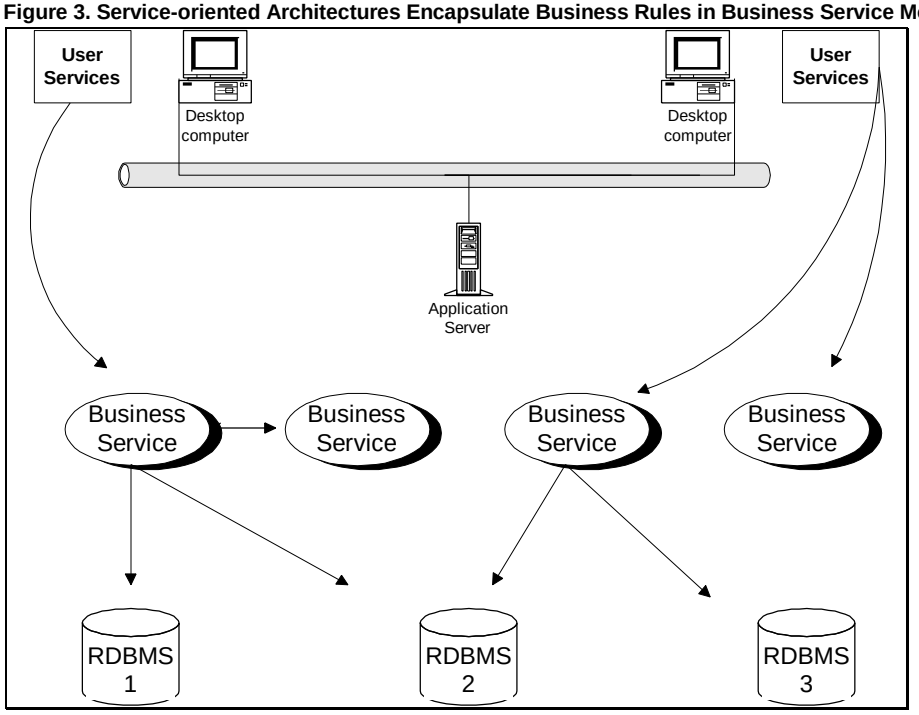


For example, in the three-tier client/server model shown in the figure above, much of the programming logic that defines the business rules is contained in the middle tier of the application. This logical tier containing the business rules is separate from the presentation logic that calls it and the databases that it in turn needs to access (2-3). This results in the ability to modify the business rules as needed, without having to modify all other participating tiers, and also leads to the possibility of leveraging these business rules in other applications.

Service Oriented Architecture

A *service-oriented architecture* takes the concept of modularity and separation of data and business rules further still. Gartner Group defines a service-oriented architecture as "a particular style of multi-tier computing that helps enterprises share logic and data. It assumes multiple software tiers … and leverages the principle that many aspects of processing logic are common to many users of some particular data set rather than being uniquely associated with one particular application… A service is a black box that hides code and data from the developer of the client application… A service-oriented architecture maximizes code reuse and minimizes the redundancy of logic and data by organizing functions into shareable, encapsulated modules that can be accessed from multiple requestors."[10]

Figure 3. Service-oriented Architectures Encapsulate Business Rules in Business Service Module



The "black box" concept originated from structured design techniques in which a module encapsulates a specific function: the inputs, outputs, and the function itself are known externally, but how the service performs its functions is not. Expanding on the scenario introduced on page 6 to include service-oriented architecture concepts:

> *After struggling to integrate two-tier client/server customer service systems from two disparate organizations into a newly merged, company-wide system, the IT planners realized it would be best to step back and take a*

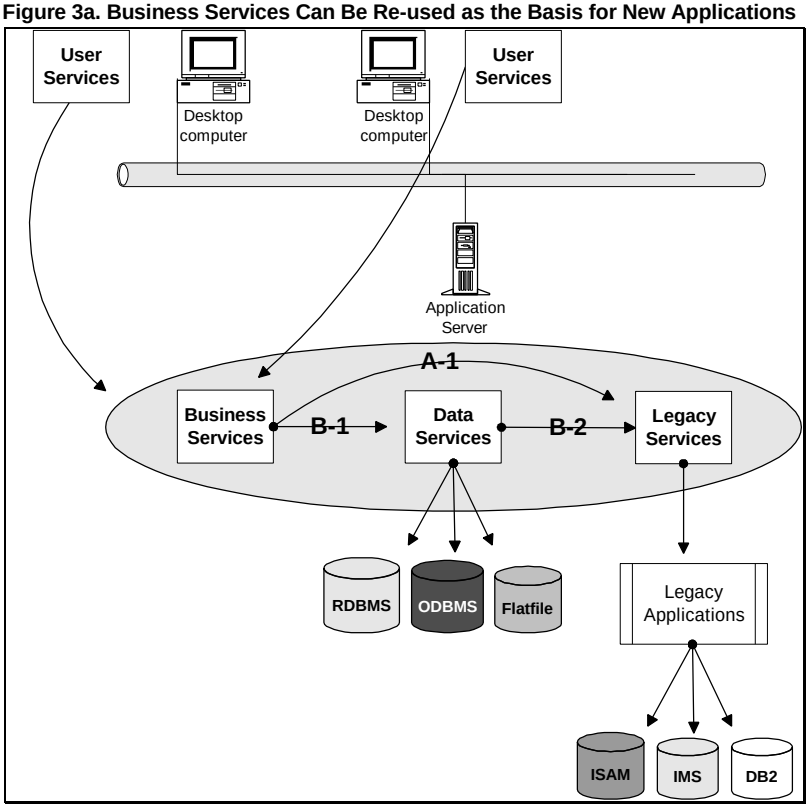
[10] "Architecture and Planning for Modern Application Styles." GartnerGroup Systems Software Architectures (SSA) Strategic Analysis Report. R. Schulte. 4/28/97.

longer term view. They decided they should provide a high-level overall architectural plan for the newly merged organization, and that taking a service-oriented approach to this architecture would be strategic to the company's success in the future. First, they put together an architecture team to identify the major functions and data constituents of all systems across both companies.

The architecture team extended the principles of the service-oriented architecture to include existing databases and legacy systems as well, and they created the high-level, simplified model shown below as a viable architecture that would enable the two organizations to leverage what they each currently had, without starting from ground zero.

For example, by abstracting the data access mechanisms as a logical tier (data services), separate from the business service, they would be able to provide a business service containing the business rules of the merged company, yet keep the databases of each of the old systems as they were. The business service could call the data services layer (B-1), which in turn could call the specific database. Or the data services layer can call the legacy service (B-2), which in turn can function as an abstraction layer to legacy applications. For example, the system might include data updates through CICS to DB2 data on the mainframe.

Figure 3a. Business Services Can Be Re-used as the Basis for New Applications



With the business rules isolated in a single tier, they would be easier to change in the future – in the case of another merger or acquisition, or even less dramatic changes, such as when prices change or state sales tax rate changes. Better still, any other new applications created can access this business service.

Thus, rather than coding and replicating the same business logic throughout the organization's systems — once for the mainframe, once for the internal, client/server network, once for Internet access, once for special business partners to access, and so on — the business service is provided on an enterprise-wide basis.[11]

The benefits of a service-oriented architecture are significant: The organization can begin to realize one key benefit, **re-use of business logic** at the application level. But a crucial piece is still missing: large enterprises depend on many behind-the-scenes network services to tie the pieces together, as discussed in the next section.

Common Services Infrastructure

A multi-tier, service-oriented architecture depends on an infrastructure of core *network services*. This infrastructure has been defined by The Burton Group (http://www.tbg.com) as the "Network Services Model.[12]" The Burton Group developed this model in 1991 and has been evangelizing the model as a basis for global, interoperable networking. The NAC endorsed the basic premise of the Network Services Model in its first position paper[13] on interoperability in 1994 and has adapted various aspects of that model for use in subsequent papers, which it refers to as the "common network services model," or simply, the "common services model."

Briefly, the key concept from The Burton Group's Network Services Model that NAC's common services model includes is this: each one of a core set of critical functions must be performed for all network entities in a distributed computing environment by a single, unified service. The NAC's mission over the past several years has been to evangelize the proposition that these common services, which become an organization's information infrastructure, must be leveraged – not re-invented with each new application that is deployed. The set of common services provides:

---

[11] According to the Gartner Group, its service-oriented architecture is "not sufficient for integrating applications that are designed by different development teams.[11]" But the NAC believes that by providing a network-services-model-based foundation and using interoperable component technology to develop business and other services, that is precisely what can be achieved. These ideas will be brought together in the Business Services Architecture discussion.

[12] "Intranets, the Network Services Model, and the Future of the NOS." Jamie Lewis, The Burton Group. Network Strategy Overview, July, 1996. Section 3.4 The Role of Component Software in Two- and Three-Tier Architectures.

[13] See "*Interoperability: A NAC Position Paper*" August 1994. Available at http://www.netapps.org

- A way for network entities of all types — human users; resources, such as printers, workstations, and servers; as well as software entities such as applications, common services, and components — to determine what other entities are available on the network, what they are called, and how to find each other. These functions are provided by **directory services.**[14]

- A means of maintaining integrity, accuracy, and privacy of all information and resources on the network. These functions are provided by **security services**, which verify the identities of people, processes, code modules, and all network entities that require authentication. Once authenticated, security service mechanisms ensure that all network entities — again, both people and processes — have access only to services and information for which they have been authorized.

- A means of ensuring that related activities can be performed in such a way that, despite any system or other failures, information and resources maintain their integrity. This function is provided by **transaction services.**

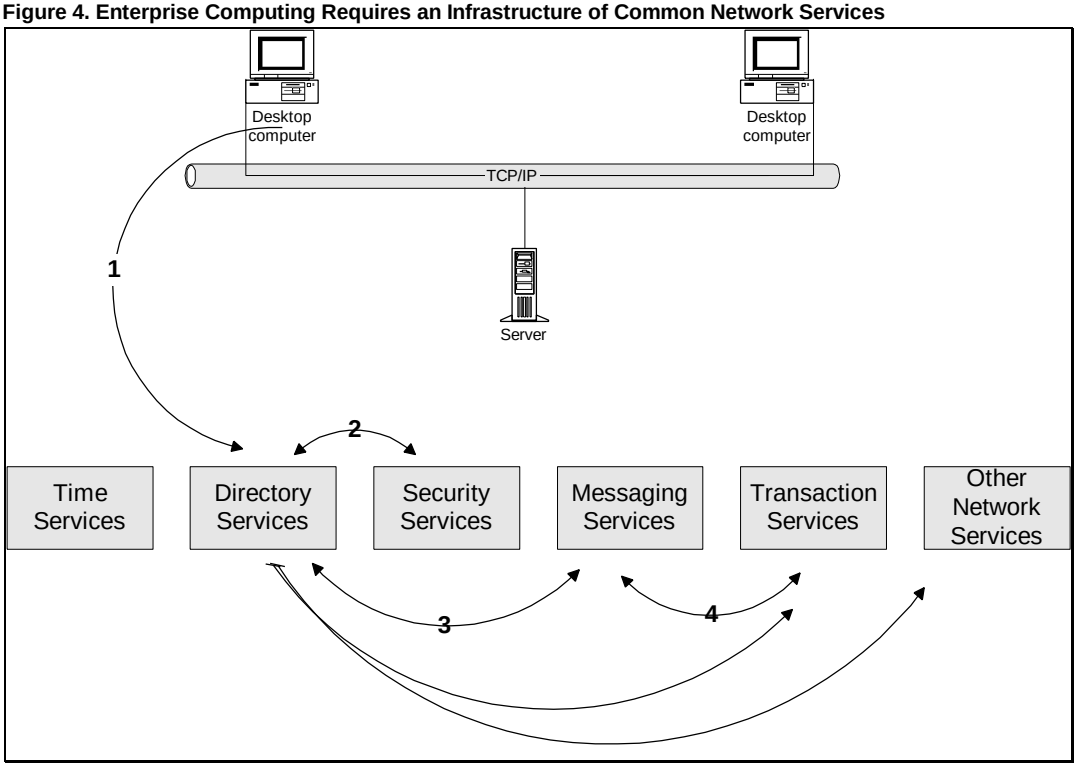
  For example, an ATM (automated teller machine) withdrawal might deduct an amount from one database table and add an amount to another table in another database. If a process or the power fails during the course of this activity, transaction services ensure that all database tables maintain their integrity (based upon the state of the transaction at the time of failure).

- A means of communicating among all network entities, including human users, even when all participants are not actively connected to the network. This functionality is provided by **messaging services**. For example, a store-and-forward email system messaging service provides a place for email messages to be stored for later delivery, when the recipient accesses his email.

  A message queuing service provides this same functionality for transaction processing activities that cannot occur in "real time" for whatever reason. For example, many OLTP (online transaction processing) systems rely on message queuing facilities to store in-process transactions when a database is off-line for reloading or backup. This communication model is also essential for wireless or occasionally connected mobile users with laptop or palmtop computers or digital appliances.

[14] Although naming and directory services are often discussed separately, particularly in detailed technical and product architectures, we use the term **directory services** to include both.

Figure 4. Enterprise Computing Requires an Infrastructure of Common Network Services



So for example, in a typical (ideal) scenario, a customer service representative might log on to the customer information system using a user ID and password.[15] The **directory service (1)** would locate the **security service (2)**, and would validate the user by comparing credentials held in the directory. The customer information system also uses the **messaging system (3)** to process approval for orders over a certain dollar amount; an email message is created and sent to the credit manager. The database tables remain in synch regardless of any failures across the system that might occur prior to approval and completion of this process because the entire process is protected by the **transaction service (4)**.

Several other common network services are part of the core set as well. For example, **management services** provide a wide range of functions including software distribution, desktop management, network and systems monitoring, problem reporting and help desk facilities, and more recently, distributed application monitoring and management. **Time services** are crucial in a distributed network environment, essentially functioning as a network clock that provides consistent time-stamping for all events throughout the network.

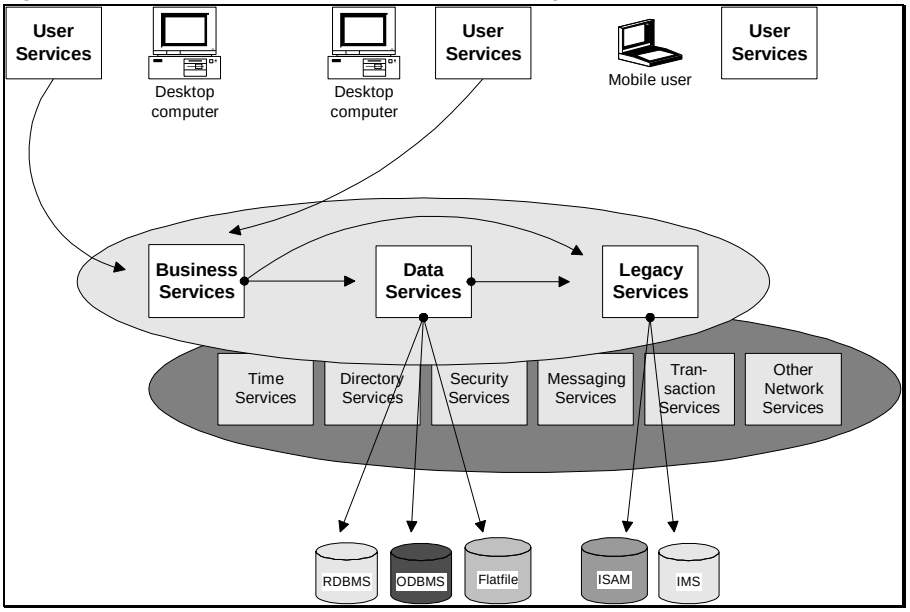
[15] Ideally, the user's token from his initial network logon would be used to access the customer information service, but that's a topic unto itself. For details, see NAC's paper on single sign-on entitled *Enterprise Directory Services Integration, Enterprise-wide Security: Authentication and Single Sign-on.* July 1996. (http://www.netapps.org)

However, full discussion of these and other common services[16] is beyond the scope of this paper. The relevant points for this discussion are these:

- the common network services must interoperate with each other
- component-based distributed applications must seamlessly integrate with the existing common network services infrastructure

The concept of being able to *leverage existing common services that can be used by all applications* has always been central to the NAC's definition of interoperability.[17]

Figure 4a. The NAC Business Services Architecture Requires Common Network Services



These common services are needed to support distributed computing in all its forms, whether limited to the confines of a single organization or spanning a world-wide global distributed computing network, and whether based on a private or public communications network. Network entities must be able to determine what other entities are available, find each other, ensure that they are who or what they claim to be and that they have the right to do what they want to do, and ensure that all processes and information maintain their integrity.

---

[16] The Burton Group's Network Services Model today includes Web Services and Object Services, in addition to the original model's File, Print, Directory, Security, Messaging, and Management services. For more information about The Network Services Model, contact The Burton Group (http://www.tbg.com). For further discussion of interoperability and the common services model, contact the Network Applications Consortium (http://www.netapps.org).

[17] "NAC's definition of interoperability has two dimensions. Interoperability provides IT managers with (1) the ability to mix-and-match the building-block components and applications that comprise the IT infrastructure; and (2) the use of a common set of service functions shared by all applications." From *Interoperability: A NAC Position Paper.* August 1994.

The recognition that applications must be supported by the common network services infrastructure is what the NAC believes is missing from other discussions of enterprise application issues. The NAC's business services application architecture, discussed on page 18, includes this important element.

Components and Component Models

So far the discussion of architecture at the enterprise-wide application level has been purely conceptual, highlighting the benefits of various approaches. But implementing an application involves choosing specific application development models and specific tools, which in turn support specific interfaces and techniques for creating business applications. (A specific organization's architecture likely includes lists of such things in the way of standards.)

The NAC defines a ***component*** as ***an executable whose behavior can be customized by an end-user without modifying source code***. Components are capable of providing the very needed benefit of re-use and flexibility, while retaining their ease-of-use. Components can be easier to use to create applications because you don't need to understand how they do what they do in order to use them, and even for highly technical people this is an advantage over having to learn yet-another programming language or programming model. This is what's meant by "ability to be customized by an end-user without modifying source code."

The NAC's definition is consistent with other industry analysts, including the Gartner Group, which defines a component as "a dynamically bindable package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at runtime. In other words, a component is a black box that is particularly friendly to the developer because it is implemented with a formal mechanism for defining and managing the parameters in the program-to-program messages."[18] The net result is that components, as we've defined them, can turn some of the chief features of object-orientation, specifically, encapsulation and abstraction, into the real business advantage of re-usable executable code that can be used to easily and quickly assemble business applications.

In practical terms, NAC defines components specifically in terms of the two key alternatives widely available today, Microsoft's ActiveX Controls and JavaSoft's JavaBeans. An ActiveX component is language independent but operating system specific (although not inherently limited to one OS, because it is based on a binary standard); the JavaBeans component is language specific but operating system neutral, as long as a Java Virtual Machine (JVM) is present.

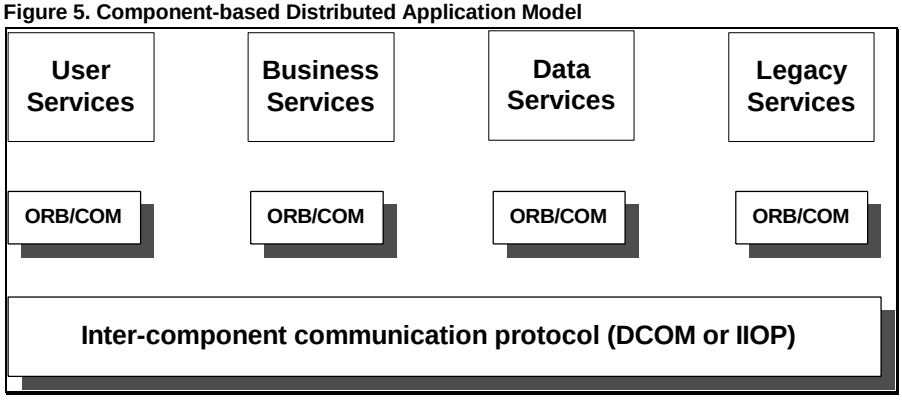
---

[18] "Architecture and Planning for Modern Application Styles." GartnerGroup Systems Software Architectures (SSA) Strategic Analysis Report. R. Schulte. 4/28/97.

To be usable, components must provide mechanisms that enable a development environment to discover the properties that can be modified and the events that it generates. Components must allow for graphical editing of their properties, and they must be directly customizable from a programming language. Finally, there must be a way to put components together — semantically link them.

To support this model in a distributed computing environment requires communications infrastructure. Microsoft's communications infrastructure that supports a distributed component model is called COM/DCOM (Component Object Model/Distributed COM). The NAC believes that the OMG's (Object Management Group) CORBA/IIOP (Common Object Request Broker/Internet-InterORB Protocol) in conjunction with JavaBeans is conceptually equivalent to the COM/DCOM/ActiveX distributed component model.[19] The figure below generalizes the key concepts from both models in the context of the service-oriented architecture.

Figure 5. Component-based Distributed Application Model



This is another view of the service-oriented architecture presented earlier, a view that brings in more specific concepts of the broker and the communication protocol (or "wire protocol"). A CORBA-compliant ORB (object request broker) handles requests and returns results among objects. With IIOP, different ORB products from different vendors can work together to handle such requests. The Microsoft component object model and DCOM provide similar functionality. In the figure above, user services, business services, data services, and legacy services[20] are implemented as components using one of these models. To some extent, you can use both models in certain circumstances. For example, you can embed ActiveX components in JavaBeans, and

[19]Although the OMG's CORBA/IIOP (CORBA 2.0) has no corollary to ActiveX Controls (nee OLE Controls) at this time, JavaSoft's JavaBeans is used by many ORB (object request broker) vendors and distributed application developers to fill this gap. And it's expected that CORBA 3.0, due by Q4 '98, will include a JavaBeans component model (currently referred to as CORBAbeans) as part of the standard, thus the NAC has anticipated this model.

[20] Actually, a component interface is wrapped around the legacy application.

embed JavaBeans in ActiveX, to the extent that the various compilers have hooks or supporting mechanisms.

The important parts of a component include its interface, methods, properties, and events. The interface is often described as the "contract" between software components because it establishes expected behavior and responsibilities. Another way of thinking of an interface is as a collection of methods (or functions): what are all the things this component can do? What can be changed? How does another component use it? These are the types of questions that methods, properties, and events answer, and the interface is the contract that makes these aspects evident. Components export one or more interfaces, each of which supports one or more methods. So for example, in the figure above, a business service component called Customer might have an interface that supports methods for requesting an account, or changing address information, or requesting a higher credit limit.

At a low level, an interface definition language (IDL) is used to define the interface programmatically. The OMG's IDL is what enables CORBA to achieve its heterogeneity because developers can define methods in any programming language that provides CORBA bindings (Java, COBOL, C, C++, Ada, Smalltalk).

The Microsoft IDL (MIDL) has its origins in DCE/RPC. Beginning with Windows NT 3.5, the MIDL compiler was extended to support COM interfaces. So although both models are based on the concept that "interface is separate from implementation," the interface definition languages and IDL compilers that they use are different.

Scripting languages are commonly used to define the interactions between components in a component-based application. Server-side scripting languages (such as JavaScript, VBScript, and JScript) can be used to create server-side applications. JavaScript is Netscape's cross-platform, object-based scripting language for client and server applications. Navigator JavaScript is used for client side applications, and LiveWire JavaScript is used server-side. Microsoft also has several scripting languages, including the Visual Basic scripting language and JScript. JScript is the Microsoft implementation of the ECMA 262 language specification. It is a full implementation, plus some enhancements that take advantage of capabilities of Microsoft's Internet Explorer.

To expand upon the service-oriented architecture scenario above to include the distributed component-based application model:

> *The architecture team articulated its vision of the service-oriented architecture to the application development (AD) organization, which had always been keenly aware of the benefits of re-use and was implementing prototypes of key business services using component-based application development tools. The AD organization wanted to leverage not only code, but skills as well: with a vast number of VB (Visual Basic) programmers in*

the group, they could make the transition to component-based applications fairly easily.

*The **ProductOrdering** business service was put together from several COM components. These included the CreditCheckComponent, which contained the rules for when to send a message to the accounts receivable clerk to approve purchases over a specified dollar limit. The Shipping component contained the shipping rates, destinations, and delivery timetables. The SalesTax component contained all the state sales tax rates. The Order component itself was built from several other components, including the Inventory component, which reduces inventory for each order accordingly; the CustomerAccount component, which records the purchase to accounts receivable (another component) and also posts the item to the customer history file.*

The CustomerAccount component was used in many other applications throughout the company. For example, the Sales organization implemented this component in its Field application using a Web browser.

These components were integrated using scripts and hosted on Microsoft Transaction Server. The client application was Web browser based, and the only expectation was that the client could support HTML, and that the client included a JVM (Java Virtual Machine). This meant that it didn't matter if the end-user had a Mac, a PC, or even a Unix workstation. When users click on the internal Web site, the HTML page is downloaded to their workstation. The Jscript pops up some dialogs that gather input information, such as Customer name, address, phone, and so forth.

This application could also have been built using CORBA/IIOP/JavaBeans components, with essentially the same result. The point of this paper is not to debate the relative merits of either approach. Both COM/DCOM/ActiveX and CORBA/IIOP/JavaBeans enable component re-use. The important question for the NAC is how to integrate both models — since NAC member organizations must support both — with the existing and evolving common network services infrastructure. That's why the NAC's Business Services Architecture, described in the next section, provides for both models.

NAC's Component-based Business Services Architecture

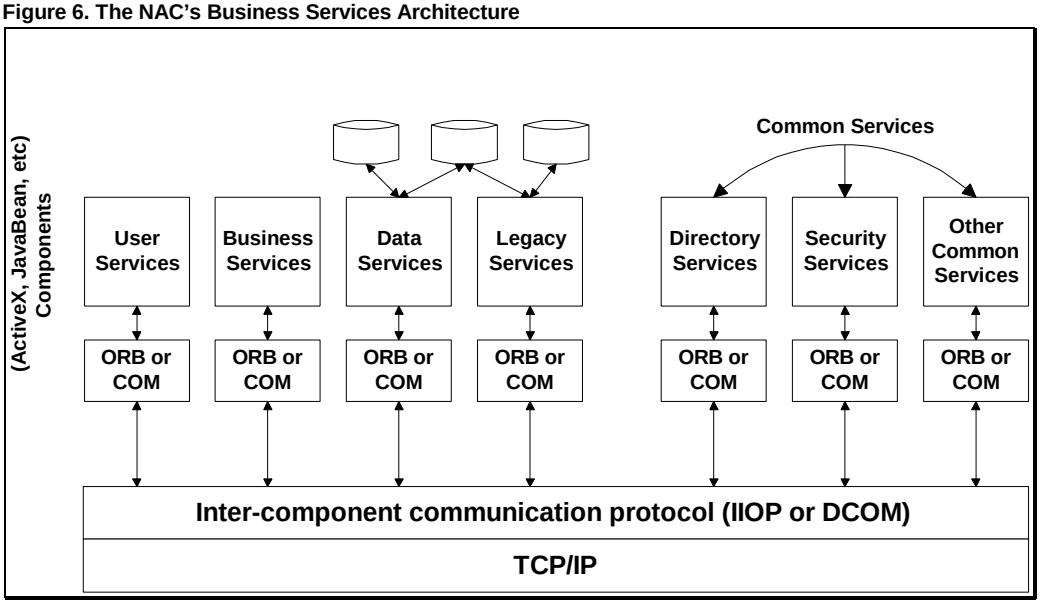
An enterprise view of IT can only result from keeping the focus on the business needs that IT must serve. One key long-term goal of an enterprise IT architecture and its associated application development processes is to enable business experts — not just technologists — to assemble the applications that they need. Building on the concepts and chief benefits of the architectures and models presented so far, the NAC's business services architecture (BSA) can provide an organization with the flexibility and ease-of-use needed to rapidly assemble, customize, and deploy new business applications with this focus. The NAC's BSA is a conceptual construct that provides:

- ***Common network services*** infrastructure that enables a seamless distributed environment with the scalability and robustness required of enterprise applications. For the foreseeable future, these services will need to support both component-based and non-component versions of other services.
- ***Multi-tier, service-oriented architecture***, in which each service provides a single function that is available exclusively through this service, and the component-based interface to each service is well-documented and widely published[21]. Service-oriented architectures are based on established design principles, including:
  - Black box (information hiding) design metaphor, including the two related design principles, modularity and encapsulation, applied to inter-system, application system, application program, service, and component levels as appropriate.
  - Partitioned application structure, in which the design of each partition or logical tier may be de-coupled from the design of the other tiers

The design of business services is completely de-coupled from the design of the other services. The business services architecture abstracts functions that comprise an application and isolates them from each other so that each can each be used effectively over the long term. Business services must be accessible to new client interfaces and must be able to shield users from behind-the-scenes implementation details such as disparate databases, legacy systems, directory and security services and the like.

[21] Although service implementations may be procedural, object-based, or component-based, it is assumed for this discussion that they are component-based, and that the interfaces are documented and published in the Component Repository for use by developers, and eventually business users, who assemble and customize applications. The interfaces also must be published via the Directory for run-time use, including the description of the particular service being "documented" as attributes, so that they can be dynamically discovered based on their capabilities and proximity, for example, rather than based on a particular name of a service instance.

Figure 6. The NAC's Business Services Architecture



NAC's business services architecture partitions applications into five constituent elements.[22] The goal of this partitioning is to ensure that any business service will be re-usable by the other constituents, including other business services as required, both internal and external to the organization.

- **User services** enable a user to interact with the business services, based on a particular view of the business service. For example, administrators, internal business users, external trading partners, and customers will all have different views of the same business service, with different presentation rules and different functionality available to them.

  User services are often thin, forms-based views tailored to the particular type of user. User services may be implemented as HTML or Java components to run in a universal browser, or they may be implemented as platform specific ActiveX components using Visual Basic, or platform specific variations of HTML and Java, depending on the requirements of the particular application and user community. Consistent with the NAC's definition of tier as a logical construct, it's important to note that user services may reside on the client machine or the business server, with only a presentation component on the client. For example, an ActiveX, HTML, or Java-based order entry form could be downloaded to the client for display in a Web browser.

[22] The full model may not be applicable to every organization. For example, some organizations may not have legacy access and integration issues, or may choose not to address them in this way. Other organizations may choose, for whatever reason, not to partition data services as a separate logical tier. The specifics are for example purposes only, and regardless of the nomenclature used in this version — data services vs. persistence services, for example — the concepts of partitioning, encapsulation, and modularity are the important elements of this discussion.

- **Business services** implement the business rules and processes that define the particular business model. Business services may be distributed across multiple business servers (which in the three-tier client/server model were referred to as the "middle tier"), as dictated by accessibility, availability, scalability, and other specific technical requirements.

  Each business service represents a single business function that is available exclusively through this service, although it may be implemented as multiple components. For example, a service such as "Product Ordering" may resolve into a series of operations involving several different components, such as customer validation, order validation, inventory update, shipping, and billing.

- **Data services** are an abstraction layer that maps business objects to the particular database schema that supports them in their stored (persistent) form. Advanced data services may support the mapping of a common business data object to multiple data stores; for example, a customer object, part of which is stored in DB2, part in Oracle, and part stored commonly in both.

  - **Databases** are the particular database management system (DBMS) instances, such as flat-file databases, relational databases, object-relational databases, object-oriented databases, or other types of data stores that are considered strategic.

- **Legacy services** provide component-based interfaces to legacy applications and data that must be accessed as-is. Legacy services can be implemented as a component-based interface "wrapper"[23] on the legacy system itself, or on a separate tier that accesses the legacy system via a gateway. The legacy services provide abstract business object interfaces to the legacy business services and data. They may be accessed from either the Business Services tier or the Data Services tier[24], or both, depending on the particular design.

  - **Databases** in this context are the particular legacy database instances, such as flat-file databases, hierarchical databases, and relational databases, that are accessed from the legacy services layer through a legacy application, such as one that is CICS or IMS based, or through legacy I/O subroutines.

- **Common services**, including directory, security, and other services made available to any of the service partitions that require them. For example, any of the other tiers (user services, business services, data services, and legacy services) must be able to:

---

[23] "A wrapper is a layer of software that provides a new interface to the program around which it is wrapped. The purpose of a wrapper is to make the underlying program accessible to an otherwise incompatible external requesting program." R. Shulte. Gartner Group SSA Research Note. "Clarifying Wrappers and Message Brokers. 10/17/97.

[24] One member company is currently prototyping data services and legacy access alternatives with the intention of making legacy customer data available as encapsulated COM or CORBA data objects via the data services tier. However, this is still a prototype, not a proven design.

- Identify themselves to the **security service** and gain an authentication token that validates their identity[25]
- Obtain from the **security service** the authorization credentials for the particular user or role/group name upon whose behalf access is requested, so that access rights can be verified (using another security service)
- Use the **directory service** to locate the service component needed for the task at hand (anything ranging from the core common services such as authentication and authorization, or a higher-level business service component (check the customer's credit history, for example)

Although it may appear obvious in the above examples, the NAC wants to emphasize that seamless and manageable implementation of this will require authentication tokens, authorization credentials, component naming, and name-to-location binding that can work from end-to-end, across the range of operating systems, network operating systems, and platforms. In other words, all authentication tokens, authorization credentials, component naming, and name-to-location binding mechanisms must be interoperable.

The goal of this service partitioning is to ensure that business services can evolve and be reused over their lifetimes, and can be made accessible to *any user services as required*, both internal and external to the organization.[26] Equally important, long term, is to design data services as an independent entity that can evolve on its own technology curve, be reused over its lifetime, and be made accessible to *any business services as required*. In addition, *common network services* functions are isolated as separate logical partitions, which makes it easier to "plug in" new implementations of these common services as necessary, without requiring redesign or rewriting of the services that use them. Thus, services designed and implemented in this model can be evolved, reused, deployed, and made accessible where and how required over their lifetime.

[25] Used in mutual authentication procedures between servers to assure each that the other is authentic and not an imposter.

26 Note that this may require some additional inter-object communication and firewall infrastructure, not discussed here, to provide for the secure routing of business services requests/responses to and from clients outside the firewall.

Enabling Component-based Application Development

The full benefit of component-based applications won't be achievable in the short term due to many factors, particularly the lack of interoperable component models and lack of integration with existing and evolving common network services. Nonetheless, organizations can lay the groundwork today in several key areas to ensure that they will be in a position to effectively implement and integrate component-based frameworks, tools, and technologies when they become available. The critical success factors, discussed in this section, include:

- **Cultural and organizational issues**: The organization must be inculcated with the fundamental concept that business applications, especially the core business logic and business data services, are key corporate assets whose useful life must be leveraged and extended by re-use.
- **Process issues**: Application development processes must be re-engineered to support a Business Services Architecture (a component-based, multi-tier, service-oriented distributed application architecture built on a common services foundation). Application assembly and customization processes must be distinguished from component development processes, with roles and responsibilities clearly delineated. In addition, a technical services organization must be integrated into the organization.
- **Technical issues**: There are many technical issues that must be resolved by vendors and implementers before successful migration to component-based application development can occur. The existing and evolving common services infrastructure must support both component- and non-component-based applications. In addition, organizations will need interoperable modeling tools and repositories.

Cultural Issues: Business Logic as a Corporate Asset

The NAC's development of the Business Services Architecture was predicated on the fundamental concept that business logic and business data should all be treated as assets, just like the other assets of the organization that are accounted for on the balance sheet. When an organization views business logic as an asset to be leveraged over the long term, it will be more inclined to move toward a service-oriented architecture, in which business logic is de-coupled from client applications, from databases, and from data access mechanisms.

Recent changes to federal accounting procedures (which will also be adopted in the public sector as mandated by the SEC for publicly held corporations) will reflect this philosophical shift, in black-and-white on the organization's balance sheet. Specifically, beginning in 1999, the cost of internally-developed software will appear on the balance sheet as an asset along with the other property, plant, and equipment

(PP&E) on the corporate balance sheet, to be amortized over a period of years of expected useful life[27].

Accounting principles aside, the NAC believes that a psychological, cultural shift in consciousness must also occur within the development organization, from the top ranks down to entry-level programmers. An organization that doesn't treat its information systems, development activities, and business applications as assets won't be psychologically prepared to migrate to the business services architecture, nor will it be in a position to effectively implement component-based application development practices.

Process Issues

Application development processes must be re-engineered to support a Business Services Architecture (a component-based, multi-tier, service-oriented, distributed application architecture built on a common services foundation). Application assembly and customization processes must be distinguished from component development processes, with roles and responsibilities clearly delineated. In addition, a technical services organization must be integrated into the organization. These issues are discussed briefly.

Re-engineering the Application Development Organization and Its Processes

In the days of centralized mainframe computing, the application development process was rigid, process-centric, and based on limited standards. With the advent of the desktop PC, heterogeneous networking, and client/server computing, much of that rigidity has given way to autonomy and individualism. One result of this has been that the typical development team has too broad a set of responsibilities — configuration management, testing, training, and documentation, for example, in addition to its core development activities. A related outcome is that — despite component-based and object-oriented design processes, development techniques, and software tools — organizations are still not gaining a great deal of re-use or leverage from re-usable code (if that weren't the case, we'd have one less reason to write this paper).

---

[27] Discussed in FASAB (Federal Accounting Standards Advisory Board) Statement of Recommended Accounting Standards Exposure Draft "Accounting for Internal Use Software." This standard will amend standards for software accounting contained in SFFAS #6. Previously, under the original SFFAS #6, "Accounting for Property, Plant, and Equipment," the cost of internally developed internal-use software was prohibited from being capitalized unless management intended to recover the costs through charge-backs. In addition, a technical feasibility study was required prior to capitalizing any costs. Once capitalized, the costs could only be amortized over a period longer than five years. Maintenance-style costs could not be included.

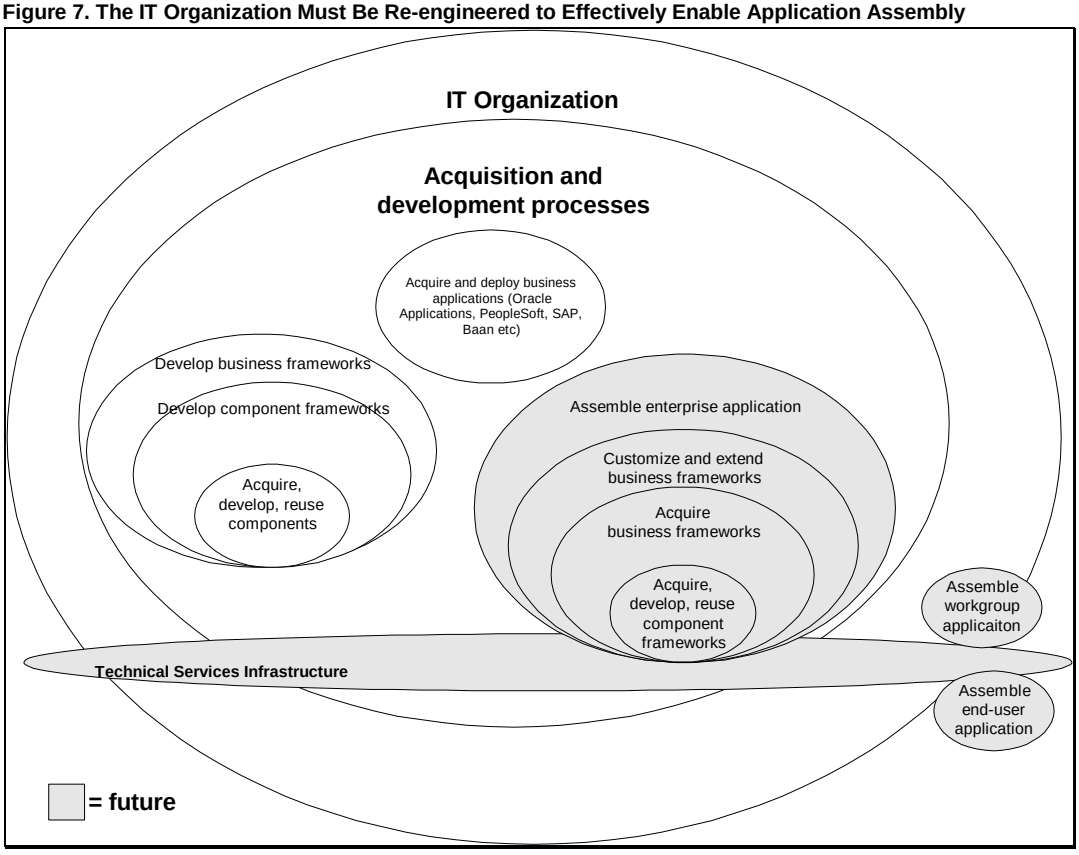
The NAC believes that organizations that attempt to develop component-based business services applications *without providing the appropriate development processes and infrastructure* will only create more non-reusable code. Thus, the IT organization will likely need to be re-engineered to provide a smarter division of labor. Development teams need to be able to focus strictly on development tasks (described in more detail below). The goal of the re-engineering effort is to put in place "just enough" process to provide the rigor required for the particular type of application development project, without stifling the creativity and productivity of the core development teams.

An important corollary goal of re-engineering the application development organization is to prepare to implement enterprise business applications based on interoperable frameworks, as they become available. Component *frameworks* are pre-built, partially assembled component sets that provide building blocks for application specific services, and transparent interfaces to whichever implementation of standards-based infrastructure services match the customer's environment. Such business services frameworks may potentially offer the organization the ability to prepare ready-to-deploy enterprise application suites from a combination of customized, pre-built frameworks and components, and application-unique components. (The business frameworks emerging as the result of IBM's "San Francisco" project[28] are one example.)

Both of these goals will require a new type of organization within IT, the *technical services organization.* The technical services group should provide the support-level tasks that would obscure the developers' focus and dilute their efforts. This group is an absolute requirement if users outside of the application development organization — the line manager, controller, administrator, and other business users — are to be enabled in their efforts to assemble applications from pre-fabricated components. Without such a technical services support group, end-users will again turn to IT for basic application development needs. (More about the technical services organization below.) The re-engineered application development organization might look something like that shown in the figure below.

[28] The San Francisco Project (http://www.ibm.com/Java/Sanfranciso) is an IBM initiative with over 200 application development companies producing server-side core business process components — rather than client-side components — that can be reused as a base for creating applications for specific industry domains.

Figure 7. The IT Organization Must Be Re-engineered to Effectively Enable Application Assembly



With effective division of labor in mind, the application development activities themselves can be partitioned into two key functional areas, the first of which requires less technical expertise than the last:

- Application Assembly and Development Process
- Component Development Process

Application Assembly

Application assembly is the process in which applications are assembled from pre-fabricated components. This process will also include developing any application-unique components or services. When the repository and other tools are available (see the *Technical Issues* section), the application assembly process will expand beyond the realm of the IT organization to encompass business experts. Accountants, marketing and sales staff, customer service reps, line managers, and other business users who will assemble their own applications to provide business function. For example, presuming that a Controller has the appropriate role-based access rights to the component repository, he could find the Credit_Limit component in the corporate repository and change the "check credit history" parameter from $5,000 to $2,500, and re-apply the component to the Business Service, all on his own.

This concept is shown in the figure above, as the ellipse labeled "Assemble End-User Application," which you'll notice extends outside the realm of the IT Organization. In the future, the activity known as "Application Assembly and Development" will likely evolve into Application Assembly and Customization, with more of the activity being performed outside the realm of IT.

Component Development

The component development process requires different skills and greater rigor in some areas than the application assembly process. Component developers will create sharable and reusable components from object-oriented programming languages and other technologies in which they are highly skilled. In addition, some developers may be called upon to acquire (internal or external), evaluate, modify as needed, and make available to the organization-at-large both component frameworks and components.

In addition, this group will evaluate, modify, and implement business frameworks, such as the frameworks emerging from the San Francisco project. Vendors are now in the process of developing applications based on the first San Francisco framework, for general ledger applications. When such frameworks become available, organizations will need developers who are technically proficient at modifying the frameworks appropriately to provide application-unique and organization-unique functions.

The component development process becomes the source of the reusable frameworks and components for the application assembly process, above — for example, developing a Credit-Limit component that will be placed in a central repository, for use by others throughout the organization.

The Technical Services Organization

Just as applications must be appropriately partitioned, application development roles[29] and responsibilities must also be appropriately divided. Re-engineering the application development process should include an analysis to determine which roles and responsibilities are best fulfilled by the core project teams and which are best fulfilled from outside the team. Again, keep in mind that the core project teams must be able to focus on the skills and tasks required to assemble and customize component-based applications, and roles that would dilute these skills should be provided from a technical services support organization[30].

---

[29] Roles do not necessarily equate to individuals, since one person may play several roles.

[30] Examples of roles that might be provided by a support organization include application architect, configuration manager, database administrator, LAN administrator, OO methodologist, process methodologist, quality assurance analyst, release engineer, repository administrator, reuse architect,

The technical services organization should have the skills necessary to implement and manage component frameworks and repositories, as well as the other support roles alluded to above. In addition, *the technical services organization should serve as the agent of change for technology, process, and culture within the larger organization* by:

- Creating common tools, architectures, frameworks, components, guidelines, and processes for use across the organization
- Enabling development groups to successfully adopt these through a mentoring process
- Modeling the desired culture in its own actions and structure.

Some of the key tools upon which the technical services infrastructure will depend are still emerging. Most important of these are modeling tools, based on the Unified Modeling Language (UML), and component repositories, as discussed briefly in the next section.

Technical Issues

There are many technical issues that must be resolved by vendors and implementers before successful migration to component-based application development can occur. The existing and evolving common services infrastructure must support both component-based and non-component-based applications. In addition, organizations will need interoperable modeling tools, repositories, and application assembly and customization tools for business users.

Interoperability Among Tools and Repositories

Before developing an application, the business problem must be effectively modeled to fully understand its dimensions and scope. Visual modeling tools provide software engineers and business experts with a graphical means of modeling business problems to enable a more complete understanding of requirements. NAC applauds the growing number of vendors, including Microsoft, Rational, and JavaSoft, that are adopting the Unified Modeling Language (UML) as the foundation for their modeling tool implementations. We believe the UML and supporting tools and techniques[31] represent

---

security analyst, technical writer, and test engineer. This is a partial list from one member organization based on their use of the LBMS tool and the Evolutionary Delivery process.

[31] Use case analysis and other tools and techniques for analyzing the business problem, producing a logical design of a service-oriented business object model, and mapping it to a decoupled physical design that matches the service partitioning requirements of the particular application.

the current best hope for improved analysis and design interoperability across service partitions, and integration with CASE tools.

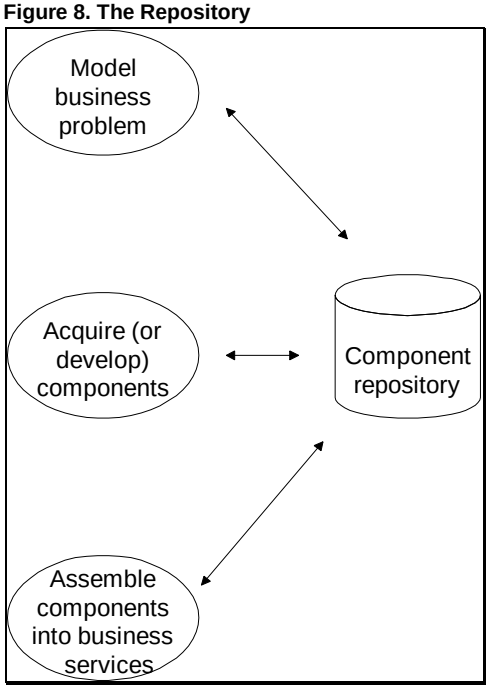
"The Unified Modeling Language, or UML, is a third-generation object-oriented modeling language. It adapts and extends the published works of Grady Booch, Jim Rumbaugh, and Ivar Jacobson, and contains improvements and suggestions made by dozens of others. The UML is being presented to the Object Management Group in the hope that it will become a standard modeling language for object-oriented development. Because the UML is meant to be applicable to the modeling of all types of systems, it applies equally well to real-time systems, client/server, and other kinds of "standard" software applications. It provides a rich set of notations and promises to be supported by all major CASE tool vendors."[32]

Repository

Components can ease the application development process, but only if every programmer in an organization can find out about them and use them according to the needs of the specific development project. In many organizations, the current "process" for discovering existing components and learning about how to use them depends on inter-personal "networking" skills rather than technical acumen. Organizations need central or distributed, synchronized repositories that are available to varying degrees, depending upon the role of the user. For example, the controller should have access to the accounting related components in the repository and be able to modify the business rules – tax rate tables, shipping charges, or prices, for example – for these components.

[32] From the Introduction to "Unified Modeling Language for Real-Time Systems Design," available at http://www.rational.com

Figure 8. The Repository

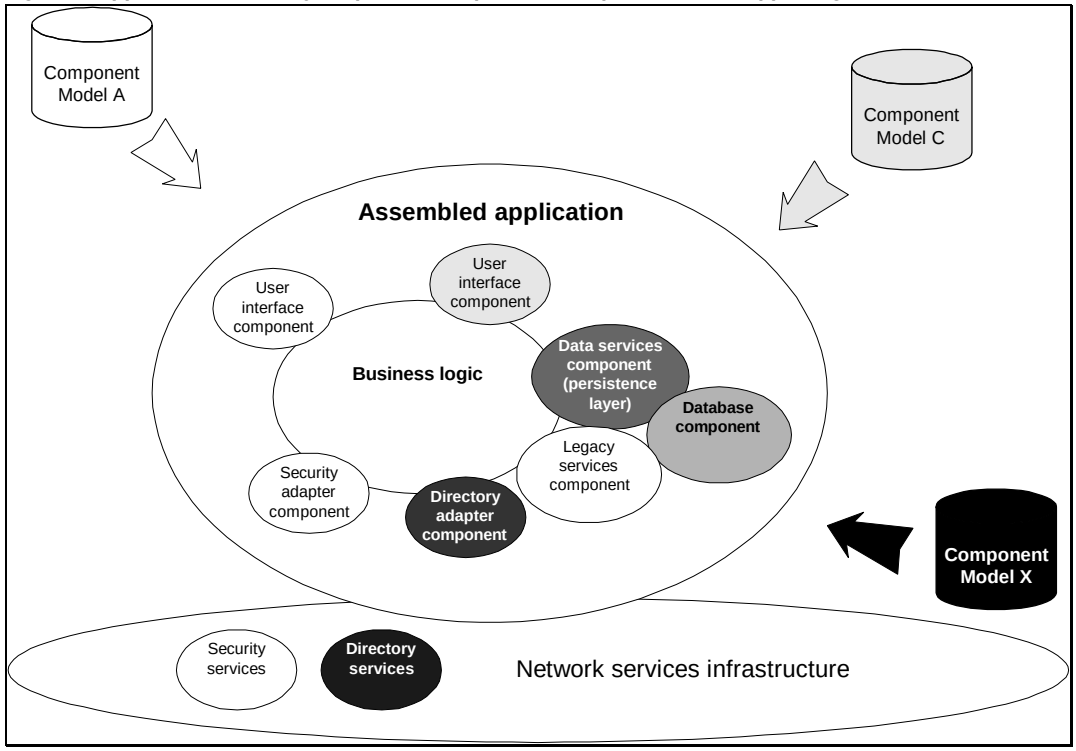


Ideally, a corporate repository should allow a component to be checked out, modified or updated, and then checked back into the repository, with all applications that use the component updated automatically. Information about all application components – the components that implement the services defined by the BSA – should be available and able to be browsed by authorized users. In the long term, "authorized users" should include the business experts who can best define the application required to solve a particular business problem.

However, the tools that would enable business users to browse a component repository and assemble or modify an application don't exist yet, and the requirements for such tools have yet to be defined. That's why in the near term, these activities will continue to require the expertise of software engineers and a team of developers that includes skills in each of the affected service partitions.

Finally, modeling tools and repositories must be interoperable across the partitions of the business services architecture. That is, a user service developed in one component model using a particular language and AD tool set should be able to interoperate with the other services —business services, data services, or legacy services —which have been developed in a different model using different languages and tools, without losing end-to-end interoperability.

Figure 9. Application Assembly Requires Interoperable Components and Supporting Infrastructure



Regardless of the component model used to create a particular component, that component should be able to be used with components from other models to create a functioning application or complete "business service."

One promising effort to address this need is the Object Management Group's specification for a CORBA-based Meta Object Facility (MOF). This is intended to define a general, vendor-neutral facility for sharing information about meta-models, components and data at a semantic level.[33] The notational language of UML is used throughout the specification.

Integration with Common Services Infrastructure

One significant technical issue for component-based application development is inadequate integration with the common services infrastructure. One example of successful integration would be that an enterprise's investment in a particular infrastructure service, say, a corporate directory, would not have to be replaced by a new product to take advantage of the component packages and services that vendors offer. Those packages should be able to exchange information and participate in business processes with existing infrastructure at the back-end. Examples of poor

[33] See the following page of the OMG website for the MOF specification: http://www.omg.org/library/schedule/Technology_Adoptions.htm#MOF_Specification

integration abound. To take one, many universities have a significant investment in Kerberos-based security, yet the credentials from that security service will not transparently translate into permissions in an NT 5.0 environment.

Directory Services

Over the past three years, vendors have begun to converge in their approaches to directory services. In fact, thanks to ground-breaking discussions at a NAC /Burton Group Conference just two years ago, LDAP (lightweight directory access protocol) has been adopted as a client interface to directory services by virtually all vendors. Some interoperability issues still exist, as the NAC will discuss in an updated directory services paper in the future, but this coalescing on this one point is good news.

The NAC defines a full-scale directory service as one which supports a naming model, can store attributes about each entity, and can support lookup by entity as well as by name. Aspects of the OSI X.500 standard have been the reference point for many products currently on the market. Such features are embodied in the directory services of NetWare's NDS, Microsoft's Active Directory, and Banyan's StreetTalk. (Microsoft's Windows NT Server 4.0 does not support this model.) Although the CORBA Services specification includes these capabilities in its Naming and Trader services specifications, no CORBA-compliant product yet provides a generic, full-featured directory service of this kind.

One possible path forward would be to build on the Java Naming and Directory Interface (JNDI), but out of the box, JNDI is essentially a standard API approach, not a component-based service with interface definition language (IDL) interfaces in the CORBA style. Certainly JNDI could be given an IDL-based wrapper without a great deal of effort. However, there is more to getting directory services properly modeled as true distributed components than such mechanical API-wrapping. This is another example of inadequate integration between existing technology and component-based approaches.

Microsoft's ADSI (Active Directory Service Interface) provides a set of COM interfaces a layer up from underlying directories. It is "built on" LDAP and Microsoft pledges smooth two-way interoperability with other LDAP-compliant servers via ADSI interfaces.

Directory services require security services for appropriate role-based control over access rights to information. ADSI security will be integrated with the NT 5.0 security model. Active Directory and Microsoft's LDAP client use Microsoft's SSPI (Security Support Provider Interface) as a way to provide security technology independence.

Microsoft Windows NT 5.0 will include SSPI providers to multiple security approaches based on SSL 3.0, Kerberos v5, and Windows NT LanManager (NTLM).

It thus appears that directory service interoperability will hinge on LDAP-based communication at least until more complete CORBA directory services appear. The Microsoft approach still seems to imply tight, OS-level ties between security and directory services. This at face value seems to move away from the notion of treating such services as just another set of components and will likely be the source of more or less subtle, complex interoperability hurdles for those trying to integrate COM and CORBA.

Security Services (Authentication, Authorization)

Many vendors have implemented the same protocols or APIs for their security implementations (our discussion is limited to Identification, Authentication, and Authorization).

GSS-API (Generic Security Services-Application Programming Interface) which can be implemented independently of the underlying authentication protocol, is used in many products. The Kerberos[34] authentication protocol has likewise been widely adopted. Supporting mechanisms include cryptographic algorithms for encrypting network messages (to secure them from eavesdropping or packet sniffing), such as SSL (Secure Sockets Layer).

The OMG's Security services are defined in "CORBAservices: Common Object Services Specification." However, the only product on the market today that has in fact implemented true CORBA security service is ICL's DAIS: in major commercial CORBA implementations (for example, IONA's Orbix and Visigenic's Visibroker), security services to date have been implemented with a mix of GSSAPI-wrapped Kerberos and SSL approaches.

Microsoft's approach as of Windows NT 5.0, will also be based on a PKI-enhanced Kerberos 5 model.

Unfortunately, these high-level similarities do not provide transparent interoperability between CORBA and COM security services. For example, *a user who has been granted authenticated credentials in one environment cannot use those same credentials to gain access to resources in the other environment.* However, from NAC's perspective, this is precisely what needs to happen.

---

[34] Kerberos, a cryptographic authentication protocol that was devised as part of MIT's project Athena (an experimental distributed computing environment begun in the early 1980s at MIT in conjunction with Digital Equipment and IBM), is a widely used authentication service. The protocol has been implemented by several vendors, including CyberSafe, and has also been adopted by the OSF for its DCE, although the DCE version of Kerberos is at this point not compatible with MIT Kerberos; implementations include both versions 4 and 5.

Interoperability Challenges of Heterogeneous IT Environments

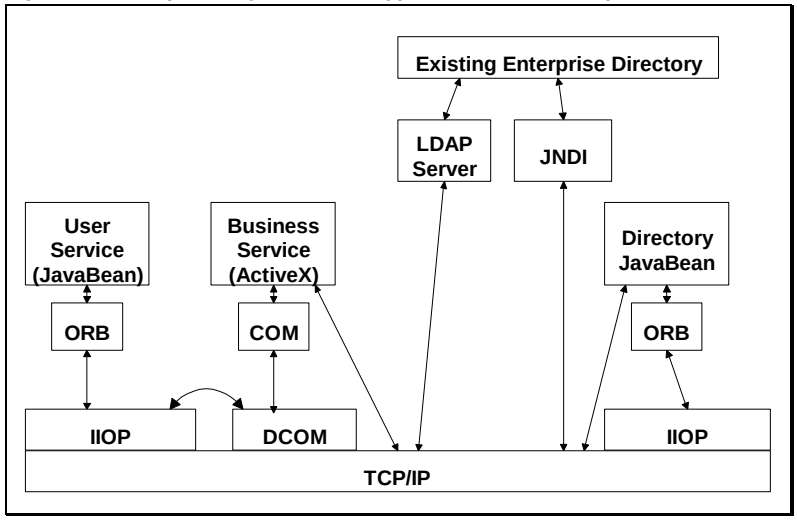
Most NAC member organizations will need to support both CORBA/IIOP/JavaBeans and COM/DCOM/ActiveX for the foreseeable future. Why? First, these organizations cannot fully control the IT environments inside a given organization today. For example, different business units may rely on different platforms for valid business reasons, and mergers and acquisitions often lead to strange, hybrid IT environments. Second, NAC member organizations increasingly want to provide access to internal IT services externally, to their customers, clients and partners. NAC members are not in control of the client platforms from which such connections will come. Support for both distributed component models is the only viable approach.

Furthermore, although implementation of component technology across the application portfolio may be a long-term goal, enterprises will not be able to (or even want to) migrate all services and applications to a component model in the short to medium term. Nor can vendors deliver component-based versions of the full suite of common network services that an enterprise might need in this time frame. Thus, organizations are faced with the additional challenge of developing interoperable applications in an architectural environment that supports both component-based and non-component-based applications.

The NAC member organizations must be able to mix and match ActiveX and JavaBeans components, and they must be able to provide access paths to both component and non-component-based common services. This can get messy.

As the hypothetical example in the figure below shows, an enterprise may query an existing corporate directory via an LDAP API call over TCP/IP when executing a server-side component-based business service using COM/DCOM/ActiveX and a client-side component-based user service in the CORBA/IIOP/JavaBeans model. That enterprise may also provide component-based access to the same directory via an IDL interface of a JavaBean that uses JNDI at the back end for directory communication.

Figure 10. Interoperability Paths in a Hypothetical Multi-Component-Model Environment



There are interoperability "gotchas" waiting at almost every connection point in this example. JavaBeans and ActiveX components require the presence of a bridge to interoperate (a variety of such bridges are available from IONA, Expersoft, Sun and others). Java virtual machines (JVM) differ from one client platform to another, wreaking havoc with a BSA component that expects a JDK 1.1-compliant JVM, and finds itself running under Microsoft Internet Explorer. Sun's Activator is a recently announced technology that can negotiate this interoperability barrier by allowing an applet to request the needed JDK support regardless of the JVM. These examples only hint at the interoperability problems that can arise. Few companies can afford the full breadth of expertise on relevant problems and solutions. This limitation is behind the recommendation to members on page 39 that they leverage their expertise and share their information on such matters.

On one hand, NAC members *will* benefit from moving to a common services model — de-coupling directory services from the multitude of particular applications that use them, for example—even in the absence of component-based versions of those common services. On the other hand, component-based services are crucial to the arrival of true "drag and drop" application assembly from components. Without component-based services, application development will continue to require highly skilled, scarce (read: expensive) IT talent to handcraft key linkages between applications and the common services infrastructure. The presence of such handcrafted code means that the application will tend to be brittle — unable to survive external IT changes without redesign and re-coding.

Manageability and Distributed Transaction Processing Monitors

The component-based Business Services Architecture will fail to deliver on its promises unless it is paired with run-time tools that can dynamically manage and scale business applications on the fly. The latest class of distributed transaction processing (DTP) monitors offers the first serious candidates for filling that role.[35] The fact that these kinds of component-based services are just beginning to get the attention they deserve is one of the primary reasons the NAC recommends deferring the use of component-based development for high-volume, mission-critical applications.

[35] See the recent reviews of several DTP monitors in *Network Computing Online*, http://techweb.cmp.com/nc/901/901ws1.html and http://techweb.cmp.com/nc/820/820r1.html

Conclusion and Recommendations

Today's IT organization is scrambling fast to deliver flexible, full-featured applications that help the organization meet and exceed its business goals in an evolving, global marketplace in which constant change is the only constant. Component-based application development has gained market share on the desktop, and is fast gaining mind share as the means that will provide this flexibility in the realm of server-based business applications. The long-term goal is that business users rather than software engineers will be able to assemble the applications they need from easy-to-use "software Legos.™"

The ability to create applications from prefabricated software components has the potential to contain (and even reduce) development, deployment, and maintenance costs throughout an application's life-cycle, **but only if migration to component-based applications leverages existing (and evolving) common network services**. NAC's Business Services Architecture starts from this premise, incorporating key features of multi-tier, service-oriented architectures, component-based application development models, distributed computing models, and the Burton Group's Network Services Model.

The NAC's BSA is designed to provide a flexible foundation while enabling organizations to leverage business rules, business processes, and business data. This paper has outlined the key concepts that have gone into this conceptual structure, with special focus on the underlying common network services model. In addition, we've presented many steps that organizations can take to ensure that they'll be ready to take advantage of the pre-fabricated component suites as they emerge in the marketplace. There is still much to be done, however, before this will become reality. We conclude with some recommendations to vendors and the NAC member organizations.

Recommendations

Although the two models, COM/DCOM/ActiveX and CORBA/IIOP/JavaBeans, are conceptually similar in many respects, components built for one model cannot simply be moved into the other model. Using gateways and bridges, client components from one model can interoperate with components from the other model. Bi-directional server side bridges between COM and CORBA are also emerging. However, as we've just stated above, an unresolved issue for NAC members is how well the component models interoperate at the back end, at the network services level, where the directory and security services exist. The NAC's recommendations to vendors are driven by these key requirements.

Recommendations to Vendors

In general, NAC encourages vendors to honor the best contributions of their competitors by *matching features* and by *defining interoperability paths between products*.

One oft-mentioned downside of supporting heterogeneous IT environments is that the developer is restricted to the "lowest common denominator" of features in the range of products being supported. NAC members have no choice but to work in heterogeneous environments. Thus we urge vendors to be aware of the marketplace in which their products play, and to raise the lowest common denominator by including support for key features of competitive products. It may look like good marketing strategy to do otherwise, but it burdens NAC companies with significant costs — with no additional value.

The NAC also urges vendors to work together as appropriate to solve mutual technological problems, and share information about solutions to benefit the industry at large. A good example of this type of effort is adoption of Lotus' InfoBus technology into the JavaBeans component architecture. If your company does not provide a particular bridging or other needed interoperability technology, cooperate fully with companies that do by providing all necessary information for their product development efforts. Microsoft's recent licensing of COM to IONA for use in the Orbix product line is an example: while Microsoft may not do CORBA, that doesn't mean that CORBA vendors can't do COM.

Other specific recommendations that will help NAC member companies move towards a component-based Business Services Architecture are:

- Provide user services component frameworks that make it easy to support the universal thin client[36], with zero configuration and deployment, because this is becoming a pervasive requirement for access by business partners, customers, and suppliers. Allow the user to choose to perform more processing on the server (the "thin" client model) or on the client (for ostensibly faster performance).
- Provide business services component frameworks that will seamlessly support the universal thin client, regardless of which language or component model was used for the user services, and regardless of whether the clients are accessing the business service from inside or outside the organization's firewall.
- Provide data services component frameworks that seamlessly support both COM- and CORBA-based interfaces to persistent business objects, including caching and object-to-database schema mapping for one or more databases.

[36] The NAC's definition of the "universal thin client" today includes these core elements: web browser, a Java Virtual Machine, and support for distributed component computing. There should be no other conditions, such as platform or operating system dependencies. The universal thin client is shorthand for a conservative assumption about what is on the desktops of all those to whom we would like to extend our business services.

- Ensure that performance, robustness, and manageability of both COM- and CORBA-based distributed components, and interoperability bridges between them, are addressed up front and in the details of optimized implementations, because this will be critical to their success in the enterprise.
- Provide component frameworks that support seamless interfaces to whichever implementation of standards-based common network services match the customer's environment. End-to-end support of role based authorization credentials obtained under a single sign-on, in conjunction with mapping of external users to a particular role based on certification by some acknowledged certificate authority, and end-to-end component/service location transparency based on common directory services, are just three examples of key requirements.
- Provide enterprise-class tools to partition and model business problems effectively, and enterprise-class repositories, both with an eye to deploying applications in a distributed, component-based, service-oriented architecture. "Enterprise-class" means robust, scaleable, common-services enabled, and usable across service partitions and organizations. As one example of common-services enabled, all tools should support common, role-based authorization credentials for controlling access to business object models, component repositories, and other sensitive information.
- Provide component-based, business-framework-based, enterprise application suites that will interoperate with each other, and with the component-based services and applications we have developed for ourselves. The IBM San Francisco project embodies many of the framework concepts that NAC would like to see implemented, particularly with the evolution of the foundation layer to a common CORBA/Java based distributed component model. However, at the moment, it appears to fall short in the area of providing seamless interfaces to standards-based common network services of the customer's choice.
- Provide server-side support for distributed transaction processing (DTP) through TP monitors that can work with a wide range of underlying technologies. Note that this recommendation is driven more by the need for scalability and manageability of high volume transactional applications than by the narrower issue of maintaining transactional integrity. As one recent study put it, "In reality, only between five and ten percent of the code in TP monitors is about synchronizing transactions."[37] This is an area of rapid evolution and market churn at this point, as TP Monitors evolve to incorporate the distributed component (ORB/COM) technologies and messaging oriented middleware (MOM) technologies, on their way to becoming what Gartner Group calls Object Transaction Monitors (OTMs).

[37] Jeri Edwards with Deborah Devoe, 3-*Tier Client/Server At Work.* John Wiley and Sons, 1997, p. 20

Recommendations to NAC Member Organizations

The end goal — application assembly by business experts using interoperable, easy-to-leverage components and component frameworks that integrate with existing and evolving network services infrastructure — will not be achievable in the short term. However, although still immature, component-based development of multi-tier, server based applications is beginning to enter the mainstream[38]. It's not too early to begin limited development of multi-tier applications based on the business services architecture, as long as the requirements for application robustness and scalability are not too taxing[39]. While doing so, keep in mind the key characteristics of the architecture.

- Intelligently partitioned strategic applications, based on the multi-tier architecture model
  - User Services support universal client and flexible role-based user views. For example, internal user, internal systems administrator, external business partner, external supplier, and so on.
  - Business Services implemented as shared services that can be reused by any application. A given business rule or process is defined once, and can easily be changed as required to match the evolving business model.
  - Data Services, supporting abstract business object interfaces to persistent business data, also implemented as shared services that can be reused by any application, without regard to where or how the data is stored
  - Legacy Services, supporting abstract business object interfaces to legacy business services and data, also implemented as shared services that can be reused by any application, without coding to legacy interfaces or being impacted by future migration to strategic interfaces
- Component model transparency, such that service partitions developed using one model (either COM or CORBA) can seamlessly interoperate with service partitions developed using the other model[40]

---

[38] For example, during the period that this paper has been under development, IONA's OrbixOTM (with Orbix for MVS), and Microsoft's Transaction Server 2.0 (with Cedar and Microsoft Message Queue (MSMQ, nee "Falcon") have become generally available on Windows NT. In addition, IBM's Component Broker (with DB2 adapter) has achieved limited availability on Windows NT.

[39] Mission-critical, multi-tier applications that must support hundreds or thousands of users are still developed primarily using transaction processing (TP) monitors. This is likely to be the case for some time to come. For examples, see *3-Tier Client/Server At Work*, by Jeri Edwards, which profiles eight real-world three-tier implementations, seven of which are based on BEA Systems' TUXEDO TP monitor.

[40] For example, COM based business services and CORBA based business services may both need to work seamlessly with the same data services tier (which may itself be either COM or CORBA based)

- Common network infrastructure services, specifically, directory, security, and all the other common network services required to support full-scale, distributed computing.
- Minimal platform dependence

The following recommendations are intended to support these key directions:

- Align the IT strategy with the business goals to the extent possible while anticipating changes that will affect the requirements on applications and infrastructure. Key factors that affect business requirements are:
  - Changing customer expectations
  - New business ventures
  - Deregulation/re-regulation
  - Mergers and acquisitions
- Encourage development organizations to endorse and actively promote the concept that the application services they provide are assets whose useful life can be extended by re-use. Evangelize the concept that business services and data services must be developed and preserved as assets. For example, as a move in this direction, establish software design and engineering guidelines that avoid the hard-coding of business rule and process specifications, data access interfaces, legacy access interfaces, or other connectivity interfaces with the presentation logic that must of necessity reside on the desktop.
- Architect for the future by migrating to a multi-tier business services architecture in which each service partition is isolated and preserved as a long-term capital asset. Start (or continue) application development migration to a network services model, which leverages common infrastructure services to achieve a seamless distributed environment with the scalability and robustness required of enterprise applications.
- Evaluate your current architecture, processes, and infrastructure from the highest level architectural perspective — "architectural" as in "urban planning," not "blueprint for a single house."[41]
- Re-engineer application development processes as needed to effectively support the multi-tier model and component-based application development. Remember that failure to provide the appropriate development processes will result in more non-reusable code.
- Adopt organizational standards appropriate for development environments, application infrastructure, network infrastructure, and corporate cultures in order to facilitate consistency and quality. At a minimum, organizations should develop and promote appropriate guidelines on development.

[41] See "Architecture and Planning for Modern Application Styles." GartnerGroup Systems Software Architectures (SSA) Strategic Analysis Report. R. Schulte. 4/28/97.

- Adopt a flexible methodology that will work in the context of your organization, developers' skillsets, and other organizational factors. There should be enough process to facilitate development, but not so much process that it stifles creativity and impedes development.
- Collaborate with other NAC member organizations to divide and conquer the information challenge we all face in integrating component services with existing infrastructure and in interoperating across disparate pieces of our heterogeneous IT environments. This could take the form of identifying areas of expertise within our companies and putting in place effective ways of leveraging that expertise by somehow sharing the "latest, best available information on solutions to problem X."
- Begin developing the framework archetypes needed to support the business services architecture. From the highest level, there are four basic service framework archetypes — user services, business services, data services, and legacy services — that provide the core functionality needed to position today's enterprise for the future. However, within these four, there are many possible variations, based on the environment and requirements of specific application types, examples of which are shown below:
  - User Services
    - Internal proprietary client vs. external "universal" client
    - COM or CORBA based component model vs. non-component based
    - Interaction type specific (inquiry vs. decision support vs. transactional)
    - Business domain specific, for example, a user view of a framework for building financial models
  - Business Services
    - COM vs. CORBA based component model
    - Interaction type specific (inquiry vs. decision support vs. transactional)
    - Business-domain specific, such as "Business Services View of A Framework for Building Financial Models"
  - Data Services
    - COM vs. CORBA based component model
    - Interface architecture based on common business objects[42] vs. interface architecture based on relational access semantics

[42] This type of interface is what is assumed in the Business Services Architecture description of Data Services, as it raises the level of abstraction to be consistent with the business object model that business users can relate to. However, one of the principle challenges associated with this is providing a performance-efficient object-to-relational schema mapping. Persistence Software, Inc., and Object Design, Inc., are two of the vendors addressing this in different ways.

- Relational SQL interface (ODBC, JDBC[43]) vs. relational data objects interface (ADO, RDO[44]) vs. native RDBMS interface (OCI[45])
- Types of supported data stores to be mapped to, such as flat file database, relational database, object-relational database, object database[46]

- Legacy Services
  - COM vs. CORBA based component model interface wrappers
  - Interface wrappers implemented on the legacy system and accessed via COM or CORBA based backbone vs. those implemented on the middle tier and mapped to a gateway interface to the legacy system
  - Screen scraping transaction interface vs. programmatic transaction interface vs. legacy data interface

Heterogeneous organizations should consider the notion of a parallel prototype path in which key service framework prototypes are developed using both COM and CORBA based component models.

[43] Open Database Connectivity. Java Database Connectivity.

[44] Active Data Objects. Remote Data Objects.

[45] Oracle Call Interface.

[46] The principal advantage of an object database (from a development perspective) is that it eliminates the need to map business objects to the relational (or some other) database schema. It accomplishes this by storing the object relationships directly in the database rather than mapping them to relational tables.

Appendix A. Technology Notes

The NAC defines a ***component*** as ***an executable whose behavior can be customized by an end-user without modifying source code***: "*...a ready-to-run package of code that gets dynamically loaded and linked into your program to extend its functionality. ActiveX controls and Java applets are components in this sense. …components share many of the characteristics of objects, particularly the need to hide their internal workings behind a well-defined interface, that is, a set of access methods…Components need to be sufficiently independent that they can be developed, sold, and installed independently, and yet they need to be interoperable so that they can leverage each other's functionality.*"[47]

A *component model* is the set of rules for creating components that can work together. A *distributed component model* extends the rules to enable components to interact across a network.

The two distributed component computing models discussed in this paper are Microsoft's COM/DCOM/ActiveX and the OMG's CORBA/IIOP, in conjunction with JavaSoft's JavaBeans. On the client side, JavaBeans and Microsoft ActiveX components can interoperate via bridges and migration assistants. Software components that use JavaBeans are thus portable to containers including Internet Explorer, Visual Basic, Microsoft Word and Lotus Notes. The same can be said for ActiveX Controls.

This section includes some background notes about these two approaches.

* * *

COM/DCOM Model, ActiveX Components

COM/DCOM/ActiveX are the labels Microsoft now applies for all technologies formerly known as OLE. OLE evolved from its roots as a document-centric "object linking and embedding" technology to a more full-featured component object model. Microsoft's COM (Component Object Model), is essentially an inter-process communications (IPC) mechanism that enables developers to expose services from one object to requestors (clients) in another object through interface calls. COM finds the object's location in the Windows registry, so developers needn't create hard-coded links to components in source code.

[47] "The Component Enterprise." Byte magazine. May 1997. Dick Pountain.

DCOM (Distributed COM) implements COM over RPC (remote procedure call), thereby distributed functionality. Formerly known as "Network OLE," DCOM enables COM clients and servers to interact remotely, over the network. DCOM is an application-level protocol consisting of extensions layered on the DCE/RPC specification that enables object-oriented remote procedure calls. DCOM defines how calls are made on an object, and how object references are represented, communicated, and maintained.

In simple terms, DCOM is the "wire protocol" for COM based components. As with COM, DCOM clients locate servers through the registry (which holds the IP address of the server containing the requested component).

ActiveX Controls are the re-incarnation of OLE controls (OCX), or COM-based components for the desktop. ActiveX Controls include enhancements specifically designed to facilitate distributing components over networks and to integrate controls into Web browsers. These enhancements include features such as incremental rendering and code signing, which allow users to identify the authors of controls before allowing the controls to run. Functions packaged in an ActiveX control can be used by any container, such as Visual Basic or Web browsers.

You can write directly to the COM model using C++, but popular tools such as Microsoft's Visual Basic and Sybase/Powersoft's PowerBuilder help mask much of the complexity.

Microsoft's ActiveX is part of Microsoft's COM/DCOM architecture, which is integrated in the Windows NT operating system. It will run on any platform that implements the full Win32 API, which is beginning to include more than just Windows 95 and Windows NT®. For example, Software AG (http://www.softwareag.com) released its EntireX/DCOM for Sun's Solaris in September 1997, and is also porting DCOM to 64-bit Digital UNIX , AIX, HP-UX, Linux, and OS/390 (MVS Open edition).

Software AG's implementation of DCOM complies with Microsoft's COM, DCOM libraries, Structured Storage, Monikers, Automation, Uniform Data Transfer, Registry, Service Control Manager, Microsoft RPC with TCP/IP support, MIDL Compiler, ActiveX Template Library, and Windows NT LAN Manager Security.

In addition, most recently Microsoft licensed COM technology to IONA Technologies (http://www.iona.com) which will enable IONA to include COM in its CORBA product line.

CORBA/IIOP Model, JavaBeans+[48] Components

The OMG (Object Management Group) is "an industry consortium whose mission is to define a set of interfaces for interoperable software."[49] The Common Object Request Broker Architecture (CORBA) is the primary specification to emerge from this 750+ plus industry-member consortium. Unlike COM/DCOM, which is at the heart of the Microsoft Windows NT operating system, the CORBA/IIOP is a specification, so it relies on ORB (object request broker) products created by vendors. The revised specification for CORBA 2.0, released in 1994, included some additions that have paved the way for accelerated implementation of ORBs. Two key additions of CORBA 2.0:

- A specification for ORB-to-ORB interoperability. The version that runs over TCP/IP is called IIOP (Internet-InterORB Protocol). This part of the CORBA specification is what can enable ORBs from different vendors to interoperate. The higher-level General Inter-ORB protocol (GIOP) defines a common data representation and a set of request/reply messages that can be mapped onto any connection-oriented transport protocol that meets a minimal set of assumptions. We refer throughout this paper to IIOP, but other mappings of GIOP have been defined. These include, for example, DCE-ESIOP, the DCE Environment-Specific Inter-ORB protocol.
- An interface repository specification. This supports a run-time distributed database of information about all registered component interfaces. Clients can find out how to use components, and component-based services can register themselves and changes in their interfaces through this repository. When paired with the *implementation* repository, the two repositories provide ways for components to be defined, located, browsed and called. These services apply to both the development phase and to the run-time environment.

As its name implies, CORBA is about distributed objects in the true technical sense of the word. The focal point of CORBA's technology for developers is its interface definition language (IDL). A CORBA object is defined and known by its interface. That interface is specified in the IDL language, a stable and complete formal language for defining any and all such object interfaces. This reliance on IDL helps guarantee both language independence and implementation independence for CORBA objects:

Since users of an object see only the IDL interface, they don't know (or have to care about) what language the object was actually coded in or, for example, what algorithms it uses internally. The task of mapping from IDL to actual programming languages is facilitated by formal mappings (and implementations of IDL-to-specific

[48] "JavaBeans+" is NAC's term to describe the fact that CORBA-compliant components can be implemented in any language with an IDL mapping, and given Java's portability, JavaBeans are a natural fit for the CORBA/IIOP model.

[49] "The Essential CORBA."

language mapping routines, or compilers) that map IDL constructs and types to corresponding elements of each of several major programming languages including C++, C, Java, Smalltalk, Ada and COBOL.

Typical developer versions of ORB products include an IDL compiler, associated header files necessary for language mappings, the ORB (runtime), Interface Repository, and Implementation Repository. (These last two run as services under Windows NT or as daemon processes under Unix versions of a specific ORB product.) Leading ORB products include IONA Orbix (which was first to market in 1991), Visigenic's VisiBroker, BEA/Digital PowerBroker, and ExperSoft PowerBroker. Visigenic's product is being incorporated into technology from Oracle and Netscape, among others, and in January 1998, Sun Microsystems announced it was abandoning its ORB product, called NEO, and would be migrating customers to the Visigenics product. In addition, several emerging products combine features of ORBs and TP monitors, products such as Sybase/Powersoft Jaguar CTS (Component Transaction Server), IBM Component Broker, and Oracle's NCA (Network Computing Architecture).

It's the JavaBean that brings the "component" aspect to the CORBA/IIOP model, in the same manner as ActiveX Controls, which is why many ORB vendors are integrating JavaBeans technology into their products. In fact, the CORBA 3.0 specification (Q4 1998) will include the JavaBeans model under the name CORBAbeans as its compound document model.

Java is an open, portable programming language developed by Sun Microsystems; Java is now managed by the JavaSoft division of Sun. Java can be used to create both applications and applets. A Java application would be written to a specific operating system, just as any other programming language might be used to write an application. An applet is a Java program that requires a Java Virtual Machine (JVM) on which to run.

JavaBeans is a platform-neutral, component architecture for Java. A JavaBean is a compiled Java component that can run on any operating system that has a Java virtual machine (JVM). Java can also be run within any application environment.

There are now over 10 different Java-based ORBs, including both commercial and freeware products, and CORBA support is being integrated into Java development toolkits. Sun, Netscape, IBM, and Oracle last fall announced plans to integrate Java more closely with CORBA through the JavaBeans architecture. Sun also announced that its proprietary Remote Method Invocation (RMI) would be implemented with support for IIOP to make components using RMI interoperable with other CORBA-compliant components. Sun has integrated CORBA support into its Java Developer's Kit, and Netscape has incorporated the Visigenic ORB and included Java-specific support for CORBA into its web browsers and servers.

Appendix B. Glossary

Activation	Preparing a CORBA object to execute an operation.
Adapter	In a CORBA implementation, the ORB component that provides object reference, activation, and state-related services to an object implementation. Different adapters may provide different kinds of implementations.
Business logic	The business rules and processes that define a particular business model. Business rules encompass entities like tax rate tables; shipping rates; prices; thresholds for performing certain business activities, for example, "add 8.5% sales tax to goods being shipped to California." Business processes encompass manaul or automated workflow processess, such as an email message sent to a credit manager to approve an order over a specified dollar limit; crediting accounts receivable; debiting inventory. Examples like these would be said to be implementing a "mail order business model."
Class	In object-oriented programming, a class is a data structure that serves as a template for the creation of objects. A class specifies the fields and methods that of the objects that will be created from it.
Class Factory	A special COM class that is responsible for creating new instances of another class within a server. This provides a common gateway for all clients to activate multiple classes within a server program.
Class ID	A GUID that identifies a COM class. Typically abbreviated CLSID.
Class Table	A list of the Class IDs and Class Factory pointers for the currently running COM servers on the given machine.
Client	In the context of distributed client/server applications, the process that requests the services of another process. In object-oriented programming, the code that invokes an operation on an object.
Component	An executable software module that encapsulates specific functionality behind published interfaces.
CORBA	Common Object Request Broker Architecture
Encapsulation	The practice of making a software entity, object, or code module self-contained, with its internals hidden. Encapsulation leads to

	flexible design because internal structures can be modified without affecting the rest of the application.
Enterprise JavaBeans	Extensions to the JavaBeans component architecture (released in JDK (Java Developers Kit) 1.1 which provide an API optimized for building scaleable business applications as reusable server components.
Framework	In the context of component-based application development, object-request brokers, and object-oriented development tools, a "framework" refers to pre-packaged business components that provide necessary base functionality for either vertical or horizontal applications. Similar to a template of business objects that you can use or modify for your own purposes.
GUID	Globally Universal IDentifier. A 128-bit identifier created by using the current date/time, a clock sequence, and incremented counter, and the IEEE machine identifier, usually acquired from a network card.
Inheritance	The property of an object-oriented programming language that enables a programmer to create a new type by adding strucutre and behavior to a pre-existing type.
Interface	A strongly typed, semantic contract between client and object. A collection of methods. An interface is identified by a GUID called an IID (Interface ID). In COM/DCOM, interfaces are typically named beginning with a capital I (for example, ICustInfo). In CORBA, interface is defined as "the listing of the operations and attributes that an object provides… [including] the signatures of the operations and the types of the attributes. An interface definition ideally includes the semantics as well. An object satisfies an interface if it can be specified as the target object in each potential request described by the interface."
Interface ID	A GUID (globally universal identifier) that identifies a COM interface.
Interface object	In CORBA, an object that describes an interface. Interface objects reside in an interface repository.
Interface pointer	A pointer to the interface's vtable in memory, or to an RPC proxy. The vtable then redirects to the desired method.
Interface repository	In a CORBA implementation, a storage place for interface information. The interface repository is one of the client-side structures of a CORBA ORB. It is a runtime distributed database that containes machine-readable versions of the IDL-defined

	interfaces. Interface repository APIs enable components to dynamically access, store, and update metadata information.
Java IDL	Java Interface definition language. A CORBA compliant interface definition language for Java. Provides interoperability and integration with the industry standard CORBA for distributed, heterogeneous computing.
JDBC	A database-independent connectivity API included in the Java Developers Kit. Sometimes defined as "Java Database Connectivity."
JTS	Java Transaction Services. A low-level Java API providing access to transaction managers.
JNDI	Java Naming and Directory Interface. A unified Java interface to multiple naming and directory services.
JMAPI	Java Management API. An open, extensible interface for managing enterprise networks over the Internet and intranets. Part of the Java Enterprise
JMS	Java Message Services—provides a standard Java API for enterprise messaging services such as reliable queuing, publish and subscribe communication and various aspects of push/pull technologies.
Marshalling	The process of packaging interface data into RPC packets for delivery across process or network boundaries.
Method	A piece of code that performs a given function, accepting and returning data and optionally manipulating an objects' state.
Model	An abstraction of a real-world entity.
Object	In object-oriented design, an object is an instance of a class, containing data and the methods that operate on it. An object conceptually is also a model, representing an abstraction of a real-world entity.
Object-oriented	A technological approach to software engineering, design, and programming that is vastly different from procedural oriented techniques in that, rather than breaking programs up into linear algorithmic solutions, problems are broken up into functional units called objects.
ORB	Object request broker.
OTS	Object transaction server.

Persistent object	An object that exists until it is explicitly deleted.
Proxy	A small binary loaded into the client's process space that acts as a "front-end" to the remote server's interface. The client calls the proxy just as it would the server, and the data is marshalled to the stub which in turn calls the server object.
Registry	A datafile on each machine that contains indexed information about classes, interfaces, software settings, preferences, etc.
RMI	Remote Method Invocation. In interface for Java remote distributed object computing which enables an object in one Java Virtual Machine to invoke methods on objects running in a remote Virtual Machine.
Repository	The definition depends on the context. An interface repository is a datastore for interface information. A component or object repository could be a datastore at the center of a modeling or development tool.
Server	In the context of distributed client/server applications, the process that provices functions or services to a requesting process.
State	Conditions or properties that are subject to differences at different points in time.
Stub	A small binary loaded into the server object's process space that accepts marshalled RPC packets from the proxy and calls the necessary method.
Type Library	A binary object that contains information about a specified class and the interfaces it exposes. It is used by tools such as Visual Basic to get Class ID, Interface ID, and method information.

References

Books

Business Objects: Delivering Cooperative Objects for Client-Server. Oliver Sims. McGraw-Hill. 1994.

Client/Server Architecture. Alex Berson. McGraw-Hill. 1996.

Client/Server Programming with Visual Basic. Kenneth Spencer; Ken Miller. Microsoft Press. 1996.

Code Complete. Steve McConnell. Microsoft Press. 1993.

CORBA Design Patterns. Thomas Mowbray, Raphael Malveau. John Wiley & Sons, Inc. 1997.

The Essential CORBA: Systems Integration Using Distributed Objects. Thomas Mowbray, Ron Zahavi. John Wiley & Sons, Inc. 1995.

Instant CORBA. Robert Orfali, Dan Harkey, Jeri Edwards. John Wiley & Sons, Inc. 1997.

Principles of Transaction Processing for the Systems Professional. Philip Bernstein; Eric Newcomer. Morgan Kaufmann Publishers. 1997.

3-Tier Client/Server At Work. Jeri Edwards with Deborah DeVoe. John Wiley & Sons, Inc. 1997.

UML Distilled: Applying the Standard Object Modeling Language. Martin Fowler with Kendall Scott. Addison-Wesley. 1996.

Understanding ActiveX and OLE: A Guide for Developers and Managers. David Chappell. Microsoft Press. 1996.

What Every Software Manager Must Know To Succeed With Object Technology. John Williams. SIGS Books. 1995.

Research Papers, Technical Notes, Articles

AD Technology 2001: Building Infrastructures as Tools Mature. Gartner Group Applications Development and Management Strategies (ADM) Strategic Analysis Report. J. Sinur, M. Blechar, K. Kleinberg, M. Merriman, B. Williams, M. Light. 10/25/96

The Application Server: From Monolith to Objects? Part 2. Gartner Group Systems Software Architectures (SSA) Research Note. Y. Natis. 11/21/96

Architecture and Planning for Modern Application Styles. Gartner Group Systems Software Architectures (SSA) Strategic Analysis Report. R. Shulte. 4/28/97.

Beware the Universal Middleware Myth. Gartner Group Systems Software Architectures (SSA) Research Note. R. Shulte. 4/29/97

Component Models Move to the Server. Gartner Group Systems Software Architecture (SSA) Research Note. Y. Natis. 9/4/97

The Component Object Model: Technical Overview. Paper adapted from an article appearing in Dr. Dobbs Journal, December 1994. (http://www.microsoft.com/oledev/olecom/Com_modl.htm)

CORBA Compliance Does Not Guarantee Portability. Gartner Group Systems Software Architecture (SSA) Research Note. M. Pezzini. 2/5/97

DCOM and CORBA Side by Side, Step by Step, Layer by Layer. Chung, Huang, Yajnik; Bell Laboratories, Lucent Technologies. Liang, Shih, Wang; Institute of Information Science, Republic of China. Wang; AT&T Labs, New Jersey. (http://www.research.att.com/~ymyang/papers/HTML/DCOMnCORBA.S.html)

Defining Components and Distributed Objects Gartner Group Systems Software Architectures (SSA) Research Note. R. Shulte. 10/21/96.

Flow Control: The Fourth Application Tier. Gartner Group Systems Software Architecture (SSA) Research Note. Y. Natis. 11/30/96

Greater Java: A Continent Emerging? Gartner Group Applications Development & Managment Strategies (ADM) Research Note. Y. Natis. 6/27/97

The Impact of Component Software on Three-Tier Designs Gartner Group Systems Software Architectures (SSA) Research Note. R. Shulte. 1/15/97.

Industry Trends Scenario: Rethinking the IT Investment Paradigm. Gartner Group Industry Trends & Directions (ITD) Strategic Analysis Report. McNee, Austin, Baylock, Blechar, Burton, Cappuccio, Fenn, Germann, Goodhue, Keller, Keyworth, Magee, Pucciarelli, Raphaelian, Schlier, Stenmark, Terdiman, West, Windkler, Cushman. 3/28/97

The Internet: Its Role in the Software Revolution and Its Impact on Enterprises Gartner Group Internet Strategies (INET) Strategic Analysis Report. D. Smith, D. Bosik. 7/16/97.

"Into ORBit. Object Request Brokers: Servers of the 21st Century." *Network Computing* magazine. 3/1/97.

Intranets, the Network Services Model, and the Future of the NOS. The Burton Group. Jamie Lewis. Network Strategy Overview, July, 1996. The Burton Group (http://www.tbg.com).

"Is DCOM Truly the Object of Middleware's Desire?" (Lab Review) *Network Computing* magazine. A. Frey. July 15, 1997.

Is Netscape ONE the One? The Burton Group Network Strategy Bulletin. Jamie Lewis. September 1996. The Burton Group (http://www.tbg.com).

Microsoft Active Server. The Burton Group Network Strategy Report. Jamie Lewis. March 1997. The Burton Group (http://www.tbg.com).

The Microsoft Repository. Philip Bernstein, Brian Harry, David Shutt, Jason Zander (Microsoft Corporation); Paul Sanders (Texas Instruments, Inc.). Proceedings of the 23rd VLDB (Very Large Database) Endowment, Athens, Greece. 1997. (http://www.microsoft.com/Repository)

Middleware: The Foundation for Distributed Computing. Gartner Group Client/Server Strategic Analysis Report. Y. Natis, R. Schulte, M. Light. 10/25/96.

Netscape One: A Platform for CORBA Ubiquity. Gartner Group Applications Development & Management Strategies (ADM) Research Note. D. Smith. 9/27/96

The Next Wave: Component Software Enters the Mainstream. David Chappell. April 1997. (http://www.rational.com/support/techpapers/nextwave)

"New Technologies Require Structured Methodologies." *Application Development Trends*. C. Trepper. July 1997.

Normalization of Logic Yields Flexible Application Servers. Gartner Group Systems Software Architectures (SSA) Research Note. Y. Natis. 1/15/97

On Persistence Services and Persistent Data. Gartner Group Systems Software Architectures (SSA) Research Note. A. Percy. 4/29/97

The Orbix Architecture. IONA Technologies. November 1996. (http://www.iona.com/Products/Orbix

Re-engineering Application Development. A Microsoft Corporation-Texas Instruments White Paper. June 1995. (http://www.microsoft.com/Repository/articles.htm)

Unified Modeling Language (UML) Summary. Rational Software Corporation. 3/19/97. (http://www.rational.com)

"What 1997 Means for Repository Technology." A. Tannenbaum. *Application Development Trends*. July 1997.

"Why Partition: Options for Applications Partitioning." *Application Development Trends.* D. Kara. May 1997.

Specifications, Technical Documents

The Common Object Request Broker: Architecture and Specification. (Chapter 1, "The Object Model," Chapter 2, "CORBA Overview," Chapter 12, "General Inter-ORB Protocol," and Chapter 13A, "Internetworking Architecture." Revision 2.0, July 1995. The Object Management Group, Inc. Updated July 1996. (http://www.omg.org)

DCOM Technical Overview. Microsoft Corporation. 1996. (http://www.microsoft.com/nt)

The Component Object Model Specification. Draft Version 0.9, October 24, 1995. Microsoft Corporation and Digital Equipment Corporation. Copyright © 1992-95 Microsoft Corporation.

Distributed Component Object Model Protocol —DCOM/1.0. Internet-Draft. Nat Brown, Charlie Kindel. Microsoft Corporation. November 1996. (ftp://ds1.internic.net/draft-brown-dcom-v1-spec-01.txt)[This has not been updated in the required six months, and so apparently has fallen by the wayside…]

Distributed Java: Securing Java Client-Server Applications. David Weisman. The Open Group Research Institute. May 7, 1997 (http://www.osf.org/RI/PubProjPgs/jade_1pg.htm)

JavaBeans™ API 1.01 Specification. JavaSoft. A Sun Microsystems, Inc. Business. July 24, 1997. (http://java.sun.com/beans)

Using the Beans Development Kit 1.0: A Tutorial April 1997. JavaSoft. A Sun Microsystems, Inc. Business. (http://java.sun.com/beans)

Email Lists and Newsgroups

The email lists below are a starting point for getting up to date information of varying technical depth about a variety of topics. In most cases, you can subscribe to a "digest" version of the email list to avoid getting flooded by email — some of these lists are extremely active. In all cases, leave the "subject" field of your email message blank.

CORBA — CORBA-Dev@qds.com Send e-mail with "subscribe" in the body of your message. The OMG maintains mailing lists about Java and CORBA, but they are available only to OMG members. Contact the OMG for details (http://www.omg.org). For newsgroups, start with comp.object.corba

Java — Send e-mail with "subscribe" in the body of your message to JavaLobby@iceworld.org. There are several Java newsgroups tailored to different interests, including:
comp.lang.java.advocacy
comp.lang.java.announce
comp.lang.java.api
comp.lang.java.beans
comp.lang.java.misc
comp.lang.java.programmer
comp.lang.java.security
comp.lang.java.setup
comp.lang.java.tech

Java/CORBA — Newsgroups covering Java/CORBA include comp.object.corba and in the comp.lang.java. Comp.lang.java.corba is an un-moderated group is for aspects of Java technology that are related to CORBA software development, products, and standards.

JavaCORBA@luke.org Send e-mail and write "subscribe" in the SUBJECT line. For complete directions on subscribing and additional information on this mailing list (including archived discussions), visit JavaCORBA online.

Microsoft DCOM Mailing List — The DCOM mailing list covers discussions about writing distributed COM-based code. You can subscribe to Microsoft's DCOM mailing list by sending an email message to **LISTSERV@LISTSERV.MSN.COM** with the command **subscribe dcom <your_firstname your_lastname> digest** in the message body.

Microsoft's position on Java — Visit http://www.microsoft.com/java/issues/techsupfaq.htm for a discussion of issues such as the suitability of JFC (Java Foundation Class) and JNI, Microsoft's support of Remote Method Invocation, and information about upcoming class libraries. For more Java information, visit http://www.microsoft.com/Java , or subscribe to Java-COM@LISTSERV.MSN.COM

Object Technology Users Group Email List — The Object Technology User's Group (OTUG) email list is a non-profit program sponsored by Rational Software Corporation and the Lockheed Martin Advanced Concepts Center. The OTUG email list focuses on the Unified Modeling Language (UML), the de facto industry standard modeling language with origins in the modeling languages of Booch, Jacobson/OOSE, OMT, and other methods. Subscribe to this email forum for active discussions about object-oriented analysis, object-oriented design, and related topics by sending an email message to majordomo@rational.com with the command subscribe otug <your_firstname your_lastname> in the message body.

Rational Rose Technical Forum For discussion of technical features of Rational Rose modeling tool, subscribe to the Rational Rose email forum by sending an email message to majordomo@rational.com with the command subscribe rose_forum in the message body.

Vendor Web Sites

Developing distributed object-oriented or component applications requires modeling tools, integrated development environments, code generators, IDL generators, testing and debugging tools. Use this product directory as a reference for your own research.

Vendor, Sample Product Name	Web Address
BEA Systems, Inc.BEA Builder, Jolt	http://www.beasys.com
BMC Software Patrol® Management Suite	http://www.focalpoint.com
Borland Delphi, C++ Builder, JBuilder	http://www.borland.com
Compuware Corporation UNIFACE	http://www.compuware.com
CrossRoads Software	http://www.crossroads-software.com
Dynasty DYNASTY	http://www.dynasty.com
Expersoft	http://www.expersoft.com
IBM VisualAge, San Francisco Project, CBToolkit	http://www.ibm.com
Magna Software Corp MAGNA X	http://www.magna.com
Mercury Interactive Corporation LoadRunner, WinRunner, Xrunner	http://www.merc-int.com
Micro Focus Micro Focus COBOL	http://www.mfltd.co.uk
Microsoft Visual Basic, Visual C++, Visual J++	http://www.microsoft.com
Neuron Data Elements Enterprise/C	http://www.neurondata.com
Object Design, Inc.	http://www.odi.com
Oracle Developer/2000, Designer/2000	http://www.oracle.com
ParcPlace VisualWorks	http://www.parcplace.com
Persistence Software, Inc.	http://www.persistence.com
Planetworks Interspace	http://www.planetw.com
Prolifics (A JYACC Company) JAM, JAM/WEB	http://www.prolifics.com
Rational Rose	http://www.rational.com
Rational (formerly Pure Software, Pure Atria) EMPOWER, Visual Quantify, PurePerformix, Performix/Web	http://www.rational.com
Seer*HPS®	http://www.seer.com
Sterling Software COOL:Gen (formerly known as Composer), KEY:ObjectView	http://www.sterling.com
Sybase/Powersoft PowerBuilder	http://www.sybase.com
Unify Vision	http://www.unify.com