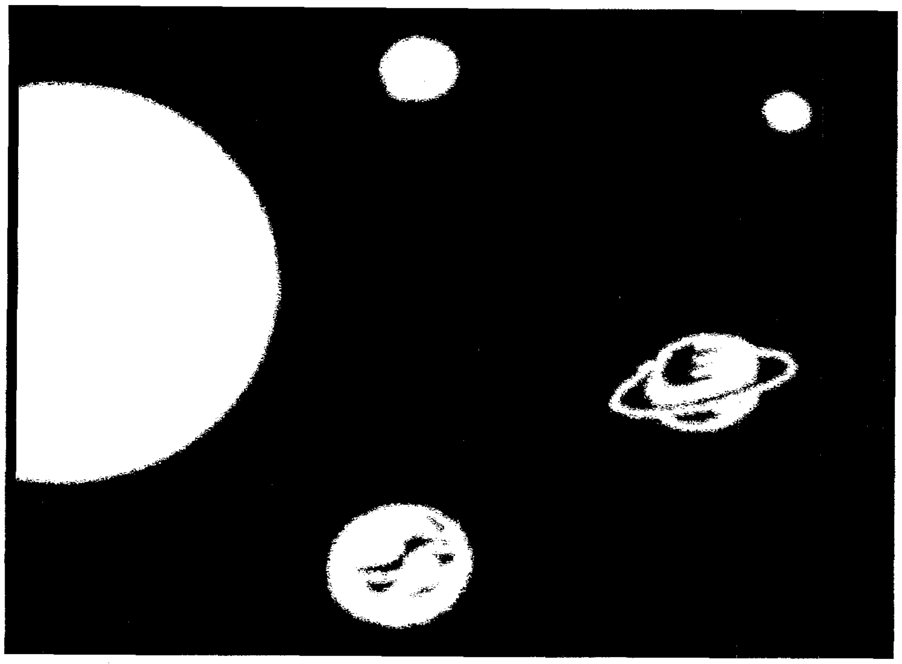
Architectural Design of a Common Operating Environment

The architecture for the Global Command and Control System's new common operating environment will comprise several modern architectural concepts currently used in complex, distributed systems. We plan to reduce the environment's complexity by using architectural components supported by existing commercial products and standards.



SHAWN BUTLER AND DAVID DISKIN
Defense Information Systems Agency

NORMAN HOWES AND KATHLEEN JORDAN
Institute for Defense Analyses

The Global Command and Control System is the next-generation replacement for the legacy World Wide Military Command and Control System, which has served the command and control needs of several high-level US military commands for more than 20 years. The GCCS is implemented on heterogeneous Unix workstations as opposed to the WWMCCS's mainframe implementation. The GCCS will eventually extend down to tactical levels of command and control not encompassed by the earlier system. For this and other reasons, the Defense Information Systems Agency is undertaking the architectural design of the new GCCS common operating environment.

0740-7459/96/$05.00 © 1996 IEEE

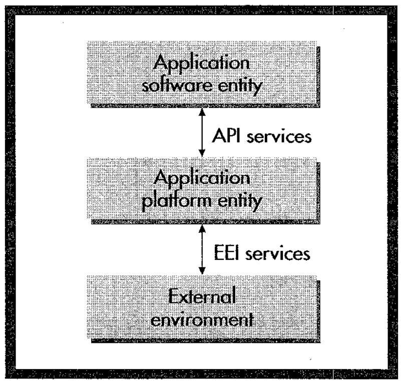


Figure 1. *A generic technical reference model defines the services and interfaces common to all DoD information-processing systems.*

In a recent *IEEE Transactions on Software Engineering* guest editorial,[1] David Garlan and Dewayne Perry explore the concept of software architecture. Two aspects they focus on are the architecture of a specific system and the architectural style of classes of systems. We have included both these aspects in our description of the GCCS's COE architecture. Because the GCCS already has a COE, we refer to it as the legacy COE. The COE that the Design Working Group is developing will be the architecture to which the legacy COE migrates.

The COE's requirements are documented in the *GCCS Common Operating Environment Requirements*[2] and the *GCCS Baseline Common Operating Environment.*[3] The Design Working Group, chaired by one of us (Butler), began work on the design in February 1995. From the beginning, the design group decided to focus first on the architecture of the COE and then to use the architecture as a framework for further design activity.[4] The design group decided on the architectural style and then began work on specifying the components of the system's specific architecture. To help understand the individual components of the system, their interrelationships, and the policies and guidelines that govern these relationships, the group constructed many typical command-and-control scenarios and stepped through them.

This effort caused the rethinking of the original client/server architecture. The current COE has a variant of the original architecture and can be characterized as client/broker/server, or more specifically, a client/server with a subscription broker. This component provides several key features we consider crucial. First, neither the broker nor the system in general knows anything about the operation or implementation of any of the clients or servers. This satisfies the system's fault-tolerance and graceful-degradation requirements. It also supports the requirements for openness and system evolvability.

Second, the data that clients consume and the names of specific services are the only global knowledge available to the system. This eliminates the need for building knowledge about other applications or other servers into a system's various applications or servers. Furthermore, other servers and applications can be made completely transparent to the system. If designers take advantage of this approach, it can result in the greatest possible simplicity in application and service design.

Third, data consumed by clients is transferred from the location where it is produced to a local, client-owned exchange immediately upon production so that it will be available for real-time, response requirements. This is essential for real-time and near real-time behavior.

Fourth, data plays the significant role in this architecture. To support both real-time and fault-tolerant operation, the architecture can distinguish between persistent data and real-time or near real-time, nonpersistent data. Clients subscribing to real-time or near real-time ,nonpersistent services are assured that the transfer of this type of data is optimized at the possible expense of persistent data. The broker component determines the priority of any given data class.

STRUCTURAL OVERVIEW

To label an architectural style as client/server or as client/broker/server conveys different meanings to different industry professionals. Our definitions may not coincide exactly with yours. For example, by a *server*, we do not necessarily mean a physical processor on a network, and by a ***broker***, we do not mean an object request broker as described in the Common Object Request Broker Architecture terminology,[5] although we believe that the architectural style we specify could be implemented with CORBA.

Also, some aspects of our architectural style relate to the specific nature of the COE. These cannot be characterized by any current standard terminology. These system-specific aspects must help ensure compliance with the DoD *Technical Architecture Framework for Information Management.*[6] Being TAFIM-compliant is essentially independent of whether the system has a client/server architectural style. Compliance has to do with the services the system provides, how they relate to each other, and how the user accesses them. This affects how we have chosen to organize the services of the COE at the highest level.

The TAFIM contains a technical reference model that establishes a vocabulary and defines services and interfaces common to all DoD information-processing systems. The TRM represents a framework for providing services needed to move from legacy stand-alone systems to a distributed-computing environment. It defines terminology, service, and interfaces. The TRM consists of three principal components: the *application software entity*, the *application platform entity*, and the *external environment*. Figure 1 shows how the application software entity interacts with the application platform entity via the application program interface services, and the application platform entity interacts with the external environment via the external-environment interface services.

Figure 2 shows how the TRM can be further refined by decomposing the boxes shown in Figure 1. In this diagram, shaded areas show the services provided by the COE, and nonshaded areas show the areas of service provided by the initial version of the COE. Later versions may include the nonshaded service areas as well.

As Figure 2 shows, the COE provides services on both the application software entity and the application platform entity. For that reason, we divided the services of the COE into two groups: The *infrastructure services* correspond to the application platform entity services in the TAFIM; the *common support applications* correspond to the support services that are part of the application software entity in the TAFIM.

Each of these two service categories has five subdivisions, as shown in Figure 3. The specific services provided by the common support applications will continue to evolve. However, the infrastructure services that essentially provide the distributed-computing environment for the COE should evolve slowly over time.

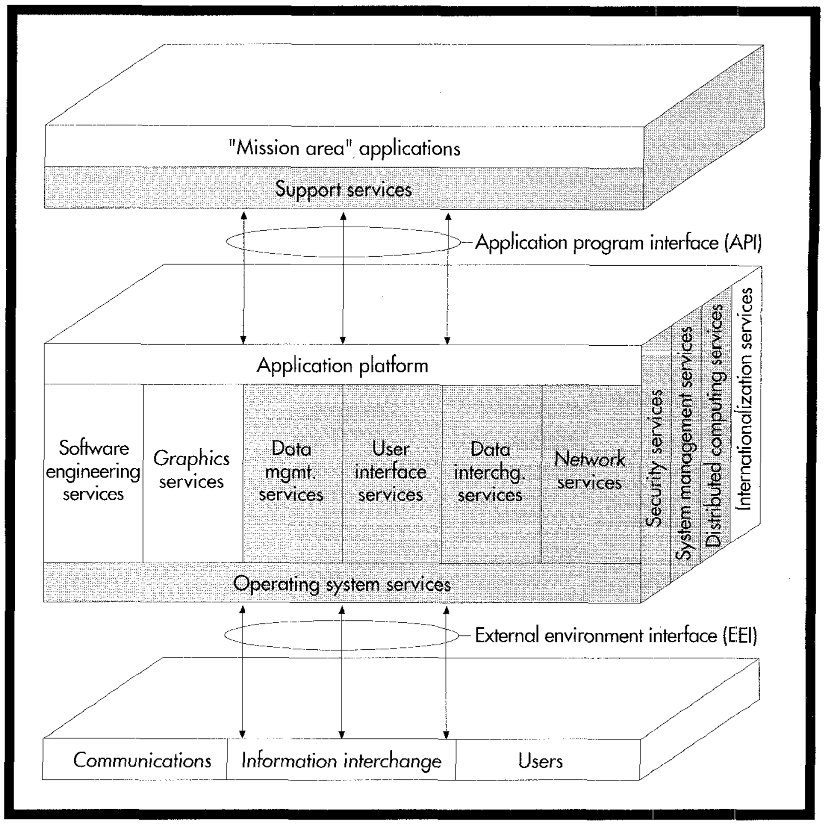

"Mission area" applications
Support services
Application program interface (API)
Application platform
Software engineering services
Graphics services
Data mgmt. services
User interface services
Data interchg. services
Network services
Security services
System management services
Distributed computing services
Internationalization services
Operating system services
External environment interface (EEI)
Communications
Information interchange
Users

***Figure** 2. Detailed technical reference model.*

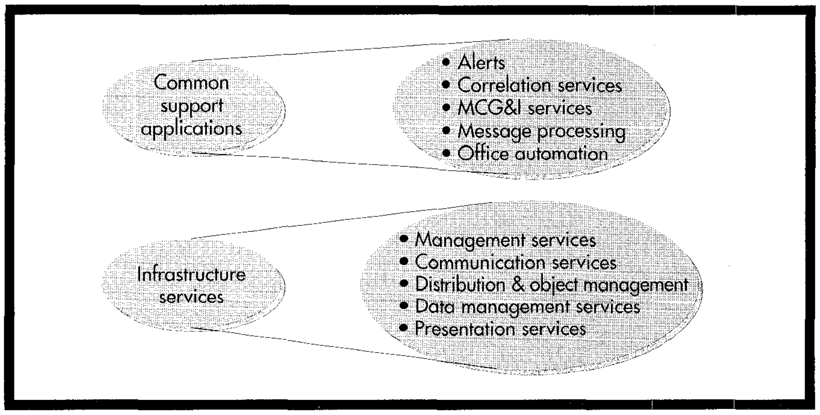


***Figure** 3. COE services.*

ARCHITECTURAL STYLE

Our concept of a COE *client* is a computer program, such as a mission application, that needs a service. A *service* is some function common to several programs, such as retrieving a certain category of data or performing some extensive calculation. Because the candidate function is common to many programs, the system's designers decide to implement it as a service that all these programs can use, rather than embed the function in each program. This makes the individual programs simpler, which in turn makes them easier to maintain and debug. It also reduces costs by avoiding redevelopment in each programming task and provides standard solutions that contribute to the interoperability and portability objectives.

Servers are self-contained processes designed to execute independently and to be accessed by several clients simultaneously. In practice, servers are themselves often clients of other servers because system designs can involve hierarchies of programs that serve other programs. Programs can *assume roles* as either clients or servers. For example, Figure 4 shows a correlator program assuming the client role to fulfill its server function.

A client is a data consumer whether it

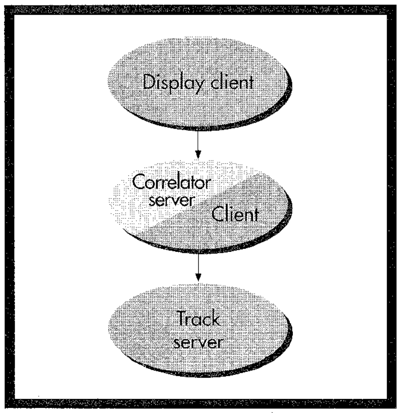


Figure 4. Simultaneous client and server. Arrows indicate that a client is requesting the service from a server.

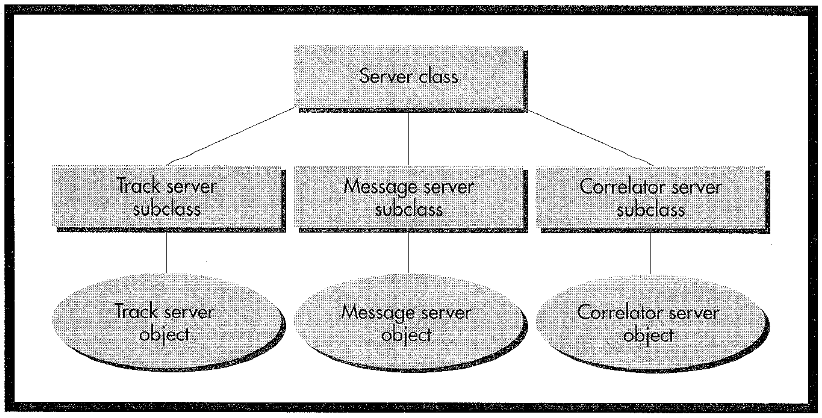


Figure 5. A Server class and multiple server objects.

is a server playing a client role or a mission-application client. Clients receive data of specific categories from the servers responsible for those categories. In the COE, servers own and are responsible for major data categories. Data is categorized by what it describes rather than by its source, although the source may be an attribute of the data available to the client. For example, all track data describing the position, attributes, and status of an airborne vehicle such as a helicopter would be categorized together regardless of that data's source: radar installation, navigational database, or intelligence organization. The COE requires that data of a given category be in a category-specific format; if it is not, the server will convert the data to a category-compliant format.

COE's object orientation. The COE architecture uses object-oriented design concepts. We define an object as a computational entity that combines both data and operations, and behaves in an expected manner. An object has a set of attributes and a set of operations it must provide. When a client requests a particular service, the system sends a message to the appropriate server object. Because they are encapsulated, objects need only know about the specifications of other objects, not the objects' implementation details. The specification acts as a "contract" between communicating objects. Thus, the underlying object implementations can be changed without affecting any other objects as long as the specifications themselves remain unchanged.

Servers inherit some behavior from the server class. Server objects are instances of classes that have the Server class as an ancestor—a parent of a parent. We use behavior inheritance because our system has a set of functions that all services should perform the same. The GCCS COE provides such a service template by means of the Server class shown in Figure 5.

Service initialization begins when the COE instantiates a server object, referred to simply as the server, which then registers with the COE. This consists of providing its server type, location, identification, and so forth to the subscription broker. The server then subscribes to each of its data providers. At this point, the server is ready to handle client requests and provider events.

When the server receives a provider update it commits the change to its database, which may be memory resident or may reside in external storage. Client requests for data often take the form of a subscription. Each client that needs data of a certain category from a server must first subscribe to the server that owns the data.

There are two types of subscription: continuous update and ad hoc. When a client requests a continuous update subscription, the server continually updates a local database for the client with the subset of this category of data that the client defines. Clients define the subset of a given data category by means of a filter. With an ad hoc subscription, the server only responds to the client with data updates when the client sends a specific request. The client can specify a new filter with every subsequent request. The continuous update subscription is a data-push scheme, whereas the ad hoc subscription is a data-pull scheme.

Client/broker/server. So far, for simplicity's sake we have described subscriptions as if the client and server explicitly negotiated these themselves. This is not the case; the COE architecture provides a subscription broker. The subscription broker manages on the clients' and servers' behalf the connections they must deal with. With a subscription broker, clients need only register with the broker instead of with several servers.

The subscription broker functions as a management server, although it need not be implemented as a server. It is often advantageous for the subscription broker to have its own distributed architecture, such as Martin Boasson has described.[7,8]

Distribution of the subscription broker facilitates fault tolerance and has other advantages. Although the subscription broker can be a fairly complicated design component, it can greatly reduce the complexity that application programs must deal with: Adding this one complex component relieves the application programs of the complexity and duplication of providing the capabilities themselves.

Figure 6 shows a subscription brokerage service implemented as a multithreaded server. The subscription broker acts analogously to a stockbroker in an interaction between a client and several companies selling stock. Rather than having each client buy stock directly from all the companies, it is more practical to have specialists, called stockbrokers, handle these transactions for a small fee. The stockbroker hides the complexities of negotiating stock transactions from the buying client. Similarly, the subscription broker in the COE client/server architecture hides the complexities of interacting with servers. As Figure 6 shows, multiple instances of client threads of control let the subscription broker maintain multiple clients simultaneously, thereby avoiding bottlenecks in the client-to-server communications paths. Also, there are multiple server threads of control that let the broker maintain simultaneous connections with multiple servers. In Figure 6, the dashed arrows denote subscriptions.

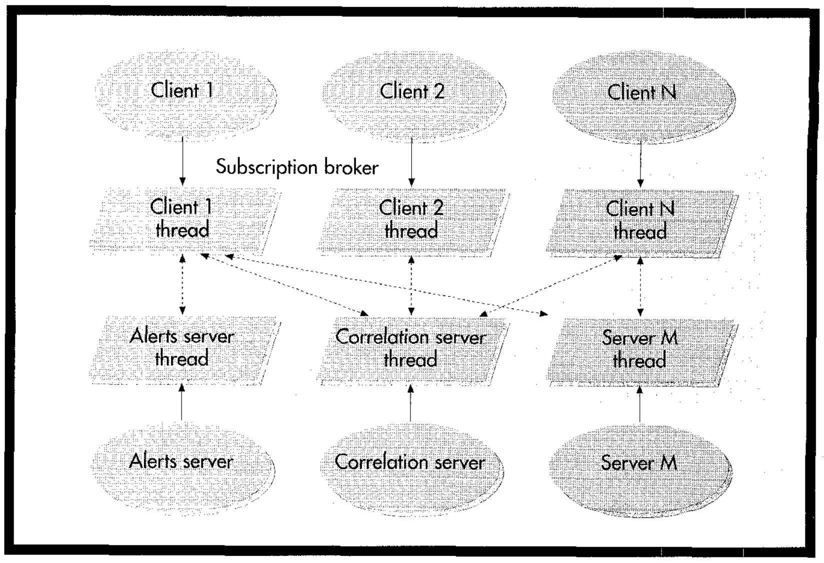


Figure 6. A subscription broker, here implemented as a multithreaded server, can maintain simultaneous connections with multiple clients and servers.

All the connections take place within the broker. The broker must maintain a database of which clients are allowed to get data elements from which servers. This reduces the complexity of both the clients and the servers. Now each server need only be connected with the broker. The broker manages all the server connections for the clients and all the interconnections among the servers themselves.

In Boasson's work, the subscription broker is implemented as a collection of separate processes called heralds. These heralds play the role of the threads in Figure 6. However, each herald can execute on a different network node or multiple heralds can execute on the same node. Moreover, in Boasson's implementation—called Splice for subscription paradigm for the logical interconnection of concurrent engines—the heralds are all identical and use a common protocol independent of which clients are connected to which servers. The data communications needs of the clients are established by the distributed actions of their heralds, thereby avoiding a common point of failure. The multithreaded server implementation of the subscription broker can also be made fault tolerant by having multiple copies of the broker on each LAN within the network.

All clients that are currently registered subscribers receive the same continuous updates. However, the API the broker presents to the client uses the filter provided by the client to update only the client's local database with those data elements that it is interested in. As clients subscribe and unsubscribe, the server determines the current client data requirements and tailors the continuous updates accordingly.

COE'S ARCHITECTURE

Based on discussions at the Software Engineering Institute in 1994,[1] Garlan and Perry define a software architecture as

> *the structure of the components of a program/system, their interrelationships and principles and guidelines governing their design and evolution over time.*

In the Draft Architectural Design Document for the GCCS,[4] we provide separate chapters to document the components,[4] interrelationships,[5] and the policies and guidelines.[6] Not all the infrastructure services and common support applications will be implemented by servers; those components that do not directly affect distribution considerations are therefore excluded from the following description. As Figure 3 shows, the COE architecture is divided into infrastructure services and common support applications. Infrastructure services include communications, management, distributed computing, data management, and presentation—those capabilities necessary to move data throughout a network. Common support applications are common end-user applications and services, such as office automation, message processing, alerts, and correlation.

The COE architecture is inclusive in that it contains services similar to those provided by other models, such as the OMG's Object Management Architecture and the Distributed Computing Environment. The OMA object services roughly correlate to the COE distributed computing and data management services, while the DCE services correlate to the COE communications, management, and distributed computing services. We have yet to determine if a Corba-type object request broker could serve as the COE subscription broker. Early prototypes of the COE include both DCE and Corba implementations as off-the-shelf pieces of the architecture.

In addressing the needs of a command-and-control environment, the COE architecture is specifically designed for a collection of relatively high-bandwidth LANs connected into a

WAN by means of various comparatively lower-bandwidth communications links. This feature of the environment dictates the COE's communications architecture.

The security server is the first to be initiated when the system is brought up, because no other server can run without it.

Communications services. These provide for various communication modes between distributed application clients. They also provide services such as "talk" and "broadcast," and high-level protocols such as SMTP, FTP, and Telnet. Also, the GCCS must be able to communicate with external systems such as the Naval Tactical Data System and the Theater Battle Management System.

These services are provided by communications servers, which are categorized as either network or channel-specific servers according to the transport medium they use. The former use LANs and WANs; the latter use serial, NTDS, and parallel media.

Communications servers may use underlying interprocess or internode communication protocols, but the details of how communication is accomplished are hidden from the individual mission applications and other servers. Thus, mission applications see a single communications protocol and interface independent of platform, operating system, networking software, and transport medium.

The COE maintains at least one instance of a network server in each GCCS network LAN, and will generally have several instances, depending upon the installation's external communications requirements. The network servers may employ a broadcast technique to reduce bandwidth on the relatively low-bandwidth communication links. For example, if several clients on one LAN have a subscription with a server on another LAN, that server will send only one copy of the data to the clients' LAN server. The clients' LAN server would then transfer the data, with the cooperation of the broker, to each requesting client. A network server may also reduce message traffic on the low-bandwidth links by batching multiple smaller data packets when appropriate.

A GCCS site frequently must receive data from or send data to external devices or systems not connected to a LAN. For example, shipboard GCCS installations will require connectivity to tactical intelligence and other independent systems. The connectivity interfaces are typically serial but may also be parallel. As another example, some sites may require a serial interface between two GCCS LANs when the LANs are at different security levels. This approach typically requires a man-in-the-loop or sanitizer to preserve security. Such details are hidden from clients.

Management services. These include services for network management, system administration, security, and security administration. Only the security service will be implemented as a server. This server is the first to be initiated when the system is brought up, because no other server can run without it. It is replicated and its database is distributed. The security server should be immune to attacks involving the capture, modification, and retransmission of legitimate messages.

Distributed computing services. These let geographically distributed applications and services interoperate as if located on the same computing platform. They also support applications that may be physically or logically distributed among computer systems in a network, but wish to maintain a cooperative processing environment. Finally, they provide data interchange services that support the interchange of data between applications. COE's distributed computing services are implemented by three server types: name, time, and object interchange.

The name server provides a generic naming and aliasing capability for the objects that can be used by the mission applications to access or invoke these objects—for example, Track_Data for a track file or Big_App for a mission application. The name server provides many of the functions of a name or directory service in other distributed computing environments. In addition, the name server tracks the current status of the COE servers. Servers register with the broker during initialization. During registration, a server gives the broker its name, location, and the service it provides. The broker passes this information to the name server, which stores the knowledge of where these named objects are located and how to access or invoke them. It also lets local aliases override a systemwide name for an object, which allows for different object views by different mission applications.

The time server provides a time model for all mission applications on the network, based on time zones, time scales, and commonly used time formats. It performs platform-specific conversions, depending on where the mission application is executing or where another object is located that supports the mission application. Some operating systems store local time in the system clock while others store other time representations, such as Greenwich mean time. The time server provides a time service that works identically for all mission applications.

Because command-and-control applications can have unique requirements with respect to time, a COE time server is provided. Our system uses the Distributed Computing Environment's DCE Time Service as the underlying time server.

Data management services. These provide access to files and databases distributed over the network and furnish basic management functions to maintain operational integrity and application availability. They provide the administrative services required to configure, operate, and maintain database management systems. They also support requirements for the definition, storage, and retrieval of data elements or objects from both monolithic and distributed relational database management systems and object management systems. These data management services are implemented by three server types: object-base servers, data management servers, and file servers.

The object-base server provides the model for the distributed objects of the system. It provides the framework for object bases, as opposed to databases, and is the repository for all methods that can be employed on or by an object. This server stores the executable code for each method or the code for "messaging an object" to perform the method. It stores the knowledge of relations between objects, as well as the attributes of the objects and their relationships. It also hides the implementation of objects from the mission applications.

The data management server provides read-and-write access to multiple commercial relational databases such as Oracle or Sybase, while presenting a single logical relational database view to mission applications. This single, logical view capability includes SQL services. The server provides encapsulation for legacy systems that lets mission applications access data from a legacy system as if it were a database. This is called a wrapper. The legacy system may compute the requested data on the fly or, more commonly, compute it periodically, with the computations stored in a database managed by the data management server. The data may be "stale" if computed periodically, but to the mission application it is retrieved exactly the same as an on-line database retrieval.

The file server provides a common view of the file system regardless of the specific platform the file resides on, and a common transaction service that provides the capability to manage atomic transactions across any number of cooperative common support applications.

Presentation services. These are the end-user input and output support services. They are provided by the desktop manager, which supports the requirements for presenting and managing a common menuing system. The desktop manager also provides multimedia support for imagery, voice, video, and animation. To date there have been no servers identified for this class of services.

Common support applications services. These are implemented by several servers, libraries, and APIs. The services include the alerts, correlation, message-processing, office automation, and MCG&I (mapping, charting, geodesy, and imagery) services. Although no server for the office automation service has been identified yet, a map server provides part of the MCG&I service.

The alerts service routes messages regarding alerts and other system events to agent tasks in mission-application-display services for posting to workstations, to the tactical picture, or to trigger other system or user action. The alerts service contains alert and other event message formats and controls the posting, reviewing, queuing, and dequeuing of alerts and other events. It also provides the ability to dynamically define alerts and other events and the system action related to them. The service provides a standard API for alert generation. It also provides an interactive user interface for creating and transmitting alerts. The interface allows the creation and naming of alert routing lists, which can include addressing by role, duty group, or user ID.

The alerts service provides a prioritization scheme for all alert processing as well as a degree of alert accountability. If an alert cannot be delivered and the alert is directed to an individual, position, or headquarters, the alerts service generates an e-mail alert message to the addressee.

Dynamic triggering of alerts in response to system events is controlled by an alerts server. This server subscribes to those servers necessary to determine if an event has happened. When an event occurs that has an associated alert, the alerts server uses the communications services to transmit the appropriate alert, which it chooses from the alert routing list.

The correlation service supports the requirement for presentation of a com-

Dynamic triggering of alerts in response to system events is controlled by an alerts server.

mon view of the battle space. This service correlates collateral-level inputs from various sensor and intelligence sources and provides input to the tactical plotting function of the MCG&I service. The service provides correlation for five data categories, each with its own server: tracks, electronic intelligence, communications intelligence, image intelligence, and infrared intelligence. In addition to its correlation

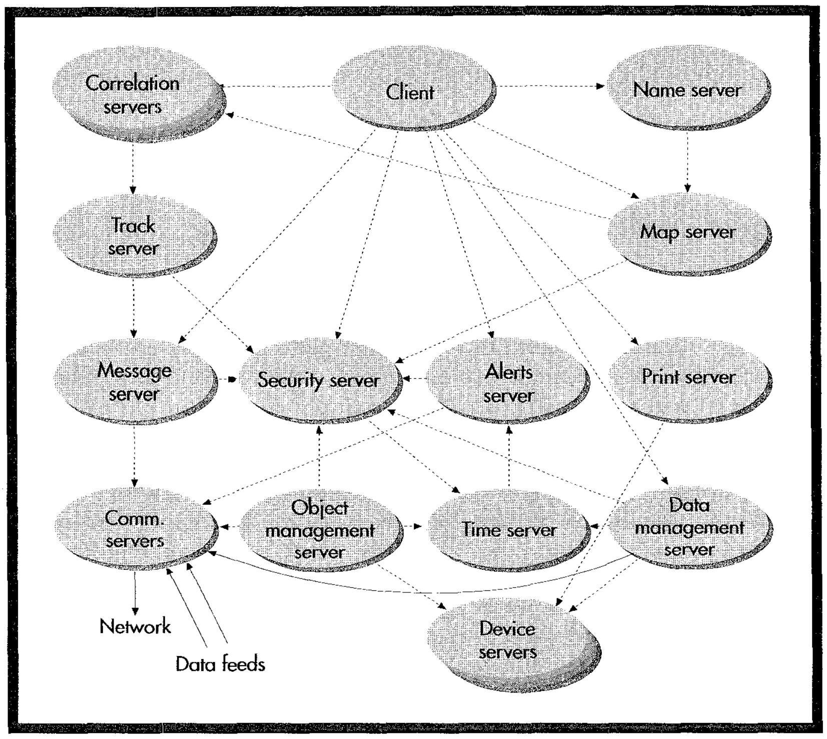


Figure 7. Client/server subscription management without a broker.

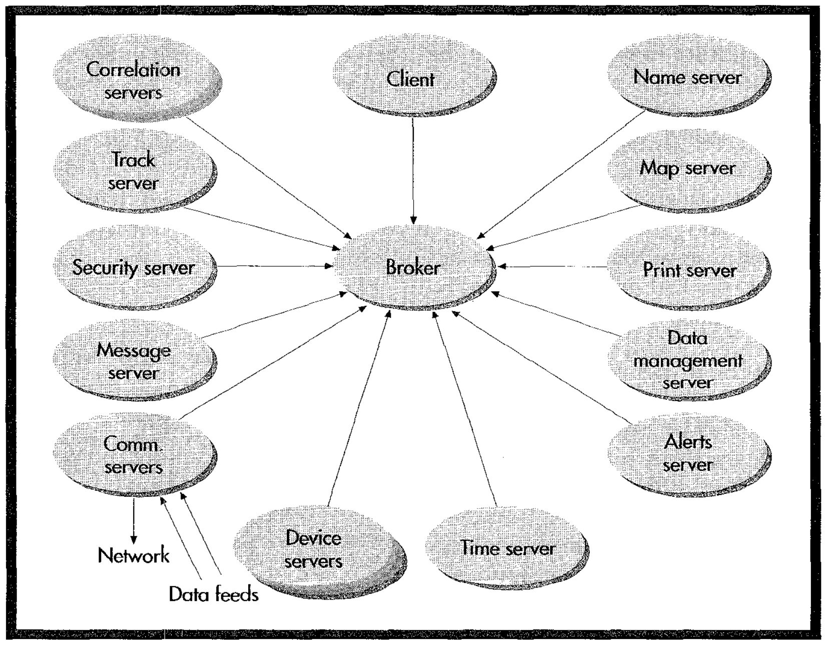


Figure 8. Client/server subscriptions managed through a central broker.

functions, the service provides terrain-delimited ground tracking and kinematic analysis to statistically project ground tracks using terrain information.

Client/server vs. broker models. Figure 7 shows the subscriptions a typical client would have with these servers and the subscriptions among the servers, that is, the connection the clients and servers must manage themselves if a subscription broker is not present. The head of each arrow points to the server being subscribed to, while the origin of each arrow indicates the particular client or server subscribing to a service.

Figure 8 shows the connections to these servers through the subscription broker. Comparing Figures 7 and 8, we see that not only is the number of connections between a typical client and the servers reduced from 9 to 1, but that the broker is positioned to handle the management of network faults for the client.

For a given client, if a connection to a server goes down, the broker must sense that a client's connection has terminated without a termination request, and that the termination may be for several clients that have not requested one. The broker automatically sends an alert via standard network management protocols to notify users and administrators that a server terminated abnormally.

Our subscription broker design philosophy supports the importance-based real-time behavior described by Norman Howes.[9] Emphasis is placed on which server owns and produces which data categories, which clients consume which data categories, and which data categories are more important than other data categories. It is even possible to support the dynamic importance ranking of data categories. The subscription broker then assures that subscription updates of the highest importance are transmitted first on the low-bandwidth links that connect the various geographically distributed LANs into a single WAN. On a given LAN, the bandwidth should be adequate to support all transmissions of subscription updates from clients to servers. But this may not be the case on the low-bandwidth links that interconnect the LANs.

This prioritizing scheme makes optimal usage of limited bandwidth resources and assures that real-time applications can coexist with, say, commercial off-the-shelf applications within this architecture. ◆

REFERENCES

1. D. Garlan and D. Perry, "Introduction to the Special Issue on Software Engineering," *IEEE Trans. Software Eng.*, Apr. 1995, pp. 269-274.
2. *GCCS Common Operating Environment Requirements*, Defense Information Systems Agency, Arlington, Va., 1994.
3. *Baseline Common Operating Environment*, DISA, Arlington, Va., 1994.
4. *Draft Architectural Design Document for the Global Command and Control System (GCCS) Common Operating Environment (COE)*, DISA, Center for Software, Arlington, Va., 1995.
5. Object Management Group, (OMG), *The Common Object Request Broker: Architecture and Specification*, John Wiley & Sons, New York, 1992.
6. *Department of Defense Technical Architecture Framework for Information Management, Version 2.0*, DISA, Arlington, Va., 1995.
7. M. Boasson, *Complex Interactive Systems, The Effective Use of Knowledge*, forthcoming.
8. M. Boasson, "Control Systems Software," *IEEE Trans. Automatic Control*, Vol. 38, No. 7, 1993, pp. 1094-1106.
9. N. Howes, J. Wood, and A. Goforth, "The Peer Tasking Design Method," *Proc. IEEE Workshop on Parallel and Distributed Real-Time Systems at the 1995 Int'l Parallel Processing Symp.*, IEEE Press, Piscataway, N.J., 1995.

Shawn Butler is a graduate student at Carnegie Mellon University. Her research interests are software architectures and security. Previously, she was the chief engineer for the Defense Information Infrastructure Common Operating Environment. She has worked extensively in command and control, as a systems developer and a user.

Butler received a BS in linguistics from Pennsylvania State University and an MBA in information systems from Case Western Reserve University.

Norman R. Howes is a research staff member at the Institute for Defense Analyses. His research interests include distributed and parallel software system architectures, distributed and parallel real-time systems, and computational mathematics. He has been on the faculties of Texas Christian University, the University of Houston, and George Mason University.

Howes received a BS in mathematics from Eastern New Mexico University and a PhD in mathematics from Texas Christian University.

David Diskin is a computer scientist in the Software/Data Architecture Engineering Department, Center for Computer Systems Engineering, Defense Information Systems Agency. He has been involved with software process assessment/improvement and software measurement at the Center since 1991, helping to launch both programs. His area of interest is object-oriented technology, particularly its application to the Common Operating Environment being developed for the Defense Information Infrastructure. Before joining DISA, Diskin was a senior member of the technical staff at the Contel Technology Center, where he worked on both the software metrics and software process programs. He also worked at the U.S. Census Bureau for many years and served as director of the Bureau's Software Engineering Process Group.

Diskin received a BS in mathematics from the University of Michigan and an MS in applied mathematics from Purdue University. He is a member of IEEE and ACM.

Kathleen Jordan is a research staff member at the Institute for Defense Analyses in Alexandria, Virginia. She supports DISA in the specification of the Defense Information Infrastructure COE. Her research interests include requirements engineering and distributed computing.

Jordan received a BS in physics from Muhlenberg College and an MS in software systems engineering from George Mason University.

Address questions about this article to Butler at Computer Science Department, Carnegie Mellon University, 2336 Eldridge St., Pittsburgh, PA 15217; shawnb+@cs.cmu.edu.