A NETWORK-CENTRIC ARCHITECTURE FOR DISTRIBUTED C4I SYSTEMS

Kathleen Lossau
Austin Info Systems, Inc. Austin, Texas
lossau@ausinfo.com

ABSTRACT

The goal of this architecture paper is to describe a more efficient and effective standardized framework for distributing information in a networked or web-based environment. This architecture provides a framework so many diverse networked applications can work in a collaborative environment sharing data and displays. Technologies such as CORBA, DCOM, and RMI provide the mechanics for communication and distribution of information. The objective is to provide an architecture where multiple distributed applications and users can share information in a heterogeneous environment using a wide range of applications and data within the framework without compromising any of the COE DII/NGII standards and objectives. This architecture is designed to support maximum platform portability and adherence to software standards. This document describes the existing data sharing problem, presents an architecture to solve the problem, and describes the integration of these application components and services into existing C4I systems.

PROBLEM

In current environments, many applications display information from the same source, but in different ways. For example, message traffic may be displayed on a map (from location information), in a spreadsheet browser (displaying all header information – message type, dtg, etc), and on a graph displaying frequency analysis information (time and frequency). Although each of the applications accesses the same data source (e.g. a database record) to obtain its information, they can rarely share the information directly with other applications. Each application needs to be familiar with where and how the data is stored.

Sharing Data

The reason this problem occurs is that applications, even object oriented applications, create an entirely new object from the data they retrieve. This object usually represents a subset of the available data. The object may even generate derived attributes (e.g., generate a MGRS location from latitude and longitude fields). Each application formats the data based on how it needs to use or display the information. Sharing these objects directly between applications has inherently been difficult because only a subset of the data is available and the second application must understand how the other application has formatted the data. In many applications where there is true object sharing, a unique identifier for the information is shared and the second application retrieves the data from the same original source as shown in Figure 1.

![Diagram of sharing data between applications](image)

Figure 1: Sharing data between applications

While there is a mechanism for sharing information, the users of the application are not sharing the objects (as each application has its own internal object), nor are the applications working in a collaborative environment. The database is still necessary for changes in the data. It is also expected that each application knows in what data source to find 'record # 12345' and does not address accessing new types of data discovered.

Collaboration

Sharing information in a collaborative environment means more than accessing information from the same data sources. One can envision two users who access locally-networked data collaborating on a problem where each users accesses: local data, shared external data, and data being modified for collaboration that can come from any local or external data source.

Figure 2 shows a potential collaboration environment where users are accessing data from local, external, and collaboration sources. Applications must understand where to retrieve the information (e.g., repository 'A' record # 12345). The collaborative environment must also provide for data redundancy when Analyst 1 and Analyst 2 both bring in the same data from the commonly accessible external data. More sophisticated deconfliction techniques are needed for
resolving data redundancy from separate local data sources (e.g., repository 'A' and 'B').

Figure 2: Collaborative Environment

Dynamic Data Access

A third problem we face in a distributed environment is the vast number of applications and data repositories that are available. With each application creating its own representation of data, retrieving only the information it needs to adequately display the object, they do not have the data necessary to send to the other application. While accessing common data between local applications provides a simple mechanism for sharing data, it does not support collaboration. In addition, it does not support being able to dynamically access new types of data that can be encountered in a distributed environment (e.g., Repository 'A' format is unknown to 'Analyst 2').

In today's client/server (2-tier) environments, applications are written to access data in a specific format (e.g. from the Oracle Database). Selections are made during application development that determines which data sources are accessed. Specific client (application) server (data) solutions are implemented. Data in other formats are unavailable to the application. Any time a data source changes (e.g. a switch in databases or even a simple schema change), changes in all applications that access that data are likely. This is why systems often create their own copies of data rather than share a single source of data. When a system uses a copy of the data, the applications in the system are assured of having the data in an understandable format. Then, if a change is made in the original data source, only a small change is necessary to the component that copies the data (usually the parser). The problem with this solution is that the object itself is not shared. In this solution, it is difficult to communicate between applications developed independently, because each application contains its own copy of the data and has no real reference back to the original data source, it is difficult to ensure that applications are appropriately updated or informed of changes in the data.

With a 3-tiered approach, each application can access information from all data sources within the same interface. The middle tier handles the brokering between the application's format and the data source's format. The example in Figure 3 describes the 2-tier problem and some of the benefits of a 3-tiered solution.

In our current client/server systems (2-tier architectures), we are required to modify each application to take advantage of new and evolving data sources. In addition, as we develop new client/server applications, they must be built with apriori knowledge of what data sources it uses. The 3-tier solution provides a mechanism to add new data sources without altering applications, and to build new applications without apriori knowledge of all the data sources it may want to access.

Figure 3: 2-Tier vs. 3-Tier Approach

SOLUTION

Our solution to the problem of sharing data in a collaborative environment where applications can dynamically access new data sources is to provide a set of communication services that are usable by existing and newly developed applications. These services will enable the application to access and share local and distributed objects.

Applications display and change data. The data can come from numerous sources. Although the application has to display the data, it has no apriori knowledge of where or how the data is stored. Applications use services to notify, find, access, and understand the data it is retrieving. Our architecture directly integrates the following services:

- Event service - publisher/subscriber mechanism of notifying data sources that information has changed
- Trader service (naming service) - a mechanism of finding where the data source is located
- Security service - a mechanism for controlling access of the shared information
- Dynamic mapping - a mechanism for understanding the data being passed, mapping information from domain attributes to presentation or application attributes.

![Collaborate Diagram]

**Figure 4: Services needed for collaboration**

Foundation services are the underlying software used for sharing data. CORBA and DCOM have much of the same functionality, while RMI is simply a mechanism for invoking remote Java methods. Web servers and object oriented databases are also considered foundation services.

**Applications**

The solution is to put a layer of abstraction between the data and the applications without hindering performance. This abstraction layer is a derivative of the presentation-abstraction-control pattern [Buschmann et al 1996]. The next step is to provide a mechanism for sharing the data in a distributed environment. Using Distributed Object-Oriented techniques, we are able to provide a mechanism for sharing information at its source without having to copy the data. Distributed object techniques are often described to work with objects stored in an object-oriented database (e.g., POET, Object-Store) or defined in an object-oriented applications (e.g. one written in C++, Java, small-talk, etc.). We can also provide a mechanism for distributing information in relational databases and legacy applications.

We look at a system function in terms of three components: Application, abstraction, and data source. Each is a distinct component and is likely controlled and developed by different groups running in a distributed environment. The application accesses the data in the same manner whether running in a local or wide-area network. The difference between a local and wide-area network is security and performance. Site administrators control the access to local-area networks, through the use of firewalls or closed networks. Performance of the network in accessing the information in wide-area networks is beyond the control of the local users.

In today’s distributed environment, we expect the capability to display a xy graph to be in a separate application (or applet) from a component that browses a table. The architecture must be flexible enough to support multiple design approaches by various software developers. Building a generic, solid, and well-documented framework lowers the life cycle and operation costs of newly developed systems by providing a common application framework. This framework includes the communication layer for accessing and sharing data, as well as templates for data sources and applications.

This distributed architecture allows C4I systems to move beyond traditional stovepipe applications and function as seamless, integrated, and highly responsive sets of cooperating tools. The abstraction layer (as shown in Figure 3) provides a mechanism for applications to dynamically determine what data is available and its format. Applications are aware of the visual attributes to be displayed (e.g., a map needs latitude, longitude, name), and are able to dynamically determine the attributes of the data (e.g., a message has a header, time, location). Users and administrators can map the domain attributes to the application specific attributes through easy to use graphical tools. The users can therefore easily display information from new and evolving data sources after simply defining the attribute mapping.

![Consistent Data Presentation Diagram]

**Figure 5: Consistent Data Presentation**

There are more benefits from this dynamic interactive architecture than simply being able to take advantage of new and evolving data sources. There is an inherent mechanism for collaboration at multiple levels. Users not only collaborate on the same data elements, but can also share information as they are creating it, providing built-in, white-board capabilities. All applications access the same data source, but potentially display it differently depending on the application and the context. Figure 5 shows how two users may access the same piece of information in different applications. Notification events describe how updates and highlights may be handled.

The abstraction layer is different than a client library connecting to a server, because the location of the referenced
object can be anywhere on the network. For example, an application may create and contain the referenced object being used by multiple resources. Although the distributed object may not be persistent (and therefore, not stored in a database), it can still be shared with other applications and services. For example, a mapping system may display objects from multiple sources (databases and applications). The objects from some applications may actually only live for a short time, as in the animation of a simulation. It is important for the architecture to enable these applications (e.g., the map and simulator) to communicate without having to understand the format of the data that the other application needs. This architecture supports the plug and play of data sources and applications through the use of application services such as dynamic mapping and event notifications. When a better map application becomes available, we do not want to have to rewire our other applications to handle the communication with the new mapping system. This is a chronic problem with today's systems.

Application Services

An event service is necessary for the client to receive notifications about changes in data from the server. A publisher/subscriber event service is recommended, so that the server does not have to maintain any state information about the connected clients. In a publisher/subscriber event service, each process registers with the event service. A process (e.g., the client) can then register for certain events from specific objects or objects from a given class. For example, a map may want to be notified when the location of an object is updated, or it may want to be notified when an overlay object is deleted. The alternative to an event service of this type is a polling mechanism. The client polls the server for known changes. The event service is a much cleaner mechanism as it limits the network traffic to only those notifications that each client has interest [Gill 97]. There are a number of publisher/subscriber event services available for CORBA. Some are provided with the ORB and others are available as third-party products. It is important to select an event service that all applications and data sources will use, as this is the common element for collaboration and data consistency.

The trader service and/or naming service is used to find a given source. The trader service has the capability to find a source by searching for information about the source (e.g., keyword search for a data source). The naming service is used by both CORBA and DCOM (called the registry) to locate a given data source by a published name.

The move to distributed systems increases vulnerability with more places to attack across the networks. In a distributed system with independent management, there may not be consistent policies, so it becomes difficult to set up and manage the distributed system securely. Security can be implemented using CORBA security via the transaction service, using DCOM DCE security technology, Java security, web security, and firewalls. Java accepts all downloaded programs and runs them within a security "sandbox". The sandbox is a fence that surrounds the program and keeps it away from private data. As long as there are no holes in the fence, the information is safe. Java security relies on the software implementing the sandbox to work correctly. Web security can be implemented through secure SLIP or PPP connections and firewalls. Although security in a networked environment leaves it more vulnerable to attacks, employing these security services provides threat protection.

Dynamic mapping is a mechanism used by applications to optimize the displayed information independent of how the data is stored. Therefore, the application should not be concerned with the structure of the domain data, as long as the domain attributes can be mapped to display attributes. This eliminates the need for application builders to consider how the domain data is stored when building the displays. Each component dynamically discovers the structure of the domain object and loads in the attribute mapping or queries the user for the mapping.

New data types and new data sources can now be displayed in the application software as soon as the attribute mapping is defined. This eliminates software changes to support new or changing data structures. The mapping between how a domain object is stored and how a client component displays an object is handled through graphical interfaces for and persistent storage of the attribute to attribute mapping. In the example described in Figure 6, the domain data is a company vehicle and the display data is an icon on the map. Location is stored in the database as a string and converted to lat/lon degrees for display purposes. Likewise, the map symbology selects an icon based on an image (logo) and vehicle.

Foundation Services

Distributed object technologies like CORBA, DCOM, and RMI provide the capability of accessing shared objects and
treat the objects as if they were local. This allows each application to actually manipulate objects and their attributes and have the other applications view those changes.

There are many debates about which distribute object technology is better: DCOM or CORBA [Chung 1998]. Both offer the user the ability to distributed objects between applications. Both are relatively easy to use. Both also offer the ability to remotely spawn an application to handle an inactive object (e.g. an object that is no longer available because its application has closed). The differences boil down to three categories: platform portability, robustness of available software, and future trends.

![Figure 7. Communication Mechanisms](image)

The CORBA standard has been developed out of a large consortium of over 200 companies. Although, CORBA is not currently a DIIOE standard, the ORBIX implementation by IONA has been selected as a potential extension. DCOM supporters question the future of CORBA. However, the standard is still going strong and has many more vocal supporters than DCOM. CORBA is both platform and language independent [Stal 1998].

DCOM is based on DCE/RPC. It is a Microsoft product. It works well on Windows NT and Windows 95. There are some third party vendors porting DCOM to other platforms, but DCOM is still basically a Microsoft product. DCOM supports multiple OO languages.

Java’s Remote Method Invocation (RMI) is another alternative, although this distributed object mechanism is limited to Java applications. It is platform portable, as is Java, but not language portable.

Despite the controversy, selecting the appropriate solution: CORBA, DCOM, or RMI often depends on the particular application [Jacobs 1998]. It is critical for a generic framework to support each of these communication methodologies. The dynamic attribute mapping capability described earlier addresses how objects are defined in other applications and mapped to objects understood by each application. Communicating with multiple data sources (applications and databases) and mapping these new data sources dynamically is the key to providing this flexible generic architecture.

Web servers and object-oriented databases are also part of the foundation services. The architecture is generic enough to support any robust web-server or object-oriented database. Using JAVA and ODBC standards limits the dependency on a particular implementation of either of these choices. Although in the case of an object-oriented database, an application may use implementation specific code to optimize the access to the particular database.

**CONCLUSION**

The architecture demonstrates how applications can seamlessly access existing, new, and evolving data sources in multiple types of applications in a distributed environment. The architecture supports sharing of data instead of duplicating data, and provides a level of collaboration between C4I applications that does not exist in current environments. Existing systems (e.g. Common Analysis and Reporting Environment, and WebC4I) have used this architecture to provide a distributed C4I framework to integrate new and legacy applications in a collaborative environment.

This architecture provides a mechanism for sharing objects in a collaborative environment and eliminates the dependency of apriori knowledge of data sources. The benefits are: a longer life span for legacy systems, reduced life-cycle management costs, and a mechanism to change data and/or applications without effecting other system components.

**REFERENCES**


