December 1998

Web Based Development Using n-tier Technologies

Paresh Chobhe
Temple University

Munir Mandviwalla
Temple University

Web Based Development Using n-tier Technologies

Paresh Chobhe
Munir Mandviwalla
Computer and Information Sciences
Temple University

Introduction

This report from the Comparison and Evaluation Laboratory (CEL) of the Computer and Information Science department at Temple University compares web-oriented n-tier systems development architectures. The focus is on comparing the Distributed Component Object Model (DCOM) with Java + Common Object Request Broker Architecture (CORBA) approach. The comparison is grounded on the experience of a team solving a benchmark task using both approaches. The benchmark task focuses on creating a browser accessible "contact management system" that keeps track of the relationships that a large company has with customers, brokers, and individuals. The benchmark task includes elements of transaction processing and workflow that are applicable to many other organizational tasks. The report is different from reports issued by consulting organizations and reviews by magazines:

- Based on an in-depth application of each technology to a specific task
- The students and instructor discussed and implemented the technologies in a structured academic setting.

Benchmark Task

Our benchmark/reference task is a sales force automation and contact management scenario for a large multinational organization. A browser accessible "contact management system" that keeps track of the relationships that a large company has with customers, brokers, and individuals. The users of the system include salespeople who will record and review contacts, managers who will review the information, and general management of the company. Specifically, brokers and managers distributed in time and place need to add/change/edit customers, agents, contact events, and relationships. The important high level entities include customer, broker/agent, region, products, salespersons, business relationships, and contact events. We assume a large sales force (5000+), multiple regions (10+), many customers (10000+), and large volumes (10 customer contact a day/person). From the hardware and software perspective, we assume that users have access to modern browsers, but we cannot specify the type of browser. The speed of connection can vary from 28.8 dialup to high speed T1. On the server end, we assume almost unlimited flexibility in hardware and software. The type of inquiries and reports can include customer search, broker search, customer information, branch report, and others. Security is an important consideration given the potentially high exposure. Other considerations are volume, performance, ease of development, and control of projects involving many developers.

The Technologies

Given the complexity and performance requirements of our benchmark task an n-tier-distributed approach is the logical choice. Sophisticated web-based applications are generally n-tier applications because completing a request spans at least three different processes:

a. Client Browser
b. HTTP Server
c. Application (Active Server Pages, CGI and so on)

Currently, the main technologies for n-tier web based development are Distributed Component Object Model (DCOM) based applications and the Java + Common Object Request Broker Architecture (CORBA) combination. DCOM is from Microsoft, the current market leader in computing, while Java has become an significant force in application development especially in distributed applications that can leverage its "built for network" design. Java has its own distributed computing architecture called remote method invocation (RMI). However, RMI has received little support in the industry given that CORBA is a relatively well understood and perhaps more reliable technology. We believe that DCOM and Java + CORBA represent the two most viable architectures for distributed computing. The remainder of this section presents a brief overview of each architecture, and the individual pieces that we focused on as part of "solving" our benchmark task.

COM/DCOM

The Component Object Model (COM) is an object based programming model designed to allow development of software components using a variety of languages, tools, and platforms. It allows inter-process communication between binary components running on the same machine. Distributed Component Object Model (DCOM) is an extension to COM wherein objects can communicate over the network. In the case of COM the client object interacts with the server object that is residing

on the same machine. However, in the case of DCOM the server object can be residing on any computer on the network and is accessed by the client using remote procedure calls. The client object does not need to know where the server object resides on the network. Also the code to execute a remote object is similar to executing one locally. Microsoft Transaction Server (MTS) is a key piece in DCOM based development. MTS provides rollbacks if DCOM based components fail, coordinates the creation, maintenance, and destruction of components, and provides resource pooling, security, and administration. The MTS layer also simplifies the development and deployment of DCOM based applications. The following outline shows how DCOM based applications are created with Visual Basic 5.0 Enterprise Edition, Microsoft Transaction Server, and Internet Information Server:

1. Create a server Active X Component in VB. This component will expose specific methods (e.g., retrieve data). The component can use specific Transaction Server services by referencing the Transaction Server libraries.
2. Register the component as a "package" in Transaction Server.
3. Create a client application (e.g., as an Active X document that runs inside Internet Explorer). The client can reference the exposed properties of the server component.
4. Create a setup program for the client application. During the setup process refer to special files created during the creation of the server component. This will enable the client application to find the server component over the network. Also specify the location of the server machine.
5. Run the client (e.g., by downloading from a web server). The client will automatically find the server component and ask it to run the exposed methods, the server will return the results to the client, which will then display them on the screen.

Java/CORBA

Java is a well-known programming language that is suited for distributed development. Two key features are the ability to "write once and run anywhere" and the built in support for the network. CORBA was designed to allow intelligent components to discover each other and interoperate on an object bus. CORBA objects are blobs of intelligence that can live anywhere on a network. They are packaged as binary components that remote clients can access via method invocation. Both language and compiler used to create server objects are totally transparent to clients. The key element in the marriage of Java and CORBA is the use of IIOP as the underlying communications layer and the availability of robust middleware or transaction monitors to enable n-tier development. Communications is accomplished directly in Java by writing sockets program. However, these programs will have to start at a very low level and will not have the well-tested location transparency, security, and ease of development provided by CORBA brokers. The following outline shows how Java/CORBA based applications are created with Java 1.1, Orbix Object Broker, and Orbix Web middleware.

1. Write in a special interface definition language (IDL) and describe the interfaces to the object or objects that are intended for use.
2. Compile the IDL using the IDL compiler provided by the particular ORB (e.g., the Orbix ORB). This produces the stub and skeleton code that implements location transparency. Identify the interfaces and classes generated by the IDL compiler needed in order to invoke or implement operations.
3. Write code to initialize the ORB and inform it of any CORBA objects that have been created.
4. Compile all the generated code and the application code with the Java compiler.
5. Run the distributed application.

Evaluation

The factors to be used for evaluation were first discussed internally and with our project sponsor. A short list of factors was identified and discussed further. The final list includes five major categories:

- Getting started (installation, quality of documentation, learning curve, cost)
- Development (user interface, programmatic control, time to develop, debugging facilities, stability, and performance)
- Features (scalability, security, database connectivity, database manipulation, team programming and version control)
- Distribution (control versions in the field, file sizes, accessibility, manageability)
- Meet task requirements (i.e., how well the technologies met the individual requirements)

Because of space restrictions we provide below a short synopsis of our results.

Getting Started. The DCOM solution was very easy to install. However, the quality of the documentation was poor, and there was very little of it available. Understanding how the various pieces work together to develop an n-tier web application was confusing. Multiple ways of doing the same task further complicated things. It took us an incredibly long time to get past what we call the "active soup" of terminology that these technologies introduce. For example, Active X Exe, Active X Component, Active X Document, and Active X Control are all different but yet very similar technologies. The installation for CORBA and Java was also fast, but making it work was more challenging. If the developer understands C++ then Java is relatively easy. Conceptually CORBA is also relatively easy to understand because the architecture is laid out in using logical and consistent terms.

Development environment: The VB 5.0 development environment was easy to use. Most of the time was spent on testing because of the complexity of each layer in the application. Because VB, Transaction Server, IIS are not fully integrated, debugging is difficult. All the technologies at least from the development perspective are very stable. Compared with above, the

user interface of development tools for Java and CORBA are poor. There are various tools for simplifying Java programming but during the time of our project they either concentrated on Java 1.0 or they implemented their own classes for user interface widgets requiring the user to download very large class files. Debugging in this environment is also painful.

Features: DCOM brings with it the powerful security features of Windows NT along with the additional security (role based component access) provided by MTS. Database manipulation and connectivity were relatively simple given the easy availability of ODBC based connections to databases. Visual Basic provides a strong team programming and version control features coupled with Visual SourceSafe. The Java/CORBA solution was more difficult. Not very long ago there was no way to easily connect a Java program to a Database. Currently, Java 1.1 provides JDBC (analogous in use and concept to ODBC). But most database vendors do not support JDBC at the back end. We suspect this will change soon. We had to use a Java-ODBC bridge with unknown performance implications.

Deployment: The complexities of each type of user interface (Active X EXE/DLL, Active X Document EXE/DLL) and server components (Active X DLL/EXE) made the task of deploying the application very confusing. Though the setup wizard walked us through the steps, each step was not clear enough in what it was doing and what would be the end result. A key problem is that Active X controls and documents only work inside Internet Explorer. The big advantage of the Java/CORBA solution is that it is portable to any system without too much effort.

Task requirements**:** Our specific task requirements include 24 hour access to data worldwide, volume, use from multiple platforms, use on low speed connections, ad-hoc use, security on the front end, and security on the back end. We feel that both architectures are still too new to assume 24-hour access and large volumes of processing. The most well known and reliable piece in this equation is the CORBA specification but its integration with Java is still a work in progress. On use from multiple platforms, the Java/CORBA architecture is the clear winner. However, if organizations are forced to use too many classes from different vendors for small problems then the resulting download time will be high. The DCOM solution assumes a certain level of infrastructure on the client and thus may be better for low speed access. We were unable to fully explore most of the security issues. However, we suspect that both architectures are comparable.

Conclusion

In conclusion the major problem with the DCOM approach is the poor documentation further hampered by an overuse of terminology that confuses even experienced developers. The major problem with the Java/CORBA approach is poor integration, which is not surprising since the ORB vendor and the Java vendor tend to be different. Both technologies hold considerable promise for the future. Perhaps what is most interesting is that at the end of the day the architecture and approach of both set of technologies is almost exactly the same. However, it takes considerable effort to peal away the terminology and see the similarities.

During the AIS presentation we will present a much more detailed explanation of each technology, the tools used, the differences and similarities, and the pros and cons. In addition, the presentation will include a detailed walkthrough of developing a sample application. A detailed report will also be posted on the CEL website at http://ww.cis.temple.edu/cel.

Acknowledgement

We would like to thank Paul Weinberg of CIGNA Corporation for his participation and support of this project.

References

References available upon request from (first) author (pchobhe@thunder.ocis.temple.edu).

Disclaimer

The evaluative comments in this report are based on a project conducted in an academic setting by faculty and students. The evaluative comments do not necessarily represent the opinions of our sponsor(s).