Considerations for Partitioning Application Activities in a Multi-Tiered Environment

Kurt Jordan[1]
Computer Information Systems and Information Technology
Purdue University Calumet
Hammond, Indiana 46304

Abstract

Web services, a type of multi-tiered application, is gaining in popularity. With any type of multi-tiered application, decisions are made concerning the partitioning of application activities. This paper describes the considerations that should be taken into account when deciding on which tier a particular activity of a multi-tiered application should be placed.

Keywords: leading edge and emerging technologies, E-Commerce, E-Business, multi-tiered architecture, distributed systems

1. Introduction

Web services and other types of multi-tiered applications continue to group in popularity. Gartner expects that J2EE, one of the more popular development environments for Web services, is entering into two years of growth in use (Driver, 2003). Developing multi-tiered applications is a complicated activity. The developer is presented with many choices to make during the development process. One such choice is on which tier a particular module of the application should reside.

2. Multi-tiered Applications

A multi-tiered application has its activities separated into tiers. Each tier runs on a separate computer platform. A Multi-tiered application can be identified by the number of tiers it has. An application composed of two parts each of which run on two separate computers is called a 2-tiered application. An application composed of three parts each of which run on three separate computers is called a 3-tiered application.

Each tier in a multi-tiered application is typically responsible for some aspect of the application. The 1st tier handles presentation activities, such as application startup and connection functions, activity selection functions, data input and data validation, and output display and disposition. (Ben-Natan, 2002) The 1st tier also controls the flow of information between 1st and 2nd tiers.

The 2nd tier handles general application processing, such as data transformation and manipulation, executing business rules, controlling process flow and proper sequencing of events. The 2nd tier also controls the flow of information between the 2nd and 3rd tiers. The 2nd tier typically has access to other services not directly available to the 1st tier, such as security, communications, encryption, process synchronization, and access to other networked services.

The 3rd tier handles requests for data services, usually in the form of a relational database management system, such as DB2, Oracle, or SQLServer. This tier handles activities such as data queries, updates, deletes and insertions. It may also execute stored procedures on the data.

3. Usual and Customary Application Activities

The sequence of events in the execution of a simple multi-tiered application starts with

the 1st tier. Execution begins with application startup and initial setup activities. The user would then select the desired service to be performed, by choosing a menu option or clicking a hyperlink, for example.

Any necessary information or data would be supplied and checked for validity or reasonable values. The data is packaged in the required format and sent to the 2nd tier. The 1st tier can either wait for the results to be returned, or continue with other activities. When the results are ready, the 1st tier can format and display the results, save the results to long term storage, or otherwise dispose of the results as specified by the user. The 1st tier typically runs on some kind of desktop computer. This desktop computer is frequently called a client.

When the 2nd tier receives the service request and data from 1st tier, it can perform checks for validity. This may not be necessary if checking occurred in the 1st tier. The 2nd tier may issue requests for any necessary data to the 3rd tier. Next, the 2nd tier executes the requested service. More data access requests to the 3rd tier might occur as necessary. This first or primary service could also call other services, both locally and on other computer systems. In this case, any required data would be packaged in the proper format and sent to the other, secondary service for processing.

When the results from the secondary service are returned, the primary service continues. When the primary service is finished, the results are packaged in the proper format and either sent back to the 1st tier, or sent to a central repository for later retrieval. The 2nd tier typically runs on a server computer with more processing power, disk storage and memory than a typical client. This server computer would be configured to handle multiple simultaneous requests for service.

The 3rd tier is responsible for executing data service requests and returning data to the 2nd tier. The 3rd tier can execute some application logic on the data in the form of stored procedures or triggers. Like the 2nd tier, this tier typically runs on a more powerful server computer with the capability to handle multiple simultaneous requests for data service.

4. The Partitioning Dilemma

The previous descriptions are what traditionally happen in each tier of a multi-tiered application. In actual practice, the developer has much freedom to combine activities on the same tier, or separate activities onto different tiers. The 2nd tier business logic could be executed on the client along with the 1st tier activities. In this instance, tiers 1 and 2 reside on the client and tier 3 would reside on a separate database server computer. This is an example of a 2-tiered application with a fat client.

In another instance, the 2nd and 3rd tier activities could be combined on the server computer, with the 1st tier activities remaining on the client. The application could do the data editing tasks on the server computer system as well (although this is considered a trivial processing task that today's desktop computers can handle with little trouble). This is an example of a 2-tiered application with a thin client.

The developer must make the decisions concerning on which tier a particular activity will reside during the design phase. IBM's application design guidelines indicate that systems "can fail to meet initial requirements, such as performance, because the system was not optimized for the chosen environment" because the placement of resources is determined independently, instead of in an integrated manner. (IBM, 1991) The number of possible combinations is influenced by several considerations. Prudent developers will study the considerations and use them to arrive at an optimal solution to the application requirements.

5. Designing m-Tiered Applications

Before assigning application activities to specific tiers, it is necessary to develop the application at an abstract level so that particular application activities can be identified. Then, the activities can be assigned to their respective tiers. Prudent developers designing multi-tiered applications use some sort of methodology to design and develop the application. This paper does not try to support or attack any design/development methodology used to identify the activities the application should be capable of per-

forming. It is addressing only the considerations that could be used to assign already-identified modules to a particular tier.

In order to illustrate how the activities used in the forth-coming example were arrived at, assume the task is to develop a web-based customer information management application. The application will allow the addition of new customers, the display of existing customer information, the updating of customer information and the removal of a customer.

We start by making a first pass through the application and identifying the activities the application will perform. This is done at a high, abstract level. Determining on which tier an activity is placed does not require a very high level of detail. This first pass might result in the following list of activities:

user logon
user logout
add new customer
display customer detail
update customer detail
remove customer

The developer would make another pass, this time providing slightly more detail to identify the individual steps needed to carry out that activity. Again, this would be done at a high, abstract level. At this point, the developer is starting to move toward the detailed program design. It is helpful to think of most computerized activities as being divided into three general steps: prepare to perform the activity; perform the activity; perform post-activity functions. Preparing to perform the activity consists of functions such as data acquisition, data error detection and correction, and packaging data into proper format. Post-activity functions can be reporting the results of the activity (success or failure), and disposition of any resulting data or reports.

For example, looking at the *add new customer* activity, you would identify the individual steps that would have to be performed. Preparing to add a customer would involve the input of new customer data and the verification or validation of that data for acceptable values. If the input data pass the validation steps, the new customer is created. Finally, the results of the creation request are displayed. The slightly more detailed steps to add a new customer might be:

add new customer
    accept new customer data
    verify valid data values
    if data valid
        insert data into database
    display results of insertion request

Notice the steps still describe what will happen at an abstract level. Some or all of the steps can still be defined in more detail. Continue the same steps with the other high level activities previously listed above. A third and subsequent passes will provide more detail until the developer identifies all the activities the application should do. However, the developer still wants some level of abstraction to preserve the flexibility to use whatever resources or languages that might be available on a particular platform.

6. Assigning Modules to a Tier

After identifying the application activities, the next step is to look at the activities and organize them by tier. Any data input, data validation, output disposition, connection login/logout, and user customization activities are done in the 1st tier. Business processes and any compute-intensive processing or activities requiring data access are done on the 2nd tier. Actual database access is done on the third tier.

Returning to the create new customer example, the data acquisition and validation steps would be done on the 1st tier. The validated data would be sent to the 2nd tier, which would be responsible for requesting the customer data be inserted into the database. The insertion activity would be carried out on the 3rd tier, most probably using SQL. The results of the insertion request would be returned by the 3rd tier to the 2nd tier and then to the 1st tier, where it would be displayed to the user.

add new customer
    1st tier: accept new customer data
    1st tier: verify valid data values
    1st tier: if data valid
    2nd /3rd tier: insert data into database
    1st tier: display results of request

Each activity in the application should be analyzed in a similar manner to identify on which tier a specific step would be performed.

7. Reassigning the Modules to Other Tiers

In a perfect world, each tier of a 3-tier application would reside on a separate computer system. As mentioned earlier, the 1st tier typically runs on a client. The 2nd tier runs on a more powerful application server computer. The 3rd tier runs on a more powerful database server computer.

In reality, it might not be possible, or desirable to partition the application to run on three different computers. The number of available server computers might be limited. The business process logic (2nd tier) and database management software (3rd tier) might run on the same computer system. The computer on which the database management software runs might not be powerful enough to support both the RDBMS and the business process logic, so the business process logic might run on the client along with the 1st tier logic.

Three basic configurations for a multi-tiered architecture are:

> Tier 1 contains user interface and business process logic, tier 2 contains database access – 2 tier architecture
>
> Tier 1 contains user interface logic; tier 2 contains business process logic and database access – 2 tier architecture
>
> Tier 1 contains user interface logic, tier 2 contains business process logic, and tier 3 contains database access – 3 tier architecture

The first two reflect a 2-tier architecture. The third reflects a 3-tier architecture.

The reasons why a developer would choose one of the previous three basic configurations are numerous. The next section begins to examine some of these reasons, beginning with the 1st tier.

8. Reasons to Place Business Processes on the Client

For financial, logistical or political reasons, there might not be an application server computer available to host the business processes. You might have only a handful of clients and a centralized database on a database server computer with which to work. In this case, the choice is between running the business logic on the database server computer or on the client. If the database server computer is not configured to handle the extra load of the business logic as well as the database activity, the only option is to place the business logic on the client.

Perhaps the business logic takes minimal processing power. You might decide that managing customer records does not require heavy CPU or memory resource. The computing power available on the client is adequate to handle the processing needs for this application, so the business logic could be placed on the client with little impact on application performance. However, data will have to move from the database server computer to the client, potentially increasing the load on the network, and raising security and data integrity concerns.

Yet another consideration for placing the business processing load on the client is the number of potential users. The number of users of the application might be so small that the expense of a separate application server computer is not justified. It might be easier or less expensive to simply upgrade the few clients that would use the application so they have the processing power and memory capacity to handle the computing requirements.

In spite of having a very powerful application server computer available, the load on that server might be such that the performance of the application is unacceptable. An application that was initially designed as a true 3-tier application might enjoy better performance if some of the 2nd tier activities are moved to the client. These types of situations might not be identifiable during the design and development phase, but should be in the minds of the developers. Some testing could be done to determine response times on the application server computer and the results extrapolated using queuing

theory to get a ball-park figure of the maximum number of simultaneous processes the proposed application server computer can tolerate before performance degrades below acceptable levels.

9. Reasons to Place Business Processes on an Application Computer

If the application has strict security requirements or needs, you could better control them by centralizing the secure components of the application on an application server computer. Application components with minimal or no security requirements can be safely placed on the client. In some situations, security needs are more important than issues such as performance. For example, even though the computing requirements for the application are low, meaning they could safely be done on the client; transporting sensitive information down to the client for processing might not be acceptable from a security standpoint. So, the decision might be to use an application server computer for the added security even though the processing load would be trivial.

Security concerns and other complicated business rules might cause data editing and validation activities to be placed on the second tier. For example, suppose the application uses a browser that contains a HTML form with embedded JavaScript to perform data editing and validation. A knowledgeable user could save the page containing the form and edit it to bypass the JavaScript validation code and then send unvalidated data that contains errors to the second tier for processing. The second tier might be written to expect validated, correct data values. One best-case scenario is bad values are entered into the database. One worst-case scenario is the application crashes on the application server computer.

The application might have special process synchronization requirements necessary to ensure integrity - record locking, or special sequences of steps that must be done as an atomic, uninterruptible unit. The synchronization would need to be controlled on a centralized computer system. The business logic might require communication with other services on other computers that are not available to the clients. That communication would require the business logic be done on a computer that has access to the necessary communication resources.

The use of modules in a compiled language vs. stored procedures in a database also might determine where business processes execute. Compiled languages place the processing load on the application server computer, or on the client. You could do some performance tuning by distributing the load more evenly among several application server computers, or further partitioning the business activities between the client process and the application server process.

As a rule of thumb, activities such as sharing of data items among different application modules, enforcing a required level of security, or enforcing an ordered sequence of events to ensure integrity are activities that should be done on the application server computer - in the 2nd tier.

10. Reasons to Place Business Processes on the Database Server Computer

Many of the same reasons for placing the business-processing load on the client apply to placing the load on the database server computer. No separate application server computer is available. Existing clients might not be powerful enough to handle the increase in processing power or memory capacity needed to carry out the business processing logic. Upgrading the clients to handle the load might not be feasible for several reasons. Too many clients would have to be upgraded. The clients might be too old to be upgraded and would have to be replaced.

The business logic might be in the form of stored procedures in the database. These stored procedures execute on the database server computer. Many third party software systems come with stored procedures. Customers who purchase such third party software have no choice where to place them. This forces the business logic onto the database computer, making a 2-tiered application out of what might have been originally intended as 3-tier. The application server and client are idle while stored procedures execute.

11. Network Traffic Considerations

Compared to using typical interprocess communication techniques, using the network to communicate between application modules is slow. The application can expect relatively poor performance, especially with high volume of network exchanges. The impact is even more pronounced when modules communicate with physically remote locations. The remoteness can cause significant network delay. Control over application response times in these situations might not be possible.

The developer should consider re-grouping modules that communicate often over the network to the same tier if the number of exchanges is significant enough to influence response times. If the modules can't be relocated, consider re-organizing processing activities so data can be grouped together and sent as a bundle, instead of a few data items at a time. The decision whether or not to do this might be affected by other concerns such as security or the sequential timing aspects of when the processing of a particular set of data occurs.

12. Non-Technical Considerations

As with most computerized projects, cost can have a big influence on how the applications are developed. One benefit of using very powerful server computers is to share scarce, expensive resources among many users. The goal is to decrease costs, using a few powerful servers instead of many powerful desktop machines. The presence of an application server computer gives the developers more flexibility in deciding on which tier application modules will reside. The absence of an application server limits the developer's choices.

Political issues can also influence application development. Questions of ownership of the computing equipment or the data can complicate application development. Some won't want to give up control over what they see as their data or computing resources. An unfortunate result is that the application could be spread across multiple computing facilities even though that is not the most efficient design. Or the application might not take advantage of the existence of computing equipment that would have provided better performance.

13. Conclusion

Designing multi-tiered applications requires special attention by the developer. Several considerations exist for placing business processing logic on the 1st tier, on the 2nd tier and on the 3rd tier. Cost, performance, processing power, network load, security, process synchronization and political issues can all influence the decision. Prudent developers will use these considerations to make intelligent decisions on proper placement of activities in a multi-tiered application.

14. References

Ben-Natan, Ben and Ori Sasson (2002), IBM WebSphere Application Server, McGraw-Hill, page 658-659

Driver, Mark, (May 30, 2003), Strategic Analysis Report, Note Number R-20-0812, Hype Cycle for Application Development, Research and Advisory Services, Gartner Group

IBM, (September, 1991), International Technical Support Centers Red Book, Client/Server Computing Application Design Guidelines: A distributed Relational Data Perspective, page 81

[1] jordank@calumet.purdue.edu