Implementation of a Data Driven Web Application using the Distributed
Internet Applications (DNA) Architecture

Rahul Rana

Submitted to Committee Members
Dr. W. Homer Carlisle
Dr. Kai-Hsiung Chang
Dr. Wen-chen Hu

Department of Computer Science and Engineering
Auburn University
June 8th, 1999
Acknowledgement

I would like to thank Dr. W. Homer Carlisle for his supervision, guidance and encouragement during the many discussions we had in exploring the filed of DNA. I would also like to thank Dr. Hu and Dr Chang for their guidance and suggestions.
VITA

Rahul Rana, son of Dr. H.S. Rana and Raj Rana, was born May 20th, 1975, in New Delhi (India). He entered R. V. College of Engineering (Bangalore University), India, in June 1993, and graduated with a degree in Computer Science and Engineering in June 1997. He entered the Graduate School at Auburn University, in April 1998.
Contents

Chapter 1 Introduction .................................................................................................................. 1
  1.1 DNA Methodology ............................................................................................................. 2
  1.2 Benefits of DNA ............................................................................................................... 3
  1.3 Distributed Client/Server Applications Vs DNA .............................................................. 4

Chapter 2 Three-tiered System .................................................................................................... 7
  2.1 Three-Tiered Strengths ...................................................................................................... 7

Chapter 3 MTS Overview .......................................................................................................... 9
  3.1 Application Layer (First Tier) .......................................................................................... 10
    3.1.1 Role of IIS and ASP .................................................................................................. 11
  3.2 The Middle-Tier .............................................................................................................. 11
  3.3 The Bottom-Tier ............................................................................................................. 11
  3.4 Building Components ..................................................................................................... 12
  3.5 MTS Component Code Template .................................................................................... 13
  3.6 Using components without MTS ..................................................................................... 13
    3.6.1 The problem with holding Component Instances ..................................................... 14
  3.7 Providing Persistent Context for Components .................................................................. 15
  3.8 How MTS provides Persistent Context .......................................................................... 16
  3.9 State and the Component Context .................................................................................. 18
  3.10 Object Life under MTS .................................................................................................. 19
  3.11 Resource Pooling .......................................................................................................... 21

Chapter 4 Transactions with MTS ............................................................................................. 21
  4.1 Transactions and distributed components ........................................................................ 22
  4.2 Managing Transactions .................................................................................................... 22
  4.3 MTS and Database Transactions ...................................................................................... 23
  4.4 Initiating a transaction ...................................................................................................... 23

Chapter 5 MTS benefits and Comparison with Other Technologies ........................................ 25
  5.1 MTS Benefits .................................................................................................................. 25
  5.2 MTS Compared with other technologies .......................................................................... 26
    5.2.1 CORBA ..................................................................................................................... 26
    5.2.2 Enterprise JavaBeans .............................................................................................. 27
  5.3 Disadvantages of MTS .................................................................................................... 28
Chapter 6 Project Specifications

6.1 Administrators’ Resources

6.2 Faculty Resources

6.3 Students’ Resources

Chapter 7 Conclusion

Bibliography

Appendix A
1 Introduction

Both small and large companies have been building robust applications for personal computers that continue to be more powerful and available at a lower cost. While these applications are being used by millions of users each day, new forces are having a profound effect on the way in which software developers build applications today and the platforms on which they develop and deploy their applications.

The increased presence of Internet technologies enables global sharing of information between small and large businesses and individuals as well. The Internet has sparked a new creativity, resulting in new businesses popping up overnight, running 24 hours a day, seven days a week. Competition and the increased pace of change are putting increasing demands for an application platform that enables application developers to build and rapidly deploy highly adaptive applications to gain strategic advantage.

It is now possible to think of these new Internet applications as needing to handle literally millions of users—a feat difficult to imagine just a few years ago. As a result, applications need to deal with user volumes of this scale, be capable of reliable 24-hour-a-day operation and be flexible enough to meet changing business needs. The platform that underlies these types of applications must also provide a coherent model along with a set of infrastructure and prebuilt services for enabling development and management.

The Microsoft application platform consists of a multitiered model called Windows Distributed Internet Applications (Windows DNA) and a comprehensive set of infrastructure and application services. Windows DNA unifies the best of the services available on personal computers, application servers and mainframes, the benefits
inherent in client-server computing and the best of Internet technologies around common, component-based application architecture.

1.1 DNA Methodology

DNA methodology is really an umbrella term that covers many existing technologies to help us design and implement robust distributed applications. It visualizes this whole application as a series of tiers, with the client at the top and the data store at the bottom. The core of DNA is the use of business objects in a middle tier of the application, and this is sported by two new technologies: Microsoft Transaction Server (MTS), which is a component manager offering full transaction support; and Microsoft Message Queue Server (MSMQ), which provides the fault tolerance required in a distributed application. Together with existing technologies, these make it simple to build a DNA System.

In DNA, business objects are developed as software components. These components can be accessed by the client interface application or by another component, and can themselves call on other components, data stores, etc. Componentization of business rules brings many benefits, such as easier maintenance, encapsulation of the rules, protection of intellectual copyright, etc.

So, DNA is an approach to design that can speed up overall development time, while creating more reliable and fault tolerant applications that are easily distributive over a whole variety of networks.
1.2 Benefits of DNA

- **DNA helps to design and build multi-tier client/server applications [4]**
  DNA provides a structured approach to creating applications whose components are clearly separated into distinct functional groups, with common communication protocols linking these groups. This provides the benefits of faster and less error-prone design and implementation and development, and interchangeability of components.

- **DNA provides client transparency [4]**
  No matter what the 'back end' of the application does, or how it does it, the front end (or client) needs no knowledge of this. As long as it follows the DNA protocol and processing guidelines, the client can almost be anything - from a standard web browser to a specially developed application written almost in any programming language.

- **DNA applications provide full transactional processing support [1]**
  In applications of any real level of complexity, multiple operations are performed at different levels of the application, and at different times. To guarantee integrity of the results, there needs to be a control over each set of operations as a whole, as well as monitoring of every individual step. DNA, and the associated software plumbing components, can accomplish this almost transparently and seamlessly.
DNA can be used to create applications that are fault tolerant [2]

Even in today’s world of reliable and high-availability communications, no network can ever be 100% guaranteed to give continuous and fast performance. A distributed application needs to be able to cope with network delays and software failures, while protecting data integrity and providing high availability and reliability.

DNA is ideal for distributing applications [6]

Once an application becomes distributed, i.e. divided into separate parts linked by a network, the problem of communication between the parts arises. In the past, the developers have often had to create their own custom formats and techniques for passing information between each component of the application, leading to longer design and implementation periods, an increased number of bugs, and poor interoperability between different applications. By standardization of the communication protocols and interfaces, and by removing the need for the programmer to be concerned with the plumbing that connects each part, developer productivity and application reliability are boosted.

1.3 Distributed Client/ Server Applications Vs Distributed internet Applications (DNA)

Client/server is an approach that lets the processing tasks of an application be divided so as to provide optimum efficiency. If there is of spare power available on the client machine, it makes sense to do most of the processing there.
However, if the company's requirements mean that only simple terminals are distributed across the enterprise, then the only place left to do the work is on the server – the client does no processing at all.

The processing itself can be divided up into different sub-tasks, such as the interface code, the business rules code, the database code, and between them the network operating systems. This is generally referred to as an n-tier application.

If we are running a dumb terminal or thin client network, all but the interface code runs on the server as shown in figure 1.1.

![Thin Client Diagram](image1)

**Fig 1.1: Thin Client**

On the other hand, if we were running a fat client network, the client would have the power to carry out processing of the information, so work is done on the client. Figure 1.2 illustrates the fat client model.

![Fat Client Diagram](image2)

**Fig 1.2: Fat Client**
On the internet, we have to adapt our traditional views of where the processing should take place. DNA provides a single solution to all our client/server problems. DNA provides a three-tier (or more) approach to application design, but in reality still follows the same broad pattern as existing client/server technologies. The difference is that it's designed with the Internet in mind, and can handle things that are traditionally difficult in this environment. In particular it defines business objects, which carry out all the tasks of accessing back-end data and ensuring that transactions take place properly. The DNA architecture is shown in figure 1.3.

Fig 1.3: DNA
2. **Three-tiered system**

An argument for three-tiered systems is as follows. By partitioning applications cleanly into presentation, application logic, and data sections, the result will be enhanced scalability, reusability, security, and manageability. Three-tiered approaches map well to a number of important application development challenges. Internet and Intranet application developers need easy ways to use browsers as the presentation interfaces to line-of-business and electronic commerce applications.

![Diagram of Three-Tiered System](image)

**Fig 2.1 A Three Tired Application**

2.1 **Three-Tiered Strengths**

Three-tiered architectures are often called server-centric, because they uniquely enable application components to run on middle-tiered servers independent of both the presentation interface and database implementation. While there is no requirement that each unit runs on a different physical machine, to deliver maximum benefit, three-tiered systems must run with presentation components on desktop
clients or browsers, application logic on middle-tiered servers, and data on dedicated
database servers. The independence of application logic from presentation and data
offers many benefits:

- Developers can use powerful development tools such as Visual Basic and
  Visual J++ to develop reusable application components, instead of using more
  limited stored procedure languages.

- Administrators can replicate application components to run on multiple machines
  simultaneously. This spreads client loads across multiple machines and enables
  higher availability, scalability, and performance. Application component
  replication (as opposed to data replication) is not possible with two-and-a-half-
tiered architectures, because stored procedures must run in a single database.

- Application components can share database connections. This lowers the
  number of total sessions that the database server must support, improving
  performance. With a two-tiered system, the database must allocate a connection
  for every user.

- Middle-tiered application components can be secured centrally using a common
  infrastructure. Access can be granted or denied on a component-by-component
  basis, simplifying administration.
3 MTS Overview

MTS provides a server-centric environment for developing and deploying three-tiered applications based on Microsoft's Component Object Model (COM) technologies. In MTS applications, application logic components run under the control of MTS on servers and are invoked by presentation-centric components running on clients via Microsoft's Distributed COM (DCOM) technologies. Clients can be a mixture of conventional applications and Active Server Page (ASP) scripts running within Microsoft Internet Information Server (IIS). This framework is shown in figure 3.1 [1]

Fig 3.1
Application logic components can access a number of different databases, such as Microsoft Message Queue Server (MSMQ), and, via Microsoft SNA Server 4.0, CICS and IMS applications. Access to databases and resources is done through MTS Resource Dispensers that perform services such as connection pooling automatically. MTS also supports automatic transactions so that access to data and resources is done with all-or-nothing protection.

3.1 Application Layer (First-tier)

The process of handling application requests from Web servers is more difficult than it sounds:

- Most Web servers provide little or no application infrastructure themselves. They can invoke an application or script, but it is up to the developer to provide basic services such as database access.
- Most application development technologies that provide good infrastructure services don’t integrate well with Web servers. For example, a common problem is maintaining information about users between browser requests in the connectionless world of the Web.

Conventional solutions to these issues rely on mechanisms such as Common Gateway Interface (CGI) scripts, which work well enough for basic applications, but are not scalable or robust enough for line-of-business or electronic commerce applications.
3.1.1 The Role of IIS and Active Server Page Scripts

MTS, combined with Microsoft Internet Information Server 4.0 (IIS), is ideal for building production-quality three-tiered applications with Web-based front ends. IIS provides state-of-the-art facilities for delivering visual and interactive pages to browsers and a powerful application mechanism called Active Server Pages (ASPs). ASPs are text files that contain a combination of standard HTML commands and scripts based on Visual Basic and JavaScript.

Web page designers use ASPs by building standard Web pages containing references to URLs with .ASP extensions. For example, it is common to send input forms to users and designate in the return URL an ASP that processes the response. When users fill in the form and send their response, IIS sees that the URL contains a .ASP extension and executes the appropriate ASP script. The script in the ASP file can do basic calculations, access databases via ODBC interfaces, and build an HTML stream to send back to the browser. Most important, ASPs can also call components running under MTS.

3.2 The Middle – tier

The middle tier would be where MTS comes into picture. It is here where we place our components. An in-depth explanation of this tier is explained later in this report.

3.3 The Bottom – tier

The bottom-tier is where the database is stored. The database could be a single one or we could use a distributed architecture where we have more than one database. The database could be anything from CICS, Oracle. Access to SQL server. For this
project we have used SQL Server 6.5. The components in MTS communicate with the database.

3.4 Building Components

We can build presentation and application logic components with any tool or programming language that can generate COM-compliant dynamic-link libraries (DLLs). For components to work efficiently within the MTS environment, we must observe several simple rules:

- Components must create a reference to their MTS Context Object by making a simple API call. Creating the reference enables the component to take advantage of MTS services such as transaction and security support.
- Do not save state information across transaction boundaries within components (e.g., in local or global variables). Components that save state are less scalable, because MTS cannot recycle their resources when they finish executing. State should be kept in databases or in the Shared Property Manager (SPM) in MTS and retrieved by components when needed.
- When a component completes execution successfully, it must call the SetComplete method on the MTS Context Object. This tells MTS that this component wishes to commit any work it has performed when all components involved in the transaction finish executing. Calling SetComplete also tells MTS that it can recycle any resources held by the component.
- If a component cannot complete executing successfully, it must call the SetAbort method on the MTS Context Object. This tells MTS that it should abort the current transaction and roll back all changes made by components
involved in the transaction. Calling SetAbort also tells MTS that it can recycle any resources held by the component.

3.5 A Sample MTS Component Code Template

When components follow these rules, applications can take advantage of MTS benefits such as enhanced scalability, performance, and management with no additional development. Figure 3.2 shows the structure of an MTS component. [6]

```
Set ctxObject = GetObjectContext()
(Application Code)
Set objfoo = ctxObject.CreateInstance()
(Application Code)
If (OK)
  ctxObject.SetComplete
Else
  ctxObject.SetAbort
```

Fig 3.2: MTS code template

3.6 Using components without MTS

When we use a component in an ASP page, for example, we have to create it before we can use it. The sequence of events is:

- Instantiate the component to create an instance of it
- Initialize the component instance, so that it is ready for use
- Use the component instance within our page
• Destroy the component instance after use

For example the following code in our project would create an instance of a component in an ASP page:

```vbnet
Set Obj=Server.CreateObject(“facult.addstudnen”) //create instance of the component
Obj.lname=Request.Form(“last-name”) // get the parameters
Obj.fname=Request.Form(“first-name”)
Obj.ssn=Request.Form(“ssn”)
Success=Obj.add // get result from component
'Rest of code
```

### 3.6.1 The Problem with holding Component instances

Although we don’t necessarily have to destroy a component, it is destroyed explicitly as soon as the last reference to it is lost—i.e. when our page finishes executing, the results have been sent to the client, and the ASP code is removed from memory. Because we held the reference until our page was complete, the component was retained in memory. Even while it’s not being used, it is taking up resources on the server.

In a multi-user environment, where there are high demands on server resources, it makes sense to destroy each instance of a component as quickly as possible. We could have improved the behavior of our page by adding a line that destroys it directly after the call to that method that provides the result:

```vbnet
Set Obj=Server.CreateObject(“facult.addstudnen”)
Obj.lname=Request.Form(“last-name”)
```
Obj.fname=Request.Form("first-name")

Obj.ssn=Request.Form("ssn")

Success=obj.add

Set obj=Nothing  // destroy the interface

'Rest of code

However, if we decide that we need to use the component again later in the page, we will then have to create a new instance. We will also have to set all the properties again, because the new instance will contain the default values. In this case we often tend to hang on to our object instead of explicitly destroying it, because it is far slower and much less efficient to keep recreating it.

3.7 Providing Persistent Context for Components

MTS gets around the problems of component creation and destruction by providing a pool of component instances in an intelligent and comprehensive way. MTS can provide a component to an application on demand, and allows other applications to use the component when the first one is just holding on to the component instance, but not actually using it. It does this by fooling the application into thinking that it still holds a reference to the component, when in fact MTS has spirited it away while the application wasn’t looking and given it to someone else.

If the application suddenly wants to use the component that it thinks it’s holding a reference to, MTS rushes around the pool of available instance and steals one from another application. Only if there are no available instances does it create a new one, and hand it over to the first application. This rather risky-sounding technique is called persistent context, and it works because MTS provides a substitute ‘dummy’
context to the application, which thinks this, is a real component. What it actually has, of course, is an empty shell.

3.8 **How MTS provides Persistent Context** [11]

To explain how MTS provides persistent context, let us consider a scenario where we have three clients A, B and C who want to at some point access a component that is on the server, providing instances to clients.

In the first stage consider clients A and B to be using instances of the component. Client C has not yet created an instance of the component. This is depicted in figure 3.3.

![Diagram of MTS operation](image)

**Fig 3.3**

What has actually happened here is that MTS has intercepted the calls from A and B that create the component instances, and instead created virtual context objects and returned references to these, rather than references to the component themselves.
By reading the interface to the component it can tell what function the component provides, and thereby impersonate this interface within the context object.

Now in the second stage client A has finished using the component, but still has a reference to it. What it's actually holding on to, of course, is the context object and not the component. MTS reclaims the component instance and returns it to the pool of available instances. Now C has requested an instance of the component. MTS creates a context object for this client C and links it to the instance of the component that was being linked to Client A. This is depicted in figure 3.4.

![Diagram of Client A, Client B, Client C, Context Object, Component, Instance 1, Instance 2, Transaction Server, Stage two.]

**Fig 3.4**

But, in stage three, client A wants to use the component again, maybe by calling another method within it. Unfortunately client C is still using it, but client B has temporarily finished with its instance of the component so MTS can grab this one and pass it to client A – by pointing A’s context object to this instance. Notice that client A
will use this component instance without being able to tell that it is a different instance to the one it used last time. This is depicted in figure 3.5.

![Diagram of Component Instance](image)

**Fig 3.5**

So MTS can provide a seamless context for each client application for the life of the page or application process. The clients don't realize that the objects they are referencing are being constantly shuffled around behind the scenes.

### 3.9 State and the Component Context

Now the question we have here is that how does MTS know when an application has finished using a component instance? One obvious solution is to explicitly destroy the object with `Set objectvariable=Nothing`, or implicitly, when the page or process is complete. MTS will know that the object can be returned to the pool and the application's context object is then destroyed. But this is only half the story.

We know that an application may hold through an instance of a component right through its lifetime. However, it may not actually be using the component all the time, and this is just the point where MTS needs to be able to reclaim the object and
let other applications or processes use it. This is achieved by adding a couple of lines of code in our component.

When a component has finished a task, it calls the SetComplete or SetAborts method of its context object, which effectively tell MTS that the component has finished the task and that it no longer requires its internal state to be maintained – i.e. all the data it was holding has either been returned to the client or stored permanently on disk. At this point MTS knows that it can reclaim the component and use it again elsewhere. When the original application comes back to use the component again, MTS can provide a fresh copy of it with default values for all properties.

One thing to bear in mind is that MTS can only reuse a component when the client application no longer expects the state to be the same as before. In other words, property settings and other information stored inside the component will probably not be the same as the last time it used the component. The application must reset any values that it needs by, for example, setting all the properties again.

3.10 Object Life Under MTS

One of the principal reasons that MTS can easily allow many users to access our objects is that they are around as long as we need them. When we create an object that is being hosted in MTS, which requires transaction support, the request is intercepted by MTS, which creates an MTS Context Wrapper object for our object. The 'real' object is not actually created.

It is only when we actually call a routine on the object that an instance of your object is actually created. This is known as Just In Time (JIT) activation. This saves on
resources because we don't actually have an instance hanging around until we actually need it. Finally when the routine ends the 'real' object is destroyed. This is known As Soon as Possible (ASAP) deactivation. The client is completely unaware of any of this happening because it's holding a reference to the Context Wrapper object and not the 'real' object.

3.11 Resource Pooling

Another resource saving performed by MTS is that it is able to pool resources. This allows limited resources, such as database connections, to be shared amongst a greater number of clients. Traditionally a client would be allocated resources for the lifetime of the application. This quickly uses up the available resources and hence limited the number of clients.[11]

Under MTS, the resources are pooled so that clients only demand resources when they need them and release them back into the pool when they are done.

Resource Pooling, JIT activation, ASAP deactivation, and many other MTS features are enabled by the MTS object called the Context object.
4 Transactions with MTS

A transaction can be thought of as a series of steps in which every step must be completed successfully or nothing will happen at all. If one step fails then all steps effectively fail, and any changes that have taken place are rolled back as if they had never happened.

For example a transaction can be composed of the following steps:

- Request to add a new order
- Begin transaction
- Remove product from Products table
- Send order to shipping
- Bill credit card
- End transaction

MTS makes transaction management transparent to the component developer. Developers do not have to write begin or end transaction statements into their application code. Developers can even use different tools and languages to build their components. They simply declare components to be transactional using the Transaction Server Explorer. Then, when a component begins execution, MTS starts a transaction automatically if one is required.

When the component accesses a resource such as a database, message queue, or mainframe application, MTS automatically enlists the resource in the transaction. If the component calls another component, the called component also joins the transaction automatically. When all components in a transaction complete their work, MTS initiates a full two-phase commit to either commit or abort the work.
4.1 Transactions and Distributed Components [5]

Transactions are an essential tool for building server-centric applications from component software. To date, most applications are developed as a monolithic application by a single team of developers where it is relatively easy to design transactions properly. But, as companies turn to component-based development, and use prebuilt components developed by other companies, transaction protection becomes much harder to ensure.

4.2 Managing Transactions

In order to take advantage of many features that MTS offers, we need tell MTS when we have finished a transaction and whether it was successful or not. In order to do this, we need to use the context object. Each of the object created under MTS has its own context object. The object contains information about the 'real' object's execution environment such as transaction status and security information.

More importantly, the context object also has 2 methods.

- Set Complete
- Set Abort

If the transaction was successful then we need call Set Complete and the changes will be completed. If for any reason, an error occurred, then we need to call Set Abort and the changes will be rolled back.

The transaction could be represented as shown in fig 4.1
4.3 MTS and Database Transactions [7]

MTS also allows to perform a database updates under transactions. MTS will do this by using an additional component installed with MTS, or SQL Server called the Distributed Transaction Coordinator (DTC). All communication with the database will occur through the DTC. Because all updates to the database go through the DTC, it can control the updates, these updates, committing or rolling back when necessary.

4.4 Under what circumstances does Microsoft Transaction Server initiate a transaction?

Microsoft Transaction Server initiates a transaction when an application component marked as "Requires a transaction" is invoked, provided the component's caller is
not already part of a transaction. If the caller is already part of a transaction, then the called component will join the caller's transaction.

Microsoft Transaction Server initiates a new transaction when an application component is invoked that is marked as "Requires a new transaction". This happens whether or not the caller is already part of a transaction.

When Microsoft Transaction Server initiates a transaction on behalf of a component, it stores the transaction in the component's object context.
5 MTS Benefits and Comparison with other technologies

5.1 MTS Benefits

When an ASP (Active Server Page) calls an MTS component, the request comes in to MTS as though it came from any other client. The component can:

- Perform any calculations required by the application logic.
- Access one or more databases via ODBC 3.0-compliant drivers (e.g., SQL Server 6.5 which we have used).
- Call other components that perform other parts of the application logic, enabling component reuse. In fact, companies can reduce development time by composing applications from a pool of prebuilt and purchased components.
- Because MTS provides automatic transaction support, if any error condition occurs within an MTS component or the ASP script, MTS will roll back all changes made to databases (including those on the mainframe, if applicable) and message queues.
- MTS transactions provide important data integrity protection required by production-quality systems. For example, without transaction protection, orders could be sent without collecting payments, and accounts could get out of balance.
- MTS also provides a sophisticated component run-time environment that can handle a large number of users and deliver consistently good response times.
- Automatic Thread pooling: As requests come from clients, MTS automatically assigns threads to components from a pre-allocated pool. When a component finishes executing, MTS reclaims the thread. This reduces the overhead of thread creation/deletion for better performance.
5.2 MTS compared to other technologies

In many ways, MTS defines an entirely new category of product. MTS uniquely provides a combination of:

- A component-based development model
- Strong infrastructure services such as transaction and security support
- Message queuing, mainframe application, and Web server integration

The two closest competitors to MTS are CORBA-compliant Object Request Brokers (ORBs) that support the Object Transaction Service and Sun’s Enterprise JavaBeans strategy.

5.2.1 CORBA-Compliant Object Request Brokers [10]

While the vision of the Object Management Group (OMG) remains exciting to most corporations, the CORBA approach suffers from a number of weaknesses:

CORBA is not a product. The CORBA specification is the result of committees staffed by many different companies with competitive distributed object technologies.

To gain agreement, OMG committees frequently had to write specifications that were vague enough to satisfy members with competitive approaches. The result is that CORBA-compliant products from different vendors are only minimally interoperable.

In other cases, specifications exist, but no vendor currently provides a compliant product.

Important run-time issues such as how to create and destroy server-based instances
of objects, and how to pool resources such as database connections, are not addressed by the CORBA specifications. CORBA-compliant products, as a result, frequently experience significant scalability problems.

The CORBA specification does not detail support for important component-oriented features such as managing an object's security and transaction requirements externally. In practice, this severely limits object reuse.

Most vendors of CORBA-compliant Object Request Brokers (ORBs) are small and have limited development resources. Consequently, they have focused on developing core ORB capabilities and not on more complex services such as the Object Transaction Service (OTS), which provides transaction management.

To use transactions with ORBs, customers have to purchase third-party OTS implementations that add complexity and overhead. Access to message queuing facilities and mainframe-based applications is usually provided by third parties as well. Most customers end up doing substantial amounts of systems integration.

Most popular ORBs offer extremely limited systems management capabilities.

The bottom line is that CORBA-compliant ORBs provide a simple way to build basic distributed object applications, but do not provide the infrastructure required for deployment in a production-quality capacity.

5.2.2 Enterprise JavaBeans [11]

More recently, Sun Microsystems announced an initiative called Enterprise JavaBeans. Like CORBA, Enterprise JavaBeans promises developers that they can build reusable components and run them anywhere on a network. Like CORBA, the Enterprise JavaBeans strategy has significant weaknesses. Enterprise JavaBeans is not a product. It is a specification that is the result of input from many companies with
competing interests. Pragmatic issues such as database connection pooling are not addressed directly by the Enterprise JavaBeans specification. The Enterprise JavaBeans specification does not detail support for important component-oriented features such as administrative interfaces for managing an object's security and transaction requirements externally.

Microsoft expects that companies will need to assemble various technologies from different vendors, and do systems integration, to incorporate important features such as transactions and message queuing into their applications.

In fact, until the Enterprise JavaBeans specification becomes more complete, it will be difficult for vendors to build compliant products and for customers to assess the viability of Enterprise JavaBeans in their environment.

5.3 Disadvantages of MTS

- Lacks in cross platform support.
- CORBA may remain the leading-edge middleware transaction model for networked objects. MTS transaction support is suitable for low end processing, but gaining ground quickly.
- Property settings are lost while creating a new instance of the component.
6 Project Specifications

The project involved the Development of a suite of active web pages to investigate Microsoft's MTS and analyze the pros and cons of a DNA System. The project was for the Business School in Auburn University. The project followed a three-tier architecture with Microsoft Transaction Server (MTS) as the middle tier. The project was developed using Active Server Pages. It involved extensive scripting using VBScript and Java Script. The project also included design and development of a database in SQL Server 6.5. ADO was used for data access using ODBC 3.0. User accounts were being maintained on two NT Servers and the authentication was handled through NT Challenge Response.

The web pages are designed for an on line homework system. The system can be divided into three parts. One set of the web pages are for the students, another for the faculty and the last one for the administrators. The function of each are explained as below.

6.1 The Administrators' Resources

This page gives all the necessary options to Add/Delete Faculty, View/Update Faculty Profile, Add/Delete Courses. All the administrative functions can be performed only by the administrator.

6.2 The Faculty Resources

Here the faculty member can log in with his userid and password. Once he logs in he has an option to select from a list of classes that he is currently is teaching. Once he chooses the class he currently wants to work on, he will get a list of options on the left navigator screen. The main options are
• **Add/Delete Class:** With this option the faculty can add or delete a class. For adding a class, he can create a new class and copy the text file from Austudy (Auburn University) and paste it in a text window. There is a program developed, which will tokenize this file and insert the student details into the database by calling the appropriate component.

• **Student profiles:** With this option the faculty can view and change the profile of his students. Also the faculty can change a students password.

• **Add/Delete questions:** With this option the faculty can add questions and give a due date by which the students are required to submit the assignment. The faculty can also submit a model answer which will be shown to the students, only after the due date is over. Other settings include maximum marks, maximum length of answer, etc.

• **Slide dates:** If a faculty at some point decides to extend the due date for the assignments for the whole class, group of students or a group of students then he can choose this option and make his selection.

• **Grade Assignments:** Once the due date of the assignment is over the faculty can go to this page and grade all the assignments. All the answers submitted by the students are available here. The faculty can navigate through the pages and gives his grade by clicking a single button. The grading results are available to the students in their respective pages immediately.

• **Grade sheets:** If a faculty wants to download the current grade statuses of his class in excel format then he can choose this option.
6.3 Students’ Resources

Similar to the faculty pages a student has his unique id and password by which he can log into his page. Once he logs into his page he will have to select the class he currently wants to work on. Once he selects the class, he has the following options for navigation.

- **Personal Information**: A student can change his current profile by using this page.
- **Homework**: Here is where a student can submit his assignment or edit his assignment if the due date isn’t over as yet. He can also use this page to view his grades.
- **Grades**: A student can choose this page to view his grades in the current quarter for all his assignments. He also gets to see where he currently stands in the class because the minimum, maximum and average for a particular assignment is also shown.
7 Conclusion

The growth of Internet computing is creating explosive demand for businesses to deploy solutions on servers. While the Internet has achieved its initial growth by providing an easy way to publish and share information online, businesses now realize that they can achieve even more by Web-enabling their line-of-business applications for Internet and intranet use. Instead of simply publishing sales brochures and product catalogs, businesses can operate accounting systems and order-entry systems on servers with users accessing the shared business functions with browsers.

Companies are also realizing that they are wasting huge amounts of programming effort by developing single-use two-tiered applications instead of three-tiered applications with reusable components that run on servers. By developing application logic once in component form and running components on a server, a variety of different presentation interfaces can be supported. And, since components can call other components easily, duplicating programming efforts is minimized.

Historically, there has been a problem with this server-centric vision. Lacking tool and infrastructure support, building and deploying server-based components is much harder than building single-use applications. Server-based applications need to be more reliable than desktop applications, because the impact of system failure or data corruption can affect an entire business, not just a single user. This mandates a sophisticated infrastructure that is costly to develop and difficult to maintain. By many estimates, infrastructure development represents 30%–40% of the average server-centric development project. To be viable, server-based applications need to be as easy to deploy and maintain as desktop applications, without requiring developers to
have special skills or to build costly, complex infrastructure code.

It was with these goals in mind that Microsoft developed Microsoft Transaction Server and included it with Microsoft Windows NT and Microsoft Windows 95. MTS combines the flexibility and low cost of desktop applications with the mission-critical transaction processing features normally found in high-end mainframe systems. And, because Microsoft Transaction Server is based on Microsoft's Component Object Model, MTS is accessible to a wide range of developers without expensive retraining.
BIBLIOGRAPHY


http://www.microsoft.com/msj/0198/mtscom/mtscomtop.htm


[6] Chris Dellinger ,“Employing Microsoft Transaction Server in Multitier Applications”.
July 1997 issue of Microsoft Interactive Developer.


[11] Alex homer and David Sussman, "Professional MTS MSMQ with VB and ASP"

Wrox press 1998
'Component for Faculty Resources

MultiUse = -1 'True
Persistable = 0 'NotPersistable
DataBindingBehavior = 0 'vbNone
DataSourceBehavior = 0 'vbNone
MTSTransactionMode = 0 'NotAnMTSObject
End
'
'local variable(s) to hold property value(s)
Private mvvarName As String 'local copy
Private mvvarID As Long 'local copy

Public Function PrepString(ByVal strClassInfo As String) As String
    Dim intTrimFrom As Long
    Dim intWhere As Long
    Dim strWhatLeft As String
    Dim strfind1 As String
    Dim strfind2 As String
    Dim strfind3 As String
    Dim strfind4 As String
    Dim strfind5 As String
    Dim strfind6 As String
    intTrimFrom = InStr(1, strClassInfo, "EMAIL ADDRESS", 1)
    intWhere = Len(strClassInfo) - (16 + intTrimFrom)
    strWhatLeft = Right(strClassInfo, intWhere)
    strfind1 = Chr(34) & Chr(34) & ","
    strfind2 = Chr(34) & Chr(34)
    strfind3 = Chr(34) & "," & Chr(34)
    strfind4 = Chr(13) & Chr(10)
    strfind5 = Chr(34)
    strfind6 = Chr(10) & Chr(13)
    strWhatLeft = Replace(strWhatLeft, strfind1, ",")
    strWhatLeft = Replace(strWhatLeft, ",", ",")
    strWhatLeft = Replace(strWhatLeft, strfind3, ",")
    strWhatLeft = Replace(strWhatLeft, strfind4, ",")
    strWhatLeft = Replace(strWhatLeft, ",", ",")
    strWhatLeft = Replace(strWhatLeft, strfind5, ",") & ","
    PrepString = Replace(strWhatLeft, ",", ",")
End Function

Private Function Replace(ByVal str As String, ByVal strfind As String, ByVal strreplace As String) As String
    Dim lenfind As Long
    Dim intfind As Long
    Dim part1 As String
    Dim part2 As String
    Do
        lenfind = Len(strfind)
        intfind = InStr(1, str, strfind, 1)
        If intfind = 0 Then
            Exit Do
        End If
        part1 = Left(str, intfind - 1)
        part2 = Right(str, Len(part1) + lenfind + 1)
        part1 = part1 & strreplace
        str = part1 & part2
    Loop
Replace = str

End Function

Private Sub Split(ByRef str As String, ByRef strCut As String, ByRef temp() As String)
    Dim i As Long
    Dim strcpy As String
    ReDim temp(1) As String
    Dim intTrim As Long
    intTrim = InStr(1, str, strCut, 1) - 1
    str = Left(str, intTrim)
    strcpy = str
    i = 1
    Do
        temp(i) = Left(strcopy, 1)
        strcpy = Right(strcopy, Len(strcopy) - 1)
        If strcpy = "" Then
            Exit Do
        End If
        i = i + 1
        ReDim Preserve temp(i) As String
    Loop

End Sub

Public Function DoData(ByRef strOutput) As String
    Dim intLen As Long
    Dim intCutTo As Long
    intLen = Len(strOutput)
    intCutTo = InStr(1, strOutput, ":", 1)
    DoData = Left(strOutput, (intCutTo - 1))
    strOutput = Right(strOutput, (intLen - intCutTo))
End Function

Public Function IsFaculty(ByRef login As String) As Boolean
    'Functions that checks
    'if the user is a faculty member.

    Dim SQLStr As String
    Dim qryfac As QueryDef
    Dim rsrODBC As Recordset
    DBEngine.DefaultType = dbUseODBC
    Dim conNewConnection As Connection
    If Len(login) = 0 Then
        IsFaculty = False
        Exit Function
    End If
    Set conNewConnection = OpenConnection _
        ("New", dbDriverNoPrompt, True, _
        "ODBC;DATABASE=sreevschool;DSN=sree_vschool")
    Set qryfac = conNewConnection.CreateQueryDef(""")
    With qryfac
        SQLStr = "SELECT Faculty.Username FROM Faculty "
        SQLStr = SQLStr & "WHERE Username =""
        Get SQLStr
        If login = rsrODBC.Fields("Username") Then
            IsFaculty = True
        Else
            IsFaculty = False
        End If
    End With
End Function
SQLStr = SQLStr & "login & 
qryfac.SQL = SQLStr
Set rstODBC = qryfac.OpenRecordset(dbOpenDynaset)
rstODBC.MoveLast
  If rstODBC.RecordCount > 0 Then
    IsFaculty = True
  Else
    IsFaculty = False
  End If
End With
rstODBC.Close
conNewConnection.Close
End Function

Public Function addClassString(ByVal InputString As String) As Boolean
Dim intLen As Long
Dim intComma As Long
Dim SSN As Long

Dim intName As Long
Dim intLine As Long
Dim name As String
Dim Blank As String
Dim Email As String
Dim temp() As String
Dim LastName As String
Dim Minit As String
Dim FirstName As String
Dim SQLStr As String
Dim qryfac As QueryDef
Dim qryfac1 As QueryDef
Dim rstODBC As Recordset
Dim wrkFaculty As Workspace

DBEngine.DefaultType = dbUseODBC
Dim conNewConnection As Connection
If Len(InputString) = 0 Then
  addClassString = False
  Exit Function
End If

Set conNewConnection = OpenConnection _
  ("New", dbDriverNoPrompt, True, _
   "ODBC;DATABASE=sreevschool;DSN=sree_vschool")
Set ntu = CreateObject("NTAccess.User")

intLen = Len(InputString)
  intComma = InStr(1, InputString, ",", 1)
  SSN = CLng(Left(InputString, (intComma - 1)))
  InputString = Right(InputString, (intLen - intComma))
  intLen = Len(InputString)
  intComma = InStr(1, InputString, ",", 1)
  name = Left(InputString, (intComma - 1))
  InputString = Right(InputString, (intLen - intComma))
  intLen = Len(InputString)
  intComma = InStr(1, InputString, ",", 1)
Set wrkFaculty = CreateWorkspace("wrk1", "", "", dbUseODBC)
Set conNewConnection = wrkFaculty.OpenConnection("sreevschool", dbDriverNoPrompt, , "ODBC;DATABASE=sreevschool;DSN=sree_vschool")
Ldate = Date
SQLStr = "INSERT INTO STUDENTS_NEW
(StuSSN,StuFName,StuMinit,StuLName,StuUsername,StuPassword,StuEmail) VALUES (" & SSNN & "," & FirstName & "," & Minit & "," & LastName & "," & StuUsername & "," & SSN & "," & Email & ")"
conNewConnection.ExecuteSQLStr, dbRunAsync
If conNewConnection.RecordsAffected <> 1 Then
addClassString = False
conNewConnection.SetAbort
Else
Set qryfac1 = conNewConnection.CreateQueryDef(""")
conNewConnection.SetComplete
With qryfac1
Prepare = dbQUUnprepare
SQLStr = "UPDATE SLOGIN SET LLDate = " & Ldate & " WHERE SSN=" & SSNN
qryfac1.SQL = SQLStr
Execute QueryDef qryfac1
End With
End If

' Set qryfac = conNewConnection.CreateQueryDef(""")
' With qryfac
'SQLStr = "INSERT INTO STUDENTS_NEW
(StuSSN,StuFName,StuMinit,StuLName,StuUsername,StuPassword,StuEmail) VALUES (" & SSNN & "," & FirstName & "," & Minit & "," & LastName & "," & StuUsername & "," & SSN & "," & Email & ")"
'qryfac.SQL = SQLStr
"Set rstODBC = qryfac.OpenRecordset(dbOpenDynaset)
'ExecuteQueryDef (qryfac)
'End With
'qryfac.Close
'Set qryfac1 = conNewConnection.CreateQueryDef(""
'With qryfac1
'SQLStr = "UPDATE SLOGIN SET LLDate = " & LDate & " WHERE SSN=" & SSN
'qryfac1.SQL = SQLStr
'ExecuteQueryDef (qryfac1)
'End With
  ntu.AddUser UserName, SSN, "Student"
  Set ntg = CreateObject("NTAccess.Groups")
  ntg.AddUserToGroup UserName, "hwstudent"
  SSNN = CLng(SSN)
  Ldate = Date
rstODBC.Close
conNewConnection.Close
wrkFaculty.Close
addClassString = True
if ntg.err
  ntg.SetAbort
else
  ntg.SetComplete
End Function

Public Function addStudent(ByRef OffID As Long, ByRef SSN As Long, ByRef Email As String, _
                           ByRef LastName As String, ByRef Minit As String, _
                           ByRef FirstName As String, ByRef UserName As String, ByRef OffCamp As Boolean) As Boolean
Dim addDate As Date
Dim ocamp As Integer
Dim PWD As String
Dim SQLStr As String
Dim rstODBC As New ADODB.Recordset
Dim conNewConnection As Connection
Dim Chn As New ADODB.Connection
Dim cmd As ADODB.Command
Dim rs As New ADODB.Recordset
Dim present As Boolean
Dim Max As Long
'DBEngine.DefaultType = dbUseODBC

'-----checks for valid inputs :
If Len(SSN) = 0 Or Len(Email) = 0 Or Len(LastName) = 0 Or Len(Minit) = 0 Or Len(FirstName) = 0 Then
  addStudent = False
  Exit Function
End If

'----- Checks if student already in file :
present = False
Chn.Open "hwork"
SQLStr = "SELECT StudSSN FROM Students "
SQLStr = SQLStr & "WHERE StudSSN = "
SQLStr = SQLStr & SSN & ""

'SQLStr = "Select * from Students"
Set cmd = CreateObject("ADODB.Command")
cmd.CommandText = SQLStr
cmd.ActiveConnection = Cn
cmd.CommandType = adCmdText
cmd.CommandTimeout = 120
rstODBC.Open cmd, 1, 1
If rstODBC.RecordCount > 0 Then
    present = True
End If
rstODBC.Close
'-----creates student details if not existant & adds him/her as NT user
If present = False Then
    PWD = CStr(SSN)
    addDate = Now()
If OffCamp = True Then
    ocamp = 1
Else
    ocamp = 0
End If
SQLStr = "INSERT INTO STUDENTS
(StudSSN,StfName,StMinit,StfName,Username,Passwd,OffCamp, StEmail) VALUES (" & SSN & ","," & FirstName & ","," & Minit & ",", & LastName & ","," & UserName & ",", & PWD & ",", & ocamp & ","," & Email & ")"
Cn.BeginTrans
Cn.Execute (SQLStr)
Cn.CommitTrans
Set ntu = CreateObject("NTAccess.User")
tnu.TrimErrorCodes = True
ntu.User = "admin"
ntu.AddUser UserName, SSN, "Student"
Set ntg = CreateObject("NTAccess.Groups")
tng.TrimErrorCodes = True
ntg.AddUserToGroup UserName, "hwstudent"
If cn.err
    Cn.SetAbort
Else
    Cn.SetComplete
End If

'-----Checks if student is already present in class
SQLStr = "SELECT StudSSN FROM ClassRoll"
SQLStr = SQLStr & " WHERE OfferingID = "
SQLStr = SQLStr & " AND StudSSN = " & SSN
cmd.CommandText = SQLStr
rstODBC.Open cmd, 1, 1
If rstODBC.RecordCount > 0 Then
    addStudent = False
    rstODBC.Close
    Cn.Close
    Exit Function
Else
    '-----if not member of the class, then adds the student
    rstODBC.Close
    SQLStr = "SELECT MAX(ClassRoll.SeatID) FROM ClassRoll"
cmd.CommandText = SQLStr
rstODBC.Open cmd, 1, 1
If rstODBC.RecordCount > 0 Then
    Max = rstODBC.Fields(0)
Else
    Max = 0
End If
Max = Max + 1
rstODBC.Close
Cn.BeginTrans
SQLStr = "Set IDENTITY_INSERT dbo.ClassRoll ON"
Cn.Execute (SQLStr)
Cn.CommitTrans
"---NOTE FOR PROGRAMMER: add the dateadded feature in the insert statement"
SQLStr = "INSERT INTO ClassRoll(SeatID, OfferingID, DateAdded, StudSSN, NewSeat, withdrew, sslide) VALUES(" & Max & "," & OffID & "," & addDate & "," & SSN & ", 0, 0, 0)"
Cn.BeginTrans
Cn.Execute (SQLStr)
If Cn.err
    Cn.SetAbort
Else
    Cn.SetComplete
    Cn.CommitTrans
End If
Cn.Close
addStudent = True
End Function

Public Sub ExecuteQueryDef(qdfTemp As QueryDef)
Dim errLoop As Error

' Run the specified QueryDef object. Trap for errors,
' checking the Errors collection if necessary.
On Error GoTo Err_Execute
qdfTemp.Execute (dbFailOnError)
On Error GoTo 0

Exit Sub

Err_Execute:

' Notify user of any errors that result from
' executing the query.
If DBEngine.Errors.Count > 0 Then
    For Each errLoop In DBEngine.Errors
        MsgBox "Error number: " & errLoop.Number & vbCrLf & _
            errLoop.Description
        Next errLoop
    End If
Resume Next

End Sub
Public Function slideByDays(ByRef qID, ByRef studID As Long, ByRef offerID As Long, ByRef days As Long) As Boolean

Dim Seat As Long
Dim cur_date As Date
Dim SQLStr As String
Dim Max As Long
Dim rstODBC As New ADODB.Recordset
Dim Cn As New ADODB.Connection
Dim cmd As New ADODB.Command

If Len(studID) = 0 Or Len(offerID) = 0 Or Len(days) Then
    slideByDays = False
    'Err.Raise Number:=vbObjectError + 1051, Source:=myObjectID, Description:="Invalid argument"
End If
Cn.Open "hwork"

'-----Selects the present deadline for the assignment

SQLStr = "SELECT DISTINCT(Questions.EndDate) FROM Questions "
SQLStr = SQLStr & "WHERE QuestID = "
SQLStr = SQLStr & qID
cmd.CommandText = SQLStr
cmd.ActiveConnection = Cn
cmd.CommandType = adCmdText
cmd.CommandTimeout = 120
rstODBC.Open cmd, , 1, 1

If rstODBC.RecordCount > 0 Then
    cur_date = rstODBC.Fields(0)
Else
    slideByDays = False
    Exit Function
End If
rstODBC.Close
SQLStr = "SELECT MAX(New_DeadLine.ExtentionID) FROM New_DeadLine"
cmd.CommandText = SQLStr
rstODBC.Open cmd, , 1, 1
If rstODBC.RecordCount > 0 Then
    Max = rstODBC.Fields(0)
Else
    Max = 0
End If
Max = Max + 1
rstODBC.Close

'-----Checks if Student in class
SQLStr = "Select SeatID from ClassRoll where OfferingID = " & offerID & " AND StudSSN = " & studID
cmd.CommandText = SQLStr
rstODBC.Open cmd, , 1, 1
If rstODBC.RecordCount > 0 Then
    Seat = rstODBC.Fields(0)
Else
    slideByDays = False
rstODBC.Close
Cn.Close
Exit Function
End If
rstODBC.Close

'----inserts a record into new_deadline
cur_date = CDate(cur_date + days)
SQLStr = "INSERT INTO New_DeadLine(ExtentionID, OfferingID, SeatID, QuestionID) VALUES (" & Max & "," & offerID & "," & Seat & "," & qID & ")"
Cn.BeginTrans
Cn.Execute (SQLStr)
If Cn.err
    Cn.SetAbort
Else
    Cn.SetComplete
Cn.CommitTrans

SQLStr = "UPDATE New_DeadLine SET EndDate=" & cur_date & "," where ExtentionID=" & Max
Cn.BeginTrans
Cn.Execute (SQLStr)
Cn.CommitTrans
Cn.Close
slideByDays = True
End Function

Public Function slideByDate(ByRef qID, ByRef studID As Long, ByRef offerID As Long, ByRef eDate As Date) As Boolean
Dim Seat As Long
Dim cur_date As Date
Dim SQLStr As String
Dim Max As Long
Dim rstODBC As New ADODB.Recordset
Dim Cn As New ADODB.Connection
Dim cmd As New ADODB.Command

'----Input validation
If Len(eDate) = 0 Or Len(studID) = 0 Or Len(offerID) = 0 Then
    slideByDate = False
Exit Function
End If

Cn.Open "hwork"

'----Selects the present deadline for the assignment
SQLStr = "SELECT DISTINCT(Questions.EndDate) FROM Questions"
SQLStr = SQLStr & "WHERE QuestID = "
SQLStr = SQLStr & qID
Set cmd = CreateObject("ADODB.Command")
cmd.CommandText = SQLStr
cmd.ActiveConnection = Cn
cmd.CommandType = adCmdText
cmd.CommandTimeout = 120
rstODBC.Open cmd, , 1, 1
If rstODBC.RecordCount > 0 Then
cur_date = rstODBC.Fields(0)
Else
  slideByDate = False
  Exit Function
End If
rstODBC.Close

'--Selects Max of extensionID
SQLStr = "SELECT MAX(New_DeadLine.ExtensionID) FROM New_DeadLine"
cmd.CommandText = SQLStr
rstODBC.Open cmd, 1, 1
If rstODBC.RecordCount > 0 Then
  Max = rstODBC.Fields(0)
Else
  Max = 0
End If
Max = Max + 1
rstODBC.Close
'--Checks if student's in class
SQLStr = "Select SeatID from ClassRoll where OfferingID = " & offerID & " AND StudSSN = " & studID
cmd.CommandText = SQLStr
rstODBC.Open cmd, 1, 1
If rstODBC.RecordCount > 0 Then
  Seat = rstODBC.Fields(0)
Else
  slideByDate = False
  rstODBC.Close
  Cn.Close
  Exit Function
End If
rstODBC.Close

'--- updates new date to new_deadline
cur_date = eDate
SQLStr = "INSERT INTO New_DeadLine(ExtensionID, OfferingID, SeatID, QuestionID) VALUES (" & Max & "," & offerID & "," & Seat & "," & qID & ")"
Cn.BeginTrans
Cn.Execute (SQLStr)
Cn.CommitTrans
SQLStr = "UPDATE New_DeadLine SET EndDate = " & cur_date & " where ExtensionID = " & Max
Cn.BeginTrans
Cn.Execute (SQLStr)
Cn.CommitTrans
Cn.Close
slideByDate = True
End Function

Public Function setNewDate(ByRef qID, ByRef eDate As Date) As Boolean

Dim SQLStr As String
Dim rstODBC As New ADODB.Recordset
Dim Cn As New ADODB.Connection
Dim cmd As New ADODB.Command

'---Input validation
If Len(eDate) = 0 Or Len(qID) = 0 Then
    setNewDate = False
    Exit Function
End If

Cn.Open "hwork"

'-----Selects the present deadline for the assignment

SQLStr = "SELECT DISTINCT(Questions.EndDate) FROM Questions"
SQLStr = SQLStr & " WHERE QuestID = "
SQLStr = SQLStr & qID
Set cmd = CreateObject("ADODB.Command")
cmd.CommandText = SQLStr
cmd.ActiveConnection = Cn
cmd.CommandType = adCmdText
cmd.CommandTimeout = 120
rstODBC.Open cmd, , 1, 1

If rstODBC.RecordCount > 0 Then
    rstODBC.Close
    SQLStr = "UPDATE Questions SET Questions.EndDate = " & eDate
    SQLStr = SQLStr & " WHERE QuestID = " & qID
    Cn.BeginTrans
    Cn.Execute (SQLStr)
    Cn.CommitTrans
Else
    rstODBC.Close
    Cn.Close
    setNewDate = False
    Exit Function
End If

Cn.Close
setNewDate = True
Exit Function

Public Function moveEndDate(ByRef qID, ByRef days As Long) As Boolean
Dim cur_date As Date
Dim SQLStr As String
Dim rstODBC As New ADODB.Recordset
Dim Cn As New ADODB.Connection
Dim cmd As New ADODB.Command

'-----Input validation
If Len(days) = 0 Or Len(qID) = 0 Then
    moveEndDate = False
    Exit Function
End If

Cn.Open "hwork"

'-----Selects the present deadline for the assignment

SQLStr = "SELECT DISTINCT(Questions.EndDate) FROM Questions"
SQLStr = SQLStr & " WHERE QuestID = "
SQLStr = SQLStr & qID
Set cmd = CreateObject("ADODB.Command")
cmd.CommandText = SQLStr
cmd.ActiveConnection = Cn
cmd.CommandType = adCmdText
cmd.CommandTimeout = 120
rstODBC.Open cmd, 1, 1

If rstODBC.RecordCount > 0 Then
    cur_date = rstODBC.Fields(0)
rstODBC.Close
cur_date = CDate(cur_date + days)
SQLStr = "UPDATE Questions SET Questions.EndDate = " & cur_date
SQLStr = SQLStr & " WHERE QuestID = " & qID
Cn.BeginTrans
Cn.Execute (SQLStr)
Cn.CommitTrans
If Cn.err
    Cn.SetAbort
Else
    Cn.SetComplete
Else
    rstODBC.Close
    Cn.Close
    moveEndDate = False
    Exit Function
End If

Cn.Close
moveEndDate = True
End Function

Public Function getfromdb(ByVal idcopy As Long) As Boolean
Dim dataconn As New ADODB.Connection
'dataconn.Open "'DSN=Student;DBQ=C:\admin\student.mdb;DriverID=25;FIL=MS Access;MaxBufferSize=512;PageTimeout=5","",""
'Dim cmdtemp As ADODB.Command

'cmdtemp.CommandText = "SELECT * from studdata where ID = idcopy"
'cmdtemp.ActiveConnection = dataconn
'Dim rsdata As New ADODB.Recordset
'rsdata.Open cmdtemp, 0, 1
'Dim wrkFaculty As Workspace
'Dim conOffering As Connection
'Dim rstClass As Recordset
'Dim success As Long
'Dim sqlstring As String
'Set wrkFaculty = CreateWorkspace("Student", "", "", dbUseODBC)
'Set conOffering = wrkFaculty.OpenConnection("Student", dbDriverNoPrompt, "ODBC;DSN=Student")
'sqlstring = "SELECT * from studdata where ID = idcopy"
'Set rstClass = conOffering.OpenRecordset(sqlstring, dbOpenDynamic, dbRunAsync, dbOptimisticValue)
'If rstClass.RecordCount <> 0 Then
    getfromdb = True
Else
    getfromdb = False
'End If
'stClass.Close
'conOffering.Close
'wrkFaculty.Close

Dim dbMydb As Database
Dim recEmployees As Recordset
Set dbMydb = OpenDatabase("C:\netdata\sreevschool")

Dim strSQL As String
strSQL = "Select * from studata"
Set recEmployees = dbMydb.OpenRecordset(strSQL)

If recEmployees.RecordCount <> 0 Then
getfromdb = True
Else
getfromdb = False
End If

End Function

Public Sub addtodb(ByVal name As String, ByVal idcopy As Long)
Dim dataconn As New ADODB.Connection
dataconn.Open "DSN=Student;DBQ= C:\admin\student.mdb;DriverID=25;FILE=MS Access;MaxBufferSize=512;PageTimeout=5;""

Dim cmdtemp As ADODB.Command

cmdtemp.CommandText = "INSERT INTO studata values(name,idcopy)"

End Sub

Public Property Let id(ByVal vData As Long)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.ID = 5
    mvarID = vData
End Property

Public Property Get id() As Long
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.ID
    id = mvarID
End Property

Public Property Let name(ByVal vData As String)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.name = 5
    mvarname = vData
End Property

Public Property Get name() As String
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.name
    name = mvarname
End Property
Component for Student Resources

MultiUse = -1 'True
Persistable = 0 'NotPersistable
DataBindingBehavior = 0 'vbnone
DataSourceBehavior = 0 'vbnone
MTSTransactionMode = 0 'NotAnMTSObject
End

Public Function selectDate(ByRef qID, ByRef SSN As Long, ByRef offerID As Long) As Date

Dim cur_date As Date
Dim SQLStr As String
Dim Max As Long
Dim rstODBC As New ADODB.Recordset
Dim Cn As New ADODB.Connection
Dim cmd As New ADODB.Command

If Len(SSN) = 0 Or Len(offerID) = 0 Or Len(qID) = 0 Then
    selectDate = Null
    Exit Function
'Err.Raise Number:=vbObjectError + 1051, Source:=myObjectID, Description:="Invalid argument"
End If
Cn.Open "hwork"
'-----Selects the present deadline for the assignment

SQLStr = "SELECT New_Deadline.EndDate FROM New_deadline, ClassRoll "
SQLStr = SQLStr & "WHERE New_deadline.QuestionID = "
SQLStr = SQLStr & qID
SQLStr = SQLStr & " AND New_deadline.OfferingID = " & offerID
SQLStr = SQLStr & " AND ClassRoll.SeatID = New_Deadline.SeatID "
SQLStr = SQLStr & " AND ClassRoll.StudSSN = " & SSN

cmd.CommandText = SQLStr
cmd.ActiveConnection = Cn
cmd.CommandType = adCmdText
cmd.CommandTimeout = 120
rstODBC.Open cmd,, 1, 1

If rstODBC.RecordCount > 0 Then
    rstODBC.MoveNext
    cur_date = rstODBC.Fields(0)
    selectDate = cur_date
    rstODBC.Close
Else
    rstODBC.Close
    SQLStr = "SELECT DISTINCT(Questions.EndDate) FROM Questions "
    SQLStr = SQLStr & "WHERE QuestID = "
    SQLStr = SQLStr & qID
    cmd.CommandText = SQLStr
    rstODBC.Open cmd,, 1, 1
    If rstODBC.RecordCount > 0 Then
        cur_date = rstODBC.Fields(0)
        selectDate = cur_date
        rstODBC.Close
    Else

Else
  rstODBC.Close
  Cn.Close
  Exit Function
End If
End If

Cn.Close
End Function