3 Tears in (Almost) 3 Hours: A 3-Tier Application Workshop

Michael Ferber

CA-World 2000
eBusiness Solutions in Internet Time
JI075LNA/B

Introduction

Three-tier application development is the state-of-the-art way of implementing business solutions. A wide choice of development tools gives us the flexibility to choose the best for every tier. Modern operating systems are also designed to support this kind of application. After a short introduction of the principles of a 3-tier application, we will get our hands dirty and create a sample application. We will see different development tools, including Jasmine *ii* and CA-Visual Objects, and operating system utilities working together. In the end we should have a good understanding about which components are involved and how they should work together to design and build a 3-tier application.

The Windows DNA Technology

Before we start talking about a 3-tier architecture, we should first take a look at the design principles supporting a 3-tier application. The 3 tiers are:

- Data Layer
- Business Logic Layer
- Presentation Layer

Microsoft is one of the driving forces behind distributed application development. To distinguish themselves from the rest of the world they called their 3-tier architecture Microsoft Windows Distributed interNet Applications Architecture, or Windows DNA.

Windows DNA is Microsoft's framework for building business solutions. It is an application architecture that embraces and integrates the Internet, client/server, and PC models of computing for distributed computing solutions.

The heart of Windows DNA is the integration of Web and client/server application development models through the use of COM. Windows DNA services are exposed through COM for applications to use. These services include component management, dynamic HTML, Web browser and server, scripting, transactions, message queuing, security, directory, database and data access, systems management, and user interface.

Getting Started

Technology Overview

The following list of products is involved in the creation of our 3-tier application:

- The back end is an RDBMS like MS SQL Server.
- For developing middle-tier components we work with CA-Visual Objects 2.5, Microsoft Visual Basic and Jasmine *ii*.
- The rich client is created by using CA-Visual Objects 2.5, and the thin client is a Web application created by using Microsoft Frontpage and/or Microsoft Visual Interdev.

Integrating the SQL Database in Jasmine *ii*

To integrate any information source into the Jasmine *ii* infrastructure you need to create a new namespace.

Select the "Root" node in your Jasmine *ii* Browser treeview, click the right mouse button and select "New-Namespace". A new entry in the tree is created, named something like "NewNamespace_0". Select this tree view item and select "Namespace Definition" in the right pane of the Browser. Enter a proper namespace description and select "rsoledb" as provider.

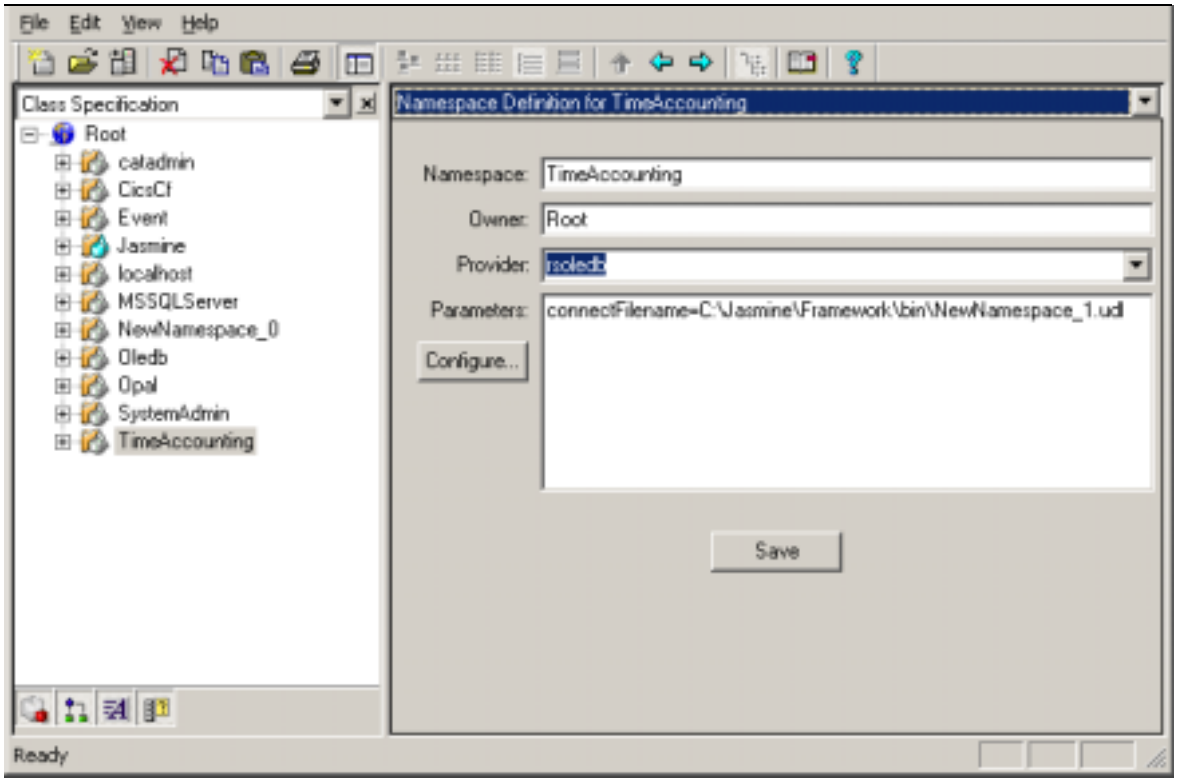

Figure 1: Jasmine *ii* Browser

After you clicked on the "Configure.." button, a "Data Link Properties" dialog is opened where you now specify the details of your OLE DB provider and connection.

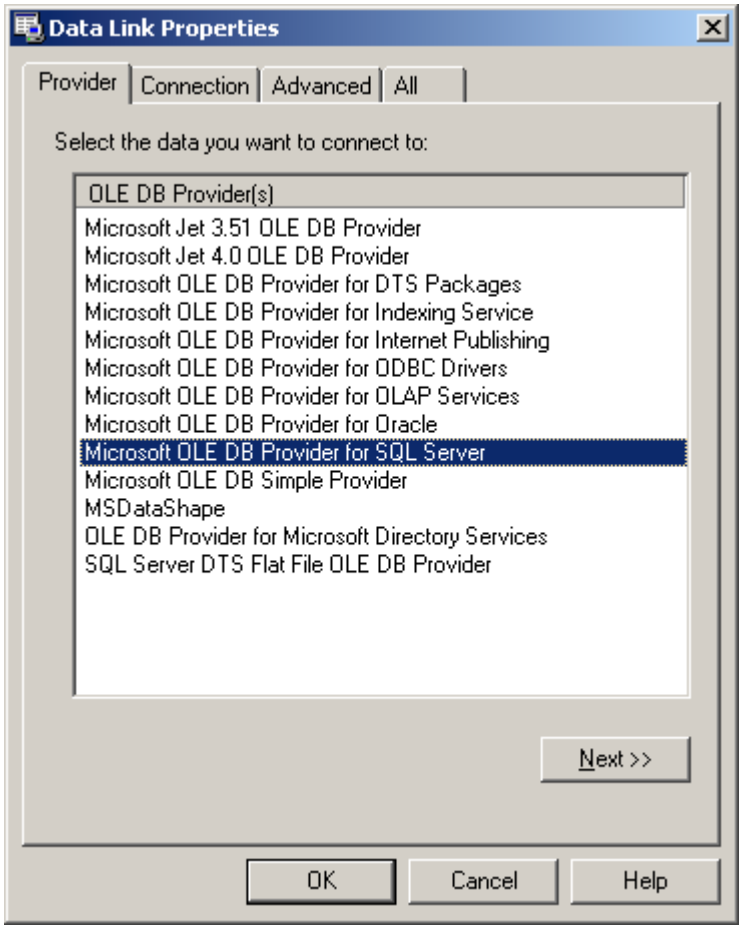

Figure 2: Data Link Properties - List of Providers

If want to work with a data source where an ODBC definition has already been created, just point to that driver to connect to your database. In the case of accessing an SQL server database, select the server, user name and login password to enable Jasmine *ii* later to create a connection to your data source.

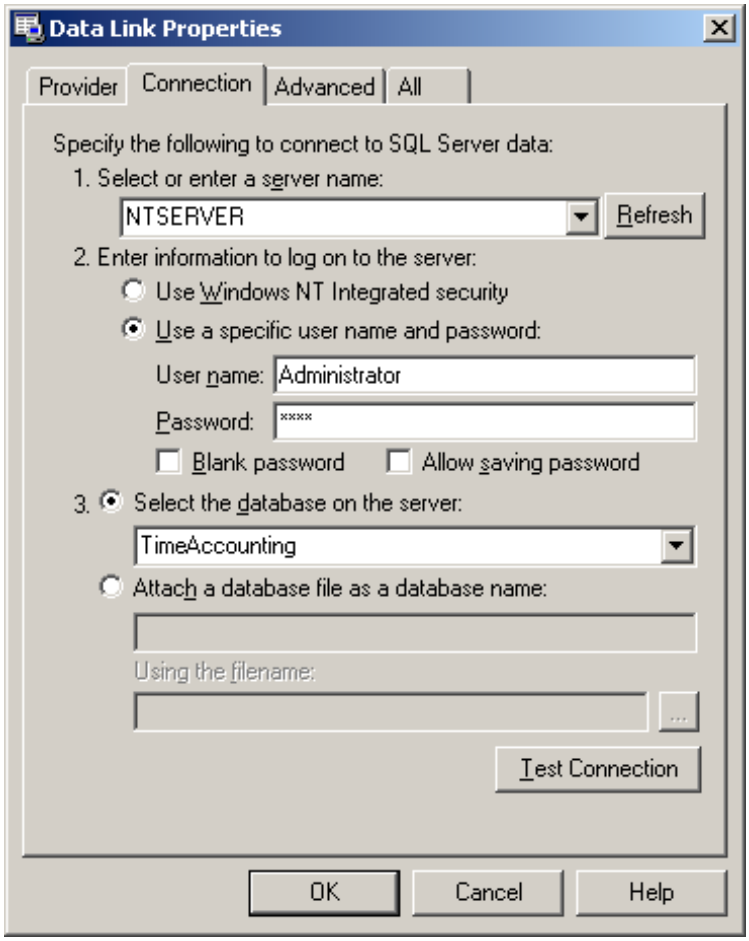

Figure 3: Data Link Properties - Connection Definition

Save all the information in the dialog by pressing the OK button and by pressing the "Save" button in the right pane of the Browser application.

Layer 2 Language Bindings for COM

Jasmine *ii* language bindings are a set of tools included with Jasmine *ii* that enables you to communicate with the Jasmine *ii* environment and information sources. Jasmine *ii* incorporates a three-layer, object-oriented architecture. The lowest layer has the basic building blocks from which all other objects and interfaces are built and represented. Layer 1 implements a dynamic object model that is ideal for manipulating meta data and building tools, such a browsers, that need to provide access to information sources without knowing their content in advance. Layer 2 provides a static model of information source data and is an ideal interface for building business applications. The Layer 2 language binding is the ideal way to use Jasmine *ii* in Active Server Pages (ASP).

Jasmine *ii* language bindings provide application developers with a consistent object-oriented interface to information sources and services from the many languages and development environments supported.

The Example: Time Accounting System

The example we want to use to show the architecture of a 3-tier application is a Time Accounting System. This system allows you to specify projects and define estimated times you plan to spend on these projects. You have a list of users who work for you and they have to report the time they actually spent on these various projects.

Data Layout

We begin the design process by examining the requirements for the persistent data in our application and the design of the tables in our data-model. As outlined, we need to store information about projects, users and a third table, which defines the kind of work you do, like "Database Design", "On-LineHelp" or "Installation Procedure." All the information comes together in a table which is like a work sheet, where every user enters the time he spent on various tasks within a project. For different kinds of work the user has to enter new records.

Our data-model should be normalized, and we work with internal unique keys to identify entities.

Table Projects

Name	Key	Data Type	Size	Nulls
PROJECTID	Yes	int	4	No
PROJECTDESCRIPTION	No	char	40	No
ALLOCATEDHOURS	No	float	8	Yes
USEDHOURS	No	float	8	Yes

Table Users

Name	Key	Data Type	Size	Nulls
USERID	Yes	int	4	No
USERNAME	No	char	40	No

Table Types

Name	Key	Data Type	Size	Nulls
TYPEID	Yes	int	4	No
TYPEDESCRIPTION	No	char	40	No

Table TimeAccounting

Name	Key	Data Type	Size	Nulls
TRANSACTIONID	Yes	int	10	No
USERID	No	int	4	No
PROJECTID	No	int	4	No
DAY	No	datetime	8	No
HOURS	No	float	8	Yes
CHARGEABLE	No	bit	1	No
DESCRIPTION	No	char	30	Yes
TYPEID	No	int	4	No

We now take this data-model and create the tables in our favorite database, like Jasmine ODB, Oracle or SQL Server.

Creating Data Objects

The second step in creating our 3-tier application is to create a set of so-called data objects. These components are responsible for the accuracy, completeness and consistency of the data they own. The implementation of these objects follows certain rules, which are determined by the use of COM+ technology. These components must be implemented as in-process components. Each component must provide a class factory for each COM class in the component, as well as a type library that describes all of the COM classes and their interfaces.

Within each method are two things you need to do. First you need to let COM+ know when you have finished with the object's state. Secondly, you need to handle errors within your object and let COM+ know whether the method was successful. You do this by using the *ObjectContext* associated with each object. The SetComplete() method indicates that your object has completed its work and the SetAbort() method indicates an error.

Since you should not retain state information in your object across method calls, a typical method template in Visual Basic will look like this:

```
Public Function dbMyFunc()

On Error GoTo ErrorHandler
       :
       ' Do Work
       GetObjectContext.SetComplete
       Exit Function
ErrorHandler:
       GetObjectContext.SetAbort
       Err.Raise
End Function
```

Now it is time to think about the data object technology we will use to manipulate the information in our database. The good thing here is that we have choice. We can either go with Microsoft's ActiveX Data Objects (ADO) or we can integrate our persistent data in the Jasmine *ii* environment and use Jasmine *ii*'s COM binding technology.

The following script shows how to use the Jasmine *ii* COM binding object to connect to the database and retrieve a list of records:

```
Set oJasmine = CreateObject("JasmineCOMBinding.Jasmine")
Set oJasmineSession = oJasmine.Connect(Nothing)
Set oInsured = oJasmineSession.Root.DeepFind("Jasmine/insuranceCF")
Set item=Nothing
Set query=oInsured.CreateQuery("Insured")
Set items=query.Run()
For Each item in Items
       Response.write(item("lastName"))
       Response.Write(" ")
       Response.Write(item("firstName"))
next
Set items=Nothing
oJasmineSession.Commit()
oJasmineSession.Disconnect()
Set oJasmine = Nothing
```

Building Business Objects

With the data objects in place we can turn our attention to the substance of our application, the business objects. Business objects encapsulate real-world business operations, independent of how the data they use is stored. Your business objects will use the already defined data objects. Business objects usually encapsulate multiple step operations and can access multiple data objects.

In our example, a business object would be needed to recalculate the amount of time already used for a specific project every time a user enters a record in the time accounting system regarding this project. The business object needs to access two data objects, the project data object and the time accounting data object.

The business objects follow the same rules we have seen before with data objects. They must be implemented as in-process components, and they must provide a class factory for each COM class in the component, as well as a type library that describes all of the COM classes and their interfaces.

Because your business object uses multiple data objects or multiple business objects or a mix of multiple data and business objects, you have a process called composing functionality.

State management and error handling now become an issue. Components are being developed individually, without any knowledge of the other components they are working with. If an error occurs in one component, the actions already performed by other components might need to be undone. COM+ associates with every COM object an object context. This object context contains COM+-managed state information which is used to provide services to your objects, such as caller's identity, the activity and the transaction identifier. To inherit aspects of the caller's object context, you use the CreateObject() method of the object context to create new objects. The use of this method ensures that context information is passed properly to those objects.

If you do not use the object context to create subordinate objects, those objects might not participate in a transaction.

Each component now calls SetComplete() or SetAbort(), and the whole transaction is marked as complete only if every single subordinate component completed its transaction. Otherwise, the whole process is rolled back.

Error handling should also be considered when designing your business objects. In Windows NT and Windows 2000 you can use the system event log to save error information inside your components; there is no need to create your own error management tool.

We also want to provide a source code template for business objects in Visual Basic:

```
Public Function busMyFunc()

On Error GoTo ErrorHandler
       :
       Set obj1 = GetObjectContext.CreateInstance("dbObj1,Obj1" )
       obj1.dbMyFunc()
       ' Do some work
       Set obj2 = GetObjectContext.CreateInstance("busObj2,Obj2" )
       obj2.busMyFunc()
       ' Do some more work
       GetObjectContext.SetComplete
       Exit Function
ErrorHandler:
       GetObjectContext.SetAbort
       Err.Raise
End Function
```

Packaging Components with COM+

Windows 2000 includes a tool called "Component Services," which is the old Microsoft Transaction Server now updated and included in the operating system, and bundled with the old COM technology into something new called COM+.

COM+ supports object pooling, which should optimize the performance of COM components by pre-allocating and efficiently re-using resources.

COM+ also includes a complete role-based security model. This term refers to a concept whereby a single component can provide different sets of services that depend on a user's permissions under Windows 2000.

COM+ enables component queuing, which can be used to implement time-independent applications. That means your middle-ware components must not exist at the moment when a call to this component is made. What is important is only that the calls are processed eventually.

Your set of middle-ware objects is grouped together in so-called packages. Packages later can be exported and redistributed.

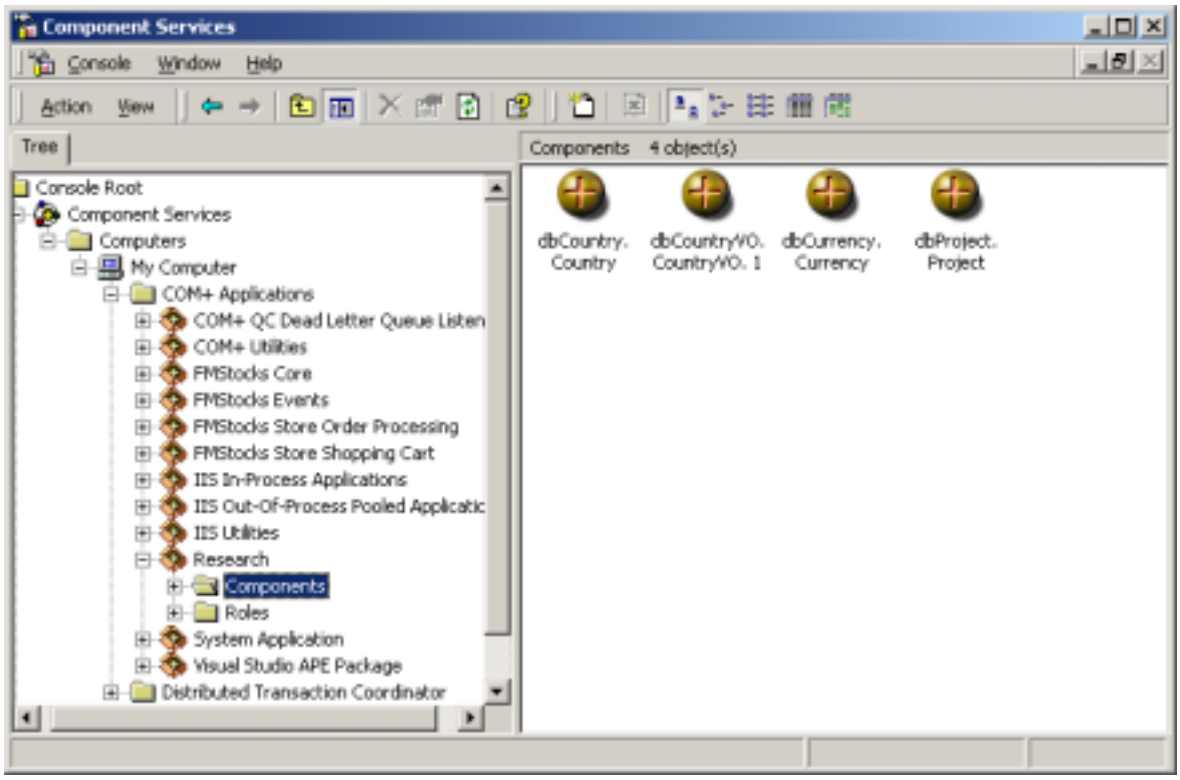

Figure 4: Component Services (COM+)

To add a component to a package, simply right-click on the "Components" menu item of the package you wish to work with and select the "New - Component" menu item. This will display a component wizard that will allow you to either install a new component or add an already installed component to the package.

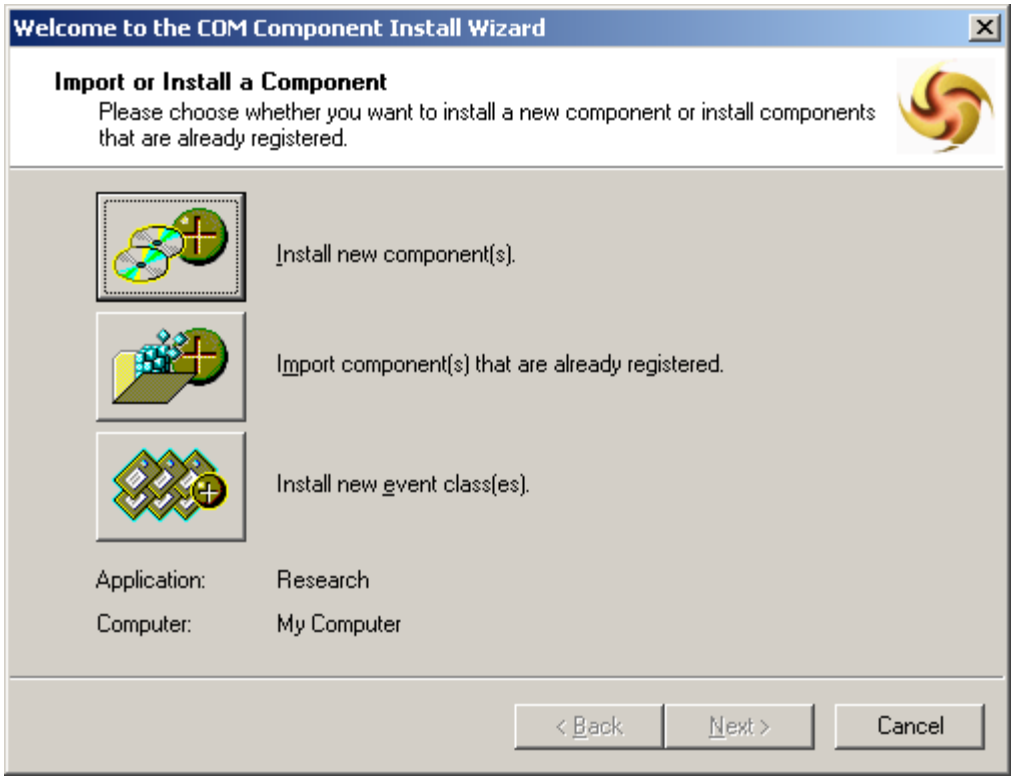

Figure 5: Install a new component in COM+.

You select the component(s) you wish to include in this package. The wizard shows you the files and the components they contain.

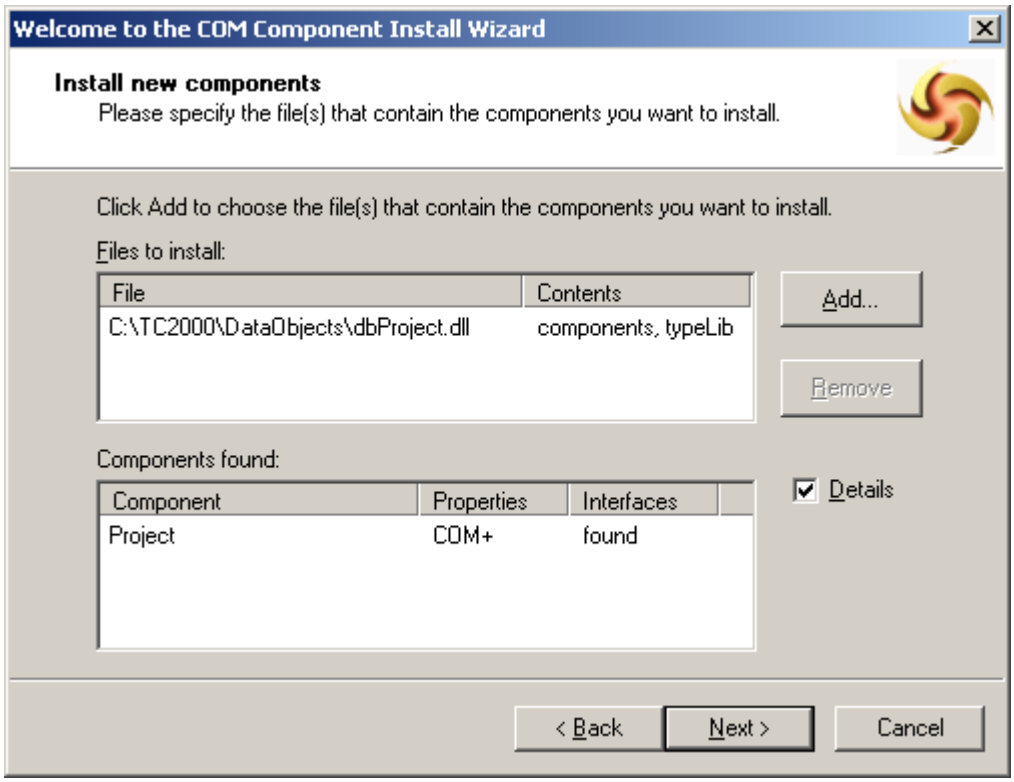

Figure 6: Complete the installation of a new component.

ASP User Interface

Global.asa File

Global.asa is a file maintained on a Microsoft Internet Information Server (IIS) for each application. This file is processed automatically by the server when:

- The IIS application starts and stops.
- Individual users start and stop browser sessions that access the application's Web pages.

The Global.asa file typically contains scripts to initialize application or session variables, connect to databases, send cookies, and perform other operations that pertain to the application as a whole.

We establish a connection in the Session_OnStart() procedure and close the connection in the Session_OnEnd() procedure:

```
Sub Session_OnStart
       Session.Timeout = 5
       Set Session("Jasmine") = CreateObject("JasmineCOMBinding.Jasmine")
       Set Session("JasmineSession") = Session("Jasmine").Connect(Nothing)
End Sub
Sub Session_OnEnd
       Session("JasmineSession").Disconnect()
       Set Session("Jasmine") = Nothing
End Sub
```

Design the Pages

Now it's time to start designing Web pages: select a basic screen layout and generate the necessary pages. A two-way HTML Editor enables you to generate a screen layout very fast, but also offers you full support for scripting and fancy HTML-tag support.

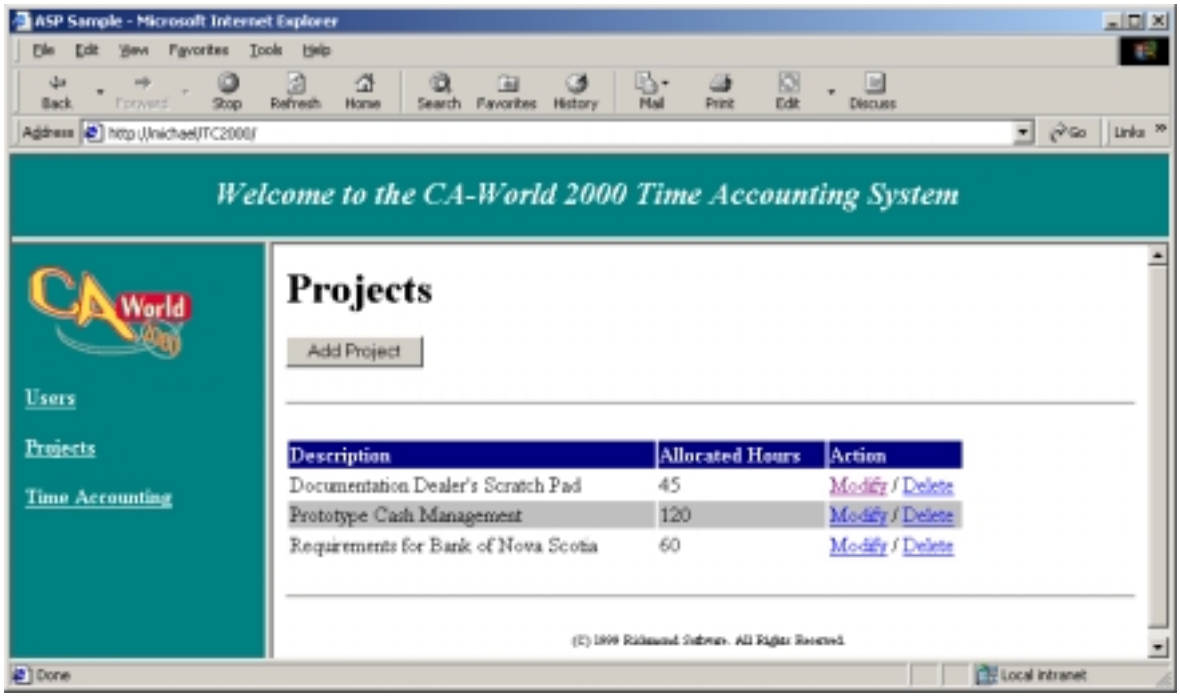

Figure 7: List of projects in the ASP sample.

To fill the list of projects we use a script in an ASP file to write to the Response object:

```
Set oRS = oDB.ListAll
Response.Write( "<table border='0' width='80%'>" )
Response.Write( "<tr>" )
Response.Write( "<td width='55%'
       bgcolor='#000080' style='color: #FFFFFF'>
       <b> <font color='#FFFFFF'>Description</font></b></td>" )
Response.Write( "<td width='25%'
       bgcolor='#000080' style='color: #FFFFFF'>
       <b> <font color='#FFFFFF'>Allocated Hours</font></b></td>" )
Response.Write( "<td width='20%'
       bgcolor='#000080' style='color: #FFFFFF'>
       <b> <font color='#FFFFFF'>Action</font></b></td>" )
Response.Write( "</tr>" )
i = 1
Do While Not oRS.EOF
       Response.Write( "<tr>" )
       Response.Write( "<td width='55%'" )
       if i mod 2 = 0 then
              Response.Write( "bgcolor='C0C0C0'" )
       end if
       Response.Write( ">" )
       Response.Write( oRS( "PROJECTDESCRIPTION" ) )
       Response.Write( "</td>" )
       Response.Write( "<td width='25%'" )
       if i mod 2 = 0 then
              Response.Write( "bgcolor='C0C0C0'" )
       end if
       Response.Write( ">" )
       Response.Write( oRS( "ALLOCATEDHOURS" ) )
       Response.Write( "</td>" )
       Response.Write( "<td width='20%'" )
       if i mod 2 = 0 then
              Response.Write( "bgcolor='C0C0C0'" )
       end if
       Response.Write( ">" )
       Response.Write( "<A HREF='ProjectEdit.ASP?ID=" &_
                     oRS("PROJECTID") &_
                     "' target rbottom>Modify</A>" )
       Response.Write( " / " )
       Response.Write( "<A HREF='ProjectDelete.ASP?ID=" &_
                     oRS("PROJECTID") &_
                     "' target rbottom>Delete</A>" )
       Response.Write( "</td>" )
       Response.Write( "</tr>" )
       i = i+1
       oRS.MoveNext
Loop
Response.Write( "</table>" )
```

Windows Client

To create a "classical" Windows client, we use CA-Visual Objects 2.5. CA-Visual Objects includes all the tools you need to create a rich user interface, like visual editors, windowpainters, Jasmine *ii* integration and a fully object-oriented 32-bit compiler.

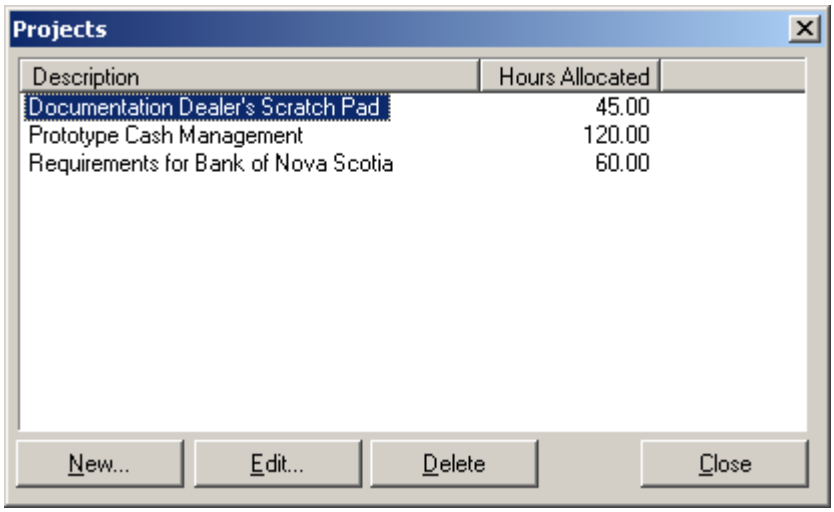

Figure 8: List of projects in the CA-Visual Objects client.

After designing the window with the Window Editor, you include the code to fill the list view with the list of projects in the PostInit() method:

```
METHOD PostInit(oParent,uExtra) CLASS NewWindow1
:
oRS := oData:ListAll()

DO WHILE !oRS:EOF
       oItem := ListViewItem{}
       oItem:SetValue( oRS:Fields:[ Item , "PROJECTDESCRIPTION" ]:Value , ;
              #Description )
       oItem:SetValue( oRS:Fields:[ Item , "ALLOCATEDHOURS" ]:Value , ;
              #Allocated )
       SELF:oDCListView1:AddItem( oItem )
       oRS:MoveNext()
ENDDO
RETURN NIL
```

Installing the Web Client

The installation of the Web client requires that you have a Web server installed either on your machine or on your local intranet. Windows 2000 includes a personal Web server, which is just great to test your Web pages, scripts and executables. You either register your Web application as the default one on your Web server or you provide an alias from which you can launch your Web application.

In this example, we set up an alias called TC2000 and point this alias to the directory where our Web application is located. Later we can launch the application from the Internet Explorer with the following URL:

http://localhost/TC2000

To create a new alias, we open the Internet Information Services and select the "Default Web Site" node.

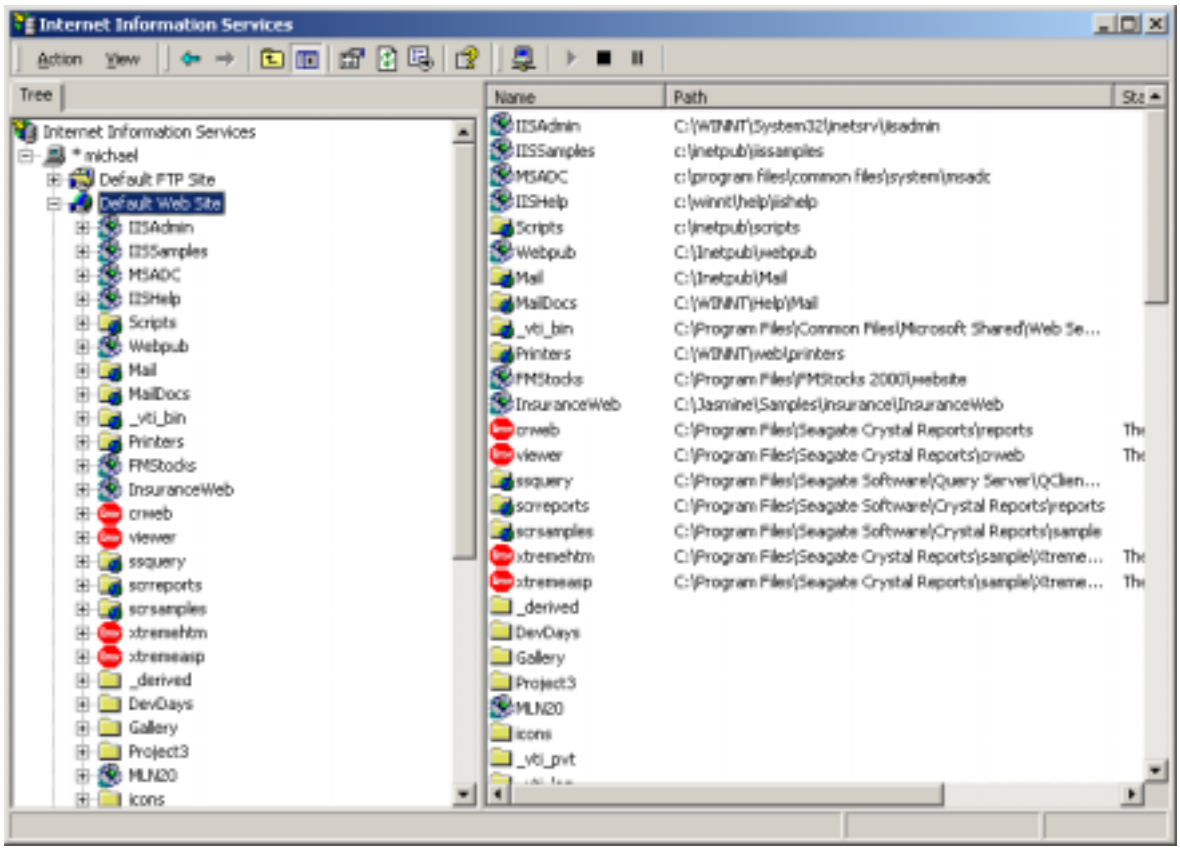

Figure 9: Internet Information Services

We click the right mouse button and select "New Virtual Directory" from the pop-up menu. A wizard is now opened where we fill in the information about the alias name of the new virtual directory and the name of the path on our machine where the Web application files (HTML pages, ASP pages, GIF files, ...) are located.

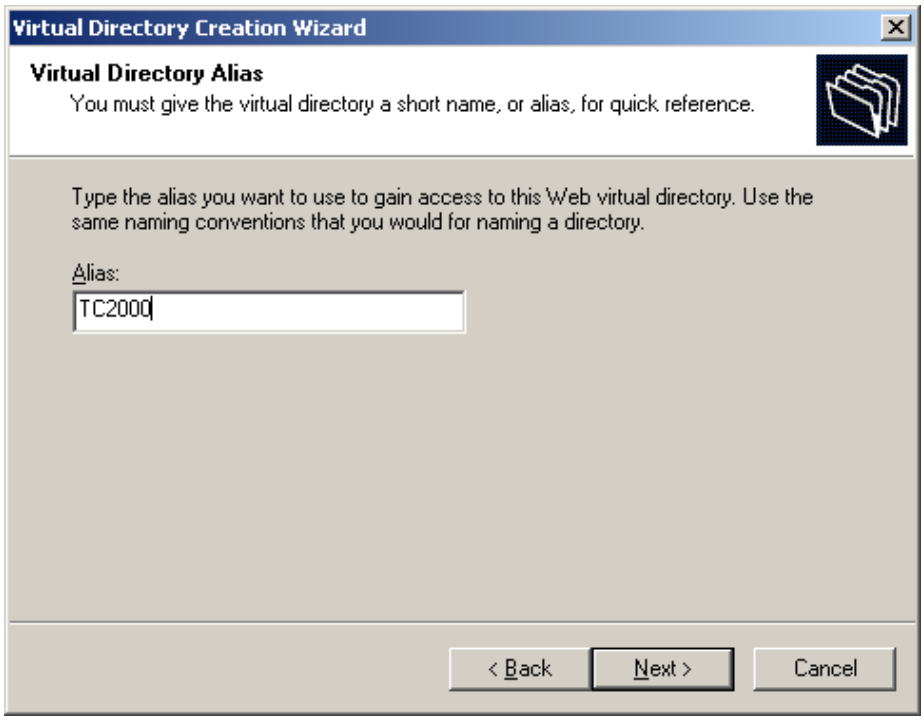

Figure 10: Define the alias for a new virtual directory.

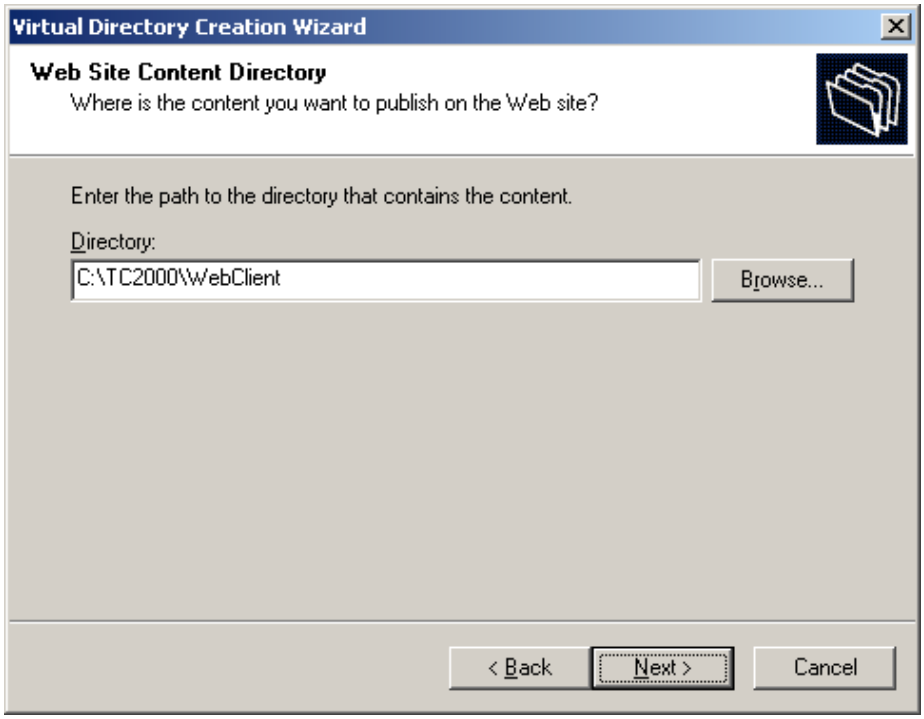

Figure 11: Select the directory of your Web application.

We finally define access permissions for this directory and finish this wizard.

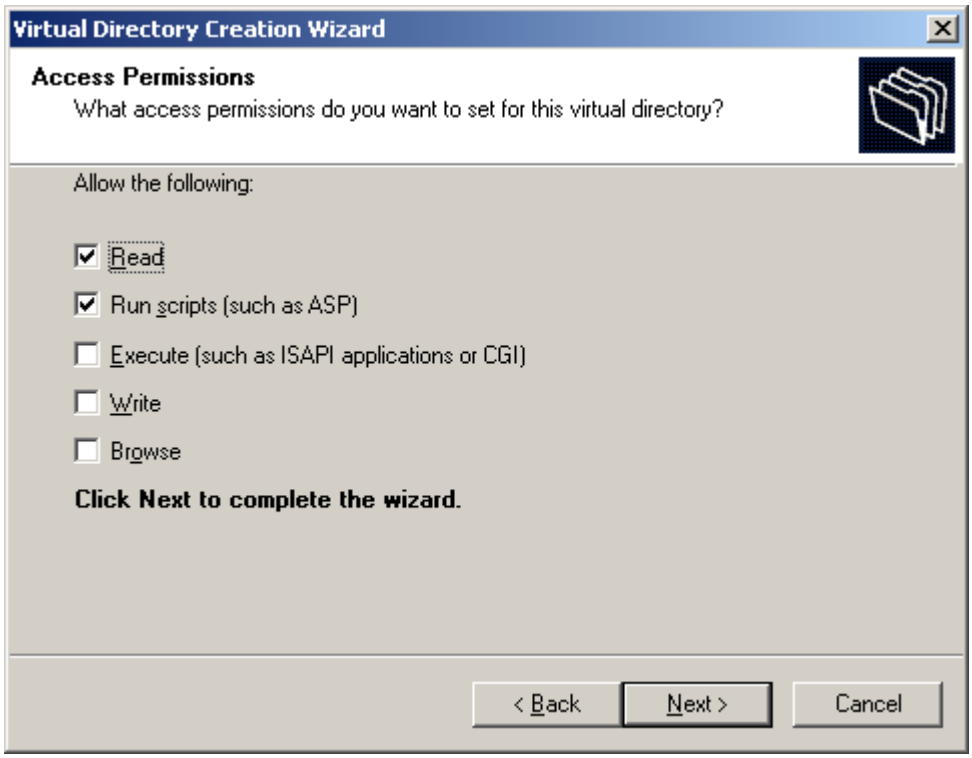

Figure 12: Access permissions for your Web application.

The new virtual directory is now listed in our IIS screen and shows the list of files found in the directory. IIS supports a list of file names as default files; typically, it is called deafult.htm or default.asp. This file is automatically launched when the user calls the above noted URL.

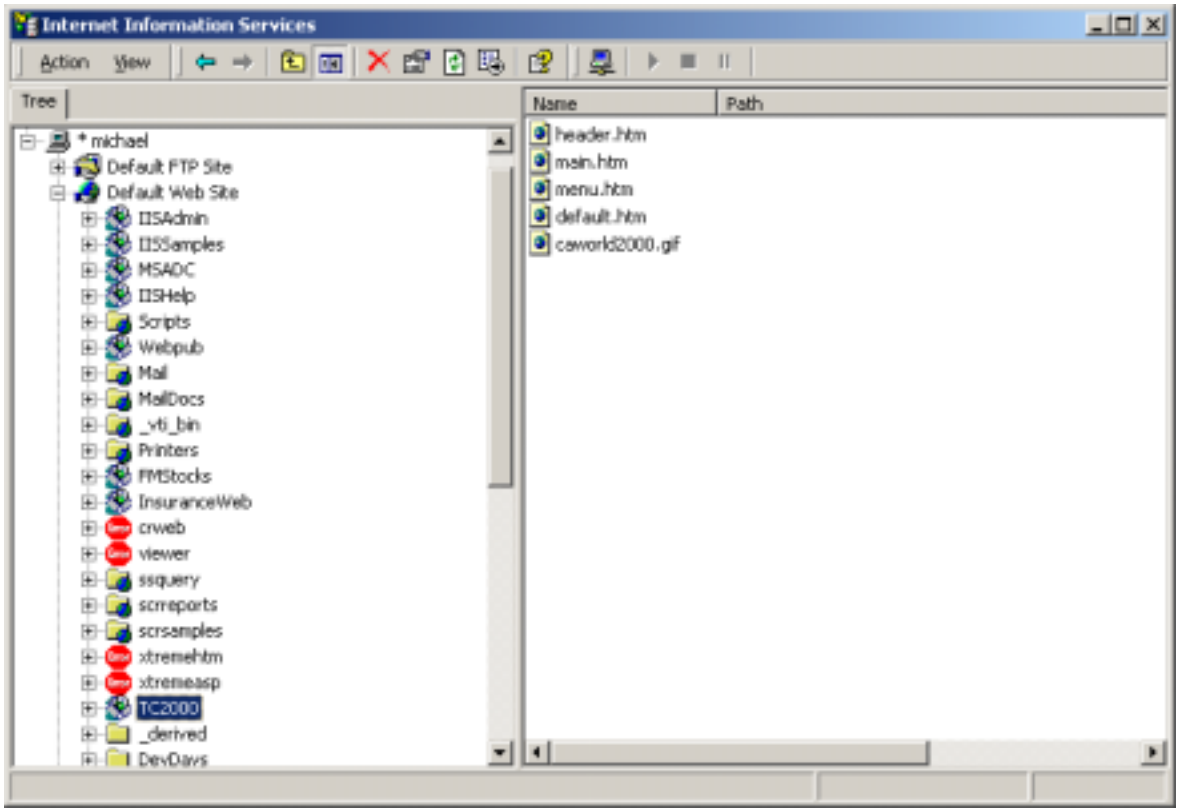

Figure 13: IIS after creating a new virtual directory.

Conclusion

Now that you understand the principles of designing and developing 3-tier applications, it is time to make the choice: select the best tools for the job, and start with your own ideas on creating business solutions following the principles we have learned here.

Bibliography

Mary Kirtland. *Designing Component based Applications.* Microsoft Press 1999.

Ross Chappel. *Using Microsoft Transaction Server with CA-Visual Objects 2.5.* TechniCon 1999 Session.

Office Developer Documentation. Microsoft Corporation, 1999.

Michael Ferber came into contact with dBASE III and Clipper in 1985 through a practical during his studies of applied mathematics at the University of Augsburg, Germany. His first project with Clipper was a medical documentation application for a neurological research project, followed by other commercial and medical programming projects during his studies. After working as a Senior Programmer for companies in Germany and Switzerland, he now lives with his family in Canada and works for UK-based Richmond Software in Toronto. Michael has spoken at conferences in Germany and the USA. You can reach him via CompuServe 100023,1153 or via email at FerberMichael@compuserve.com.