7. Part 2 of the Solution:
The Architecture Transformation Process

This chapter presents a simple and systematic process, which transforms existing WinBISs into the downloadable architecture for the purpose of deployability improvement. I will first outline the process from beginning to end. I will then discuss some activities of the process in detail.

7.1 The Client Program Architecture Transformation Process

To be precise, such BIS architecture transformation involves only the client programs. It is for this reason that I simply propose a client program architecture transformation process (see Figure 5). Repeat this process for each client program of a WinBIS, until they are all transformed into program warehouses, and then the WinBIS will be transformed into the downloadable architecture.

The process covers the various project management and engineering activities. The process begins with the activity that prepares a plan for the project. This activity and the activity that monitors and controls the project

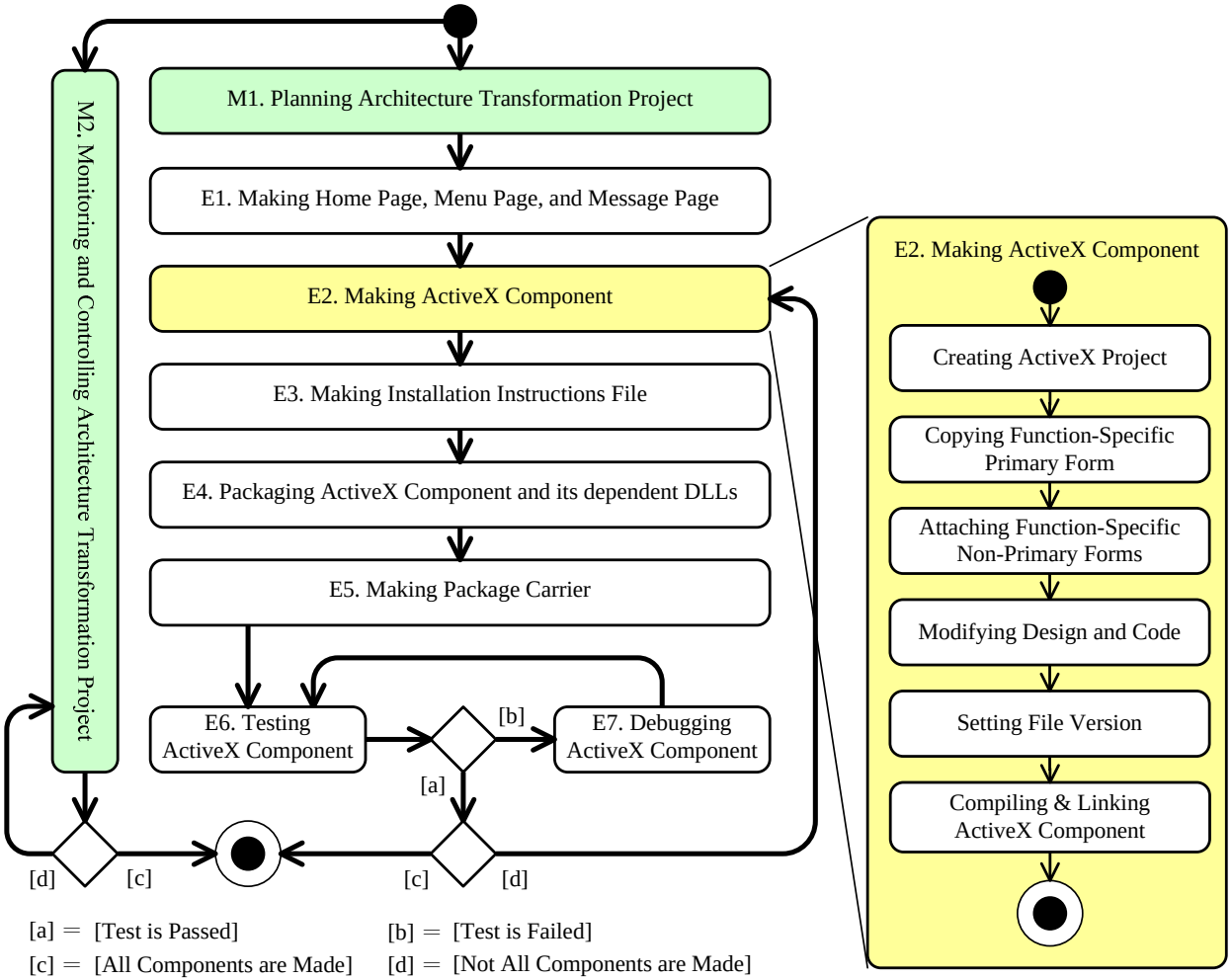


Figure 5. Client Program Architecture Transformation Process Model

against the plan are both typical project management activities. Engineering activities are repeatedly performed. The first engineering activity is to make the home page, menu page and message page. The next is to make the ActiveX component using a VB-like tool. The third is to make the installation instructions file. The fourth is to package the ActiveX component and its dependent DLLs. The fifth is to make the package carrier. The sixth is to test the ActiveX component and its dependent elements that include the installation instructions file, the package carrier, and DLLs. Finally, debug the ActiveX component and its dependent elements if the test fails.

Please note that signing CAB files that include component packages and DLL packages with a valid digital certificate is not necessary for Intranet- or Extranet-based downloadable BISs. A digital signature is a way to tell the end-user to accept the CAB files. It is an indication of the author of the CAB files. In an environment where everybody knows this CAB file is safe, and they are likely to accept it, the developer can ask the end-user to configure the browser's security settings to accept and download unsigned CAB files.

7.2 Making the ActiveX Component

The code of a specific function transforms into an ActiveX component by using a VB-like tool. Such ActiveX component-making activity consists of the following steps (see Figures 3, 4 and 5). Additional information about the VB-like tools can be found in the related documents (Borland, 2005; Microsoft, 1998; SyBase, 2004).

1. Create a new ActiveX component project. Such a project in VB is called "ActiveX control project", while in Delphi it is called "ActiveForm project". After that, the VB-like tool brings up a blank form that will embed in the package carrier. Such an embedded form in VB is called "UserControl form", in Delphi is called "Active form", and in this paper is called "embedded form".

2. Open the primary form of the function. Then copy all components and code of the primary form (except the VB-like tool auto-generated code) to the embedded form via the clipboard.

3. Add all non-primary forms of the function to the project.

4. Slightly modify the design and code of the function, such as to rid the embedded form of the pull-down menu; to modify the global variable related code in response to the change from global variable to session cookie; to write code for communication with Web scripts and other ActiveX components; or to manipulate the Web browser. The

modifications are unconcerned with business logic.

5. Specify the file version of the ActiveX component.

6. Compile and link the project to produce the ActiveX component.

7.3 Making the Installation Instructions File (INF File)

An INF file is a text file, which can be created or modified using any text editor in which the user can control the insertion of line breaks. It provides installation instructions that the browser uses to download and install an ActiveX component and DLLs required by the component. Moreover, an INF file is made up of different named sections. Some sections have predefined names and some sections have a name defined by the writer of the INF file. A line with a square bracket ([]) signals the start of a new section. The word inside the square bracket is the section name. All the lines that follow are considered the same section, until the next section starts. Note that I will confine our discussion to the needs of the downloadable architecture. For complete information about the INF files, refer to the related documents (Microsoft, 2007b).

The downloadable architecture uses three types of sections to control the download and installation of an ActiveX component and its dependent DLLs. I discuss these sections in more detail below. In addition, Figure 6

shows an example of the INF file for the MyProfile component. This INF file can be modified to download and install any ActiveX component by changing MyProfile's information to the desired ActiveX component's information.

7.3.1 The [Add.Code] Section

This section provides a complete list of all the files that end-users have to download and install. FileName=SectionName is used in this section to specify a required file, and to map the file to a section of the same name. During the actual installation process, the browser will install each file needed in reverse order, as stated in this section. Hence, the developer needs to ensure that the ActiveX component (OCX file) is listed first in this section, followed by dependent DLLs, with the least-dependent DLL at the bottom of the list.

7.3.2 The Component Section

This section contains all the information necessary to download and install the ActiveX component, and should be named the same as the ActiveX component's file name. The following lines are required in this section.

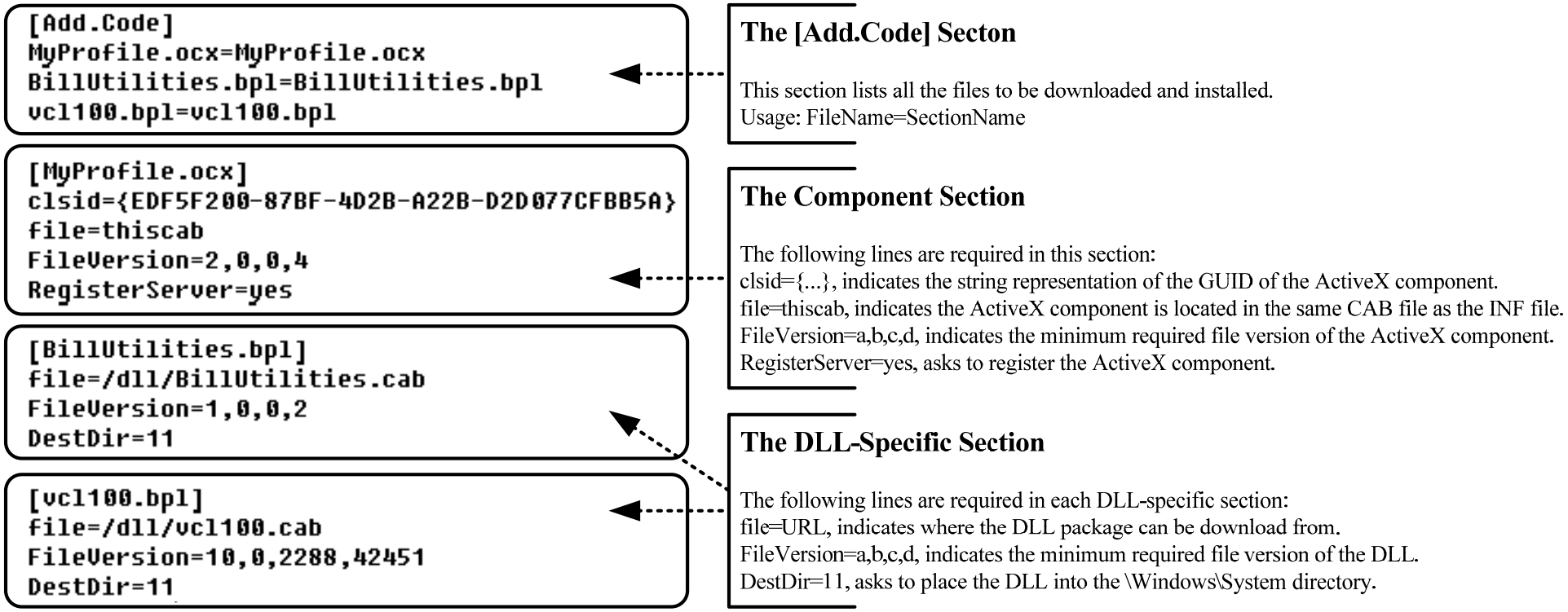


Figure 6. Example Installation Instructions File

1. clsid={12345678-1234-1234-1234-123456789ABC}. This line specifies the string representation of the GUID of the ActiveX component, enclosed in braces {}.

2. file=thiscab. This line indicates the ActiveX component is located in the same CAB file as the INF file.

3. FileVersion=a,b,c,d. This line specifies the minimum required file version of the ActiveX component.

4. RegisterServer=yes. This line asks to register the ActiveX component.

7.3.3 The DLL-Specific Section

Each DLL-specific section contains all information necessary to download and install a dependent DLL of the ActiveX component, and should be named the same as the dependent DLL's file name. The following lines are required in each DLL-specific section.

1. file=URL. This line indicates where the DLL package can be downloaded from.

2. FileVersion=a,b,c,d. This line specifies the minimum required file version of the DLL.

3. DestDir=11. This line asks to place the DLL into the \Windows\System directory.

7.4 Packing the ActiveX Component and Its Dependent DLLs

Each ActiveX component and its installation instructions must be packaged in a CAB file, called a component package. Each ActiveX component's dependent DLL must also be packaged in a CAB file, called a DLL package. Developers can use the Cabarc.exe command line utility, which comes with the Microsoft Cabinet Software Development Kit (Cabinet SDK), to create these two CAB files. The Cabinet SDK can be downloaded from this location: http://msdn2.microsoft.com/en-us/library/ms974336.aspx.

Component packages are created using the n command, followed by the name of the component package to be created, followed by the name of the INF file, in turn followed by the name of the OCX file, as shown in Figure 7. DLL packages are created using the n command, followed by the name of the DLL package to be created, finally followed by the name of the DLL file, as shown in Figure 8. For detailed instructions on using the Cabarc.exe utility, see the Microsoft Cabarc User's Guide (Cabarc.doc) included in the DOCS folder of the Cabinet SDK.

```
Usage：cabarc  n  PackageFileName  InfFileName  ComponentFileName
e.g.：cabarc  n  MyProfile.cab  MyProfile.inf  MyProfile.ocx
```

Figure 7. Packing the ActiveX Component Using Cabarc.exe

```
Usage：cabarc  n  PackageFileName  DllFileName
e.g.：cabarc  n  BillUtilities.cab  BillUtilities.bpl
```

Figure 8. Packing the Dependent DLL Using Cabarc.exe

7.5 Making the Package Carrier

To put it more precisely, package carriers have three functions: (a) to create and initialize an instance of a function-specific ActiveX component that the end-user wants to use; (b) to specify the minimum required file version of the ActiveX component, and the download location for the component package; and (c) to perform some user interface processes to increase the usability of the downloadable BIS.

The easiest way to create and initialize an instance of the ActiveX component, as well as to specify the file version of the ActiveX component and the download location for the component package, is to use the OBJECT html tag (see Figure 9). The important attributes of the OBJECT tag are summarized below.

1. CODEBASE: Specifies the download location for the component package. As an option, developers can specify the file version of the ActiveX component by attaching a number sign (#) and file version at the end of the CODEBASE path. If #version is not specified and a version of this ActiveX component is already installed on the end-user machine, no download occurs. If #version is specified and an earlier version of the ActiveX component is already installed on the end-user machine, the component package will be downloaded. If the ActiveX component is not present on the end-user machine, the component package will be downloaded whether #version is specified or not.

2. CLSID: Specifies the string representation of the GUID of the ActiveX component. The format is "CLSID:12345678-1234-1234-1234-123456789ABC".

3. ID: Specifies a document-wide name of the ActiveX component instance that client-side Web scripts can reference.

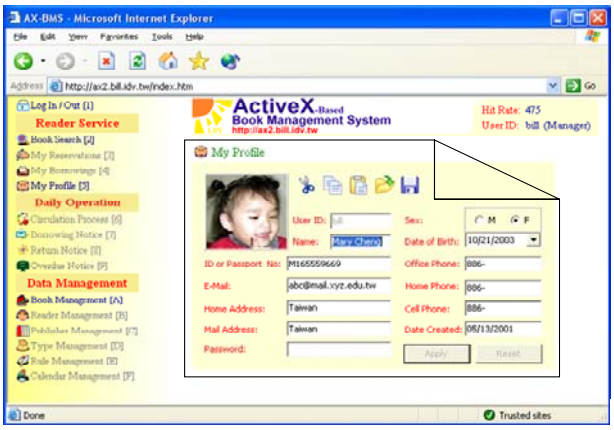
Unfortunately, Microsoft has made changes to the way that IE handles ActiveX components. Microsoft lost in a key Web browser patent ruling. The ruling forced Microsoft to redesign IE to get around the patent. Therefore, end-users cannot directly interact with ActiveX component instances created by the OBJECT tag. End-users can interact with such ActiveX component instances after activating their user interfaces, by

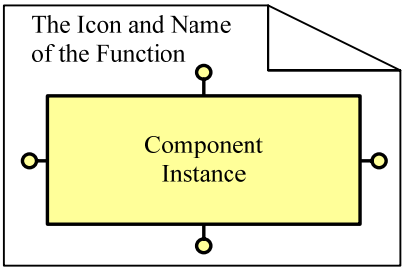
either clicking it or using the TAB key to focus on it and then pressing the SPACEBAR or the ENTER key (Microsoft, 2007a). For this reason, I will make use of an external script file to create and initialize an ActiveX component instance rather than using the OBJECT tag. ActiveX component instances created from external script files immediately respond to end-user interaction and do not need to be activated. Figure 10 contains an example of creating and initializing an ActiveX component instance using the external script file.

Additionally, package carriers perform three types of user interface processes that include (a) displaying an icon and a text string that identify the function to the end-user; (b) showing the end-user the ActiveX component is being downloaded and installed; and (c) avoiding showing a disabled ActiveX component instance. Figure 10 demonstrates such user interface processes.

```
<OBJECT
  CODEBASE=/MyProfile/MyProfile.cab#version=2,0,0,4
  CLSID=CLSID:EDF5F200-87BF-4D2B-A22B-D2D077CFBB5A
  ID=ax>
</OBJECT>
```

Figure 9. Example of Creating an ActiveX Component Instance Using the OBJECT Tag






```
<!-- MyProfile.asp - Example Package Carrier, is implemented using JavaScript and ASP -->

<%@ LANGUAGE=JavaScript %>

<HTML>
<HEAD>
   <META HTTP-EQUIV=Content-Type CONTENT=text/html;CHARSET=iso-8859-1>
   <STYLE>
      BODY {color:green;margin:0;margin-left:10}
   </STYLE>
</HEAD>

<!-- names the package carrier -->
<BODY ID=MyProfileAsp>

<!-- avoids to show a disabled ActiveX component instance -->
<%
if ( Request.Cookies("LoginName") == "" )
%>
   <TABLE><TR><TD><IMG SRC=/img/stop.gif></TD><TD>Please Login First</TD></TR></TABLE>
<%
else
{
%>
   <!-- begin to display information about the ActiveX component is being downloaded and installed -->
   <TABLE><TR>
      <TD><IMG SRC=/img/fun05.gif></TD>
      <TD ID=FunTitle>My Profile – Downloading <IMG SRC=/img/processing.gif></TD>
   </TR></TABLE>

   <!-- creates an ActiveX component instance by using an external script file -->
   <SCRIPT LANGUAGE=JavaScript SRC=MyProfile.js></SCRIPT>

   <SCRIPT LANGUAGE=JavaScript>
      //finish to display information about the ActiveX component is being downloaded and installed
      function showLoaded(){if(document.readyState=="complete")FunTitle.innerText="My Profile";}
      document.onreadystatechange=showLoaded;

      //initializes the ActiveX component instance
      ax.tabIndex=0;
      ax.focus();
      ax.color=0xE0FFFF;
   </SCRIPT>
<%
}
%>
</BODY>
</HTML>
```

// **MyProfile.js -** External Script File, is implemented using JavaScript.

```
var obj=document.createElement("object");                          //creates an ActiveX component instance
MyProfileAsp.appendChild(obj);                                     //inserts the ActiveX component instance into the package carrier
obj.codeBase="/MyProfile/MyProfile.cab#version=2,0,0,4";           //sets the URL for the component package
obj.classid="CLSID:EDF5F200-87BF-4D2B-A22B-D2D077CFBB5A";          //sets the GUID of the ActiveX component
obj.id="ax";                                                       //names the ActiveX component instance
```

Figure 10. Example Package Carrier