Issues in Migrating PERPOS to a New Development Environment

Sandra Laib
William Underwood

Technical ITTL/CSITD 07-010

October 2007
(Revised November 2008)

Information Technology and Telecommunications Laboratory
Georgia Tech Research Institute
Atlanta, GA

The Army Research Laboratory (ARL) and the National Archives and Records Administration (NARA) sponsor this research under Army Research Office Cooperative Agreement W911NF-06-2-0050. The findings in this paper should not be construed as an official ARL or NARA position unless so indicated by other authorized documentation.

ABSTRACT

The Presidential Electronic Records Prototype System (PERPOS) was developed to support archivists in processing Presidential e-records. Accession, arrangement, preservation, review, description, and creation of finding aids are supported. The data management system includes the schema for the repository as well as metadata for arrangement, review, preservation and description. PERPOS provides an environment for the experimental application of advanced information technologies to archival processes.

The development began with the creation of tools rapidly developed using the Visual Basic 6 (VB6) language and the Visual Studio Integrated Development Environment (IDE). This IDE is no longer supported by Microsoft. The question is: Can PERPOS be maintained by migration of the VB6 code to a development environment such as VB 2005 or Java?

This report describes the user interfaces, class structure and data models of the PERPOS system. An exercise in successful migration from Visual Basic 6 (VB6) to Visual Basic 2005 operating in the Net Framework is described. An exercise in successful migration from VB6 to JAVA is also described.

It is concluded that to improve the maintainability of PERPOS, the more complex projects of the PERPOS architecture should be refactored into smaller, simpler classes. Migration tools are available to support the migration of the Visual Basic 6 code to Visual Basic 2005 in the .NET framework or to Java. Due to interoperability of VB6, VB2005 and Java through the Common Object Model (COM), it is possible to migrate the VB6 code incrementally, rather than all at once. Since all our research prototypes are currently being implemented in Java, we are inclined to migrate the VB6 code for PERPOS to Java.

TABLE OF CONTENTS

TABLE OF FIGURES

1. Introduction

1.1 Background

The Presidential Electronic Records Prototype System (PERPOS) was developed to support archivists in processing Presidential e-records. Accession, arrangement, preservation, review, description, and creation of finding aids are supported. The data management system includes the schema for the repository as well as metadata for arrangement, review, preservation and description. PERPOS provides an environment for the experimental application of advanced information technologies to archival processes.

PERPOS was developed using a method known as evolutionary prototyping. An initial prototype was constructed to learn more about the problems of separating operating system and office application software files from user-created files and viewing personal computer (PC) files in legacy file formats. Once the prototype had been used in processing actual PC files from the White House Offices and the requisite knowledge gained, the prototype was adapted to satisfy the now better-understood needs.

Archivists who used PERPOS learned that there were some files that could not be viewed. The files included password protected or encrypted files, damaged files, and files in obsolete formats for which there were no viewers. The prototype was extended to include the capabilities to recover passwords from protected or encrypted files, to use the recovered passwords to decrypt files, to repair damaged files, and to convert obsolete file formats to current or standard formats for which there were viewers. Then the prototype was used again to process electronic records, more was learned, and the prototype was re-adapted based on archivist's recommendations.

Then the prototype was used again, more was learned, and the prototype readapted. This process of prototype use, learning and re-adaptation repeats until the prototype system satisfies all the needs and has thus evolved into a system. The resulting, but still evolving system, is an Electronic Records Repository and an Archival Processing System [1].

The development began with the creation of prototyped tools rapidly developed using the Visual Basic 6 (VB6) language and the Visual Studio Integrated Development Environment (IDE). Microsoft's Visual Studio has evolved into a development environment that no longer supports development of source code in the original VB6 syntax. Microsoft Visual Basic 2005 (VB8) is an object-oriented computer language implemented on the Microsoft .NET framework. The Microsoft .NET framework is a software component that can be added to or is included with the Windows operating system. A program developed using .NET is not compiled to machine language but to a language called Microsoft Intermediate Language (MSIL) or Common Intermediate Language (CIL). An MSIL application is executed by a virtual machine called the Common Language Runtime (CLR). The .NET framework is intended to make it easier to port applications to different platforms and to reduce the vulnerability of applications and computers to security threats.

1.2 Purpose

To maintain PERPOS as an environment for experimentation with advanced technologies, it is necessary to migrate the system to a current development environment. It is also desirable that the system be portable to Linux as well as current Microsoft operating systems. The purpose of this report is: (1) to provide an overview of the current architecture and data model of PERPOS, (2) to investigate the issues in migration of VB6 code to Visual Basic 2005, (3) to investigate the issues in migration from VB6 code to JAVA, and (4) to summarize the maintenance and migration issues.

1.3 Scope

The next section describes the user interfaces and class structure of the PERPOS system. Section 3 describes the PERPOS data model. Section 4 discusses issues in migration to Visual Basic 2005 operating in the Net Framework. Section 5 discusses issues in migration to JAVA. Section 6 summarizes the maintenance and migration issues.

2. PERPOS Object-Oriented Design

The Presidential Electronic Records PilOt System (PERPOS) was developed to assist archivists in both Systematic and FOIA processing of electronic records. It is comprised of three user-interface executable (exe) components and five dynamically linked library (dll) components. The three executable components are the Archival Repository Tool (ART.exe), the Archival Processing Tool (APT.exe) and The FOIA Search Application (FOIASearch.exe). Both the Archival Processing Tool and the FOIA Search Application have a COM interface that can return objects or send events when called from other applications. Four of the dynamically linked libraries have a COM interface as well. They are the Archival Assistant Tool DLL (AATVB.dll), the Archival Repository DLL (ARAVB.dll), the Manifest Library DLL (ManifestLibrary.dll), and the StorageMngmntAssist (StorageManagementAssistant.dll). The StorageMngmntAssist has a user interface as well. The last DLL is the AATVC.dll. This is a C-language API containing methods to construct TAR headers, calculate SHA-1 values for files, and translate dates from one format to another.

2.1 Archival Repository Tool

The Archival Repository Tool is one of the user interface components in PERPOS. This component is a standard exe and is the main interface to the Archival Repository component. The current implementation of this tool is in the Visual Basic 6 language using an Explorer style interface.

This component is designed to assist archivists with Accessioning electronic records, Describing record series, Systematic Case Management, and FOIA Case Management. Figure 1 shows how one of these activities can be selected in the user interface form.

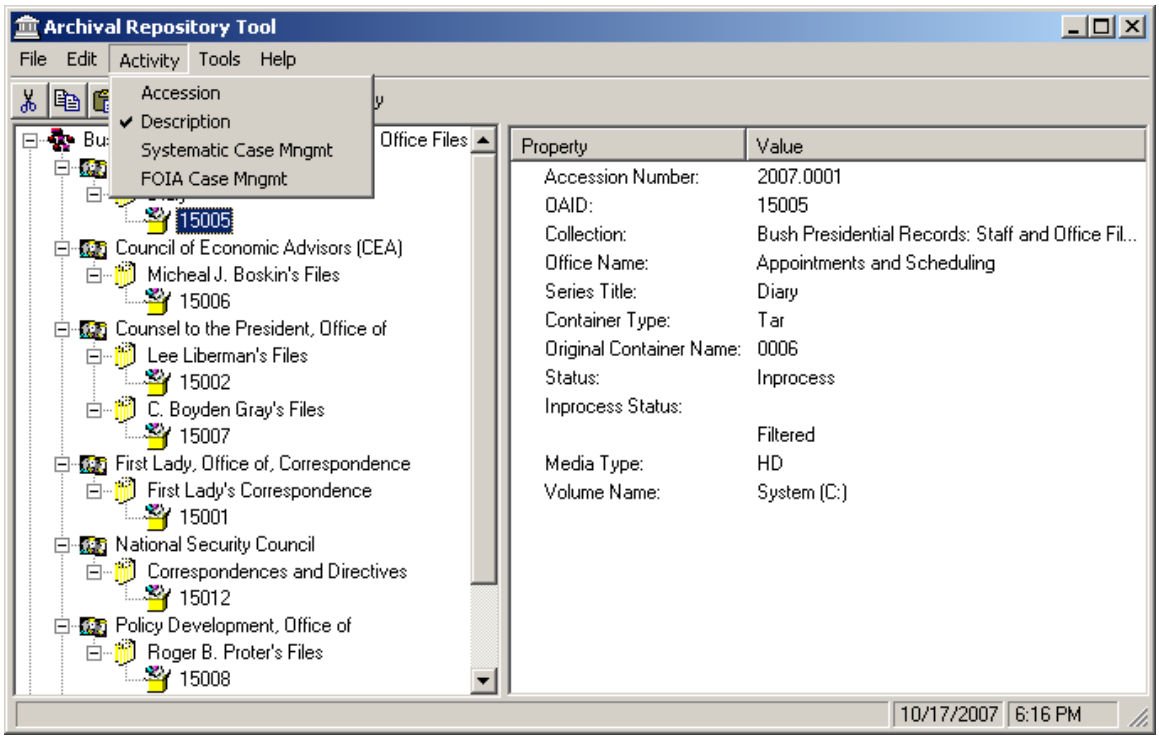

Figure 1. User Interface to the Archival Repository Tool

Figure 2 shows the class diagram for this component. This component is comprised of forms, modules, and classes. The main form for this component is the frmART form. This is the form that appears when the component is started. The main class is the Explorer class. This class gives the component the appearance of the Explorer interface. The frmART form delegates the behavior of the two main controls on the form to an object created from the Explorer class when the form is first displayed. These two controls are the TreeView control in the right-hand pane and the ListView control in the left-hand pane (see Fig. 1).

There are two main interface classes—IActivity and IExplorerItem. In Figure 2, these classes are represented as circles (or lollipops). An *interface* is a class from which no objects can be created and whose class name begins with an 'I'. Instead of creating objects, an Interface Class defines behaviors that all objects created from associated or derived classes must implement. This allows methods to be created that have an Interface as a parameter instead of a class, allowing all classes that implement that Interface to be passed to the method. It also allows for early binding and faster execution than with late binding. Early binding also allows for type checking of method calls in the compile stage instead of waiting for runtime errors.

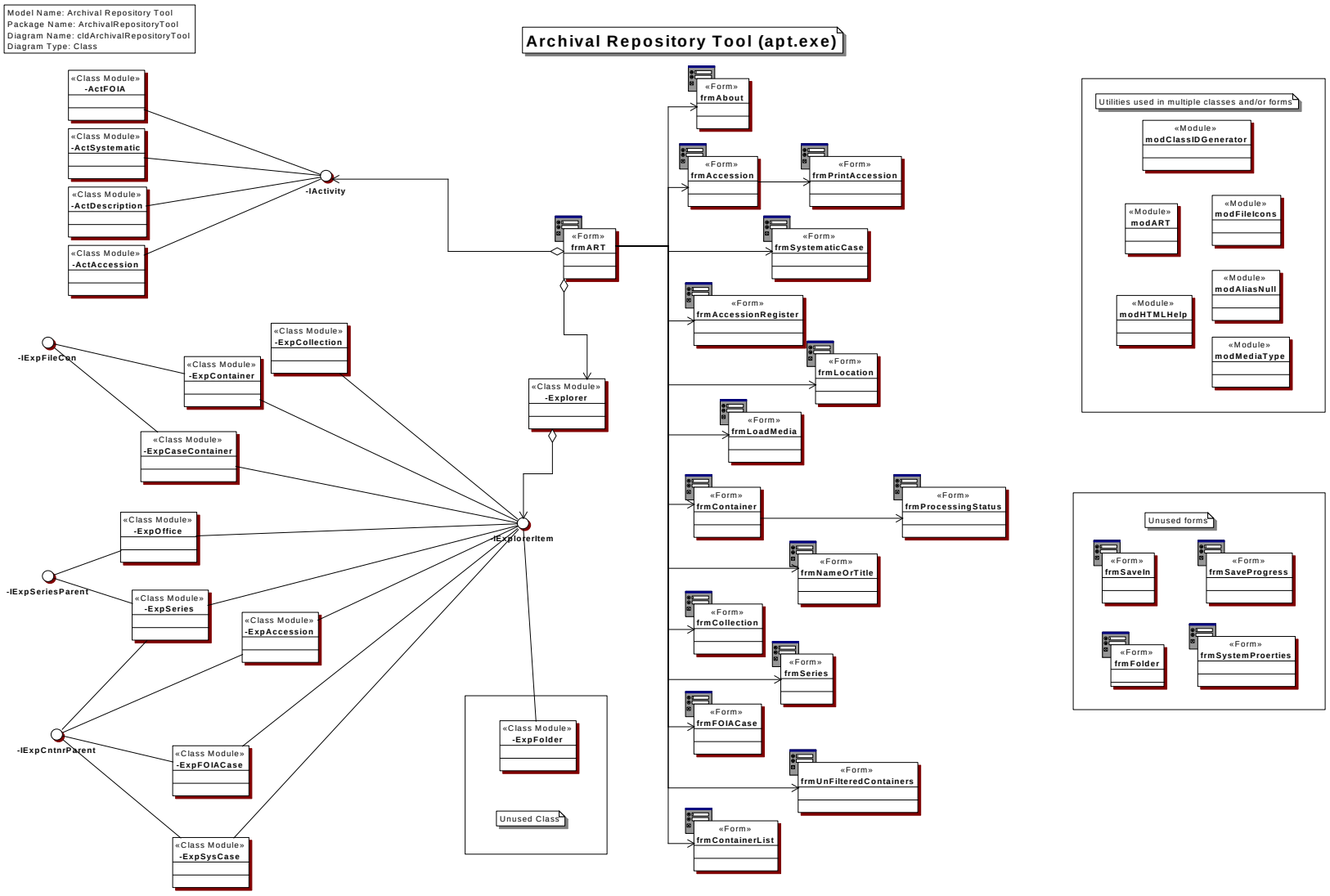

Figure 2. Class Diagram for the User Interface to the Archival Repository Tool

The IActivity class has four classes that implement its behavior. These four classes correspond to the four items on the Activity menu in Figure 1. Any object created from any one of these classes is considered to be an IActivity object. Each Activity class knows the root IExplorerItems that is associated with that activity. When the user selects an Activity from the Activity menu, an IActivity object is created from one of the four Activity classes. This IActivity object then creates the root IExploreItems associated with that activity.

The IExplorerItem class has nine classes that implement its behavior. These nine classes correspond to the nine types of items that can be displayed as a node in the TreeView control in the left-hand pane of the form. Any object created from one of these classes is an IExplorerItem object. These IExplorerItem objects are added to the Explorer object. When an IExplorerItem is selected in the left-hand pane, the Explorer queries it about what to display in the ListView control in the right-hand pane. When the user double-clicks on an object with a +, or clicks on a + symbol, the Explorer object asks the associated IExplorerItem object to return a collection of children IExplorerItems to be added to the Explorer object. To do this, each of the classes that implement IExplorerItem knows what type of children it is supposed to create and whether it can create any children at all. That is why there are nine classes that implement IExplorerItem. Each of the IExplorerItem classes corresponds to a class in the Archival Repository component. The convention here is to start class names that implement IExplorerItem with 'Exp' followed by the Archival Repository class name. An example of this convention would be class name ExpOffice to display information about the Office class object. In case of the ArchivalCollection, the IExplorerItem class is ExpCollection. The word *Archival* was added in front of the Collection class name because 'Collection' is a reserved word in the VB language.

The other interface classes are IExpCntnrParent, IExpFileCon, and IExpSeriesParent. The IExpCntnrParent interface is implemented by ExpAccession, ExpFOIACase, ExpSysCase, and ExpSeries. Originally, all of these classes could add a container, select a container and return a container parent object (not its IExplorerItem parent object). An example of a container parent is the FOIACase object in the Archival Repository. Some of these methods are now stubs. A container can be manually added to an Accession or a System Case, but not to a Series or a FOIACase. The select container allows the ExpContainer object to be highlighted after adding it to the Explorer object. Strictly speaking, these methods no longer are used for ExpFOIACase or ExpSysCase, since their children objects were changed to ExpCaseContainer objects.

The IExpFileCon interface was created because of the addition of the ExpCaseContainer class. Its main method returns the Archival Repository Container object. This allows the ExpContainer and ExpCaseContainer to be treated the same for opening and viewing the contents of a physical container.

Though it has been changed, the original design allowed a Series to have a SubSeries. That is why both the ExpOffice class and the ExpSeries Class implemented the IExpSeriesParent interface. They both could create ExpSeries IExplorerItems.

Each activity has its own set of menu items that can be activated. It also limits the forms that can be shown and the IExplorerItems that can be created. Figure 3 shows how selecting the Description Activity limits what forms are available and what IExplorerItems can be displayed. This diagram also shows child relationships between IExplorerItems that were too complex to show in Figure 2.

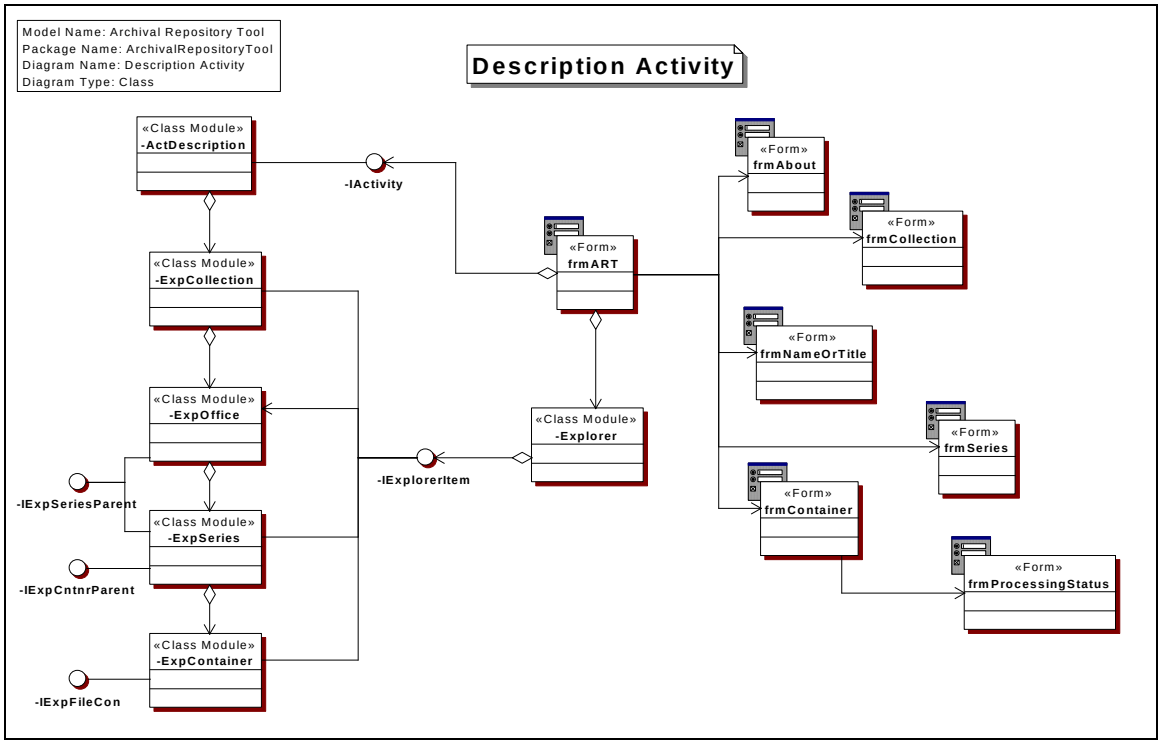

Figure 3. Classes and Forms of the Description Activity

2.2 Archival Repository DLL

The Archival Repository DLL (ARAVB.dll) is a business object and business object data transfer component. It is the main interface to the Archival Repository Database (ART.mdb).

The class diagram for the ARAVB.dll is shown in Figure 4. The main public class in this component is the Repository class. This class is a factory class. A *factory class* is a creatable class that is used to create and initialize other classes in the component. In this component, the Repository class is the only creatable public class and is used to create all other public classes.

The main private class in this component is the business object data transfer class (BODT). This class contains all the references to the ART.mdb, its tables and queries.

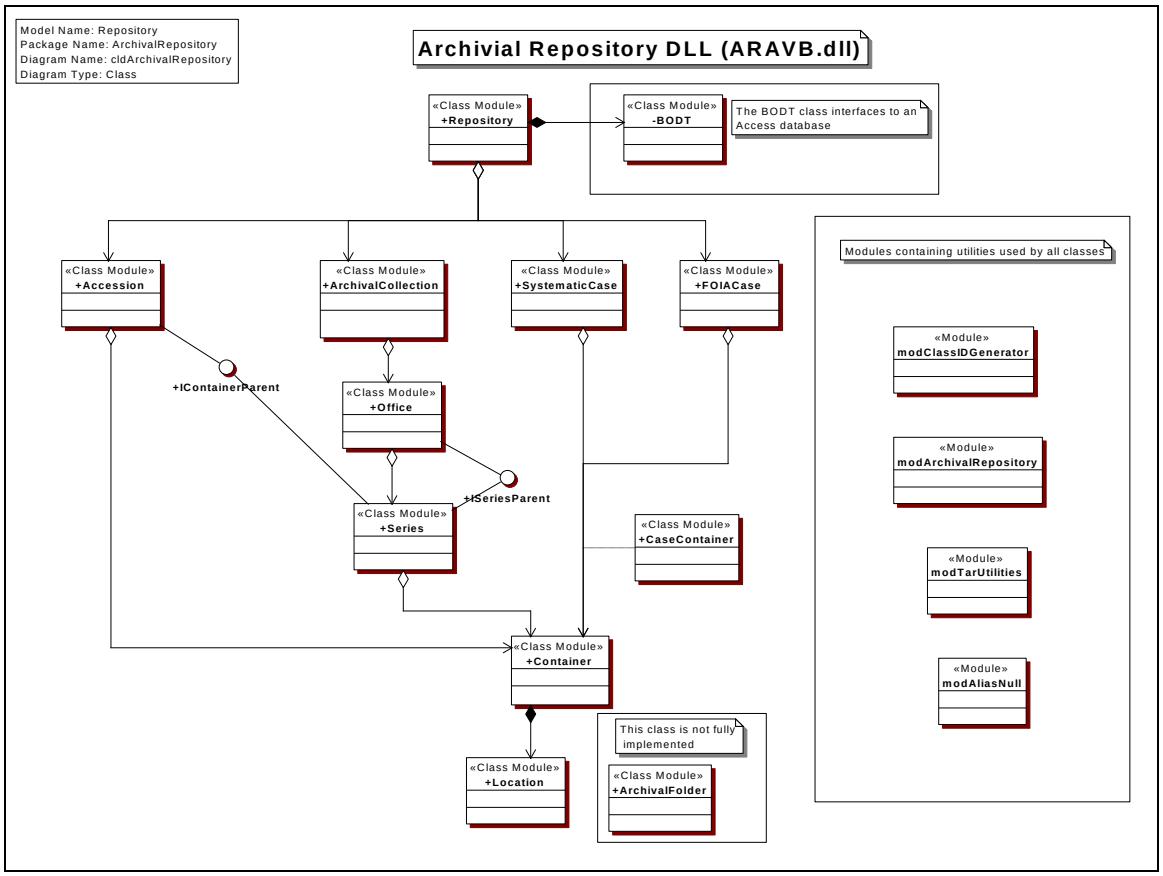

Figure 4. Class Diagram for the Archival Repository DLL

2.3 Archival Processing Tool

The Archival Processing Tool is another user interface in PERPOS. This component (APT.EXE) is the main interface for both systematic and FOIA processing of accessioned containers. The tool is implemented in the Visual Basic 6 language using an Explorer style interface. This component is an ActiveX exe. This means that it can be run as a stand-alone program or it can be called from inside another component such as the Archival Repository Tool.

This component is designed to assist archivists with processing containers of electronic records. This is accomplished by assisting the archivist in exploring the contents of a container, filtering a container to remove non-record files (if needed), rearranging the contents (if needed), preserving e-records, and finally reviewing the contents of individual files for FOIA exemptions and PRA restrictions. Figure 5 shows the activities that can be performed on the records in a container.

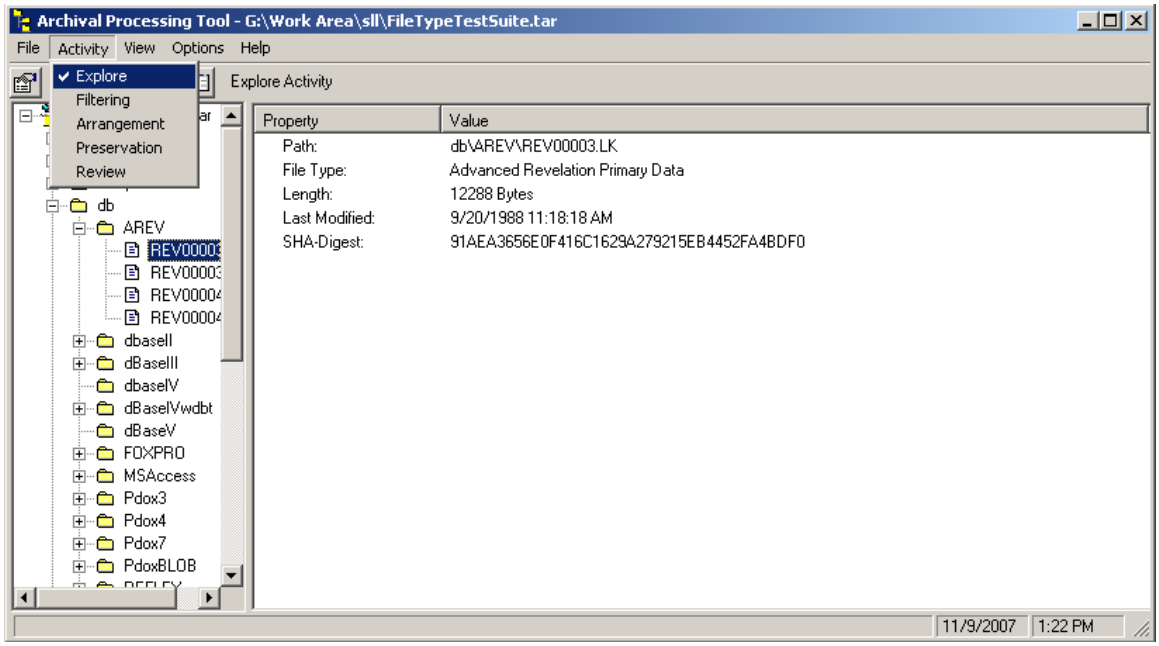

Figure 5. User Interface to the Archival Processing Tool

Figure 7 shows the class diagram for APT.exe. Like the Archival Repository Tool (ART), this component is comprised of forms, modules, and classes. Unlike the ART component, this component does not start up in the main form. This component starts up in the Main method in the modAPT.bas module. Because of the length of time required to start up the main form frmAPT, a splash form is the first form shown on startup as a stand-alone component. The splash form is conveniently named frmSplash. The main form for this component is the frmAPT form. This form appears when the component is used to open a container from inside another component or after the frmSplash has appeared. A container can be a file system directory or a tar file.

The main class is the CExplorer class. This class helps give this component the Explorer appearance. The frmAPT form delegates the behavior of the two main controls on the form to an object created from this class when the form is first displayed. These two controls are the TreeView control in the right-hand pane and the ListView control in the left-hand pane. This class is slightly different from the Explorer class in the ART component. It has been given extra functionality to enable it to assist the archivist in Arrangement and other activities. The CExplorer class allows both files and folders to be opened from the ListView control (or right hand pane) as well from the TreeView control. It also allows drag and drop activities in the TreeView control. Items can be dragged from one location in the TreeView control and dropped in another location, or dragged from the ListView control and dropped in the TreeView control. The drag and drop capabilities were added to assist in the Arrangement Activity.

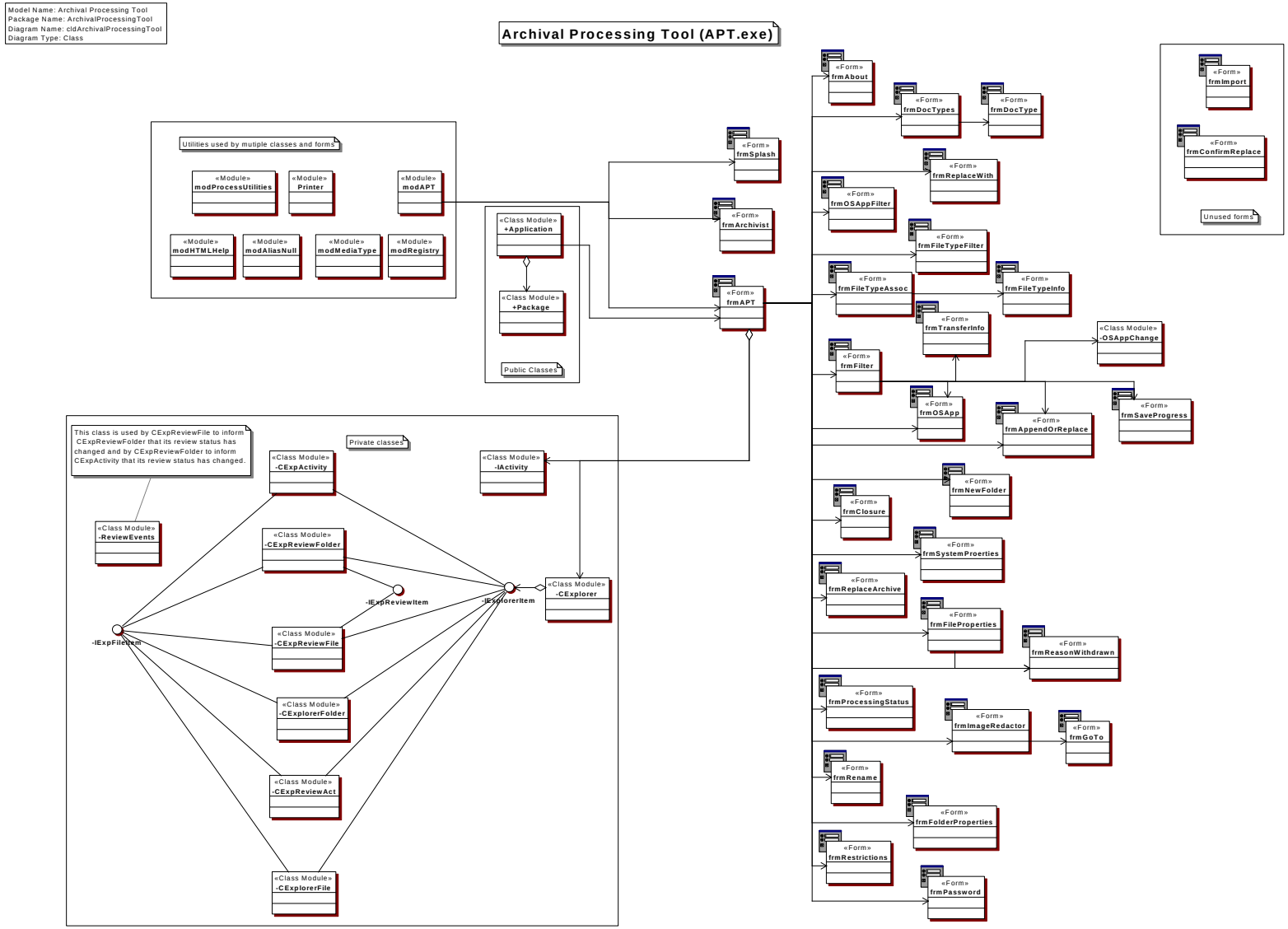

Figure 6. Class Diagram for the Archival Processing Tool User Interface

Unlike the ART component there is not an Activity class corresponding to each activity. There are only two Activity classes and they are for the two types of display. One type of display consists of container, folder and file icons. The other type of display is of review container, folder and file icons. A Review container has a container icon plus an additional checkbox that can be checked, partially checked, or empty. Review folders have an icon that contains a folder icon plus an additional checkbox that can be checked, partially checked, or empty. Review files have a file icon that can be different colors depending on the review status. Only the Review activity uses the review folder and file icons. All other activities use the simple folder and file icon displays. Figure 5 shows an explorer activity that uses the simple container, folder and file icons. Figure 6 is a Review Activity display that shows files that have not been reviewed as a white document icon. It also shows an open file as green and a closed file as red. It shows partially checked and unchecked folders. It also shows a partially-reviewed container and folder icons.

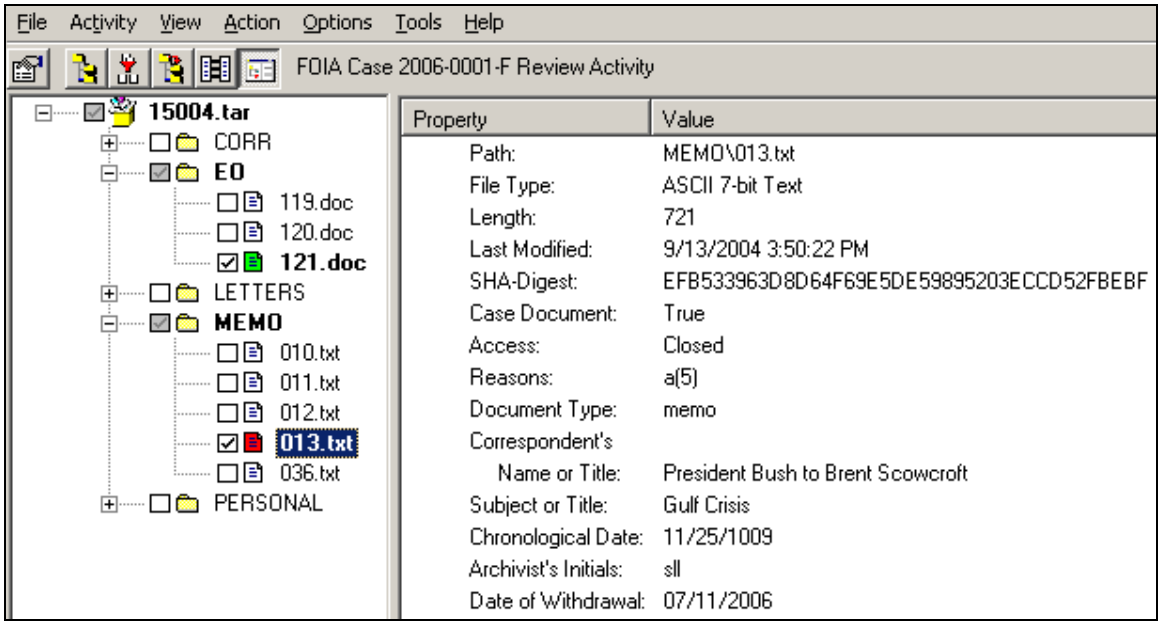

Figure 7. A Container Opened in the Review Activity

2.4 Archival Assistant Tool

The Archival Assistant Tool (AATVB.dll) is a business object and business object data transfer component. The AATVB.dll is the main interface to several temporary databases. The class diagram for the AATVB.dll is shown in Figure 8. The main public class in this component is the ArchivalAssistant class. This class is a factory class. It is used to create and initialize other classes in the component. In this component, the ArchivalAssistant class is the only creatable public class and is used to create all other public classes. There are several private classes in this component devoted to the business object data. These classes contain the references to the temporary databases, their tables and queries. The class names for each of these classes ends with the suffix 'BODT'.

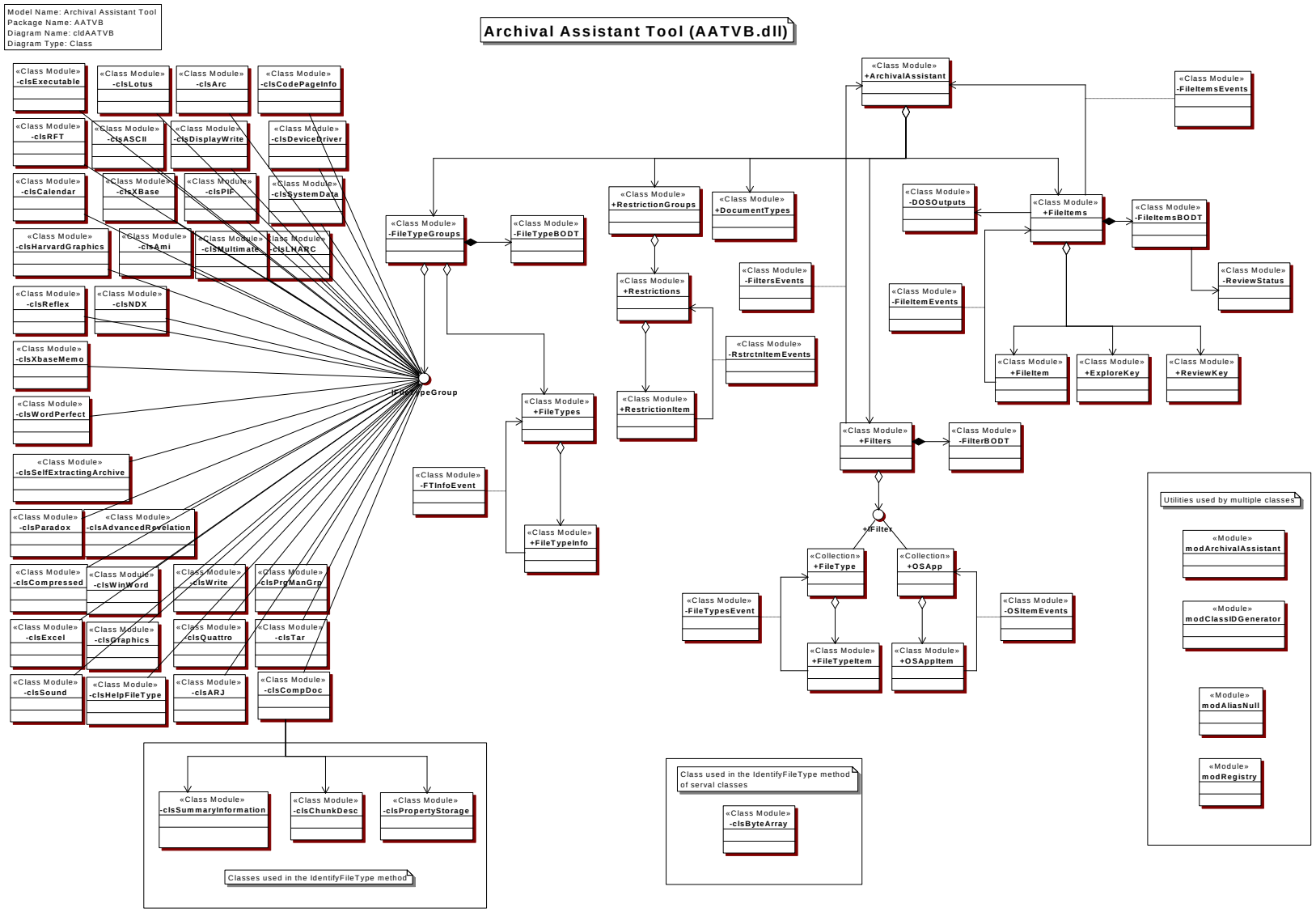

Figure 8. Class Diagram for the Archival Assistant Tool

There are several intermediate notification classes. The class names for each of these classes ends with the suffix ‘Events’, and in one case ‘Event’. These classes are used in aggregate situations. There is one of these classes in each case where children classes need to send an event back to its parent. A single object of one of these classes is created WithEvents in the parent object with a reference being passed to each of the children when they are created. The child class calls a method in the notification object that then sends an event to the parent. In most cases, these classes are attached to an association in the diagram.

This component is the most complex of all components. In the beginning, there were no databases tied to this component. All the information about the contents of the container was in the container, either in the tar header or the manifest file. No complicated searches were necessary, so the container was loaded into a collection. Only filtering information was included before the SHA-1 hash code was included for unique file identification. This information was kept in a comma-delimited file. Later information about exemptions and restrictions was added as the review activity implemented. Additional information was handled as a separate section of the comma-delimited file. To support the review activity, document types were added. Some user-defined file type information was added. A section was added and represented as tab-separated values. The user could add or change the possible extensions to the filename of a file format type and specify a command line with parameters that could be used to open a file with a program other than Quick View Plus.

Additional functionality was handled by the same component, since there was a single data file (APT.dat). The data file was read by the ArchivalAssistant class which created an additional class and used the data to initialize its attributes. The filtering information has since been moved into database files that can be loaded. A default empty filter database file is created when the ArchivalAssistant class first initializes. A file type database is created from the knowledge contained in the class implementing IFileTypeGroup. A default DocumentType collection is created. Default collections are created for PRA restrictions, FOIA exemptions and MISCellaneous exemptions. When the APT starts, the APT.dat file is read and the defaults are replaced. This approach was adopted because the APT.dat file could be damaged or be accidentally deleted and the system still had to work.

In hindsight, it would be better to break this component into several smaller components. The filtering information is already in its own database. The necessary business objects and their business object data transfer class could be moved into their own dll. The access restrictions and exemptions and their necessary business objects and their business object data transfer class should be moved into their own dll. This way the Image Redactor could be moved out of the APT.exe and access the new dll for redaction information.

The FileTypeGroups and the necessary business objects and their business object data transfer class could be moved into its own dll. The drawback to doing this in the past was the necessity of the FileTypeGroups and the FileItems classes to be in the same dll so that an entire tar file could be filtered at one time by only opening the tar file once. This could

be worked around by creating a memory-mapped file that one process could open, but both processes could access.

2.5 Manifest Library

The Manifest Library is a business object and business object data transfer component. The class diagram for the ManifestLibrary.dll is shown in Figure 9. It has one public creatable class, the Manifest class. Its business object data transfer class is the ManifestBODT class. This class is different than the BODT classes in other components because it does not connect to its own database. After the Manifest object is created it has a method that adds its tables to an existing database. The Archival Assistant Tool and the Archival Repository components both use the Manifest Library.

Components call the Manifest method 'AddToDatabase' with a connection string. The Manifest object calls its ManifestBODT object to add the tables and the queries that it needs for a database. ManifestBODT uses these tables and queries. The other BODT's know enough about the structure of the Manifest table to have queries of their own to interact with it. For example, the FileItemsBODT has a query that uses tFileItems and the Manifest table to calculate the percent of all folders reviewed. The Archival Repository's BODT has a query that updates the Manifest tables CaseDoc field based on the CaseDocument table.

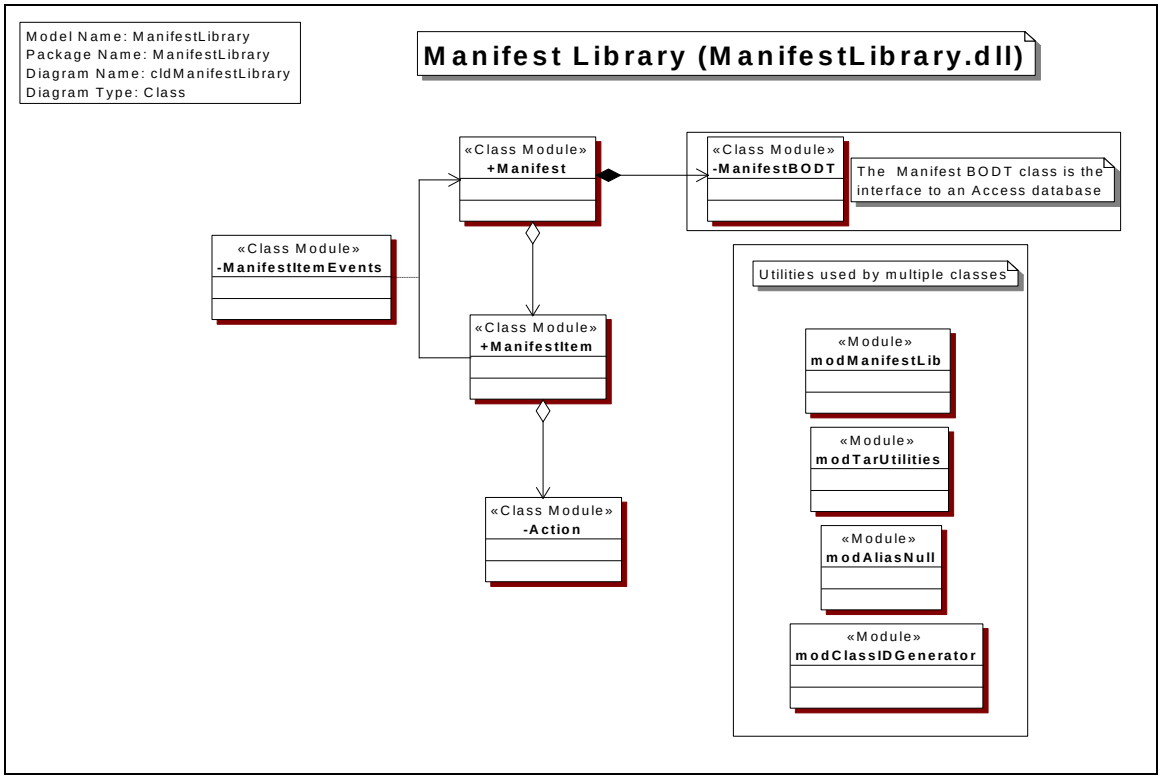

Figure 9. Class diagram for the Manifest Library DLL

2.6 Storage Management Assistant

The Storage Management Assistant dll is used to interface to the system registry where the information is stored about storage locations for the accessions, the repository, the work area and the library. This component also has a user interface so that more than one component can use, modify or display the information with the same look and feel. Its class diagram is shown in Figure 10.

Prior to Accessioning containers, the APT has to be opened by the Administrator user to assign a storage location for the accessioned containers. This is because in the early stages of prototype development, there was only the APT. Once it was decided containers had to be accessioned, this ability should have been moved to ART. If this is done, ART can be the first application that is run after installation, instead of opening an empty APT just to setup were to put accessioned containers.

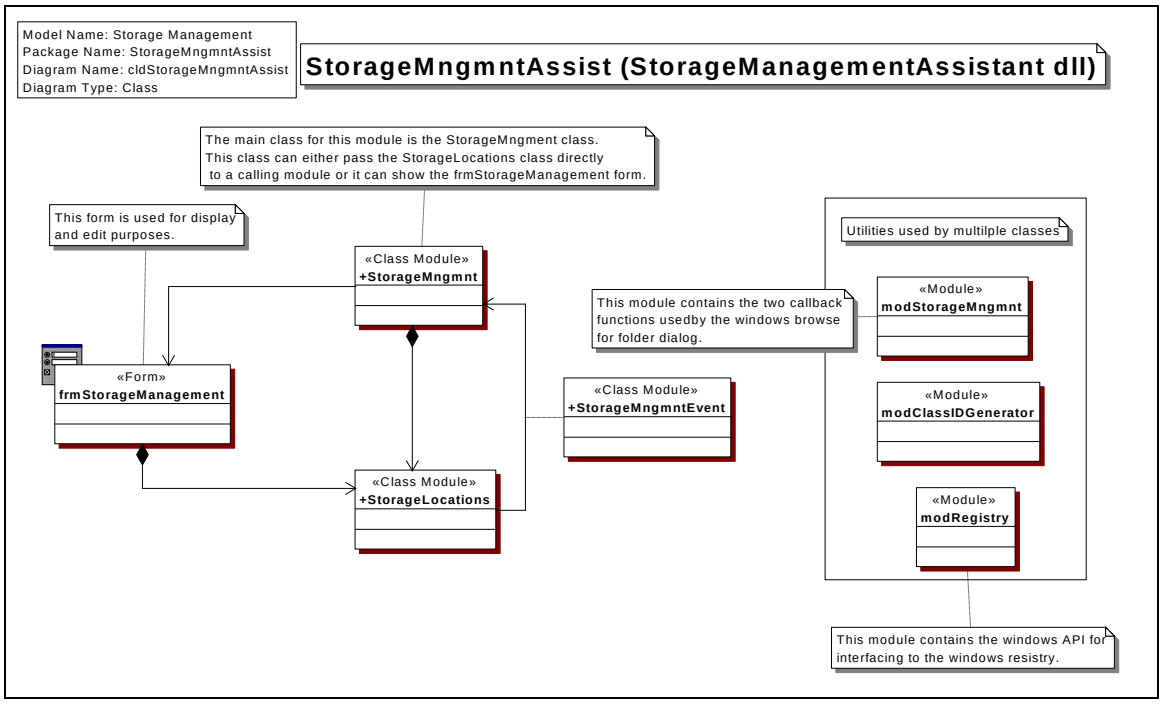

Figure 10. Class diagram for the Storage Management Assistant DLL

2.7 FOIA Search Application

The FOIA Search Application is a user interface component. Though this component is an exe, like the APT.exe, it is a COM object that is called from other components. Also like the APT.exe, the application starts up in the Main method in the modFOIASearch module. If the application does not start up in automation mode, the frmSearch is shown. When an automation object is called by another application, this is called *automation mode.* When it is called in stand-alone mode the result set is saved to a file.

This component is designed to assist archivists in searching for electronic records related to a FOIA request. Once a search query has been entered, this component displays the relevance of items in the result set, allows an archivist to view the individual items in the results set, and enables the archivist to limit the results based on a relevance number. Its class diagram is shown in Figure 11.

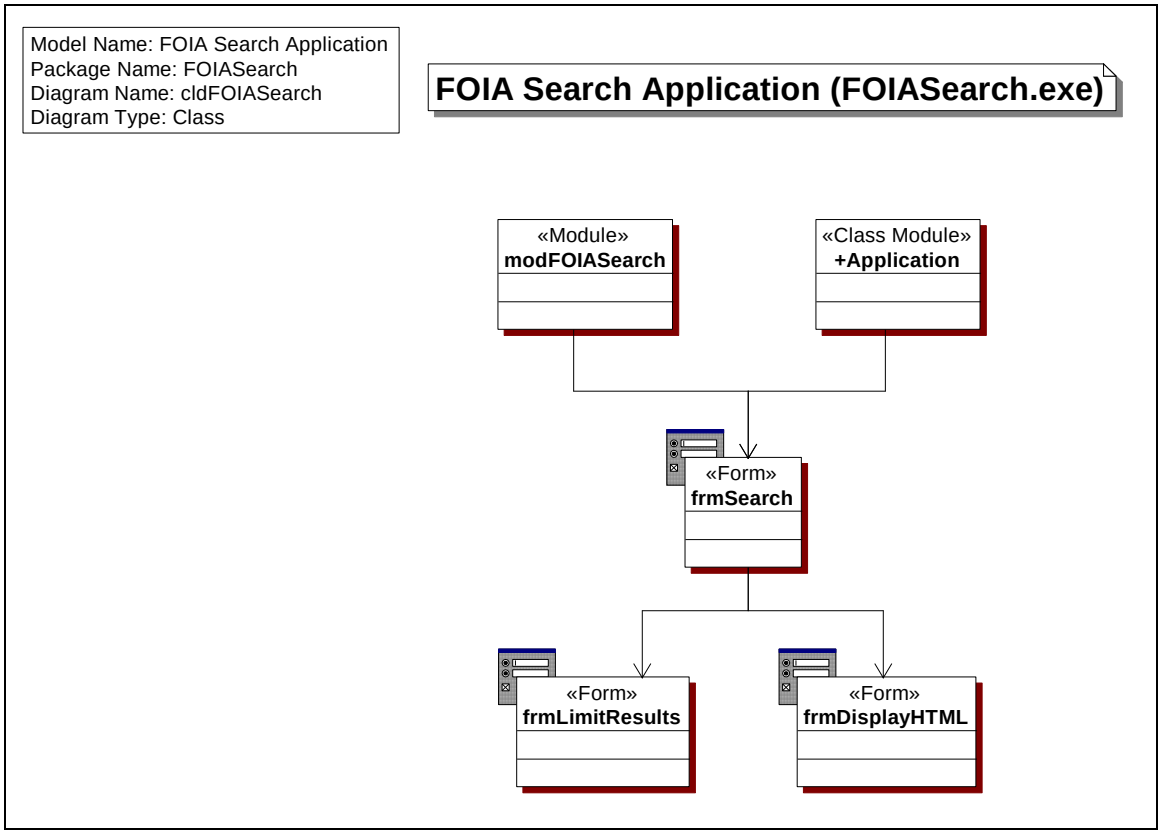

Figure 11. Class Diagram for the FOIA Search Application

3. PERPOS Data Models

3.1 Archival Repository Database

The Archival Repository Database is the primary database of PERPOS. It contains the accession register, archival catalog, and systematic review and FOIA cases. Figure 12 shows the 16 tables in this database and their relationships to one another. The three tables in the box are added by the ManifestLibrary component and are used when containers are being referenced. In addition to the tables, 104 queries are stored in the database, 28 of which are created by the ManifestLibrary component. The ManifestLibrary tables are temporary. Whenever a container's Manifest is referenced, the old Manifest tables are deleted and new ones created.

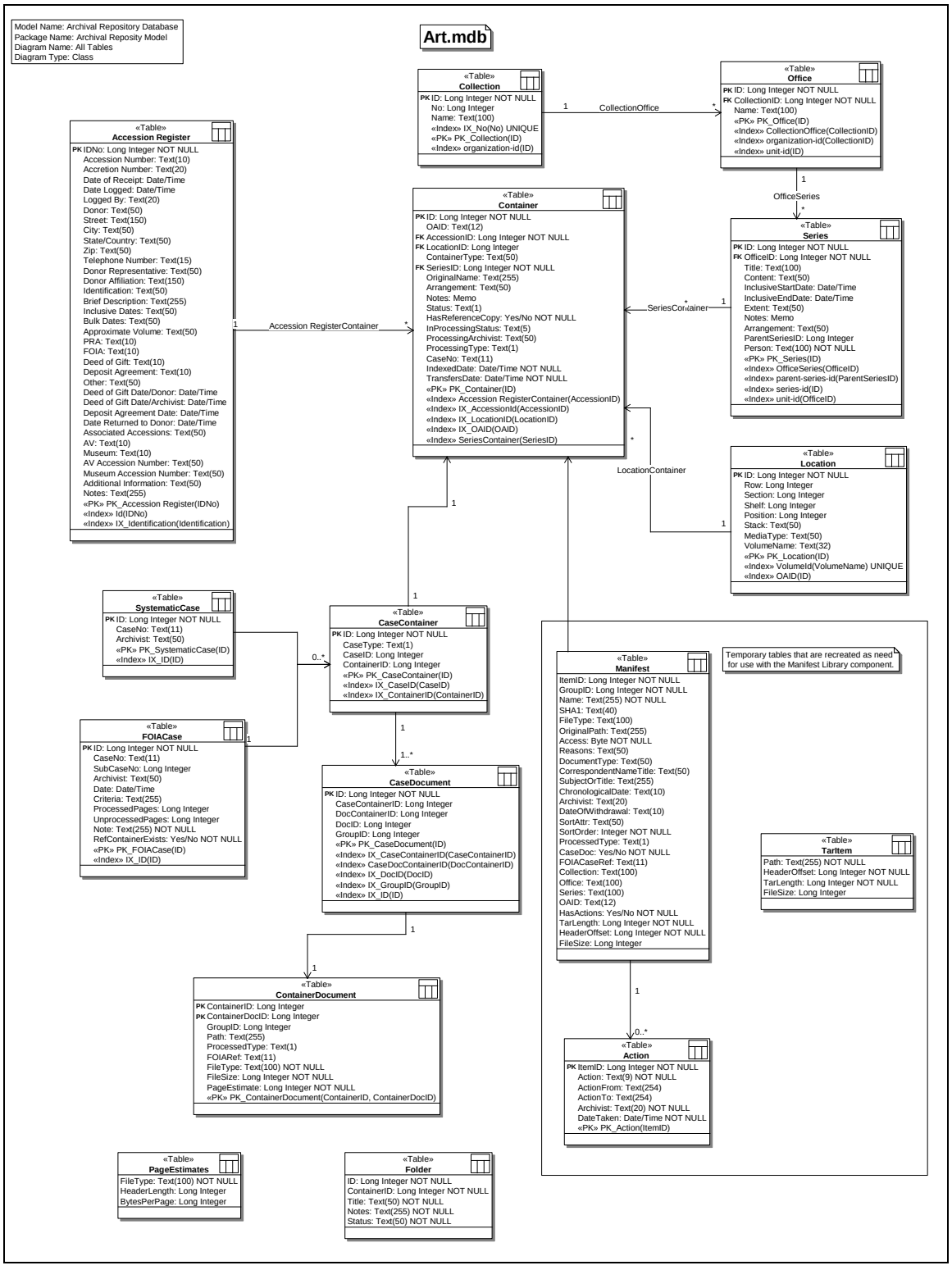

Figure 12. The Archival Repository Database Model

The relations in the diagram that are named were created using the relationships interface in Microsoft Access. They are part of the original design for the database and enforce referential integrity. This means that Access will return an error if a record is deleted while it is referenced by another table. This also means that if the data in several tables is to be deleted, a strict order of deletion must be maintained.

The relationships that are not named do not enforce referential integrity. Integrity for the unnamed relationships is enforced by the business and business data transfer objects. Most of the relationships occur as part of a query definition.

The Archival Repository Database is stored in the Art.mdb. This database is stored in the "Archival Tools" folder under the hidden "Application Data" folder for "All Users" under the "Documents and Settings" folder. The database was placed in this location so that all users could have read and write privileges to its contents. The database is under a hidden folder so that it is only accessed through the user interface in the Archival Repository Tool described in section 2.1. The actual interface to the main database tables and queries is through the ArchivalRepository.dll component described in section 2.2.

3.2 Archival Assistant Tool Databases

The ArchivalAssistant.dll in the Archival Assistant Tool component creates three temporary Access databases that are described in the following sections. It also has methods to load one user data file, usually the apt.dat, and split it into three temporary text files or to create the three text files from default information. The first text file contains user-defined Document types. The second text file contains FOIA exemptions, PRA restrictions and Misc restrictions. The third text file contains user-defined file type information such as file name extensions and associations with converters, extractors, password recovery tools and decryptors.

3.2.1 FileType Database

The FileType information is stored in a temporary database file that is created when the FileTypeGroups object is created by the ArchivalAssistant object in the Archival Assistant Tool component. Though there is only one table in this file, there are 16 queries (or stored procedures). This is a full Microsoft Access database. The FileTypeGroups object contains a FileTypeBODT object which creates the database. The fields and the index of the table are shown in Figure 13.

The FileTypes table is initially filled from an array of information about known file types passed to FileTypeBODT from the FileTypeGroups object. The FileTypeGroup compiles the array by collecting information about file types from each of the classes implementing IfileTypeGroup. After the user data file has been read, the user-defined information is added.

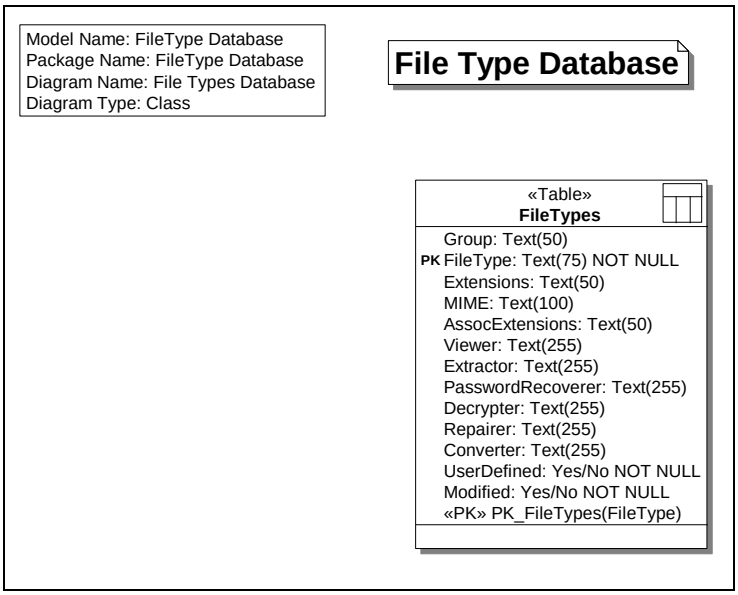

Figure 13. The File Type Database Model

3.2.2 Filter Database

When the ArchivalAssistant object is created from the Archival Assistant Tool component, an empty Filter database is created in the user's temporary files location. This database has three tables (shown in Figure 14) and 15 queries. The ArchvialAssistant Tool has methods to save a Filter to a user-defined location and to load a filter that was previously created. When the method to load a Filter is selected from the ArchivalAssistant, a copy of the database is placed in the user's temporary directory. This allows the user to choose whether or not to save changes made during a session.

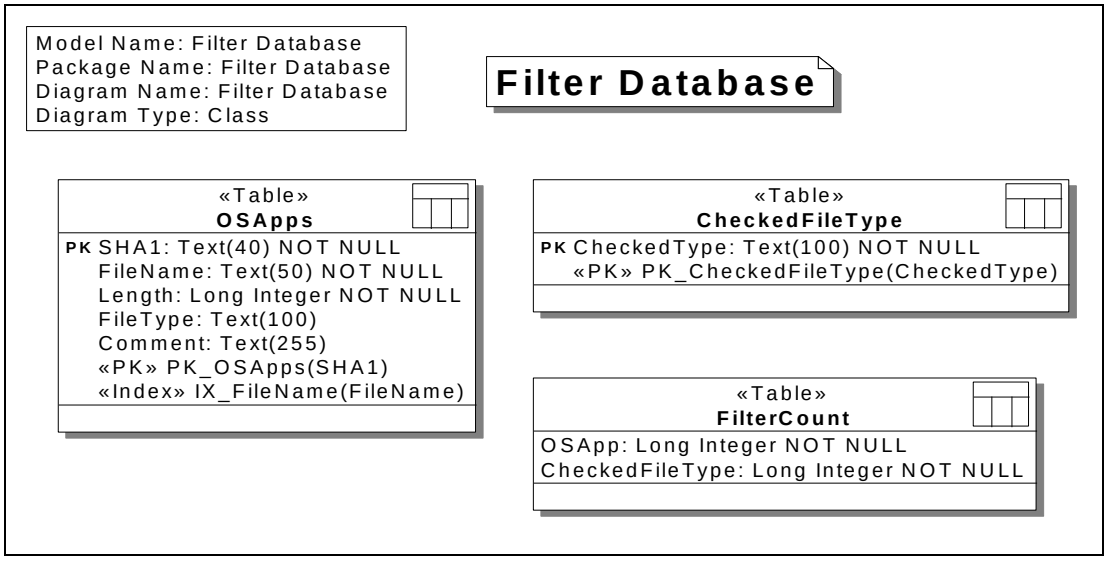

Figure 14. The Filter Database Model

The file name extension on a filter database is .flt even though it is a Microsoft Access database. The filter that is loaded onto the user's machine during installation is the OS-App Hash Code and File Type.flt. This filter was created while filtering actual Bush Presidential e-record containers.

3.2.3 FileItems Database

The FileItems database is stored in a temporary database that is created when a container is opened. Its data model is shown in Figure 15. Additional FileItems databases are created during the filtering activity. This database is created by a FileItemsBODT object in the Archival Assistant Tool component each time a FileItems object is created. The FileItemsBODT creates the FileItems table and 46 queries. The FileItems table is filled mainly from the tar header of a container or the tar header information is collected through the use of the AATVC.dll API. There are additional fields that are used during processing.

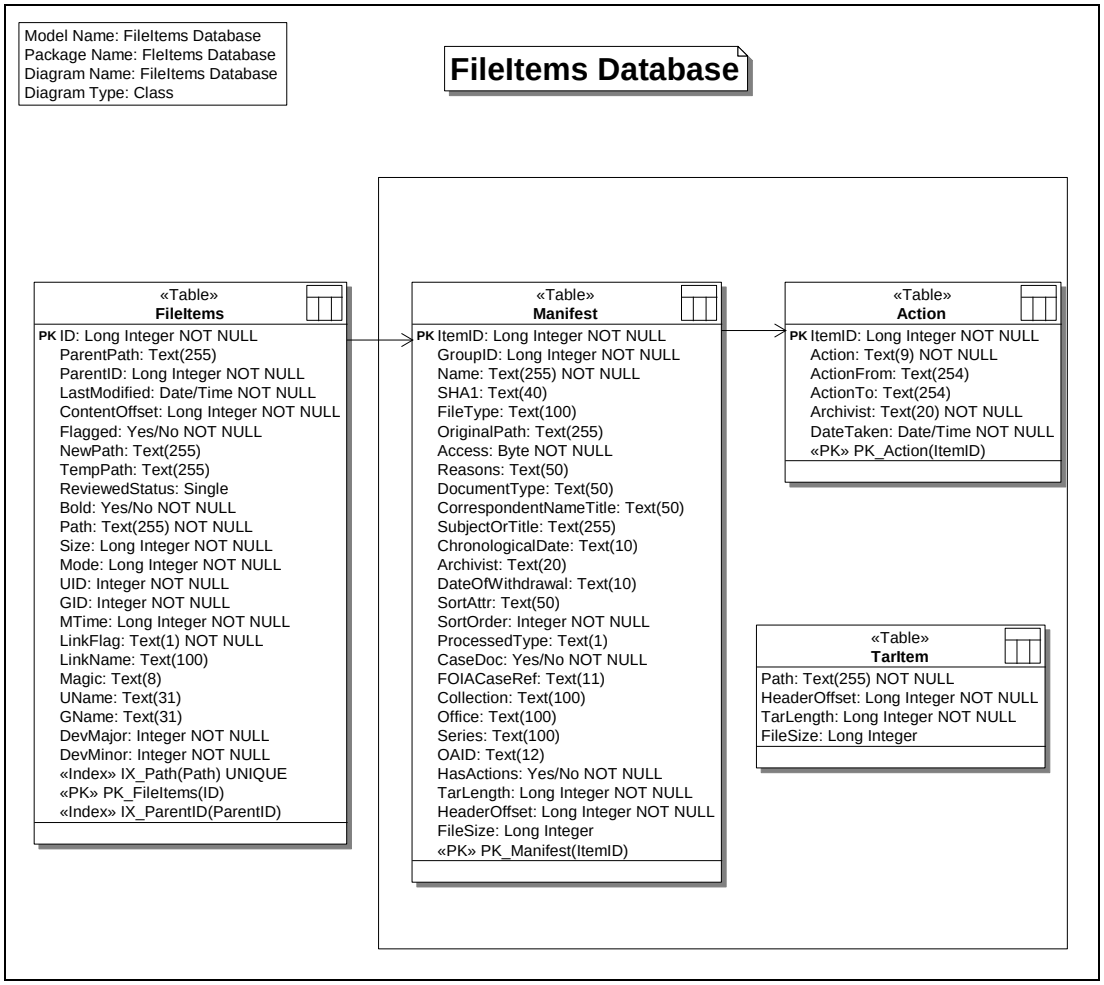

Figure 15. The File Items Data Model

Each FileItems object contains a Manifest object and each Manifest object contains a ManifestBODT. The three tables in the box shown in Figure 15 and an additional 28 queries are added to the FileItems database by the ManifestBODT. Manifest and Action tables are obtained from the manifest in a tar file or are built from the contents of the FileItems table.

3.3 FOIA Search Oracle Database

The FOIA Search Database contains the tables and stored procedures necessary to index containers and perform FOIA searches. The FOIA Search database model is shown in Figure 16.

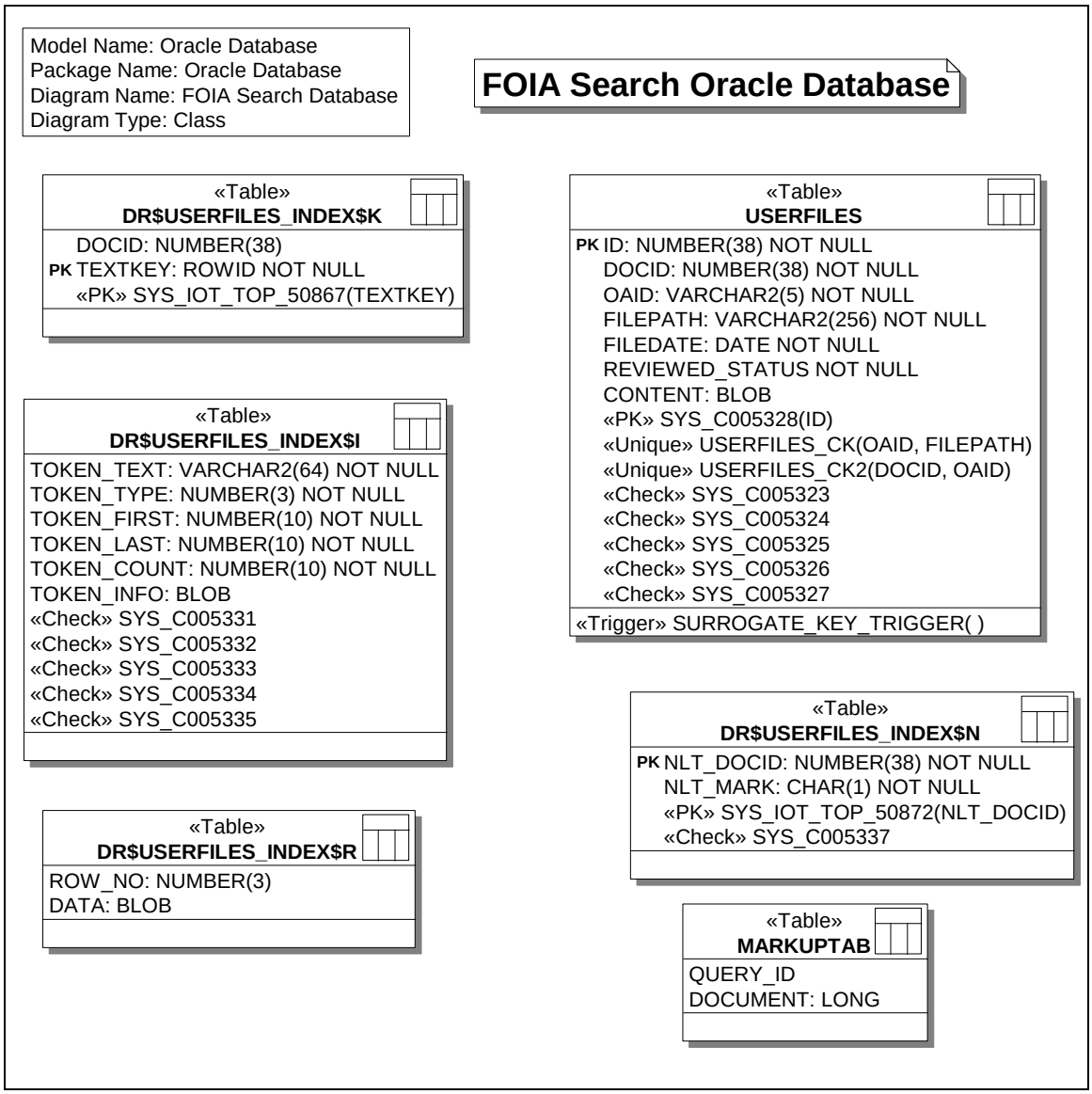

Figure 16. The Oracle Data Model for FOIA Search

The FOIA Search database tables are created using an Oracle script file (ArchivalToolsOracleConfiguration.sql). This script file creates the main table (USERFILES). It also creates a "Text Index" on the Content field. This field is a BLOB type field (a field that can hold up to 4GB of data). The Content field actually holds the full content of a single file.

The "Text Index" is a collection of tables and indexes that store information about the text in the stored field that is indexed. All the tables with the prefix DR$USERFILS_INDEX$ are part of the Text Index. The index is created using the INSO filter. The INSO filter is a set of programs that convert files in proprietary formats to Text or HTML file formats.

4. An Exercise in Migration from VB6 to Visual Basic 2005

The PERPOS prototype system was developed using Visual Basic 6 (VB6) and the Visual Studio development environment. Visual Basic 6 is a language and Visual Studio is an environment that supported rapid application development.

The primary issue in upgrading from Visual Basic 6 to Visual Basic 2005 is that the Visual Basic Net broke backward compatibility. Visual Basic 2005 is built on Visual Basic Net and the Net Framework. This means that instead of upgrading like was done from VB4 to VB6, that migration is necessary. Visual Basic was not upgraded to Net. It was rewritten. There are books written about migrating VB6 to NET [2]. Microsoft provides extensive documentation covering changes in syntax, debugging applications, and code samples [3].

There are three steps to the migration from VB6 to VB8. First, use a tool like the Microsoft Code Advisor for Visual Basic 6 [4] that gives advice on migration to Visual Basic 2005. Second, the Microsoft Upgrade Wizard [5] included in VB2005 is run to translate as much code to VB2005 as possible. Third, the code is refactored (or cleaned up) manually or using a refactoring tool.

This section describes some of the issues encountered in migrating the ManifestLibrary project from VB6 to VB2005. The ManifestLibrary dll consist of 9 components, 5 classes and 4 module files.

4.1 Code Advisor for VB6

The Code Advisor for Visual Basic 6 identifies 28 issues that the Upgrade Wizard has problems with in translating VB6 code to VB2005. The first step was to run the Code Advisor for VB6 on the chosen dll. The chosen dll is the ManifestLibrary.dll. Figure 17 summarizes the issues the Code Advisor identified for the ManifestLibrary DLL.

Issue	Description	Occurrences
Late Binding of Variant or Object	Variables, parameters, and return values typed as Variant or Object can cause problems when upgrading.	20
Variant-Returning String Function	Variant-returning string functions are not supported in Visual Basic .NET. Use the String-returning version of the function, which has a '$' suffix.	14
Non Zero Lowerbound Arrays Not Supported	Visual Basic .NET does not support the use of arrays that have a lower-bound index other than zero.	10
As Any Not Supported	API Declare statements that include parameters typed using 'As Any' will not be upgraded.	4
'#If' blocks are not reliably upgraded	When a #If condition evaluates to False, the #If...#End If block is not upgraded. The Upgrade Wizard does not reliably evaluate whether #If conditions are True or False.	40

Figure 17. Issues Identified by the Code Advisor in Translating the ManifestLibrary to V52005

"Late Binding of Variant or Object" issues need to be deferred until after the Upgrade Wizard has run. This issue falls into two main categories. One category is code that can be replaced with overloaded methods. These are methods with the same name but different arguments or return values. The other code has to do with the way nulls are handled in database return values.

The "Variant-Returning String Function" is one of the easiest to fix. The Code Advisor places 'FIXIT:' comments in code. It also places a button on the tool bar that allows the user to go to the next FIXIT comment.

One of the hardest issues to fix is the "Non Zero Lowerbound Arrays Not Supported". Modification of the code that calculates indexes as well as the "Dim" and Redim" statements that define arrays, is required to fix this issues.

The "As Any Not Supported" is caused by two Windows APIs. The "CopyMemory" and the "ZeroMemory" APIs are replaced once the code has gone through the Upgrade Wizard. These APIs can be replaced with code found on the Internet [6].

The "'#If' blocks are not reliably upgraded" is easy to fix, but time consuming. These blocks are debugging blocks and one debugging file that need to be removed. These debugging blocks are in most if not all class files. These blocks were added to a Blank Class template that was used for most classes.

The Code Advisor was also run on the Archival Processing Tool, Archival Assistant Tool, Archival Repository, Archival Repository Tool, FOIA Search Application, and the Storage Management Assistant projects. The results are summarized in Figure 18.

Summary of the Code Advisor Project Reports

VB Projects	Components	Issues Count
ArchivalProcessingTool (APT)	53	430
Archival Assistant Tool (AATVB)	74	888
ArchivalRepository	18	165
ArchivalRepositoryTool (ART)	45	408
FOIASearch	5	16
ManifestLibrary	9	88
StorageMngmntAssist	7	52
Totals	211	2047

Issues	StorageMngmntAssist	ManifestLibrary	FOIASearch	ART	ArchivalRepository	AATVB	APT	Issue Totals
Late Binding of Variant or Object	16	20	1	147	31	221	78	514
Missing Option Explicit	0	0	0	0	0	0	0	0
Soft Binding of Form or Control	0	0	0	0	6	0	54	60
Soft Binding using ActiveForm and ActiveControl	0	0	0	0	0	0	13	13
Variant-Returning String Function	5	14	1	12	18	29	22	101
LSet Not Supported for User-Defined Types	0	0	0	0	0	0	0	0
OLE Control Not Upgraded	0	0	0	0	0	0	0	0
No Line Control in Visual Basic .NET	0	0	0	0	0	0	3	3
No Shape Control in Visual Basic .NET	0	0	0	0	0	0	0	0
UpDown Control Not Upgraded	0	0	1	0	0	0	1	2
Property/Method/Events Not Upgraded	4	0	3	111	0	0	115	233
Non Zero Lowerbound Arrays Not Supported	0	10	0	0	11	95	12	128
Incorrect Use Of Enumeration	0	0	2	12	0	1	23	38
As Any Not Supported	3	4	0	9	3	7	15	41
Changing <property> Not Supported	0	0	0	3	0	0	0	3
Property Page Not Upgraded	0	0	0	0	0	0	0	0
Designer Not Upgraded	0	0	0	0	0	0	0	0
Missing/Corrupt Reference or Component	0	0	0	0	0	0	0	0
Non-TrueType Font Not Supported	0	0	0	0	0	0	1	1
Keyword Not Supported	0	0	0	0	0	3	2	5
Return Has New Meaning	0	0	0	0	0	0	0	0
Option Base 1 is not supported	0	0	0	0	0	0	0	0
On ... GoTo is not supported	0	0	0	0	0	0	0	0
DAO Data Binding	0	0	0	0	0	0	0	0
RDO Data Binding	0	0	0	0	0	0	0	0
'#If' blocks are not reliably upgraded	24	40	8	112	96	532	88	900
Printer Object and Printers Collection Not Upgraded	0	0	0	2	0	0	3	5
Single Threaded Controls	0	0	0	0	0	0	0	0
Totals	52	88	16	408	165	888	430	2047

Figure 18. A Summary of the Reports Created by the Code Advisor

4.2 Upgrade Wizard for VB

The Upgrade Wizard lists additional problems. The Upgrade Wizard lists issues in two major categories—"Compile Errors" and "Warnings." The deferred changes to the "CopyMemory" and "ZeroMemory" API calls discussed above result in six compile errors. There is .NET code available on the Internet that is equivalent to Copy Memory that can be used to fix these errors [6]. The warnings refer to changes in the way methods behave. For instance, he "Get" and "Put" methods are replaced with FileGet and FilePut. These do not behave exactly as the methods they replace.

In VB2005, the way that structures are passed to API calls has changed. Marshalling was done behind the scenes in VB6. In VB2005 explicit marshalling instructions are need to tell the system how to pass the structures. The Upgrade Wizard gives the marshalling instructions where fixed-length strings were used. Since fixed-length strings are not allowed, the variable is defined as a character array with the length specified in the marshalling instructions.

4.3 Testing

The "Compile Errors" and the "Warnings" identified by the Upgrade Wizard fixed until there are no compile errors or warnings. Then a test application that can make COM calls to the dll is created to test that the behavior of the compiled VB2005 code has not changed.

It was found during testing that methods that work with numeric variables have changed behaviors. The Upgrade Wizard changed all "int" variables to "Short" and all "long" variables to "Integer". In VB2005 there are unsigned numbers. This means that if numbers are changed to Octal string, the unsigned version of the number must be used. Where "int" was used in working with tar files. "UShort" must be used. Where "long" was used, "UInteger" must be used. Where octal numbers were converted to decimal, they caused numeric overflow errors. When negative numbers were converted to octal numbers, the resulting values were incorrect.

The functional test of the migrated ManifestLibrary component used a VB6 test project to reference the compiled VB 2005 version of the ManifestLibrary component. The VB 2005 version of the ManifestLibrary was able to create a manifest object, open a manifest file, and display information from the manifest file.

4.4 Refactoring

Code Refactoring is the process of changing a computer program's internal structure without modifying its functional behavior. This is done to improve the readability of the code, to simplify code, to improve performance, to improve maintainability, or to adhere

to a new programming language paradigm. Refactoring is sometimes informally referred to as "code cleanup."

Some refactoring was performed in the earlier steps. For example, Code Advisor was used to assist in removing dead code. Dead code includes declarations for unused variables and methods that are never called. As a second example, the CheckSum is calculated to create a tarheader and for validating a tarheader. This code was pulled out into its own method that returned a 0 for an invalid header or a short greater than zero for a valid header.

The Upgrade Wizard uses the "Microsoft.VisualBasic.Compatibility" and the "Microsoft.VisualBasic.Compatibility.Data" libraries to mimic some of the VB6 behavior. Some of this code can be replaced with Net equivalents. This is done to utilize the full capacity of the Net framework.

The uses of the FileGet and the FilePut should eventually be replaced with the use of Stream and Buffer objects. The objects using the "Microsoft.Scripting.Runtime" can also be replaced with Net objects.

Refactor! for Visual Basic 2005 [7] enables Visual Basic developers to simplify and re-structure source code inside of Visual Studio 2005, making it easier to read and less costly to maintain. *Visual Basic Upgrade Companion* is a similar refactoring tool [8]

5. An Exercise in Migration from Visual Basic 6 to JAVA

This section describes some of the issued encountered in migrating the FOIASearch project from VB6 to Java using VB Converter [9]. The FOIASearch project has 3 forms, 1 class, and 1 module. It and the Archival Repository Tool connect to an Oracle database.

5.1 VB Converter

The VB Converter is installed as an Add-in to Visual Basic. This Add-in shows up as a tool bar with 5 buttons. The first button is the Make button. This button performs the actual conversion of VB6 to Java. It displays a progress bar during compilation. Then it displays a report showing the number of files successfully compiled and any syntax errors that occurred (See Fig. 19). It creates Java class files that will run as stand-alone applications. The project is a component that can be called from any Java-Enabled source. In this case, the component is run inside a Java-Enabled Browser.

The second button is the Run button. It will run a successfully compiled project in the Java Virtual Machine (JVM). This shows how the project will look and work as a stand-alone application.

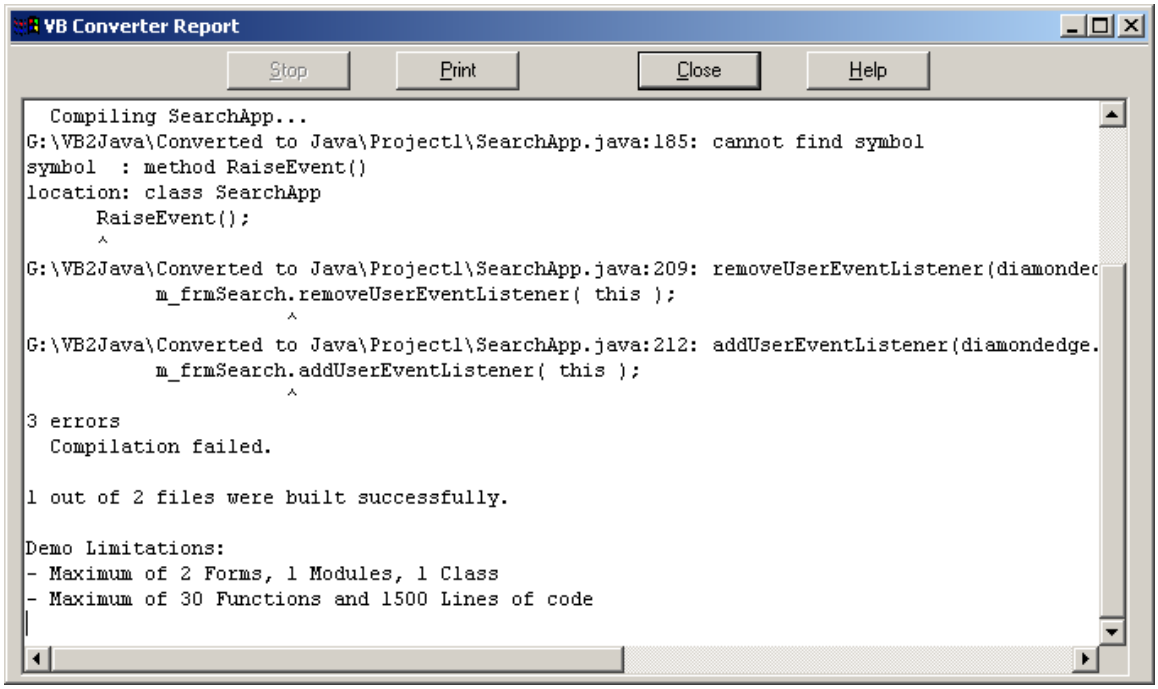

Figure 19. A VB Converter Compiler Report

The third button is the VB Converter Options button. This lets the user customize some of the conversion. Figure 20 shows the VB Converter Options Dialog Box. Under the general tab the user can determine the kind of Java GUI control to be used. In the Database tab, the user can indicate which JDBC drivers to use in place of the ones in the VB code. Windows database connections are made through ODBC drivers or OLEDB drivers. In Java, database connections are made through JDBC drivers. The Environment tab is used to indicate were to find the Java runtime, the Java compiler, and where to output the converted files.

The third button on the toolbar is the VB Source Analyzer. Figure 21 shows a VB Source Analyzer Report. It counts the lines in the forms and the code of the currently loaded project. It tells of possibly unsupported controls and special controls that are used. It then calculates the number of licenses necessary to convert the project.

The fifth button is the Help button. Selecting this button displays the table of contents for VB Converter's documentation. Part of this documentation is a list of supported or partially supported features. Examples of unsupported functionality are calls to the Windows API. The only supported Windows API call is *to sleep*. Though the App Object is partially supported, the StartMode is a property. This property is often use in ActiveX exes. If the StartMode is not VbSModeAutomation, the project starts as a stand-alone application. Otherwise, the project starts as an ActiveX component.

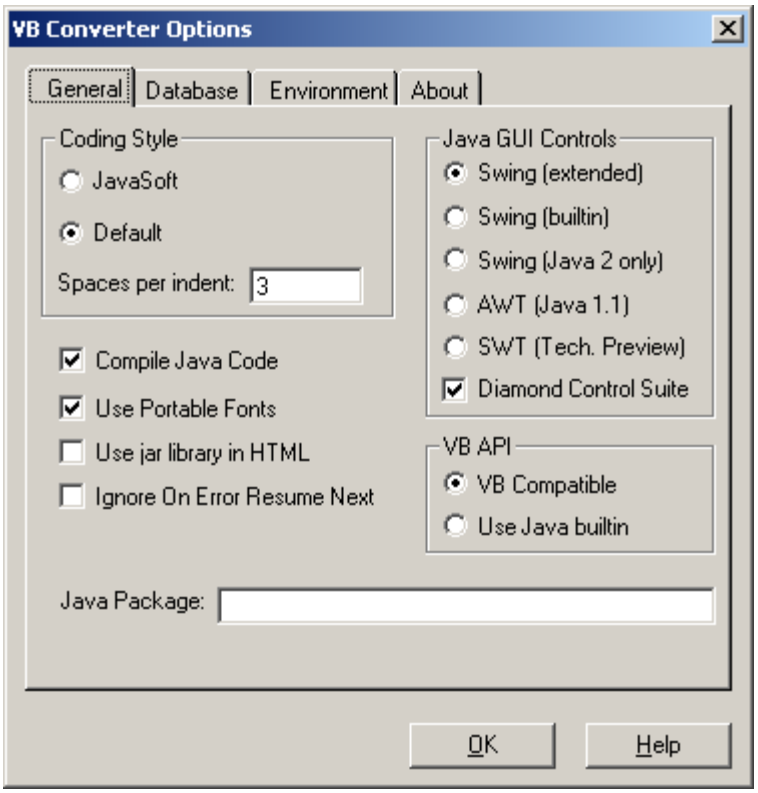

Figure 20. VB Converter Options Dialog Box

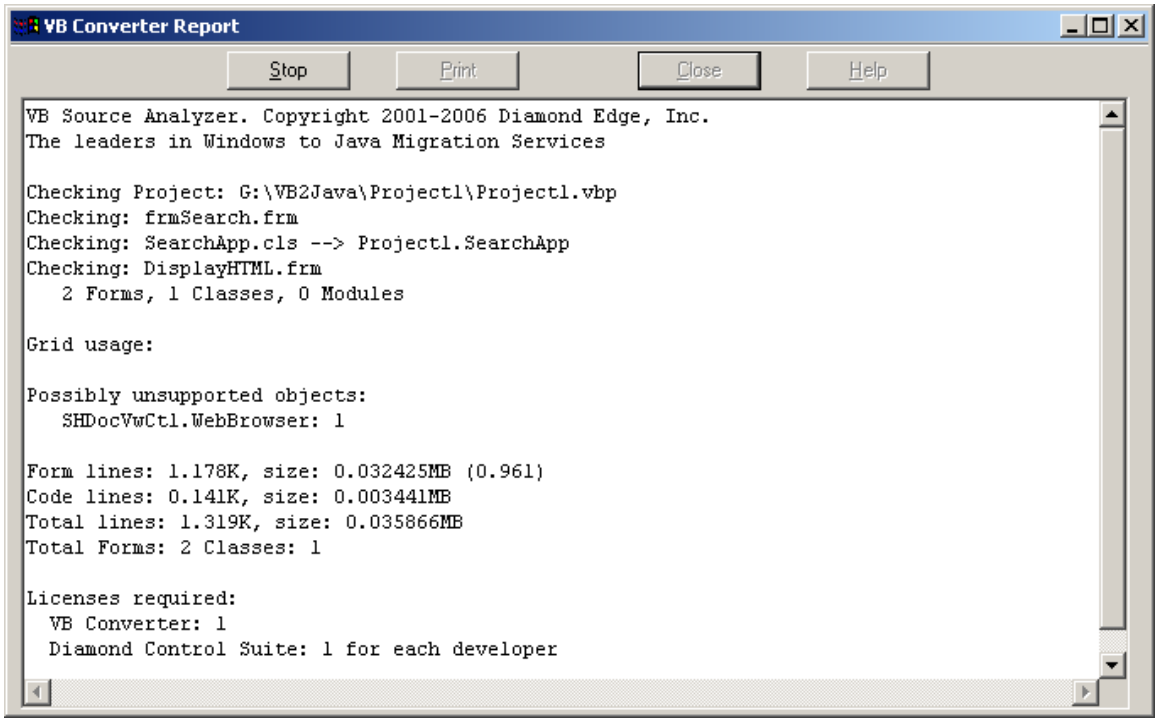

Figure 21. A VB Source Analyzer Report

5.2 Testing

When the VB Converter has converted VB6 code into Java and class files, one should review the VB Converter Compiler report. This report and a log file that is created for the project indicate where modifications to the VB code are needed. In the FOIASearch program there were several places with *with* blocks. VB converter only partially supports the conversion of *with* blocks. There were several cases where there was an error reported on lines of code that started with the *with* (.) operator. In those cases, it was necessary to remove the *with* block and to explicitly declare the entire expression.

It is possible to add Java code to the VB project by starting the line with *'java* followed by a Java statement. The VB compiler ignores this code since it appears to be a comment. VB Converter removes the leading *'java* from the line and leaves the Java code there. It is also possible to mark a line of code as VB only. This is done by placing the comment *'vb_only* above the line and *'end_vb_only* below the line. Again, these statements appear as comments to the VB compiler but everything between the two comments is ignored by VB Converter. This way it is possible to modify the code so it still works in VB but is more easily converted by the VB Converter,

When there are no more error messages in either the conversion step or the compiling step, the Java project is imported into a Java IDE. This allows the code to be debugged. At this point, it is necessary to apply breakpoints and step though the code to see what it is really doing. It the case of this project, it was found that one of the VB classes called *Application* was not behaving correctly because there is a class in Java that is called *Application*. Changing the name of the VB class to *SearchApp* solved this and several other several problems.

Another case of naming conflict occurred when in VB a method was named *GetResults* and a property was named Results. In VB, the method that is called to get the property *Results* is named *Results*. The method starts with a "Public Property Get Results() As Recordset" statement. In Java, this method was converted into "public Recordset getResults()". In VB, where there was a call to *GetResults*, the VB Converter replaced it with "getResults()". This meant that the *GetResults* method was never called in the Java code.

Another place where there were changes in behavior is where the Active Data Objects (ADO) code was replaced with Java code. The Java code is built on Java Database Connectivity (JDBC). The JDBC is based on Microsoft's Open Database Connectivity (ODBC). The VB Converter documentation indicates that, "The JDBC data source and database names you assign to an applet must exactly match the ODBC data source and database names used to test the Visual Basic form prior to converting it to a Java applet." Everywhere database connection was made, JDBC Drivers and Data Sources had to be defined.

The functional test of the FOIASearch component consisted of a VB6 project to reference the compiled Java FOIASearch component. The Java version of FOIASearch displayed a

user interface that matched the VB6 FOIASearch interface. The JDBC interface to the Oracle and Microsoft Access databases required changing the access method from Active Data Objects (ADO) to ODBC. Difficulties were encountered in achieving this interface.

5.3 Refactoring

Some refactoring of the Java code was done during testing and debugging. More refactoring was needed to take advantage of features that are available in Java that are not available in VB6. For instance, refactoring is needed to replace all Windows API calls.

FOIASearch in VB6 is an ActiveX exe. It can operate either standalone or as a component. VB Converter does not generate Java code from an ActiveX exe that can both standalone and be run as a component. In Java, if a class starts up as a component, the *init* method is used. If the class starts as a stand-alone program, it has a *main* method. A Java class can have both methods, but during refactoring it is necessary to add them to Java code created by VB Converter.

6. Summary

PERPOS manifests about six years of experience in collaborative, evolutionary development of tools to support archivists in processing Presidential electronic records. It has proven an excellent environment for experimental evaluation of advanced technologies in support of archival description, preservation, and review of electronic records. Given the obsolescence of its development environment, Visual Studio for Visual Basic 6, the primary issue is how to maintain the PERPOS software so that it can continue to be an environment for experimentation.

The VB6 code should be refactored into a more object-oriented design before attempting to migrate to Visual Basic 2005 or to Java. In particular, code should be pulled out of complex projects to create new simpler projects. For example, the image redaction user interface in the Archival Processing Tool could be moved into a separate project. This is a candidate for refactoring because, starting with Windows XP, the three image controls that are used in image redaction are no longer being shipped as part of the operating system. The part of the Archival Assistant Tool that deals with FOIA exemptions and PRA restrictions could be removed and placed in its own dll so that both the APT and the new Image Redactor could reference them.

Visual Basic 6 only supports interface inheritance. Moving to an object-oriented programming language such as VB 2005 or Java will enable us to achieve true class inheritance, that is, both interface and implementation inheritance.

The PERPOS data models are implemented in Oracle, Microsoft Access and a text data file. The persistent databases, such as the Archival Repository Tool database, should be migrated to Oracle for consistency and scalability.

The two exercises in migrating PERPOS components to VB 2005 and Java resulted in components that were functionally the same as the components written in VB6. The migration tools were judged useful, though substantial manual recoding was necessary.

The Common Object Model (COM) is an interface standard for software components introduced by Microsoft. There is interoperability between Visual Basic 6, Visual Basic 2005 and Java through COM objects. This makes it possible to migrate from VB6 incrementally, rather than all at once.

The advanced technologies that have been, or are being, developed for use with PERPOS include tools for information extraction, document type recognition, metadata extraction, speech act recognition, automatic description, and checking for access restrictions, They are implemented as components that interface to the functional components of PERPOS and utilize the e-record resources of the PERPOS record repository. Since all of our research prototypes are currently being implemented in Java, we are inclined to migrate the VB6 code for PERPOS to Java.

References

1. W. Underwood, S. Laib and M. Hayslett. Reference Manual for PERPOS: An Electronic Records Repository and Archival Processing System, Version 3.1. PERPOS TR ITTL/CSITD 06-2, ITTL, Georgia Tech Research Institute, September 2006.

2. D. Appelman. Moving to VB.NET: Strategies, Concepts and Code. Springer-Verlag, 2001.

3. Microsoft Visual Basic 6.0 Migration Resource Center. http://msdn.microsoft.com/en-us/vbrun/ms788233.aspx

4. Microsoft. Code Advisor for Visual Basic 6 http://msdn.microsoft.com/en-us/vbasic/ms789135.aspx

5. John Hart. What's New with the Visual Basic Upgrade Wizard in Visual Basic 2005. http://msdn.microsoft.com/en-us/library/ms379614(VS.80).aspx

6. Adnan Samuel. Equivalent of CopyMemory in .NET. www.codeproject.com/KB/vb/CopyMemory_in_Net.aspx

7. Developer Express, Inc. *Refactor! for Visual Basic 2005*. http://msdn2.microsoft.com/en-us/vbasic/ms789083.aspx

8. ArtinSoft. Visual Basic Upgrade Companion (VBUC). http://www.artinsoft.com/pr_vbcompanion.aspx

9. Diamond Edge, VB Converter, Java Edition. http://www.diamondedge.com/products/Convert-VB-to-Java.html