ACE Student Tracker

Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Chao Ma, B. Eng.

Graduate Program in Computer Science and Engineering

The Ohio State University

2011

Master's Examination Committee:

Furrukh Khan, Advisor

Rajiv Ramanath
Abstract

ACE Student Tracker is a system that tracks Applying, Current and Employed graduate students in the Department of Electrical and Computer Engineering at The Ohio State University. In this thesis, we first overview its functionality and then focus on the technical aspects: the architecture, the use of Sharepoint to integrate authentication, authorization and user interfaces of an ASP.NET application and SQL Server Reporting Services (SSRS) reports, and the use of LINQ to SQL as the main data access technology. Various security considerations, design choices, development practices and lessons learned are presented.
Acknowledgments

First, I would like to thank Dr. Furrukh Khan and Dr. Rajiv Ramnath for advising me through my graduate studies and giving me the opportunity to work on this project. Their guidance will keep benefiting me in my career.

Second, I would like to thank Dr. Robert Lee for sponsoring and supporting the project.

Third, I would like to thank all colleagues for the cooperation and assistance in the project.

Finally, I would like to thank my family for always backing me with their love.
Vita

June 2007.............................................Bachelor of Engineering

Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, China

April 2008 -- Present..............................Graduate Administrative Associate

Department of Electrical and Computer Engineering
The Ohio State University
Columbus, Ohio

Fields of Study

Major Field: Computer Science and Engineering
Table of Contents

Abstract ................................................................................................................................. ii
Acknowledgments .................................................................................................................. iii
Vita ........................................................................................................................................ iv
List of Figures ....................................................................................................................... vii
List of Listings ...................................................................................................................... viii
Chapter 1: Overview .......................................................................................................... 1
  1.1 Functional Overview ................................................................................................... 1
  1.2 Technical Overview .................................................................................................... 2
  1.3 Thesis Organization ................................................................................................... 5
Chapter 2: Developing an ASP.NET Application in Sharepoint ....................................... 7
  2.1 Introduction ................................................................................................................ 7
  2.2 Sharepoint Security .................................................................................................. 7
  2.3 Development Environment and Practices ................................................................ 11
  2.4 ACE Web App Overview ........................................................................................... 13
Chapter 3: Accessing Data with LINQ to SQL ................................................................ 18
  3.1 Introduction ................................................................................................................ 18
  3.2 Object Relational Mapping ....................................................................................... 18
  3.3 Queries ....................................................................................................................... 20
  3.4 Inserts, Updates and Deletes ................................................................................... 25
Chapter 4: ACE Batch Import .......................................................................................... 33
  4.1 Operation ................................................................................................................... 33
4.2 Security ........................................................................................................................................35
4.3 Design .........................................................................................................................................36
Chapter 5: Development Lessons Learned .........................................................................................42
  5.1 Gathering Requirements and Feedback .......................................................................................42
  5.2 Protecting Data ..........................................................................................................................43
  5.3 Migrating a System to a Production Environment .......................................................................44
Chapter 6: Conclusion and Future Work .............................................................................................45
  6.1 Conclusion ..................................................................................................................................45
  6.2 Future Work ...............................................................................................................................45
Bibliography .........................................................................................................................................47
List of Figures

Figure 1.1: ACE technical architecture. .................................................................4
Figure 2.1: Database tables used in database row based authorization ................................16
Figure 4.1: Class diagram of input file parsing step 1 .................................................37
Figure 4.2: State diagram of input file parsing step 1 .................................................37
Figure 4.3: Class diagram of input file parsing step 2 .................................................39
List of Listings

Listing 3.1: A sample LINQ to SQL query ..................................................................................21
Listing 3.2: A sample disconnected LINQ to SQL update ..........................................................26
Listing 3.3: A sample that shows the rules of using method Attach ........................................28
Listing 3.4: A sample disconnected LINQ to SQL delete ..........................................................30
Listing 3.5: A sample disconnected LINQ to SQL delete with concurrency conflict detection fixed........31
Chapter 1: Overview

1.1 Functional Overview

ACE Student Tracker (ACE) is a system that tracks Applying, Current and Employed graduate students in the Department of Electrical and Computer Engineering (ECE) at The Ohio State University (OSU).

Many people have questioned the need for ACE after they learned the above elevator pitch. They thought existing university systems at OSU such as Student Information System (SIS) have already fulfilled the exact same purposes. But the truth is university systems are not enough:

1. For PHD admissions,
   1) Each applicant’s application materials are scattered over several university systems and mails.
   2) ECE’s desired admission process is not incorporated by university systems.

2. For enrolled graduate student academic records,
   1) Some ECE data are not tracked by university systems.
   2) Some data definitions in ECE are different than those in university systems.

3. For graduate associate employment records,
   1) The raw history data stored in university systems are not usable to faculty.
   2) ECE staff needs a new internal appointment request process.

4. Not all university systems are accessible by relevant faculty and staff.

Therefore, ACE has been developed to complement the functionality of existing university system rather than duplicate them:
1. The admissions part of ACE serves as the department PHD admissions system where the staff stores combined electronic applications and tracks their completeness status; The Admission Committee and staff cooperate to make admission decisions; The Fellowship Committee and the faculty search and review potential candidates.

2. The academic records part of ACE tracks information regarding the academic records of enrolled student in the graduate programs such as degree level, adviser, various milestone examinations, matriculation time, graduation time, etc.

3. The employment records part of ACE tracks graduate associate employment history records processed by ECE staff from the raw data of university systems. Faculty also uses this part of the system to make appointment requests to staff to employ graduate associates.

1.2 Technical Overview

There are 3 technical subsystems in ACE -- ACE Web App, ACE Reporting and ACE Batch Import, which share the same database ACE DB:

1. ACE Web App is the main subsystem that users interact with. The interface is in the form of Dynamic HTML pages generated by ASP.NET, through which users can perform create, read, update and delete (CRUD) operations on ACE DB.

2. ACE Reporting consists of a set of reports executed by SSRS. It also presents a web user interface, through which ACE DB data are shown in a printer-friendly fashion and can be retrieved in formats other than HTML such as PDF, Word, Excel, etc.

3. ACE Batch Import is a command line application written with C# and routinely run by administrators to ease the manual data entry burden of users. It verifies and inserts records into ACE DB based on files from SIS. The resulting log files serve as the basis for following data entries and corrections.
Although ACE Web App and ACE Reporting are powered by different web technologies, we want to integrate them as much as possible so that they provide coherent experiences to users and administrators. If we had decided to develop ACE Web App as a traditional ASP.NET web application and use the default SSRS web application -- Report Manager to manage and serve reports, we would have difficulties achieving the goal at least in terms of security:

1. Even though both applications can be configured to use the same authentication method -- Windows Authentication, the authentication steps are separate, which means users are required to re-enter the same credentials when they navigate from one to the other;
2. Each application needs its own authorization rules configured while conceptually the roles and related permissions are almost identical. This creates administration burdens which may lead to unnecessary errors;

To avoid these problems, we decided to integrate both subsystems into Sharepoint by running SSRS in Sharepoint integrated mode and developing ACE Web App inside Sharepoint. This way, a Sharepoint web application functions as the host of both and provides a shared security infrastructure, as follows:

1. For authentication, navigating between ACE Web App and ACE Reporting is within the boundary of the same Sharepoint web application and so no additional login is required;
2. For authorization, reports of ACE Reporting and pages of ACE Web App are treated identically as items of ACE Sharepoint document library and so only 1 set of Sharepoint authorization rules is needed for access control.

Additionally, Sharepoint provides a consistent and professional look-and-feel for both subsystems.

Figure 1.1 illustrates the technical architecture of ACE.
ACE is deployed onto 2 separated servers -- a database server and an application server. The database server is a dedicated database server that hosts ACE DB and system databases of Sharepoint and SSRS. The application server, on the other hand, hosts all the subsystems built on top of ACE DB. This separation results in an architecture that better separates the duties of database administrator and application administrator [1]. Pages of ACE Web App and report of ACE Reporting are deployed into a Sharepoint document library. The content of the document library is stored in the Sharepoint content database. When a user requests a report, Sharepoint asks SSRS to retrieve the report from the content database, executes and renders it. Sharepoint then returns the rendered report embedded in its own user interface back to the user.
For page requests, Sharepoint retrieves pages from the content database and execute them itself. The pages reference ACE Web App DLL which contains code-behind classes of pages [2] and ACE Web App Entity Classes. ACE Web App Entity Classes mainly serve as the data access layer of ACE Web App. The data access layer relies on LINQ to SQL as the primary data access technology. The main advantage of LINQ to SQL over traditional ADO.NET is the productivity boost -- it greatly reduces the amount of time writing and debugging data access code. However, using LINQ to SQL requires understandings of many advanced .NET platform features. So, in the case of ACE Batch Import, we still use traditional ADO.NET since its future maintainers are not expected to learn LINQ to SQL.

1.3 Thesis Organization

Chapter 2 introduces Sharepoint, presents its security settings, describes the development environment and practices for Sharepoint pages and overviews ACE Web App -- the ASP.NET application developed inside Sharepoint.

Chapter 3 introduces LINQ to SQL -- the data access technology used by ACE Web App, presents its object relational mapping and explains how to perform queries, inserts, updates and deletes with it.

Chapter 4 describes the operation, security and design of ACE Batch Import.

Chapter 5 summarizes the development lessons we have learned in developing ACE with regard to gathering requirements and feedback, protecting data and migrating a system to a production environment.

Chapter 6 concludes the thesis and presents the future work.
We do not have a chapter for SSRS -- the underlying technology of ACE Reporting, since it is already documented thoroughly by others. Developers can follow [3] to configure SSRS for Sharepoint integration and [4] to develop SSRS reports.
Chapter 2: Developing an ASP.NET Application in Sharepoint

2.1 Introduction

Sharepoint is a web-based enterprise content management system developed by Microsoft. It is also a development platform built on top of ASP.NET that developers can extend and build applications around. SSRS is Microsoft’s web-based reporting product, which can be installed in Sharepoint integrated mode so that it presents reports through Sharepoint, utilizing its security model and user interface. As a result, we can unify the authentication, authorization and user interfaces of a regular ASP.NET application and SSRS by using a Sharepoint web application to host both of them.

Developers can follow [3] to configure SSRS for Sharepoint integration and [4] to develop SSRS reports. In this chapter, we present how we have developed an ASP.NET application in Sharepoint.

2.2 Sharepoint Security

Taking advantage of the virtual path provider model [5] in ASP.NET that abstracts away the locations of page source files, Sharepoint can retrieve pages from its content database and then further process them [6]. As a result, developers and power users can conveniently upload their ASP.NET pages to Sharepoint document libraries and have them accessible through Sharepoint. Convenience often comes with potential security risks. To prevent malicious users from attacking an entire Sharepoint web farm from their uploaded pages and/or DLLs invoked thereby, Sharepoint processes pages in Safe Mode [7] and uses Code Access Security (CAS) [8] to govern the behaviors of DLLs. The rest of the section shows how we have configured Sharepoint security settings.
When developing a data-driven web application like ACE, ASP.NET data binding expressions and data source controls can drastically reduce the code that needs to be written. However, Sharepoint disallows both of them by default. For data binding expressions, it is because they are considered server side scripts which are prohibited in general to prevent malicious code. For data source controls, they are considered unsafe controls since it is possible to set their properties in certain ways to directly access databases or invoke DLLs to achieve that indirectly.

[7] shows how to modify the SafeMode section in web.config of the hosting Sharepoint application to allow server side scripts and to allow unsafe controls for items in certain folders. But those settings should be used with caution because of the above reasons. In the case of ACE, since we are only using Sharepoint as a host application of a regular ASP.NET application and SSRS, rather than as a large scale enterprise content management system, we have concluded that our security policy can allow side scripts and unsafe controls as long as there is an alternative security measure to compensate for them. The compensating security measure is to define a custom permission level “Read without Source”\(^1\) in Sharepoint that includes only read permissions to the end results of uploaded items but not their source files. Thus, only the resulting HTML pages can be read but not the source files of ASP.NET pages, definitions of reports and report data sources. Users are only given “Read without Source” permission level to the document library in which server side scripts and unsafe controls are allowed. Only the server farm administrator has write permissions and read permission to source files to the document library.

In .NET, all components are governed by CAS. When a component tries to perform certain operation, .NET checks the permissions granted to the component in its CAS policy and decide whether the operation by the component is allowed. Developers do not need to worry about CAS when they developing client side applications or regular ASP.NET applications because those types of applications are given full trust by the

---

\(^1\) “Read without Source” permission level should only include the following permissions: “View Items” under “List Permissions”, “View Pages” and “Open” under “Site Permissions”. 8
default CAS policy of .NET. Sharepoint is designed as an enterprise content management system which can be extended by third party components. In order to guard against potential malicious components, it by default executes all components that are deployed into the local bin folder in the minimal trust level, which has a very limited set of permissions. For example, invoking ADO.NET based data access libraries are disallowed in the minimal trust level.

One way to go around CAS is to deploy DLLs in the Global Assembly Cache (GAC), which is a fully trusted location by Sharepoint. However, DLLs deployed in GAC are accessible to all applications in the same machine and so have larger attack surfaces. Instead, we still deploy ACE Web App DLL to the local bin folder and instruct Sharepoint to fully trust ACE Web App DLL by defining a custom CAS security policy and reference it from web.config, following the guide in [9] [10] [11] [12].

Sharepoint by default has ASP.NET impersonation [13] enabled. Together with the default Integrated Windows Authentication, it means Sharepoint uses the current user’s Windows identity to access external resources. SQL Server by default also uses Integrated Windows Authentication. This combination of default settings places extra burdens on administrators’ shoulders because additional Active Directory (AD) configurations are needed for allowing delegation [14] [15] [16] and an additional set of authorization rules is needed to control access to ACE database for all users’ Windows accounts, which we would like to avoid.

We do not want to simply disable impersonation since it is an application wide setting and we are not sure whether other parts of the hosting Sharepoint web application depend on it. Besides, impersonation is generally a good security practice since users’ identities are less privileged than the Sharepoint application pool identity which will be used to access external resources if impersonation is disabled. As a result, we have decided to use a SQL Server Authentication login for all user access in ACE database, programmatically impersonate the Sharepoint application pool identity locally only to decipher the encrypted connection string stored in web.config and IPSec to protect the communications. An alternative
approach would be to still use a Windows login for all user access in ACE database and programmatically impersonate that account to access the database following [13]. The disadvantage of that approach is that its programming model and configurations are more complex. [17] shows how to create Windows account logins and user mappings. The steps are almost identical to those of SQL Server Authentication logins except for the authentication types. Since SQL Server Authentication transmits credentials in clear text, IPSec is used to encrypt communications between the application server and the database server. We have also configured Windows Firewall to take advantage IPSec’s authentication to allow inbound packets to SQL Server from only trusted computers. [18] and [19] show how to configure IPSec on Windows servers. It is recommended that the configurations are done via Group Policy so that they can be easily applied to other domain servers in the future. [20] shows how to create a Windows firewall rule that requires IPSec encryption and authentication for users. Administrators can create a rule that requires authentication for computers instead and follow [21] to further configure the rule to apply to SQL Server service. Finally, we need to handle the storage of the connection string which contains the SQL Server Authentication credential and the access to it. We have followed [22] and [23] to store the connection string in web.config and encrypt it via RSAProtectedConfigurationProvider. Instead of granting access to the key container to all users’ Windows accounts, we have granted it only to the Sharepoint application pool identity. In data access layer, we have encapsulated the connection string retrieval into an entry point method for all other methods that handle CRUD operations, utilizing static method RunWithElevatedPrivileges [24] of static class SPSecurity to temporarily impersonate the Sharepoint application pool identity to access the key container.

Last, to prevent cross-site scripting, ASP.NET framework has a feature to disallow unencoded HTML content to be processed by the server [25]. However, it conflicts Sharepoint’s own features and so is turned off in web.config.
As a result, developers need to constrain input in ASP.NET pages more carefully in addition to encoding dynamic output from databases. [26] explains the guidelines on how to prevent cross-site scripting in ASP.NET pages.

### 2.3 Development Environment and Practices

The development of Sharepoint pages is divided into 2 steps (each corresponding to a different part of the development environment):

1. Debug and test pages, data access code, etc., using Visual Studio development server and security related code using a regular ASP.NET web site in Internet Information Services (IIS) whose virtual directory points to the Visual Studio web site project;

2. Deploy them into the beta testing Sharepoint host web application to test the look-and-feel and modify pages if necessary.

The advantage of this separation is that the coupling between the development environment of the ACE Web App and Sharepoint environment has been reduced. For example, when we migrated ACE to the production environment, we didn’t have to wait for the installation and configurations of Sharepoint to complete before we could test database configurations and data access code, which can be done directly from Visual Studio and IIS. Also, debugging ASP.NET pages and code via Visual Studio development server is faster than that via directly deploying them to Sharepoint, in which developers have to wait for the solution items to be deployed to Sharepoint and the AppDomain [27] of Sharepoint to be reloaded [28] in each debug session.

Correspondingly, we have 2 ACE databases in the development environment:

1. The alpha testing database is used to debug and test pages and code run by Visual Studio development server and IIS;

2. The beta testing database is tested by staff and faculty with real data migrated from the existing system and imported into the production database.
The schema of both databases needs to be identical in order for pages and code to work without changes. The normal way is to modify the scripts generated by SQL Server Management Studio (SSMS), specifying the target database in the `USE` statement. But we found the following sequence is faster when the database size is small and changes in an iteration span multiple tables: directly making schema changes in the beta testing database, backing it up, restoring it to the alpha testing database and changing the security settings of users in the alpha testing database.

To simulate the Sharepoint environment in our Visual Studio solution, we use a separate class library project to contain the DLL code and make sure the folder structure of the web site project closely resemble what is in Sharepoint. Although the Sharepoint master page does not work in our Visual Studio solution because of its Sharepoint elements and special syntax, we have created a mockup master page which has `empty ContentPlaceHolder` surrogates for real master page ones and modified the top level master page of ACE Web App to link to them. This way during deployment nothing needs to be changed to pages except the master page reference in the top level master page only if it also needs to be redeployed. As explained in last section, we use static method `RunWithElevatedPrivileges` of static class `SPSecurity` to programmatically impersonate the Sharepoint application pool identity to access the key container to decipher the connection string in web.config of the hosting Sharepoint web application. It causes an exception if the method is used outside Sharepoint\(^2\). As a result, we have used the default conditional compilation constant `DEBUG` to alternate between key access with and without programmatic impersonation. `DEBUG`’s definition can be easily switched on and off via Build tab in Properties page of the class library project and it is the only thing that needs to be changed to the class library project before and after the deployment. When debugging via Visual Studio development server, code runs under the current user’s Windows account [29] and so developers’ Windows accounts need to be given access to the key container for the alpha testing database connection string.

\(^2\) It seems that Microsoft Sharepoint Foundation 2010’s assembly (version 14.0.0.0) raises an exception in the described scenario but Windows Sharepoint Services 3.0’s assembly (version 12.0.0.0) does not.
Occasionally, there is a need to debug the DLL code directly against Sharepoint or security related code against the IIS web site. This can be achieved by manually attaching Visual Studio Debugger to the IIS worker process (w3wp.exe) that runs the Sharepoint web application or the regular ASP.NET web site (use application pool identity to tell which w3wp.exe to attach). Developers need to make sure that the Program Database (PDB) files in Visual Studio class library project exactly matches the PDB files that were created when building the DLL referenced by the worker process [30] [31]. [32] provides an FAQ on Visual Studio Debugger and [33] shows how to debug Microsoft’s source code for .NET.

Last, for errors and exceptions that originate from Sharepoint, either configuration ones or development ones, Windows Event Viewer often provides helpful information that is not available in the stack trace.

2.4 ACE Web App Overview

The building blocks of ASP.NET pages are instances of controls. Although the controls that ship with ASP.NET provide a lot of functionality, we have had to customize a lot of them by inheritance and method overriding to fix bugs³, modify or add functionality. Customization by inheritance is not trivial since developers need to understand the inner workings of built-in controls that are often huge in terms of code size⁴. Another way to customize controls is to compose user controls with existing controls through delegation. The advantage of this over inheritance based customization is that it is similar to black-box reuse and so developers need to understand less about what happens within composing controls. However, developers have less control over internal behaviors of controls and so some bug fixes or workarounds cannot be achieved this way. Besides, if the composing controls are complex and have large numbers of public members, the code needed to implement delegation may be non-trivial -- for each public member in the composing controls whose functionality needs to be exposed, developers need to create a corresponding wrapper member for it in the composed user control.

³ Microsoft Connect (http://connect.microsoft.com/) is reliable source for bug fixes and what makes it more helpful is the workarounds posted by other developers for reported bugs.
⁴ An invaluable tool for researching existing controls’ behaviors is .NET Reflector.
For simplicity, in the rest of this section, we’ll use the original control name prefixed with “Custom” to refer to a control resulting from customization by inheritance.

Based on their data scopes, pages in ACE Web App can be divided into several groups like grad application pages, enrolled grad student pages, etc. Each group of pages has a similar structure: a search page and one or more detail pages. On the search page, users can utilize search control instances to specify search criteria to find student records and the data presentation control instance displays the results with summary information and links to their detail pages. A detail page uses its query string to identify the student record that the page is for and uses data presentation control instances to show database records conceptually contained by the student record. Data presentation control instances on a detail page also provide interfaces for users to perform insert, update and delete operations. Data source control instances are used to link all the data presentation control instances to data access code in the DLL. In ACE, we use CustomDetailsView and CustomGridView as the data presentation controls, CustomObjectDataSource as the data source control. Search controls are user controls, the cores of which are CustomTextBox, CustomDropDownList, CustomCheckBoxList.

The ASP.NET pages deployed into the Sharepoint document library are basic units of role based authorization in ACE Web App. For example, to fulfill the requirement that graduate counselors can modify the education background records of students whereas faculty can only view them, we need to have 1 page for graduate counselors through which data can be modified and 1 page for faculty through which data can only be read. Each page will be deployed to a folder in the Sharepoint document library to which only users in the corresponding role are granted “Read without Source” permission (explained in Section 2.2). In other words, role based authorization is achieved by the contents of pages that various roles are allowed to read from Sharepoint.
In ACE, there are generally 3 conceptual data access permission levels for a database record: no read, read and write. For database records that are designed to be showed on a same page, although different roles have different permission levels for them, the control markups and code used to implement different permission levels have a lot in common. In order to reuse them, for each group of database records, we have developed a user control to contain all their control markups and code. In each page we declare an instance of the user control with property values appropriate for the target role. A user control instance programmatically changes the states of the contained control instances depending on its own mode.

Here is an example that explains how this is implemented for education background records on grad application detail page. We use a CustomDetailsView instance together with a CustomObjectDataSource instance to present a student’s education background records. The page user control contains their instance declarations and related code for write permission level. In CustomDetailsView, we have a SetReadOnly method that hides all the buttons used to perform write operations or change a CustomDetailsView instance to non-read only modes with write operation interfaces. In CustomObjectDataSource, we have a SetReadOnly method that invalidates the property values used to invoke write operation methods in the DLL. During each page request, the user control instance reads the property value set by the containing page to determine its mode. If the mode’s permission level for the education background records of the student is read, both SetReadOnly methods of the CustomDetailsView and CustomObjectDataSource instances will be invoked. If it is no read, the entire CustomDetailsView instance will be hidden and its link to the CustomObjectDataSource instance severed. If it is write, no changes need to apply. The actions that involve the CustomObjectDataSource instance are used to make sure this method is secure by removing the links from data presentation control instances to data access methods in the DLL. The actions that involve the CustomDetailsView instance are designed to change user interfaces but together with the event validation feature [34] in ASP.NET, they provide an extra line of defense. [35] shows how this
feature helps prevent users from spoofing event postback mechanism to click an invisible button. To utilize this feature, developers first need to make sure `EnableEventValidation` setting is not turned off in `Page` directive or `web.config`. Second, customized controls must be decorated with `SupportsEventValidationAttribute` so that base controls validate events for them [36]. All roles should be granted “Read without Source” permission to the folders containing shared resources by all pages such as page user controls, master pages, etc.

While role based authorization via ASP.NET pages is the main authorization scenario in ACE, it cannot fulfill one of our requirements that demands authorization within a role. The requirement is that faculty can only view the academic records and employment records of students that they advise or support. Neither developing a page for each student nor creating a permission granting folder for each faculty is practical. As a result, we have developed a database row based authorization in code-behind classes with the help of database data. Figure 2.1 illustrates the database tables involved, which are simplified from tables in ACE DB.

![Database tables](image)

Figure 2.1: Database tables used in database row based authorization
The majority of the authorization rule is already in the CurrentAdvisers table between the CurrentStudentRecords table and FacultyBasicInfos table. The only missing thing is the links from faculty information to their account in AD. As a result, we added the UserAccounts table and stores the links through the foreign key (FK) relationship between the UserAccounts and CurrentAdvisers table. Additionally, we added the AdditionalFacultyViewers table between the FacultyBasicInfos table and the CurrentStudentRecords table to store authorization rules between non-adviser faculty and students.

In the code-behind classes of detail pages containing academic records and employment records, during each request, a data access method is invoked to check whether the current user belongs to the CurrentAdvisers or AdditionalFacultyViewers table. If so, normal data is returned to users. Otherwise, error message is displayed. An important prerequisite of this method is the integrity and confidentiality of communication between the application server and database server, which is provided by IPSec in ACE (See Section 2.2 for more details).
Chapter 3: Accessing Data with LINQ to SQL

3.1 Introduction

LINQ (stands for Language Integrated Query) is a Microsoft .NET Framework component that adds native data querying capabilities to .NET languages [37]. LINQ to SQL is Microsoft’s entry-level LINQ-enabled object relational mapping (ORM) implementation for SQL Server [38].

LINQ to SQL is built on top of traditional ADO.NET utilizing its data access infrastructure. With LINQ to SQL, developers can save a great amount of time and effort in writing and debugging data access code. For example, it uses parameterized SQL queries behind the scene to prevent SQL injection [39], which frees developers from writing lengthy parameterized SQL queries themselves. Also, because database operations are accomplished through the strongly typed object models inherent in and generated by LINQ to SQL rather than text based Transact SQL (T-SQL), the compiler can help catch a lot of errors which would otherwise lead to runtime exceptions. Additionally, Visual Studio’s productivity tools for strongly typed languages such as IntelliSense and Refactor are available to data access code written in LINQ to SQL. Overall, LINQ to SQL improves software quality and development productivity.

3.2 Object Relational Mapping

LINQ to SQL provides 2 ways for developers to map classes to relational data: one is through an external XML mapping file and the other is by decorating classes with attributes. There are 2 code generation tools available to help generating and maintaining the code and mapping: SQL Metal and Object Relational Designer (O/R Designer). SQL Metal is a command line tool that supports both external XML based mapping and attribute based mapping, whereas O/R Designer provides a graphical user interface (GUI) in
Visual Studio and only supports attribute based mapping [40]. SQLMetal can be used to generate code and mapping of the entire database in a single batch with default settings but customizations afterwards needs to be achieved in other means. O/R Designer, although limited to attribute based mapping, provides an intuitive and familiar GUI for incremental customizations. In ACE, there is nothing that prevents us from having a model decorated with attributes. Since the productivity gain by using O/R Designer is huge, we have used attribute based mapping in ACE. Also, SQLMetal can be used to generate database markup language (DBML) metadata for the entire database, which can serve as the initial input to O/R Designer. This further simplifies the process of generating class definitions and their attribute based mapping from relational data. [41] and [42] include lists of attributes used in mapping.

LINQ to SQL classes that are mapped to database tables and views are called entity classes [43]. As an entry level ORM implementation, LINQ to SQL only supports mapping an entity class to a single table or view. Although it is possible to map multiple entity classes to the same table, we have not found much practical use of it except when used to map class inheritance. [44] summarizes and compares 3 schemes of mapping class inheritance to relational tables: table per hierarchy (TPH), table per concrete class (TPCC) and table per class (TPC). LINQ to SQL only has built-in support for TPH, which is explained in [45]. In short, in TPH, entity classes in the same class hierarchy are mapped to a single table and a discriminator column is used to mark the type of each record. After specifying the mapping details in O/R Designer, LINQ to SQL includes the class inheritance part in the generated entity class definitions and at runtime it uses the value in the discriminator column to map a table record to an entity instance of the correct type in CRUD operations. However, if entities are complex, having different entities in the same table makes the database design hard to understand and vulnerable to change. In ACE, we only use TPH for a simple class hierarchy with 1 level of inheritance. A coincidental benefit of it is that we could use a unique check constraint (which only works within table boundaries) on a column that should have unique values across instances of all types in the hierarchy, which otherwise would be implemented using in a more complex way such as triggers or in the object-oriented (OO) layers above. Also, while exploring TPH with LINQ to
SQL, we discovered it is hard to achieve a common scenario where child classes have a specialized 1-to-many relationship where the underlying table of the class hierarchy holds unique keys referenced by others. [46] describes our attempts and the final workaround. During our attempts, we found and reported a bug that at least exists in .NET 3.5 and it was confirmed by Microsoft [47] [48]. The bug prevents developers from having the properties of child classes mapped to columns whose values are database generated.

Because the entity class model generated by LINQ to SQL closely resembles the relational model, developers often need to define their own classes composed of entity classes that better fit requirements. In ACE, we have a business object class for each entity class in order to:

1. Flatten nested entities resulting from joined query results so that it is easier for ASP.NET controls to perform member access.
2. Transfer additional non-entity members that will be used to perform database write operations.

The entry points of CRUD operations are DataContext instances. DataContext class can be thought as the manager of LINQ to SQL which abstracts away the database management system (DBMS). It provides access to database table abstractions -- the generic class Table<TEntity>, where TEntity is an entity class. Developers can then instruct LINQ to SQL to perform CRUD operations on corresponding tables. Behind the scenes, DataContext does all the hard work to enable this, such as managing database connections, translating LINQ to SQL commands to T-SQL commands that SQL Server understands, tracking the states of entities accessed in the current session, detecting and handling concurrency conflicts in the database write operations, etc. O/R Designer also generates a custom DataContext class specifically for the target database, which provides direct member access to database table abstractions through properties.

3.3 Queries

Listing 3.1 shows a sample LINQ to SQL query, which is simplified from code in ACE.
public static IList<ZV_GradAppEducBkgd> SelectZV_GradAppEducBkgdSample
(Guid gradAppRecordGUID, int startRowIndex, int maximumRows)
{
    using (ACEDBDataContext db = ACEDBDataContext.NewReadOnlyContext)
    {
        IQueryable<GradAppEducBkgd> bkgds = db.GradAppEducBkgds
            .Where(x => x.GradAppRecordGUID.Equals(gradAppRecordGUID));

        IQueryable<ZV_GradAppEducBkgd> zv_bkgds = from bkgd in bkgds
            from zRank in db.Z_GradAppEducBkgdRankChoices
            .Where(x => x.GradAppEducBkgdRankChoiceGUID.Equals(bkgd.RankGUID))
            .DefaultIfEmpty()
            select new ZV_GradAppEducBkgd
            {
                GradAppEducBkgd = bkgd,
                ZV_Rank = zRank.Rank,
            };

        IQueryable<ZV_GradAppEducBkgd> orderedBkgds = zv_bkgds.OrderBy(x => x.ZV_Rank)
            .ThenBy(x => x.GradAppEducBkgd.GradAppEducBkgdGUID)
            .Skip(startRowIndex).Take(maximumRows);

        return orderedBkgds.ToList();
    }
}

Listing 3.1: A sample LINQ to SQL query

The above query is used to get education background records associated with the specified application record after sorting and paging. Line 6-7 shows the subquery that filters education background table by the specified primary key (PK) of an application record. Line 9-19 shows the subquery that joins filtered results to a category table [49] to form the desired business object. Line 21-24 shows the subquery that further sorts and pages the results. Line 9-19 is written in a mix form of SQL-like query expressions (from, select, etc.) and dot notation query operators (Where, OrderBy, etc.). Both are valid ways to express LINQ to SQL query and as Listing 3.1 shows they can be used together. The choice of which to use depends more on developers’ personal preferences and query expressions are actually translated to query operators by the compiler [50]. In ACE, query operators are mostly used except when joins are involved. Query expressions are more readable when expressing joins, especially when there are multiple joins in the query. Line 9-19’s way to join tables is also recommended by [51] because it is more succinct and flexible than utilizing the built in join clause in query expression.
The result types of most query operators are of generic interface `IQueryable<T>`. An `IQueryable<T>` instance does not represent the actual database tuple results but contains information about how to retrieve them, as well as the LINQ to SQL query represented as an expression tree (a kind of abstract syntax tree [52]) and a provider instance that knows how to translate expression trees into the underlying query language of the data source (T-SQL of SQL Server) and execute them [53] [54]. The actual query results are retrieved when an `IQueryable<T>` instance is first enumerated, which in Listing 3.1 is at Line 26 when `ToList()` is called.

The `ACEDBDataContext` class is the custom `DataContext` class generated by O/R Designer for the ACE DB that provides direct access to database tables. An instance `db` is created at Line 4 and used as the entry points of the LINQ to SQL queries at Line 6 and Line 11 to reference database table abstractions -- `db.Z_GradAppEducBkgdRankChoices` and `db.GradAppEducBkgd`, which are `Table<TEntity>` instances. The lifetime of `DataContext` instances is meant to be short: It is designed for one “unit of work” [55]. A `DataContext` instance that lives too long is likely to return stale data because it returns the cached results after the initial data retrieval [56]. Also, instance members of `DataContext` class are not thread-safe [55] [57]. As recommended in [57] and [58], in ACE all methods that handle CRUD operations create and use their own `DataContext` instances and dispose of them before exiting. In Listing 3.1, this is achieved by scoping `db` within the outermost `using` block (starts at Line 4 and ends at Line 27). Additionally, developers need to be careful not to enumerate an `IQueryable<T>` instance after disposing of its entry point `DataContext` instance, because otherwise an exception will be thrown. The explanation is as follows:

The `Table<TEntity>` class also implements `IQueryable<T>` and a `Table<TEntity>` instance serves as its own provider instance, which in turn uses the entry point `DataContext` instance to translate and execute LINQ to SQL queries. When we apply standard operators (defined in `Queryable` class) on a
source IQueryable<T> instance, the provider instance of the resulting IQueryable<T> instance is the same as that of the source IQueryable<T> instance [54]. As a result, all the IQueryable<T> instances that originate from a Table<TEntity> instance via standard operators use the Table<TEntity> instance as the provider instance, which in turn uses the entry point DataContext instance to translate and execute LINQ to SQL queries. Consequently, the entry point DataContext instance should live longer than IQueryable<T> instances that originate from it. In Listing 3.1, no IQueryable<T> instances are used outside the using block of db and orderedBkgds is enumerated to return the results before exiting the block.

When writing LINQ to SQL queries, developers need to be aware that there are a finite number of .NET common language runtime (CLR) constructs for which LINQ to SQL query provider supports the translation [59]. But the coverage of those building blocks is large enough so that most declarative set-based T-SQL queries can be expressed equivalently in LINQ to SQL. For example, one handy C# operator that we often resort to when conditional logic is needed is the ternary operator ?:, which will be translated to case expressions in T-SQL that can be nested and contain subqueries as inner expressions. In ACE, LINQ to SQL queries are able to fulfill all the query requirements besides the ones involving binary large objects (BLOB) and the explanation is as follows: LINQ to SQL uses Binary class to represent binary objects; Binary class is an immutable version of byte[] and stores all the content in memory, which will use up server memory very fast in a concurrent environment; So we use traditional ADO.NET to read/write BLOBs in a stream fashion by following [60] without its bugs.

Based on those building blocks, developers can easily encapsulate common queries in methods and return IQueryable<T> for reuse. Listing 3.1 is actually combined from 2 methods in ACE -- with Line 9-19 put in a separate reusable method whose function is to form the desired business object out of the input parameter. A finer level of reuse can be achieved by writing methods that build expression trees. In Listing
3.1, query operators Where and OrderBy take lambda expressions as input parameters. Lambda expressions are compiled into expression trees (in the form of the abstract class Expression) when used with LINQ to SQL query operators. In this way, developers specify specific expression trees that are static at runtime. To generate expression trees that are dynamic at runtime, developers need to use static methods in Expression class to explicitly construct expression trees. The construction process encapsulated in methods can then be reused and the resulting expression trees can be passed to query operators to compose queries. In ACE, it is very common that users need to search for records that match a certain number of criteria. For example, Admission committee members need to search for applicants with interests in research areas that they are in charge of. Both the number of research areas and the values of research areas are given at runtime and so cannot be implemented as in Listing 3.1. To solve this, we have built a generic method that generates expression trees based on user supplied search criterion items. The expression trees are then consumed by Where query operator. Because the method is generic it can work with any entity class and is reused in all similar search scenarios. [61] shows how to create queries that sort results depending on users’ runtime input and we have enhanced its method in ACE by modifying it to use generics and be able to parse general T-SQL order by clause.

Developers also need to be aware that some query translations by LINQ to SQL are not as straightforward as they seem, which may cause bugs if not handled carefully. For example, Null Semantics mentioned in [62] are common causes of bugs when dealing with nullable types. [59] provides a list of references concerning the details of data type and function translations, which developers need to master. Also, as almost everything else, the LINQ to SQL query translation process contains bugs. For example, [63] describes the symptoms of a bug that at least exists in .NET version 3.5. Line 23 in Listing 3.1 is a stronger remedy for the bug than the method mentioned in [63]. In short, when query operator Skip is used to page results, an input value of 0 will cause LINQ to SQL to consider it as a no-op whereas other input values will cause it to be translated into a T-SQL query using ranking functions to order the results and then paging results. Since the order of results is unknown when no order is specified, it is possible the same
record that appears on the first page under an unknown order caused by \texttt{Skip(0)} also shows up in a following page under a specified order caused by \texttt{Skip(n)}. Line 23 fixes this by always ordering the result on a unique column (PK in this case) right before paging so that a consistent order is imposed in both \texttt{Skip(0)} and \texttt{Skip(n)} situations.

To troubleshoot bugs and performance problems, 2 tools are especially helpful: LINQ to SQL Debug Visualizer -- a Visual Studio add-in that can show T-SQL queries during debugging, which is explained in [64] and SQL Profiler\textsuperscript{5} -- a tool that can show how almost everything take places in SQL Server, which is explained in [65].

### 3.4 Inserts, Updates and Deletes
In a connected environment, often the same \texttt{DataContext} instance is used to track an entity’s identity, tracks and process its changes, detects and resolve concurrency conflicts in its whole lifetime, which makes performing database write operations in LINQ to SQL relatively straightforward. But web applications such as ACE live in disconnected environments, where entities are serialized to HTTP responses to users and resurrected or born via deserializing users’ HTTP requests. This completely disconnects each entity and its creator \texttt{DataContext} instance if any. The recreated entity is not recognized by the creator \texttt{DataContext} instance and by the time it is recreated, the creator \texttt{DataContext} instance should already be disposed (see section 3.3 for reasons why \texttt{DataContext} instances should be short-lived). Also, the built-in concurrency conflict resolution cannot be used in most disconnected scenarios since for short-lived \texttt{DataContext} instances, the concurrency conflict resolution needs to happen right after concurrency conflict detection and so no user input can be taken in the middle. Consequently, whereas inserts are used in the same way as in connected environments, deletes and updates should be used differently in disconnected environments. [66] and [67] provides brief introductions to working with

\textsuperscript{5} Developers need to include RPC related events in SQL Profiler traces to see T-SQL statements generated by LINQ to SQL.
disconnected data, which we have found far from sufficient to guide developers. Thus, the rest of this section documents our experiences.

Listing 3.2 shows a sample disconnected LINQ to SQL update, which is simplified from code in ACE.

```csharp
public static int UpdateSample(ZV_GradAppEducBkgd oldBO, ZV_GradAppEducBkgd newBO)
{
    using (ACEDBDataContext db = ACEDBDataContext.NewContext)
    {
        int affectedRecord = 0;
        db.GradAppEducBkgs.Attach
            (newBO.GradAppEducBkgd, oldBO.GradAppEducBkgd);
        ChangeSet set = db.GetChangeSet();
        if (set.Updates.Contains(newBO.GradAppEducBkgd))
        {
            db.SubmitChanges();
            affectedRecord = 1;
        }
        return affectedRecord;
    }
}
```

Listing 3.2: A sample disconnected LINQ to SQL update

The method in Listing 3.2 is used to update an education background record. It is called after the server receives a user’s HTTP request which contains the original values (represented by input parameter oldBO) and the modified values (represented by input parameter newBO) of the education background record. The return value is the number of records modified in the database by the method.

The essence of the method is the statement at Line 8 and 9, where the Attach method of generic class `Table<TEntity>` is called. This method attaches the newly created entity contained in newBO to the fresh DataContext instance db so that db can identify it, track and process changes to it. The other input parameter oldBO contains a reincarnation of the entity with the old values as set in the previous
HTTP response to users, which will be used by db as the original values of the entity in the change tracking service, and to generate any T-SQL statements needed to detect concurrency conflicts during submission. If entity values in newBO and oldBO are different, db adds the entity to its to-update list and will only update the fields that are different during submission. Line 11-13 programmatically checks whether an entity is in the list so that the number of records modified in the database can be set accordingly. When SubmitChanges is called at Line 15, db iterates over those lists, generates and sends T-SQL statements to the database to persist all the changes in a single transaction. If there are errors causing the transaction to abort and rollback such as detected concurrency conflicts, SubmitChanges throws an exception that contains the details. The user interface such as ASP.NET pages up in the call stack can handle exceptions and decide how to display error messages to users or confirmations when operations succeed. If there are concurrency conflicts, after seeing the new entity values in the database, users can decide either to accept them or modify them and resubmit.

Developers may ask: “why cannot we update like we do in connected environments?” [66] says that if we cannot use Attach, we can first query the entity from the database to replace the entity in oldBO, set its values according the values in newBO and then call SubmitChanges, which is similar to what happens in connected environments. But this invalidates concurrency conflict detection. The concurrency conflict detection scheme in LINQ to SQL works by generating T-SQL statements that compare the original values of the entity to the database values when performing T-SQL write operations. If there is any discrepancy, it means the database values have already been updated or deleted by others. If we follow [66]’s way, when performing write operations, the DataContext instance will use wrong values (the values in the newly fetched entity from the database) as the original values and try to compare them with database values to detect conflicts, whereas the correct original values are the values in oldBO. It nullifies the basis of LINQ to SQL’s concurrency conflict detection.
The key to perform write operations correctly in disconnected environments is to provide accurate disconnected modification information to a new `DataContext` instance, which `Attach` is designed for.

There are several overloaded versions of `Attach` [68]. They can all attach a disconnected entity to a `DataContext` instance which will manage the entity subsequently, but they differ in whether the entity is considered as modified by the `DataContext` instance’s change tracking service and if so whether original values are supplied. The rules of using `Attach` are very strict and it is very easy to overlook one of them and thus receive a runtime exception, which accounts for a lot of discussions in Microsoft Developer Network (MSDN) forum. We have compiled several common scenarios and show the rules in Listing 3.3.

```
1 public static void RulesUsingAttach 
2        (ZV_GradAppEducBkgd oldBO, ZV_GradAppEducBkgd newBO) 
3 { 
4     using (ACEDBDataContext db = ACEDBDataContext.NewContext) 
5         using (ACEDBDataContext db1 = ACEDBDataContext.NewContext) 
6             { 
7                 db.GradAppEducBkgds.Attach(newBO.GradAppEducBkgd); 
8                 //System.InvalidOperationException: 
9                 //Cannot attach an entity that already exists. 
10                 //db.GradAppEducBkgds.Attach(newBO.GradAppEducBkgd); 
11                 //System.Data.Linq.DuplicateKeyException: 
12                 //Cannot add an entity with a key that is already in use. 
13                 //db.GradAppEducBkgds.Attach(oldBO.GradAppEducBkgd); 
14                 //System.NotSupportedException: 
15                 //An attempt has been made to Attach or Add an entity that is not new, 
16                 //perhaps having been loaded from another DataContext. This is not supported. 
17                 //db1.GradAppEducBkgds.Attach(newBO.GradAppEducBkgd); 
18                 //OK 
19                 db1.GradAppEducBkgds.Attach(oldBO.GradAppEducBkgd); 
20             } 
21             using (ACEDBDataContext db2 = ACEDBDataContext.NewContext) 
22                 { 
23                 //System.NotSupportedException: 
24                 //An attempt has been made to Attach or Add an entity that is not new, 
25                 //perhaps having been loaded from another DataContext. This is not supported. 
26                 //db2.GradAppEducBkgds.Attach(newBO.GradAppEducBkgd); 
27                 //OK 
28                 db2.GradAppEducBkgds.Attach(newBO.GradAppEducBkgd); 
29             } 
30         } 
31     } 
32 }
```

Listing 3.3: A sample that shows the rules of using method `Attach`
Listing 3.3 uses the simplest form of Attach but the usage applies to all 3 versions. At Line 7, the entity in `newBO` is attached to the `DataContext` instance `db` and statements at Line 11, 15, 20 and 31 will give us exceptions if uncommented with the exception details above them. Line 11 is easy to understand and just do not try to attach an entity instance to the same `DataContext` instance again. Line 15 tries to attach another entity instance to `db` but would fail since it has the same PK value as an entity already tracked by `db` -- the entity in `newBO`. This is because `DataContext` class’s identity tracking service relies on PK value(s) to identify entities and identities should be unique. Another implication of this is that developers cannot update PK values through LINQ to SQL because they define entity identities. So when designing database tables, special PK columns should be used rather than combinations of meaningful columns whose values may change over time. Line 20 tries to attach an entity to a `DataContext` instance while it is already being tracked by another one. Line 31 shows that even after the hosting `DataContext` instance of the entity in Line 20 is disposed of, it still cannot be attached to another `DataContext` instance. Line 23 is valid because the entity instance is not already associated with any `DataContext` instances and there is no entity identity clash within the `DataContext` instance. Line 23 has a useful derivation. Occasionally, we need to first query database and perform write operations depending on current database values. To achieve this in disconnected mode, we need to create a transaction with isolation level serializable so that the database values are locked during the transaction [69], query the entity with one `DataContext` instance (which associates the entity with it) and use another `DataContext` instance to `Attach` so that no rule is violated. Dealing with transactions is explained in more detail later in this section.

Listing 3.4 shows a sample disconnected LINQ to SQL delete, which is different than the approach in ACE.
In Listing 3.4, an unmodified entity with its original values is attached at Line 5 and the rest of the statements are the same as in connected environments. The reason why this approach is not used in ACE is that LINQ to SQL has a problem in concurrency conflict detection with deletes as shown in [70]. In short, if the entity record in the database has already been deleted when Line 9 executes, LINQ to SQL does not raise a ChangeConflictException like it does for updates and so the code in Listing 3.4 cannot tell the difference between the situation that the record is deleted by the current method and the one that the record has been deleted previously. A data access library should only detect the conflicts but not decide whether the conflict can be ignored. It should be the caller method’s responsibility to decide how to handle the conflict. In ACE, we use a different way to accomplish deletes, which is showed in Listing 3.5.
public static int DeleteSample2(ZV_GradAppEducBkgd newBO) {
    int affectedRecord = 0;
    using (TransactionScope ts = DataContextManager.NewRequiresNewSerializableScope)
    using (ACEDBDataContext db = ACEDBDataContext.NewContext)
    {
        GradAppEducBkgd entityInDB = db.GradAppEducBkgds
            .SingleOrDefault(x => x.GradAppEducBkgdGUID.Equals(newBO.GradAppEducBkgdGUID));
        bool alreadyDeleted;
        if (entityInDB == null)
        {
            alreadyDeleted = true;
        }
        else
        {
            alreadyDeleted = false;
            if (object.Equals(entityInDB.LINQVersion, newBO.LINQVersion) == false)
            {
                throw new ChangeConflictException(
                    "Row changed before trying to delete.");
            } db.GradAppEducBkgds.DeleteOnSubmit(entityInDB);
        }
        if (alreadyDeleted == false)
        {
            affectedRecord = 1;
        }
        ts.Complete();
    }
    return affectedRecord;
}

Listing 3.5: A sample disconnected LINQ to SQL delete with concurrency conflict detection fixed

The method in Listing 3.5 first queries for the entity from the database at Line 8-10. It then uses a boolean variable to indicate whether the entity has already been deleted at Line 12 and set its value at Line 16 or 20 depending on the situation. If the entity has not been deleted, the method compares the value in the rowversion field [71] of the entity with that in the old values in the previous HTTP response to user (represented by newBO) to determine whether the entity has been changed at Line 22 and if so it raises a ChangeConflictException at Line 24-25. In ACE, Line 8-30 has been re-factored into a reusable
method which can be used in all such situations. The caller method can then decide how to handle the conflict situation.

To ensure the correctness of the method, it must be made sure that the entity values queried from the database at Line 8-10 will not be changed during the lifetime of each method’s execution. This is achieved by wrapping all relevant operations in a transaction with isolation level serializable so that the database values are locked during it [69]. The transaction is implicitly managed by the `TransactionScope` [72] instance created at Line 5. If an ambient transaction is available, `SubmitChanges` enlists in it rather than creating one of its own [73]. The implicit transaction commits if the method execution hits Line 37. Otherwise, it aborts and rolls back all the changes made in the same transaction. In ACE, almost all methods handling database write operations are not as simple as the one shown in this section, in that they may contain multiple operations whose atomicity, consistency, isolation and durability (ACID) need to be protected together. However, they all use `TransactionScope` class to accomplish this.

A requirement of using the method in Listing 3.5 is that the entity table must include a `rowversion` column that is mapped to an entity class field. This is not a problem for ACE because we have the liberty to change database schema and have a `rowversion` (also called `timestamp`) column for each table so that it can be used to detect concurrency conflicts. For scenarios where this is impossible, developers may need to write methods to compare values in each and every field of the two entities.
Chapter 4: ACE Batch Import

4.1 Operation

ACE Batch Import is a command line application routinely run by administrators to import applicant records into ACE DB from input files downloaded from SIS by graduate counselors. It consists of 2 physical files: SISCSVBatchImport.exe and ACEDBBruceDLL.dll (ACE Web App DLL). It takes 3 arguments: an input file path, a file type and a database connection string. .NET Framework 3.5 is required for ACE Batch Import’s execution.

There are 2 types of input files:

1. PHD pending file that contains a list of PHD program applicants that are in “Pending” stage in SIS, which means their materials are considered complete by the graduate school and are ready for departmental review. PHD pending file is imported every week during normal admission time but the frequency can be as high as every 1 or 2 days during peak time (December to February);

2. MS admit file that contains a list of MS program applicants that are in “Admit” stage in SIS, which means they have already been recommended for admission by the department and approved by graduate school. MS admit file is imported every quarter before the end of second week when all applicant records for that quarter is removed from SIS.

Both files are originally in Excel file format (.xlsx suffix). Graduate counselors or administrators need to save them in comma separated value (CSV) file format (.csv suffix) from Excel and pass those .csv files to ACE Batch Import.

There are output 4 log files for both input file types:
1. Data error log. If an application record in the input file contains an unexpected value in any column, the row number in the input file and errors will be recorded in this log. Normally this log should only contain general information about the batch without any application records. If not, it is an indication that the code of ACE Batch Import needs to be modified in response to the SIS changes; In the meantime, graduate counselors can use the log to locate original rows in the input file and manually enter them via ACE Web App.

2. Duplicate ID error file. If a record in ACE DB and an application record in the input file have identical OSU ID values but different admission term values, the row number, OSU ID value and admission term value of the latter one will be recorded in this log. If so, graduate counselors need to review ACE Web App data and compare errors with them to decide the actions need to be taken. For PHD pending files, the most common cause is that an applicant applies a second time. In this case, graduate counselors need to manually create a new graduate application record with the correct admission term value in ACE Web App rather than overwriting the admission term of the existing application record because admission committee wants to see the application history. Also, simply overwriting the admission term of the existing application record without resetting admission results or application materials attached will make the application overlooked by admission committee when searching or reviewed with old materials. For MS admit files, the most common cause is that a MS student delays admission to a later term. In this case, graduate counselors can simply overwrite the admission term of the existing application record since currently MS applications are not reviewed in ACE and so history is not necessary. Should the process change, the actions may need to be adjusted correspondingly.

3. Already imported record log. If a record in ACE DB and an application record in the input file have identical OSU ID and admission term values, the row number, OSU ID value, admission term value and whether the record in ACE DB was manually entered or batch imported will be recorded in this log. No actions need to be taken for logged records.
4. Successfully imported record log. If an application record is successfully inserted to ACE DB, the row number, OSU ID value and admission term value will be recorded in this log. Graduate counselors can use this log as the basis of further data entry.

If there is any fatal error during the batch import, all 4 log files will output the fatal error instead of normal content. This would happen when the input file structure is unexpected, such as wrong column number, wrong column name in the header, etc., and the code of ACE Batch Import must be changed.

Finally, here is a list of possible command line output from ACE Batch Import:

1. If the batch import is successful, it outputs "Batch import Successful. See log files for results".
2. If there is any input error for command line arguments, the error and the correct usage will be output to command line.
3. If there is any exception during the batch import, the details including the stack trace will be output to command line. This is because ACE Batch Import is only intended to be executed by administrators or developers, who need exception details to troubleshoot.

4.2 Security

Administrators should use a least privileged Windows account rather than administrator accounts to run ACE Batch Import in case the program executes in an unexpected way. The account should have permissions to create files in the input file folder since that is where the log files will be generated. The account can be used in Windows Integrated Authentication to SQL Server database engine so that no account information needs to be specified in the database connection string. No matter whether the account used to connect to the database is a Windows account or a SQL authentication account, it should be only added to the “ACE_Batch_Import” database role when setting its user mapping on ACE and not given permissions on any securables directly. “ACE_Batch_Import” are given only select and insert permissions on appropriate tables that ACE Batch Import needs to access and it makes sure that if the program behaves unexpectedly, the scope of damage can be limited. Since the database server’s Windows Firewall group
policy only allows secure connections from a list of authorized computers to the SQL Server database engine, the computer where ACE Batch Import is run must be added to the IPSec group policy to connect securely and to the Windows Firewall group policy to be authorized.

4.3 Design

There are 3 stages for each batch import: command line argument validation, input file parsing and database operations. The latter 2 stages are coordinated by a BatchImportDirector instance. In the command line argument validation stage, the following items are validated:

1. The number of arguments should be 3.
2. The input file path should be valid. A test StreamReader instance will be created to access the file with the input file path.
3. There should not be any file in the input file folder having the same name as that of any log file that will generated. This is used to prevent log files being overwritten by accidental re-execution of the program.
4. The database connection string should be valid. A test SqlConnection instance will be created to connect to the database with the connection string.
5. The file type should be M (for MS admits files) or P (for PHD pending files).

In the input file parsing stage, the program reads and parses each row in the input file. The parsing of each row can be subdivided into 2 steps. The first step is to create an InputFileRow instance which stores the row fields in an IList<string> instance. The second step is to convert the InputFileRow instance to an SISAdmissionExcelParsedRow instance, which knows how to insert itself into ACE DB with the further processed fields. A large part of step 1 is implemented following the state design pattern [74]. Figure 4.1 shows the class diagram and Figure 4.2 shows the state diagram. For simplicity, only the main methods are shown in those figures without input parameters and return values.
Figure 4.1: Class diagram of input file parsing step 1.

Figure 4.2: State diagram of input file parsing step 1.
Note in the class diagram, BatchImportDirector associates with the IInputFileRowFactory interface rather than the concrete ExcelCSVRowFactory class. This enables developers to easily add concrete factory classes for other file formats should SIS change input file formats. To help understand the state diagram, here is an explanation of the CSV file format generated by Excel: if a field does not contain any double-quote or comma, the field is separated by commas. Otherwise, the field is enclosed by a pair of double-quotes and then separated by commas. Furthermore, a double-quote is represented by 2 double-quotes in the field and a comma is still represented by itself.

After the ExcelCSVRowFactory instance uses the state transitions to create an InputFileRow instance, it uses the IInputFileRowStructureValidator instance that passed by the BatchImportDirector instance to validate the general structure of the InputFileRow instance. There are 2 concrete classes of the IInputFileRowStructureValidator interface: HeaderRowStructureValidator<E> and DataRowStructureValidator<E>. The type parameter of both takes an enum that is defined for a specific input file type. Corresponding to the 2 input files types, we have AdmissionMSAdmitFieldEnum and AdmissionPHDPendingFieldEnum. The enumeration values of both enums are names of input file columns whose integer values correspond to column indices. This way the column indices can be used in a “strongly-typed” way in the rest of the code. The enumeration values are also decorated with StringDescriptionAttribute instances containing as strings the expected column names in the header rows of input files. A HeaderRowStructureValidator<E> instance uses those strings to check whether the field values of the header InputFileRow instance are expected. Also, both HeaderRowStructureValidator<E> and DataRowStructureValidator<E> instances check whether the number of fields in the InputFileRow instances match the number of enumeration values in the enums. If any validation fails, the whole batch import will abort since the general structure of the input file is not correct, which is considered a fatal error. As a result, the 2 enums need to be modified should input file structures change.
Step 2 of the input file parsing stage is implemented following the abstract factory [75] design pattern. Figure 4.3 shows the class diagram. For simplicity, only the main methods are shown in those figures without input parameters and return values.

The **BatchImportDirector** instance uses an **ISISAdmissionExcelParsedRowFactory** instance to further parse the **InputFileRow** instance and store the further parsed field information (represented by **ParsedField<V>** instances) and errors (represented by **ParsedFieldError** instances).
instances) in SISAdmissionExcelParsedRow instances. This step depends on the category records in ACE DB. For example, the value in "Admit Term" column for each row is parsed and converted to a pair of values that will be inserted into ACE DB. The pair of values is stored in a ParsedField<AdmitTermDBValues> instance with 2 members: PlannedEnrollmentYear of type short and PlannedEnrollmentTermGUID of type global unique identifier (GUID). PlannedEnrollmentTermGUID contains the PK value of the category record in Z_AcademicTermChoices table of ACE DB that corresponds to the term string contained in "Admit Term" column. ACE Batch Import uses constants to store PK values of category records used in parsing. Those constants are contained in the HelperCategoryTableGUIDs.cs file under the Helper folder. For existing constants, we have referenced category record PK constants in ACE Web App DLL to make them consistent. For the new ones developers should directly assign PK values in ACE DB to them.

The BatchImportDirector instance then uses the operations exposed by SISAdmissionExcelParsedRow instances to conduct the database operations stage of the batch import. All operations in this stage are wrapped in the same transaction using TransactionScope class as explained in Section 3.4 and so if any of operation fails, all operations will abort and roll back. As mentioned in Chapter 1, all database operations are achieved via ADO.NET but in the development we still use constants to reference the metadata of LINQ to SQL entity classes in ACE Web App DLL such as table names, column names, etc., to construct T-SQL statements. The main advantage is that when the database schema and LINQ to SQL object mapping change, the metadata also changes. As a result, we do not need to change T-SQL statements for small changes like table or column name changes and the compiler will catch errors caused by columns or tables being removed. Also, this way table and column names are used in a “strongly-typed” way to prevent spelling mistakes, which future developers can use strings instead of LINQ to SQL metadata references to still take advantage of. The constants are contained in HelperDBTableColumnStrs.cs file under Helper folder. Each batch import will generate a GUID to be inserted with records into database and output to the log files. Row numbers and creation timestamps are
also inserted into the database. Those pieces of information can be helpful if later a batch import is found problematic and needs to be manually revoked.
Chapter 5: Development Lessons Learned

5.1 Gathering Requirements and Feedback

Our development process was iterative and incremental. In each iteration, we focused on a small part of the system. We interviewed customers to get a basic idea, designed and implemented a prototype, met with customers again to get feedback, then went back to re-factor and make adjustments to the system.

Gathering requirements and feedback plays an important role in the process. Here are the lessons we have learned:

1. Use prototypes to elicit requirement details.
   Customers normally only have vague conceptions of what they want. As a result, it is next to impossible to get a complete set of requirements in the first interview. A prototype can be a big help to elicit further details. After customers see a prototype, they get a better idea of their own requirements and a better feel of how technologies can help them, both of which make the interviews easier.

2. Research unusual data in existing systems to discover rainy day scenarios.
   Customers often focus on sunny day scenarios when they are interviewed and so most of the time it is up to developers to discover rainy day scenarios. The unusual data in existing systems that have been forgotten by customers often reveal hidden exceptional case requirements.

3. Focus on what the data are before thinking about how the data are inputted and outputted.
   Customers usually focus on the user interfaces of data input and output. However, developers need to first focus on designing a robust and flexible data model that serves as the basis of user interfaces. As a result, developers should try to lead customers to first answer and think about what the data are.
4. For a process that involves multiple parties, make sure to interview customers of all roles. Customers normally only consider the requirements of their own roles. It is very important for developers to interview customers of all roles in order to get a big picture of the process, possible conflicts and responsibility distribution between various roles. Developers should also present those pieces of information to customers that have overlooked them to get further feedback. If a system is developed without fully understanding the entire process, future changes can be much more difficult to deal with than the pressures from anxious customers pressing for a working system.

5.2 Protecting Data

When working with data, developers need to be very careful because if data are contaminated, it can be very difficult to restore them to correct states, especially in a working system (a production system or a development system whose data will carry over to a production system). Here are the lessons that we have learned:

1. Always test code with a shadow copy database first.
   
   This not only applies to application code but also database scripts. The overhead of maintaining the structures of 2 databases identical maybe nontrivial but it is certainly worth it to keep the data clean. Also, this forces developers to think about testability when writing code and result in code that is data location ignorant, which is easier to be migrated between environments.

2. Make sure data can be traced back to original data after migration or transformation.
   
   This often means storing additional information about a migration or transformation. For example, for each record that is migrated or transformed from original data, an identifier of the migration or transformation, the original data source, the primary key of it in the original table or the row number of it in the original file can be stored. If a previous migration or transformation is incomplete or wrong and the data need to be fixed, those pieces of information may be the only ones that can be relied on to link to original data accurately from a working system.
3. Make data snapshots before performing dangerous tasks.

The extra stored information mentioned in the above point would be meaningless if developers do not have a correct original data snapshot to start with. Data snapshots can also be used to prevent human errors such as accidental deletion of a table. It is recommended that data snapshots being taken before the following tasks: schema changes, running code that directly modifies data on working systems even after it is tested on a development one, data migration or transformation.

5.3 Migrating a System to a Production Environment

The lessons we have learned from migrating a system to a production environment are:

1. Set up the development environment as close as possible to the production one before writing code.

   The first advantage is that code requires minimal changes to work during the migration. It also forces developers to deal with higher risk issues such as architecture and security model first. Last, the production environment will not be in control of developers. The development environment can be used to experiment various configurations, the experience gained from which can then be used to make documentation and train administrators.

2. Involve administrator early in the development process.

   The role of administrators is very important since administrators are the ones who control the production system environment and maintain it. It is critical to involve them early in the development process for security and operation advices since they may impact how system should be developed.
Chapter 6: Conclusion and Future Work

6.1 Conclusion

In this thesis, we described how we have used Sharepoint as a host application of an ASP.NET application and SSRS reports to integrate their authentication, authorization and user interfaces. Most importantly, we explained:

1. The influence of Sharepoint on the whole architecture;
2. The security considerations and configurations of Sharepoint;
3. The development practices and design choices of an ASP.NET application that is developed inside Sharepoint.

We explained how we have used LINQ to SQL as the main data access technology. Most importantly, we explained:

1. The possibility and limitation of object relational mapping in LINQ to SQL;
2. How to write reusable and dynamic LINQ to SQL queries and caveats;
3. The correct way to perform LINQ to SQL write operations in a disconnected environment.

We also summarized the development lessons learned in developing ACE.

6.2 Future Work

Software development is endless. For any software system, there is always a next iteration to make it better. Here are the goals for future iterations:

1. Use ASP.NET AJAX library [76] to provide better user experience.
2. Re-design and re-implement ACE Web App with ASP.NET MVC Framework [77] whose paradigm leads to better separation of concerns.

3. Re-factor the data access layer of ACE Web App into separate data access web services that can be reused by other subsystems.

4. Make adjustments to the system based on users’ feedback, fix bugs and improve the documentation.
Bibliography


[48] “LINQ Inheritance Error When Attributes of Child Classes Are Mapped to Computed Columns”, http://social.msdn.microsoft.com/Forums/en-US/linqtosql/thread/5b1463e4-baca-4d0e-8dd5-b3a95a04bd53


[74] Eric T Freeman, Elisabeth Robson, Bert Bates and Kathy Sierra, Head First Design Patterns, O'Reilly Media, 2004, p. 410

[75] Eric T Freeman, Elisabeth Robson, Bert Bates and Kathy Sierra, Head First Design Patterns, O'Reilly Media, 2004, p. 156
