Regis University
ePublications at Regis University

All Regis University Theses

Summer 2006

An Access Control Middleware Application

Gary W. Withrow
Regis University

Follow this and additional works at: https://epublications.regis.edu/theses

Part of the Computer Sciences Commons

Recommended Citation
Withrow, Gary W., "An Access Control Middleware Application" (2006). *All Regis University Theses*. 412.
https://epublications.regis.edu/theses/412

This Thesis - Open Access is brought to you for free and open access by ePublications at Regis University. It has been accepted for inclusion in All Regis University Theses by an authorized administrator of ePublications at Regis University. For more information, please contact epublications@regis.edu.

Regis University
School for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

Use of the materials available in the Regis University Thesis Collection ("Collection") is limited and restricted to those users who agree to comply with the following terms of use. Regis University reserves the right to deny access to the Collection to any person who violates these terms of use or who seeks to or does alter, avoid or supersede the functional conditions, restrictions and limitations of the Collection.

The site may be used only for lawful purposes. The user is solely responsible for knowing and adhering to any and all applicable laws, rules, and regulations relating or pertaining to use of the Collection.

All content in this Collection is owned by and subject to the exclusive control of Regis University and the authors of the materials. It is available only for research purposes and may not be used in violation of copyright laws or for unlawful purposes. The materials may not be downloaded in whole or in part without permission of the copyright holder or as otherwise authorized in the "fair use" standards of the U.S. copyright laws and regulations.

REGIS UNIVERSITY

SCHOOL FOR PROFESSIONAL STUDIES

MASTER OF SCIENCE
IN
COMPUTER INFORMATION SYSTEMS

An Access Control Middleware Application

Gary W. Withrow

June 12, 2006

Project Paper Revision/Change History Tracking

Version	Submitted To	Date	Changes
Original	Michael Busch	04/28/2006	Original submission
Revision #1	Cory Graham Michael Busch	05/14/2006	
Revision #2	Cory Graham Michael Busch	05/21/2006	Grammar
Revision #3	Cory Graham Michael Busch	05/27/2006	Removed bulleted lists Corrected references
Final Version	Cory Graham Michael Busch	06/12/2006	

Abstract

Information in any modern organization is a very important topic. A company's information is arguably the single most important asset a company owns. Loss or compromise of the corporate information assets can lead to serious financial impact on a company's bottom line. Currently most corporate information is stored on network storage devices. These storage devices provide quick and easy access to the information from anywhere in the world. These same storage devices can also expose the information to its greatest vulnerability, attack by a hostile entity. The current network security best practice calls for a strategy named 'Defense in Depth'. This strategy uses a series of defensive layers to secure the network and the data it contains. There is a realization that no single defensive technology is one hundred percent effective. Samples of external looking defenses include firewalls, anti-virus gateways, proxy servers, virtual private networks (VPN), and complex passwords. The design of these protective measures serves to protect the network from attack by parties outside of the local area network. In additional to the external defenses, there are also internal defense mechanisms as well. These include locking the server room door, complex passwords, file encryption, network access restrictions, and keeping the user database up to date.

One often overlooked technology when designing the network security system is physical access to the company's facilities. The goal of physical access control is to manage who goes where within an organization and when they go there. In addition, a physical access control system can provide physical intrusion detection and notification to the appropriate security personnel. If a specific

individual is not within the facility, he/she should not be attempting to log in to the network.

This project developed and demonstrated a non-typical approach to the management architecture for a physical Access Control System (ACS). It examines the minimum set of requirements necessary to manage an access control system as well as focuses on using a user interface (UI) that a network administrator is familiar with. It is felt that removing the “unknown and complex” interface normally associated with physical access control software, companies will be more willing to add this additional layer of defense to their network security design.

The project utilizes Microsoft® Active Directory (AD) as the primary user interface. It also utilizes the Windows® event logging service to provide the user with event and alarm messages in a human readable format. A data store consisting of Microsoft SQL Server database dedicated to the management of the hardware sub-system.

Table of Contents

1 Introduction **1**

1.1 Problem Statement 1

1.1.1. Access Control System (ACS) Overview 2

1.1.2. ACS and the Data Defense Strategy 5

1.2 Review of Existing Situation 5

1.3 How will this Project Address the Existing Situation 7

1.4 Goals of the Project 8

1.5 Barriers and/or Issues 8

1.6 Scope of the Project 9

2 Project Research Methodology **10**

2.1 Research Methods Used 10

3 Project Methodology **11**

3.1 System Development Life-Cycle Model Followed 11

3.2 Conceptual Design Phase 11

3.3 Project Implementation Issues 14

3.3.1. Modifications to Existing User Interfaces 15

3.3.2. Microsoft Active Directory 15

3.3.3. Windows Event Log Service 16

3.3.4. Middleware Application Design Architecture 16

3.3.5. Design Architecture 17

3.3.6. Implementation Language 19

3.3.7. C++ 21

3.3.8. Java 21

3.3.9. C# 22

3.3.10. Implementation Language Summary 23

3.3.11. Implementation Methodology 23

3.3.12. Persistent Data Storage 24

3.3.13. Microsoft Active Directory 25

3.3.14. Flat Files 26

3.3.15. Database 26

3.3.16. Persistent Data Storage Summary 27

3.3.17. Design Tools 27

3.4 Summary of the Project Implementation Issue Decisions 31

4 Project Detailed Design and Implementation **32**

4.1 Introduction 32

4.2 Project Detailed Design Phase 33

4.2.1. User Interface Layer 33

4.2.1.1 Overview 33

4.2.1.2 Users and User Groups 34

4.2.1.3 Doors and Door Groups 34

4.2.1.4 Schedules 34

4.2.1.5 Event Reporting 35

4.2.2. Business Logic Layer 36

4.2.3. Communication Services Layer 39

4.2.3.1 VertX Hardware System Interface 40

4.2.3.2 Microsoft Active Directory Interface 41

4.2.3.3 Windows Event Log Interface 42
4.2.3.4 XML Message Consumer Interface........ 42
4.2.3.5 Database Interface........ 43
4.2.4. Data Objects Layer 44
4.2.4.1 Card Holder Object 45
4.2.4.2 Door Object........ 46
4.2.4.3 Schedule Object........ 46
4.2.4.4 Door Group Object........ 47
4.2.4.5 Access Level Object 47
4.2.4.6 Access Group Object 48
4.2.4.7 Event Objects........ 48
4.2.4.8 Hardware Configuration Object........ 49
4.3 Detailed Database Design 49
4.3.1. Database Schema Design Process 49
4.3.2. Factory Default Data 50

5 Conclusions 51
5.1 Did the Project Meet Initial Expectations 51
5.2 Lessons Learned 51
5.2.1. N-Tier Architecture........ 52
5.2.2. Database Design 53
5.2.3. C# 54
5.2.4. Design Patterns 54
5.3 What Would I Have Done Differently 54
5.4 Future Application Development........ 55
5.4.1. Where the Project Can Go........ 55
5.4.2. Active Directory Extensions 55
5.4.3. Control Requirement........ 56
5.4.4. XML Message Service 57
5.4.5. Database Extensions 58

Table of Figures

Figure 1 - Conceptual Design 12
Figure 2 – VBScript to Add Card Number to an Active Directory User Object 16
Figure 3 - High Level Design 19
Figure 4 - Event Message Sequence Diagram 29
Figure 5 - Application Use Cases 32
Figure 6 - Business Logic Layer Classes 37
Figure 7 – Communication Service Layer Class Relationships 39
Figure 8 - Data Layer Class Relationships 45

1 Introduction

1.1 Problem Statement

Corporate information has immeasurable value to a company. Loss or corruption of this information can lead to serious financial consequences for the stockholders, employees, suppliers, and customers of the company. The need to protect this information has become increasingly apparent to the executive boards of most major corporations. The software industry has developed numerous defense mechanisms to combat information loss. These include firewalls (both software and hardware), anti-virus gateways, patch management applications, file encryption techniques, and data backup solutions. No single defensive measure is one hundred percent impenetrable. The SANS institute has recommended a "Defense in Depth" strategy when designing and implementing the defense of corporate information assets to mitigate the risks to corporate data (2005). To accomplish this goal, the IT professional places a series of defensive mechanisms around the company's data storage and networks. The attacker must breach all of the layers before the information is vulnerable. Each additional layer provides an additional barrier to the information. However, each additional layer incurs costs associated with the purchase, implementation, and maintenance of the product. In addition, each layer can add additional bandwidth requirements to the network that will influence the access speed of the data.

A layer of defense that is often overlooked, particularly in small businesses is an Access Control System (ACS). The primary function of the ACS restricts the access of individuals to the facility and therefore the physical data network. The IT professional uses this additional layer of protection to limit potential periods of vulnerability. However, typical ACS applications are complex and the user must address many details to keep the system effective.

Regular administration of the typical ACS package is often difficult. To an over worked IT professional it is another job that may only get erratic attention or be setup so everyone has access to all the areas of the company at any time. This is not a very effective policy.

1.1.1. Access Control System (ACS) Overview

Who goes **Where** and **When**!

These words sum up the purpose of any access control system (ACS). To accomplish its task successfully, an ACS usually has a physical component and a software component. The design of these two components is typically uses a proprietary protocol. Therefore, they will only work with each other.

The physical component of the ACS requires that control of each door through some mechanism, usually electro-mechanical locks on doors, and/or door position monitoring hardware. The next requirement of the ACS is the identification of each user to the system. The accomplishment of this requirement is through some mechanism, usually a card, keypad, or biometric reader.

The establishment of the identity of the user is with a token (typically a card or key fob), a Personal Identification Number (PIN), or a biometric identifier (iris, fingerprint, or hand geometry). Each mode of identification will result in an identification number that is unique to each user. The ACS evaluates the access privileges of the user based on the following criteria: the assigned privileges, the door, and the time of the presentation. Either the door is unlocked to allow the user to enter or the door remains locked based on the result of the evaluation of this set of criteria. The software component receives a report is made of the result of the access decision through a messaging mechanism from the hardware. The software component displays these messages to the system administrator.

Provisioning the system provides effective oversight and manageability. The ACS software component organizes users and doors into groups and then assigns the user privileges based on a schedule. Provisioning typically occurs in two distinct phases. First, is the initial provisioning occurs when the hardware is first in the facility. The second phase of provisioning is an ongoing process throughout the life of the ACS system.

Initial provisioning includes the assignment of a physical address and a name to each door in the system. It may also include setting of specific device attributes in the system. There are both software and hardware attributes. An example of a software attribute is the address of the host PC that will be controlling or monitoring the system. An example of a hardware attribute might be the type of switch installed on a specific door. Provisioning a typical access control system will include several hundred of these attributes.

The ongoing provisioning includes updating the card database when adding a user to the system, removing a user from the system, or changing his/her privileges in the system. These events can occur rarely or on a daily basis depending on the size and complexity of the system and the granularity of the control established by the system administrator. The design of the ACS software component must provide an efficient interface in either extreme. Often systems become overly complex in order to handle all of the potential scenarios that a system must address.

In addition to providing oversight and provisioning, the ACS software component must also provide methods for controlling various functions and devices within the system (H. Knight, personal communication January 25, 2006).

A well-planned ACS system provides significant benefits to an organization. First, it can provide convenience to users by allowing access without requiring them to remove a key from a purse or pocket. At the same time, it will secure the facility reducing the possibility of unwanted eyes wandering into a sensitive area. Next, it can provide an easy to verify form of identification of employees, contractors or guests by simply wearing the badge on the outside of the users clothing. Next, it can save the property owner money by reducing the cost of managing physical door locks and keys in a facility. Lost keys can add to up significant cost and windows of vulnerability to a company. Next, it can provide evidence of entry and exit through a log of user activity, showing when and where a specific user has been as well as who used which door and when.

However, ACS systems also have some drawbacks, these include a single vendor providing both the hardware and software components. Typically, these systems include a proprietary interface between the hardware and software. This has locked the end user into a single vendor for life of the entire access control system. If either the hardware or software becomes outdated, the owner must replace the entire system. This also restricts the functionality available to that set of functionality the system vendor has provided in the system.

Finally, the ACS software component may contain 20 or more screens with numerous sub screens, presenting a very complex interface to the system administrator. Often, the system administrator may only need to perform some limited subset of the system functionality to achieve effect control. The system complexity and steep learning curve inhibit the use of the system.

1.1.2. ACS and the Data Defense Strategy

Network connections are typically not accessible outside of a building. Controlling the authorized access to a building limits the vulnerability of the corporate network to attack from the inside. The attacker must first gain physical access to the building and then access to a network connection.

1.2 Review of Existing Situation

ACS software is a very complex application. Most ACS applications have hundreds of configurable parameters. This configurability leads to a complex user interface consisting of as many as twenty screens with numerous options of each

screen. Often, the administrator may only need to perform some limited subset of the overall system functionality to achieve effective access control. The system complexity and steep learning curve inhibits the use of the system.

The designs of all of the current access control systems on the market include a high degree of coupling between the hardware system and the software application using proprietary protocols. The current access control purchaser can expect to use the same system for ten years before replacing it. As new ACS requirements are recognized, the end user must modify their system to address each new requirement. This assumes the system provider has also modified their system to address the new requirements. There is often a delay in this development effort while the vendor evaluates the market to ensure there is a wide enough demand for the new functionality to justify the cost of development and deployment. This delay exposes the user to some period of increased vulnerability that the new functionality would address.

Typically, access control vendors tend to market their products to several different market segments. These segments are divided by the feature set each segment has determined are required to meet its specific access control needs. These segments span the range from the centrally managed enterprise wide installation to a single office installation managed by an off-site contracted service. The ACS vendors include the complete suite of functionality even if the target market segment does not require some features. This increases the complexity of the software component of the system.

1.3 How will this Project Address the Existing Situation

This project will demonstrate that a software vendor could write an access control application with a familiar set of User Interface (UI) screens. The hardware sub-system used in this project includes several hundred configurable parameters. The system designers included this configurability to allow the adoption of the hardware by as many different ACS vendors as possible. This configurability includes a serious drawback; the system can be very complex to initially setup. This configurability has allowed the adoption of this hardware platform by software development partners that target their products to all segments of the ACS market. This project addresses this issue by focusing on a single potential market segment. With this restriction, the number of configurable parameters that the installer or user needs to change is significantly reduced.

In addition, the design of the hardware's software interface is an "open" application programming interface (API). The corporate policy is to make the API available to any software development partner that wishes to implement a software product that will interface with the hardware. The software vendor does not need to sell the hardware to qualify as a software partner. The intent is to provide as much flexibility to the end user of the hardware as possible. The implication is that a customer could change software vendors without changing the field hardware. A situation that is not currently available with the current access control products.

1.4 Goals of the Project

The goal of this project is develop a “familiar” user interface to an access control system. Familiar means a minimum number of configuration screens and familiar interface designs to provide the basic required functionality. The project will use existing user interfaces wherever possible. If more than one option exists, a preference will be toward an interface that is familiar to a typical Windows® information technology (IT) professional.

There are several basic requirements for an access control system. First, the user must be able to enter new users or change existing users in the system. Second, the user must be able to monitor system activity. Which card user accessed which door and when as well as the ability to monitor basic system performance. Finally, the user should be able to add new schedules and very rarely new doors to the system.

1.5 Barriers and/or Issues

A significant barrier to the successful completion of the project is to identify an interface that will handle the users and doors of the company facilities. This interface must be able to export new or changed data to another application in a timely manner and in a standard format. In addition, this interface must be extendable to include the identification number associated with each user and handle the doors and door groups that will be required.

There must also be a mechanism for reporting events to the user. These events must be a human readable format, not cryptic numbers usually associated with ACS messages. The IT administrator must be able to review these messages when it is required. The system must provide an archival feature for these messages for some time before deleting the message from the system.

A basis of acceptable system performance is the perceived speed at which a user can gain access after adding the user to the system. The application must meet the following arbitrary time constraint. A new user must be able to gain access to a door within five seconds after adding the user to the system.

1.6 Scope of the Project

The scope of this project is limited to a demonstration of the key concepts required to implement an access control application. These include the ability to manage a cardholder and view events generated by the access control system. These two features are two of the three primary functions a typical user will need to perform on a regular basis (H. Knight, personal communication January 25, 2006). The first phase will not address the control functionality requirement; this discussion of this requirement will be in the conclusion section.

The intention of this phase of the project is not to be a marketable version of the application, nor is it to be a complete access control application. The author acknowledges that there is functionality missing from this version. Addressing this functionality must occur before the application would be a marketable product.

2 Project Research Methodology

2.1 Research Methods Used

The hardware manufacturer conducted a series of investigations into the currently available access control hardware and software systems. These investigations consisted of reviews of company web sites, viewing demonstration software and software demonstrations at trade shows. The company collected and collated the information as market research for a new product development effort. The company that funded the research effort views the information as confidential so no further discussion of the specific information is included in this document.

Because of the research however, it became apparent that no current ACS vendor targeted the very small system user. These users would have one or two doors that needed control as well as a very small number of cardholders (less than 100). The cost to the vendor of support for these users was a significant factor in their decision not to address this segment of the market. The complexity of the software application was the leading cause of the cost factor. The user would need to call the support service due to infrequent use of the system. If the vendor charged enough revenue to cover the cost of this service, the market price of the product would be too high.

3 Project Methodology

3.1 System Development Life-Cycle Model Followed

This project development methodology was a combination of the waterfall and iterative software design methodologies. The team divided the project into the following phases: conceptual design, detailed design, implementation, and then evaluation. Within each of these phases, the team applied an iterative approach to the tasks. The team would perform some amount of work and then review the work before the performance of additional work. This methodology subdivided the application into manageable pieces for a resource limited development team. The following sections will address the different project phases.

3.2 Conceptual Design Phase

The original idea for the project developed from several years of development work on a new hardware platform offering by HID Corporation. It became obvious that the HID product would not gain acceptance in the market until the software application developers wrote software that addressed this new platform. The software application providers had made considerable investment in the current hardware platforms and the cost to migrate was generally prohibitive. From this came the idea of attempting to use any existing interface mechanisms instead of writing new ones.

The original conception of the application was similar to a "middleware" application. Middleware applications typically connect two disparate systems. These could be applications running on two different hardware platforms or two different applications running on the same hardware platform. The middleware application performs some type of translation to the data as it moves between the two systems. This application fits that conceptual model.

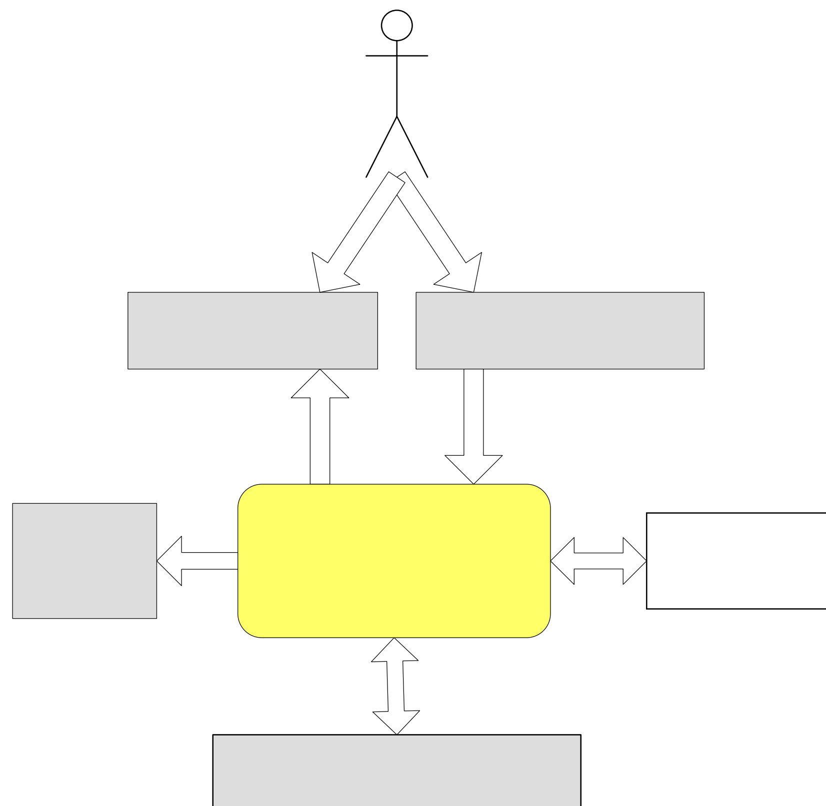


Figure 1 - Conceptual Design

At this point, a search began for the potential interface applications.

For the small business, there are two operating systems on the market, Windows and Linux. Although Linux may have several advantages over Windows servers, in 2005 Windows is still the clear choice for most businesses (DiDio). For several years, Microsoft has striven to present one look and feel to

the computer user. This is most evident in the Microsoft Office suite of applications. Microsoft has also encouraged third party application developers to adopt a similar look and feel to their applications. The model used to develop the controls in Visual Basic promotes this similar look and feel. The first time a user interacts with a new application, the familiar look and feel reduces the users' learning curve. This philosophy has served Microsoft well as indicated by their market share numbers. The “familiar look and feel” was a significant factor during the search for user interface mechanisms.

A review of the Microsoft literature indicated that Active Directory could be “extended” to add new attributes to the object definitions (n.d.). Active Directory already had the concept of users and resources (computers and printers) as well as groups of users and resources. An access control system’s software component uses the same concepts. The team decided to attempt to extend Active Directory to meet the needs of the project.

The team next addressed the mechanism to view events. Two events aided in the decision to use the existing Windows event log service. First, a previous project had used the Windows event log service to record application events and present to the user without the application itself providing this function. It proved to be quick to implement and easy to use. Second, while observing the network administrator troubleshoot a problem he first used the same event service to review the messages from the suspect application. A brief discussion revealed that he considered it a primary tool he used to administer the network. These two things made the decision to use the Windows event log

service an obvious one.

These two decisions addressed two of the three access control software features. The third, control of the system, was not included in the phase of the project. The section of this paper titled “Future Application Development” addresses this item. At this point, it was time to consider the detailed design of the application.

3.3 Project Implementation Issues

Before the project team could perform any of the detailed design work, the team had to address several issues. First, there were several issues related to the detailed design that needed to be resolved. The team must address several issues before the implementation phase could begin.

The first set of issues that the team needed to address were, what changes to Active Directory and the Windows event log service would be required. Next, the team needed to address the implementation phase issues; these included the design architecture, programming language, and implementation methodology. Finally, the team needed to address the data storage issue.

The detailed design broke the project into two distinct areas of development. First was the design of any modifications to either of the two existing UIs that were chosen and the second was the design of the middleware application that would tie the existing UIs to the hardware sub-system.

3.3.1. Modifications to Existing User Interfaces

The decision to use AD and the Windows event log service required an analysis of each of them. The analysis focused on what, if any, modifications each UI required before it was usable for the new purpose. The team examined each of the UI mechanisms separately and a description of the results of these examinations is in the next two of sections.

3.3.2. Microsoft Active Directory

Research revealed that is was straightforward to modify Microsoft's directory service, Active Directory, to add additional attributes to the existing objects. This addressed the cardholder management requirement of the project. There are three different techniques to make these modifications. Robbie Allen lists the following three methods; first, is with a graphical user interface (GUI) such as ADSI Edit, second, with a command line interface (CLI) such as the ds utilities and, third through a scripting language such as VBScript (2003). The design team decided that the scripting technique offered the solution that could be migrated to a real product the easiest. The programmer created a series of scripts that added the needed attributes to the user object in AD. For this project, the script added a single attribute (Employee Card Number) to AD to demonstrate the concept. Shown in Figure 2 is an example of the type of script that the programmer created to add the attribute to the user object.

```
Dim oemployeeCardNumber
Dim oUser3
Dim temp3
Set oemployeeCardNumber = Wscript.Arguments
Set oUser3 = GetObject(oemployeeCardNumber(0))
temp3 = InputBox("Employee-Card Number: " & oUser3.employeeCardNumber & _
             vbCRLF & vbCRLF & "If you would like enter a new number or modify
the existing number," & _
             " enter the new number in the textbox below")
if temp3 <> "" then oUser3.Put "employeeCardNumber",temp3
oUser3.SetInfo
Set oUser3 = Nothing
Set oemployeeCardNumber = Nothing
Set temp3 = Nothing
WScript.Quit
```

Figure 2 – VBScript to Add Card Number to an Active Directory User Object

3.3.3. Windows Event Log Service

The use of the Windows event log service did not require any modification. The team decided to add two additional log files to separate out the routine event messages and any events created because of the database resource. This would serve to reduce the time the administrator would take to review the ACS events. The programmer wrote two scripts in VBScript to create these two additional files.

3.3.4. Middleware Application Design Architecture

The detailed design of the middleware application required several issues be addressed, first the design architecture, second the language used to implement the design and third the design implementation philosophy. These three issues are common to any software development effort. In some organizations, the design team makes a conscious decision about each issue. In other organizations, the design team never specifically addresses these issues.

This design team made the decision to address each issue after a thoughtful and thorough research effort. A discussion of each of issue and the decision made is in the following sections.

3.3.5. Design Architecture

The team examined two architecture options. First, simply no architecture and second the N-tier architecture. The author had been involved in several software development efforts that resulted in applications consisting of 50,000 plus lines of code. The development teams delivered each of these projects over budget, beyond the expected schedule, and in one case with a reduced feature set. The author examined each project and realized that there had never been any thought given to the architecture used in the design. The design was simply the result of the implementation. Next, the team considered the N-tier architecture.

Using an N-Tier design approach has several significant advantages according to Martin Fowler who states:

- “You can understand a single layer as a coherent whole without knowing much about the other layers.
- You can substitute layers with alternate implementations of the same basic services.
- You minimize dependencies between layers. Layers make good places for standardization.

- Once a layer is built it can be used for many higher-level services." (2003)

The design team considered these stated advantages, several small applications that were designed with the N-tier architecture and the lessons learned from the previous application development efforts the author was involved in and decided that utilization of the N-tier architecture would benefit the project and the application design.

The design team subdivided the project into four significant layers, first the user interface, second the business logic required, third the communication services and finally the data objects required. Next, the team divided each layer into the classes of objects that would implement each of the required features of the system. Figure 3 - High Level Design reflects the layered architecture of the application.

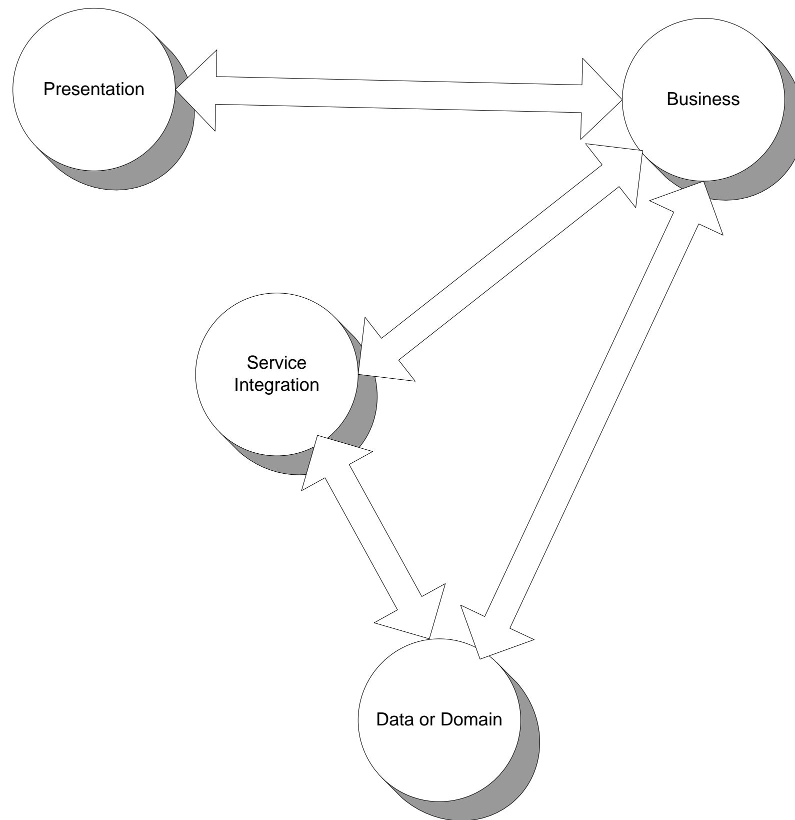


Figure 3 - High Level Design

3.3.6. Implementation Language

The next issue the team addressed was the programming language that would be used during the implementation phase of the project. Programming languages abound, certainly tens and probably hundreds of languages have been developed and promoted in the industry throughout the years. The design of many of these languages focused on very specific programming situations while the target of others was as general-purpose languages. Many of these languages would have been suitable for the implementation of this application. However, the application should be robust, once completed. Rob Sjodin, a professor in the Regis MSCIT program, described robustness as “a measure of how well the

system addresses: availability, scalability, maintainability, adaptability, extensibility, interoperability, understandability, usability/operability, reliability, manageability and securability" (July & August 2005). The team considered each of these attributes when it chose a programming language.

The language chosen will influence the robustness of the finished application in several ways. First, the language needs to include all of the required external interfaces, to a database, Active Directory and the event log service. Second, because of the required external interfaces, it needs to provide error handling and recovery. If these interfaces and error handling did not exist natively in the language, the implementation team would need to implement them as part of this project. This increased the coding effort and the risk of the project. Finally, the language needs to promote a structured implementation.

The first consideration was whether to use a procedural language such as Cobol, C or even Basic or an object-oriented (OO) language such as C++, Java® or C#® (pronounced C Sharp). The team discarded a procedural language because of their inherent lack of structure. An object-oriented (OO) programming language was preferred for the following reasons: encapsulation, polymorphism and inheritance, the three cornerstones of an OO language (Dr. D. Hart, personal communication, September 2005). An application implemented with these attributes will tend to be more robust than one that is not. The implementation of any of these languages can be a very non-OO style if the developer does not use the inherent features of the language. Regular reviews of the code with specific

attention on the structure of the implemented code ensured that the team adhered to OO model. The team examined each of these languages for its suitability for this application and a discussion of each one is in the following sections.

3.3.7. C++

C++ was one of the early object-oriented languages. It included the required interface mechanisms and error handling. C++ is an extension to the C language with the added OO constructs. Because of this heritage, C++ includes the ability to allocate and use memory directly, with it the responsibility to release the memory when the application is finished using it. C++ also includes the ability to use pointers and pointer arithmetic. These provide a very powerful mechanism in the appropriate situation. However, it is very easy to abuse the power of pointers and system crashes are the usual result. The team decided that neither of these two features would be necessary for this application.

3.3.8. Java

Java does not include the memory handling requirements or the ability to use pointers, as does C++. It is an OO language with a very structured class hierarchy and it included all of the required interface mechanisms and error or exception handling. One requirement of Java is to download and install the Java Virtual Machine (JVM) from Sun Microsystems. The JVM is freely available and there are versions that will run under the Windows operating system. Java will also run on almost every other operating system currently in use. This project targeted a Windows operating system. Therefore, the cross platform functionality

was not a requirement. In all other respects, Java would have been a suitable choice.

3.3.9. C#

Microsoft, as part of their new programming paradigm, recently introduced the .NET framework and C# as a new programming language. The intent of the .NET framework is to isolate one application from all of the others and very closely monitor the use of system resources by the application. Microsoft refers to this approach as “managed code” (Abrams, 2004). C# is fully an object-oriented programming language with very strict class hierarchy and many Microsoft implemented support classes (Robinson et al, 2003). It relies on garbage collection to clean up unused memory, removing the responsibility from the developer. The managed code aspect addresses the ability to reference un-initialized memory by the application. Ferracchiati states, “.NET introduces assemblies to replace the traditional DLL and COM components hosted in DLLs or EXEs” (2001). Traditional DLLs have introduced many serious support issues resulting one application’s installation rendering another application inoperable. The industry has referred to this situation as DLL HELL for many years. Microsoft introduced assemblies to address this issue. The team judged that these were positives for the choice of C#. However, C# is a relatively new programming language, introduced only in the last few years. The team judged this as neither a positive nor a negative.

3.3.10. Implementation Language Summary

After careful consideration of the design goals and system requirements, the team made a decision to use C# as the language used to implement the application. The team could have selected either of the other languages or even one of the many not considered. However, one of the author's personal goals was to learn C#; therefore, the team selected C#. The next consideration was the implementation methodology. The programmers would employ the methodology during the implementation phase and it would have a significant impact on the robustness of the finished application.

3.3.11. Implementation Methodology

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides formalized design patterns as an implementation strategy in a book in the early 1990s. Since they published their book, many other authors have referred to these authors as the "Gang of Four" and have expanded on their original work (Cooper, 2003). Other authors have expanded the original twenty-three patterns addressed by the Gang of Four to include several hundred in the current literature (Cooper, 2003).

Design patterns offer a structured way to address recurring needs in the application development environment. This application presented several of the programming situations directly addressed by design patterns including the abstract factory pattern, the façade pattern, and the singleton pattern.

The implementation of the design uses a series of design patterns. The team did a careful analysis of each class to assess which design pattern best fit its usage and implementation requirements. The implementation of each class followed the identified design pattern. The sections that describe each object in the various layers will also discuss the specific design pattern applied to each class.

3.3.12. Persistent Data Storage

One design consideration was whether to include a persistent data store within the application. One significant design requirement was that the formatting of the event messages be in a human readable form when presented to the user. The message needed to include the name of a cardholder and the name of the door as part of the message presented to the user. The access control hardware has no knowledge of the name of a cardholder or the name of the door, only the cardholder number, or the doors hardware address. To the system user the cardholder number or the hardware address has little meaning. There are numerous examples of this type of data translation required in any access control application. Additionally, each event message would need to be stored for possible future use by the administrator. The system can generate hundreds of events per day.

The VertX hardware platform-provisioning interface is an ASCII text file interface (except two card data files). The host software must push down the entire configuration file as a single unit. There is no mechanism to edit these files,

at least through a host software interface, once they pushed to the hardware platform. This type of interface requires that all the data required to form a particular configuration file must be available at one time within the host application.

The VertX hardware platform has over thirty configuration files that could require configuration. These store information such as the host computer's address, the configuration of the doors and the readers associated with each door. Most of these files are relatively static; however, several require changes as the administrator changes a user's access permissions. Of these files, the schedules and access groups file will need regular updating.

These requirements lead to an investigation of various techniques to store persistent data on the computer. The team considered several different options to meet this data store requirement, this included Active Directory, a series of flat files and a database. The following sections discuss each of the options and the issues associated with each.

3.3.13. Microsoft Active Directory

First, the team considered Active Directory as a possible persistent data store. The design called for AD as the user interface. Microsoft states in the Best Practices discussion that, “The schema is neither a database nor a file system. Don't treat it as such. It is better to place references in the directory that point to other data stores than to use the directory for something for which it was not designed” (n.d.). The design of AD, as a write once and read many type of data

store precludes it from serving as the persistent data store for this application.

3.3.14. Flat Files

The team considered a series of flat files as another possible data store. These offer the simplicity of implementation and ease of use. However, they do not provide any referential integrity by themselves. Referential integrity is an important consideration in the design goals. The implementation team would have to build it into the application code. This added a level of effort and complexity to the application, which the design team deemed to be undesirable.

3.3.15. Database

Finally, the team considered database as the data store. They provide a proven and well-accepted means of storing data and with some of the new functionality offered in modern programming languages they are almost as easy to interface with as flat files. Databases inherently offer the required referential integrity. Some databases are available free, Oracle 10g Personal Edition for example. Others, much as Microsoft Access, are included in a suite of business tools that nearly every company running the Windows operating system has installed. In addition, it is common for a company to have standardized on a specific database product for all of their corporate data storage needs. The design team viewed the flexibility to use a number of different vendor's database product as a positive.

3.3.16. Persistent Data Storage Summary

After careful consideration of the relevant issues, the team decided to store persistent data used by the application in a database. Access to the data is via the database interface class of the communication services layer isolating the database interface from the rest of the code. The database design evolved after an analysis of the data required by both the access control application's user interface as well as the data required by the hardware sub-system. The section titled "Detailed Database design" addresses the specifics of the data base design.

3.3.17. Design Tools

Software systems easily grow to an unmanageable size. In addition, during development, the software application requirements will change. The larger the scope of the application the greater the potential that the application requirements will change before the application finished much less deployed. The team understood these problems and decided that a visualization tool would help manage the project. The tools purpose would be to document and communicate the project requirements to various audiences throughout the project life cycle. There are numerous tools available to help manage the scope and design of a software application. Software vendors target many of these tools at the large enterprise type of project. These tools were not suitable for this project. The Unified Modeling Language (UML) serves a means to effectively communicate architecture and design details of a software development project. The visualization tool chosen for this project was Microsoft Visio.

The team used Visio to create the design documentation used to control the project. These diagrams included use case diagrams, sequence diagrams, class diagrams and entity relationship diagrams. Each type of diagram served to represent a specific aspect of the project.

The first diagram created was a use case diagram. This diagram served to identify the actors that would interact with the system and the processes each actor would require from the system. Each identified unit of functionality or process represented a single use case. The team developed several of the more complex use cases into “fully developed” use cases. The work required to expand each use cases served to identify and qualify the pre-conditions, stakeholders, flow of events, exception conditions, and post-conditions of the system. Figure 5 - Application Use Cases show the simple use case diagram developed during this project.

After the team identified the use cases, the team developed a series of sequence diagrams to document the flow of information to achieve a specific purpose. They served to identify the public methods that would be required by the various objects in the system. The code team used these diagrams during the implementation phase of the project. The following is an example of one of the sequence diagrams that the team developed. It documents the data flow when the application receives an event message from the hardware sub-system.

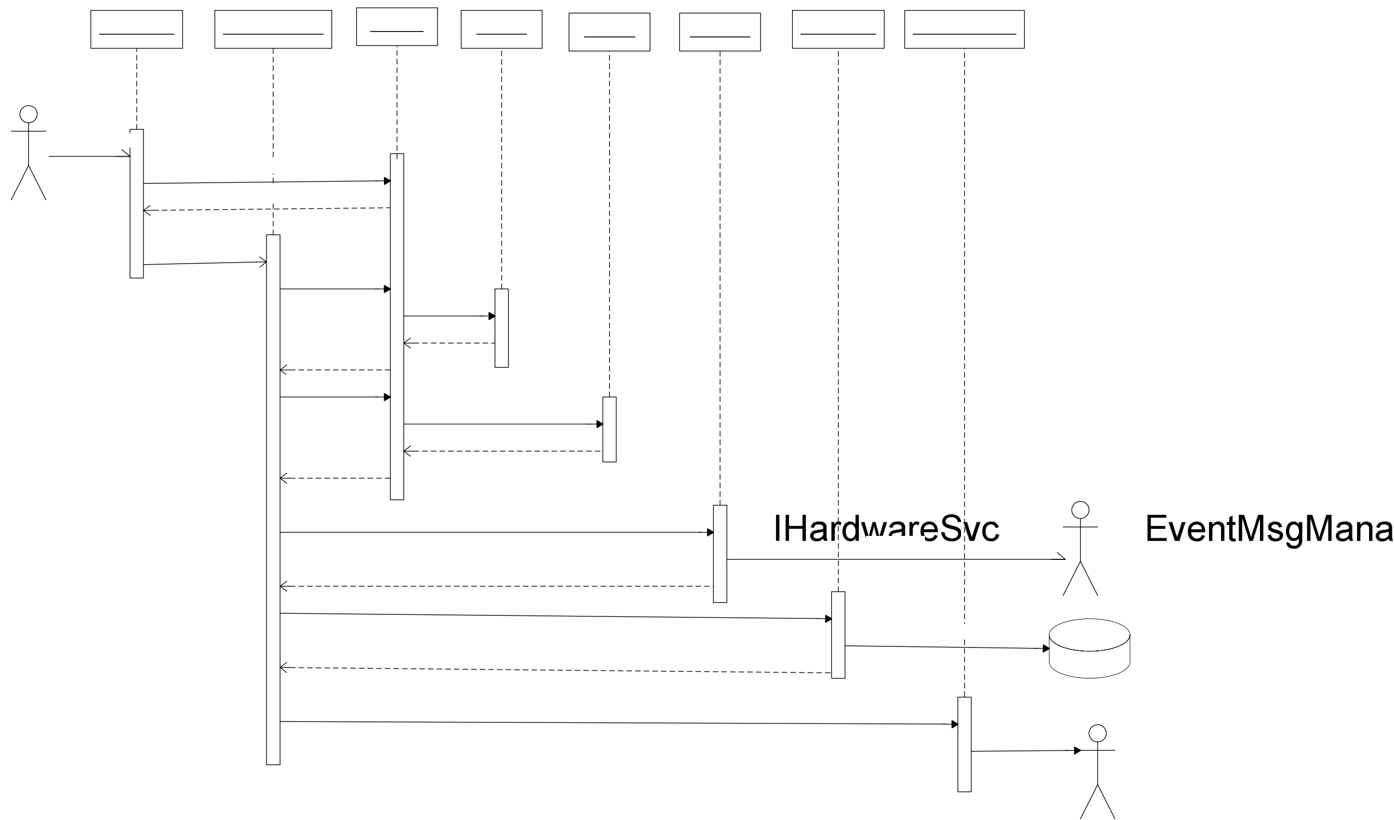


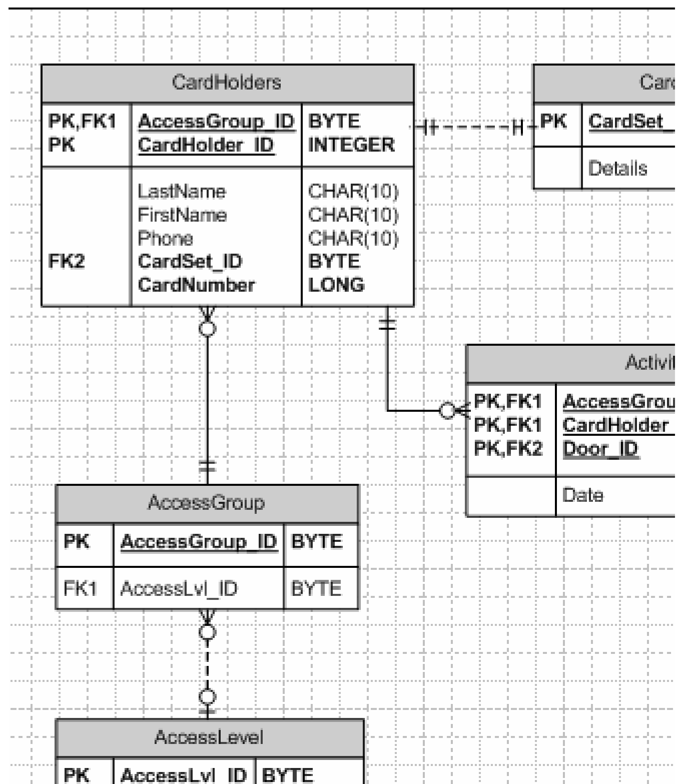
Figure 4 - Event Message Sequence Diagram

The sequence diagram proved to be the most useful to the development team during both the design as well as the implementation phases of the project.

During the design phase, the team developed a series of class diagrams that documented the objects and the relationships between the objects within the system. In addition to the relationships, the class diagrams also served to document the attributes and methods, both public and private, of each object. When the implementation phase started, these were the first diagrams that the implementation team used to code the class definitions. The class diagrams are included in the Project Detailed Design and Implementation section of the paper to

illustrate the classes that compose each layer of the N-tier architecture used in the project.

The final diagram that the team developed was an entity relationship diagram (ERD). The team used the ERD to provide a diagrammatic documentation of the tables, attributes, relationships, and constraints in the database. This shows a small portion of the ERD, the complete diagram is included as Appendix 3 on page **Error! Bookmark not defined.**. Once the team designed the initial ERD, the team normalized the design to remove any data integrity anomalies that might exist. Data anomalies are the cause of referential integrity issues that can result in “bad or corrupt data”. Corrupt data is not only valueless to a company it can actually cause serious mistakes in judgment when used in the decision making process. Once this diagram was developed, the team then converted it into the scripts that created the database tables.



A serious drawback of many of the UML tools is the complexity of the application. The learning curve can be very steep for some of the tools. Visio

proved to be a very straightforward and easy to use visualization tool. It proved to be a good choice to document and design a project of this size.

3.4 Summary of the Project Implementation Issue Decisions

The design team made the following decisions regarding the detailed design and implementation phases would use. First, Microsoft Active Directory and the Windows event logging service would provide the system user interface. The application's detailed design would employ an N-tier architecture approach using design patterns where appropriate. The implementation of the application would use the C# programming language from Microsoft. The application would include a database and the interface to the database is through a dedicated interface class. The hardware interface is via a Dynamically Linked Library (DLL) supplied by the hardware manufacturer and the application would employ a hardware interface class. With these decisions made, the project was ready to move to the detailed design phase.

4 Project Detailed Design and Implementation

4.1 Introduction

One of the artifacts of the conceptual design phase was a series of “Use Cases.” These use cases where the starting points of the detailed design phase of the project.

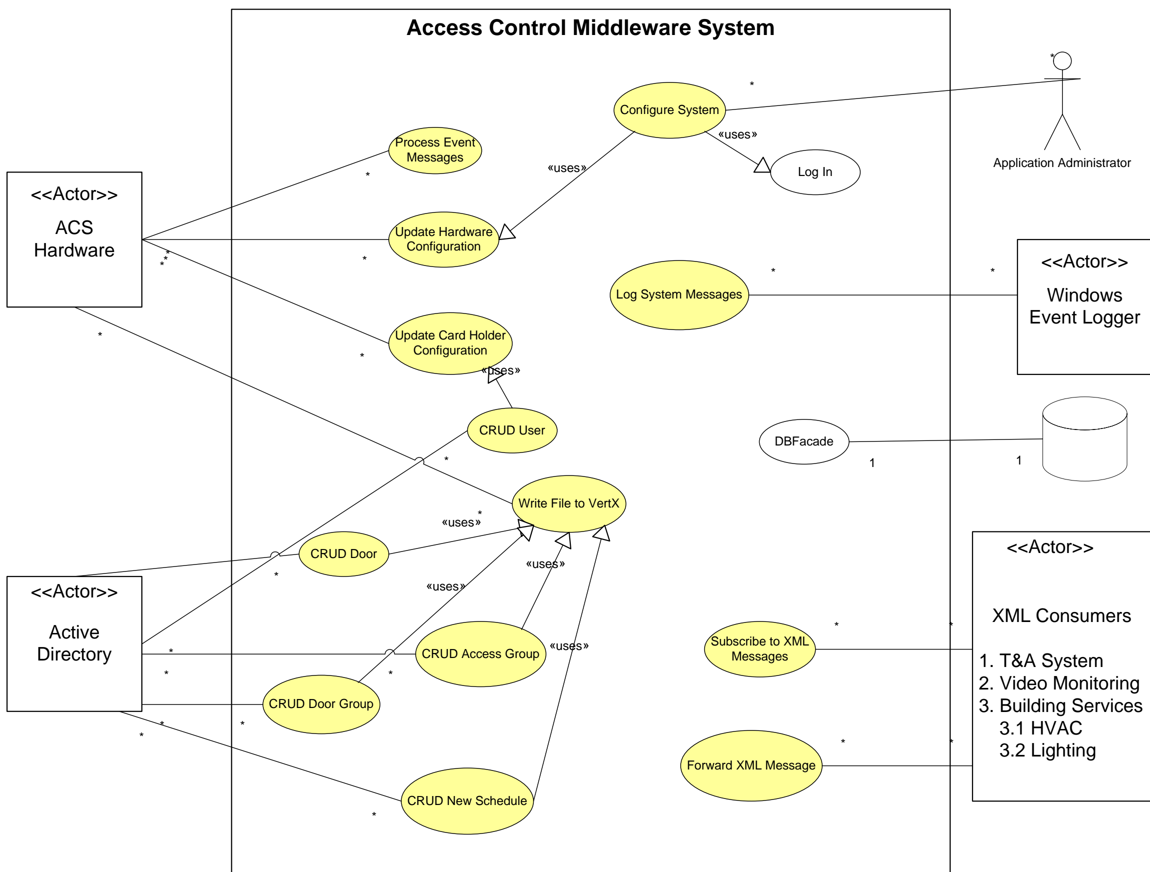


Figure 5 - Application Use Cases

4.2 Project Detailed Design Phase

The output of the design phase was a design composed of classes in distinct layers. The implementation team implemented each set of classes that comprised a layer as a separate scope of work. One of the stated advantages of an N-tier design is the loose coupling between layers. This served to demonstrate this advantage because a different team of programmers could just as well have implemented each layer without affecting the success of the project. The following sections discuss the system architecture that was developed and implemented in the application.

4.2.1. User Interface Layer

4.2.1.1 Overview

The User Interface (UI) of the application provides the interface mechanism between the user and the application. One of the primary design goals was to implement a UI that was “familiar” to a Windows IT professional. During the conceptual design phase, the design team had determined that Microsoft provided two suitable interface mechanisms. Microsoft Active Directory and the Windows event log service. The following sections describe the object concepts that the user would be required to understand.

4.2.1.2 Users and User Groups

The user and user groups in Active Directory correlate to users and groups of users within an access control system. It was an obvious extension to add the card number to the user attributes of Active Directory. The team accomplished this with a script file that interfaced with the Active Directories Service Interface (ADSI). The user concept is the 'who' of an access control decision.

4.2.1.3 Doors and Door Groups

Another central object in an access control system is the concept of a door and a door group. The computer in Active Directory is similar but it was determined that it is not close enough to be extended to represent a door. One of the extensions to the project is to develop a series of scripts or a separate DLL that will create a door object in Active Directory and define the required attributes to reflect the configurable parameters of the door in an access control system. An ACS uses the door concept to represent the 'where' of an access control decision.

4.2.1.4 Schedules

Active Directory included the concept of schedules. Schedules may be associated with users, user groups, and computers within AD. Active Directory exports the schedules to the application for use by the access control system. This presented the user with a familiar concept that is often very complex in the ACS software application. The ACS uses the schedule concept to identify the 'when' of an access control decision.

4.2.1.5 Event Reporting

Network administrators use a built-in Windows event logging service to view application, security, and system activity. The logging service is available under the Control Panel, Administrative Tools, and Event Viewer options. Applications running on the system can write events to the default application log or may extend the event logger to include new log files used by a specific application. As an application logs an event, it classifies them in one of three severity levels, information, warning, or error. As the system presents events to the user, each different type of event includes a different icon. This provides the user with a method to easily to identify the different types of events. Windows event viewer provides the ability to filter the events by event type.

This project extended the event log structure to include two additional log files, the Access Control log, and the ACS-CreateDB log. An access control system generates many events in the course of normal operation as well as numerous events when an abnormal event occurs. The ACS application classifies these different types of events into one of the three pre-defined classes of the event log service. The ACS hardware platform also provides a mechanism to define events into three classifications as well. Three event classifications is the industry standard for event classification. The Windows provided event log service included all the required functions of an event reporting system for an access control system. This produced a nice fit between what the access control industry expects and what this application delivers.

Event messages include normal activity such as a user gaining access through a door, updating the system time or when escorting a visitor through a door. Abnormal activity such as the system denying a user access at a particular door or a door remains open for an extended period. Finally, there are severe events that need immediate attention. These may be an alarm generated by a motion detector or a task in the access control system that restarts. **Error! Reference source not found.**Appendix 2 shows a screen shot of each of the different types of event messages generated by the system.

4.2.2. Business Logic Layer

The business logic statement of an Access Control System can be summarized as the need to “control Who goes Where and When”. The hardware platform utilized in this project contains the required logic to make these decisions. This project provides a mechanism to configure the hardware’s configuration files with the required data and provide a mechanism to update that data when changes are required. Changes may be required when an employee is added or leaves the company or when an employee’s work assignment changes. There may also be changes to the physical layout of the facility that the ACS must reflect.

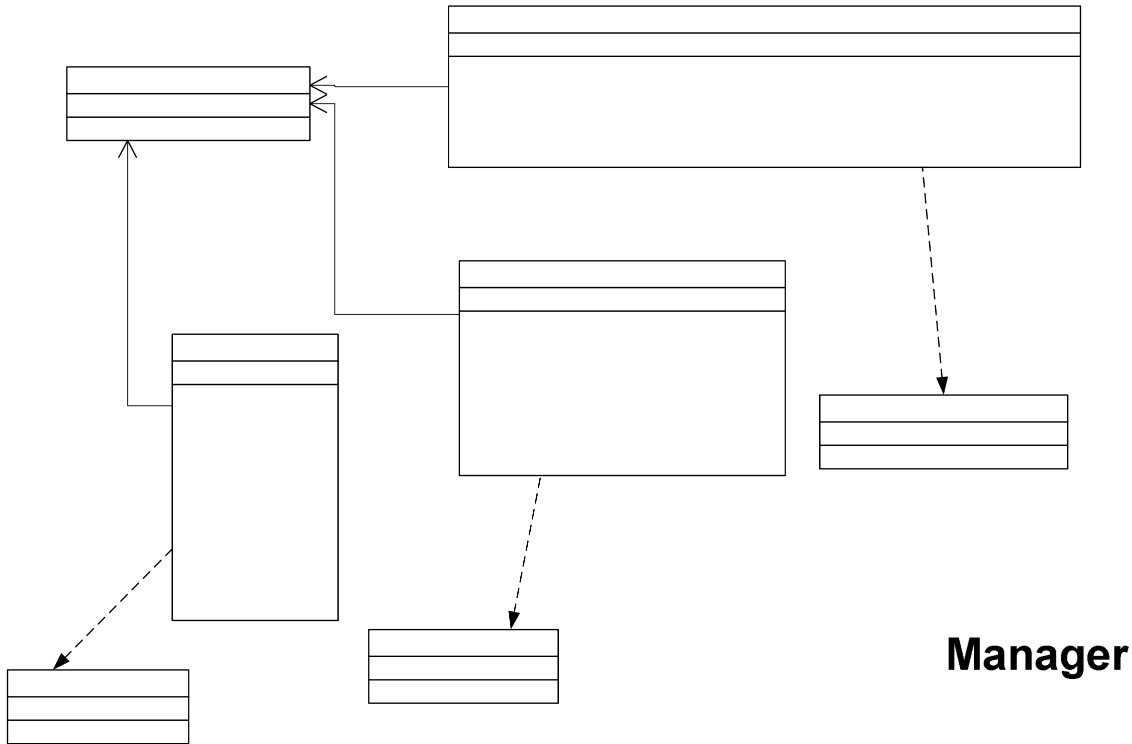


Figure 6 - Business Logic Layer Classes

This project focused on the requirement to manage the changes to the ACS hardware and only issue relevant changes. To accomplish this objective, a database was included in the design. The hardware platform includes a database in its design. It was determined that the time to query the hardware database each time there is a potential change would take too much time and might actually impact the performance of the access control system. This also requires that the hardware be in constant communication with the host computer. This is not always the case and is in fact, not required by the hardware system chosen. The Active Directory application also includes a database, however Microsoft explicitly describes it as a read many write once type of database. The intent is not to be a repository of transactional data.

The database in this application is used for two purposes, first provide a

backup of the data on the access control hardware, and second provide a fast mechanism to validate any changes to the hardware based database. As the user makes changes to users, doors and schedules in the Active Directory GUI, AD forwards these changes to the application for further processing. AD forwards any change to an object, many of these are not relevant to the access control system. The middleware application must sort through all the changes and only pass the relevant changes to the hardware sub-system.

The business logic makes the determination of the relevancy of any changes forwarded to it. The application logs changes into the database and forwards them to the hardware sub-system. This mechanism makes the application database the backup data repository for the access control application. The application designers intended it to run on a computer running on the network and expected that the IT policy will include accurate and timely data backups of the database contents.

One significant function of an ACS is to format an event message in a human readable format. The business layer provides this functionality. Event messages from the hardware system consist of a series of tokens that are abstract representations of the information. These generally are integer values and not readily understood by the system administrator.

4.2.3. Communication Services Layer

The communication services layer implements the application's interfaces to other program units. This layer does not include the User Interface. This application uses the following interfaces: hardware sub-system, Microsoft Active Directory, Windows event logger, XML message consumers, and a database interface. The implementation of each of these interfaces is as a class that contains any required data members and the entire set of associated interface functions.

Figure 7 – Communication Service Layer Class Relationships shows the relationship between the interface layer classes.

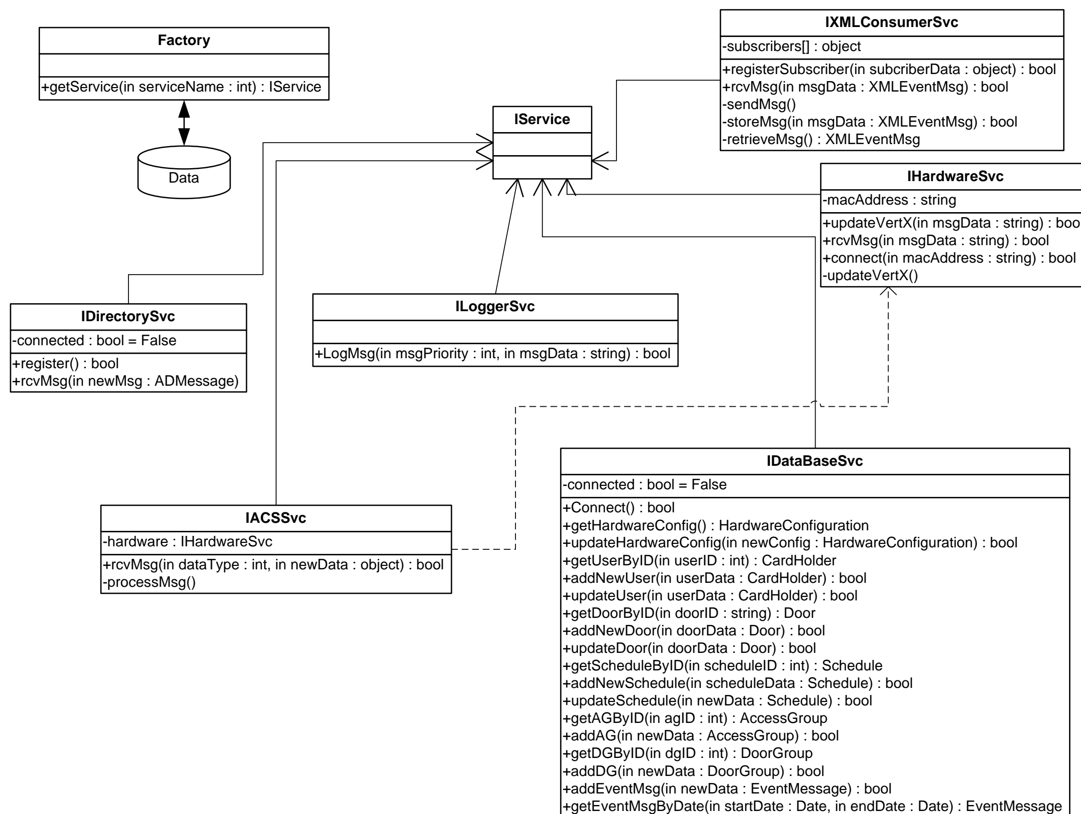


Figure 7 – Communication Service Layer Class Relationships

The team implemented the communication services layer with a Factory design pattern (Gamma, 1994). With this design, the factory returns an object of type IService to the caller. The caller knows the exact type of the class so it can then use the returned object appropriately. The Factory does not know the type or capabilities of the object it creates. By using this design pattern, the implementation team can extend the application to include additional behavior with very little impact on the current application.

During the design phase, a review of the communications service classes revealed that each class would always interface to one and only external object. In 1994, Eric Gamma stated that the intent of the singleton pattern is to “Ensure that a class only has one instance, and provide a global point of access to it.” To ensure that multiple instances did not attempt to access the same external resource, it was a decided to implement each of the classes as a singleton. A description of each different interface class is in the following sections.

4.2.3.1 VertX Hardware System Interface

This project utilizes access control hardware provided by the HID Corporation of Irvine California. The manufacturer designed this system interface as an “open” platform that a variety of ACS providers would use in their products. The intent of the product design was to remove the dependency that currently exists between the hardware and software elements of an access control system. HID Corporation does not provide the software component of an access control

system, only the hardware.

The interface to the hardware is through a Dynamically Linked Library (DLL) provided by the hardware manufacturer. The DLL provides a “C style” function call interface. This application implemented that interface in its design. The hardware interface object acts as the single point of contact between this application and the under-lying hardware system. This interface has proven to be robust and easy to implement. As the hardware system interface changes any required changes are isolated to this single class increasing the maintainability of the application.

This class provides a function call interface to the business logic layer that handles all data translation and communication with the hardware. The class provides functions to connect to the hardware, handle cardholder maintenance as well as system configuration and to receive the event and alarm messages generated by the system.

4.2.3.2 Microsoft Active Directory Interface

This interface class provides the communication channel between the Active Directory application and the project’s business layer classes. It includes the register function to notify Active Directory that this application is interested in receiving messages. It also includes the receive message function that is passed to Active Directory.

4.2.3.3 Windows Event Log Interface

This class exposes a single function to the business layer, a receive message function. When the class receives a message from the hardware it will format and forward the message to the Windows event log service for storage and retrieval by the system administrator. The system administrator may view the messages at his/her convenience and the messages are stored for reference later.

4.2.3.4 XML Message Consumer Interface

Computer applications generate large amounts of data that other applications may use. Data generated on one type of hardware or operating system may not be usable on another hardware or operating system. The cause of this interoperability situation is differing data types used in the application and a multi byte number storage artifact referred to as Big or Little Endian (Verts, 1996). The World Wide Web consortium (W3C) has adopted eXtensable Markup Language (XML) as a standardized mechanism to address the problem of data interoperability. Typically, a middleware application formats the data to match the XML schema before transporting it between applications and computer systems. XML transports the data in as string data in ASCII format. In this form, applications running on different hardware or software can use the data.

XML has received a lot of publicity, not only in the technical press but also in the popular press. A recent Google search returned 2.2 billion hits for the term “XML.” Senior company management has read much of this press prompting discussions about how XML can solve their particular problems. The access

control industry's management has also read this press and asked the same questions of their technical staff.

Within the access control industry, there is an attempt to exchange data with other building automation or enterprise applications. XML has been at the forefront of the efforts to make access control generated data available to these other applications. Unfortunately, there is has not been any effort by the industry to adopt an XML standard and at this time, the industry has not even proposed an XML ACS standard. This project includes the design of a proposed XML schema for access control messages.

This class provides a simple XML interface to other applications. At this time, the team has not demonstrated the class as there is no current consumers of the data available. Appendix 1 on page 60 shows the XML Schema (XSD) for this XML design.

4.2.3.5 Database Interface

The database within this application provides a store for persistent data. The application replicates the configuration data that is stored on the hardware sub-system to the database for redundancy as well as speed. The time to retrieve data from the hardware is significantly longer than the time to retrieve data from a database application. In addition, the data stored on the hardware is not stored in a table format or in a normalized form. This increases the complexity of the logic required to use the hardware as a data store.

The data base interface class provides all of the required functionality to connect to, store, and retrieve information from a database. This design provides a single interface point for data storage and retrieval.

The database chosen for this application is Microsoft's SQL Server. The implementation of the class includes the required syntax for this database. If another database is needed changes are only required in this class.

4.2.4. Data Objects Layer

The data objects or domain objects represent the "things" that exist in the problem domain. The data object things are typically the nouns in the use cases. When the implementation team implements these objects, they generally contain data but little or no associated functionality. In application design, the application creates and populates these objects in the communication services layer and then passes them to the business logic layer. The business logic layer then uses the data within each object as it processes the business logic.

An analysis of access control requirements revealed the following objects are required to provide an adequate representation the problem domain: cardholder, door, and a group of doors, access level, access group, a schedule, and events. The implementation of each of these objects followed the Domain Model design pattern (Fowler, 2003).

Figure 8 - Data Layer Class Relationships shows the relationships between the various classes in the data layer.

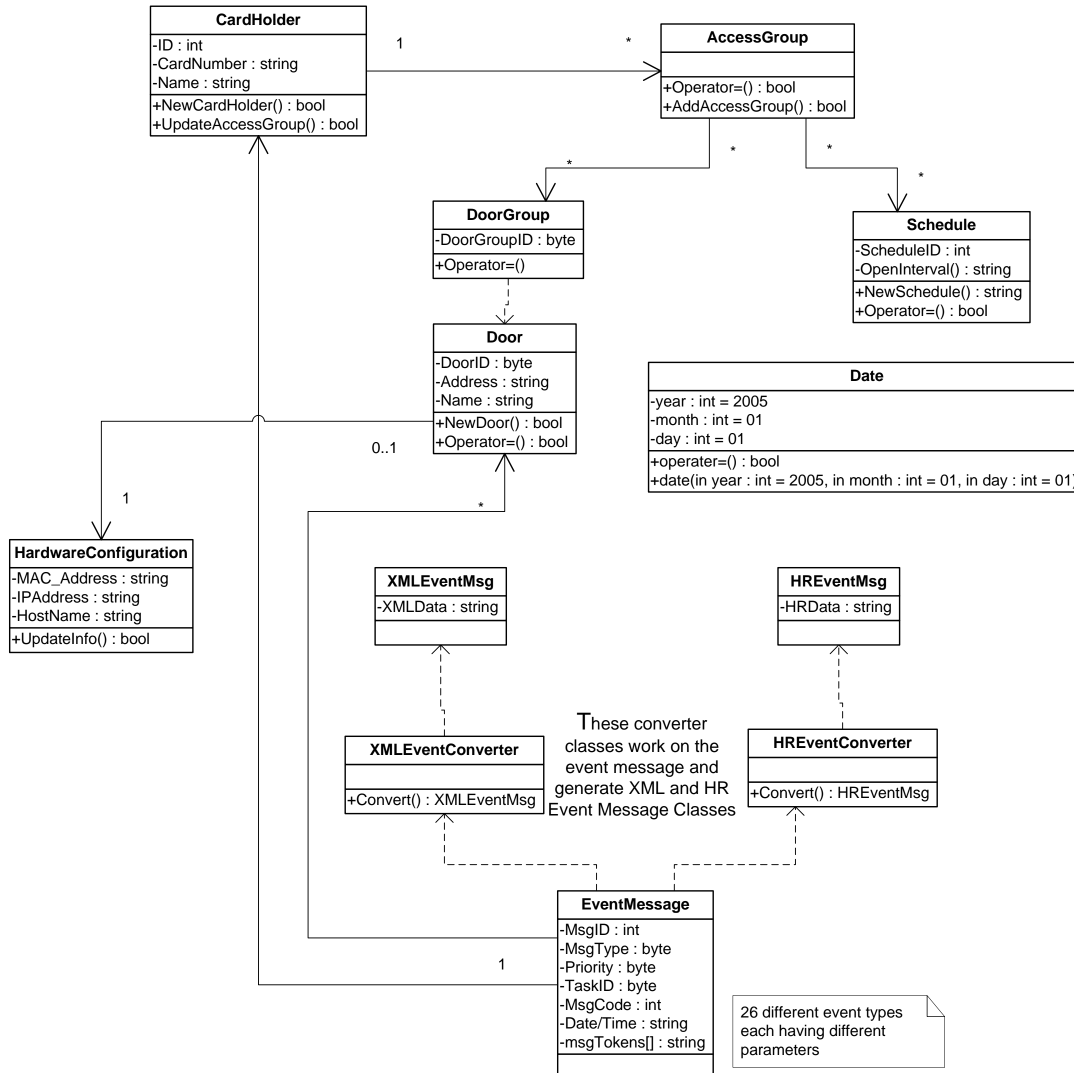


Figure 8 - Data Layer Class Relationships

A discussion of each of these objects is in the following sections.

4.2.4.1 Card Holder Object

The cardholder represents the user of the system. The system administrator grants access privileges to the cardholder object. An access privilege consists of a door and time that the user has permission to use that door. An associated card number identifies the cardholder to the system. The system administrator assigns each cardholder a unique card number to accurately track

the user's use of the system. Each user is issued a token encoded into this token is a card number. Through the card number user association, the application generates reports the user's activity or in-activity. The data included about a user includes the user's name, card number, associated access groups, and the status of the cardholder (active or in active). Each of these data points represents data that is required somewhere in the system.

4.2.4.2 Door Object

The door in an access control application represents the 'where' of the system. It may represent either the physical portal used by the cardholder or an area controlled by the portal. Within this application, a door represents the portal used by the cardholder to gain access to physical resources. When the installer installs hardware, he assigns each door a unique address to identify the door to the system. When the user presents an access control token to the system, the system uses the door address and card number to determine the cardholder privileges at the specific door. The data included about a door includes the door ID, door address, and door name.

4.2.4.3 Schedule Object

The schedule in an access control application represents the 'when' of the system. A schedule defines the in and out periods. The in period (in-schedule) defines when the resource is available and the out period (out-schedule) defines when the resource is not available. When the user presents their token to the system, the current time is associated with the card presentation event. The

system uses this time and door ID when it determines if the door is unlocked or not. A schedule includes a name, days of the week, holidays, and one or more intervals for each day.

4.2.4.4 Door Group Object

To ease the burden of administering the system, the administrator groups doors into door groups. It is common for several doors to serve similar purposes. For example, at most facilities, the front, and back are commonly referred to the “perimeter or outer doors” and the user would have the same rights to either of the doors. These two doors would then be associated into a door group.

There are situations were the administrator decides to manage each door individually. This application is able to deal with doors in this manner when indicated by the administrator. A simple assignment of one door to a door group provides this level of management control.

4.2.4.5 Access Level Object

An access level is the association of a door group with a schedule. The system administrator will grant users privileges to doors with similar purposes during the same time period. If a cardholder has permission to enter the facility through the front door at 8:00 am then the cardholder would generally have the same privilege at the rear door.

4.2.4.6 Access Group Object

An access group is an association of one or more access levels. The administrator assigns the cardholder object one or more access groups that represent the accumulated access privileges granted to the specific cardholder.

4.2.4.7 Event Objects

Event Objects represent one single event that occurred with in the access control application. When the system generates an event, the VertX hardware prioritizes the event and then passes it up to the host application as an event message. The application converts each of the event messages from the VertX to an event class object. The application uses these objects to move data between the different objects that use event data. These include the conversion classes that generate the human readable message and the XML formatted message.

A review of the VertX event message structures revealed that there are twenty-six different message structures. Each structure requires slightly different processing. The implementation team implemented the event message objects using an inheritance model to minimize the common data elements in each of the classes. The design team identified a base or parent class that included the common data elements and functions and each of the twenty-six separate event classes then inherited that class. This approach reduces the code implementation requirements, increasing the maintainability of the code. If a change is required to one of the common functions, changes are limited to a single place in the application. All of the child classes then inherit these changes.

4.2.4.8 Hardware Configuration Object

The final domain object in the application holds the specific hardware configuration parameters needed by the application. These include the MAC address, IP address and the host name. The application uses these parameters when communicating with the hardware platform.

4.3 Detailed Database Design

4.3.1. Database Schema Design Process

The design team used a multi-step process when designing the database schema. First, the design team created a data dictionary with the data required by the hardware configuration files and then the team created an Entity Relationship Diagram (ERD). Next, the team added the data required to transform the event messages into a human readable format to the ERD. A review of the XML formatted message structure revealed that no additional data was required to generate this type of message. The final step was a normalization of the data using standard database practices. The target was third normal form. Third normal form generally provides the best tradeoff between performance and data redundancies or data anomalies (Rob and Coronel, 2004).

Once the ERD was completed, a series of SQL statements were generated that would create the tables and with the referential integrity constraints required to store and maintain the validity of the data. Finally, the team generated a series of SQL statements that populated the tables with the “factory default” data that is included with the hardware.

The implementation team initially validated these SQL statements on an Oracle 9i database. Next, the implementation team executed the same set of statements on a Microsoft SQL Server database and a couple of minor changes were required. These SQL statements are now capable of creating and populating the required tables with either an Oracle or a SQL Server database. The team could extend the application, with localized changes (to the database scripts and the database interface class) to operate with other relational database applications.

4.3.2. Factory Default Data

VertX hardware includes many configuration parameters that are set at the factory. The installer or provisioner of the system does not need to change these parameters to get a functioning access control system for most users. By including the default data, an initial data load by the software is not required. Additionally, distribution of the hardware could occur through the channels that are currently in place, reducing the cost to the end user. The hardware is available through a warehouse distribution system that makes it available to anyone interested in deploying it.

5 Conclusions

5.1 Did the Project Meet Initial Expectations

The project did meet the initial set of project goals. This included using Active Directory as a user interface mechanism, using existing Windows functionality to handle event reporting and to implement the application in C# using an N-tier design approach.

The project design phase spanned several of the classes at Regis University. The data base design occurred during the MSCD 600 and 610 courses; the initial software design occurred during MSCS 600 and then refined during MSCS 630. The majority of the implementation phase occurred during MSCS 680 and 682. This focused the classroom knowledge on a specific and identifiable real world problem, aiding in assimilating the knowledge from the classroom, and moving forward on the project.

5.2 Lessons Learned

Takuya Katayama, of the Japan Advanced Institute of Science and Technology, in a presentation to the Tenth Asia-Pacific Software Engineering Conference in 2003 stated:

> It (software) has to change as its infrastructure has been changed. It has to change as its functionality has to change. It has to change as new algorithms are found which will increase its performance. There are a lot of reasons why it has to change and only software that can

stand for the change can survive.

To address this continual need for change, the design team utilized the N-tier architecture, a database, design patterns, and C# in the design and implementation of the application. A discussion of each of these decisions is in detail in the following sections.

5.2.1. N-Tier Architecture

This design philosophy has proven to provide flexible implementation that is changeable as the needs of the product change. One of the keys of managing the expected changes is a design that lends itself to change. N-Tier Architecture has proven itself modifiable as a project progresses with minimal impact of the rest of the application.

Reflecting on the quote by Mr. Katayama, the N-tier architecture confines many of these changes to a single layer. These changes might include a different database implementation or a new communication protocol. To implement these changes the team must only make changes to the technology layer class that addresses that specific part of the overall system. If the stakeholders defined different business rules or application functionality requirements, the team would confine the changes to the business layer class that defines the application functionality.

In a well-designed and implemented system, changes to any of these layers

will not affect the implementation of the other layers; many times recompilation of these other layers will not be required. This enables the software to change as the needs of the users of the software change while minimizing the impact on the system as a whole.

5.2.2. Database Design

The implementation team has two choices when converting a data base design to the actual implementation, the tables, and data in the tables. First, you instantiate the tables in the data base with all of the required attribute fields. Next, you “load” the data into the tables. Finally, you “turn on” the referential integrity constraints and correct any errors reported by the Referential Database Management System (RDMS).

The second method involves creating the database tables with all of the referential integrity constraints in place and then adding the data. This project team chose to employ this option for several reasons. First, it reinforced the referential relationships in the database design as the design evolved. Next, because a script inserted the initial data, the implementation team modified the script to comply with all of the constraint requirements. It is felt that this method provided a deeper understanding of the impact of the referential integrity constraints on the design.

5.2.3. C#

One of the stated project goals was to implement this application in C# as means to increasing the author's knowledge of and skill with the language. Previously the author's primary skill set was limited to Visual Basic 6.0 with a very limited exposure to C++. With this as a foundation, the migration to C# proved to be a straightforward process. C# includes a wide range of built in classes that abstract the underlying Windows API. Not only does this ease the learning process it also provides a built-in mechanism to enhance program reliability.

5.2.4. Design Patterns

The decision to attempt to evaluate each class and apply a design pattern to implementation proved to be a very worthwhile exercise. This approach forced the design team to design and review each class prior to the implementation of the class. The modification of several of the class designs during the review process saved valuable time during the implementation phase of the project. In addition to the time saving the review process also tended to increase the robustness of the over-all design of the application.

5.3 What Would I Have Done Differently

The one area that needed additional attention was the use of Active Directory. The author believes that extensions to AD could include the concepts of doors and access levels. The author needs to do further research and education to utilize the extensibility and this should have done earlier in the project.

5.4 Future Application Development

5.4.1. Where the Project Can Go

The next sections address specific extensions to the project. Included are several changes to Active Directory, additional control functionality, XML message changes, and finally changes to the database. Each of these would add functionality that would enhance the application.

5.4.2. Active Directory Extensions

Active Directory is a Directory Service, meaning that it will authenticate a log in and password from a user to grant access to system resources. A very interesting extension to this project would be to add to the AD algorithm a check to ensure the user has “badged in” to the access control system. This would require that the ACS application pass back to AD all of the successful grant access decision that are made. This message would include the date and time of the access as well as the door where the access event occurred. Active Directory would use this additional information when it made the decision to grant access to other system resources. This enhancement would tie the access control decision to granting of resource privileges providing further control of the network and the data stored on the network.

5.4.3. Control Requirement

The one required piece of functionality not addressed in this project is control of the system. This includes the ability to open a door on command, reset an alarm point, or lock a door. The VertX interface class includes the ability to send the required commands; the missing piece is the user interface. One option would implement a screen from within the application. This would require that the implementation team design, implement and test a GUI. One of the design goals was not to implement any new user screens. Another alternative that bears further investigation is to extend Active Directory to include a screen that handles the user input for this function. This would probably require that a DLL be created that handled the screen and commands.

Another aspect that bears investigation is to extend Active Directory to present the concept of a “door” to the user as simply another of the resources managed by AD. This would require that some additional design work to identify exactly which data be required to represent a door without including extraneous data. This would also probably require the team develop an additional DLL.

The current implementation simply uses the schedules as defined with Active Directory and forwarding them to the hardware. A better approach would be to allow the creation of a separate access control schedule. Typically, these would provide some additional time to allow the worker to arrive early and prepare for the workday before they would actually have privileges on the network.

Finally, the current implementation does not provide a mechanism that

allows the user to change the name of the various access control output points. The design team defaulted these parameters to what it considered would work in most situations. It also would require the hardware installer to wire the system in a pre-defined way. Although this is practical for most installations, some the team should do some additional investigation to achieve this flexibility.

5.4.4. XML Message Service

A significant advantage to newer access control applications is the ability to “interface” with other applications running in the environment. The access control industry considers an interface as the ability to receive some well-formatted data, typically from a database system or outputting a message. An access control application that is more than a few years old will not have the ability to either import or export data.

The team could implement this service in several different ways including a Windows message queue, in traditional client-server architecture or using asynchronous messages. A complete solution would be an implementation all of the identified methods providing the greatest interoperability of the application.

The first option would be to implement a Windows message queue. The receiving application would have to implement a message interface to receive the messages sent by the ACS. This has the advantage that the receiver does not need to be on-line all of the time. The ACS would send the messages to the queue were they would be stored until the receiving application came on-line. When the receiver application comes on-line, the message queue forwards any

stored messages to it.

The next option would be traditional client-server design. In this design, the ACS application would be the client and send messages to the receiving application. The receiving application would need to implement a server to listen for the asynchronous messages from the ACS. The ACS system would need to store undeliverable messages and forward them when the server was ready to receive them.

Finally, the team could implement an asynchronous message scheme through a callback mechanism. In this design, both the sender and receiver must be on-line at the same time to transfer messages just like the client-server design. The ACS would need to store undelivered messages. In addition, there could potentially be a requirement that part of the receiver implementation include un-managed code.

Each of these options has both advantages and disadvantages. The design team would need to perform an analysis of each to provide the correct solution of each potential user.

5.4.5. Database Extensions

There are many popular databases beyond Oracle and Microsoft SQL Server. The team should test the ‘create and populate’ database scripts against these other databases. In addition, the team would need to develop additional database interface classes to interface with the database.

A true low cost system could be implemented using Microsoft Access or Oracle Personal Edition databases. In either case, the only cost to the user would be the hardware and installation costs. This would be a very attractive option for most companies employing 100 or fewer employees.