



An eBay's Desktop Management Solution

Omar Elangbawy
omars@seas.upenn.edu
Advisor: Prof. Susan Davidson

Abstract:

For the past two years, I have been independently developing and running an online retail venue using *eBay* and *eBay Stores* for a men's clothing store which I work for outside of Penn. In this time, we have managed to open the store, create the necessary accounts, and design HTML online listings to begin selling the items in the store through *eBay* stores. However, thus far, we have relied entirely on the web-based applications provided by *eBay* to accomplish these goals in addition to a very primitive inventory control. This method has not only proven to be extremely inefficient, but also very time-consuming. At the moment, there is no way to save previous listed items in an organized and efficient manner so that the details of the item could be looked, edited, or re-listed on *eBay* at a later time; not to mention the ability to efficiently search and find items in the inventory that are online so that they can be removed from *eBay* with ease. In addition, tracking sales is limited to the past 60 days, as is customer information. When it comes to inventory management between the physical store and the online store, we have been forced to resort to separate tracking systems, one for items online, and the other for items in the store, thereby not only making it cumbersome, but also prone to more human error. Therefore, what we have now is an offline desktop application which can synchronize with the *eBay* store for the user, in this case Fashion House Inc., and can list, edit, and remove online items, but also incorporates an inventory management system, as well as a customer relationship center, which would track customer demographics and details for every online transaction. With this basis, it will not prove to be difficult at all to extend the application to allow it to serve as a point of sale application that can interact with some form of a register or computer at the physical store location.

The newly released *eBayer* 6.0 currently provides an interface to handle the complete array of online store management functions, which was previously handled by various web-based applications. It will serve as a tool to cover all aspects necessary in running an efficient *eBay* store and incorporating it with the physical store location. For example, it allows a store manager to list items in the online store, while at the same time saving the item specifics for later use and reference. Not to mention, it combines the online details

with details applicable to the actual physical store using style numbers, additional size details, and short descriptions for the store manager to utilize if necessary. Nest, it allows for easy removal of both items sold online and through the physical store. In other words, if an item was sold in the physical store, removing the item from inventory would also remove from the *eBay* store, if a duplicate does not exist in the store inventory; and similarly, if an item was sold online and the application is synced with *eBay*, *eBayer* 6.0 is built to check to see if a duplicate exists, and if so, it will re-list the item on *eBay* and update the appropriate information in the item that exists in the inventory. If a duplicate does not exist, it removes the entire item from the inventory to ensure that the same item is not sold in the physical store. Once a sale online is complete and the inventory is adjusted, the customer information for the buyer and the specifics on the item sold and the transaction are saved to the appropriate tables for later use. This information can then be used to identify likely customers for future offerings, as well as best selling items. Thus, the store is now capable of handling every necessary function to run and monitor a small-scale *eBay* store from a desktop application.

Related Work:

When researching the vast amount of products and solutions available to *eBay* users whose scale requires the use of such an organized selling technique, there are many options. As mentioned earlier, all functions necessary to run an *eBay* store are accessible using *eBay*'s web-interface, however at some point, it may become inefficient and impossible to operate a store of a certain scale using these functions. Nonetheless, the existing solutions which attempt to incorporate the store management process are, if not flawed, then very impractical for a medium-sized store. In other words, the size of Fashion House, with 900 online listings at all times, and sales ranging from \$5,000 - \$10,000 a month only from *eBay* requires the use of some automation for both efficiency purposes and to minimize as much human error as possible when organizing inventory between two separate, time-dependent venues. However, the alternative applications, although some may come close, all have their own limitations or features that do not make them as appealing as possible. Rather, most do not meet the specific needs and

simplicity desired for small to medium scale stores. Examples of certain applications and possible solutions that already exist, as well as the limitations associated with each, include:

eBay Turbo Lister:

- § **Type:** Listing tool
- § **Limitation(s):** Cannot sync with *eBay* store and can not therefore, maintain accurate and up-to-date information on items within the store. In addition, the program only supports the listing of items and does not support re-listing, removing, or editing online listings. Nor does it incorporate any type of physical inventory integration and management features.
- § **Price:** Free
- § **Conclusion:** *eBay Turbo Lister* is not a complete seller solution.

Meridian:

- § **Type:** Complete auction management solution.
- § **Limitation(s):** This tool provides the ability to create and list items on *eBay*, as well as track customer information and sales information. However, the program is extremely difficult to navigate. In addition, this specific tool does not contain an inventory management feature, thus limiting any off-line integration.
- § **Price:** Variable monthly fee (percentage of total sales)
- § **Conclusion:** Complex and incomplete solution with an added expense structure for managing the store.

Auctiva:

- § **Type:** Complete auction management solution.
- § **Limitation(s):** This solution provides the ability to accomplish many of the standard features you would expect to find. It can create and track listings, can track sales information, but only that which is

currently available through *eBay* (in other words, it does not save sales information for long term use and research). It is also capable of generating reports and tracking customer information. However, like all other available solutions, it does not incorporate a method to manage personal inventory outside of *eBay*, and in addition, from user reviews, it seems to have many errors and be extremely unfriendly on the user side.

§ **Price:** Free

§ **Conclusion:** Effective solution for use in an *eBay* store ONLY, however, not very user friendly.

InkFrog:

§ **Type:** Complete auction management solution.

§ **Limitation(s):** This solution offers the essential auction management features such as listing and sales management, but also fails to manage physical inventory. It does, however, incorporate leaving feedback, which not many auction management solutions handle for.

§ **Price:** Flat monthly fee

§ **Conclusion:** Lacks essential integration capabilities.

Andale Auction Management:

§ **Type:** Complete auction management solution

§ **Limitation(s):** Although this solution does claim to offer “robust” inventory management features, it does not incorporate any tracking system for sales and customer information. Therefore, you have no way to communicate to repeat customers and view past sales information and invoices if needed in the future.

§ **Price:** Variable Monthly Fee (percentage of sales)

§ **Conclusion:** No customer and sales information is stored, not to mention the added expense in operation costs.

Perhaps the most appealing of solutions, is the highly recommended, *eBay* developed, *Seller Assistant* auction management solution (featuring both the Pro and Basic versions). Although this solution has many features suitable for running an efficient *eBay* store, it lacks certain key features that many *eBay* stores will need. The concept of an *eBay* store is to make online sales both affordable and easy to use for medium to large sellers. Therefore, it is safe to assume that the majority of *eBay* stores also have physical counterparts. In other words, many *eBay* stores are an extension of an existing physical establishment. It becomes apparent that there is a need for the capacity to handle and manage inventory on two entirely separate fronts – online and in person. Thus, the *Seller Assistant Pro* does not include an efficient inventory management system suitable for smaller scale stores, and on top of that, it charges a monthly fee to its subscribers.

The list goes on with many independently developed applications. Nonetheless, there is not one program which will incorporate physical inventory management with auction management, including listing and sales management capability. Thus, the difference between my system and existing solutions is two-fold. First and foremost, the system is free. Prior to the recent re-design of the *eBay* API and the *eBay* Developer's Program, any one account was allowed 10,000 API calls per month (as stated in Project Revision I), which included retrieving the information for an item, information on a sale, and listing an item. This basically had the effect of limiting the size and scope of a store suitable for the *eBay* application in mind. However, as of last month, *eBay*, in order to foster increased development and creativity, restructured the cost structure of making API calls to make it free for any account. Thus, now, more than before, can more powerful solutions be made, without the necessity of charging clients based on usage statistics. Secondly, *eBayer*, will not only act as an auction and listing tool, but also as a point-of-sale inventory management tool that has been designed and implemented with the store specific necessities and concerns as well. For example, if an item was sold in a store and removed using the application, *eBayer* then synchronizes with *eBay* to remove the item from the internet listing if a duplicate item was not available for sale in the inventory. On the other hand, if an item was sold through the *eBay* store, then the quantity of the particular item would be decremented in the database, and removed completely if none

were left, thus creating a manner to prevent the item from being sold in the physical store prior to the item being shipped. Thus, not only would the application save time between tracking and managing available inventory, but it would also automate the process, making it less prone to human error. If necessary, the application can then be later upgraded to scan bar codes at the point of sale terminal, and handle physical as well as online item inventory, without any human interaction.

Technical Approach:

After analyzing the necessary requirements for the project from the seller side, or store side, I concluded that there were a few necessary tasks and objectives which would need to be met by this desktop application and its ability to sync with information from its *eBay* store:

- § Manage physical inventory in database (add, view, remove, edit, sort, etc...)
- § Manage online listings (add, remove, re-list, edit, etc...)
- § Handle for interaction between online store items and physical inventory in the following cases:
 - § When a sale is completed in the physical store and the user removes the corresponding item from the inventory
 - § When a transaction is recorded on *eBay* and the program synchronizes with the online store to retrieve the applicable transaction and buyer information
- § Maintain a customer table in the database housing information on our online customers to be updated either manually, if necessary, or once an online transaction is complete
- § Maintain a table on sales information, providing a source of history to every online sale made for use as a reference to certain details of the buyer or item purchased if necessary
- § Ability to sort entries in store tables to answer questions such as:
 - § What are our best selling items online?

- § Who are our best online customers?
- § How many returning customers do we have?
- § Creating and saving coupons and sale offers that could be sent out when necessary

Basically, the program will serve as an intermediary between the numerous *eBay* web applications and the user, while at the same time incorporating an efficient inventory management system, based on a style number. Thus, once the main objectives were analyzed, then it was time to analyze the methods I would use to complete the necessary tasks.

I began the project by surveying the different technologies and frameworks that I could have possibly used to implement the *eBayer* application. Taking into consideration all of these options, I decided to use Microsoft Visual Studio .NET 2005 as my IDE, using C# as the programming language. This decision was made based on the ease of use of the IDE as well as the abundance of available *eBay* SDK sample code written in C#. As for the database, I have decided to package MySQL 5.0 with my application because it is light-weight and free, making it easy to package within an application.

Once the development choices were made, the first thing I did was familiarize myself with the *eBay* API and how certain calls were made, as well as the features that could possibly be exploited. Using the assigned developer ID's, I would be able to access all pertinent information for a given *eBay* store in real-time after establishing and receiving an *eBay* authentication token. This was done as follows:

```
// Make the actual API call
ApiContext Context = new ApiContext();

// Credentials for the call
Context.ApiCredential.ApiAccount.Developer =
"C3T56JF4ZR3173Q49I6FBXG161A17S";

Context.ApiCredential.ApiAccount.Application =
"FASHIONHOUB5EPA911K48SN3896J89";
```

```

Context.ApiCredential.ApiAccount.Certificate =
"C559U454G9S$M67JHJ3T6-61H18UAJ" ;

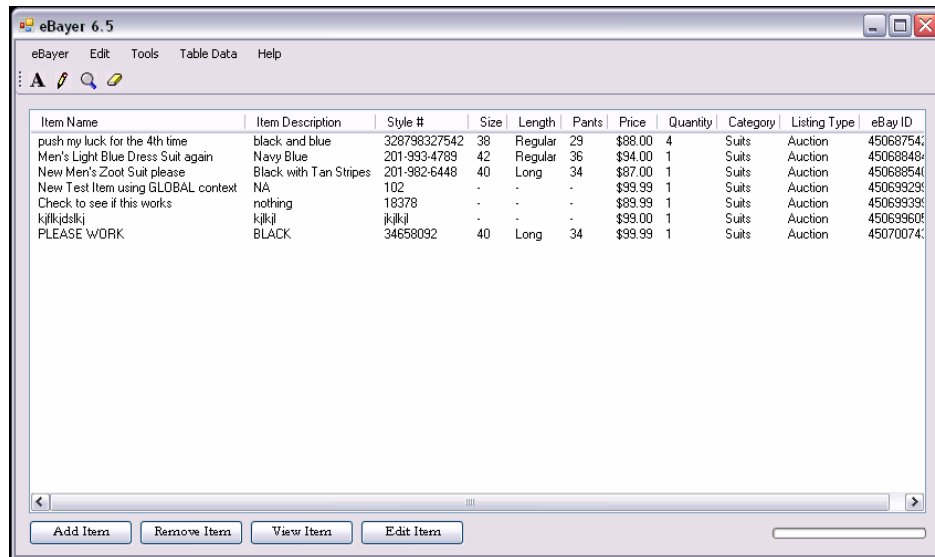
Context.ApiCredential.eBayToken =
"AgAAAA**AQAAAA**aAAAA**8YMxRA**nY+sHZ2PrBmdj6wVnY+sEZ2PrA
2dj6wJnY+lDpmAoQ2dj6x9nY+seQ**AVoAAA**AAMAAA**hAkzB/cP4kQKP
JjR+lOfrQkvkq87TyabAwlWPgp0WOPu7GJKdXP5ta52FPfwnlWl0Ulw5rF1
8XNwf8ziTfZ+/naIvhCFD+CZtOvLD0/A5rOLwTkI16SE/ny5BNSW/5XNjac
UMJtbGeOwf2yTCFrhnmKKzHhjn+v+8amsIF15owNIbFGoZR62fLJD1U9NK
BfLzaLDAKW3oD0xG+8LrmAlBdSoetrB7oOH02v51Rj9nWlXR9xvXsWS6W9Y
rxJQzKNcz8i7n46gn94ExQQuzCcXf9D5Xp56TKFztp8oPXla/ahLrwc56Z+
jGgdCh1Dk21KzKoecmJatby8HteOXNOawlVtDLUP4T6EFzzO3iDvLmoqKts
2yq1h+ukaUQoPbrnCJf4KsXY6OyOm5NIIXpmpRIjgQbhUssR7NBzGpC9JEK
PmYrUYAtogADkScVaGlsSxXv8NYilxpBN9r2plA3kvJrSDRjAxvCdnXef7C
CHp/lTZgTF1TCvw+80RxB9NIHMqa9mRKmgHypEHgPL3trHeAbtmgzqjVvX4
3lVGkTLTexMgZjIjur1EvMDIeP2bLuFK34SdJE3gF/ytoa7Fwsf44FMhz8j
L2Pm+wWOUknmMyFe/lzMpWxdPkJ1YmXGYmsQZqfHFAdT+1LvX3MnhWZcSnB
ihoelul+b/jEQbtir7tgjFQtS4QlMmeSxDuviKKFMRM7Vj7xwXKEW1r09kS
N4gEYPxP7MkVVCKB63zJoAeomTxebrLhjhDksTce4NNAaUY" ;

// Set the URL
Context.SoapApiServerUrl=
"https://api.sandbox.eBay.com/wsapi" ;

```

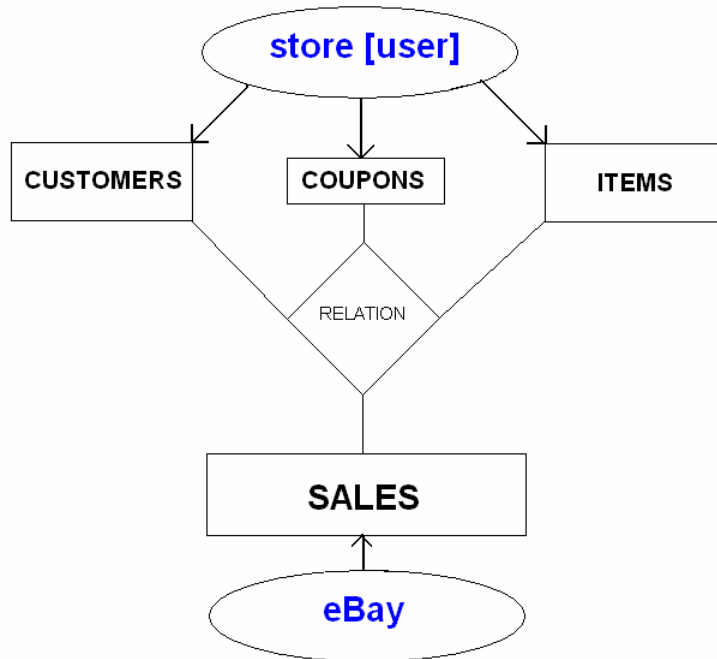
At first, I considered maintaining a database with *eBay* item information which would be updated every time the application was synced. Then, with the new, updated information, I would compare it to the database containing the physical inventory. However, because there are certain alerts which can be set to receive new sales information and feedback information, I was able to simplify the anticipated logic to the structure of the application. Instead of updating an entire database, we could now simply receive only that information that is new and handle it appropriately. Instead of two separate databases, we now only had one central location for the inventory, which now handled both online and store functions. This simplification in logic proved to be one of the most challenging concepts. It appeared as though the notifications were going to make the program a lot more efficient, if I could somehow receive the notifications. Upon attempting to incorporate this structure into the development of the program, I realized that the notifications were posted to a pre-defined URL or email address that would have to be constantly monitored by the application. The first question was, “Is it efficient to constantly check for notifications?” However, I then realized that only certain member subscription levels qualify for platform notifications. This was a tremendous setback because Fashion House did not have the ability to subscribe and would cost significantly more to implement this concept. Thus, at that point, I managed to reorganize the program,

although slightly more complicated in logic, to still utilize only one central database and occasionally check for transactions using certain filters such as the date or item Id's of the desired transactions. More will be discussed when handling transactions are discussed.



Basically the structure of the program is as follows. I have created a database, with six tables. These tables will store information on customers, items, coupons, application data, removed items, and sales – which is the relationship between customers and items. To better illustrate the functions of *eBayer 6.0*, the lifetime of an item and the associated actions are described in further detail below.

Below is a simple block diagram of the components within the system and how they interact with each other:



Adding an Item:

ADD ITEM DETAILS

Item Name [Title]

Color/Description Style Number Quantity Category ID Jacket Size

Item Description

Waist Size

Inseam

Jacket Length

Style

Material

Link to Picture

Condition

Store Category Auction Type Listing Price \$

There is one central way in which an item can be added to the inventory as seen above. Basically, the way it works is that the user will input all the necessary fields to add an item to the inventory, in this case, a suit to the inventory, which is the sole focus of

Fashion House Inc. This form not only contains necessary information for the online listing that is required by *eBay* but also contains data pertinent to the store itself for inventory purposes. The other necessary, standard inputs for the online listings such as payment methods, currency, etc... are handled and set up as defaults when an item is to be added. Once, the required fields are inputted correctly, the user will submit the information, upon which he/she will be prompted to determine if they merely want to add the item to the local inventory only, the online store inventory only, or both. If the item is to be added to the local inventory, the program will establish a connection to the database and insert the appropriate information in a new row within the table, so long as an identical item does not exist in the store's inventory. If the item is to be added online, the program will fill in the appropriate fields to the ItemType object in the *eBay* API that is created before the actual API call. Once the item is ready, the program will call the AddItemCall() which if successful will return the *eBay* item reference number (item ID relevant to the online listing in the *eBay* database) as well as the total amount in fees for the particular listing.

Managing an Item:

VIEW ITEM DETAILS

<u>Item Name [Title]:</u> PLEASE WORK	<u>Quantity:</u> 1
<u>Color/Description:</u> BLACK	<u>Category ID:</u> label1
<u>Item Description:</u> <pre><TABLE height=20 cellSpacing=0 cellPadding=0 width="100%" align=center bgColor=#7d8099 border=0> <TBODY> <TR bgColor=#6a6d8e> <TD width="63%"> <DIV align=center>MEN'S SINGLE BREASTED 100% WOOL DRESS SUIT</DIV></TD> <TD width="37%"></TD></TR></TBODY></TABLE> <TABLE borderColor=#6a6d8e height=426 cellSpacing=0 cellPadding=0 width="100%" align=center border=1> <TBODY></pre>	<u>Jacket Size:</u> 40
<input type="button" value="View in Browser"/>	<u>Waist Size:</u> 34
	<u>Inseam:</u> Unhemmed
	<u>Jacket Length:</u> Long
	<u>Style:</u> Double Breasted
	<u>Listing Price:</u> \$ 99.99
	<u>Material:</u> Cashmere
	<u>Condition:</u> New: With Tags
	<u>Store Category:</u> Men's Single Breasted Suits
	<u>Auction Type:</u> Auction
	<u>Style Number:</u> 34658092

Link to Picture: <http://img392.imageshack.us/img392/9372/dscn22498ww.jpg>

Once an item is listed online and/or stored in the local inventory, the user can do one of three things. The user can either view the item after finding it in the list view window which can be sorted a number of different ways, edit the item that has been selected, or remove the corresponding item. If the user were to view the item, a form would load all the corresponding details to the item allowing the user to note the characteristics for store purposes or edit the information by clicking on the edit button if something is found to be incorrect or needs adjustment. It also allows for the user to click on the URL to the image associated with the item for viewing through your normal web browser. In addition, the item description field has the ability to be viewed by a separate browser which will show the compiled HTML code when right clicked (identical to when adding the item).

If the user chooses to remove the item, the application will decrease the quantity if there is more than one of the same item in the inventory and do nothing if there is an item online. However, if there is only one item, it will check that the item exists online, and if so, it will remove the item from the online store automatically using the RemoveItemCall() method and confirm that a listing has just ended by returning the item reference number for the newly ended item.

The screenshot shows a window titled "Edit Item (4506885401)" with the following fields and controls:

- Item Name [Title]:** New Men's Zoot Suit
- Color/Description:** Black with Tan Stripes
- Style Number:** 201-982-6448
- Quantity:** 1 (with up/down arrows)
- Category ID:** 3001
- Jacket Size:** 40 (dropdown)
- Item Description:** <HTML GOES HERE> (text area)
- Waist Size:** 34 (dropdown)
- Inseam:** 32 (dropdown)
- Jacket Length:** Long (dropdown)
- Style:** Three Button (dropdown)
- Material:** (empty dropdown)
- Link to Picture:** http://URL
- Condition:** New: With Tags (dropdown)
- Store Category:** Men's Zoot Suits (dropdown)
- Auction Type:** Auction (dropdown)
- Listing Price:** \$ 87.00
- Buttons:** "Edit Item" button

If the user would want to edit the item, a form, which is accessible from the view item form and from the main form of the application, would be loaded and have the existing information pre-filled in the appropriate boxes. Once the changes are made and submitted, the application will edit both the data in the local table and the information online to the appropriate online item. If successful, the `ReviseItemCall()` API call will return the item reference number of the item edited and the applicable fees if warranted from the changes.

Selling an Item:

When an item is sold online, there are a number of things that happen. Due to the difficulties in *eBay*'s platform notifications option with the store account, the only manner in which the information on recent transactions is received is when the user requests to synchronize with *eBay* and retrieve data on transactions that have been modified since the last time the application was synchronized with *eBay*. To retrieve this information, the application calls the `GetSellerEventsCall()` function and passes in the appropriate time values as filters. The results can then be organized and sorted to traverse them in order and extrapolate the necessary information to be saved – among which includes specific information relevant to the buyer which is saved in the CUSTOMERS table, such as the buyer's name, email address, the item purchased, address, etc... In addition, the application will store data on the specific transaction in the SALES table such as the item ID, the price, etc... Once the information is retrieved and stored appropriately, the application will remove the item that was sold from the inventory or decrease the quantity where applicable. In addition, if there the inventory contains another identical item to the item that was sold; the application will re-list the item and update the appropriate information in the table such as the new item reference number for *eBay*. This way, we efficiently have the second link to the two part interface that this program initially intended to do.

Because the majority of the work is automated, *eBayer 6.0* still allows for the possibility to edit certain data manually and has support for doing so. Below, we see the forms that

the user can use to edit the store customer information, including adding a new customer, viewing an existing customer, editing that customer, and removing the customer from the table. The same can be done for the information on all sales made in the store.

Customer Name	eBay User ID	Address	City	State	Zip	Email Address
Omar Elangbawy	okn3	221 Jane Street	Weehawken	NJ	07086	omars@seas.upenn.edu
Zuheidi Hoyos	zhoyos	305 Winfield Avenue	Jersey City	NJ	07305	zhoyos@eden.rutgers.edu
Daniel E. Greenberg	deg	318 South 40th Street	Philadelphia	PA	19104	deg@seas.upenn.edu
Gregory Lysko	glysko	3820 Locust Walk	Philadelphia	PA	19104	glysko@wharton.upenn.edu

CUSTOMER INFORMATION

Full Name: Omar Elangbawy
eBay User ID: okn3
Street Address: 221 Jane Street
City: Weehawken
State: NJ
Zip Code: 07086
Country: USA
Email Address: omars@seas.upenn.edu
Total # of Purchases: 1
Total Purchase Amount: 99.99

[Edit Info](#)



There were a number of technical difficulties encountered during the early stages which seemed to recur as the project continued. First and foremost, this project has required me to develop a working knowledge of the .NET environment, in conjunction with C#. In addition, I have needed to learn how to use and organize the necessary database for the application using MySQL. Probably the most difficult task encountered during the early phases was the design and organization of the tables within the database. Once the main objectives of the application were established, then the structure of the database followed in suit. However, the possibility of change within the database still existed as the project went on, due to possible improvements of the organization along the way such as the addition of necessary tables such as application data and ended items. In addition, the change in structure of the *eBay* Developer's Program forced some changes to occur in the structure and design of the application, although the change was beneficial in most cases.

There are, however, certain technical difficulties which I encountered during the second phase of the project which proved to be more difficult. One of the most difficult changes to the *eBay* Developers Program structure that proved difficult to adapt to was the inability to utilize the platform notifications support from *eBay*. Because Fashion house did not have the appropriate subscription level, as well as the inefficiency of the exact method in which the notifications are sent and to be gathered, the exact mechanics of retrieving information on recent transactions had to be designed, resulting in many changes to the structure of the database and the logic in the program, such as what data needed to be saved and returned and how to handle the data received. On top of the structure in which the program was to communicate and receive transaction data, the manner in which the item was to be listed on *eBay* was slightly cumbersome. At first, I

did not have an efficient manner in which to upload and associate an image with a listing using *eBay* picture services. Luckily, Fashion House recently began to save images to URL's unassociated with *eBay* this making it easy to add the URL to the details of the particular item being added.

Once this principle challenge was solved, the last major obstacle included two more basic aspects than the previous issues. The first one was the design of the application to make it as user friendly as possible. The main difficulty was the constant changes coming from Fashion House itself and the principle user to the program once it is finished and ready to go. There were many small details that were important for convenience that proved, at times to be challenging due to the size of the project and its complexity at the time of the request. The second aspect was the manner in which I was to test the application. Because it would be difficult and inefficient to certify the application and test its functions live, as well as exponentially more work because it would require work in uploading all the items with the new formatted HTML, a method using *eBay*'s "Sandbox" environment had to be developed. This not only proved to be time-consuming but also at times quite confusing and frustrating because of the constant technical issues on *eBay*'s end in creating and validating test users. Nonetheless, the challenges encountered were fixable and did not prevent the completion of the end product to the satisfaction of Fashion House Inc.

Timetable:

Fall

- § Gained knowledge of .NET using C#, as well as MySQL
- § Learned *eBay* API functionality for expected use
- § Organization and structure of the application and its components
- § Created tables within the database for use throughout the application
 - § Items
 - § Customers
 - § Coupons
 - § Ability to add, remove, and view items in the database
 - § Application Data
 - § Sales
 - § Ended Items

- § Organized contents of entries in the items table for expected need to add items to *eBay* store

Spring

- § The ability to upload, remove, edit, and re-list items on the *eBay* store
- § The ability to download and sync automatically with *eBay*, making sure that the database is modified and saved correctly
- § Download and track sales information
- § Download and track customer demographic information
- § Ability to sort customer and sales information for increased tracking and maximization of data analysis
- § Ability to save emails to probable customers with coupons
- § Finalized user interface and organization
- § Beta-testing phase
- § Bug-fixes for errors found during beta-testing

Conclusion:

In retrospect, this project was very interesting and insightful into the *eBay* realm. Not only did it prove useful for the needs of the store, but in doing the necessary functions, there were many aspects of *eBay* that allow for a more efficient management of an online store that would have otherwise gone unnoticed. If I could have changed one thing, I would have preferred to utilize *eBay*'s platform notification feature if the store subscription allowed for it and it was more efficient in organization on *eBay*'s part. Aside from this, I feel that the difficulties encountered were normal and expected, and for the most part improved the progress and organization of the project. This application will prove to be very useful and helpful in managing the store from here on out. It will save a lot of time and effort and make it less prone to human error. Thus, in retrospect, I do not regret the choice and layout of my senior design project. Not only does it help for this particular venture, but has allowed me to become familiar with database design as well as desktop application design for whatever I may encounter in the future.

References:

[1] Community Codebase. eBay. Viewed 24, September 2005:
<https://www.codebase.eBay.com/>

[2] Developer Programs. eBay . Viewed 22, September 2005:
<http://developer.eBay.com/index.html>

[3] Developer Zone. MySQL. Viewed 23, September 2005: <http://dev.mysql.com/>

[4] .NET Tutorials. Developers.net. Viewed 21, September 2005:
http://www.developers.net/all_content/more/Tutorials/.NET

[5] Seller Tools. eBay Seller Solutions. Viewed 20, September 2005:
<http://pages.eBay.com/sell/tools.html>