Mutual Fund Batch Update System

By

Dallas Sehlhorst

Submitted to
the Faculty of the Information Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Engineering Technology

University of Cincinnati
College of Applied Science

June 2006

Mutual Fund Batch Update System

by

Dallas Sehlhorst

Submitted to
the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements
for
the Degree of Bachelor of Science
in Information Engineering Technology

Dallas Sehlhorst — Date: June 8, 2006

Tamisra Sanyal — Date: June 8, 2006

Patrick C. Kumpf, Ed.D. Interim Department Head — Date: June 8, 2006

Table of Contents

List of Figures

Abstract

FTJ FundChoice, LLC is a mutual fund servicing company that currently has a need for an improved way to allow clients the ability to trade funds on its Web site. The existing fund trading portion of its Web site only allows the trading of five (5) funds at one time, and it has limited functionality when selecting funds. Also, the company's business rules have changed significantly since the original site was developed. A new Web application fulfilling the new requirements has been developed, giving the user an easier and more efficient way to trade funds. The new site allows the client to trade all of their funds on one screen, and it gives the client an easier way to select funds to trade. The new site was developed using Microsoft technologies: Visual Studio 2005, Visual FoxPro 9, SQL Server 2005 and IIS 6. The programming language used for the Web application was C#, while T-SQL was used for the database queries/ stored procedures. The technical highlight of this project is the use of hierarchal GridViews (putting a GridView within another GridView) to allow easy viewing and manipulation of the client's account, and AJAX (Asynchronous JavaScript And XML) for the seamless selection of new funds.

Mutual Fund Batch Update System

1. Statement of the Problem

A few months ago, my employer, FTJ Fundchoice, required an improved fund trading system that could solve the shortcomings of the solution on its current Web site. The solution I proposed was to fix the limitations of my employer's fund-trading portion of its Web site and design and build new Web pages. These new Web pages should effectively solve the problems the old solution faced and also add two new features that cannot be done on the current site.

Many problems hinder the effectiveness and usability of the current fund trading solution. A major problem is a reliance on a popup browser window to select funds to trade into. This is a time consuming task since a user can only select one fund at a time as well as cumbersome since it requires users to accept popups from our site. Another problem is that users can only work with one account at a time, and many users have more than one account. Numerous users have requested the ability to see and manipulate all of their accounts at the same time. Currently, the user has to select a single account to trade with, then request trades on the individual account, and then go in and select their other accounts and perform the same requests. Since this wastes clients' time, it results in unhappy customer experiences with our trading solution.

A major reason this project has been developed is to add functionality into the trading system to allow advisors to search for client accounts that contain a certain mutual fund. The requested behavior is to allow an advisor the ability to enter a fund to search for, display all of the advisors' client accounts that have this fund, and to perform trading features on these accounts. There is no way to do this on the current

implementation of the Web site. The only way to allow advisors to do this is by having them call our Customer Service Department and requesting our company do the trades manually in our fund processing engine. This places added work on my company's employees, when an advisor could do it on our Web site. A correctly implemented solution will save both the advisor's and my company's employees' time since everything can be automated on the Web site. This proper functionality should allow the advisor to enter these trades into the Web site faster than calling, waiting for a customer service representative, and requesting trades manually.

2. Description of the Solution

The main purpose of this new system is to add a new feature that gives advisors the ability to search for all of their clients that have a certain fund and permit them to trade these clients into another fund. Currently, my company gets requests for this feature many times a week. This system also adds the functionality for advisors to request redemptions on their clients' accounts so money can be taken out of their accounts. In order for clients to perform a redemption, they have to call us and ask for money to be taken out and provide the proper forms and paperwork. This inevitably ties up Customer Service for a large portion of the day. They answer calls pertaining to redemption requests, and have to take time to mail the proper forms to the clients. It also inconveniences clients and advisors because they have to call during business hours and mail in forms, which takes considerable amounts of time. The other functions are on the current Web site, but they have limitations that I plan to correct.

Improvements of the current site include the ability to see and search for mutual funds in a drop-down list. I wanted to model the drop-down list box of the Windows Forms

programming model (see Internet Explorer's Address Bar for an example). In this model, the user can start to type in a mutual fund name or a ticker symbol and see results immediately. Since this is not standard behavior, I knew I would have to do a considerable amount of work to implement something similar. I wanted to make sure whatever solution I ended up making was the most viable to fit the requirements, so I searched for other solutions to the problem of displaying large amounts of data. The problem with designing a drop-down list box is that large amounts of data must be sent to the client during each request due to our platform having over 800 mutual funds, thus requiring longer page load times for clients and more work done on the server. After giving up on the drop-down list box implementation, I decided to investigate other possible solutions. I was determined to not resort to using a popup window again. The solution I ended up using modeled Google Suggest (see http://www.google.com/search?hl=en&q=suggest for an example). This search method allows the user to type a fund name or ticker symbol, and after the first letter is typed, a list of possible selections is displayed. As the user continues to type, the list of possible fund selections narrows down to what the user is trying to type. Then the user can either select the fund by clicking on the name with the mouse, or by using the arrow keys on the keyboard. If the user types in enough information to limit the possible selections to one (1), then they can press enter or tab to select the fund. This solution is far more intelligent than sending the entire fund list to a drop-down list box. The reasons for this are an easier user interface, reduced page loading time, and reduced load on the server. When a user requests a Web page that uses fund trading features, the server doesn't have to load and send the entire list of funds before it can serve the page to the client. This is

what helps reduce the page load time as well as the load required of the server. An easier user interface is acquired through this solution by allowing the user to type in the fund name or fund ticker symbol to find the fund he/she is looking for. The current Web site requires users to click on a binocular icon and type what they are looking for into a popup window. This requires users to have a popup blocking program which allows popups from our site, as well having to put up with the annoyance of working with another browser window. The problem with this is twofold: it takes time to select funds one at a time, and if a user has a popup blocker enabled, then it's impossible to search or even select a fund.

2.1 Features of the Fund Trading System

Ability to transfer money from one or more mutual funds to different mutual fund(s)

- Ability to reallocate deposited money to new funds
- Ability to rebalance the mutual funds in a client's account
- Adds the ability to redeem money from a client's account
- Adds the functionality to allow Advisors to search for all of their clients that have a selected fund and will allow "batch" (trade all of the clients that have this fund) trading from this fund to different fund(s)

The ability to transfer money from one or more mutual fund is called Fund to Fund Transfer. This function shows all of the client's accounts on one browser window, with each account detailing each fund in that account.

The second main feature of the system, called Allocation for New Deposits, lets the user reallocate future deposits into different funds. This feature does not change existing funds; it only changes the user's allotment of funds for new deposits. A similar feature called Rebalance performs the same functions as Allocation for New Deposits as well as

changes the user's existing account funds. This feature changes the user's entire account to match the user's selections.

The final two features of the system are for advisor access only. While advisors can access the client features as well (after they select a client to perform trades against), they need additional features clients do not have. One special feature allows advisors to search all of his/her clients' accounts for a given fund so the advisor can trade all of the clients with the selected fund into other funds. This is called All Client Transfer. The second feature permitted only to the advisors is called Redemption. This lets advisors request money to be withdrawn from a client's account.

2.2 User Profiles

It was the goal of this project to add the functionality of this system to the current Web site. However, this is not possible without a considerable amount of work due to the current Web site's use of classic Active Server Pages ASP and the new solution using ASP.NET 2.0. A simple way to accomplish this is to pass the required values the new solution needs, such as the account number and advisor ID, using hidden input forms. Another solution would be to use query string parameters sent along with the Universal Request Locator (URL) string. While these solutions seem practical, they are not secure because the values are passed in plan text to the browser which would make it easy to determine the information passed into the new system. To test and demonstrate my solution, I made a mock Web site in ASP.NET 2.0 that resembles what the next version of my company's Web site might look like, minus the extra functionality of the complete Web site. With this said the user profiles may change as the Web site is finished. The users of this system will be required to login from the mock Web site. The site will

determine the user type and set the user's role accordingly. The two users of the system are the Client, which has access to only his/her account, and an Advisor/Manager, which has access to any of his clients' accounts.

Client

Clients have the ability to request changes on their accounts in a few different ways. First, they can select a Fund-To-Fund transfer which will allow them to select a fund that they already have in their portfolio and put that money into a different fund. Another function is Reallocation. Reallocation determines how new deposits to the account gets dispersed among funds. The third function clients can perform is the ability to rebalance their accounts. This allows them to change both their existing funds and the allocation for new funds at the same time. Clients will not be given access to the advisor features such as Redemption and All Client Transfer, since they should not be allowed to view any account but their own.

Manager / Advisor

Managers and Advisors have the ability to mimic a client and do everything a client can do. The only difference is they must first select a client on which to perform the trades. Then, the functionality is exactly like the clients'. The additional features added to this Role, All Client Transfer and Redemption, give the advisors access to only needed special features.

2.3 Design Protocols

The overall design of this system is a typical three-tiered solution. The presentation layer consists of the actual views on the individual Web pages. The lines of code in the Web pages have no business rules logic in them. The actual business rules

logic is implemented in classes that the presentation layer calls upon. Since this solution uses two databases, I decided to make two classes that individually handle the separate databases. The final layer is the actual database layer. This layer accesses the databases and retrieves, updates, and deletes records. The code in the database layer is written in such a way that if our company decides to go to SQL or a different database provider, no changes in the code will need to be made. The only change required of this solution is a single line in the Web.config file. This file includes human readable XML which is easily changed in any text editor. To use a different database, the XML needs to be modified to reflect a different connection string. While this may sound immediately trivial, it has a huge benefit for my company as more developer time can now be focused on other tasks instead of rewriting database access code.

2.4 Use Case

The use case diagram can be seen in Figure 1. The use case depicts two actors, the client and the advisor and displays the functions they are allowed to access. The relations between each use case and the paths are also shown. The use case shown outside the system is my company's legacy fund trading system called Phoenix. Phoenix creates and handles the trades as well as manages every account. My system interacts with Phoenix by storing the requested trades into a database and then the backend processing programs already in place send Phoenix the trade information in a flat, comma delimited text file. I do not have to change or modify any of the programs that create the required text files for Phoenix.

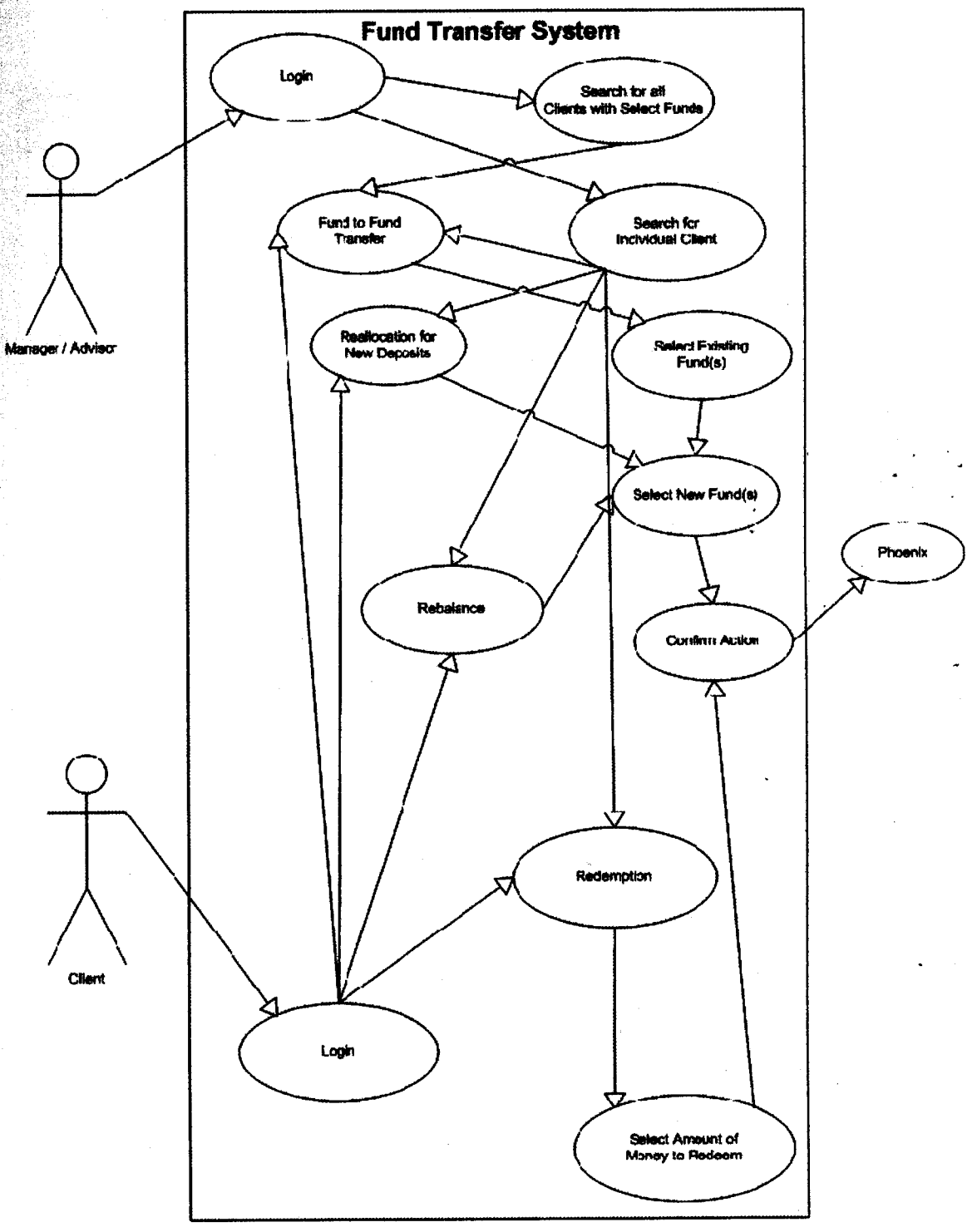

Figure 1. Use Case Diagram

2.5 Class Diagram

The class diagram of my project is shown in Figure 2. The seven (7) main classes

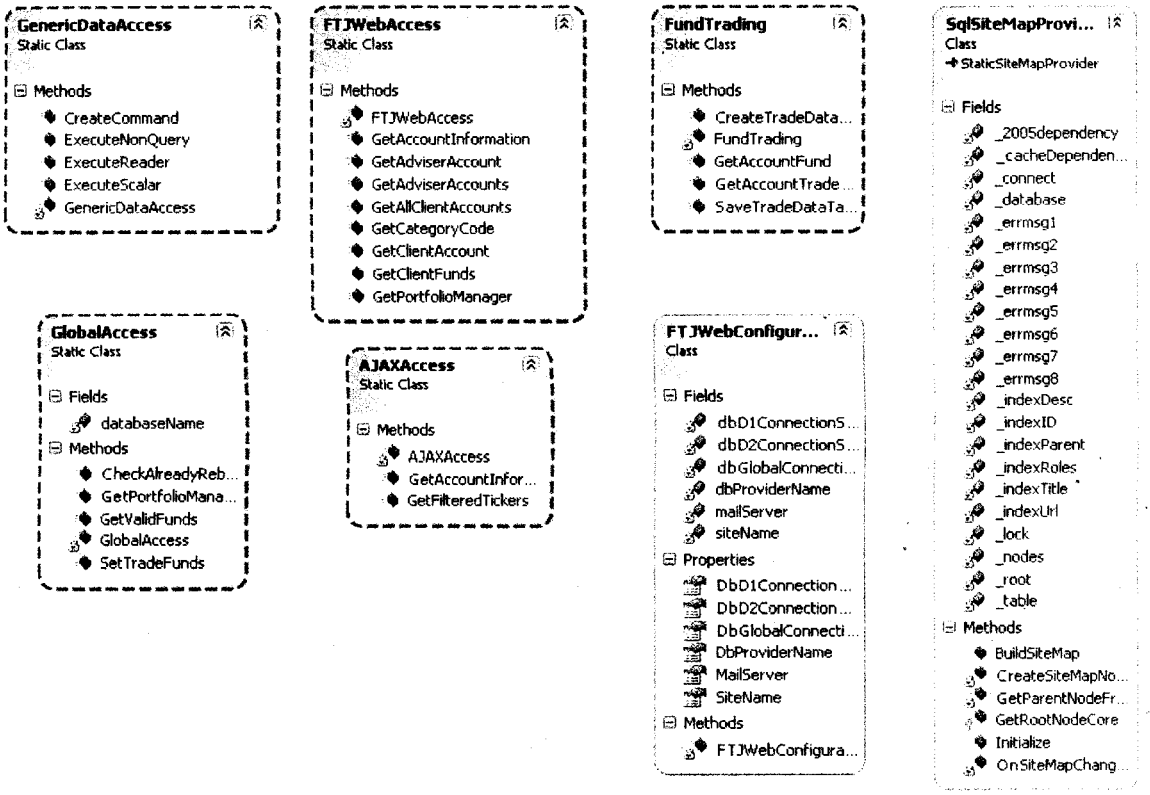

Figure 2. Class Diagram

are shown in this diagram. Since I decided to make most of my classes using the static modifier, the diagram depicts these classes with a dashed line around them. The other classes are instance classes and have a solid line around them. The reason I chose to make most of my classes static is for performance. Since these operations will be performed many times a day, having one global instance of a class always created is many times faster than having to create and destroy an instance class. This solution also reduces the memory requirements of the server since only one class is instantiated (versus having a class created for every current user accessing the system. The remaining classes are seldom used, so I felt they could be better not static.

The classes listed in Figure 2 comprise the entire application. The GenericDataAccess class is responsible for setting up data connections given the database name, and also executing the commands to get results from the database. The system does not use this class in the application layer; it only uses this class from the business layer, such as FTJWebAccess and GlobalAccess. The FTJWebAccess and GlobalAccess are the two business classes that interact with the two databases this system depends on. These business layer classes get client account numbers, get client funds in an account, get all advisor accounts, and so on. The FundTrading class is what makes the entire fund trading system work properly. This class creates a DataTable object that is used to hold the new funds the user is requesting, and saves this same object to the database after checking it for proper trades. The SqlSiteMapProvider class is used to drive the mock menu system I created. This class overrides the static SiteMapProvider class .NET ships with by allowing the use of SQL Server to store the menu data versus the default behavior of having to store the menu data in an XML file. The AJAXAccess class helps drive the AJAX functionality on the Web site by retrieving the proper data from the two databases this system interacts with. The AJAXAccess class is not used by the individual Web pages that show the AJAX behavior with the search textboxes, but rather by the ASP.NET pages that send XML to the XMLHttpRequest object on the individual pages. This will be highlighted further in the Proof of Concepts section. The final class is used to drive the setup of Web site by providing access to the e-mail server name, as well as, various other information such as the database provider name and title of the Web site.

2.6 DataTable Diagram

I chose to show a diagram of the temporary representation of trades in memory using a

DataTable object because this is technically the only form of data representation that I had to create. This DataTable is named UserSelectedFunds and is shown in Figure 3. All this object does is temporarily store the trades the user is currently working on. The primary keys that make the rows unique are highlighted with the PK modifier. The databases that the system interacts with to get account data and other information have already been created. I choose not to diagram them for security reasons and because there is no relationships between tables or primary keys on any of the tables.

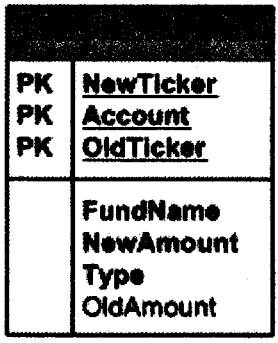

Figure 3. DataTable Diagram

2.7 User Interface

The user interface for this system follows the familiar tabular data display used on many Web sites today. Since this system requires display of large amounts of data I decided this would be the best possible implementation.

2.7.1 Interface Design

The interface design uses the ASP.NET GridView Control to display the data in a tabular format. The approach I used to display data was to use three (3) GridView controls on each page that displays account information to the user. The GridView controls are actually nested inside each other and they are created at runtime when the data is requested. I decided to use this technique because it results in a more dynamic

user interface and I believe it is easiest to use. A special effect of doing the interface this way is that more nested GridView controls can show, depending on the number of accounts the client has or the number of funds in each account. For example, if a client only has one account, then the main GridView control will only have one other GridView control in it, with another GridView control nested inside for each fund in the account. If the client has three accounts, then the parent GridView control will have three other GridView controls nested inside the parent control, as well as a GridView control for each fund in every account. This is what gives the user interface the dynamic interactivity since the interface will change dramatically depending on the number of accounts displayed.

2.7.2 Color Scheme

I chose an arbitrary color scheme for this project. The reason for this choice is that my company has not decided on its new marketing material yet. Once a decision is made and final marketing material is delivered, I can change the color scheme of the various controls and tables using a new feature in ASP.NET called Themes. Themes are basically a global Cascading Style Sheet (CSS) file that works on ASP.NET server controls as well as regular HTML markup. This means that I can change the row color, selection row color, alternating row color, header color, and so on for every GridView control using a single Theme file, and without touching the source code for the pages that use these controls.

The current color scheme is a blue color that matched a stock image my company had for use on its Web site. This image is a globe that I edited in Photoshop to remove the background from the image. This image can be seen in the upper right-hand corner of

the Web site. Every other color, such as the header color gradient and the menu system color, was chosen to closely match the color of the globe.

2.4.3 Overall Design Scheme

This section of the report details various other design ideas that were not already discussed in other parts of this document.

Database Design

The database design is straightforward, and they are already created before this project began. There are basically two main databases that contain various tables that maintain the business data. The reason for two databases is that one database contains the information needed for the Web site, such as user IDs, passwords, addresses, and so on. The other database contains information about the mutual funds such as the values of the funds, who owns which funds, fund trades, and so on. This second database gets destroyed and recreated every night to reflect the nightly trades. There are no relations or primary keys in any of the tables. Some tables have indexes on them, but this is mainly for views. I did not design the existing databases and many legacy programs rely on the current implementation of these databases so I am not permitted to change or go in and start adding primary keys or table relations. I am told the reason for not having primary keys or table relations is because the database that gets recreated every night is driven by a flat-file source. Also, information repeats itself in the original data store (the flat file source) so primary keys would cause a problem.

AJAX Design

The user interface element that allows users the ability to search for funds or accounts and automatically suggests information is implemented using Asynchronous

JavaScript And XML (AJAX) to accomplish this behavior. When a user begins to type a letter into the textbox, JavaScript on the page sends the character value to another ASP.NET page using the browser's XMLHttpRequest object. Once the ASP.NET page that handles these special requests receives the proper information, it sends out eXtensible Markup Language (XML) back to the XMLHttpRequest object that requested it. The JavaScript on the requesting page then formats this response into a DIV element on the page to display the suggestions to the user. If the user is not satisfied with the suggestion and types another letter, then the additional information is sent back to the ASP.NET page that handles the AJAX requests and repeats the behavior over again. Once the user is satisfied with the suggestion he/she can press tab, select it with the mouse, or can select it with the keyboard using the arrow keys. The JavaScript on the page fills a hidden form input with either the ticker symbol or the account number depending on where the AJAX textbox is used. The hidden value is not shown to the user; it is used by the system to track which ticker the user wants to select for trading purposes. This solution reduces having to query the database again to get the ticker symbol given the fund name or to get the account number given the account holder's name since the value is submitted in the hidden input form field.

Implementation

ASP.NET 2.0 and FoxPro were used to design and test this project. ASP.NET 2.0 is the architecture used for the Web site, while FoxPro is the backend processing program and database. The reason for using FoxPro is simply because it is the current data store in the company. That variable cannot change. I used C# 2005 for the programming language.

The reason for choosing Microsoft products over competing products is simple: my company currently pays for and has access to the Microsoft Developer Subscription Network (MSDN) to acquire Microsoft software. This allows us to develop, test and deploy solutions first using Microsoft software without having to pay for the large upfront costs until the system goes live.

3. Deliverables

The deliverables of this solution compose of the entire mutual fund trading system. This system comprises of ASPX files and their associated code-behind files. This solution also has various JavaScript files and other HTML related files such as Cascading Style Sheets (CSS) files and Theme files. To summarize, the following were identified as deliverables throughout the lifetime of this project:

- Easy to use, intuitive User Interface that allows trading of more than 5 funds at a time
- AJAX interface to allow the seamless selection of mutual funds during the trade process
- The inclusion of the three existing trade features (Reallocation, Fund Transfer, Rebalance), as well as the two new ones newly developed (Redemption, All Client Transfer)
- SQL database that mirrors the existing Fox Pro database
- A confirmation page that allows users to verify trades and submit changes if necessary
- The Web site will use similar features across different pages to allow for an easy learning curve
- A solution to the cumbersome popup ticker search box used by the current trading system. *The solution delivered uses AJAX to make the user interface more* interactive by reproducing behavior of desktop applications, making the system easier to use

- The system delivers the ability to send E-mails to the employees in the company who handle and monitor the trading activity
- The Web site also implements basic security to prevent clients from seeing accounts that are not theirs, and by allowing advisors access to only the client's accounts that they manage
- The system delivers proper business logic to prevent trading of more than a 100% of a fund, more than one trade a day, and so on

4. Design and Development

The planning for this project took place in early January of 2006. After interviewing my manager, Mr. Timothy Morehead I was able to determine the amount of work required and most importantly, the budget was formed.

4.1 Project Schedule

The schedule for this project went as planned because I was able to complete this at work. If I felt I was falling behind on tasks I would catch up on the nights and weekends to meet the deadline of each task. Figure 4 shows the project's Gnatt Chart

ID	Task Name	Start	Finish	Duration
1	Complete Project Timeline	4/3/2006	4/6/2006	4d
2	Test User Interface	4/6/2006	5/1/2006	18d
3	Adapt User Interface Suggestions	5/2/2006	5/24/2006	17d
4	Test Trading Functionality	4/20/2006	5/10/2006	15d
5	Update / Fix Trading Functionality	5/11/2006	5/24/2006	10d
6	Begin Final Report	4/10/2006	5/12/2006	25d
7	Send Final Report to Writing Lab	5/15/2006	5/16/2006	2d
8	Finalize, Turn in Final Report	6/1/2006	6/8/2006	6d
9	Tech Expo	5/19/2006	5/19/2006	1d
10	Present Final Project	6/8/2006	6/8/2006	1d

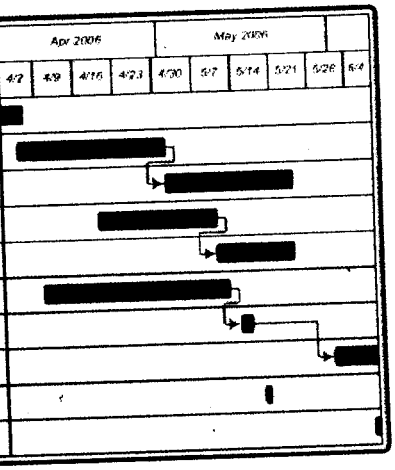

Figure 4. Gnatt Chart

and details the timeline of this project. As shown on the chart, the task that took the longest amount of time was implementing the AJAX functionality. Some other tasks that

took considerable amounts of time to complete were the designing of the User Interface of each Web page and the Master Page mock site. While Master Pages are not hard to implement, it broke existing functionality of the Web pages designed before it, especially the pages that used AJAX. The reason for this is that Master Pages rename every control on the inherited page and since the JavaScript AJAX uses calls the controls by name, the old control names became invalid.

4.2 Project Resources

The resources this project required involved a development computer, various development software packages such as Microsoft Visual Studio 2005, Microsoft FoxPro 9.0, and SQL Server 2005. I was the only developer on this project. As far as external resources go, I used an image file that was already created before I started this project. I did, however, edit the image to best fit the current design. When this system goes live, production Web servers will be required as well.

4.3 Project Budget

The budget for this project (see Figure 5) did not exceed estimates. All of the physical resources were already obtained before work on the project began. The only variable is the development time, since the solution is not live yet, that aspect of the budget may change.

All of the required software is included in the MSDN Universal Subscription and allows for use of the included software free of royalties until the solution goes live. Once the solution goes live the royalty fees are handled by the Light Edge Servers since my company is letting Light Edge host, manage, and provide the SQL database.

Item	Cost
MSDN Universal Subscription	$3,799.00
Light Edge Servers (cost per month)	2,499.00 x 12(for a year)
Development PC & 19" LCD Monitor	699.00
Development Time & Support	4,000.00
Total	**$42,486.00**

Figure 5. Budget

5. Proof of Design

To prove that my solution solves the problem statement I have included screen shots of the application and the default behavior of the system. In each section I also highlight the proper user input. To select a trading feature, the user must first select it from the menu system located on the top left of the Web site. This is shown in Figure 6.

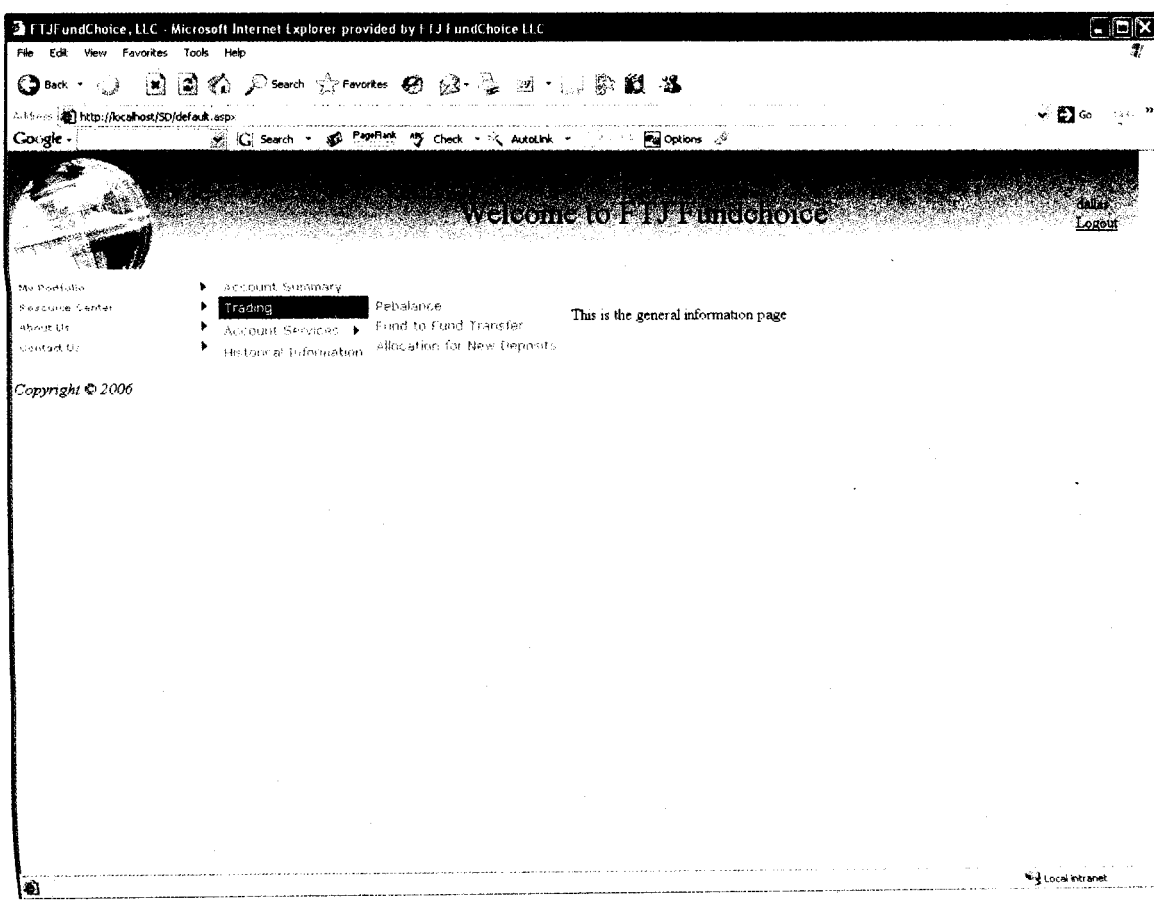

Figure 6. Selecting Trade Feature Using Menu System Screenshot

First I will go over the Fund to Fund trading feature. When either a client selects this, or after an advisor selects a client to perform trades on, the user will be shown a screen similar to Figure 6 that shows all of the user's accounts. From this screen the user can select an account to work with by clicking on the Select button. By pressing the Select button the system will know which fund the user wishes to trade since there is a Select button for each fund. The user will then be shown a screen similar to Figure 7 that just

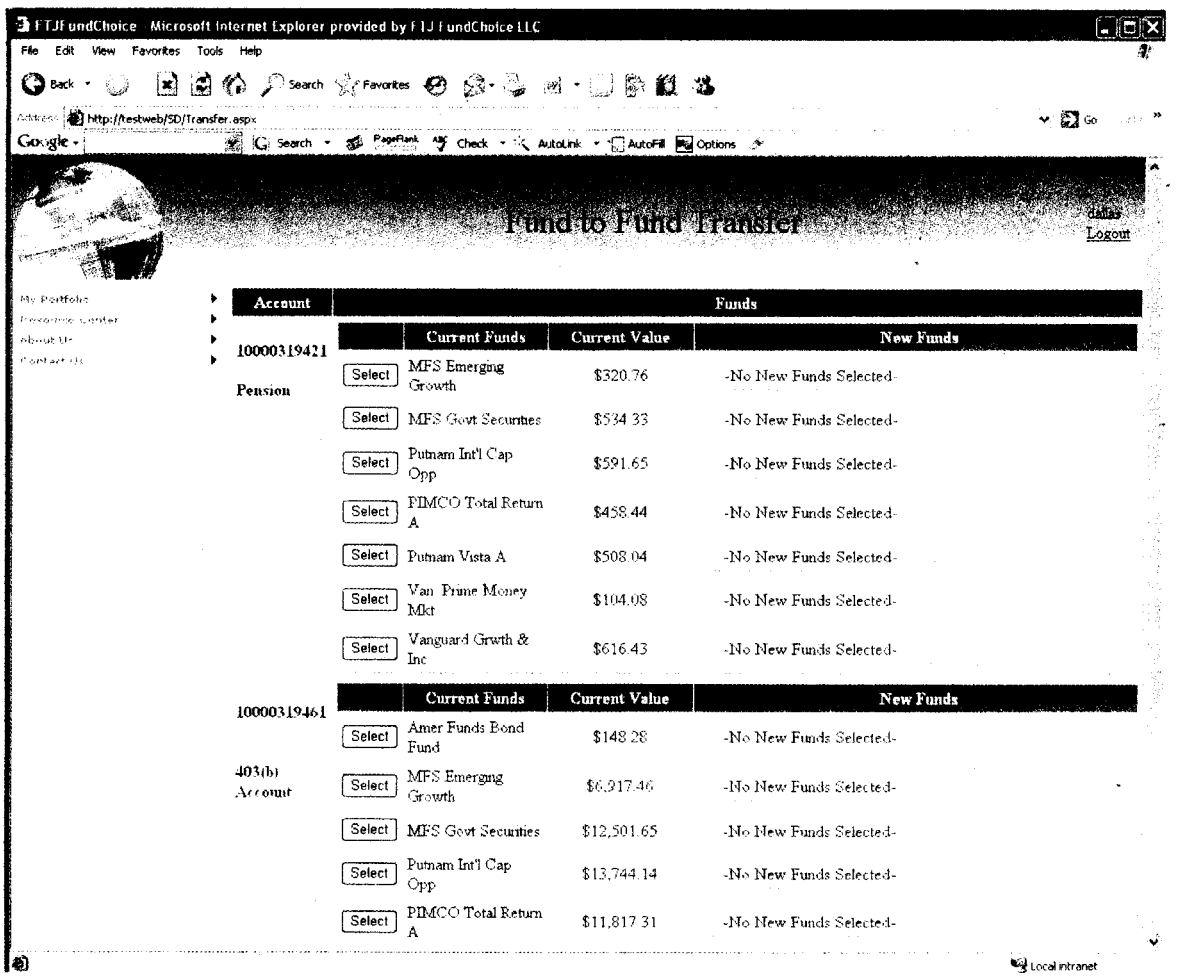

Figure 7. Client Accounts Screenshot

shows the account, the single fund the user selected, and the trades the user requested. This is done to isolate the fund so the user knows exactly which fund he/she is acting on. To add funds to trade, the user simply starts to type in the textbox labeled "Please type

ticker symbol or fund name:". The user can then begin to type into the textbox the fund they want to trade into using either the fund name or the ticker symbol. The system will use the AJAX functionality to show a list of possible funds they can select from, as shown in Figure 8. Once the user selects a fund, he/she then types the percentage amount

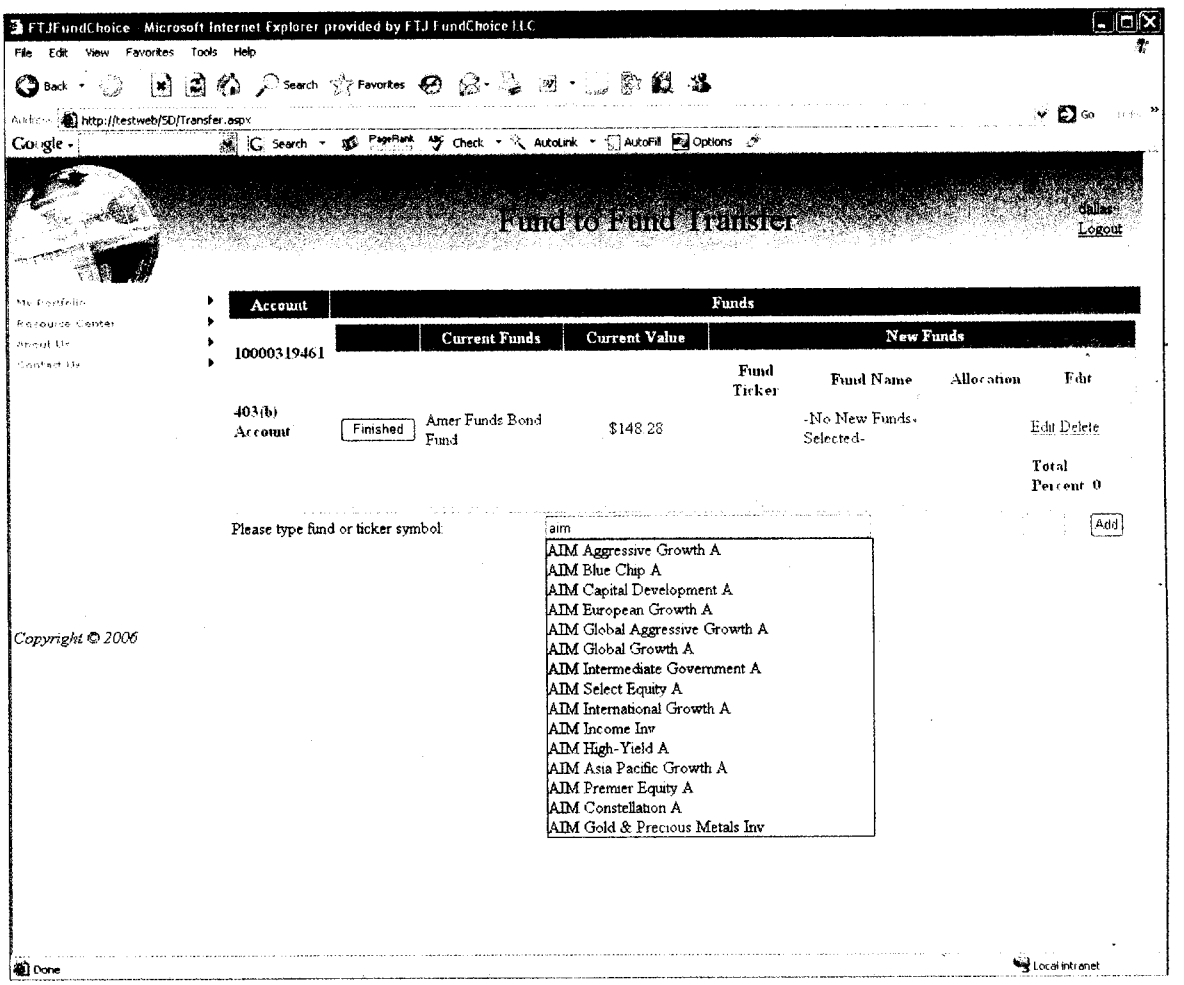

Figure 8. Add Fund Screenshot

they wish, and clicks on the Add button. Then the fund with the requested percentage will be added to the New Funds portion of the display, as shown in Figure 9. If the user wants, he/she can select another fund using the same procedure. When the user is finished with trading from this fund, he/she must click on finish to complete the trading of the selected trade from fund. The user will then be shown his/her accounts again, but

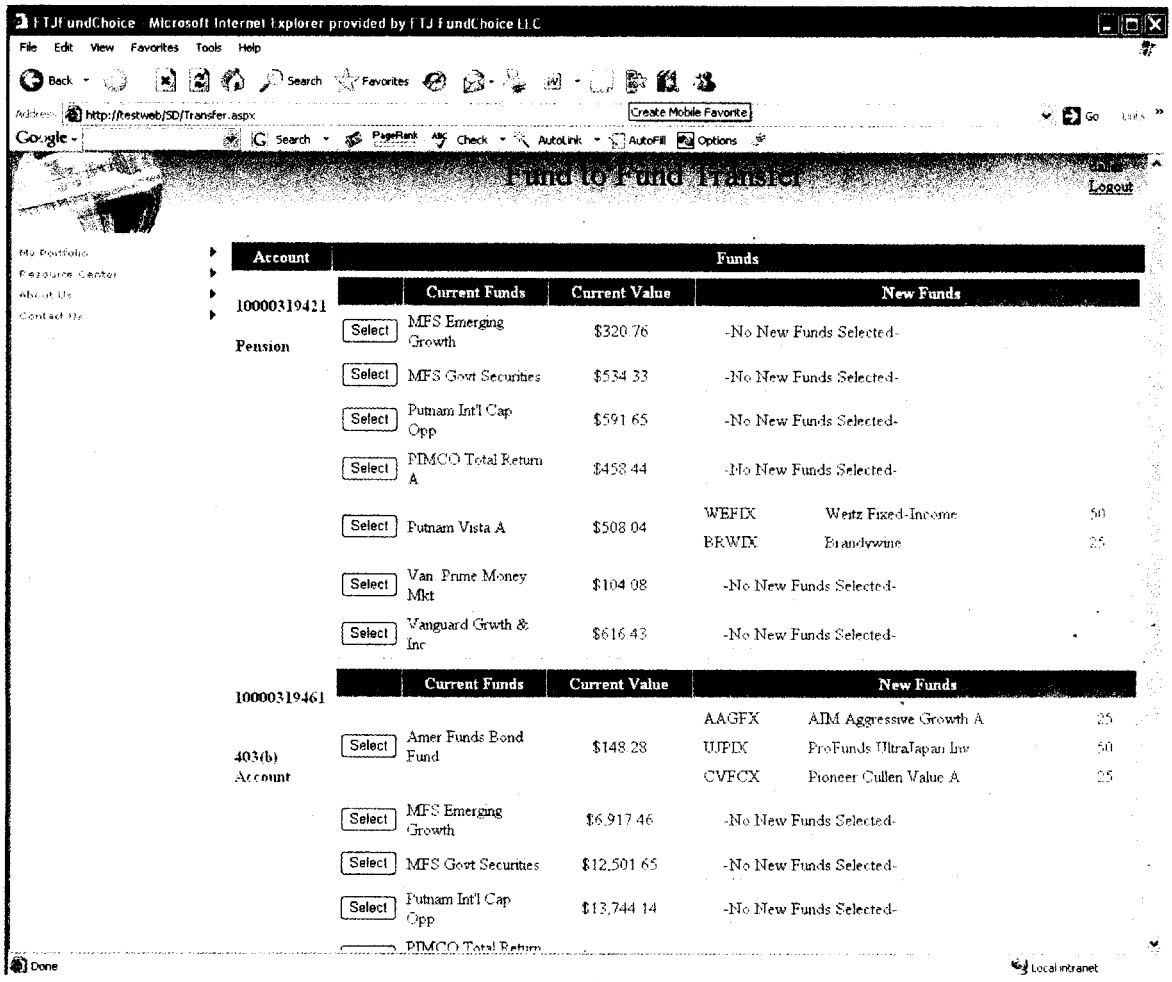

Figure 9. Funds Traded Screenshot

this time the system will show the trades the user selected previously. The user may now select another fund to trade out of, and repeat the behavior above. Once the user is satisfied with his/her selections, they click the Save Funds button on the bottom of the screen to signal that trading is complete. The system will then take the user to a new Web page that shows just the account(s) and fund(s) that have trades placed against them, as shown in Figure 10. From here the user can decide to confirm the trades by clicking the Confirm button, or Cancel the trades by clicking the Cancel button. If the user clicks the Confirm button, then the trades are saved to the database for processing. If the user

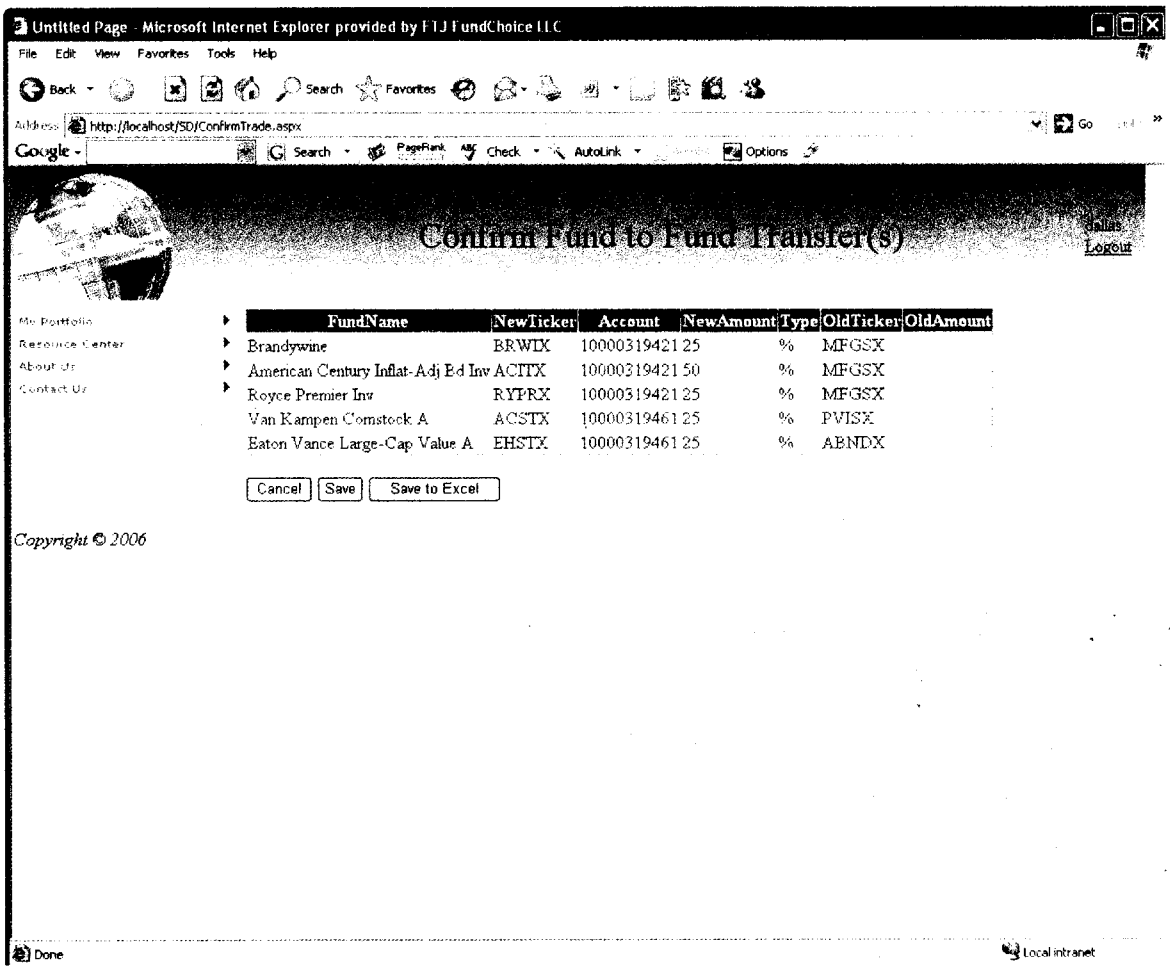

Figure 10. Confirm Trades Screenshot

presses Cancel, then the in-memory representation of the trades is destroyed and the user is notified of their action.

The remaining two features clients are allowed to use involve Allocation for New Deposits and Rebalance. These two features have the exact same user interface; the only difference is how the trades are handled by Phoenix. The difference is that Rebalance will change the user's current fund allocation as well as new deposits into the account. These two features do not change or trade individual funds, they act on the entire account. Figure 11 shows the Rebalance screenshot. From here the user can select the account they want to Rebalance or change new allocations by choosing the Select button.

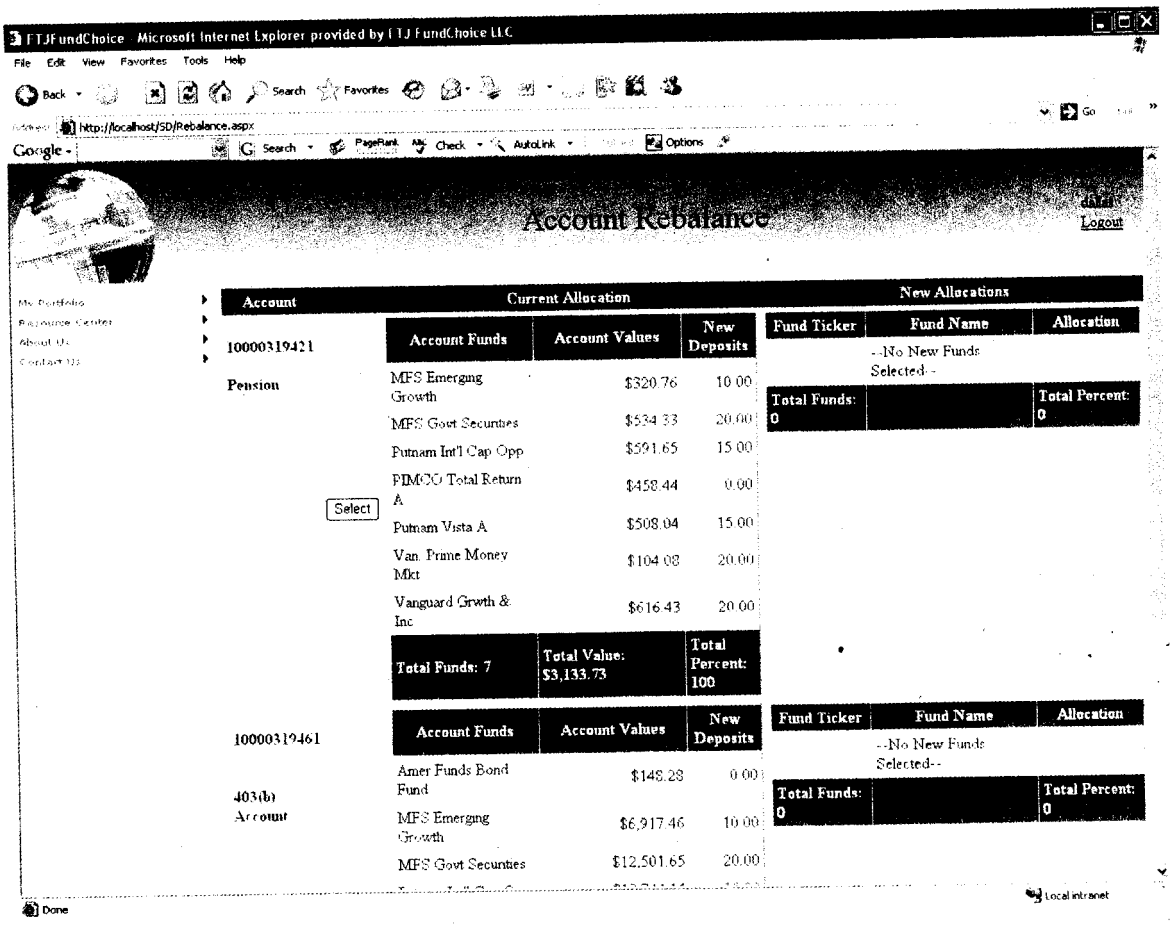

Figure 11. Rebalance Trade Screenshot

The system will then show the user a similar view with only the single account shown. From here the user can type in a fund and percentage they same way as in the Fund to Fund transfer system. The AJAX functionality is present on these features as well so it handles the shortcomings of the current Web site by providing a better way to select mutual funds. Here the user continues to select funds and percentages until the total is 100%. The percentages of the funds must total 100% in order for this to be considered a valid trade, so the system checks for proper values. Once the user is satisfied, he/she then clicks on the Finished Editing button to signal the system that trading for this account is complete. The system will show a screen similar to Figure 10 again; instead this time the

New Allocations will reflect the trades the user just selected. If the user wishes, he/she may continue on to trade a different account by following the same procedures listed above. This solution is powerful because it shows the user all of his/her accounts on the same screen, thereby eliminating the pitfall on the current site of only being able to work with a single account at a time.

As in the Fund to Fund function, the user must press Save Trades to signal the user is finished trading. The new screen is similar to the Confirm screen shown in the Fund to Fund function.

The advisors have the ability to do all three trade features the clients can, as well as an additional two. The only extra step involved is that the advisor must first select a client to perform the trades against. Once a client is selected, the functionality is the exact same as it is for clients. An example of the Advisor Client Search is shown in Figure 12.

The main reason this system was developed was to allow advisors the ability to search through his/her entire client base to find a chosen fund. The reasons for this vary, such as trading out of an underperforming fund, the fund may have closed, or the advisor may feel the money may be better off in a different fund. This functionality is called All Client Transfer, and is highlighted in Figure 13. This figure shows a textbox that the advisor must enter a fund name or ticker symbol into to begin the search process. AJAX functionality is built into this textbox as well. Once the advisor does this, they click the Search button and if the user actually has accounts that have the selected fund the advisor will be shown a list of accounts with the fund in a GridView control, as shown in Figure

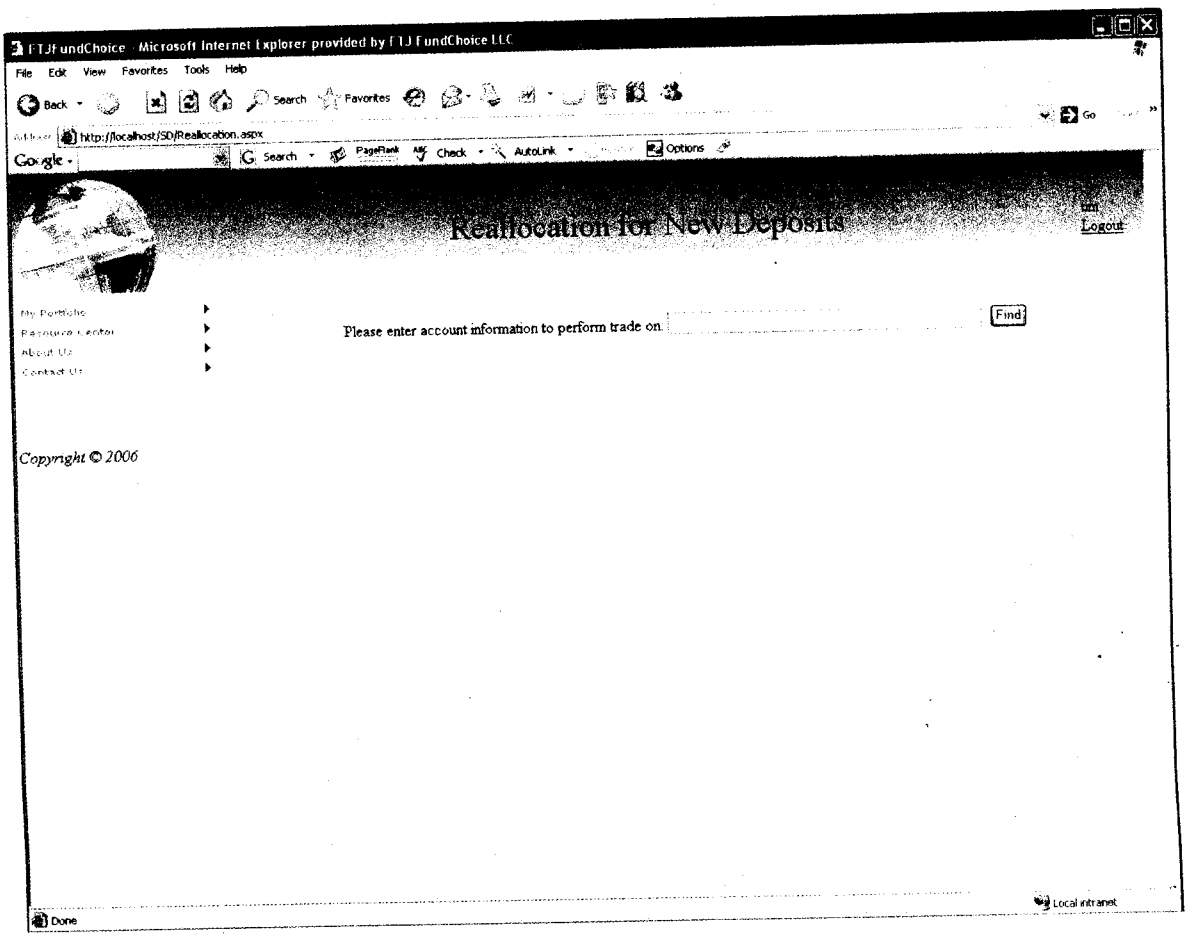

Figure 12. Client Search Screenshot

14. If no accounts are found with the selected ticker, then a message will be shown informing the advisor of this.

Now that the advisor has a list of all the accounts they control with the selected fund, they can now perform global trade features on all the accounts. This means that whatever funds and percentages they enter into the textboxes above the GridView will be entered into all of the accounts. This is the desired behavior. If the advisor chooses, he/she may click on an individual account to trade different funds. We do not anticipate

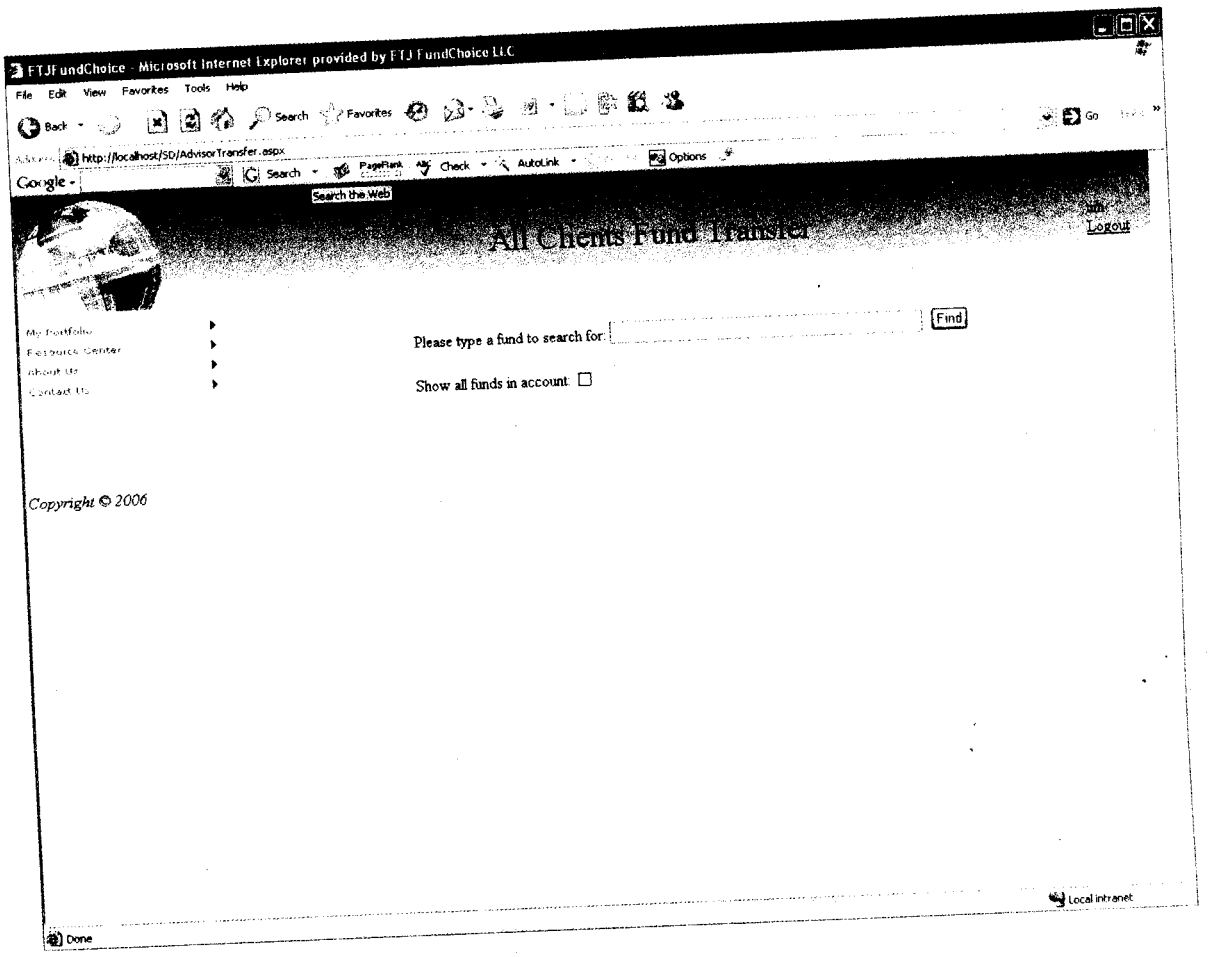

Figure 13. All Client Transfer Screenshot

many advisors using this feature, but it's there if the need arises. From here the functionality is the exact same as the Fund to Fund trade features.

The final functionality is called Redemption and it is highlighted in Figure 15. This page simply gives the advisor the ability to request money to be withdrawn from a client's account. The advisor just selects a client to act on, selects which user account to withdraw money from, and enters the amount to withdraw in the last textbox. Then the advisor clicks the Request Redemption button to send the request to one of my company's employees that handles these types of requests.

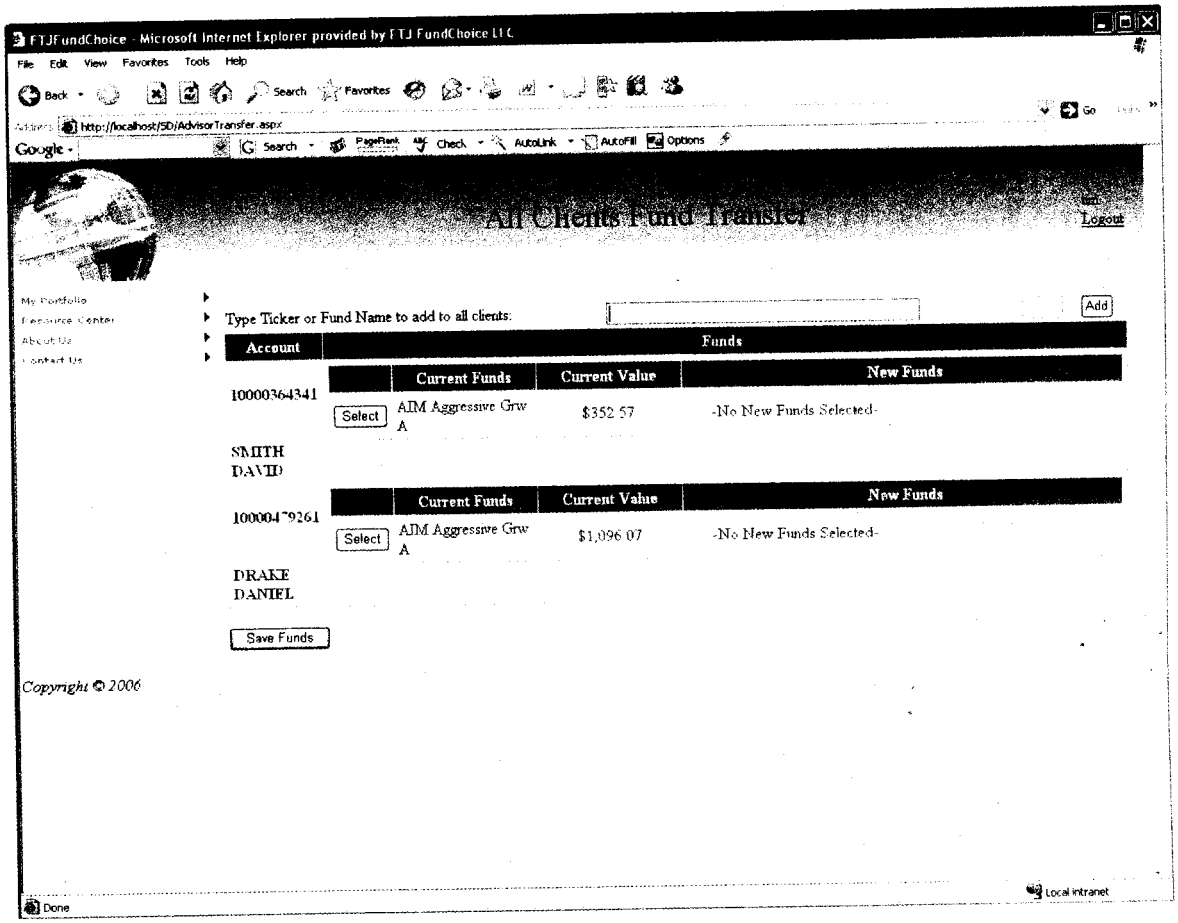

Figure 14. Advisor Accounts with Selected Mutual Fund Screenshot

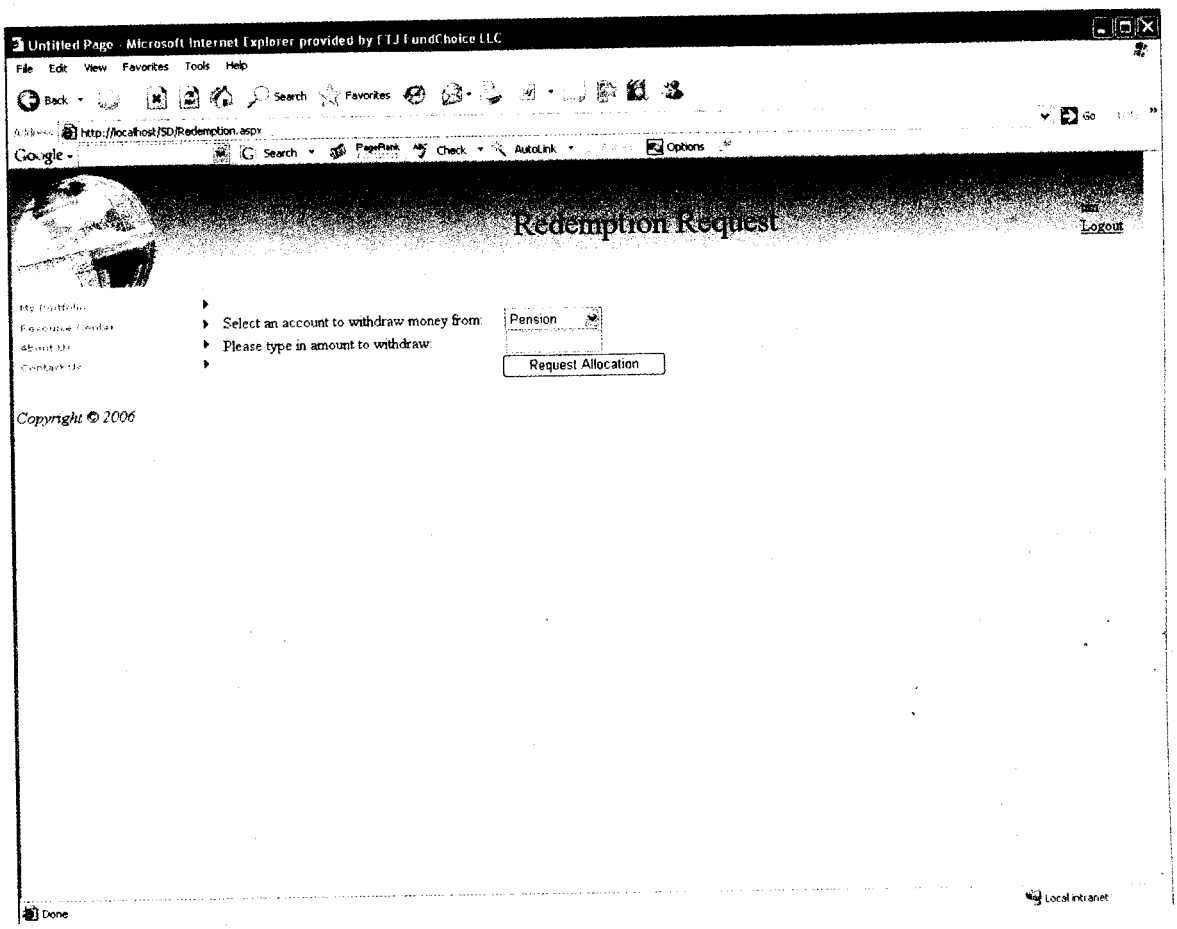

Figure 15. Redemption Screenshot

5.1 Testing

The main testing that I did for this system was basic test-driven development testing in which I tested the functionality and for correct behavior while developing the system. I have shown it to a few end users. My manager thinks the system is an affective solution. I have done user acceptance testing and business acceptance testing and it has proven to be an acceptable solution.

For unit testing I used a program called NUnit. This program requires that you write tests that verify your method works properly. It does this by sending the method test data, and checks the return results. This process is effective because it also allows

the use of test cases that should fail, and if the code written fails when it should and passes when it should then the method should be properly designed.

Another feature I used to test methods is Object Test Branch, which is built into Visual Studio.NET 2005. To test methods using this VS.NET feature you have to create a class diagram in VS.NET and right click on the class. A menu will pop up, and if Invoke Static Method is selected, a model box opens up. This box, shown in Figure 16

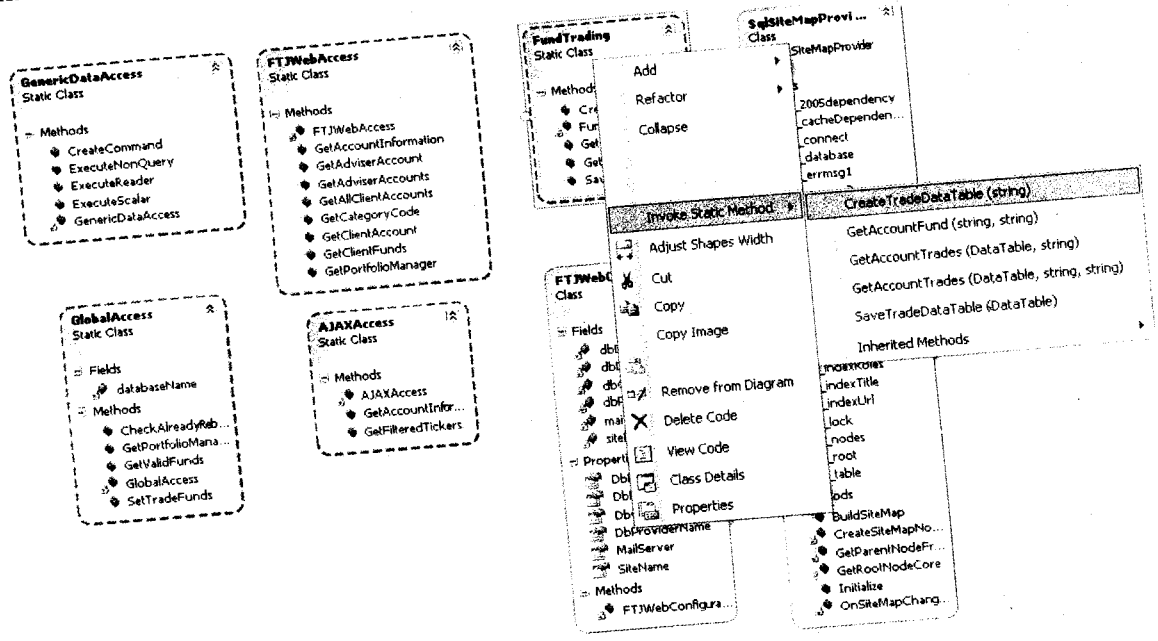

Figure 16. Object Test Branch in VS.NET 2005

allows the developer to put in values and inspect the return result without having to step through lines of debug information, or without requiring the developer to send results to the browser using Response.Write().

6. Conclusions and Recommendations

6.1 Conclusions

This system was developed to enhance my company's Web persistence, especially the mutual fund trading system. This system completely solves the limitations

the existing solution has. It also adds two additional features that will increase the value of my company's services. This solution eliminated the popup requirement, and it devised a clever solution to allow selection of funds. It also allows the ability to trade more than 5 funds at a time, as well as perform trades against more than one account at a time.

This solution was developed using Visual Studio 2005, with C# used for the programming language. I used NUnit extensively to test the code methods while I was developing the solution. Since our existing data store was a Fox Pro database, I had to write the code to access the current data store. I did write the code in a database agnostic manner, and for demonstration purposes during Senior Design II I used SQL Server 2005 as the database.

6.2 Recommendations

This project took longer than I expected, but the primary reason is that I used this as an excuse to learn the new features of ASP.NET 2.0. The GridView control is leaps and bounds more advanced than the ASP.NET 1.1 DataGrid control is. I used the additional functionality in the GridView control by doing the 3-tiered design.

A recommendation for the next version (and the version that will actually make it to the official FTJ FundChoice.com Web site) is to use an expand/collapse GridView control. This can be seen in Figure 17. The reason for this recommendation is that it allows uses the ability to select the fund or funds they want to trade using the familiar expand button. This GridView representation also affords more screen real-estate by compacting funds the user doesn't wish to trade.

For the most part, everything went as planned with this project. A surprisingly

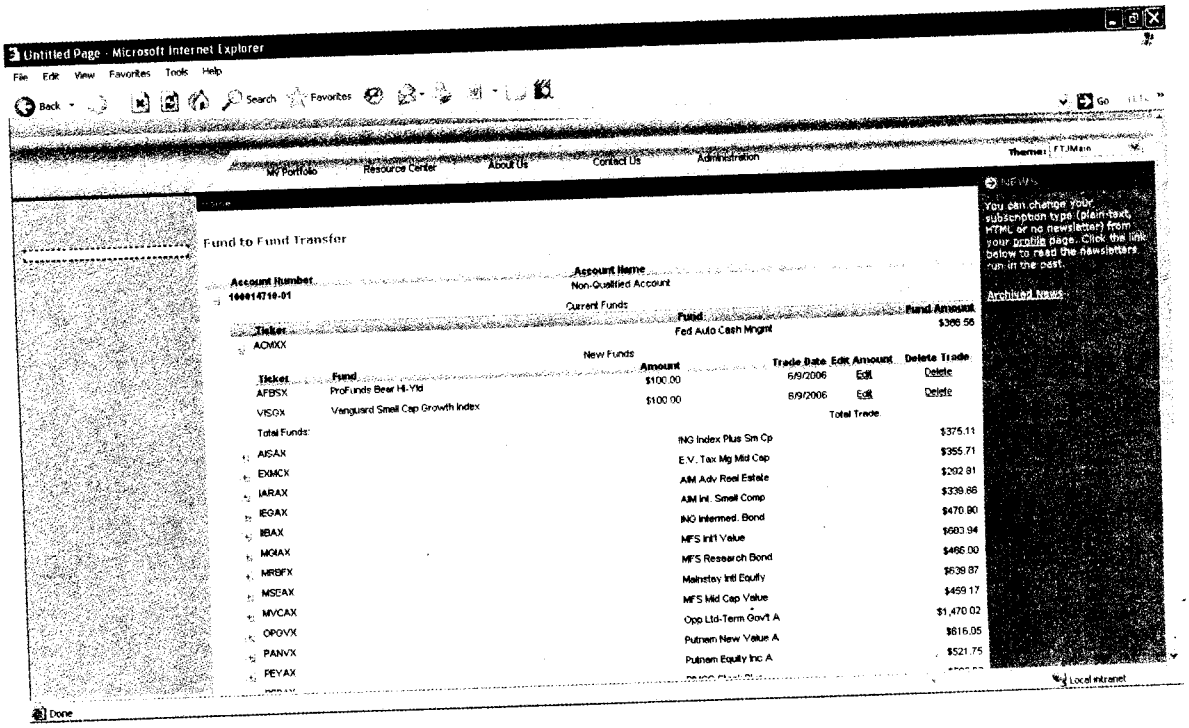

Figure 17. Mutual Fund System 2.0 Screenshot

large percentage of users that tried this system out thought it was far better to use, as well as more intuitive. For that reason, it will go on the new version of my company's Web site.

References

1. Esposito, Dino. *Introducing ASP.NET 2.0.* Redmond, Washington: Microsoft Press, 2004.

2. Jones, Allen. *C# Programmer's Cookbook.* Redmond, Washington: Microsoft Press, 2004.

3. Kittel, Michael and LeBlond, Geoffrey. *ASP.NET Cookbook.* Sebastopol, CA: O'Reilly, 2004.

4. Lee, Wei-Meng. *ASP.NET 2.0: A Developer's Notebook.* Sebastopol, CA: O'Reilly, 2005.

5. Leinecker, Richard. *Special Edition Using ASP.NET.* Indianapolis: QUE, 2002.

6. Lowe, Doug and Murach, Joel. *Murach's ASP.NET 2.0 Upgrader's Guide in C#.* Fresno, CA: Murach, 2005.

7. Lowe, Doug and Murach, Joel. *Murach's C#.* Fresno, CA: Murach, 2004.

8. MacDonald, Matthew and Szpuszta, Mario. *Pro ASP.NET 2.0 in C# 2005.* New York: Apress, 2005.

9. Morehead, Tim. IT Manager, FTJ FundChoice, LLC. Personal interview. October 25, 2005.

10. Sceppa, David. *Microsoft ADO.NET.* Redmond, Washington: Microsoft Press, 2002.