The Knowledge Fusion Resource

By

Phaedra C. Romano

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

University of Cincinnati
College of Applied Science

June 2005

The Knowledge Fusion Resource

by

Phaedra C. Romano

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

Phaedra C. Romano — May 31, 2005
Phaedra C. Romano — Date

Hazem Said — 6/1/05
Dr. Hazem Said, Faculty Advisor — Date

Patrick C. Kumpf — 6-2-05
Patrick Kumpf, Department Head — Date

Table of Contents

List of Figures

List of Tables

Abstract

The Knowledge Fusion Resource is a complex system utilizing web technologies to create an easy to access and use application for a software company's support system. This application is meant to be used as the primary application for call and support departments within a software company. Employees within those departments are able to create call records, create help tickets for problems requiring more in depth solutions, and create knowledgebase articles for use in reporting and data analysis and for solving future incoming calls. The system can be administered through the administrator console. The system was developed using C++ technology and tested thoroughly with the Apache Web Service. This is the recommended web service for running the system, but it can be used on a windows system running Internet Information Services as well. The primary goal of the system is to allow employees within call and support departments to easily record data and solve customer problems in a comprehensive and quick fashion.

The Knowledge Fusion Resource

1. Statement of the Problem

Many companies have knowledgebase systems and help desk systems to keep up-to-date records of the problems found with their products and services. A knowledgebase is a searchable database that contains any piece of information on a given topic. In the event that a knowledgebase is combined with a help desk system, the knowledgebase will typically contain information concerning various problems or issues customers raise.

With a good help desk system, tickets are created and given unique identifiers. Then they are assigned to a technician for resolution. These systems allow multiple attachments to be included in the ticket, such as screenshots, emails, and steps tried in solving the problem, as well as a solution. The information is organized so that it can be searched easily if a duplicate problem occurs. Technicians can then open the already existing ticket, look at the resolution, and quickly solve the current customer's problems. A major selling point for help desk software is that it provides the documentation needed in justifying things like salaries and requests for additional resources and/or upgrades to current resources. It is a tool for managers' budgeting and evaluation tasks (4).

Flightline Software Incorporated is a software company that makes programs for the airline industry. Currently it has support system that is incomplete. Like most help desk systems, customers call in to the support center to report problems they have encountered with the company's software solutions. The customer will talk directly to a support technician who will record his or her contact information and what their problem is. This is how many help desk systems work today according to Help Desk Management Software World. However, Flightline's system currently has security holes and solutions to problems tend not to be recorded in the system at all. This leaves technicians starting from the beginning whenever a similar issue is reported (13). One flaw in the system, according to Ron Struempf, Director of Programming for Flightline, is that Microsoft Access requires all technicians to have full access over all tables allowing them to change data that they should not be able to change. Another flaw is that there is only one field for the problem and solution. Flightline's system was incorrectly set up and leaves little to no room for growth and change. This is not only a problem for distinguishing

between what the problem is and the solution, but also for deciphering the solution, if it is even entered at all. Frequently, the solution gets lost or is never entered, which defeats part of the purpose of a knowledgebase (13). According to Julie Stone, Flightline Customer Service Representative, the current system is extremely slow as well. She states that "If it's your first call of the day, you better hope you already opened this before talking to the customer-it takes forever for Access to open!" (14) With these problems, it is understandable that the software that Flightline is currently using is inadequate for its current needs.

Since Flightline is a software company, it did not consider the option of buying a solution from another company. Its programmers are competent enough to create a solution that is completely customized for its own needs. At the time that I requested a project, Mr. Struempf was trying to determine a way for enlist one of his employees to build the solution the support department required. The problem he encountered is that this is an internal application that would cost money, but never actually produce money. This makes it less of a priority than other projects currently being developed even though this project would allow the support department to do its jobs more effectively and essentially allow it to increase the volume of tickets being resolved in a shorter amount of time (13).

2. Description of the Solution

My solution to Flightline's problem is a completely customized software application called The Knowledge Fusion Resource. It has been developed in such a way that currently employed programmers for Flightline will be able to support the product themselves. This will take little training on their part because I intend to adopt their own coding standards and business rules into the application.

Some of the key features of this solution will be:

- Ability to include multiple steps taken to resolve a ticket in separate fields
- E-mail notifications of incoming requests, outstanding tickets, and ticket resolutions for the technician and the customer
- Search options to search tickets by ticket type, steps taken, customer, technician, and resolution

- Quick and easy notification to Programming department of programming issues
- Inability to close a ticket without a solution
- Easy to pass tickets from one technician on day shift to a technician on the night shift
- Quick Links to open tickets
- Ability to generate reports based on ticket type, solutions, and number of hits
- Remote access capabilities

The Knowledge Fusion Resource has the capability of being accessed from work or from home with a unique login for each technician as long as the company has their network set up to allow outside access. With this capability, a technician could be sent out to the customer's home to manually attempt to resolve the issue and then close it immediately after completion. This also allows for Mr. Struempf to be able to check on how issues are being resolved after work, especially when the support department calls him with questions. He will be able to see if his solutions were correct or not.

Some of the key concerns:

- *Increased Security*
  I am limiting the access to tables by preventing just anyone to access information directly. Each person will have a given set of permissions that will allow them to access only the data they are authorized to do their job.

- *Easy navigational tools*
  There is a side bar that lists frequently accessed tools such as searching, creating new tickets, and closing tickets. In each section, there are buttons located in visually appealing locations to perform operations necessary for key business logic.

- *Dynamic*
  New tables for the knowledgebase are able to be added easily or modified. Forms have the ability to be expanded for new fields. New modules are easily incorporated.

- *Web Access*
  There is a Web interface that connects directly to the database allowing for easy access from any location given the correct login is supplied.

- *Maintainability*
  The program has been written in C++ with a CGI front end. This is the current technology being used by Flightline and is what they are able to support.

2.1 User Profile

There are three user profiles based on the specifications that Flightline outlined.

2.1.1 Help Desk Technician

The help desk technician is the person answering the phones in the call center. He or she creates calls and new tickets, updates existing tickets, resolves tickets, closes tickets, forwards tickets to the programming department, and searches the knowledgebase and tickets for existing tickets/problems/solutions. Resolving a ticket is different from closing a ticket. Resolving the ticket involves actually entering the solution to the problem and recording it into the database. Closing the ticket is when the customer has been notified of the resolution and the ticket is no longer requiring any more steps to be taken. A technician primarily will close tickets they have resolved, but in the event that the ticket must be sent to the programming department, and a resolution is entered by the programming department, the technician will be reassigned to the ticket for closure.

2.1.2 Administrator

The administrator has much of the same functionality as the technician. He or she is be able to search the knowledgebase, close and resolve tickets, forward tickets to the programming department, reassign tickets, update tickets, delete tickets, and check the status of tickets. Along with these basic functions, the administrator is able to create users, modify users, inactivate users, add products, add companies, and generate reports. The types of reports the administrator will be able to generate reports about the most frequently accessed tickets, employee productivity, open tickets, and tickets closed in a given time period.

2.1.3 Programming Department

The programming department has the ability to search tickets, update tickets, resolve tickets, and check ticket status. After the programmer resolves the ticket, it will be reassigned to the technician who originally had it for closure.

2.2 Design Protocols

2.2.1 Organizational Scheme

Flightline Knowledgebase is divided into three sections based on three user-roles as explained in the previous section. The user-interface is a web-based interface. Each user will go to a website and log into the system. When the person logs in, they will see a set on links on the side of the screen that are based on the functions to which they have access. The links displayed will be generated dynamically. The functions that each type of login will have can be seen in Figure 1. This use-case diagram shows each user, as defined in the previous section, logging into the system and then being able to access set functions based on the user-role. Each function is sub-divided into functions that can be accessed after the initial function has been performed. For example, when the helpdesk technician logs into the system, he or she will immediately be able to access the customer search function. The customer can be searched by name, customer id, employee number, or by airline id, all of which are listed after the customer search use-case. After performing one of the searches for the customer, the user will then be able to do one of three things: search the knowledgebase, create a call record, or create a new ticket.

2.2.2 Database Design

The database for this project is a complex system of normalized tables. These tables have one to many and many to many relationships as necessary. The main focus is the Ticket table which contains foreign keys to the Customer, Product, Problem, Solution, Computer, and Employee tables. This table is basically the link between all the tables and contains all of the information needed for a ticket. Creating a ticket will result in records being added to the Problem and Solution and possibly the Computer table. Most of the tables have indexed, auto-incrementing primary keys; the exceptions are tables necessary for the many to many relationships. The knowledgebase part of the database consists of the KBArticle,

KBAttachments, and KBSolutionsSteps tables which will be filled automatically upon closure of a ticket.

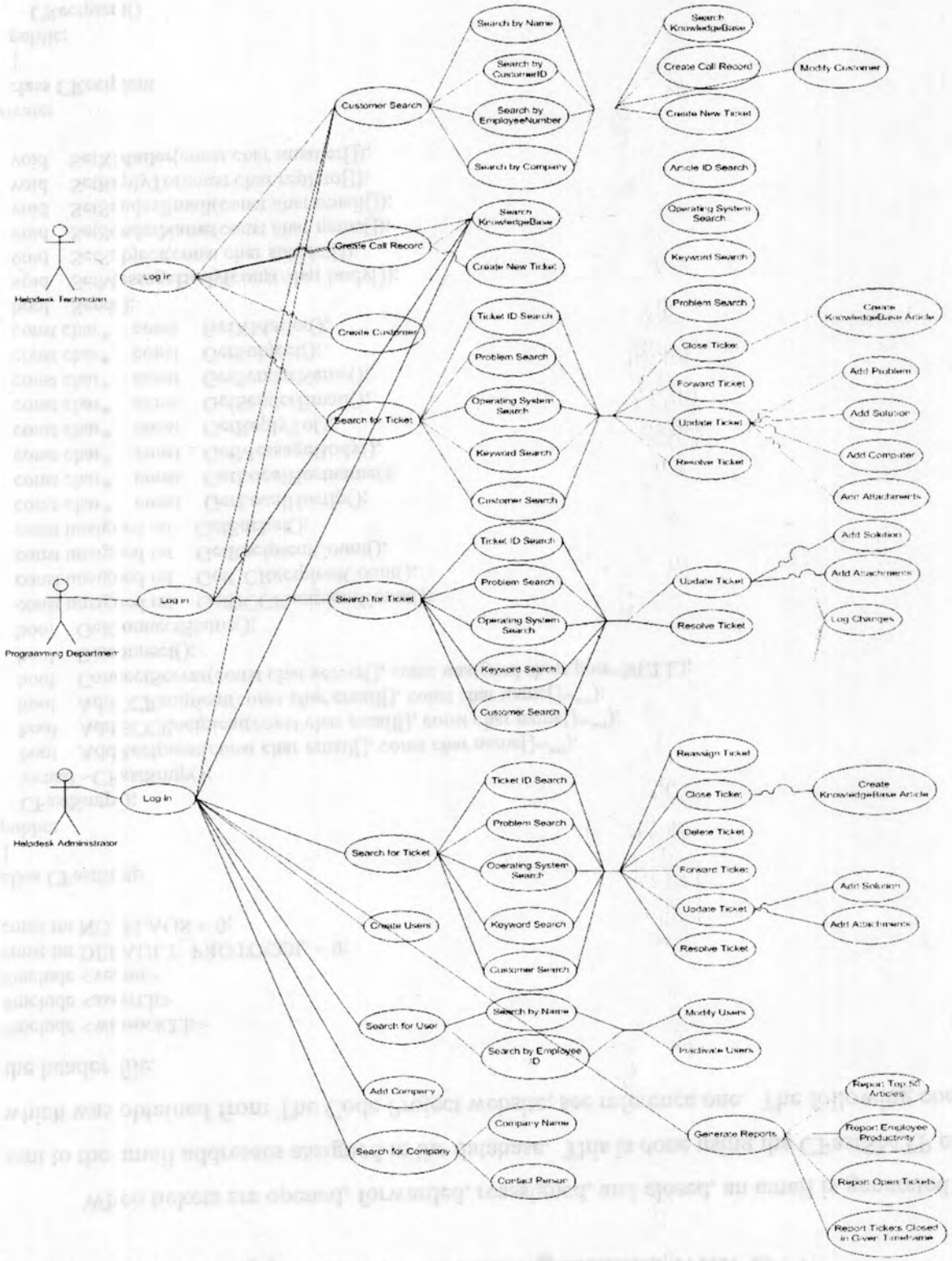


Figure 1: Use-Case Diagram

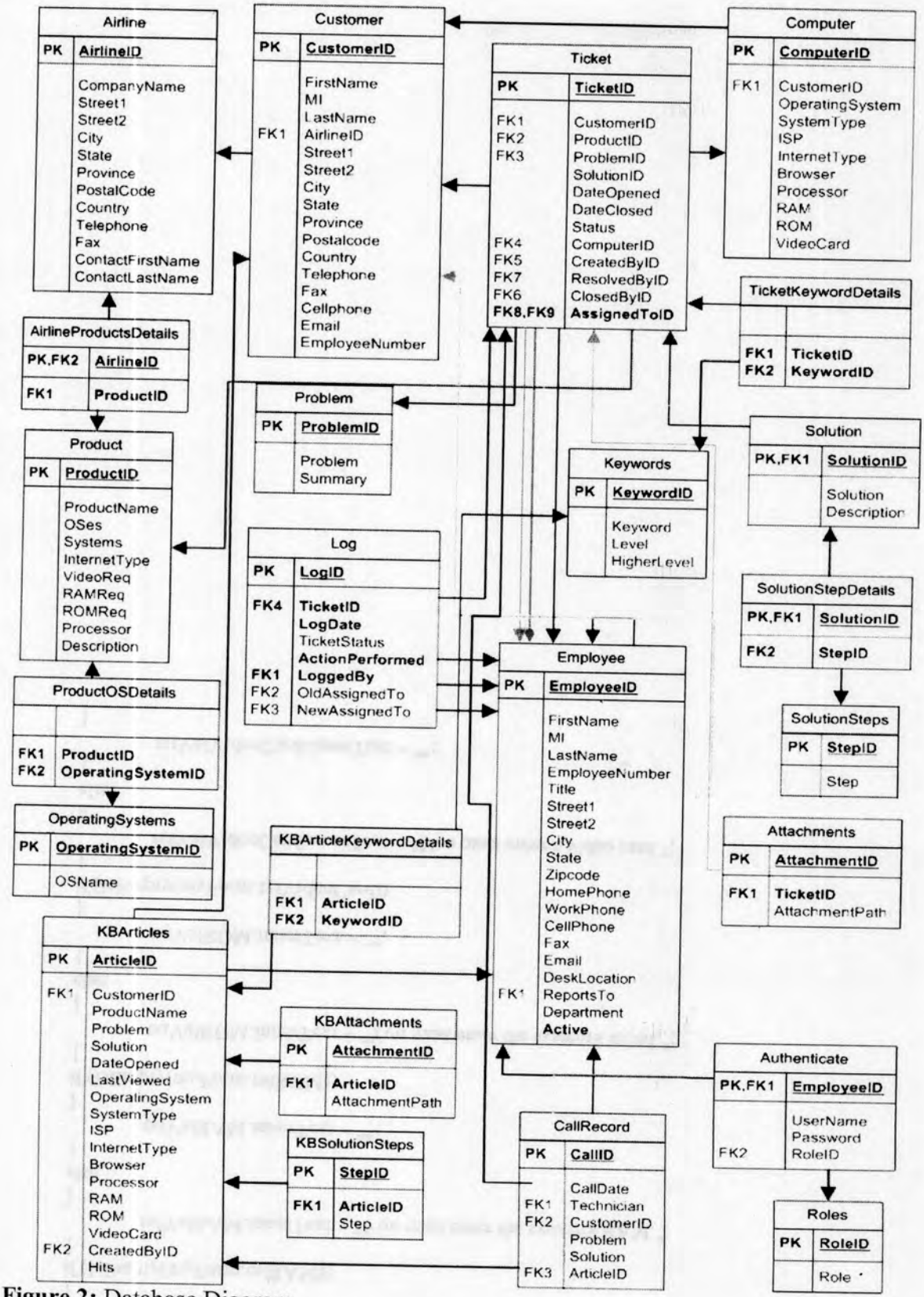


Figure 2: Database Diagram

There is a list of ten links to the top ten most frequently accessed knowledgebase articles. These links are pulled from the articles table in the database based on the hits counter located in the record. Figure 3 shows the initial logon screen, and Figure 4 shows the welcome screen for a technician that shows how the screen will be laid out.

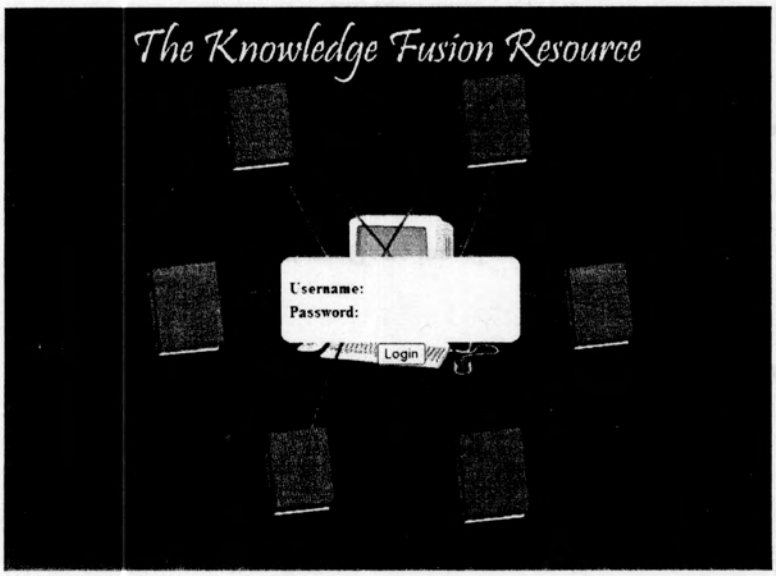


Figure 3: The Login Screen

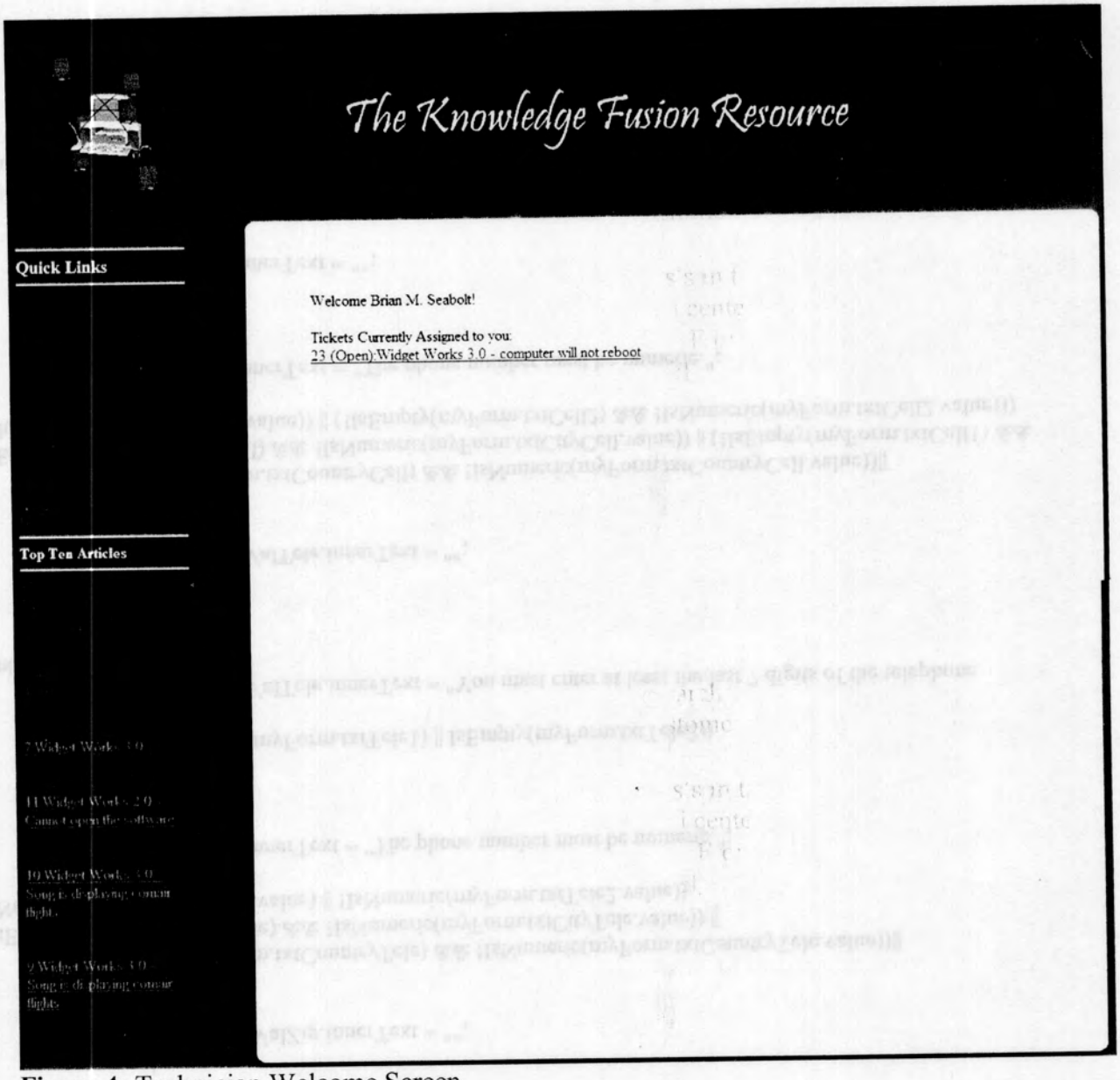


Figure 4: Technician Welcome Screen

The technician's links are as follows:

- **Customer Search:** This allows the technician to search for a customer based on name, customer id, airline id, or employee number the customer has with the company.

- **Create Call:** After looking to see if a customer exists in the database, a call record is created to keep a log of all calls coming into the call center. This section leads to solving problems via the knowledgebase or creating tickets. From here, a customer can be created, tickets are created, and knowledgebase articles are associated to calls.

- **Search Knowledgebase:** Technicians can look up an article in the knowledgebase by operating system, problem description, or keywords specified in the keyword table of the database.

- **Search for Ticket:** In the event that a user wants to find a ticket being worked on, the search will allow them to look up a ticket based on ticket id, operating system, problem description, or keywords.

The programmer's links are the same as the technician's with the exception of creating a call or creating customers. Programmers will not be answering phones and therefore do not need to have that functionality. The administrator has all the functionality that a programmer has. The administrator also has the ability to reassign tickets and delete tickets, but has administrative links as follows:

- **Create User:** This is where a new employee of FSI would be added into The Knowledge Fusion Resource. The user's personal information as well as knowledgebase specific data is entered so that the new user can log into the system and use it.

- **Search for User:** The administrator can search for a user in the system so the data on that user can be modified and activated/inactivated from the system.

- **Generate Reports:** Reports that can be generated are a list of the top fifty knowledgebase articles, employee productivity, list of open tickets, and a list of the tickets closed in the last month.

2.2.3 Interface Design/Navigation

The overall method of moving through the system is by following links in the side frame. The screens from those links are brought up in the main window. Searches performed result in a list of hyperlinks. All other screens have buttons that will interact with the database as well as bring up other screens. For a complete diagram of how the user will move through the system, see Appendices A, B, and C.

2.2.4 Icons/Graphical Symbols

The only real icon that is carried through the program is the help icon. This little icon will be in the bottom right hand corner of each screen. The user can click on this icon to bring up the help menu.

Figure 5: Help Icon

2.2.5 Color Scheme

The color scheme for this project is red, blue, and purple. The colors chosen are based on the logo for the system, as shown in Figure 6. The logo represents each piece of the name of the project. The books represent knowledge, the computer represents resource, and the purple lines indicating motion around the computer signify the fusion of knowledge and the resource. The choice of red and blue is based on fusion which can be hot or cold. The purple lines help to enforce the fusion of these two colors. The links will be these three colors. All of the regular text will be black for ease of reading and all important text will be red to indicate the importance.

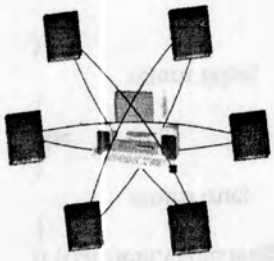


Figure 6: The Knowledge Fusion Resource Logo

2.2.6 Help

There is a help section that when clicked brings up a list of topics available. This menu will be available by clicking on the icon at the bottom of the screens that has been outlined in the *Icons/Graphical Symbols* section of this paper.

3. Deliverables

To provide a well designed and easy to use system, a wide range of deliverables have been laid out. The following deliverables were the result of the design phase of The Knowledge Fusion Resource:

1. A back-end RDBMS on SQL Server 2000.
2. Web interface developed in HTML, CGI and JavaScript.
3. Authentication for technicians, programmers, and administrators.
4. A role-specific navigation bar on every page for easy navigation of the application.
5. Unique forms and reports requested by the client.

6. Ability for authenticated technicians to:
   - Add call records, customers, tickets, attachments, and keywords into the database.
   - View knowledgebase articles, tickets, and customer information.
   - Update tickets and customers.
   - Search on multiple criteria for customers, knowledgebase articles, and tickets in the database.
   - Resolve and close tickets resulting in creating knowledgebase articles.
7. Ability for authenticated programmers to:
   - Add attachments into the database.
   - View knowledgebase articles, tickets, and customer information.
   - Update tickets.
   - Search on multiple criteria for customers, knowledgebase articles, and tickets in the database.
   - Resolve tickets.
8. Ability for authenticated administrators to:
   - Add customers, attachments, and keywords into the database.
   - View knowledgebase articles, tickets, and customer information.
   - Update, forward, reassign, resolve, close, and delete tickets.
   - Search on multiple criteria for customers, users, knowledgebase articles, and tickets in the database.
   - Create and edit users for the system.
   - Generate reports based on specific criteria about tickets and articles.

4. Design and Development

The following sections will outline the overall timeline for the research and development of this project, the hardware and software specifications for development, and the project's budget.

4.1 Timeline

Figure 7 is the timeline for the project detailing the tasks defined to be completed in order to fulfill the requirements of the deliverables.

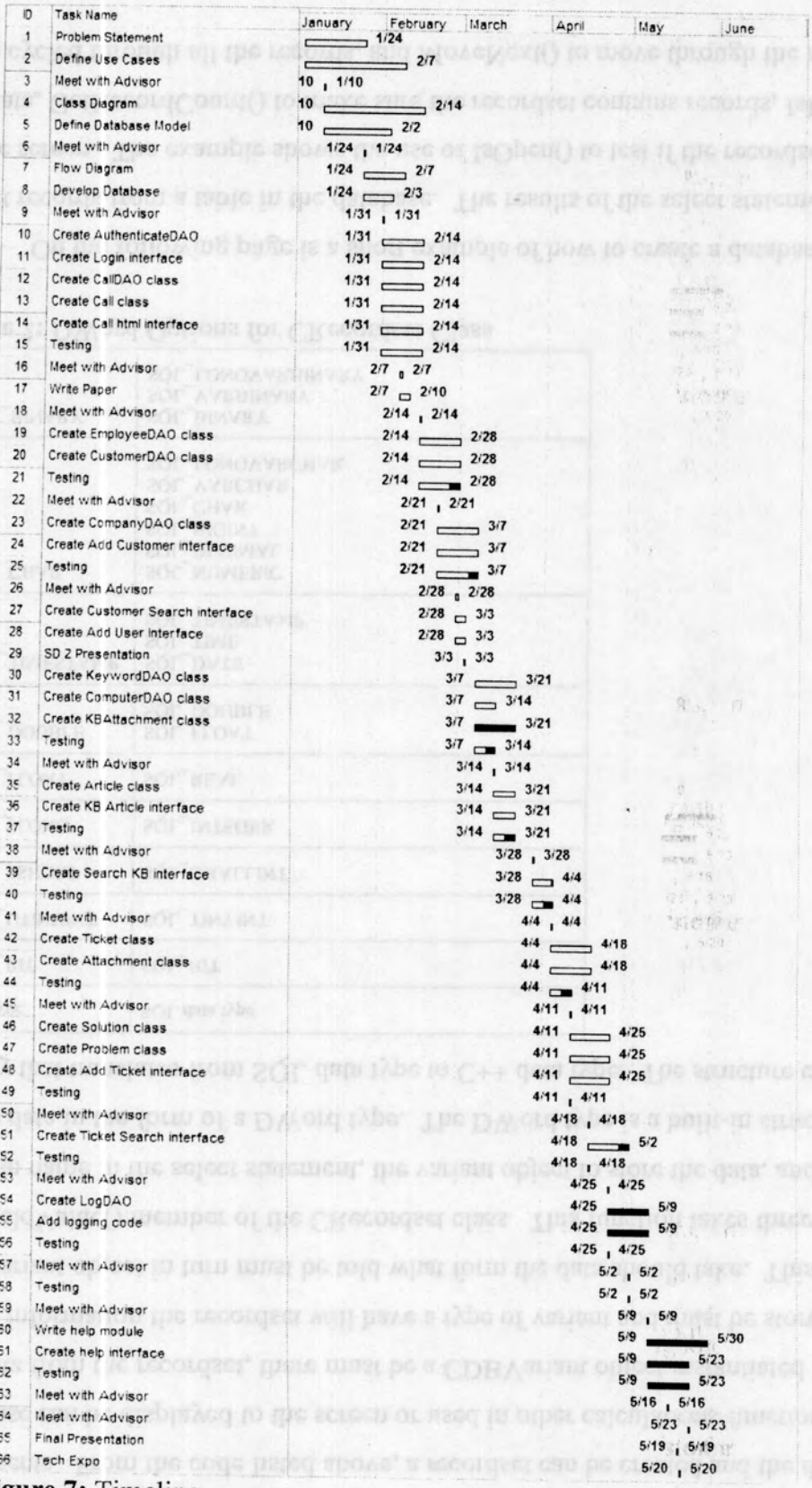


Figure 7: Timeline

During Senior Design one, I was able to accomplish researching The Knowledge Fusion Resource, developing a problem statement, and beginning the design phase of the project, namely the first drafts of the use-case diagram and flow diagram.

During Senior Design two, the design phase was tackled in full force. The use-case diagram was fully developed, as well as the database diagram, class diagram, and flow diagram. The problem statement was refined. The first stages of development began as well. Most of the classes were developed, the customer and user interfaces and the searching capabilities were developed. Call records were able to be generated, customers added, calls closed, customer could be searched, and users added. This was the complete functionality of the prototype.

For the summation of this project, the final development took place. The classes for the knowledgebase were developed. The functionality for creating tickets; creating knowledgebase articles; adding companies; searching for users, companies, tickets, and articles; and all the extended ticket functionality were developed. Users are able to forward, resolve, reassign, save, close, and delete tickets. Attachments can also be added to tickets which when the articles are automatically created are also automatically assigned to those articles. The have been completed.

4.2 Hardware Requirements

The hardware required for this product already existed prior to taking on the project. Since I designed this software with the current hardware set up Flightline has in mind, the hardware I currently own will be sufficient for development and demonstration. The specifics of the computer I used are listed below. These specifications for the computer I used exceed the specifications of the systems Flightline will be using, but I already own the laptop I that I used for development.

Processor: Intel Pentium 4 300 GHz with hyper-threading
RAM: 512MB DDR SDRAM
Graphics Card: ATI Mobility Radeon 9600
Hard Drive: 60 GB

4.3 Software Requirements

I developed this system using Microsoft Visual Studio .NET and Microsoft Sequel Server, which were obtained through the University of Cincinnati. Visual Studio was used for developing in C++ and CGI because of how easy it is to use. It located basic errors and provided me with the debugging tools needed to make the project error free. It compiled the software automatically and generated a report with descriptions of any errors it found so that I could fix them; this includes line numbers and the type of errors found.

Microsoft Visual Studio .NET: This is where I did all the development work in C++ and CGI for the user-interface. I developed this project in the older language even though I used the newest developing technology.

Microsoft Sequel Server: This is where the database that contains various tables for problems, customers, and technicians is housed. It is the power house of the knowledgebase. Currently, Flightline has a license for SQL Server so this adds no extra cost when the software is merged with their current system.

Macromedia Fireworks: This program was used in creating still images for use in screen backgrounds and graphics needed for the web pages.

NotePad: NotePad is a great Web development tool. It does not help figure out where errors occur in Web content, but it is easy to use. I used Internet Explorer for Web debugging.

4.4 Budget

Table 1 is the budget used for this system.

	Item	Explanation	Cost
HARDWARE	Intel Pentium 4 300 GHz with hyper-threading	Own	$ 181.00
	512MB DDR SDRAM	Own	74.00
	ATI Mobility Radeon 9000	Own	125.00
	60 GB Hard drive 5200 RPM	Own	125.00
SOFTWARE	Microsoft Visual Studio .NET	Own	1,079.00
	Microsoft Sequel Server	Own	1,489.00
	Macromedia Fireworks	Own	299.00
	NotePad	Own	Free
	Total Cost:		**$ 3,372.00**

Table 1: Proposed Budget Sources (2, 3, 4, 5)

5. Proof of Design

The following sections will explain how the deliverables have been met.

5.1 SQL Server 2000

The SQL Server database was developed in the middle of Senior Design two. The database design in Figure 2 details all of the tables being used in the system and the relational links between those tables. Connection to the database has been done using the CDatabase class and records are obtained using the CRecordset class. These classes exist in the MFC console library provided with Visual Studio by Microsoft. Connection is very simple, but an ODBC object must exist on the server in order for the connection to work. Figure 8 shows the class diagrams. Explicit details on how to use these two classes can be found in Appendix D.

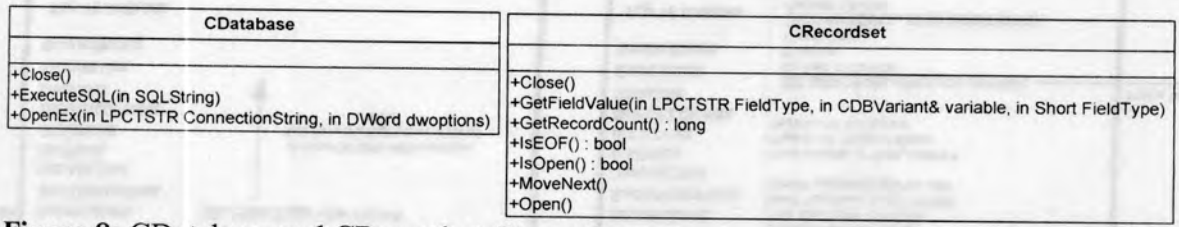


Figure 8: CDatabase and CRecordset Class Diagram

5.2 Web interface

The Knowledge Fusion Resource has been thoroughly tested using Apache Server on a Windows XP Professional machine. When Apache was installed, a cgi-bin folder was set up automatically with the ability to run executable files. An htdocs folder was also created to hold the html pages and the graphics for the site. This folder is where the Login.html page is stored. With Apache running side by side with IIS 5.1, the C++ files for this project will run without any other setup required. Figures 3 and 4 show the login screen and the welcome screen providing that proper authentication have been provided to the system. Figure 9 shows Apache running.

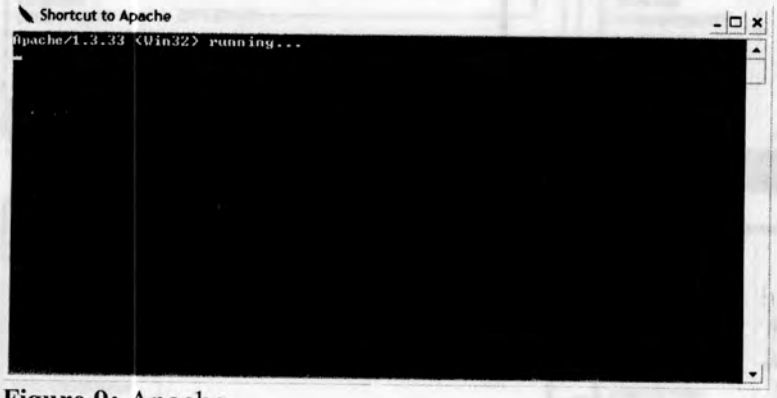


Figure 9: Apache

5.3 Authentication

The security portion of the deliverables has been met through the login screen that is first seen upon navigating to the Knowledge Fusion Resource. The user provides the system with a username and a password. Upon submission, the system goes to the database and verifies that the credentials exist in the database and are for an active employee. If user exists, a welcome screen is displayed based on the role of the employee and the user's role and employee number are stored in a cookie on the server for authentication upon every screen load and to verify that the given employee has the rights to access the page he or she is requesting. Figure 10, 11, and 12 show the technician, programmer, and administrator welcome screens respectively. If the user does not have the right privileges to access the requested page, the user is immediately logged out of the system and the cookie is deleted. The logged off screen is displayed in Figure 13.

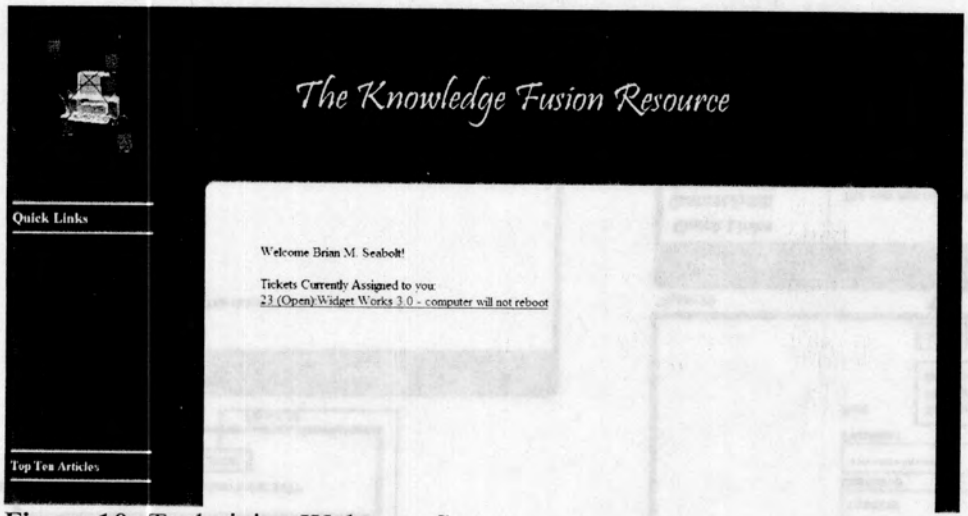


Figure 10: Technician Welcome Screen

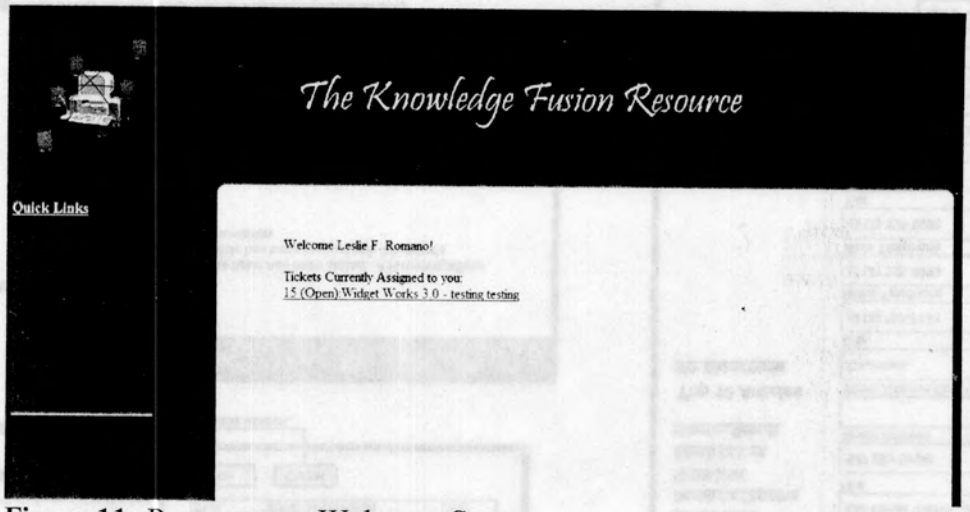


Figure 11: Programmer Welcome Screen

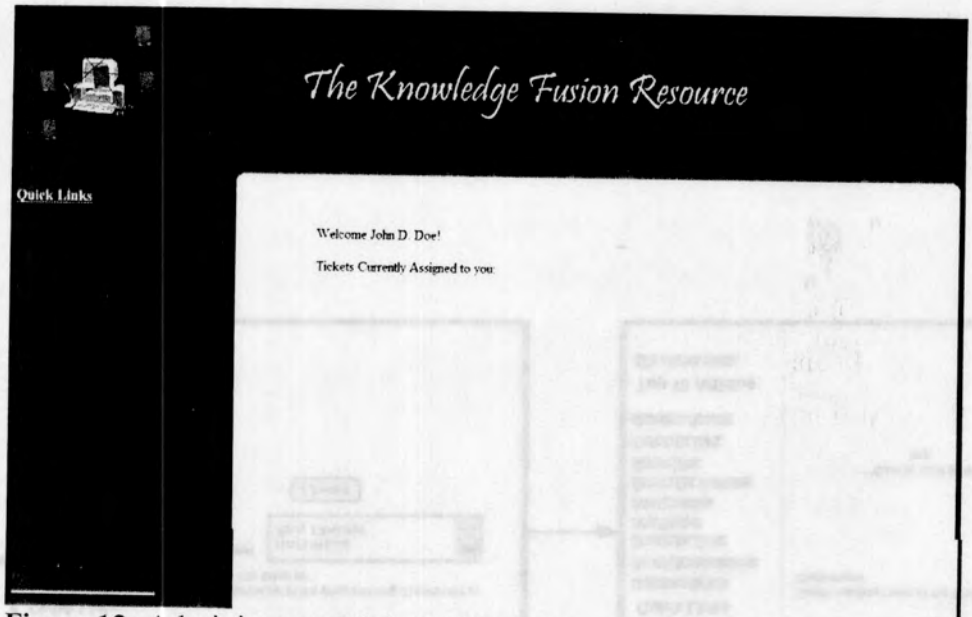


Figure 12: Administrator Welcome Screen

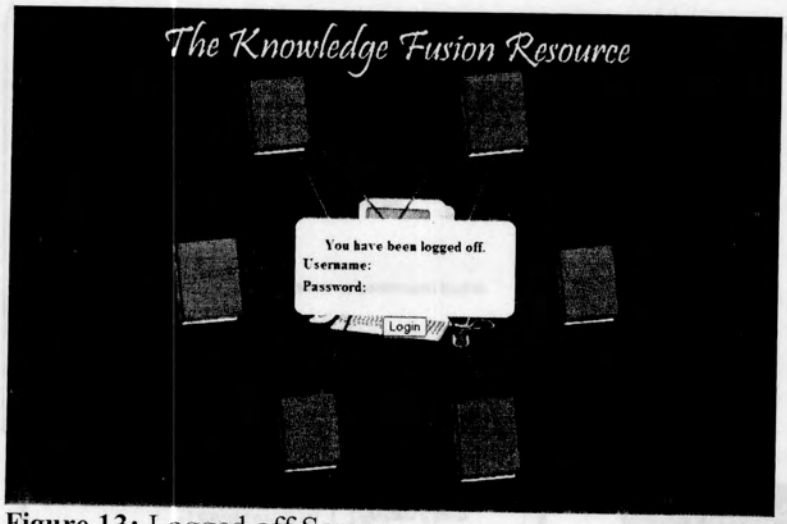


Figure 13: Logged off Screen

5.4 Role-Specific Navigation

Each user, technician, programmer, and administrator has a set of links that are specific to the role. As each page loads, it checks to see if the user has the rights to load that page and then displays the proper information based on what the user is allowed to do on this screen. There is an if statement that displays the links where appropriate. Figure 14 shows the different links for each of the users. Starting on the left and moving right is the technician's links, programmer's links, and finally the administrator links.

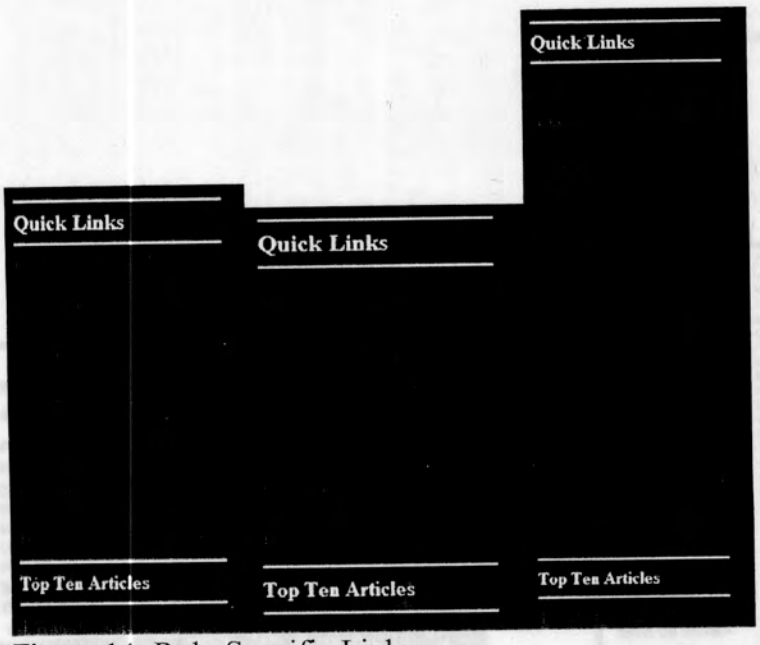


Figure 14: Role-Specific Links

5.5 Authenticated Users Actions

The following sections will details the actions that individual user roles have. I have combined these sections due to users being able to perform the same actions instead of listing each action multiple times for each of the roles. These actions also incorporate unique forms and reports, one of my deliverables.

5.5.1 Adding Records

Records can be added through the program by technicians and administrators.

5.5.1.1 Technician Adding Records

Technicians have the ability to add calls, tickets, customers, and knowledgebase articles. When the technician receives a phone call from a customer, a call record is created by clicking on the Create Call link. Figure 15 shows the call form. If the customer does not exist in the system, the technician can add a new customer by clicking the button next to the drop down list of the customers in the database. Figure 16 shows the Add Customer form.

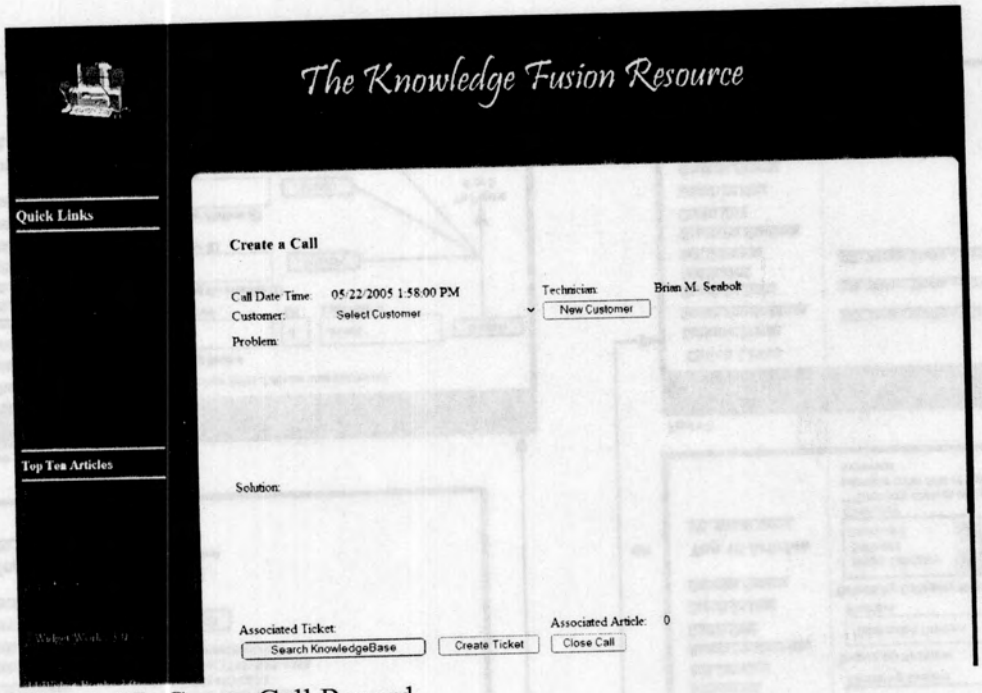


Figure 15: Create Call Record

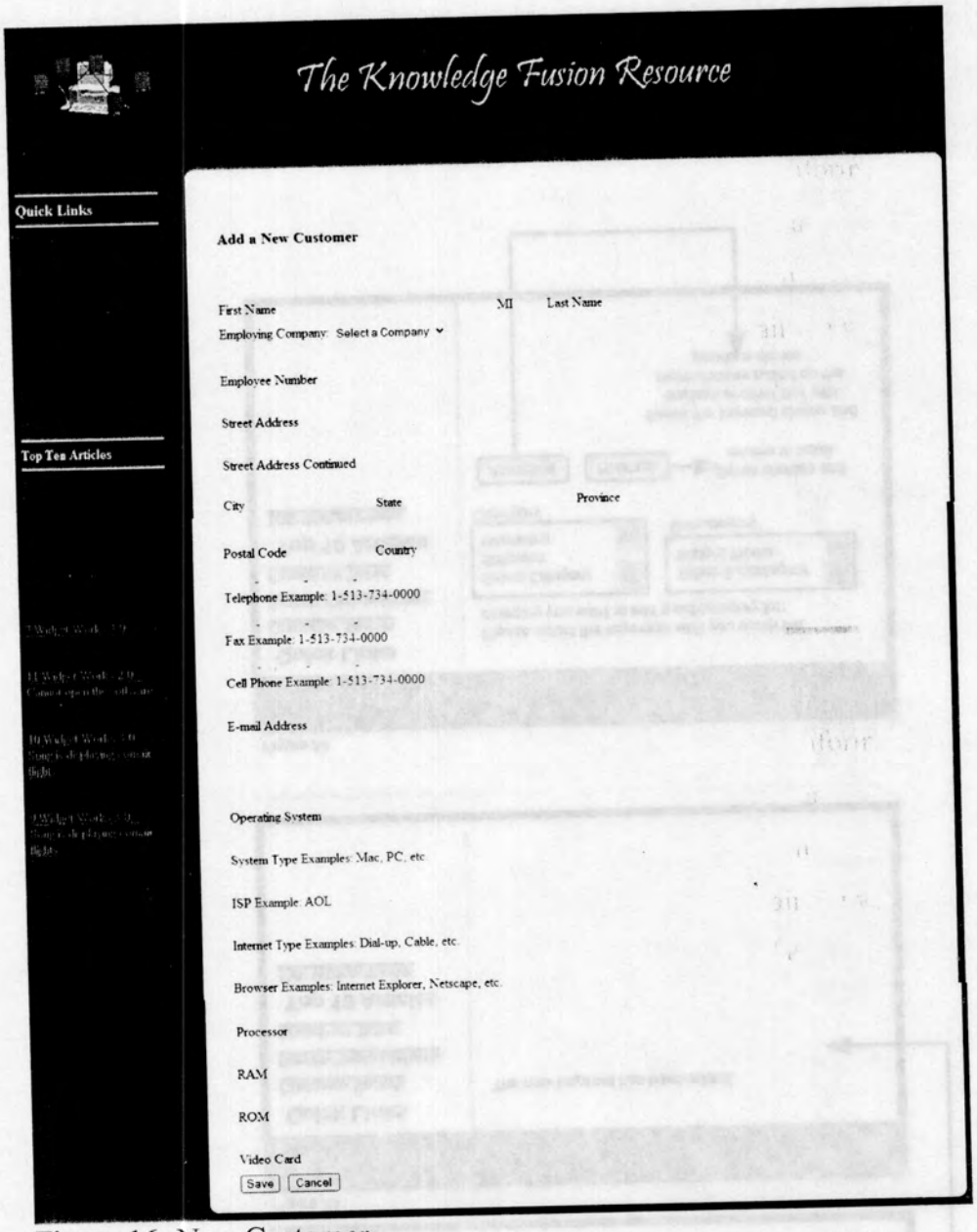


Figure 16: New Customer

The Customer form is for collecting the demographic information about the customer and the primary computer this customer uses. Upon clicking the Save button, the customer information is verified using JavaScript. If any of the data is missing or is not in the format necessary, a message is displayed next to the field that explaining what needs to be corrected before the actual save occurs. Once the data is all correct, the customer record will be added to the database and the technician returns to the call record with the new customer automatically

selected. Appendix E shows the JavaScript validation code used in verifying the customer information and similar code can be found in all of the form modules.

When the call record is ready to be closed or a ticket needs to be created for the call, the call data is saved into the database. If it is closed, the data is displayed on the screen for the technician to see, verifying that the data is actually saved. Since this product has been written in C++, the information is sent through post data and saved to the database. After saving, the id of the call record is retrieved, the call object is instantiated, and the data is sent to the browser for displaying. Figure 17 shows the screen with sample call data being displayed after closing a call.

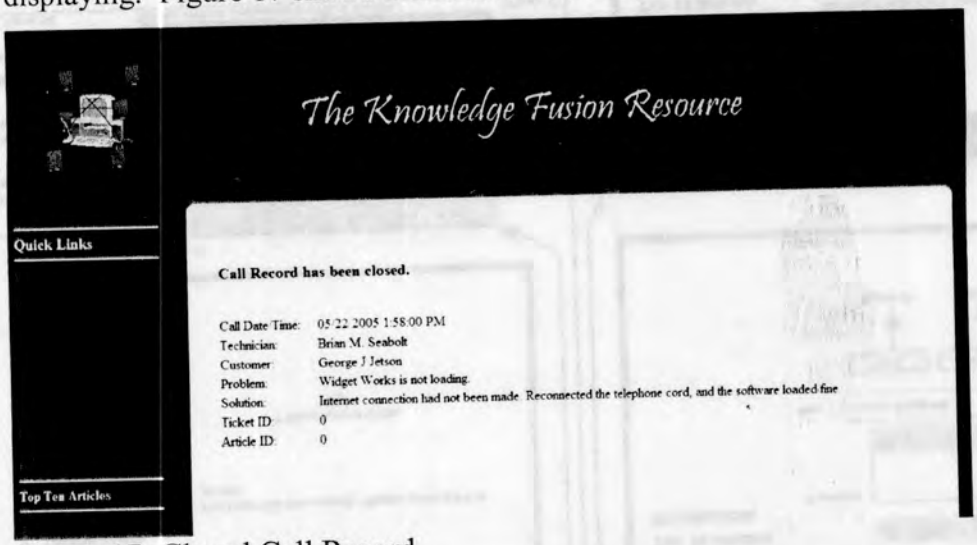


Figure 17: Closed Call Record

If the call needs a ticket to be created, the call is closed prior to creation of the ticket and then the ticket form is opened. Figure 19 shows the ticket form. A new record is inserted into the Ticket table so that a ticket number can be printed to the screen for future reference. An email is also sent to the customer stating that a ticket has been opened on their behalf and lists the details already associated with the ticket like the technician's name, the problem, and ticket id. The emailing is performed using a class called CFastSMTP which was obtained through a code sharing site called The Code Project (1). This is a free site where anyone can register to look at and use the code posted. Figure 19 shows the UML class diagram for this class. Appendix F shows the CFastSMTP class and the code inserted into this project to generate and send an email.

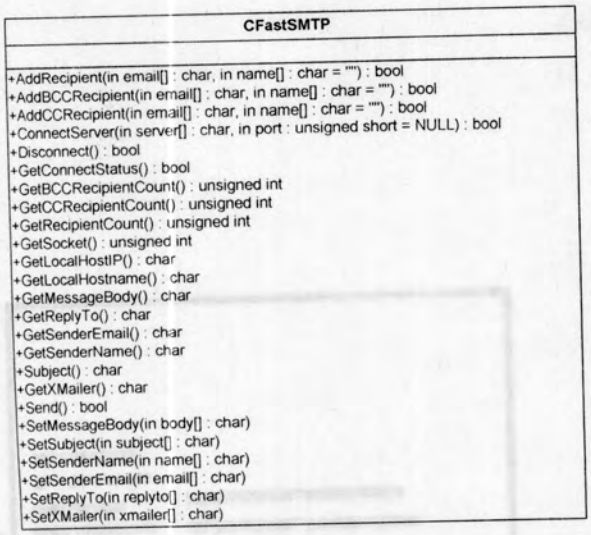


Figure 18: CFastSMTP Class Diagram

After sending the email, the ticket form will load with the customer from the call selected and the problem selected. The problem is saved into the Problem table in the summary field for displaying in the drop down list, shown below.

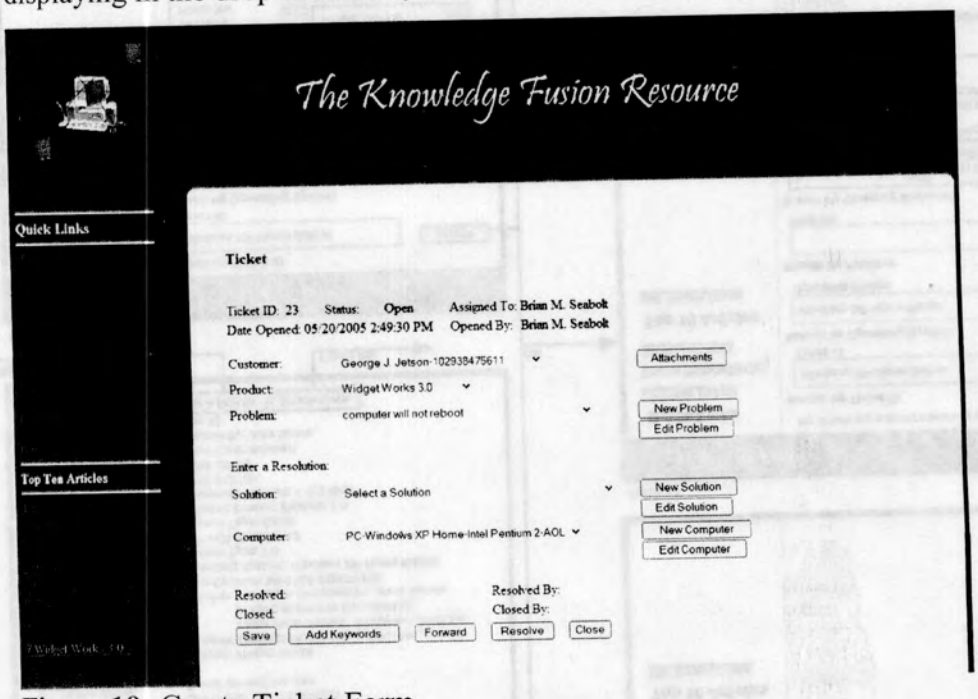


Figure 19: Create Ticket Form

Upon clicking any of the buttons, the ticket is saved before moving to the next action. Clicking the save, resolve, or close buttons will display a summary of the information to the

screen much like when a call is closed. The data is validated upon resolving submission. The ticket cannot be resolved without a resolution typed in and cannot be closed without a solution selected. Figure 26 shows a closed ticket. Upon closing the ticket, an email is sent to the customer stating that the issue has been closed and a knowledgebase article is created upon closing the ticket. This record creation has no form associated with it because it is all done behind the scenes. Figure 27 shows the article that was created. This information has been displayed to the screen through the knowledgebase article search, which will be discussed later.

Within the ticket, the technician may add a problem, solution, computer, attachments, and keywords that are associated with the ticket. The problem is added upon closing the call, but a new problem can be added in the event that there is a more important problem found that is causing the problem the customer reported. Adding a problem will simply add the problem to the table and return to the ticket with the new problem pre-selected. The add problem form can be seen in Figure 20. This works the same for adding a solution. The only difference for the solution is that ability to add steps to the solution which adds records to the SolutionSteps and SolutionStepsDetails tables. The add solution form can be seen in Figure 21. Adding a computer will add the computer, associate it with the customer in the ticket, and return to the ticket with the computer pre-selected. The add computer form can be seen in Figure 22.

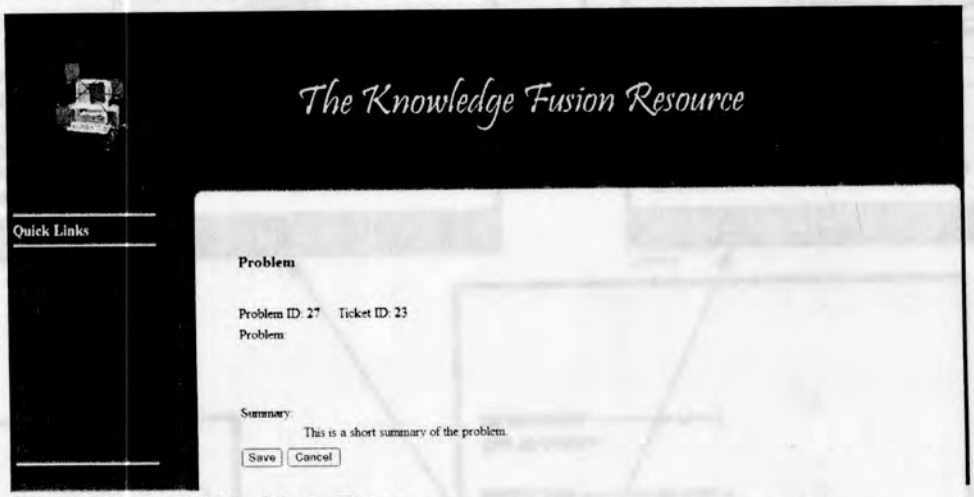


Figure 20: Add Problem Form

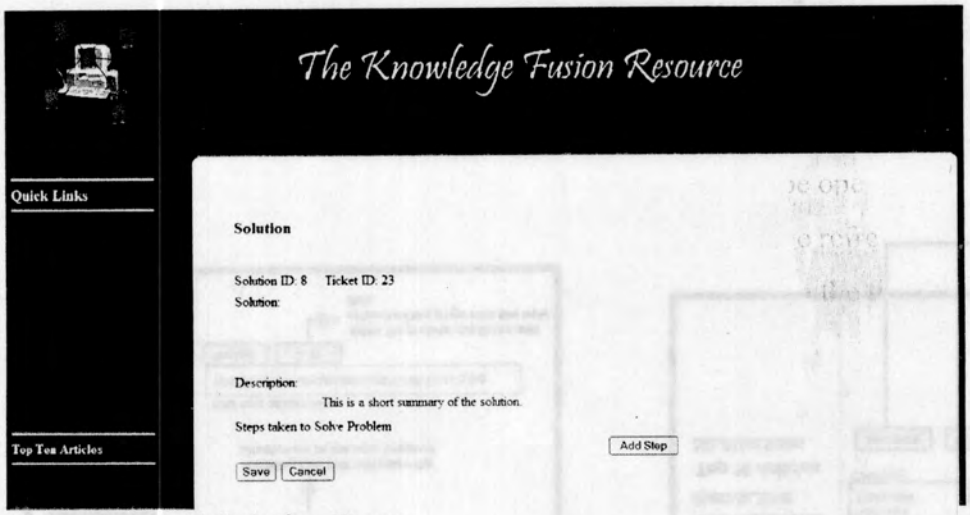


Figure 21: Add Solution Form

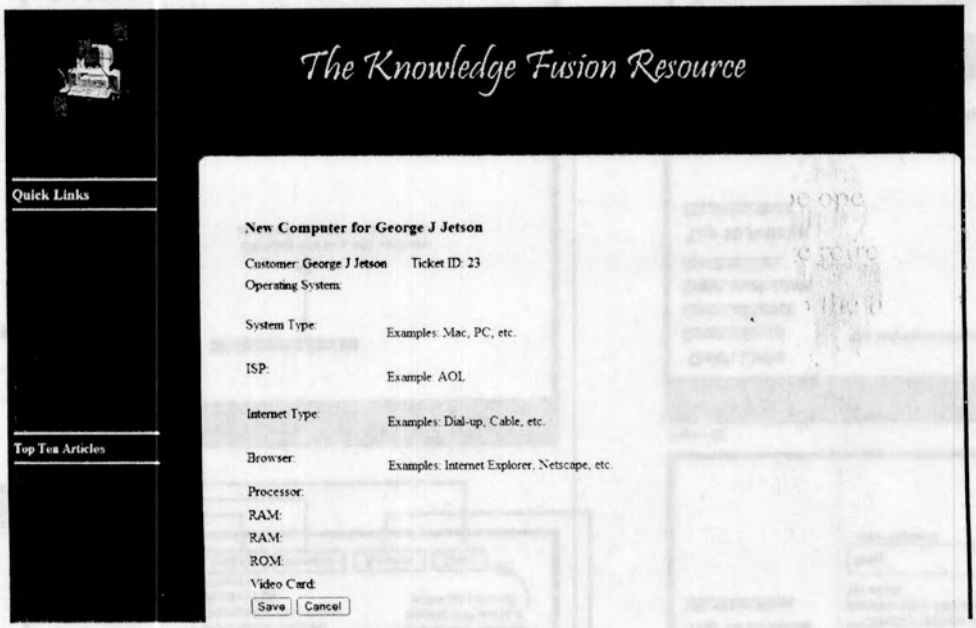


Figure 22: Add Computer Form

Adding attachments is slightly different from the other forms mentioned. Attachments already associated with the ticket are listed as links that can be opened and viewed. The user can also browse to find a file and upload it to the server. The page refreshes and lists the new attachment among the list of attachments. In order to return to the ticket, there is a finished button to indicate that the user is finished uploading files. This form can be seen in Figure 23.

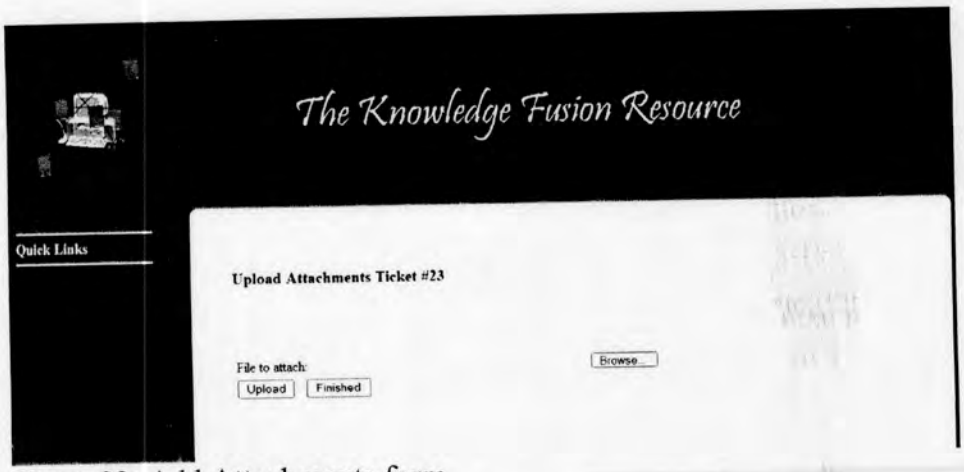


Figure 23: Add Attachments form

The functionality for the attachments is performed using the CInternetSession and CFtpConnection classes built into the MFC Library. These classes create an internet connection, get the current root directory, create a new folder for a new ticket, and will put the file being uploaded into that directory. These two classes are not documented very thoroughly and when used with a UNIX server, create different results that a Windows machine would. In order to find out if a given ticket already had a folder created on the server, I used the SetCurrentDirectory(Folder) function which would return true if the current directory was able to be changed to the parameter within the parenthesis. Otherwise, it would return false, and I called the CreateDirectory(Folder) function to create a new folder for the ticket. However, if the SetCurrentDirectory(Folder) function was successful, calling the PutFile(local file, server file, binary) would return a false and the file would not be uploaded. The reason for this is that on a UNIX server, the root directory is a forward slash, "/". However the PutFile() function interpreted that to mean the current directory instead of being a full path for the file's location. For example, the server I used to upload files to had the folder /Attachment/Ticket 15; I wanted to upload a file to this folder and used the full path in the PutFile() function to do this. The full path was /Attachment/Ticket15/test.txt. The PutFile() function interpreted the path I gave it as a partial path and tried to upload the file to /Attachment/Ticket15/Attachment/Ticket15/test.txt. What this meant was that after the SetCurrentDirectory() function returned true, It must be called again to change the current directory back to the root folder. On a windows server, this action would not have to take place, but the code for this project should work for either a windows machine or UNIX machine. Figure 24 shows the class uml diagrams for the functions used in

the CInternetSession and CFtpConnection classes. Appendix G shows how to use these two classes in more detail.

CInternetSession
+GetFtpConnection(in LPCTSTR ServerName, in LPCTSTR UserName, in LPCTSTR Password) : bool

CFtpConnection
+CreateDirectory(in LPCTSTR Directory) : bool +GetCurrentDirectory(in LPCTSTR DirectoryHolder) +SetCurrentDirectory(in LPCTSTR Directory) : bool +PutFile(in LPCTSTR LocalFile, in LPCTSTR RemoteFile, in FTP_TRANSFER_TYPE_BINARY, in BOOL SimpleTransfer) : bool

Figure 24: CInternetSession and CFtpConnection Class Diagrams

Finally, adding keywords is a very simple form that asks the user to select a keyword. Keywords already associated with the ticket are listed and any new selections are added to the list upon a page refresh after clicking the Add Keyword button. To indicate that the user is finished adding keywords, the user clicks the finished button to return to the ticket. Figure 25 shows the add keywords form.

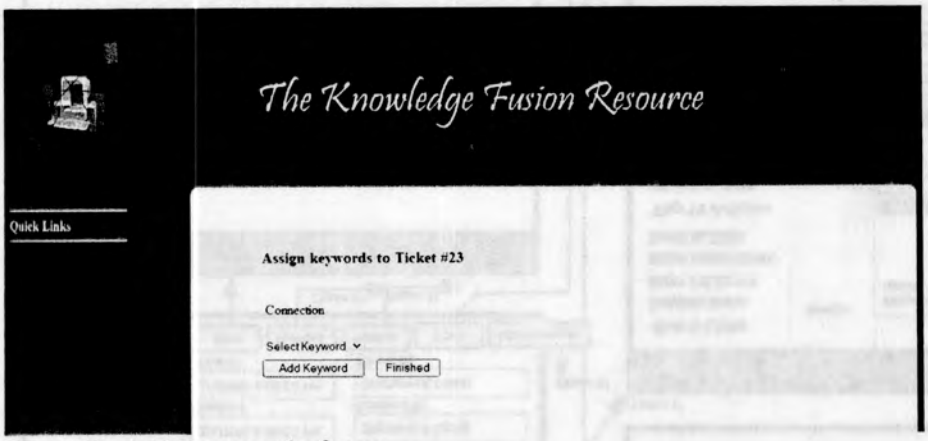


Figure 25: Add Keywords form

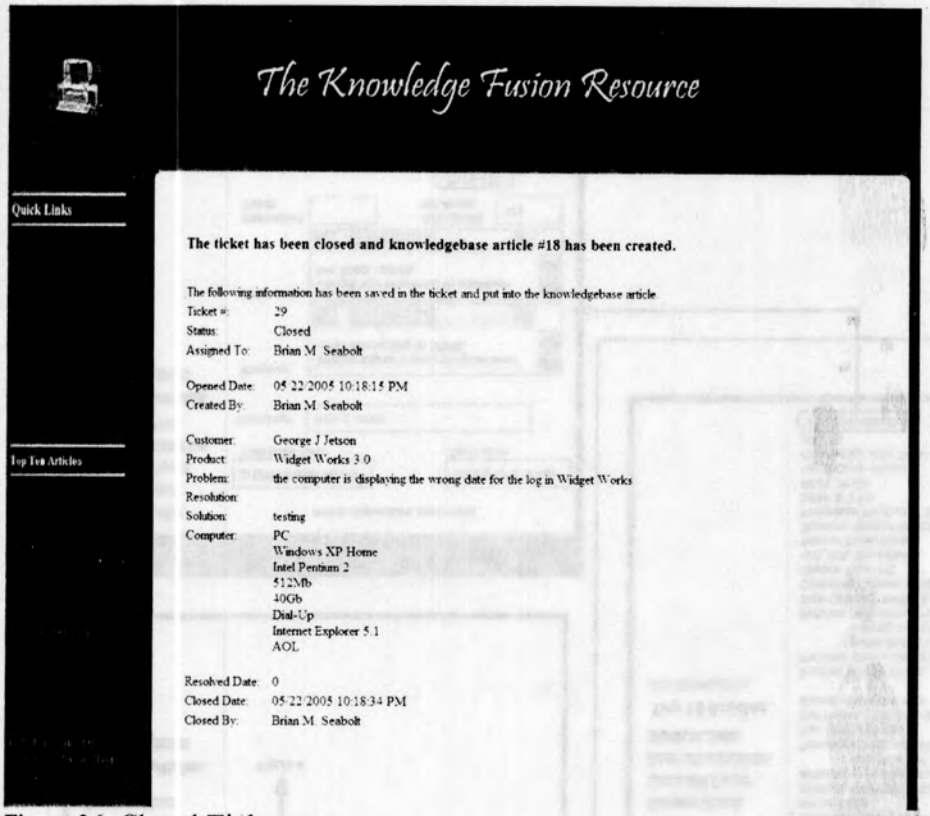


Figure 26: Closed Ticket

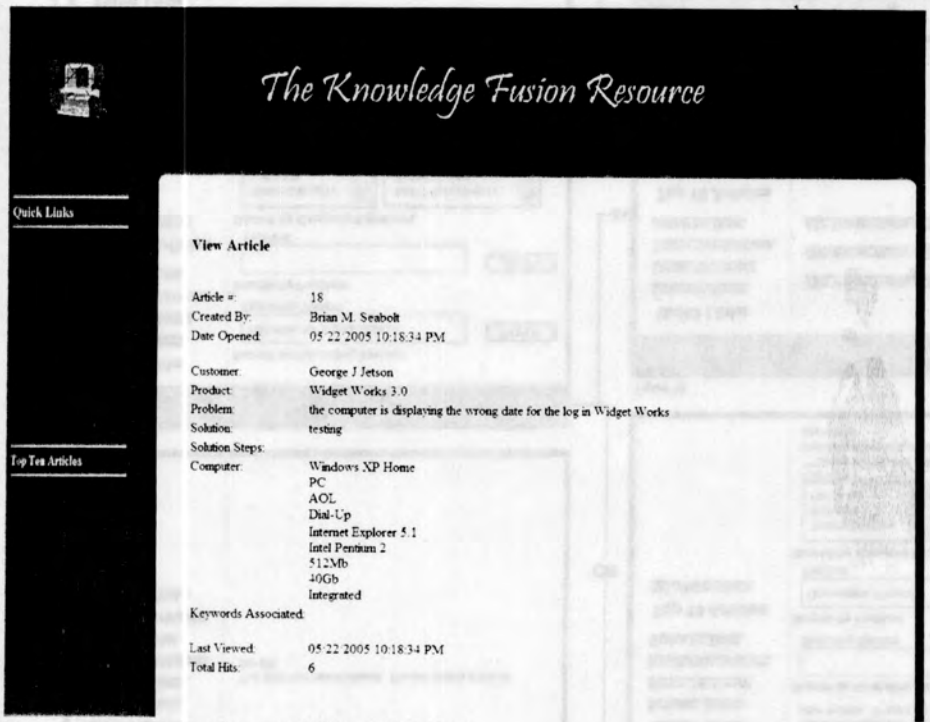


Figure 27: Article Created from Ticket

5.5.1.2 Administrator Adding Records

The administrator has the ability to add products, companies, and users for the system. Adding products is a simple form that requests some basic software information for use in recording tickets and companies. The information supplied is stored in the Product table and the product name is used in a drop down list on the ticket form and as check boxes on the company form to indicate what software a given company has purchased. Figure 28 shows the add product form and Figure 29 shows the data redisplayed after the product has been saved. Data validation does occur on this form to make sure that all of the fields have valid information. The save will not occur until after the correct data has been entered.

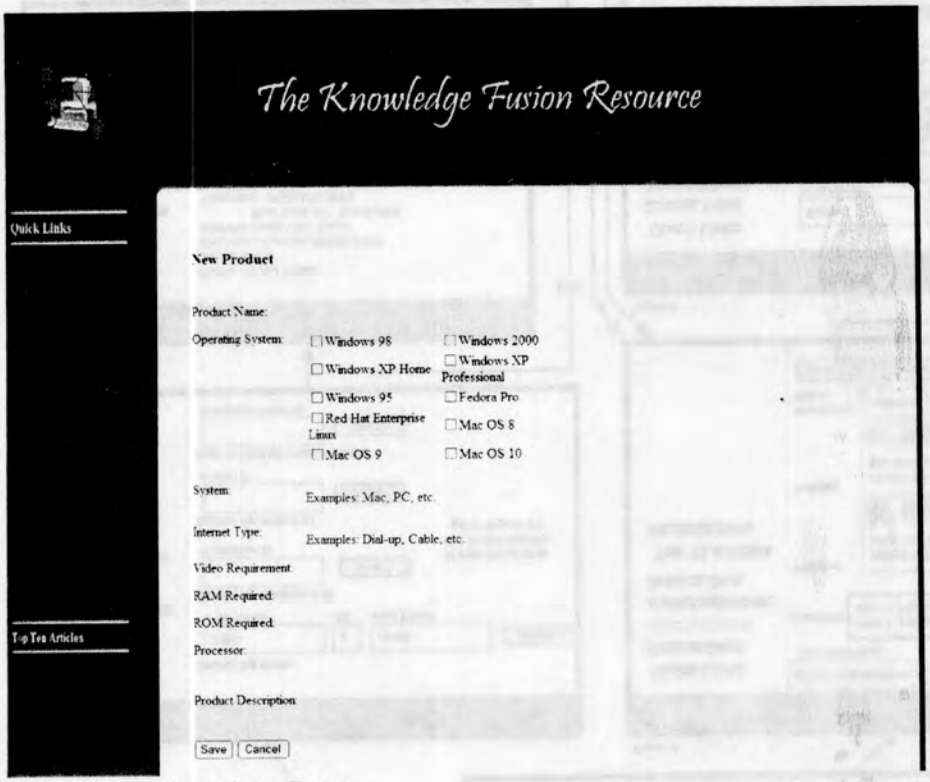


Figure 28: Add Product Form

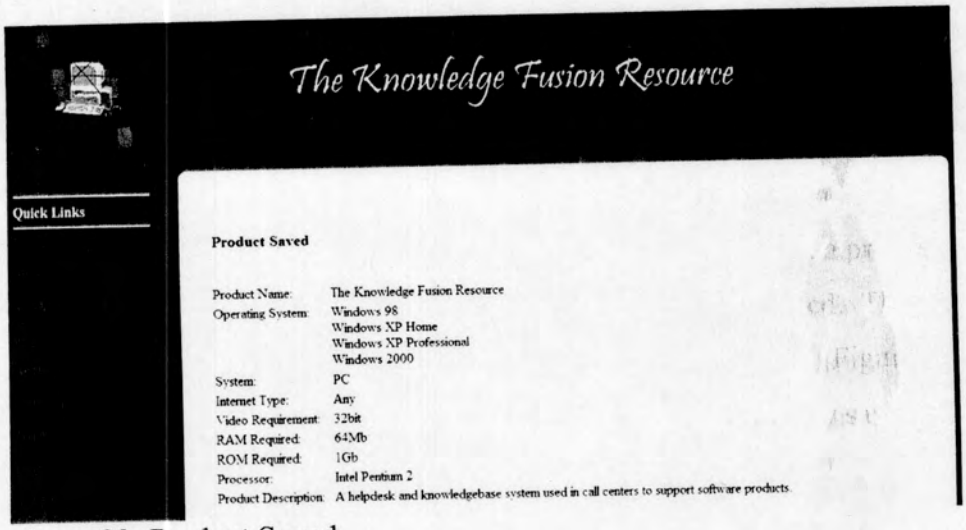


Figure 29: Product Saved

The Administrator also has the capability of adding companies. This would typically occur after a new company has purchased software products that the call center supports. The company information is entered which includes a contact person. This contact person would be who receives calls from the call center in order to verify that a customer is actually employed by the company or in the event that there is a discrepancy between a product a customer needs support for and the products the company has actually purchased. This form is also validated using JavaScript prior to submission for saving to the database. Figure 30 shows the form for adding a company, and Figure 31 shows a company successfully saved.

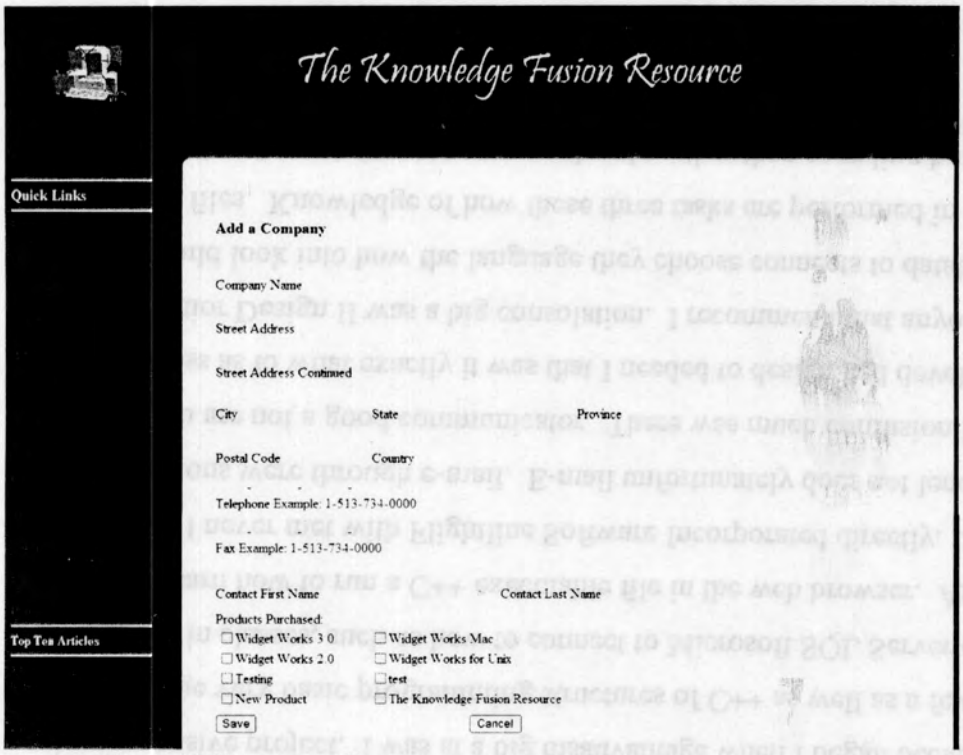


Figure 30: Add Company Form

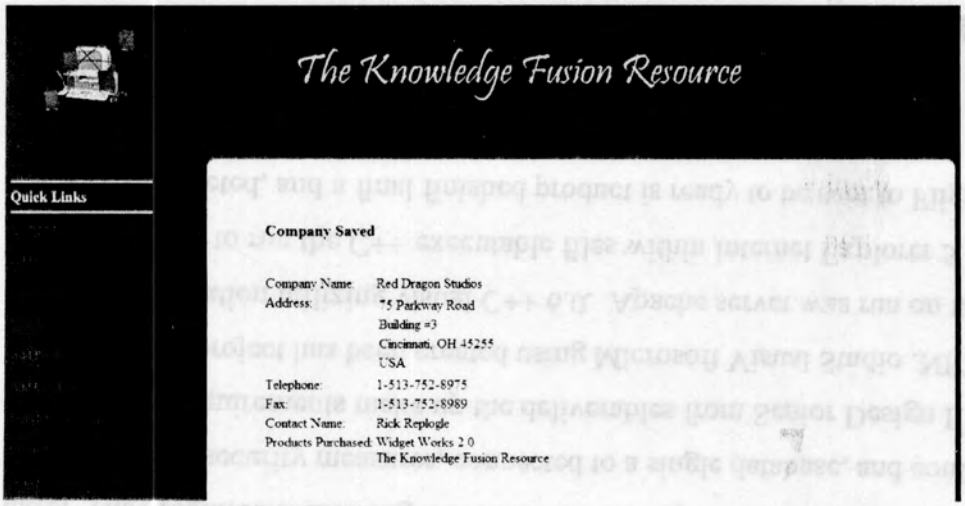


Figure 31: Company Saved

Adding new system users is another addition necessary for the system to operate. These users will be given a role by the administrator in order to log in to the Knowledge Fusion Resource and perform the tasks necessary for their jobs. This requires demographic information and log in information. This information is validated before saving to the database. An email address is required on the form in order for the automatic emailing functionality to work

properly. Figure 32 shows the form for adding a new user, and Figure 33 shows the successful save.

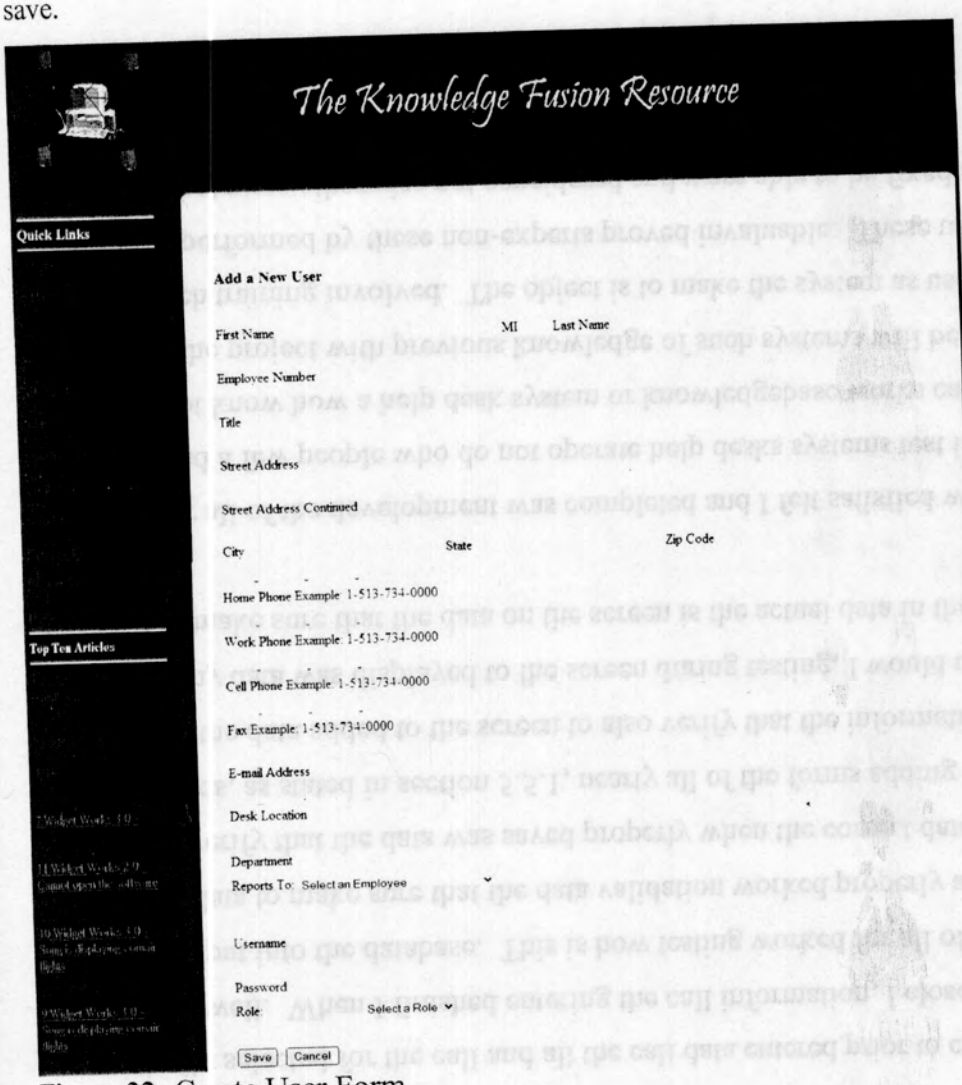


Figure 32: Create User Form

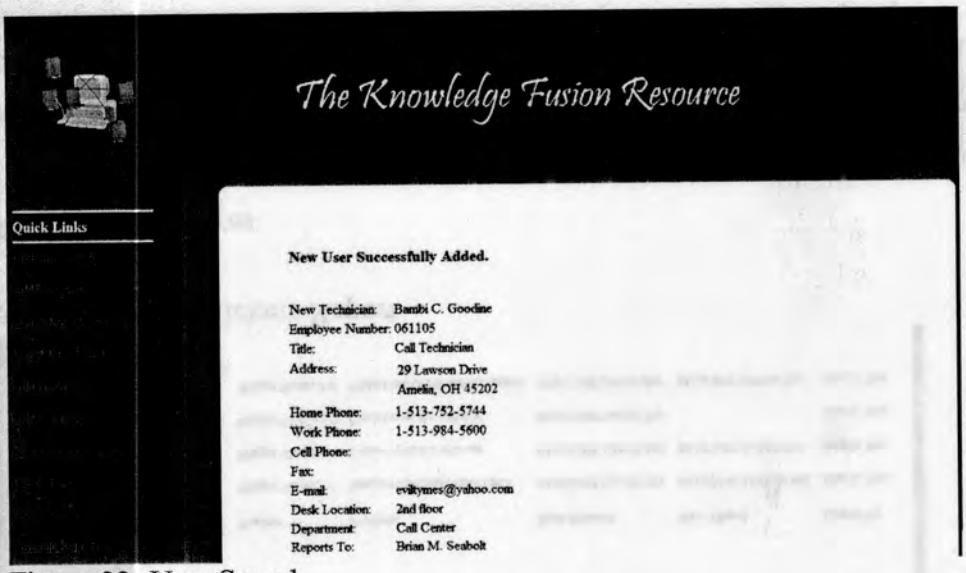


Figure 33: User Saved

In addition to these individual record additions, the administrator can perform two types of record additions within tickets already generated in the system. He or she can add attachments to tickets and add keywords. These functions are necessary in the event that the administrator has some internal attachment related to a ticket that needs to be shared or thinks that a particular keyword would be useful in searching for this ticket. Both of these forms have already been discuss in the previous section and can be seen in Figures 23 and 25.

5.5.1.3 Programmer Adding Records

The programmer has very limited adding functionality. He or she can only add records within the ticket and specifically can only add solutions, attachments, and keywords. Adding these records works exactly the same for the programmer as it does for the technician. Solutions can be added since tickets being sent to the programming department are looking to have a solution implemented programmatically. When the programmer finds the solution, he or she can add it before resolving the ticket. In the event that the new solution brings new keywords or attachments that can be associated with the ticket, he or she has this capability as well. Adding solutions can be seen in Figure 21. Adding attachments can be seen in Figure 23, and adding keywords can be seen in Figure 25.

5.5.2 Searching and Viewing Records

In order to view a record in the database, a search must be performed to find the record in question. All of the users have the ability to search for customer, knowledgebase articles, tickets, and companies. The administrator has the added search functionality of searching for users.

5.5.2.1 Searching for Customers

Searching for customers involves four different search methods: by name, by customer id, by company, and by employee number. Searching by name can be done by typing in the full first name, middle initial, and last name or by typing in a partial name and filling in the blanks with a percent sign which indicates a wildcard, or "any number of characters," in a SQL statement. Figure 34 shows the search form with information typed into the search by name fields. The search will look for customers who have a first name starting with "J". Figure 35 shows the search results for this particular search. After performing the name search, a list of customers is displayed with a button next to the name to allow the user to view the information about this person. Clicking the button will display the information in a non-editable format. Figure 36 shows the customer record being viewed by the user. The screen shot was taken from an administrator searching the system, so a button for modifying the customer is displayed. This button will not appear for the programmer. Searching by company and employee number work the same as searching by name. However, when searching by customer id, which is the id assigned to the customer in the database, only a single record will be returned which means that Figure 35 will be skipped and the search results will immediately display the customer information just like in Figure 36.

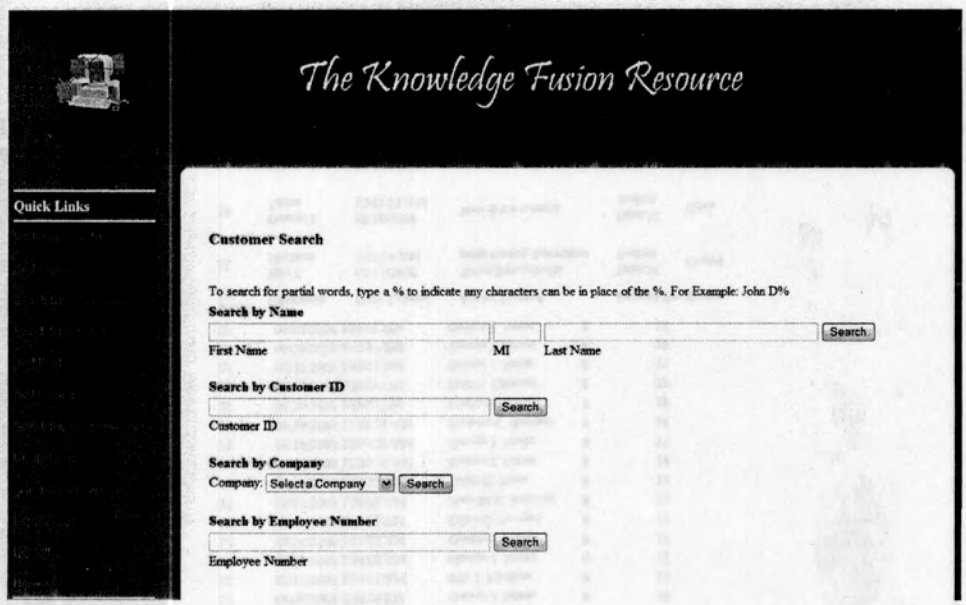


Figure 34: Customer Search Form

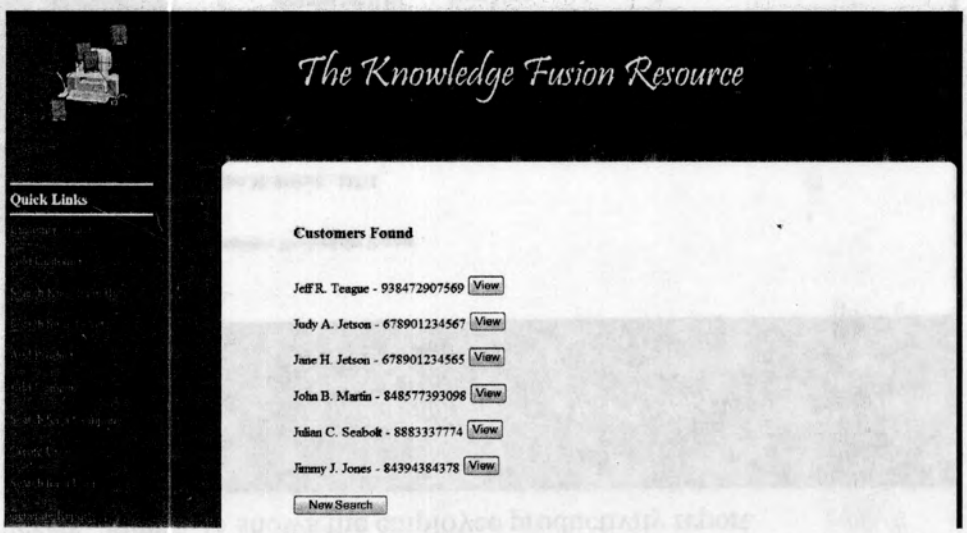


Figure 35: Customer Search Results

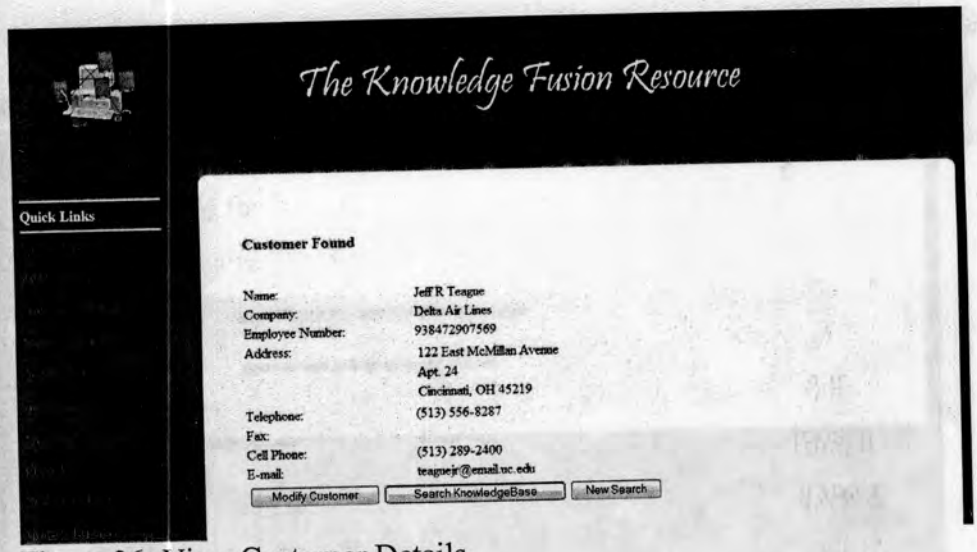


Figure 36: View Customer Details

5.5.2.2 Searching for Articles

Searching the knowledgebase can take place in three different locations. It can be done from within the call form which results in an article being associated with the call record. This method is used so that call technicians can look up an article and read the solution to the customer, thus solving the customer's problem immediately. The call can then be closed, and no ticket is needed. This is done by saving the call record before performing the article search and passing the call record id to the article search form, seen in Figure 37. The id remains hidden in the html code and not printed to the screen. When the user finds an article and views the details, there will be a button displayed at the bottom of the record asking if the user would like to associate this article to the call. If this button is pressed, the user will return to the call form and the article id will be displayed on the form. The other two ways to search the knowledgebase are by clicking on the "Search Knowledgebase" link in the sidebar or by performing a customer search and clicking the "Search Knowledgebase" button on the customer details screen. Details on exactly how the search functionality works are in the following paragraphs.

Searching for knowledgebase articles works very much like searching for customers except that the search criteria is for the article id, operating system, problem, and keywords. Searching by the article id will find a specific article and display the details of that article to the screen. Searching by operating system will look for articles that have a computer with the operating system in question related to the problem. This search can be performed just like

searching by name for the customer where typing in a "%" will search for any characters in the position of the percent sign. If there is more than one article found, the articles will be listed as links and clicking on the link will display the article details. The article search form can be seen in Figure 37 and the list of articles found can be seen in Figure 38. An actual article from the list of articles in Figure 38 can be seen in Figure 39.

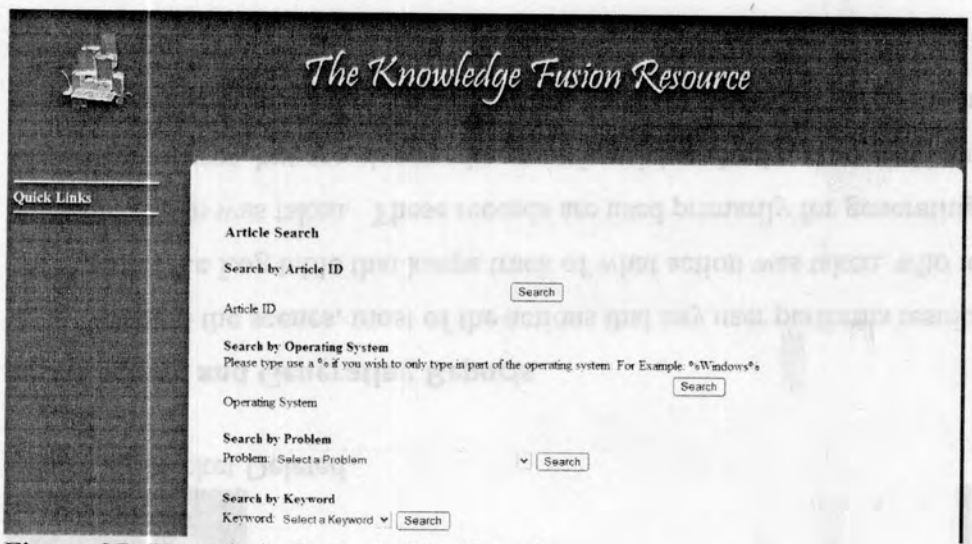


Figure 37: Knowledgebase Article Search

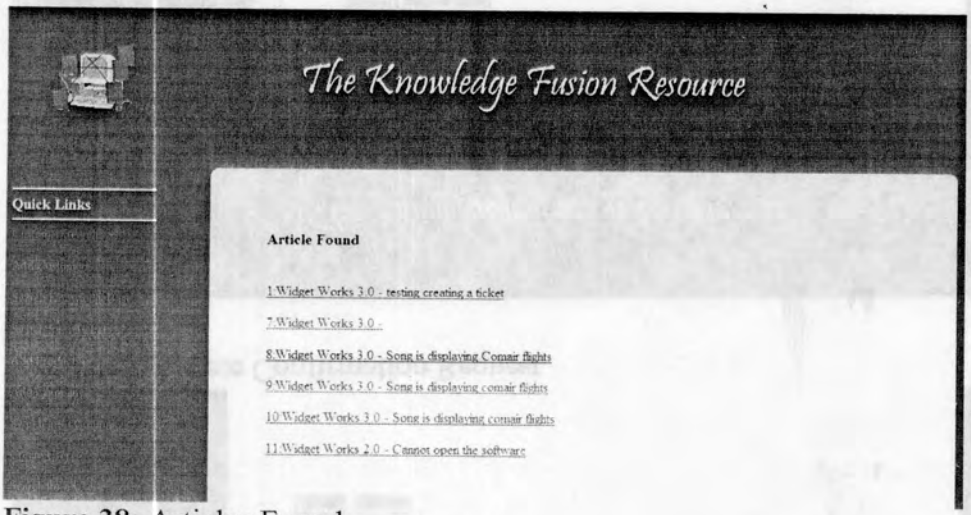


Figure 38: Articles Found

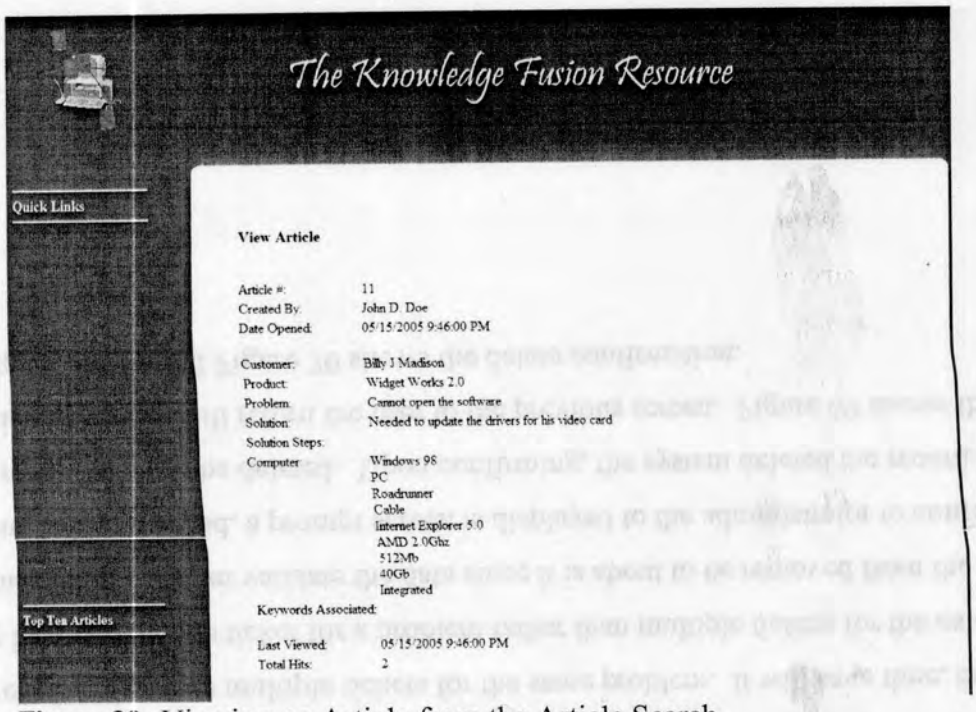


Figure 39: Viewing an Article from the Article Search

Another way to search for articles is from viewing the customer details after performing a customer search. The user can search the knowledgebase for articles related to the customer that is being viewed after the customer search by simply clicking the "Search Knowledgebase" button. The system will automatically search the knowledgebase for any articles that have the customer listed as the customer in the article and display a list of articles found, very similar to the list in Figure 38, only the name of the customer that was searched for is displayed at the top of the list. Figure 40 shows the list of articles found when searching by customer. Clicking on the link will display the article details in exactly the same manner as shown in Figure 39.

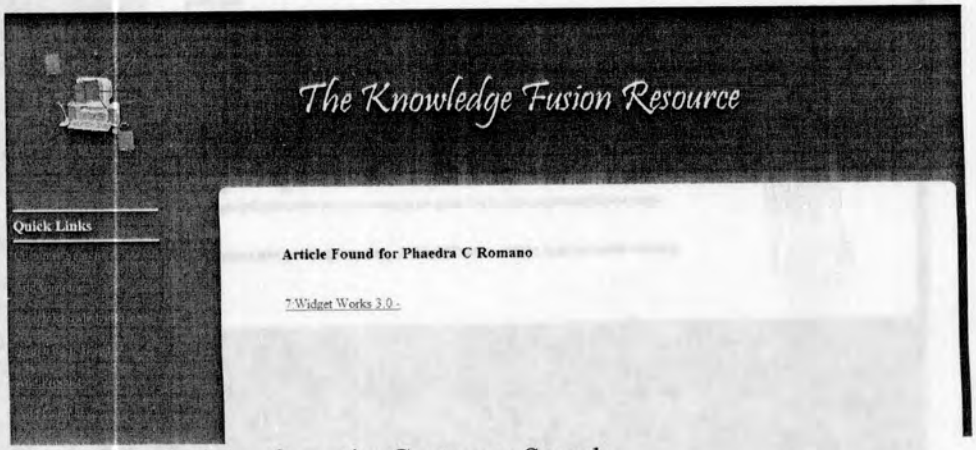


Figure 40: Articles from the Customer Search

5.5.2.3 Searching for Tickets

The ticket search is a useful tool in seeing how many tickets are open matching certain criteria. It is also useful for technicians to look up tickets they have sent to the programming department and check the status of those tickets. In the event that a ticket sent to the programming department has not made any progress, the technician will be able to see this. Also, if a customer calls in and wishes to know the status of a ticket being worked on for them, any technician will be able to look up the ticket and give the customer the necessary information.

The ticket search occurs in one way, the user will click the link in the side bar called "Search for a Ticket". This will bring up the search form, as seen in Figure 41. There are five different tickets searches that can be performed. One is the search by ticket id. If the user knows the id, the number can be typed in and the exact article will be displayed in a non-editable format. The operating system, problem, and keyword searches are identical to those discussed in the knowledgebase article search. Figure 42 shows the search results for an operating system search for "%Windows%", and figure 43 shows the details of one of the tickets from Figure 42.

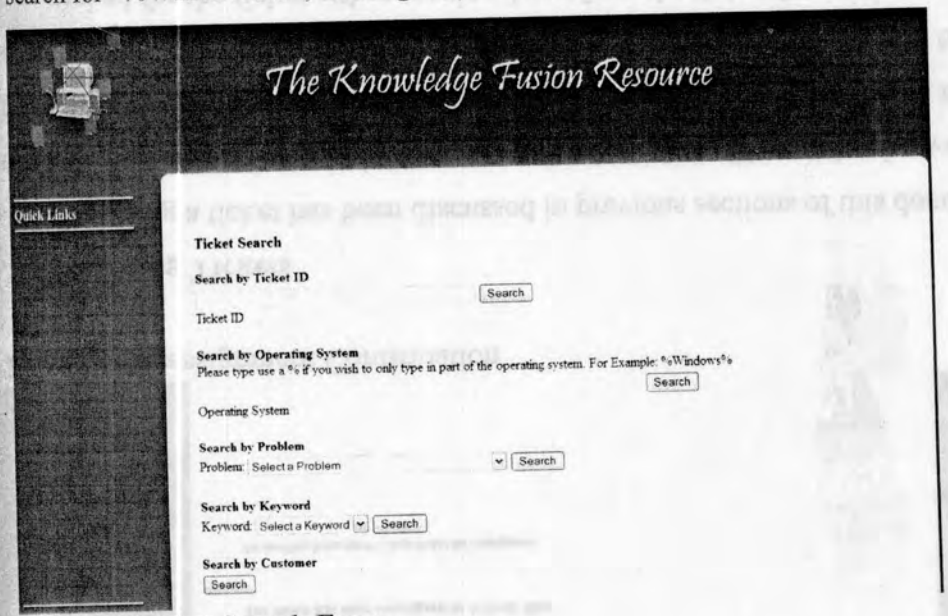


Figure 41: Ticket Search Form

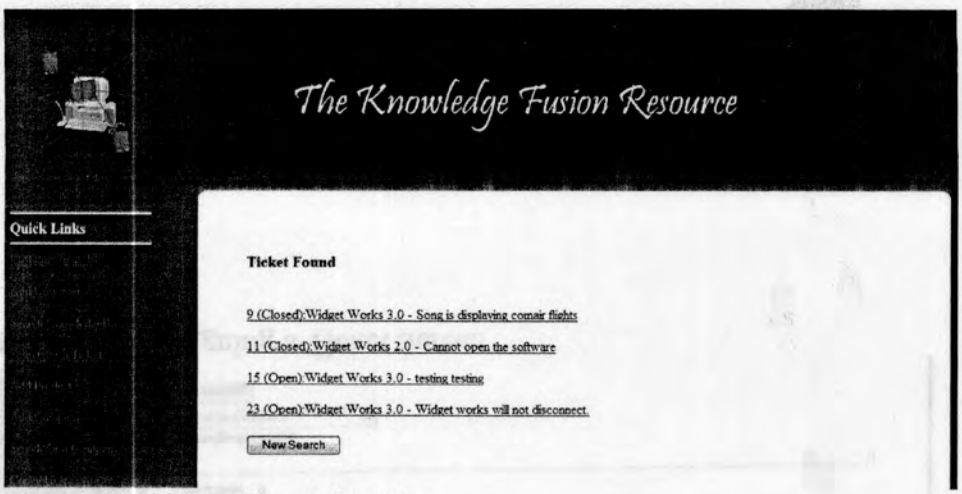


Figure 42: Ticket Search Results

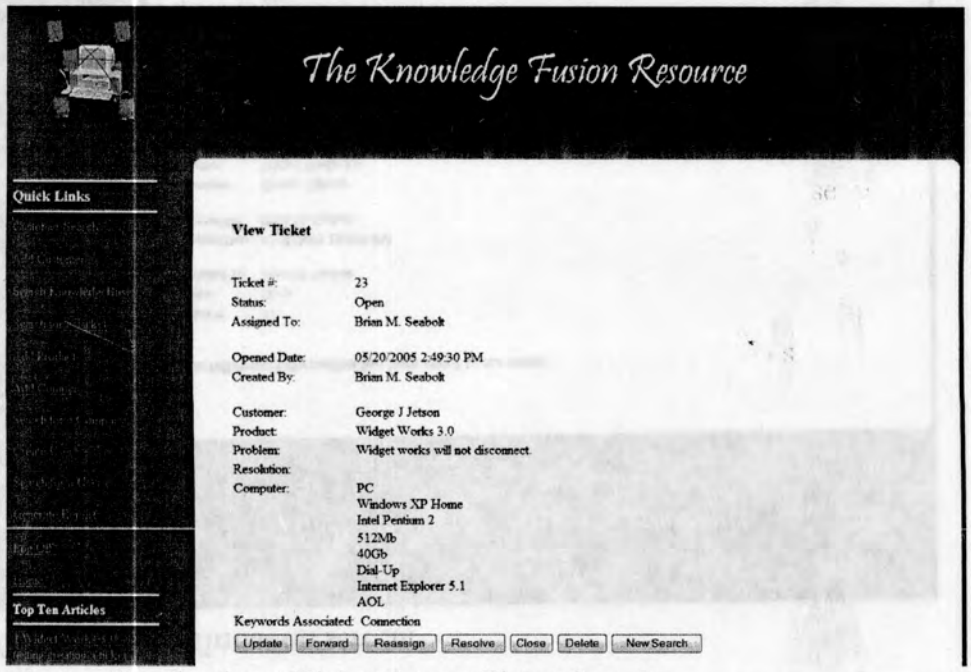


Figure 43: Viewing a Ticket from a Ticket Search

The last search option is by customer which takes the user to the customer search form and passes the ticket id behind the scenes during all of the screens previously shown in the section about the customer search. After searching for a customer in the customer search form, a list of tickets for that customer will be displayed. Figure 44 shows the search results of performing a ticket search through the customer search.

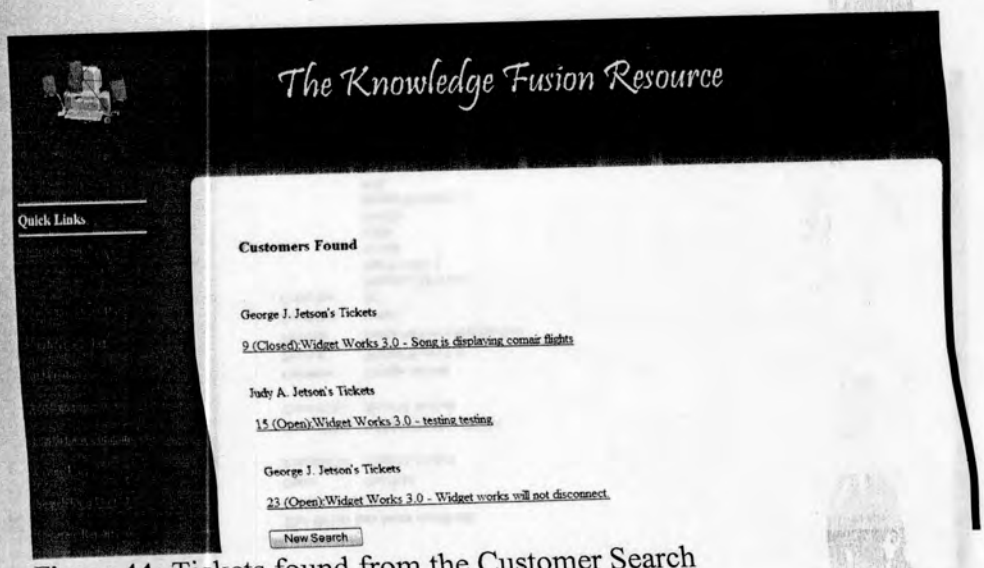


Figure 44: Tickets found from the Customer Search

5.5.2.4 Searching for Companies

Searching for a company is a very simple form. The form has two types of searches, by contact name and by company id. Searching by company id appears to the user as searching by company name due to the drop down list displaying company names. If the user cannot remember the company that a particular contact person works for, the user can search for the person's name to find the company. This search can also be performed using the "%" to search for any number of characters in the percent position. Figure 45 shows the company search form, Figure 46 shows a list of results from a contact name search, and Figure 47 shows the company details.

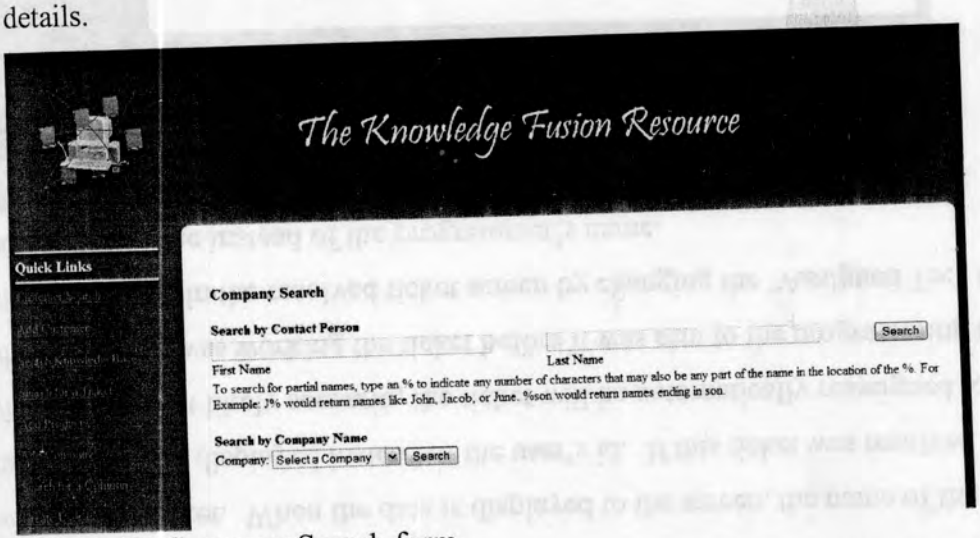


Figure 45: Company Search form

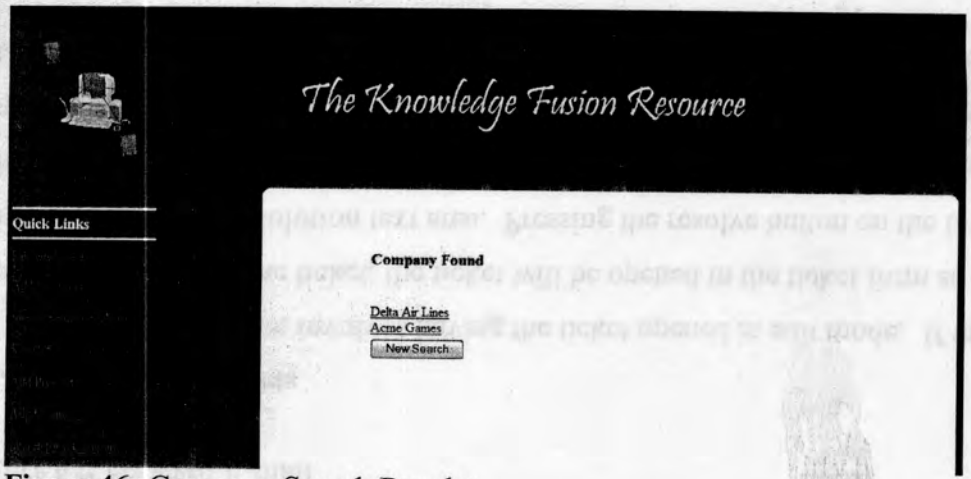


Figure 46: Company Search Results

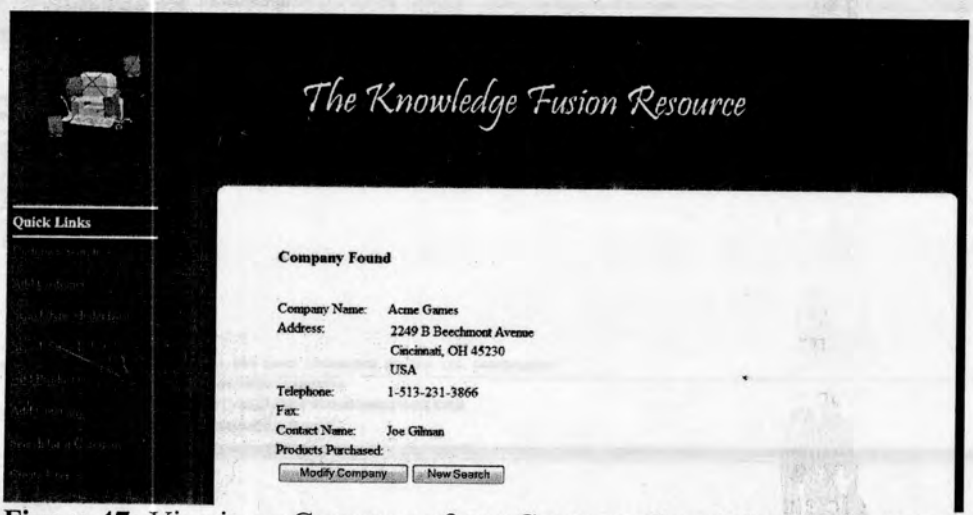


Figure 47: Viewing a Company after a Company Search (modify button only available for Administrators)

5.5.2.5 Searching for Users

The administrator is the only user who can search for other users. This functionality primarily exists in order to activate and inactivate users. This would be used in the case that a person has been fired or quit, but the administrator does not want the record for this person to be removed from the system since it is very likely that there are calls, tickets, or knowledgebase articles linked to this employee. For historical purposes, the employee information must remain in the database in order to know what calls, tickets, or articles the user is linked to.

The actual search form is very simple. It has two forms of searching, by name and by employee number. The employee number is different from the employee id because the id is an

auto-number field in the database and the employee number is a string since some companies have various ways of assigning employee numbers to new employees. Searching by name is just like searching by name in the customer search and the company search. The form can be seen in Figure 48 and the results of a user search for the first name "B%" is shown in Figure 49. Clicking on one of the links will display the user's information which can be seen in Figure 50.

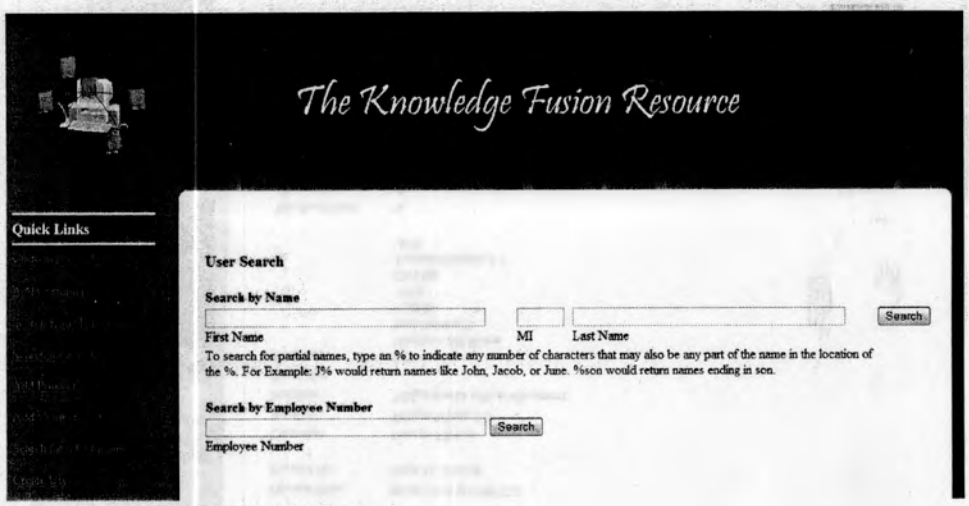


Figure 48: User Search form

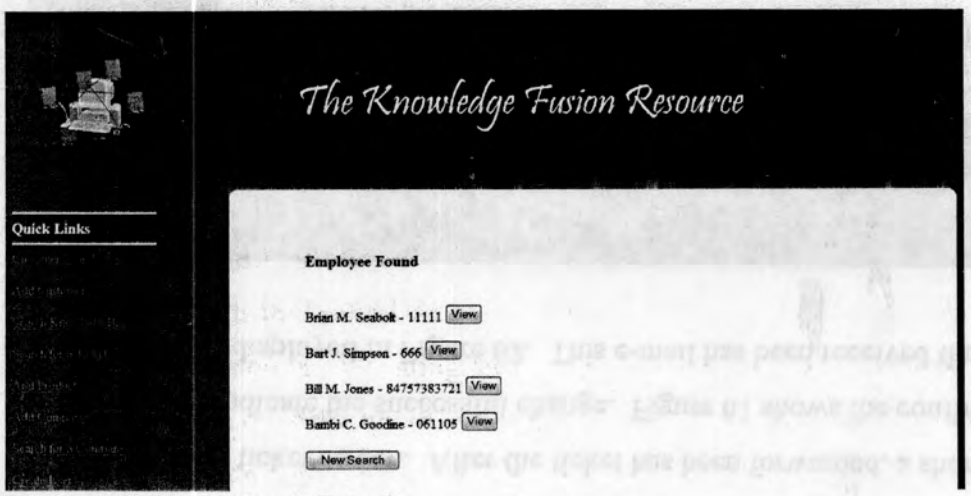


Figure 49: User Search Results

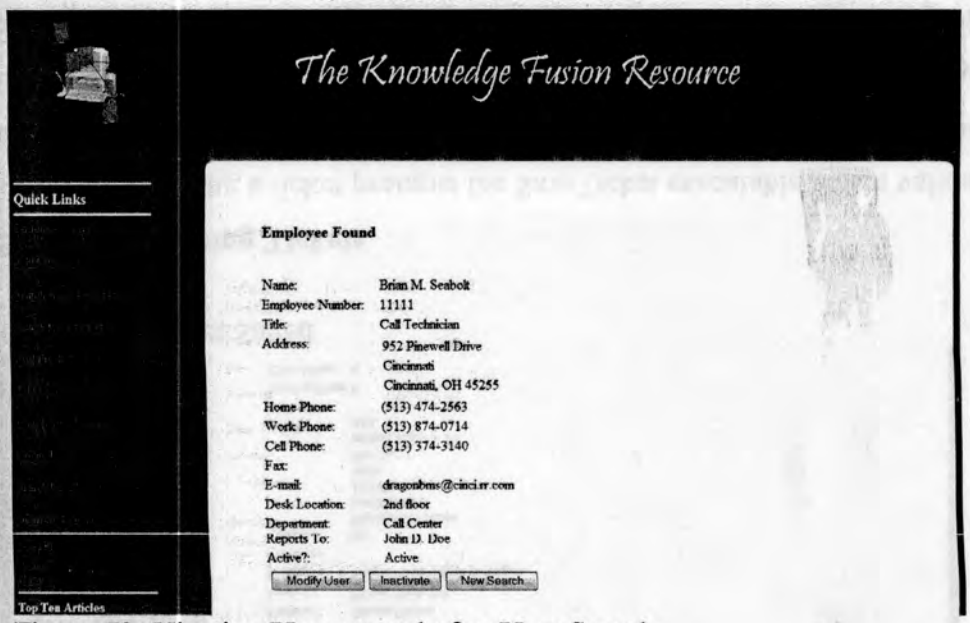


Figure 50: Viewing User record after User Search

5.5.3 Updating Records

All of the users have the ability to update some of the records. Technicians can update all of the records having to do with tickets. The administrator has the ability to update records having to do with tickets, companies and users. The programmer has the ability to modify only the problem, solution, and computer records. Each of these record modifications are validated prior to submitting to the database.

5.5.3.1 Technician Updating Records

The records the technician can update are the ticket record, problem record, solution, computer, and customer. Updating the ticket can occur by searching for a ticketing and pressing the modify button once the ticket has been chosen. This will open up the ticket form with all the data pre-filled in the fields. Prior to saving any of the changes the data is validated and then saved in the same manner as described in the adding records section, seen in Figure 51. From the ticket form, clicking the edit problem form will bring up the problem form with the selected problem's details pre-filled in the form. This can be seen in Figure 52. Updating the solution and the computer works exactly the same way. Figures 53 and 54 show the solution and computer forms respectively with the pre-filled data.

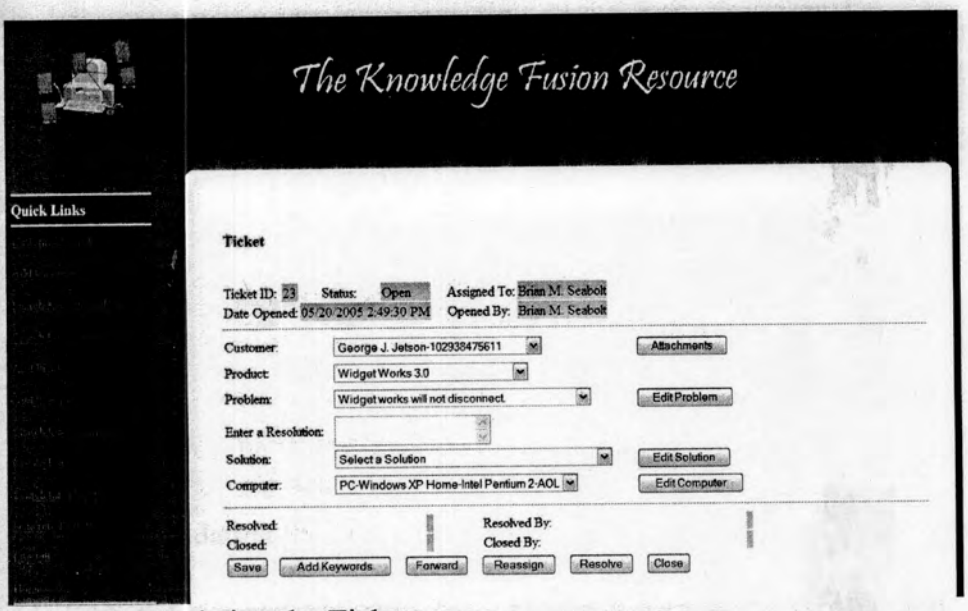


Figure 51: Updating the Ticket

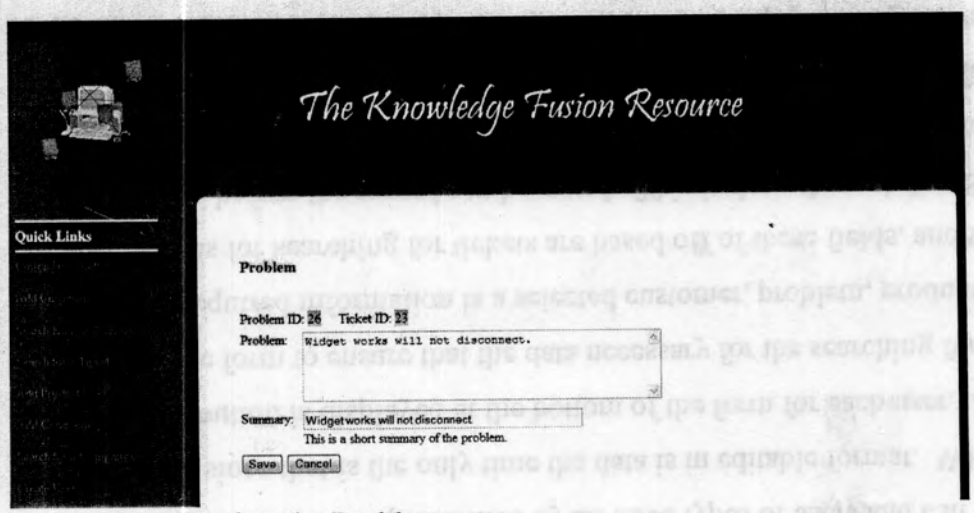


Figure 52: Updating the Problem

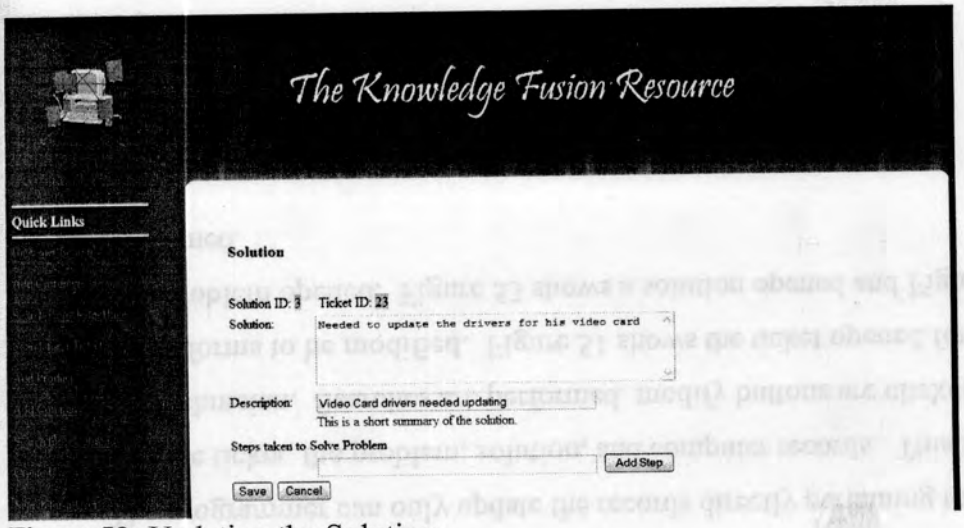


Figure 53: Updating the Solution

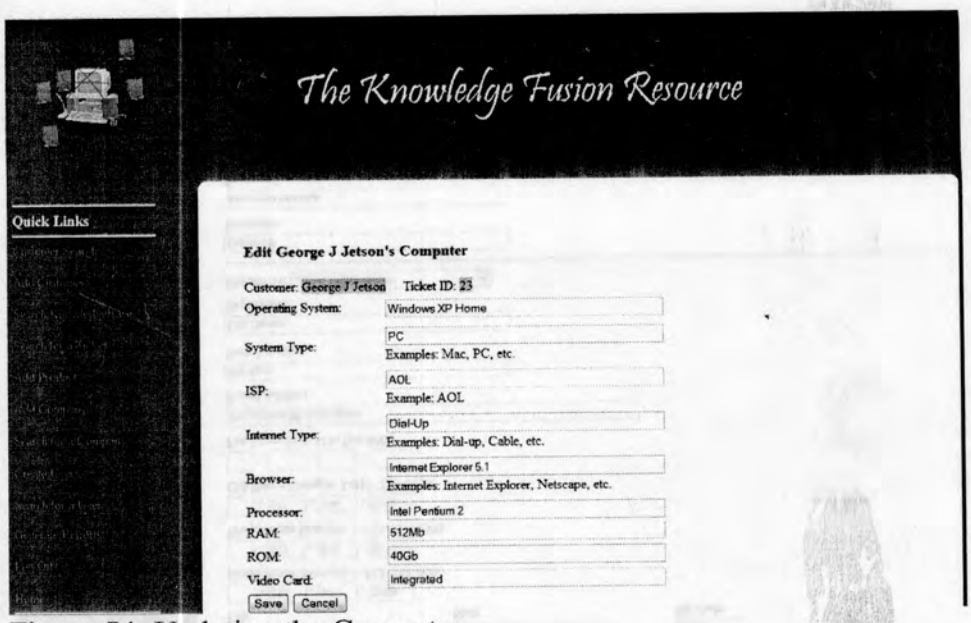


Figure 54: Updating the Computer

The last record the technician can update is the customer record. To modify the customer record, the technician performs a customer search and once the exact customer has been located and displayed to the screen, the technician can click the modify button to open the customer form with the data pre-filled. Figure 55 shows the customer form showing a customer record for modification. This form cannot be submitted without being validated.

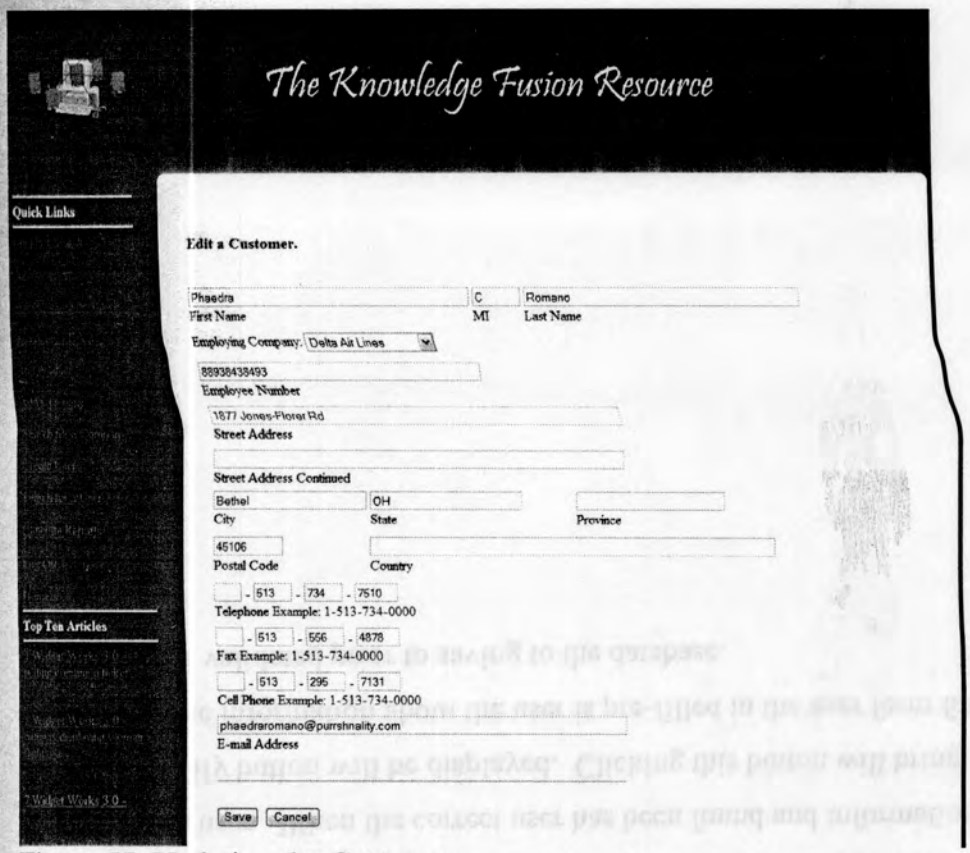


Figure 55: Updating the Customer

5.5.3.2 Administrator Updating Records

The administrator has the same updating functionality as the technician. In addition to those, the administrator can update companies and users. Both of these are performed in the same way. A search for the company results in a company being found. For the administrator, a modify button will be displayed and the company can then be opened in the company form for editing. Figure 56 shows the company in edit mode.

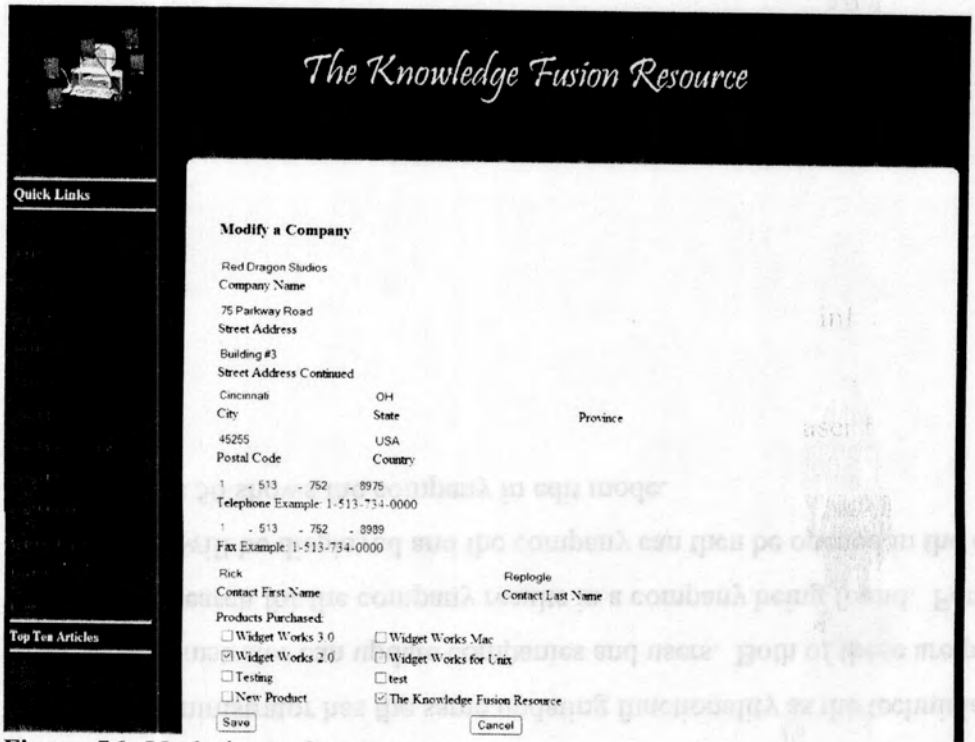


Figure 56: Updating a Company

Updating a user is exactly the same as updating a company. First the administrator searches for a user. When the correct user has been found and information displayed to the screen, a modify button will be displayed. Clicking this button will bring up the screen shown in Figure 57. The information about the user is pre-filled in the user form for editing. The information is validated prior to saving to the database.

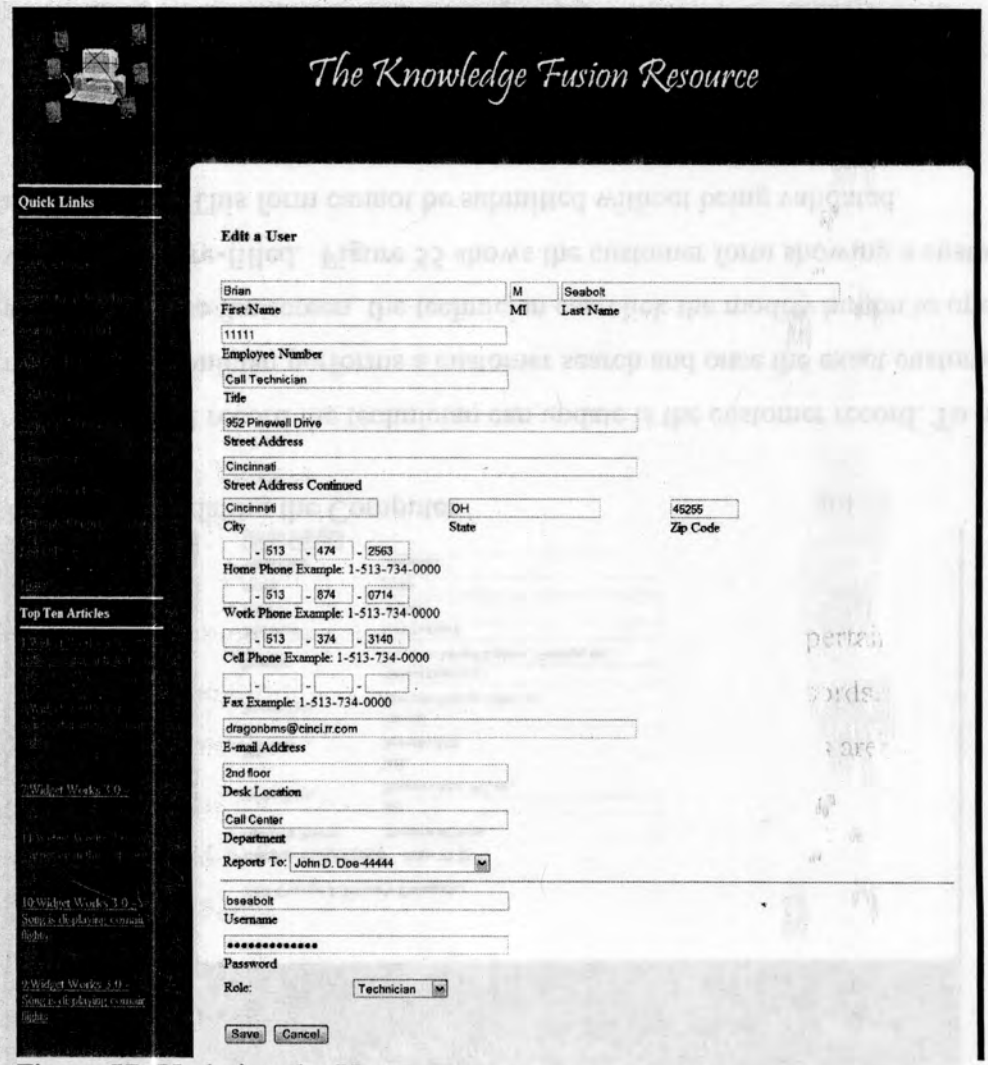


Figure 57: Updating the User

5.5.3.3 Programmer Updating Records

The programmer can only update the records directly pertaining to a ticket. He or she can modify the ticket, the problem, solution, and computer records. This is all done in the same way as the technician. Searches are performed, modify buttons are clicked, and records are opened up in forms to be modified. Figure 51 shows the ticket opened for modification. Figure 52 shows a problem opened. Figure 53 shows a solution opened and Figure 54 shows a computer opened.

5.5.4 Ticketing Actions

There are several actions that can be taken from the ticket form and from viewing a ticket. These actions are based on the role of the user looking at the ticket. From the ticket form, all of the actions force the ticket record to be saved prior to the action being performed. From viewing the ticket, saving the ticket data is skipped and the necessary screens to perform the action are displayed. These actions include saving, forwarding, resolving, reassigning, closing and deleting tickets.

5.5.4.1 Saving Tickets

Saving tickets can be performed by all three types of users and can only take place from the ticket form since that is the only time the data is in editable format. When the ticket form is open, the save button is displayed at the bottom of the form for each user. Pressing this button will validate the form to ensure that the data necessary for the searching functionality has been entered. The required information is a selected customer, problem, product, and computer. The different options for searching for tickets are based off of these fields, and therefore the data must be entered before the record can be saved. This is done through JavaScript which prevents the form from being submitted until the data is entered. Upon saving the ticket, the details of the ticket are displayed to the screen for verification of the save being successful and so the user can keep track of the information via printing the screen or simply writing down something like the ticket number for a quick search later. Figure 58 shows the data validation, and Figure 59 shows a saved ticket.

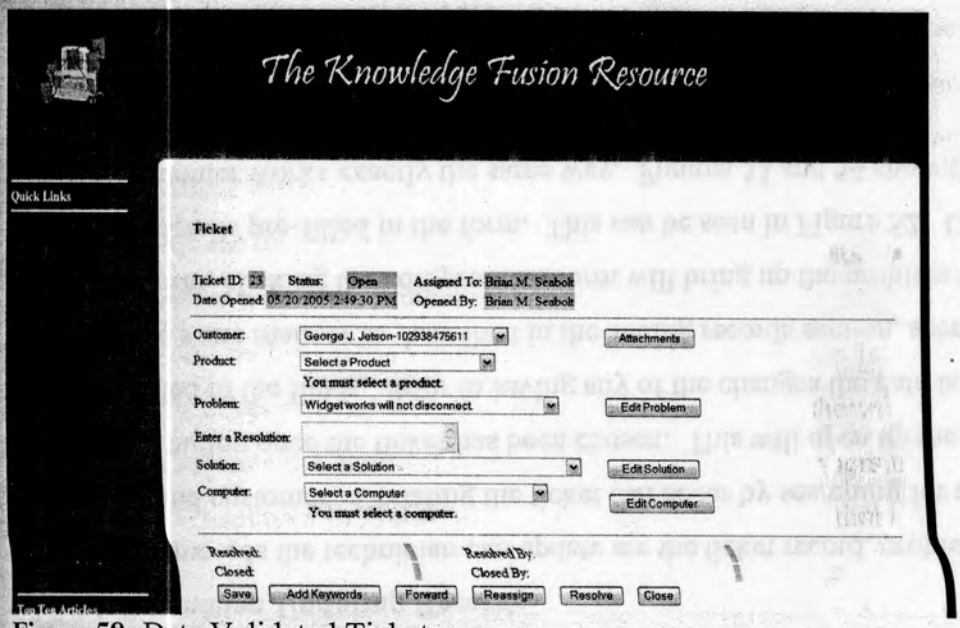


Figure 58: Data Validated Ticket

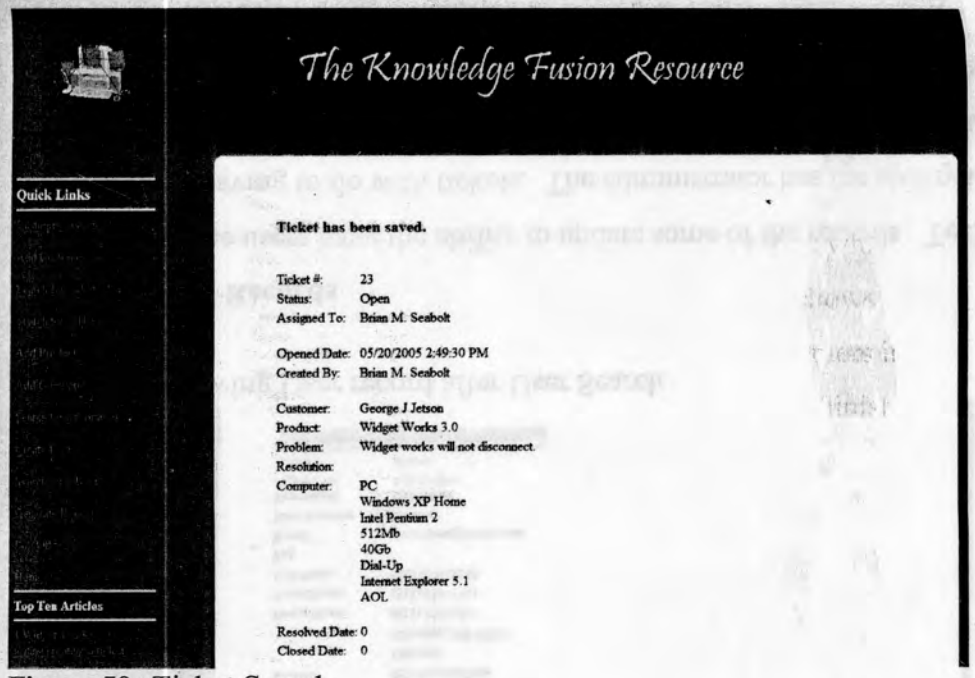


Figure 59: Ticket Saved

5.5.4.2 Forwarding Tickets

Forwarding a ticket prompts the SaveTicket executable which validates and saves the ticket prior to displaying the forwarding screen. This screen shows all of the ticket's saved information and at the bottom has a list box of all the programmers in the system since a ticket can only be forwarded to a programmer. The technician or administrator will select the

programmer to send this ticket to and press the forward button. Upon forwarding the ticket, the programmer who is being assigned to the ticket will receive an e-mail informing him or her of the reassignment. If this action was taken by an administrator, the technician previously assigned to the ticket will also receive an e-mail stating the change. Technicians and administrators are the only users who have this functionality. Figure 60 shows the forward ticket screen which includes the saved ticket information. This screen is what is displayed if the user clicked the forward button on the viewing ticket screen. The ticket form is skipped when coming from the viewing ticket screen. After the ticket has been forwarded, a short message is displayed to the screen to indicate the successful change. Figure 61 shows the confirmation screen. A sample e-mail is displayed in Figure 62. This e-mail has been received through Microsoft Outlook.

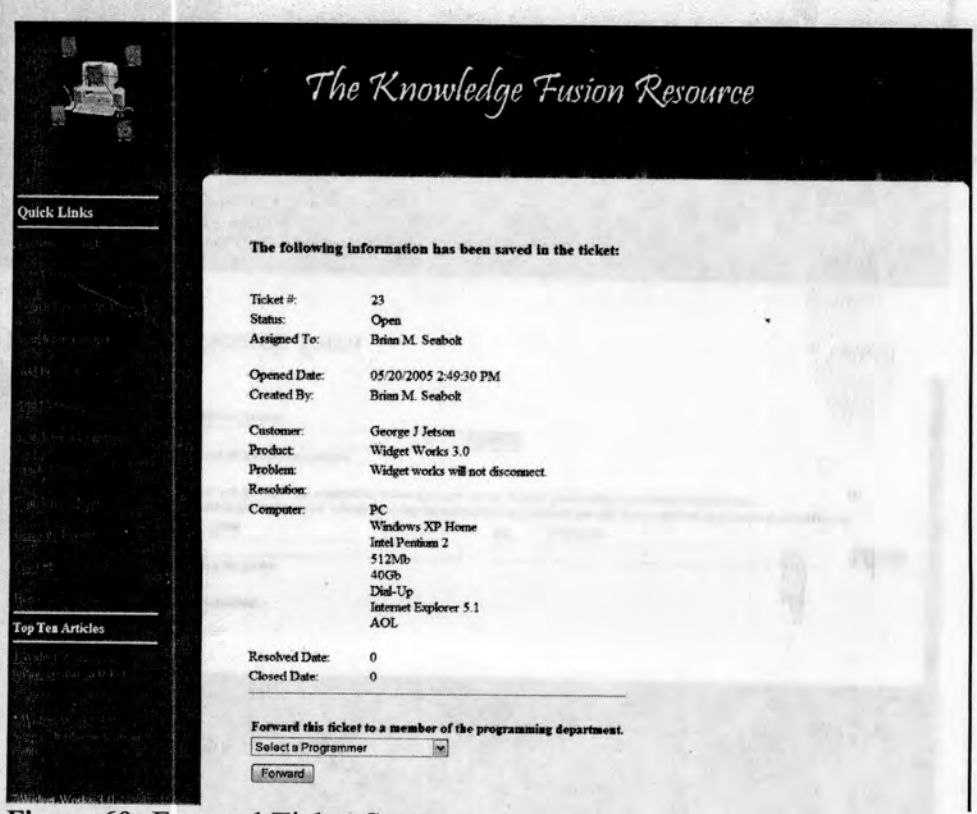


Figure 60: Forward Ticket Screen

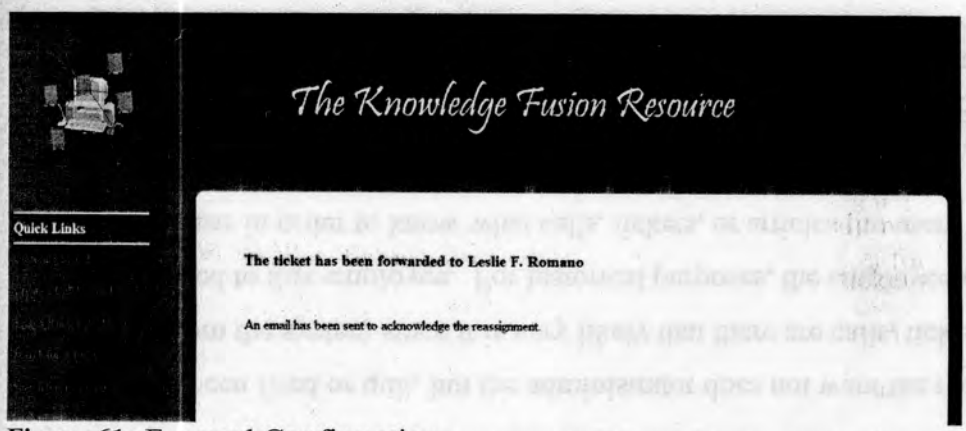


Figure 61: Forward Confirmation

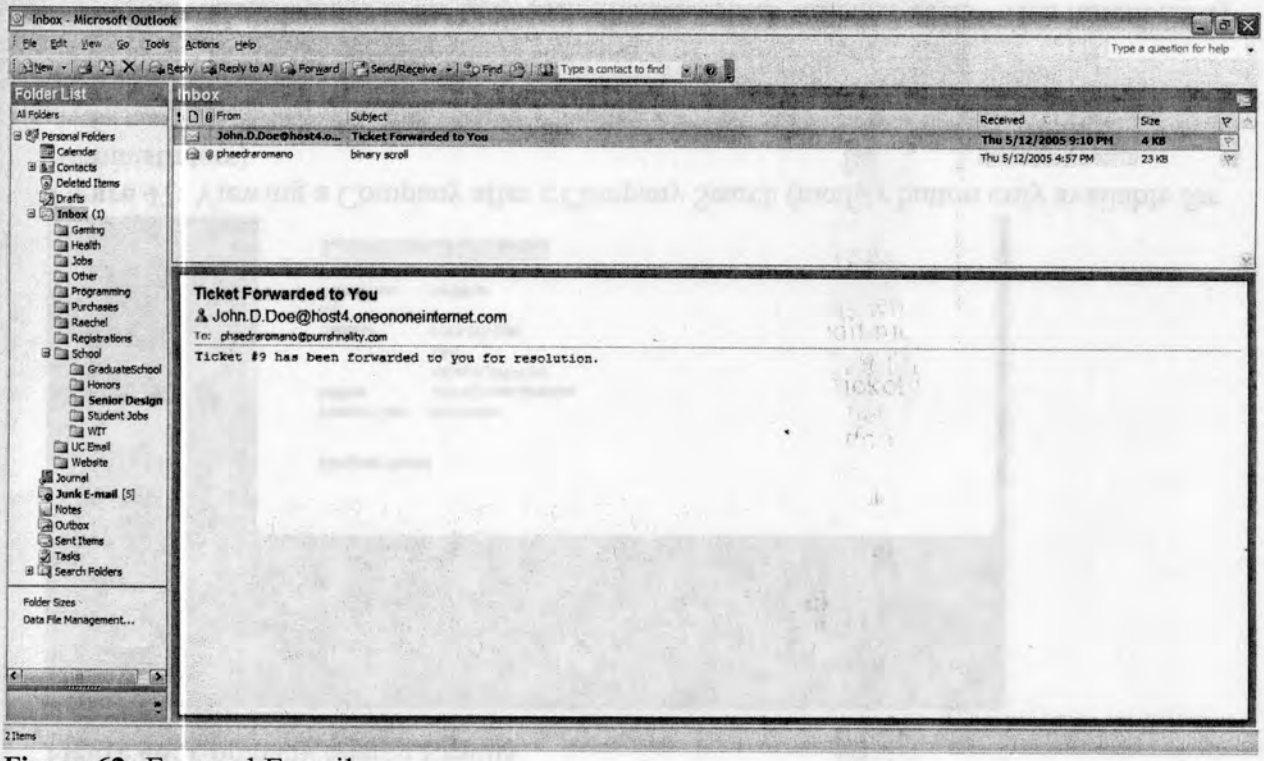


Figure 62: Forward E-mail

5.5.4.3 Resolving Tickets

Resolving a ticket involves having the ticket opened in edit mode. If the resolve button is pressed from viewing the ticket, the ticket will be opened in the ticket form so that a resolution may be added in the resolution text area. Pressing the resolve button on the ticket form will validate the form. If no information has been entered into the resolution box, a message will be displayed to the screen telling the user to enter in a resolution, as shown in Figure 63. Once the resolution has been entered and all of the data is correct, the ticket will be saved and the

information will be displayed as a confirmation to the screen, as shown in Figure 64. The resolution date will be automatically entered into the database as well as the id of the person resolving the ticket. When the data is displayed to the screen, the name of the person resolving the ticket will be displayed instead of the user's id. If this ticket was resolved by a programmer which is the most likely scenario, the ticket will be automatically reassigned to the technician who previously was working the ticket before it was sent to the programming department. This will be reflected in the resolved ticket screen by changing the "Assigned To:" to have the technician's name instead of the programmer's name.

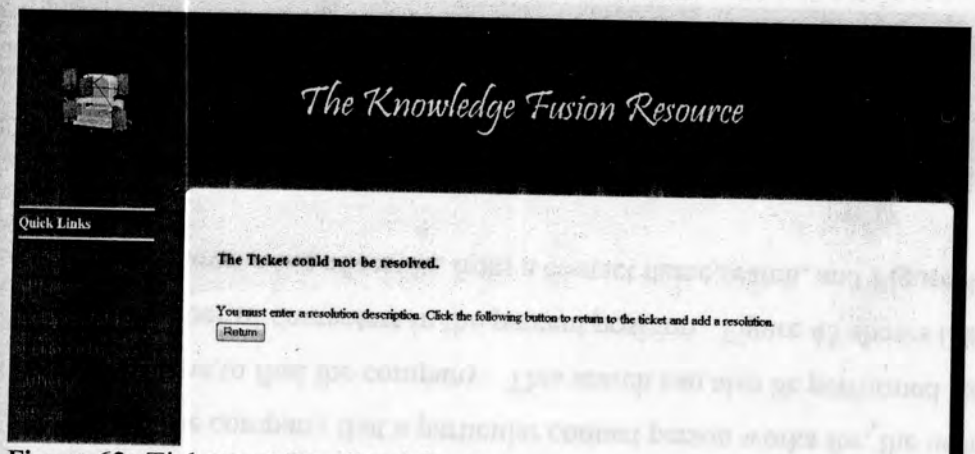


Figure 63: Ticket not Resolved Screen

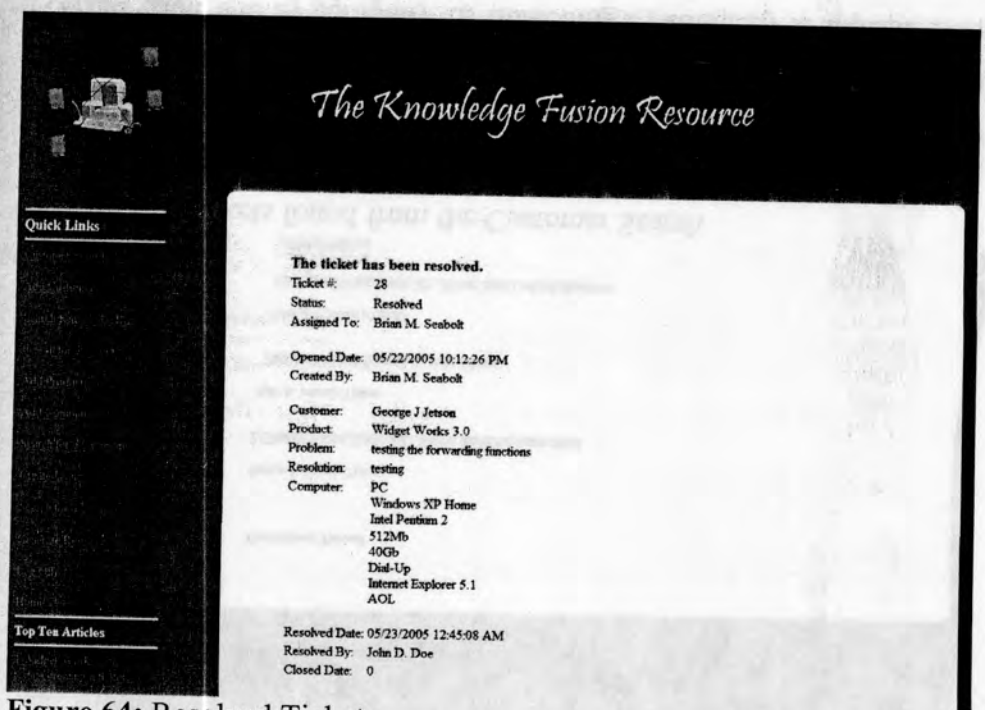


Figure 64: Resolved Ticket

5.5.4.4 Reassigning Tickets

Reassigning a ticket works exactly the same as forwarding a ticket. Only the administrator has this functionality. It can be done from the ticket form or from viewing the ticket details. Both the form and the details screen have a button that says "Reassign." From the ticket form, the ticket is saved and the details are displayed to the screen and a selection box of all the employees are listed below for reassignment, shown in Figure 65. The administrator chooses an employee and clicks the reassign button. An email is immediately sent to the person being reassigned the ticket and also sent to the person who was assigned the ticket previously. Then, a confirmation is displayed to the screen stating that the reassignment took place. Figure 66 shows the confirmation screen.

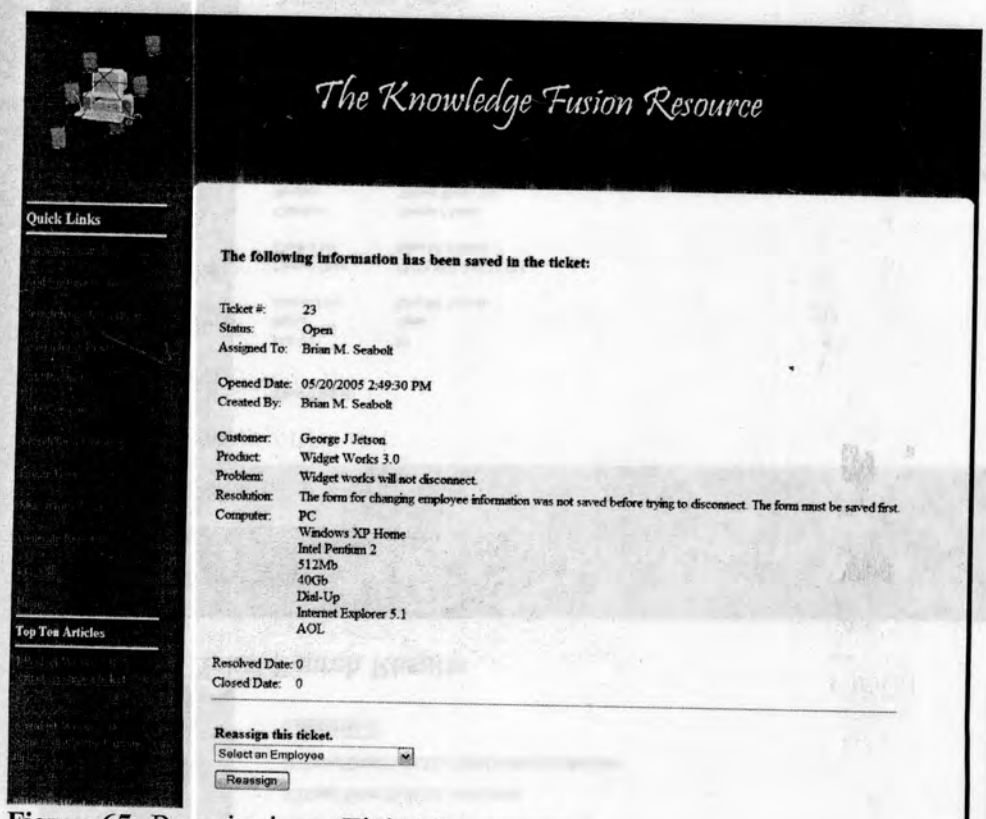


Figure 65: Reassigning a Ticket Screen

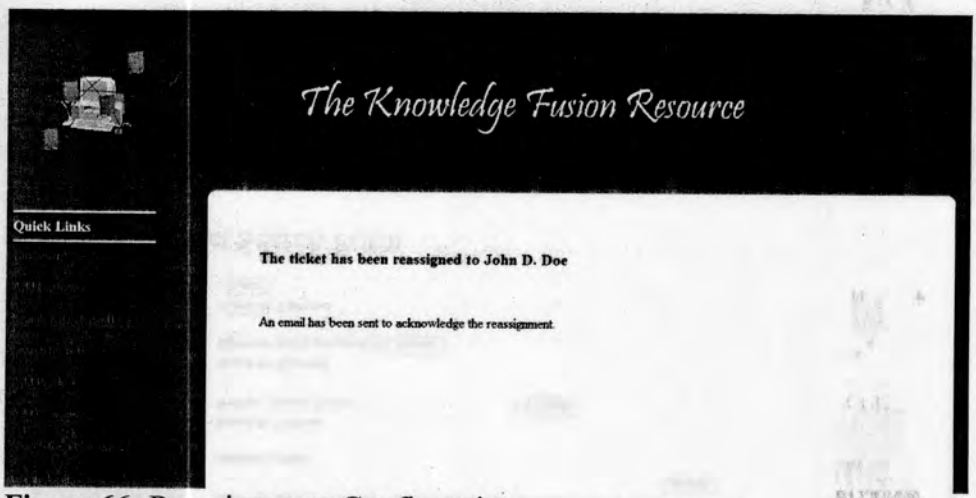


Figure 66: Reassignment Confirmation

5.5.4.5 Closing Tickets

Closing a ticket has been discussed in previous sections of this document, but to reiterate, only a technician and an administrator can close a ticket. This option is available at the bottom of the ticket form and the view ticket screen. Pressing the close button on the form validates the form which makes sure that a solution has been selected, shown in Figure 67. If a solution has been picked for the ticket either previously or from the ticket form, a screen displaying the ticket information will be displayed to the screen and an email will be sent to the customer informing him or her of the solution to the problem and that the ticket has been resolved. Figure 68 shows the ticket successfully closed.

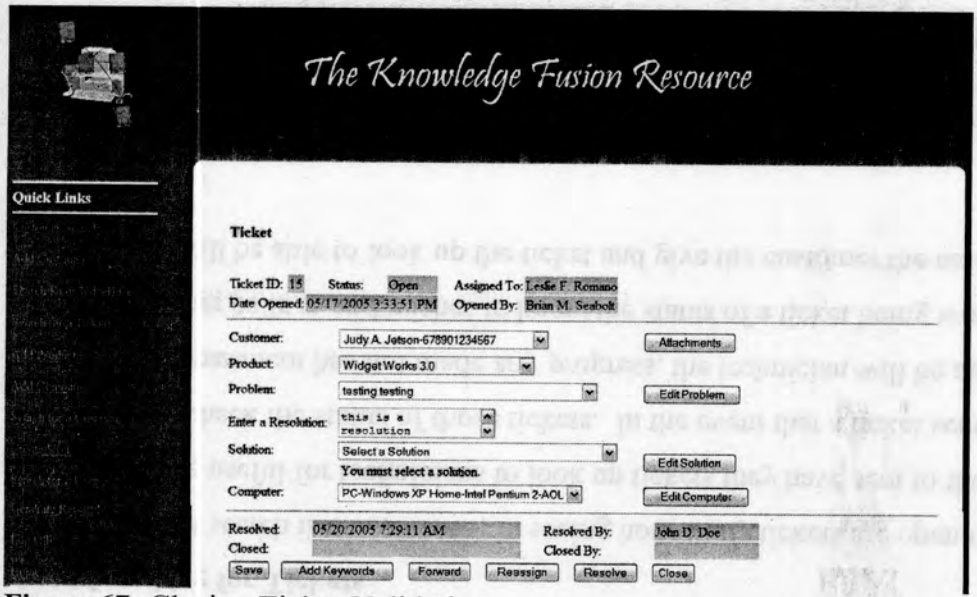


Figure 67: Closing Ticket Validation

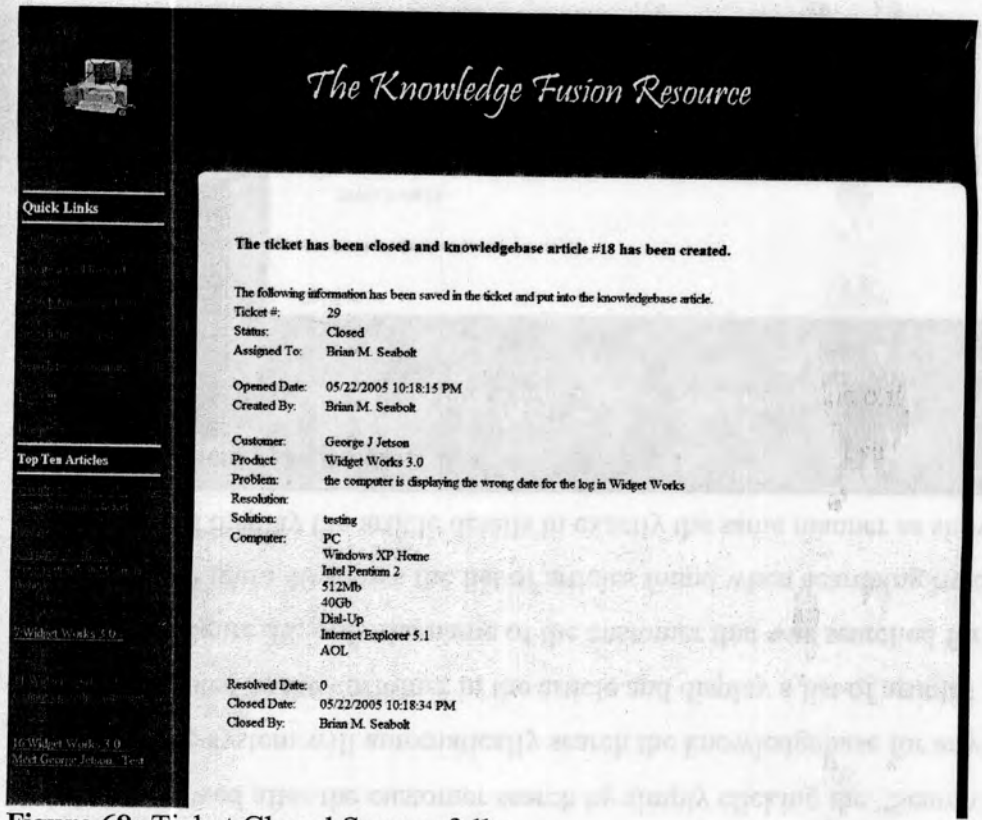


Figure 68: Ticket Closed Successfully

5.5.4.6 Deleting Tickets

Only an administrator can delete a ticket. This option has only been put in place in the event that there are multiple tickets for the same problem. It will save time, energy, and money by having only one ticket for a problem rather than multiple tickets for the exact same situation. This option does not validate the data since it is about to be removed from the system completely. Instead, a prompt screen is displayed to the administrator to confirm that this ticket is really meant to be deleted. Upon confirming, the system deleted the record from database. Clicking cancel will return the user to the previous screen. Figure 69 shows the confirmation request screen and Figure 70 shows the delete confirmation.

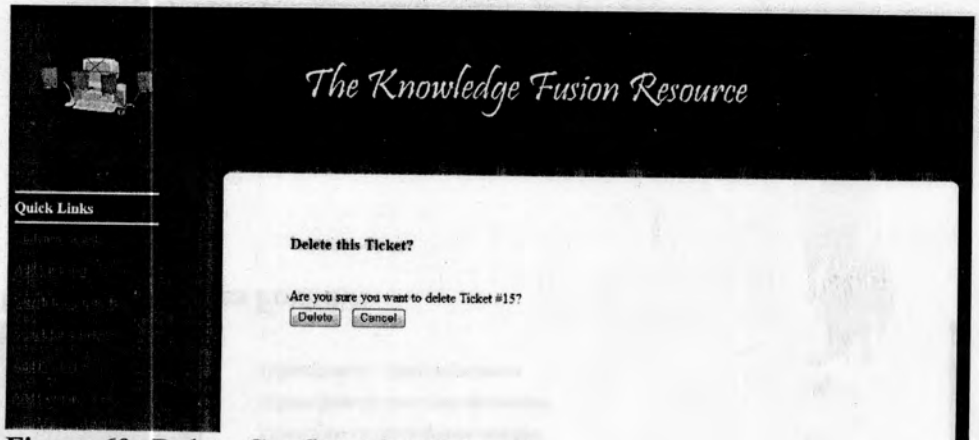


Figure 69: Delete Confirmation Request

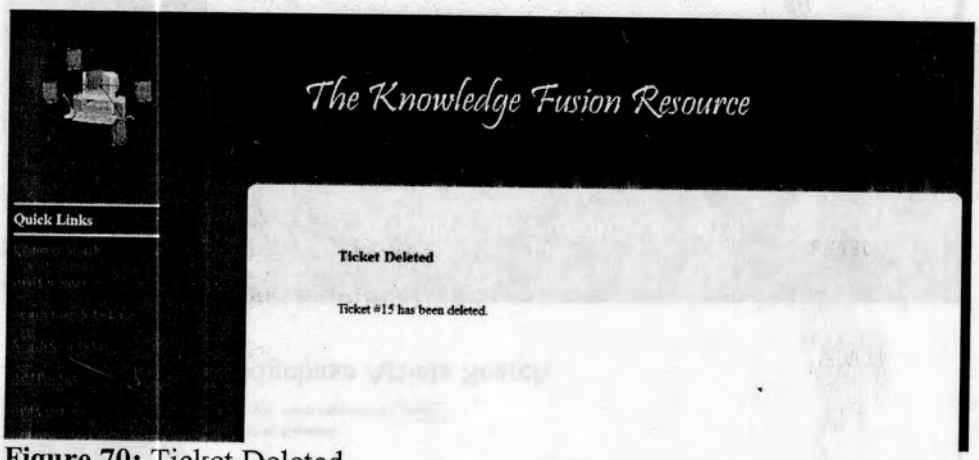


Figure 70: Ticket Deleted

5.5.5 Logging and Generating Reports

Behind the scenes, most of the actions that any user performs results in a record being generated in the Log table that keeps track of what action was taken, who took that action, and when the action was taken. These records are used primarily for generating reports that the administrator uses, but are also used to reassign tickets back to the previous technician after a programmer has resolved it. Figure 71 shows the screen with a list of reports available to the administrator.

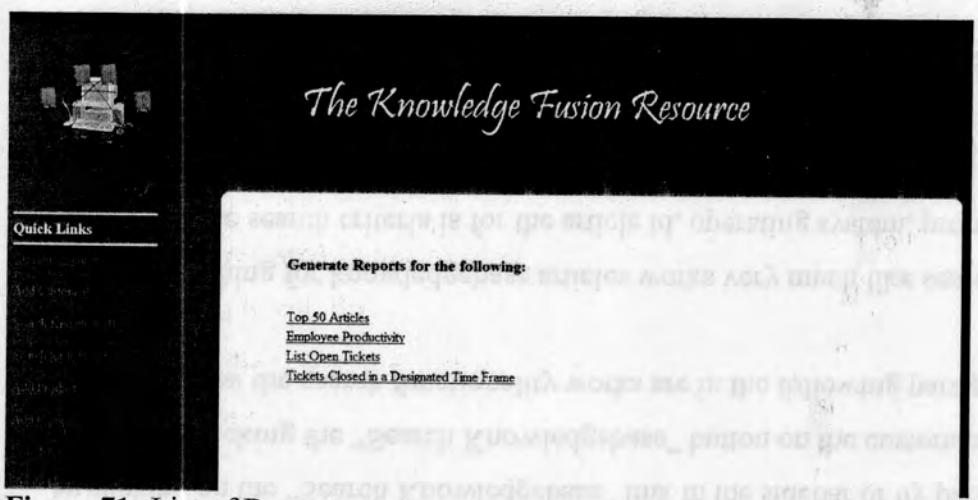


Figure 71: List of Reports

5.5.5.1 Top 50 Articles

The Top 50 Articles report is to show the administrator what articles have been accessed the most. This is to help draw conclusions on the types of problems that are occurring the most with their products, what issues need to be avoided in future products and version of old products, and whether a public announcement of how the problem to the customers should be made. The report looks at the Log table and pulls out records that show that an article was associated to a call record. This indicates that the article was used to solve the customer's problem successfully. These articles are then displayed to the screen in order of the most used to the least used in the form of links that can be clicked and then viewed on the screen. Figure 72 shows the report generated.

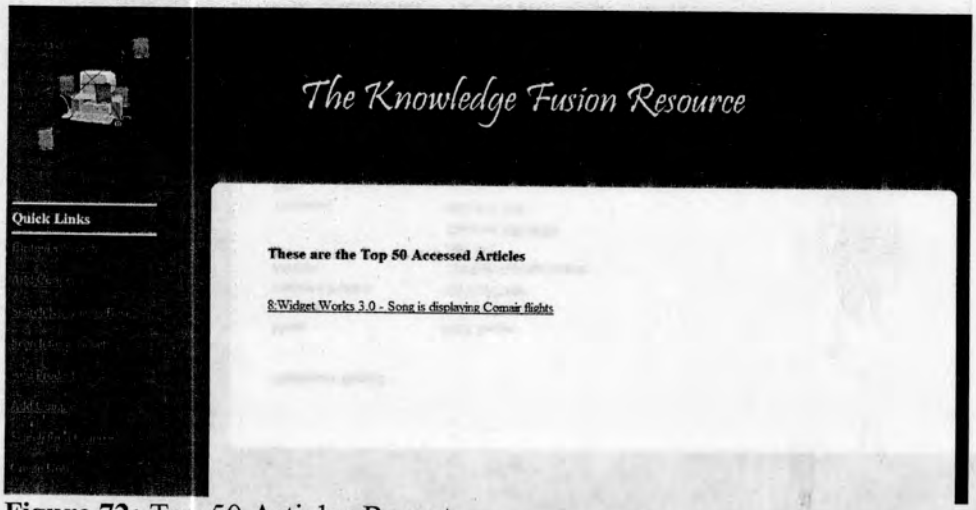


Figure 72: Top 50 Articles Report

5.5.5.2 Employee Productivity Report

The Employee Productivity Report lists all of the calls a user has created, the customer for the call, the article associated with the call if there is one, and the ticket associated with the call if there was one created. After listing the user's calls, the tickets the user has had any involvement in are listed. This is pulled from the Log table. If the user has taken any sort of action for a given ticket, the ticket will be listed. The information listed will include the ticket id as a link to view the ticket, the customer, the date the action was logged, what action took place, who it was logged by(which will be the user) and the current status of the ticket. The report generated lists the employee name and number, then the user's calls, and finally the user's tickets. Figure 73 shows the employee productivity report.

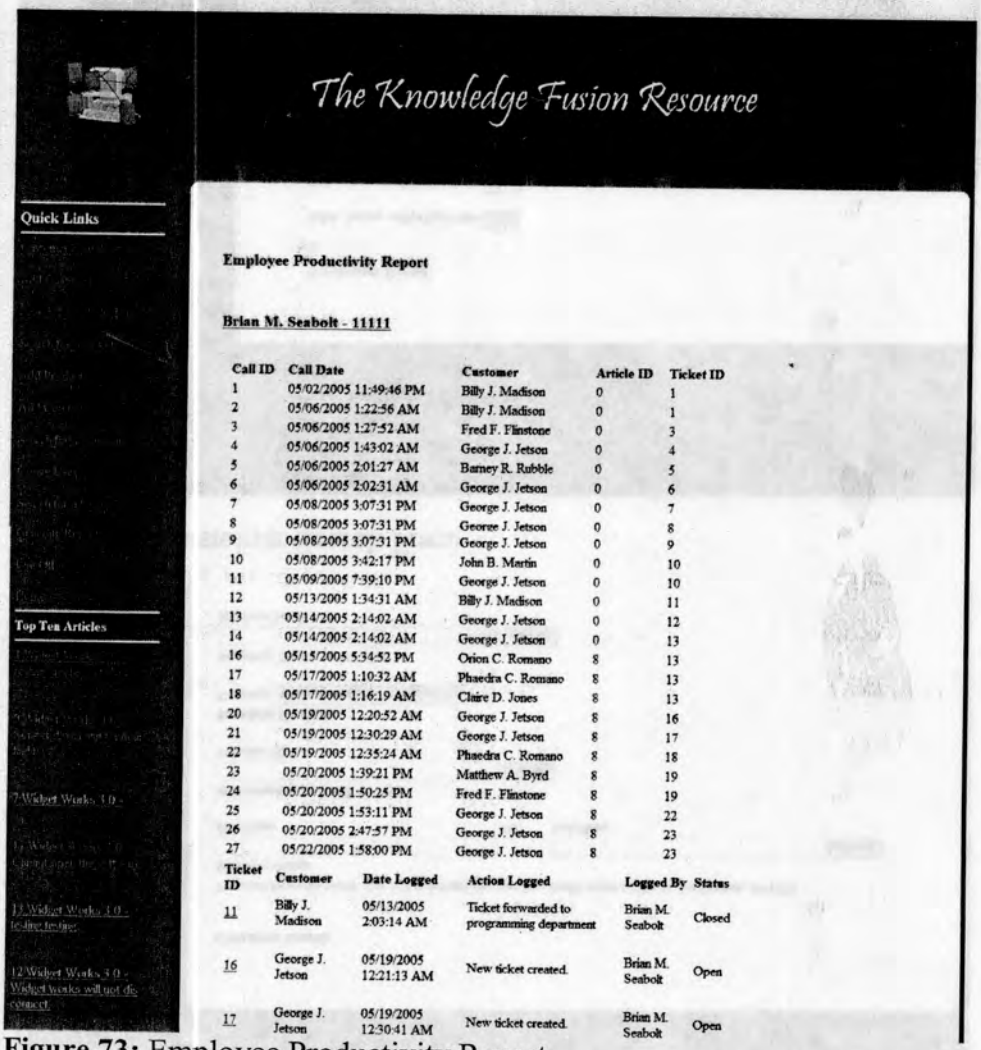


Figure 73: Employee Productivity Report

5.5.5.3 Open Tickets Report

Listing the open tickets of the system can be very useful in seeing what tickets are not being worked resolved. This report lists all of the open tickets in the system with some of the information about the ticket listed as well. The key feature of this report is that it lists the last time something was logged for this ticket and the time that has elapsed since anything was logged. This can give a quick insight into how long a ticket has been sitting without action. This allows the administrator to contact the user assigned to the ticket and find out why it has not been logged against for a while. Figure 74 shows the open tickets report.

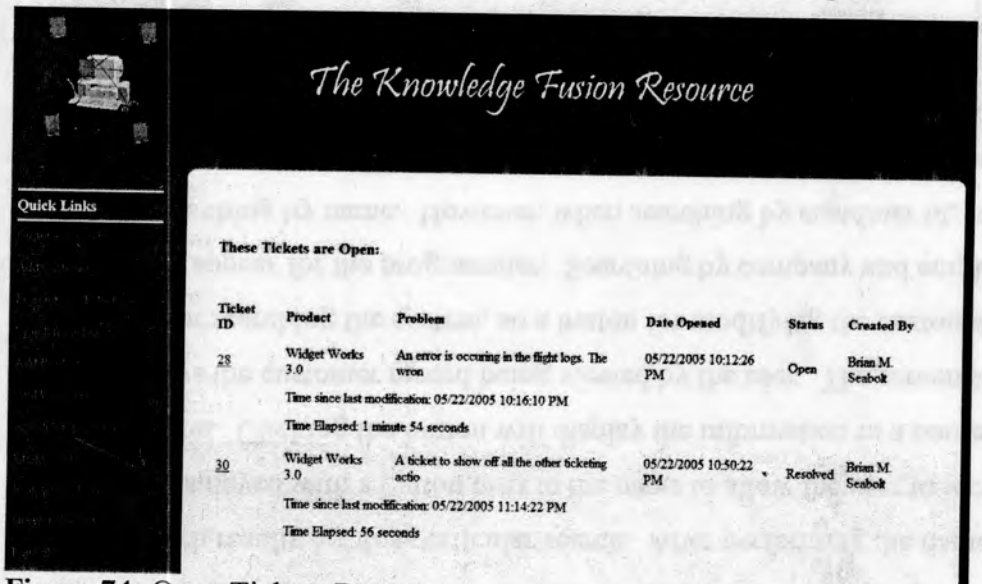


Figure 74: Open Tickets Reports

5.5.5.4 Ticket Closed Report

The tickets closed report is list of tickets that have been closed in a user-defined time frame. The report begins with a form asking for two dates to generate the report. If the start date is left empty, the SQL Server default date of 1900-01-01 00:00:00 AM is used. If the end date is left empty, the current date will be used. Figure 75 shows the form prompting the user for two dates. This form is validated to make sure that only numeric values are entered into the fields, but will allow for empty fields to be submitted due to the default values supplied in the code.

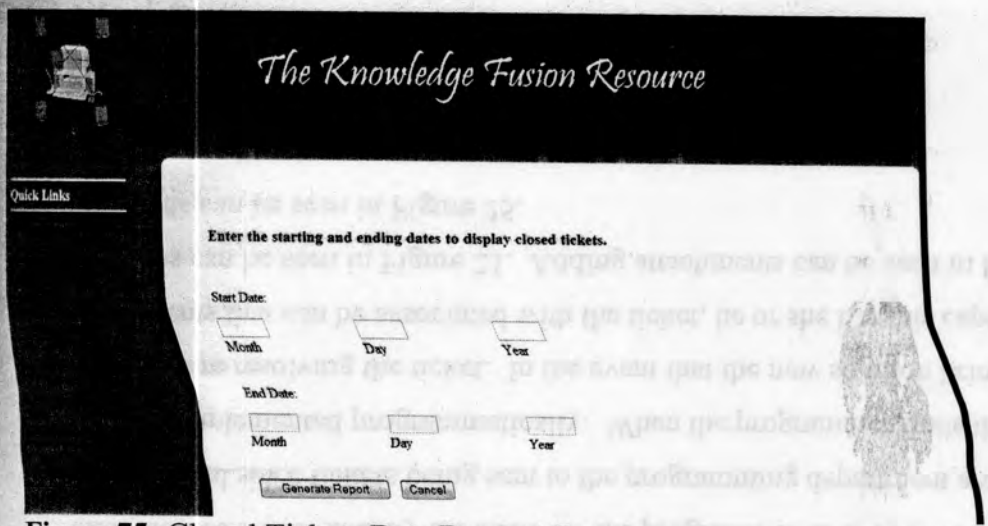


Figure 75: Closed Tickets Date Request Form

Upon submitting the form shown in Figure 75 by clicking the "Generate Report" button, all of the tickets closed during that time frame will be displayed to the screen in the form of links to a screen viewing the ticket details. The ticket id's are pulled from the Log table and then the information about the ticket displayed in the links are pulled from the actual ticket table. Figure 76 shows the closed tickets report for the time frame of 04/01/2005 to 05/30/2005.

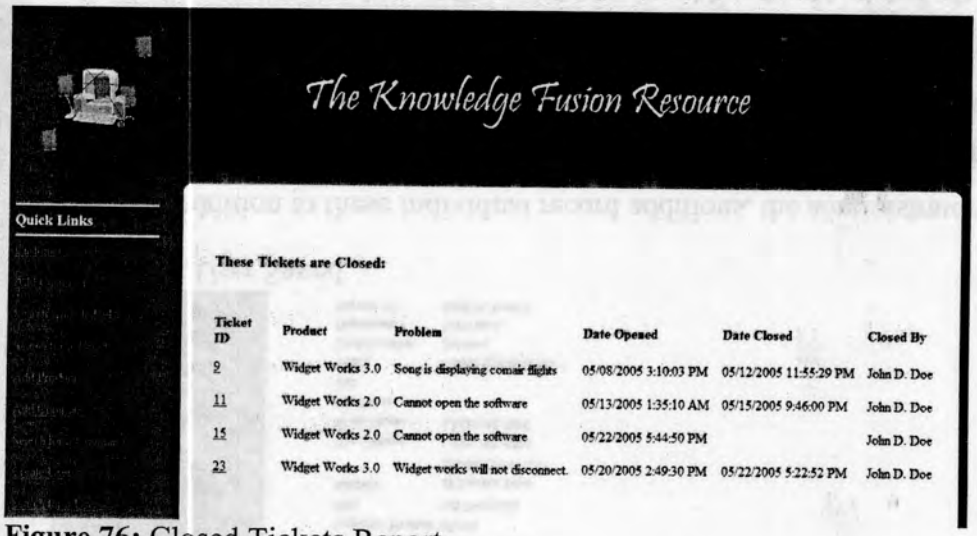


Figure 76: Closed Tickets Report

6. Testing Procedures

Testing the Knowledge Fusion Resource has been made easy with the code being very modular. Each set of html code displayed in the web browser has mostly been its own .exe file. The code was forced to be very modular in order for it to run properly through the web. This

made testing each piece very easy. When one form or screen was completed, I would load that page in the web browser to make sure it appeared as it should. After fixing any visual problems, I followed with testing how the page worked. In the event that the page loaded was a form, such as the "Create Call" form, I tested putting data in each of the fields and pressing all of the buttons on the form. This started first by pressing the "New Customer" button. Once the form for the new customer loaded, I viewed the source behind the form to make sure that all of the data from the call was hidden in the form. Then I created a customer, which included putting incorrect data into the fields to force data validation using JavaScript, and saved the customer. Once the customer was created, I went to the database and verified that the data was stored in the Customer table correctly. Returning to the call form, I verified that the new customer added was automatically selected for the call and all the call data entered prior to creating the customer was displayed as well. When I finished entering the call information, I closed to the can verified that the data was put into the database. This is how testing worked for all of the forms. I would put in incorrect data to make sure that the data validation worked properly and then would check the database to verify that the data was saved properly when the correct data was entered. When adding records, as stated in section 5.5.1, nearly all of the forms adding a record display a summary of the data added to the screen to also verify that the information was saved correctly. Whenever any data was displayed to the screen during testing, I would use Microsoft Query Analyzer to make sure that the data on the screen is the actual data in the database.

After all of the development was completed and I felt satisfied with how the project worked, I had a few people who do not operate help desks systems test it as well. If someone who does not know how a help desk system or knowledgebase works can use the project, anyone purchasing the project with previous knowledge of such systems will be able to operate it without much training involved. The object is to make the system as user-friendly as possible. The testing performed by these non-experts proved invaluable. These users attempted to perform operations that were otherwise not considered and were able to be fixed in a timely manner.

7. Conclusions and Recommendations

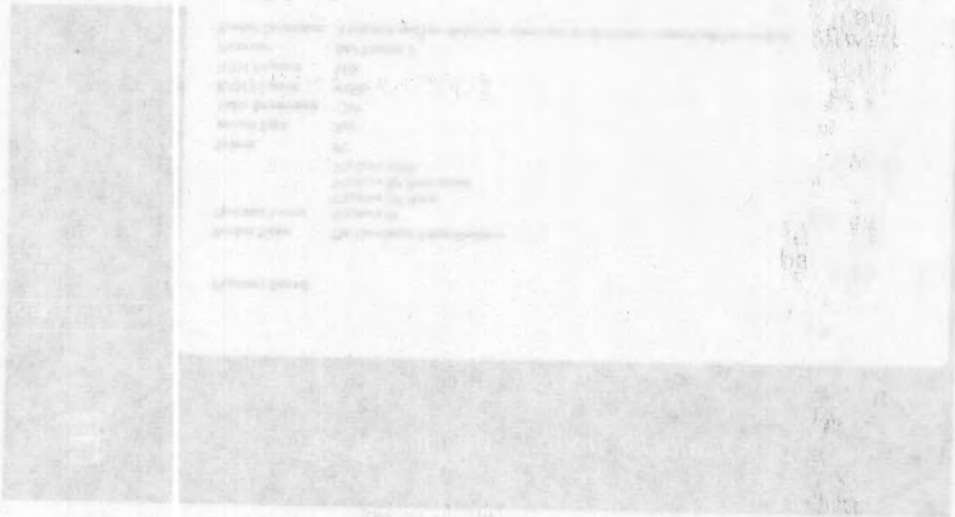
7.1 Conclusions

This project was requested by the company Flightline Software Incorporated to fulfill their need of a software application that is secure, dependable, fast, and scalable. The company could not afford to delegate this task to in-house programmers, so I offered to complete it for them. They required something that could be accessed through the internet, written in C++ and CGI, contained security measures, connected to a single database, and could be used by multiple users. These requirements make up the deliverables from Senior Design II and have been fulfilled. This project has been created using Microsoft Visual Studio .NET 2003 as a 32 Console Application utilizing visual C++ 6.0. Apache server was run on the development and testing machine to run the C++ executable files within Internet Explorer 5.1. Thorough testing has been completed, and a final finished product is ready to be sent to Flightline.

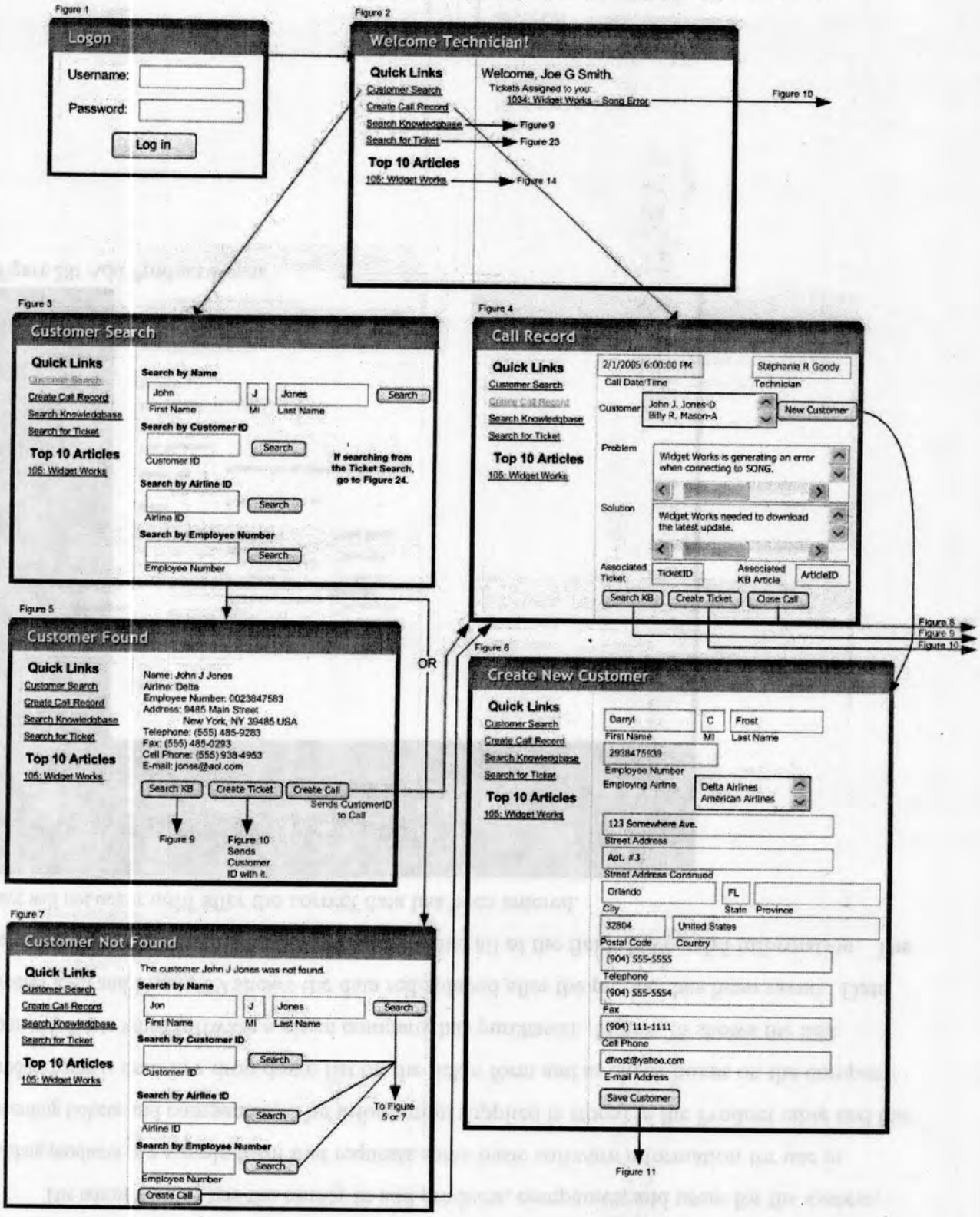
7.2 Recommendations

The Knowledge Fusion Resource has proved to be quite a challenge. I took C++ programming courses during my first year of college and this proved to be a disadvantage with such an extensive project. I was at a big disadvantage when I began because I had to refresh my memory on the very basic programming structures of C++ as well as a few that were never shown to me in classes, such as how to connect to Microsoft SQL Server or any database at all. I also had to learn how to run a C++ executable file in the web browser. Another disadvantage I had was that I never met with Flightline Software Incorporated directly. All of our communications were through e-mail. E-mail unfortunately does not lend itself to be very helpful if you are not a good communicator. There was much confusion in the beginning of the design process as to what exactly it was that I needed to design and develop. Having a design freeze in Senior Design II was a big consolation. I recommend that anyone creating a project like this should look into how the language they choose connects to databases, validates forms, and uploads files. Knowledge of how these three tasks are performed in the given language will speed up the development process tremendously rather than spending hours researching that could be spent developing.

Along with all of these hardships, I found that designing a professional looking website was also a big challenge. This project was very detailed and did not lend me time to spend on making it look professional. I was lucky enough to take a few design courses while taking Senior Design III that forced me to think about how I wanted this project to be represented. I highly recommend designated time to the overall presentation of the project, colors, graphics, and other multimedia elements. The way a project looks says something about how much time you spent on it and even says a little about the developer. This project identifies my capabilities. It pushed me to develop at a pace I did not think possible and the end result represents my capabilities as a programmer and designer.



Appendix A – Technician Flow Diagram



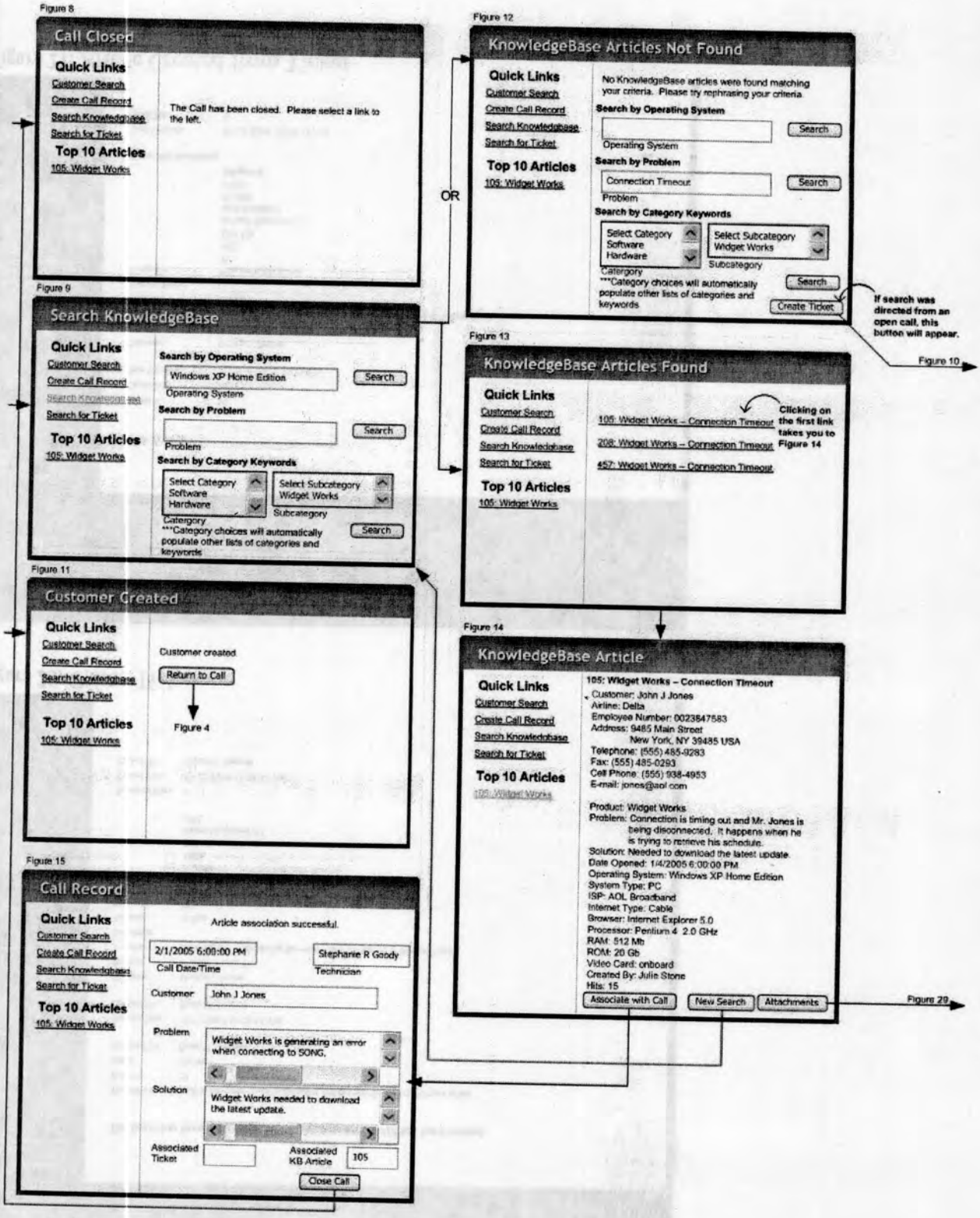


Figure 8
Call Closed
Quick Links
Customer Search
Create Call Record
Search Knowledgebase
Search for Ticket
Top 10 Articles
105: Widget Works
The Call has been closed. Please select a link to the left.
OR
Figure 12
KnowledgeBase Articles Not Found
No KnowledgeBase articles were found matching your criteria. Please try rephrasing your criteria.
Search by Operating System
Search
Operating System
Search by Problem
Connection Timeout
Search
Problem
Search by Category Keywords
Select Category
Software
Hardware
Category
Select Subcategory
Widget Works
Subcategory
***Category choices will automatically populate other lists of categories and keywords
Search
Create Ticket
If search was directed from an open call, this button will appear.
Figure 10
Figure 9
Search KnowledgeBase
Windows XP Home Edition
Search
Operating System
Search by Problem
Search
Problem
Search by Category Keywords
Select Category
Software
Hardware
Category
Select Subcategory
Widget Works
Subcategory
***Category choices will automatically populate other lists of categories and keywords
Search
Figure 13
KnowledgeBase Articles Found
105: Widget Works – Connection Timeout
208: Widget Works – Connection Timeout
457: Widget Works – Connection Timeout
Clicking on the first link takes you to Figure 14
Figure 11
Customer Created
Customer created
Return to Call
Figure 4
Figure 14
KnowledgeBase Article
105: Widget Works – Connection Timeout
Customer: John J Jones
Airline: Delta
Employee Number: 0023847583
Address: 9485 Main Street
New York, NY 39485 USA
Telephone: (555) 485-9283
Fax: (555) 485-0293
Cell Phone: (555) 938-4953
E-mail: jones@aol.com
Product: Widget Works
Problem: Connection is timing out and Mr. Jones is being disconnected. It happens when he is trying to retrieve his schedule.
Solution: Needed to download the latest update.
Date Opened: 1/4/2005 6:00:00 PM
Operating System: Windows XP Home Edition
System Type: PC
ISP: AOL Broadband
Internet Type: Cable
Browser: Internet Explorer 5.0
Processor: Pentium 4 2.0 GHz
RAM: 512 Mb
ROM: 20 Gb
Video Card: onboard
Created By: Julie Stone
Hits: 15
Associate with Call
New Search
Attachments
Figure 29
Figure 15
Call Record
Article association succesful.
2/1/2005 6:00:00 PM
Call Date/Time
Stephanie R Goody
Technician
Customer
John J Jones
Problem
Widget Works is generating an error when connecting to SONG.
Solution
Widget Works needed to download the latest update.
Associated Ticket
Associated KB Article
105
Close Call

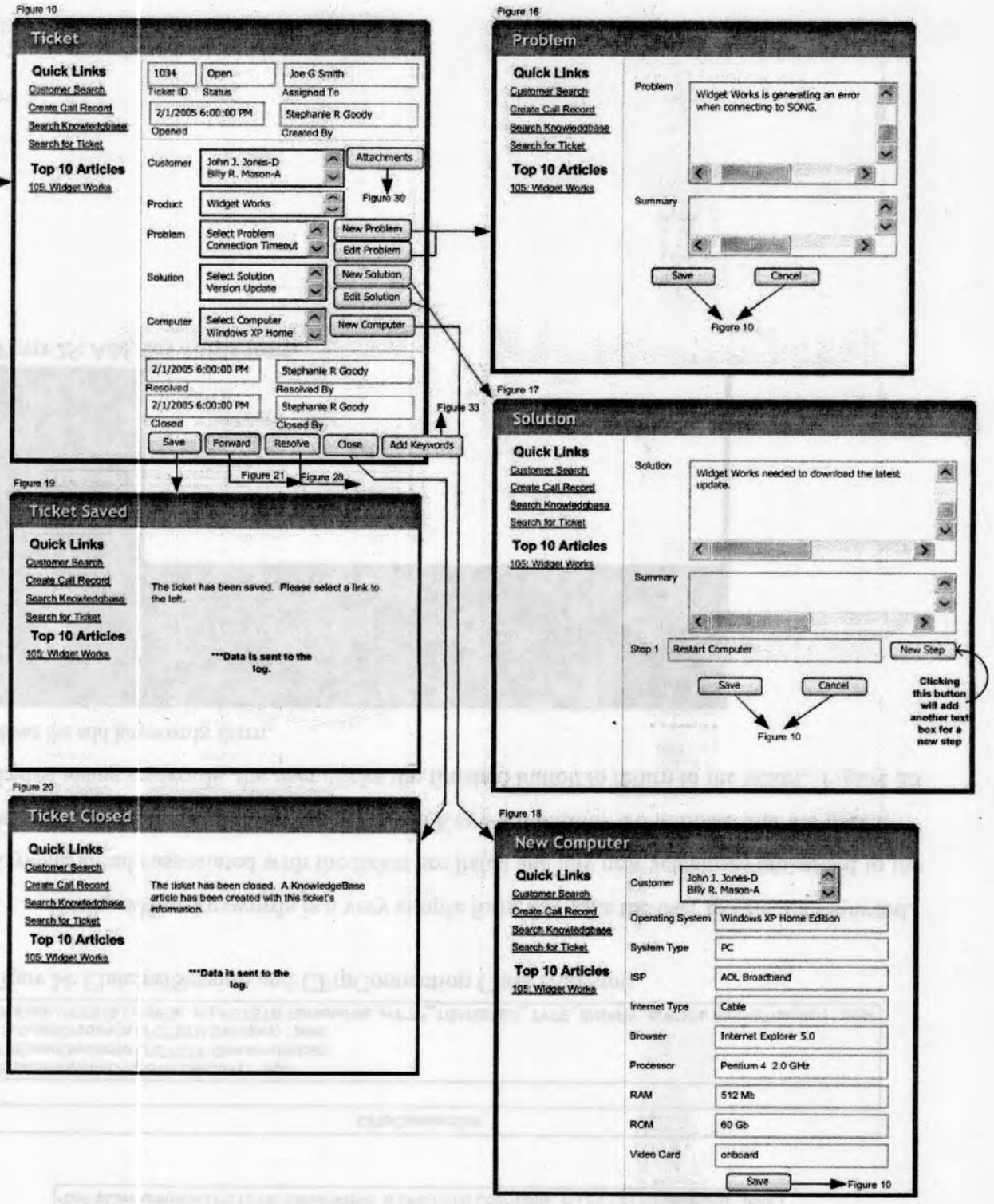


Figure 10
Ticket
Quick Links
Customer Search
Create Call Record
Search Knowledgebase
Search for Ticket
Top 10 Articles
105: Widget Works
1034
Ticket ID
Open
Status
Joe G Smith
Assigned To
2/1/2005 6:00:00 PM
Opened
Stephanie R Goody
Created By
Customer
John J. Jones-D
Billy R. Mason-A
Attachments
Figure 30
Product
Widget Works
Problem
Select Problem
Connection Timeout
New Problem
Edit Problem
Solution
Select Solution
Version Update
New Solution
Edit Solution
Computer
Select Computer
Windows XP Home
New Computer
2/1/2005 6:00:00 PM
Resolved
Stephanie R Goody
Resolved By
2/1/2005 6:00:00 PM
Closed
Stephanie R Goody
Closed By
Save
Forward
Resolve
Close
Add Keywords
Figure 33
Figure 21
Figure 28
Figure 16
Problem
Quick Links
Customer Search
Create Call Record
Search Knowledgebase
Search for Ticket
Top 10 Articles
105: Widget Works
Problem
Widget Works is generating an error when connecting to SONG.
Summary
Save
Cancel
Figure 10
Figure 17
Solution
Quick Links
Customer Search
Create Call Record
Search Knowledgebase
Search for Ticket
Top 10 Articles
105: Widget Works
Solution
Widget Works needed to download the latest update.
Summary
Step 1
Restart Computer
New Step
Save
Cancel
Figure 10
Clicking this button will add another text box for a new step
Figure 19
Ticket Saved
Quick Links
Customer Search
Create Call Record
Search Knowledgebase
Search for Ticket
Top 10 Articles
105: Widget Works
The ticket has been saved. Please select a link to the left.
***Data is sent to the log.
Figure 20
Ticket Closed
Quick Links
Customer Search
Create Call Record
Search Knowledgebase
Search for Ticket
Top 10 Articles
105: Widget Works
The ticket has been closed. A KnowledgeBase article has been created with this ticket's information.
***Data is sent to the log.
Figure 18
New Computer
Quick Links
Customer Search
Create Call Record
Search Knowledgebase
Search for Ticket
Top 10 Articles
105: Widget Works
Customer
John J. Jones-D
Billy R. Mason-A
Operating System
Windows XP Home Edition
System Type
PC
ISP
AOL Broadband
Internet Type
Cable
Browser
Internet Explorer 5.0
Processor
Pentium 4 2.0 GHz
RAM
512 Mb
ROM
60 Gb
Video Card
onboard
Save
Figure 10

Figure 21

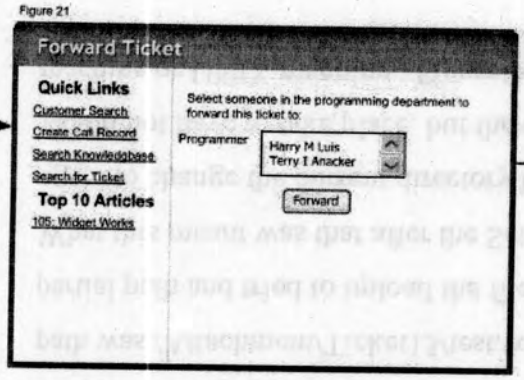


Figure 22

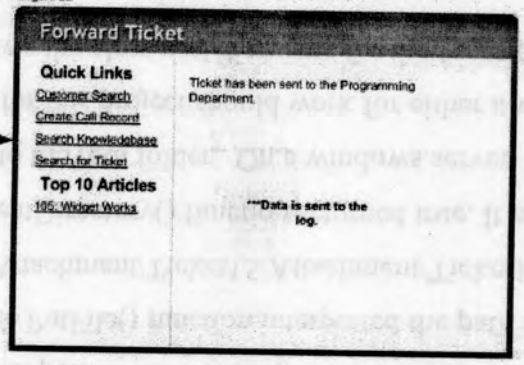


Figure 23

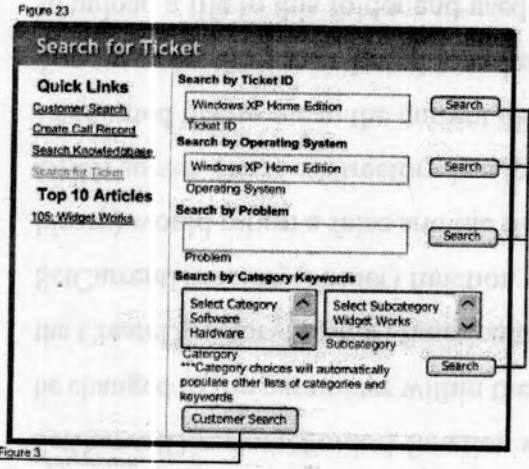


Figure 24

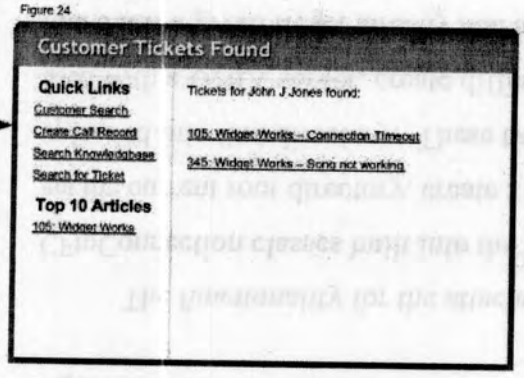
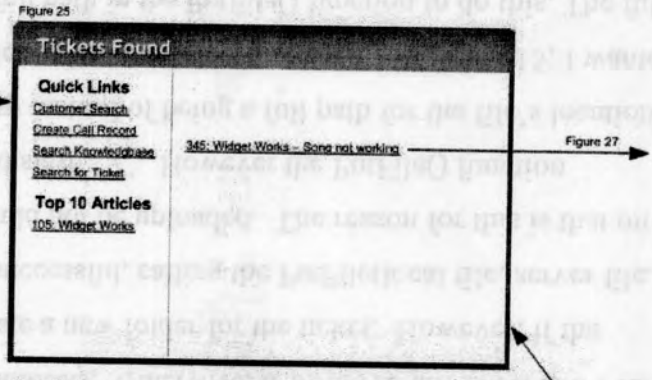
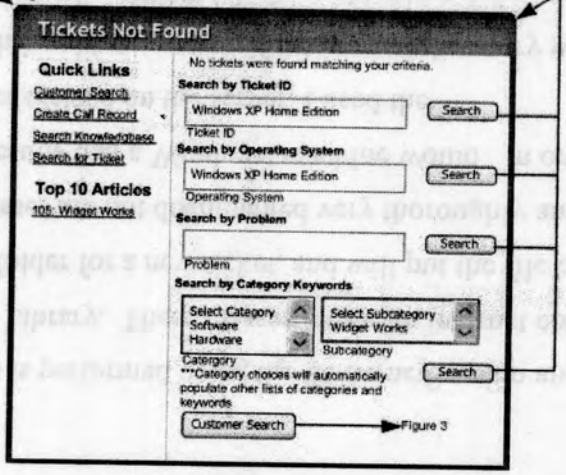


Figure 25



OR

Figure 26



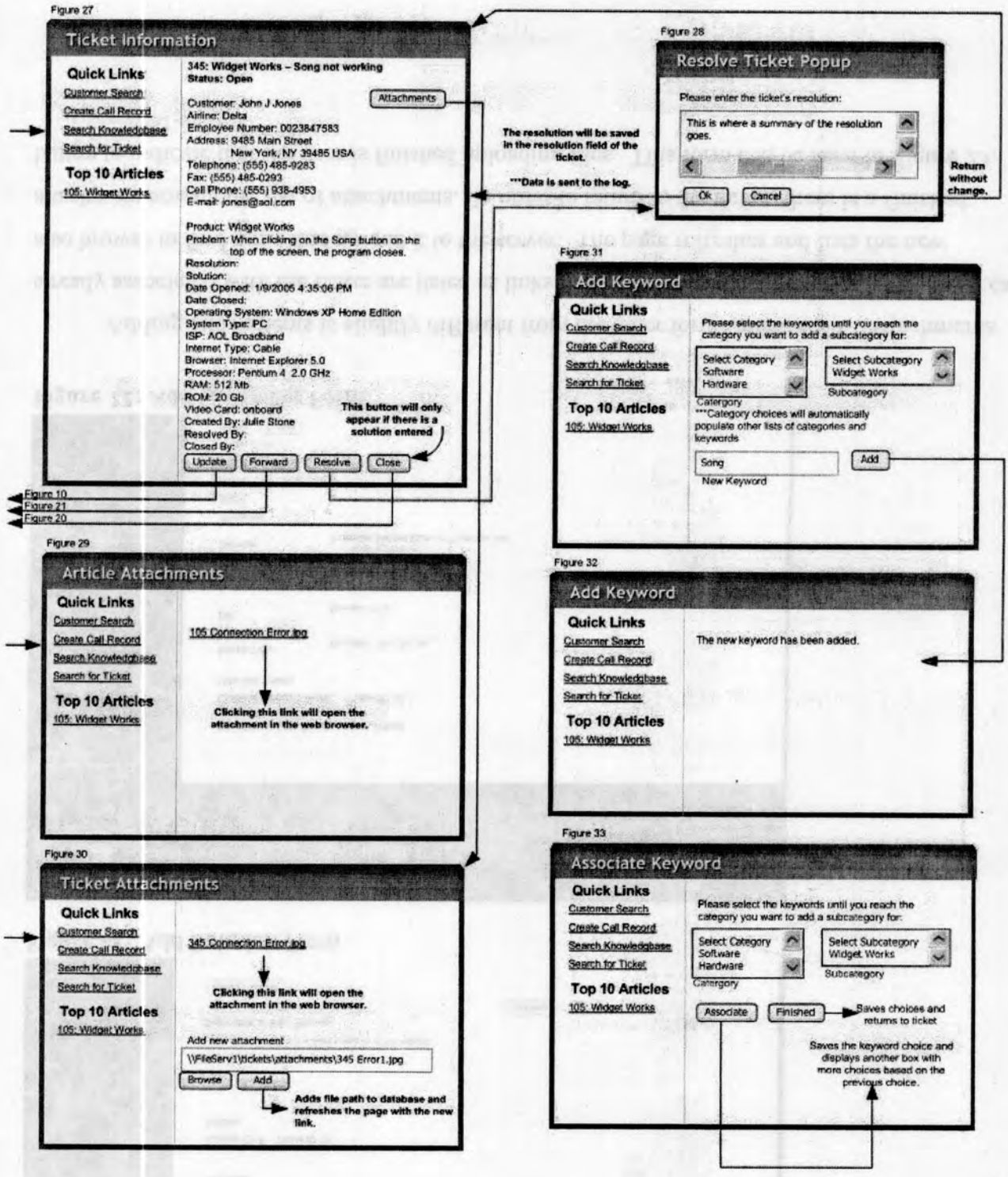
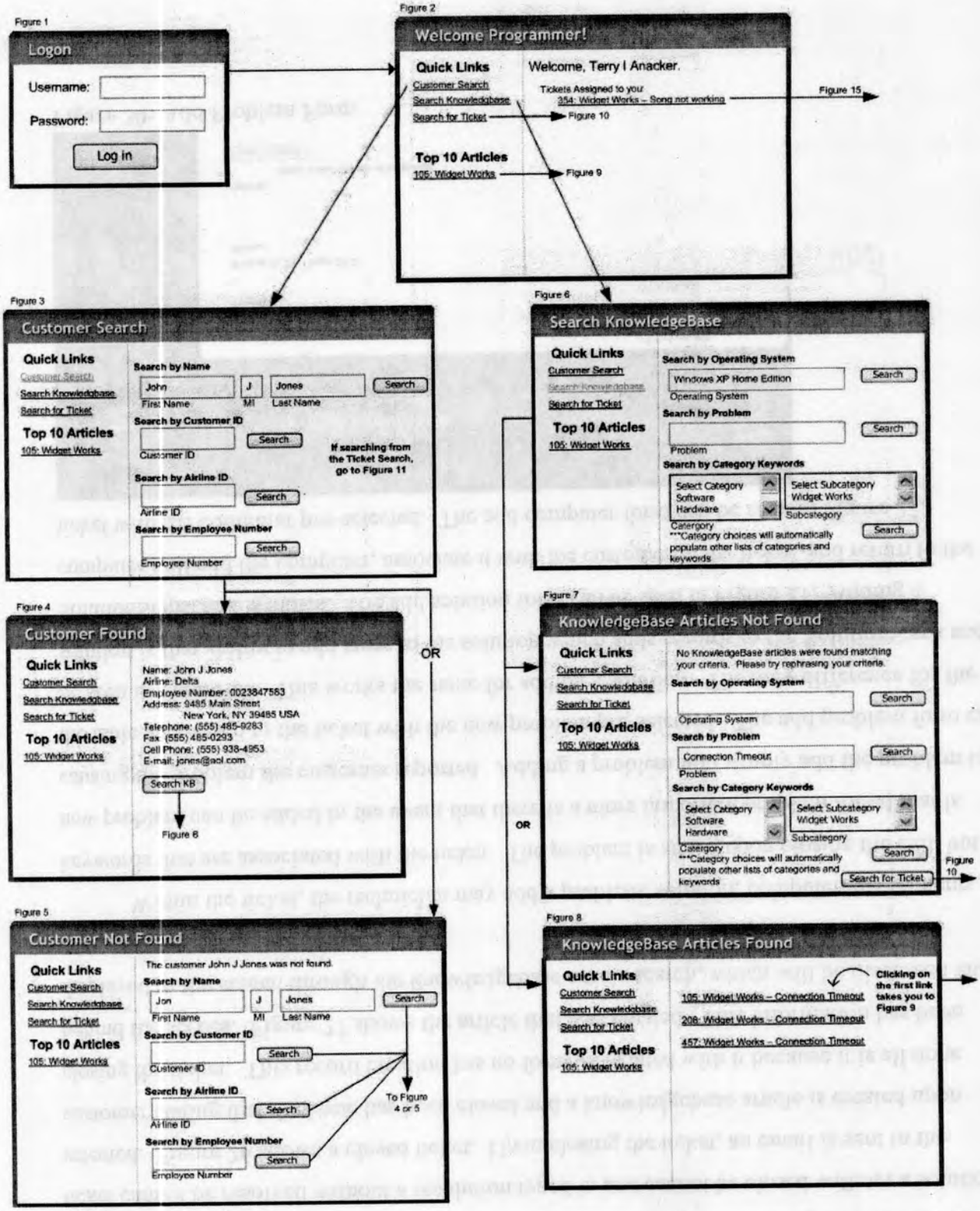


Figure 27
Ticket Information
Quick Links
Customer Search
Create Call Record
Search Knowledgebase
Search for Ticket
Top 10 Articles
105: Widget Works
345: Widget Works – Song not working
Status: Open
Attachments
Customer: John J Jones
Airline: Delta
Employee Number: 0023847583
Address: 9485 Main Street
New York, NY 39485 USA
Telephone: (555) 485-9283
Fax: (555) 485-0293
Cell Phone: (555) 938-4953
E-mail: jones@aol.com
Product: Widget Works
Problem: When clicking on the Song button on the top of the screen, the program closes.
Resolution:
Solution:
Date Opened: 1/9/2005 4:35:06 PM
Date Closed:
Operating System: Windows XP Home Edition
System Type: PC
ISP: AOL Broadband
Internet Type: Cable
Browser: Internet Explorer 5.0
Processor: Pentium 4 2.0 GHz
RAM: 512 Mb
ROM: 20 Gb
Video Card: onboard
Created By: Julie Stone
Resolved By:
Closed By:
This button will only appear if there is a solution entered
Update
Forward
Resolve
Close
Figure 10
Figure 21
Figure 20
Figure 28
Resolve Ticket Popup
Please enter the ticket's resolution:
This is where a summary of the resolution goes.
Ok
Cancel
The resolution will be saved in the resolution field of the ticket.
***Data is sent to the log.
Return without change.
Figure 31
Add Keyword
Quick Links
Customer Search
Create Call Record
Search Knowledgbase
Search for Ticket
Top 10 Articles
105: Widget Works
Please select the keywords until you reach the category you want to add a subcategory for:
Select Category
Software
Hardware
Catergory
Select Subcategory
Widget Works
Subcategory
***Category choices will automatically populate other lists of categories and keywords
Song
New Keyword
Add
Figure 29
Article Attachments
Quick Links
Customer Search
Create Call Record
Search Knowledgbase
Search for Ticket
Top 10 Articles
105: Widget Works
105 Connection Error.jpg
Clicking this link will open the attachment in the web browser.
Figure 32
Add Keyword
Quick Links
Customer Search
Create Call Record
Search Knowledgbase
Search for Ticket
Top 10 Articles
105: Widget Works
The new keyword has been added.
Figure 30
Ticket Attachments
Quick Links
Customer Search
Create Call Record
Search Knowledgbase
Search for Ticket
Top 10 Articles
105: Widget Works
345 Connection Error.jpg
Clicking this link will open the attachment in the web browser.
Add new attachment
\\FileServ1\tickets\attachments\345 Error1.jpg
Browse
Add
Adds file path to database and refreshes the page with the new link.
Figure 33
Associate Keyword
Quick Links
Customer Search
Create Call Record
Search Knowledgbase
Search for Ticket
Top 10 Articles
105: Widget Works
Please select the keywords until you reach the category you want to add a subcategory for:
Select Category
Software
Hardware
Catergory
Select Subcategory
Widget Works
Subcategory
Associate
Finished
Saves choices and returns to ticket
Saves the keyword choice and displays another box with more choices based on the previous choice.

Appendix B – Programmer Flow Diagram



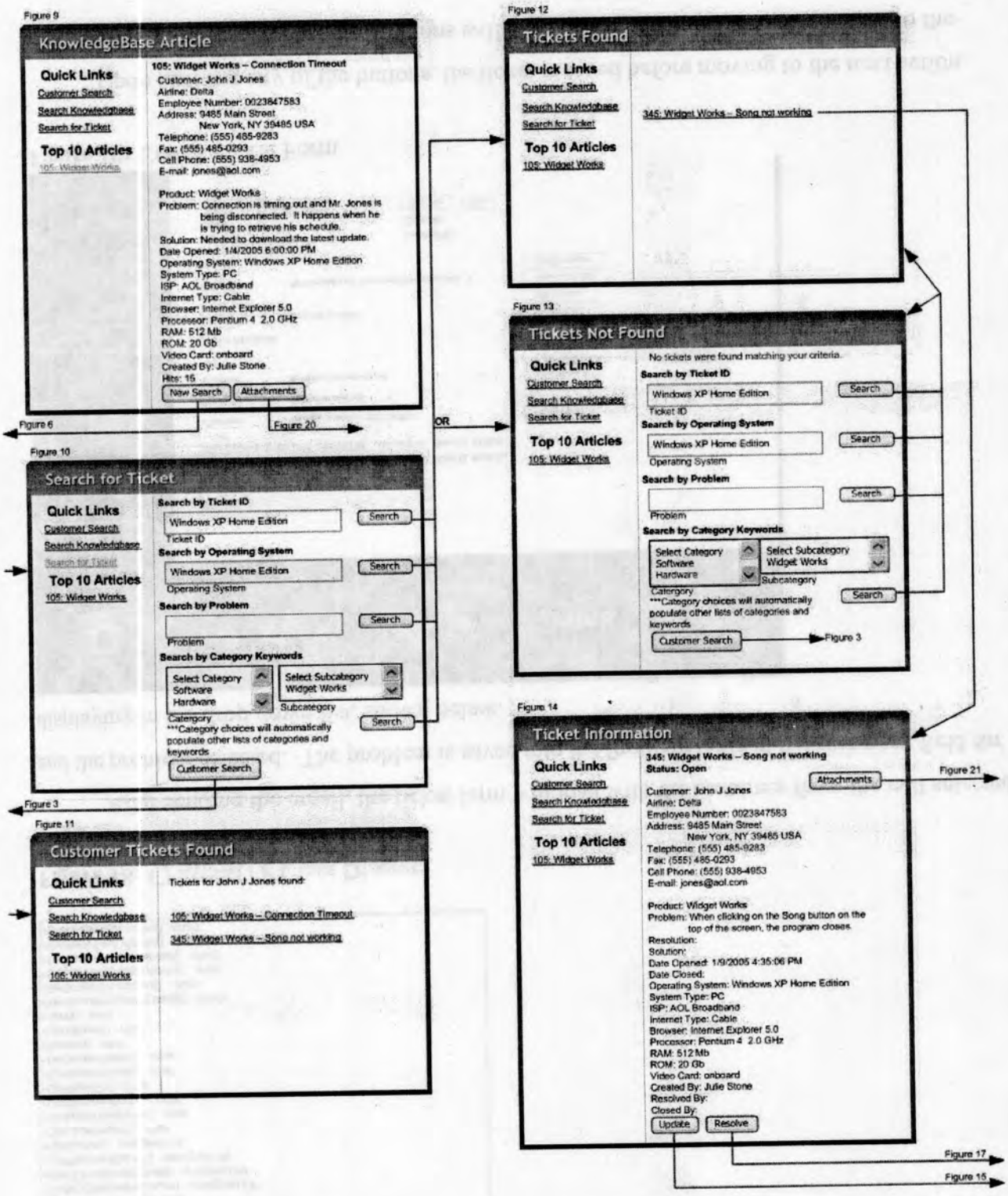


Figure 9
KnowledgeBase Article
Quick Links
Customer Search
Search Knowledgebase
Search for Ticket
Top 10 Articles
105: Widget Works
105: Widget Works – Connection Timeout
Customer: John J Jones
Airline: Delta
Employee Number: 0023847583
Address: 9485 Main Street
New York, NY 39485 USA
Telephone: (555) 485-9283
Fax: (555) 485-0293
Cell Phone: (555) 938-4953
E-mail: jones@aol.com
Product: Widget Works
Problem: Connection is timing out and Mr. Jones is being disconnected. It happens when he is trying to retrieve his schedule.
Solution: Needed to download the latest update.
Date Opened: 1/4/2005 6:00:00 PM
Operating System: Windows XP Home Edition
System Type: PC
ISP: AOL Broadband
Internet Type: Cable
Browser: Internet Explorer 5.0
Processor: Pentium 4 2.0 GHz
RAM: 512 Mb
ROM: 20 Gb
Video Card: onboard
Created By: Julie Stone
Hits: 15
New Search
Attachments
Figure 6
Figure 20
Figure 12
Tickets Found
345: Widget Works – Song not working
OR
Figure 13
Tickets Not Found
No tickets were found matching your criteria.
Search by Ticket ID
Windows XP Home Edition
Ticket ID
Search
Search by Operating System
Operating System
Search by Problem
Problem
Search by Category Keywords
Select Category
Software
Hardware
Category
Select Subcategory
Widget Works
Subcategory
***Category choices will automatically populate other lists of categories and keywords
Customer Search
Figure 3
Figure 10
Search for Ticket
Figure 3
Figure 11
Customer Tickets Found
Tickets for John J Jones found:
105: Widget Works – Connection Timeout
345: Widget Works – Song not working
Figure 14
Ticket Information
345: Widget Works – Song not working
Status: Open
Attachments
Figure 21
Resolution:
Solution:
Date Opened: 1/9/2005 4:35:06 PM
Date Closed:
Resolved By:
Closed By:
Update
Resolve
Figure 17
Figure 15

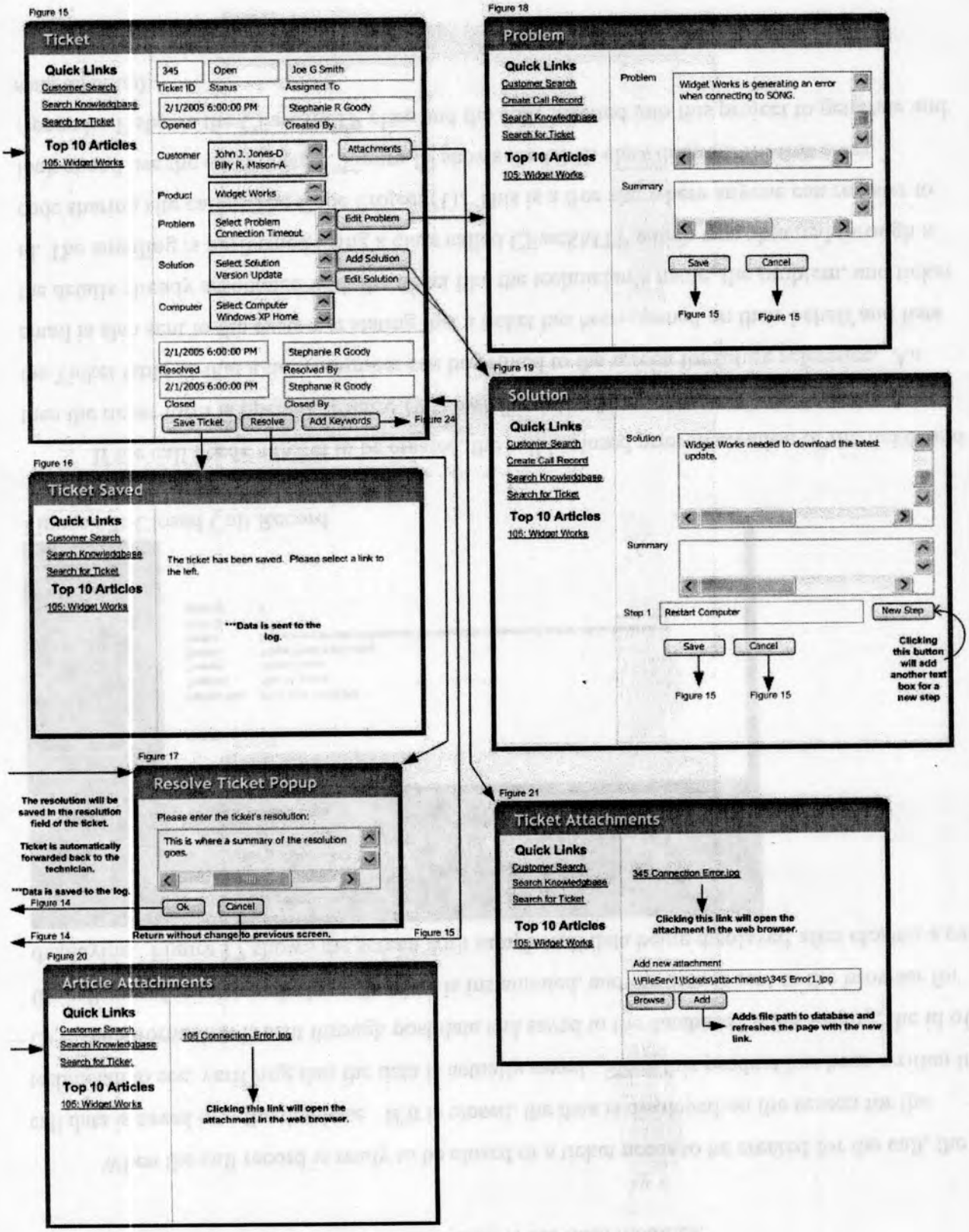


Figure 15
Ticket
Quick Links
Customer Search
Search Knowledgbase
Search for Ticket
Top 10 Articles
105: Widget Works
345
Ticket ID
Open
Status
Joe G Smith
Assigned To
2/1/2005 6:00:00 PM
Opened
Stephanie R Goody
Created By
Customer
John J. Jones-D
Billy R. Mason-A
Attachments
Product
Widget Works
Problem
Select Problem
Connection Timeout
Edit Problem
Solution
Select Solution
Version Update
Add Solution
Edit Solution
Computer
Select Computer
Windows XP Home
2/1/2005 6:00:00 PM
Resolved
Stephanie R Goody
Resolved By
2/1/2005 6:00:00 PM
Closed
Stephanie R Goody
Closed By
Save Ticket
Resolve
Add Keywords
Figure 24
Figure 18
Problem
Quick Links
Customer Search
Create Call Record
Search Knowledgbase
Search for Ticket
Top 10 Articles
105: Widget Works
Problem
Widget Works is generating an error when connecting to SONG.
Summary
Save
Cancel
Figure 15
Figure 15
Figure 19
Solution
Quick Links
Customer Search
Create Call Record
Search Knowledgbase
Search for Ticket
Top 10 Articles
105: Widget Works
Solution
Widget Works needed to download the latest update.
Summary
Step 1
Restart Computer
New Step
Save
Cancel
Figure 15
Figure 15
Clicking this button will add another text box for a new step
Figure 16
Ticket Saved
Quick Links
Customer Search
Search Knowledgbase
Search for Ticket
Top 10 Articles
105: Widget Works
The ticket has been saved. Please select a link to the left.
***Data is sent to the log.
Figure 17
Resolve Ticket Popup
Please enter the ticket's resolution:
This is where a summary of the resolution goes.
Ok
Cancel
The resolution will be saved in the resolution field of the ticket.
Ticket is automatically forwarded back to the technician.
***Data is saved to the log.
Figure 14
Figure 14
Return without change to previous screen.
Figure 15
Figure 21
Ticket Attachments
Quick Links
Customer Search
Search Knowledgbase
Search for Ticket
Top 10 Articles
105: Widget Works
345 Connection Error.jpg
Clicking this link will open the attachment in the web browser.
Add new attachment
\\FileServ1\tickets\attachments\345 Error1.jpg
Browse
Add
Adds file path to database and refreshes the page with the new link.
Figure 20
Article Attachments
Quick Links
Customer Search
Search Knowledgbase
Search for Ticket
Top 10 Articles
105: Widget Works
105 Connection Error.jpg
Clicking this link will open the attachment in the web browser.

Figure 22

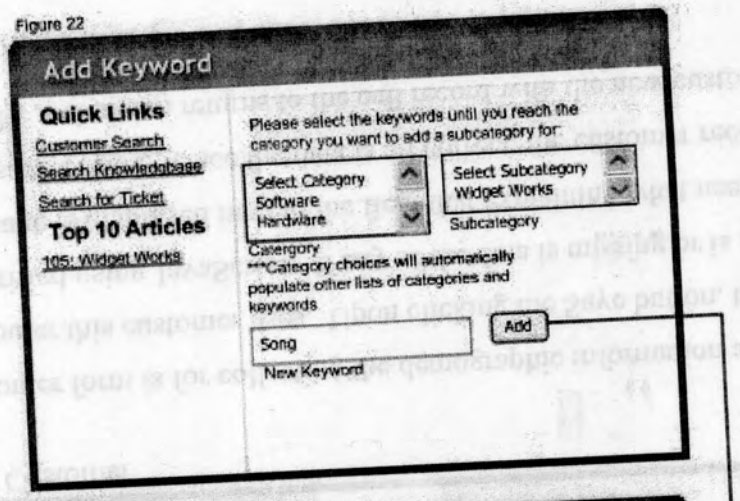


Figure 23

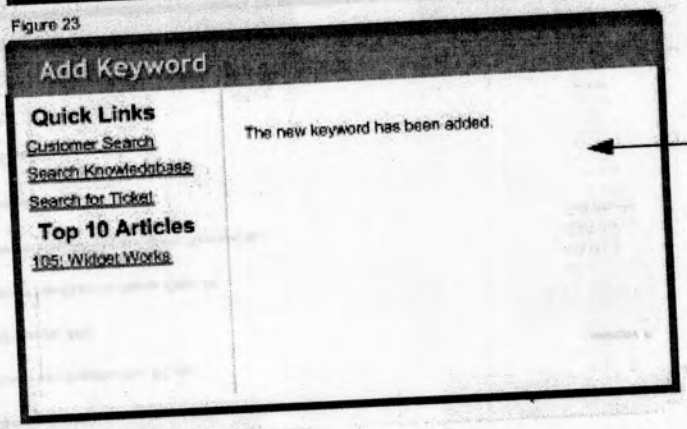
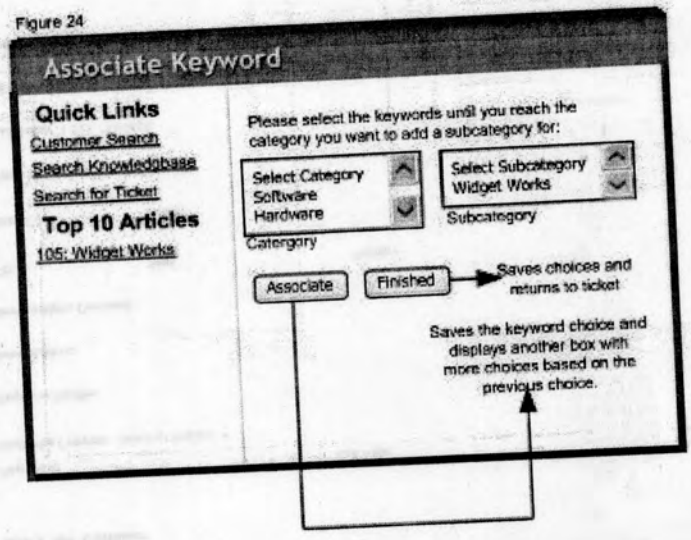
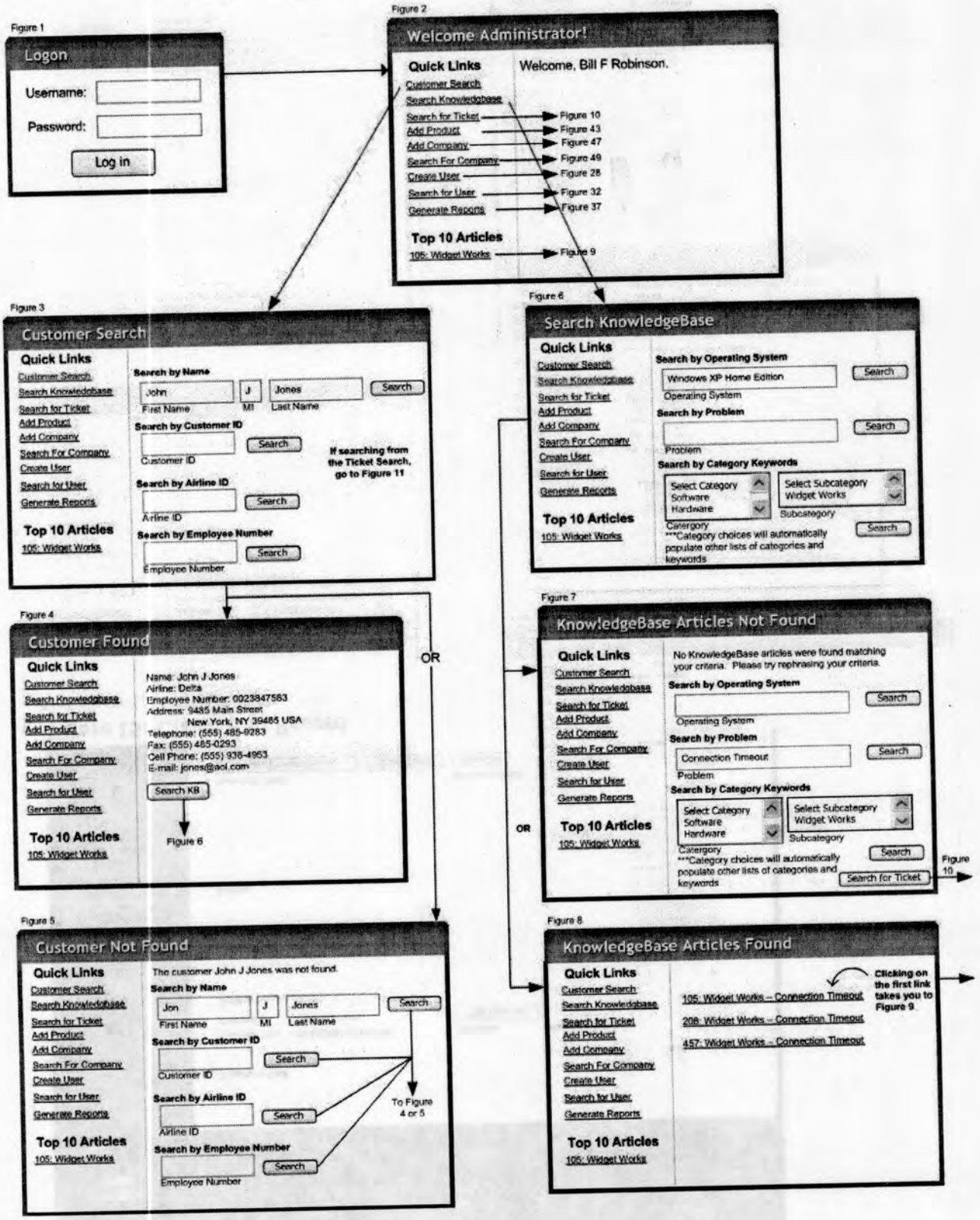


Figure 24



Appendix C – Administrator Flow Diagram



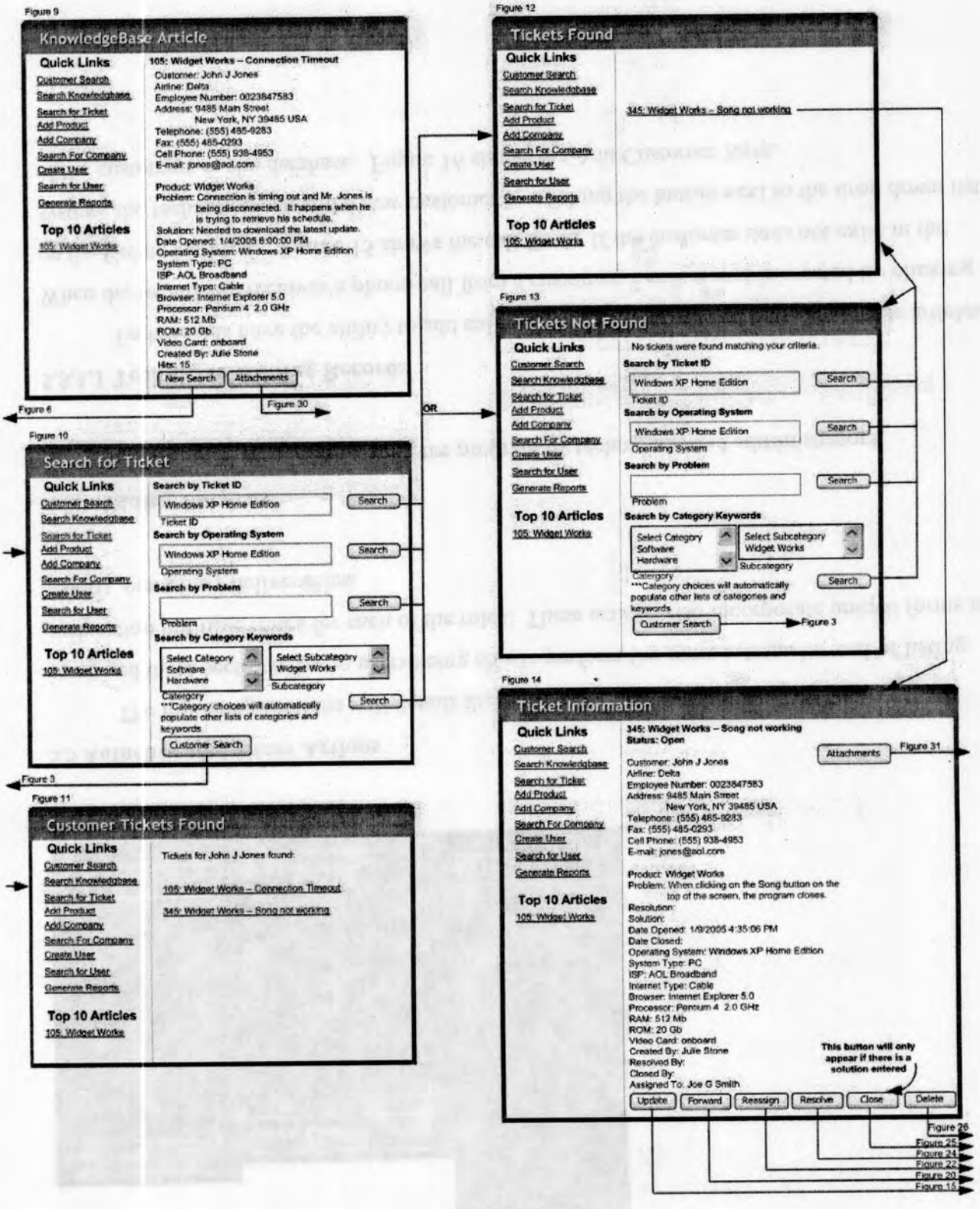


Figure 9
KnowledgeBase Article
Quick Links
Customer Search
Search Knowledgbase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
105: Widget Works – Connection Timeout
Customer: John J Jones
Airline: Delta
Employee Number: 0023847583
Address: 9485 Main Street
New York, NY 39485 USA
Telephone: (555) 485-9283
Fax: (555) 485-0293
Cell Phone: (555) 938-4953
E-mail: jones@aol.com
Product: Widget Works
Problem: Connection is timing out and Mr. Jones is being disconnected. It happens when he is trying to retrieve his schedule.
Solution: Needed to download the latest update.
Date Opened: 1/4/2005 6:00:00 PM
Operating System: Windows XP Home Edition
System Type: PC
ISP: AOL Broadband
Internet Type: Cable
Browser: Internet Explorer 5.0
Processor: Pentium 4 2.0 GHz
RAM: 512 Mb
ROM: 20 Gb
Video Card: onboard
Created By: Julie Stone
Hits: 15
New Search
Attachments
Figure 6
Figure 30
Figure 12
Tickets Found
Quick Links
Customer Search
Search Knowledgbase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
345: Widget Works – Song not working
OR
Figure 13
Tickets Not Found
Quick Links
Customer Search
Search Knowledgbase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
No tickets were found matching your criteria.
Search by Ticket ID
Windows XP Home Edition
Ticket ID
Search
Search by Operating System
Windows XP Home Edition
Operating System
Search
Search by Problem
Problem
Search
Search by Category Keywords
Select Category
Software
Hardware
Category
Select Subcategory
Widget Works
Subcategory
Search
***Category choices will automatically populate other lists of categories and keywords
Customer Search
Figure 3
Figure 10
Search for Ticket
Quick Links
Customer Search
Search Knowledgbase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
Search by Ticket ID
Windows XP Home Edition
Ticket ID
Search
Search by Operating System
Windows XP Home Edition
Operating System
Search
Search by Problem
Problem
Search
Search by Category Keywords
Select Category
Software
Hardware
Category
Select Subcategory
Widget Works
Subcategory
Search
***Category choices will automatically populate other lists of categories and keywords
Customer Search
Figure 3
Figure 11
Customer Tickets Found
Quick Links
Customer Search
Search Knowledgbase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
Tickets for John J Jones found:
105: Widget Works – Connection Timeout
345: Widget Works – Song not working
Figure 14
Ticket Information
Quick Links
Customer Search
Search Knowledgbase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
345: Widget Works – Song not working
Status: Open
Attachments
Figure 31
Customer: John J Jones
Airline: Delta
Employee Number: 0023847583
Address: 9485 Main Street
New York, NY 39485 USA
Telephone: (555) 485-9283
Fax: (555) 485-0293
Cell Phone: (555) 938-4953
E-mail: jones@aol.com
Product: Widget Works
Problem: When clicking on the Song button on the top of the screen, the program closes.
Resolution:
Solution:
Date Opened: 1/9/2005 4:35:06 PM
Date Closed:
Operating System: Windows XP Home Edition
System Type: PC
ISP: AOL Broadband
Internet Type: Cable
Browser: Internet Explorer 5.0
Processor: Pentium 4 2.0 GHz
RAM: 512 Mb
ROM: 20 Gb
Video Card: onboard
Created By: Julie Stone
Resolved By:
Closed By:
Assigned To: Joe G Smith
This button will only appear if there is a solution entered
Update
Forward
Reassign
Resolve
Close
Delete
Figure 26
Figure 25
Figure 24
Figure 22
Figure 20
Figure 15

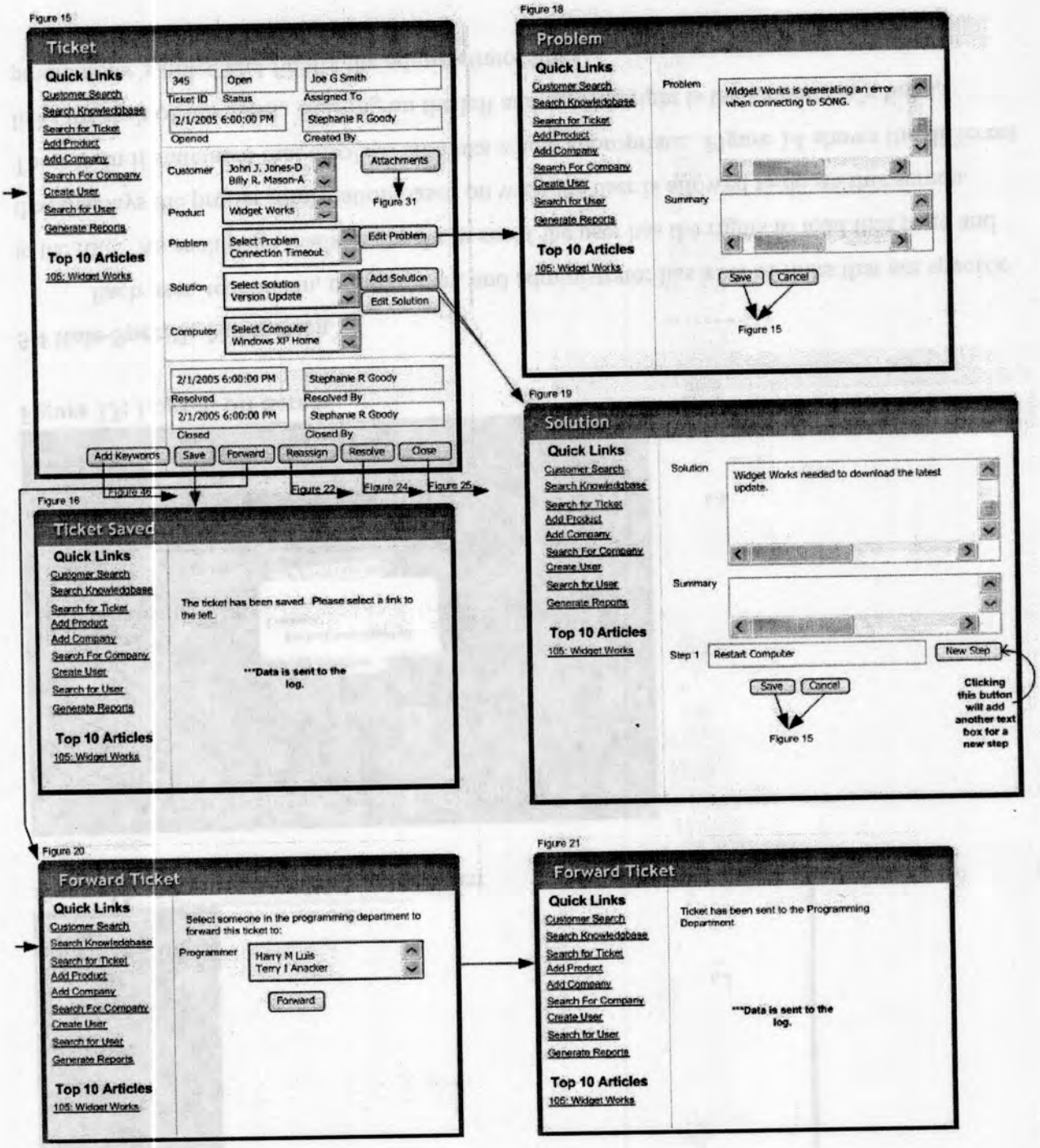


Figure 15
Ticket
Quick Links
Customer Search
Search Knowledgebase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
345
Ticket ID
Open
Status
Joe G Smith
Assigned To
2/1/2005 6:00:00 PM
Opened
Stephanie R Goody
Created By
Customer
John J. Jones-D
Billy R. Mason-A
Attachments
Figure 31
Product
Widget Works
Problem
Select Problem
Connection Timeout
Edit Problem
Solution
Select Solution
Version Update
Add Solution
Edit Solution
Computer
Select Computer
Windows XP Home
2/1/2005 6:00:00 PM
Resolved
Stephanie R Goody
Resolved By
2/1/2005 6:00:00 PM
Closed
Stephanie R Goody
Closed By
Add Keywords
Save
Forward
Reassign
Resolve
Close
Figure 18
Figure 16
Figure 22
Figure 24
Figure 25
Figure 18
Problem
Quick Links
Customer Search
Search Knowledgebase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
Problem
Widget Works is generating an error when connecting to SONG.
Summary
Save
Cancel
Figure 15
Figure 19
Solution
Quick Links
Customer Search
Search Knowledgebase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
Solution
Widget Works needed to download the latest update.
Summary
Step 1
Restart Computer
New Step
Save
Cancel
Figure 15
Clicking this button will add another text box for a new step
Figure 16
Ticket Saved
Quick Links
Customer Search
Search Knowledgebase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
The ticket has been saved. Please select a link to the left.
***Data is sent to the log.
Figure 20
Forward Ticket
Quick Links
Customer Search
Search Knowledgebase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
Select someone in the programming department to forward this ticket to:
Programmer
Harry M Luis
Terry I Anacker
Forward
Figure 21
Forward Ticket
Quick Links
Customer Search
Search Knowledgebase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
Ticket has been sent to the Programming Department
***Data is sent to the log.

Figure 22

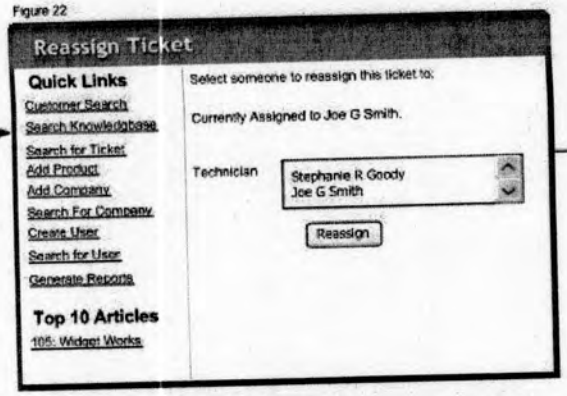


Figure 23

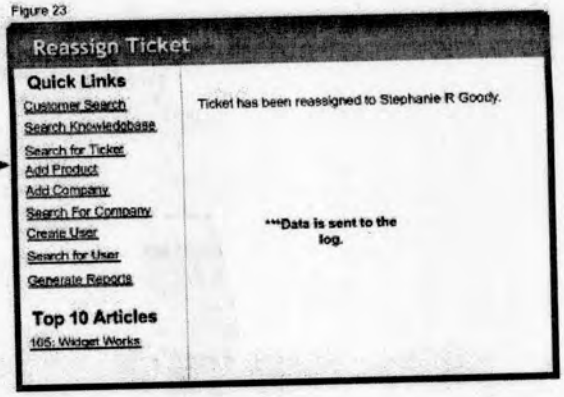


Figure 24

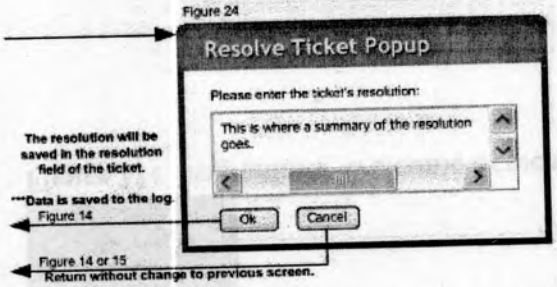


Figure 25

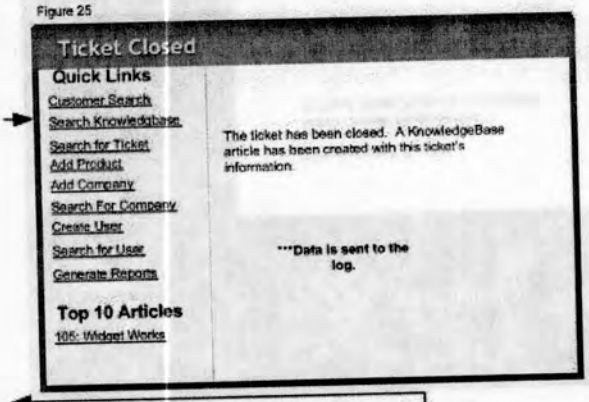


Figure 26

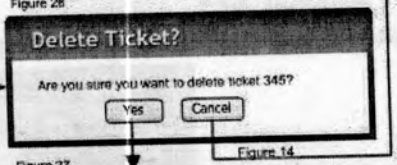


Figure 27

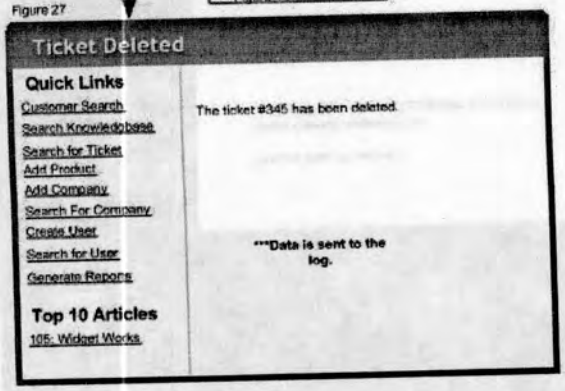


Figure 28

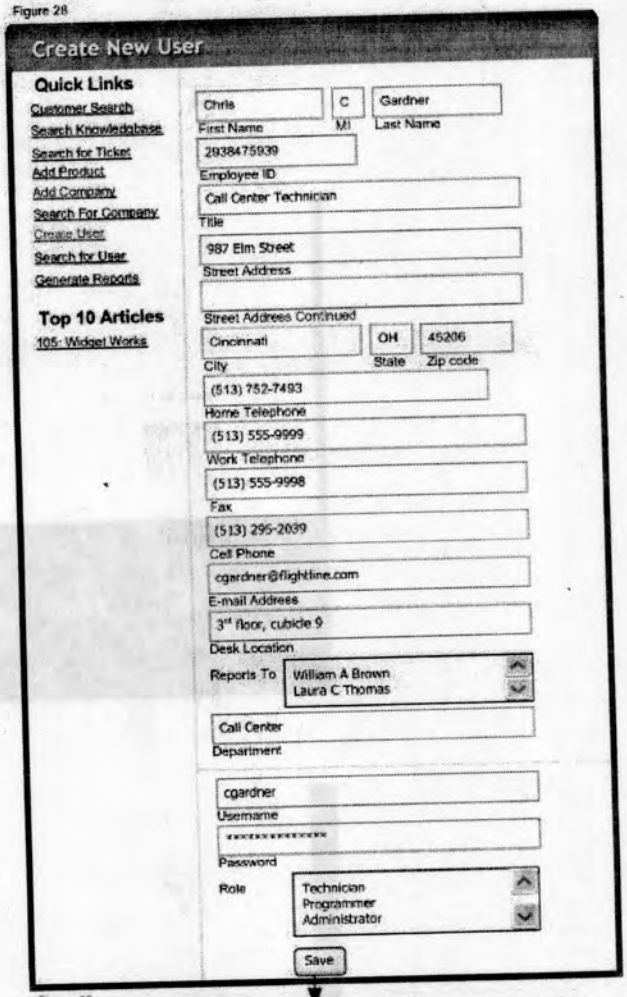
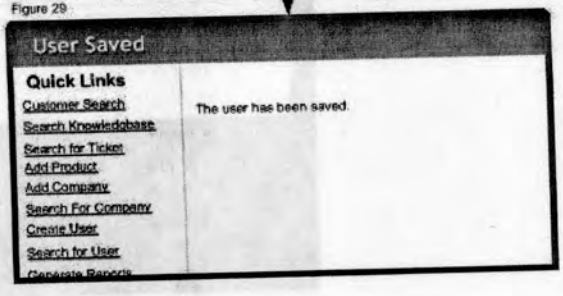


Figure 29



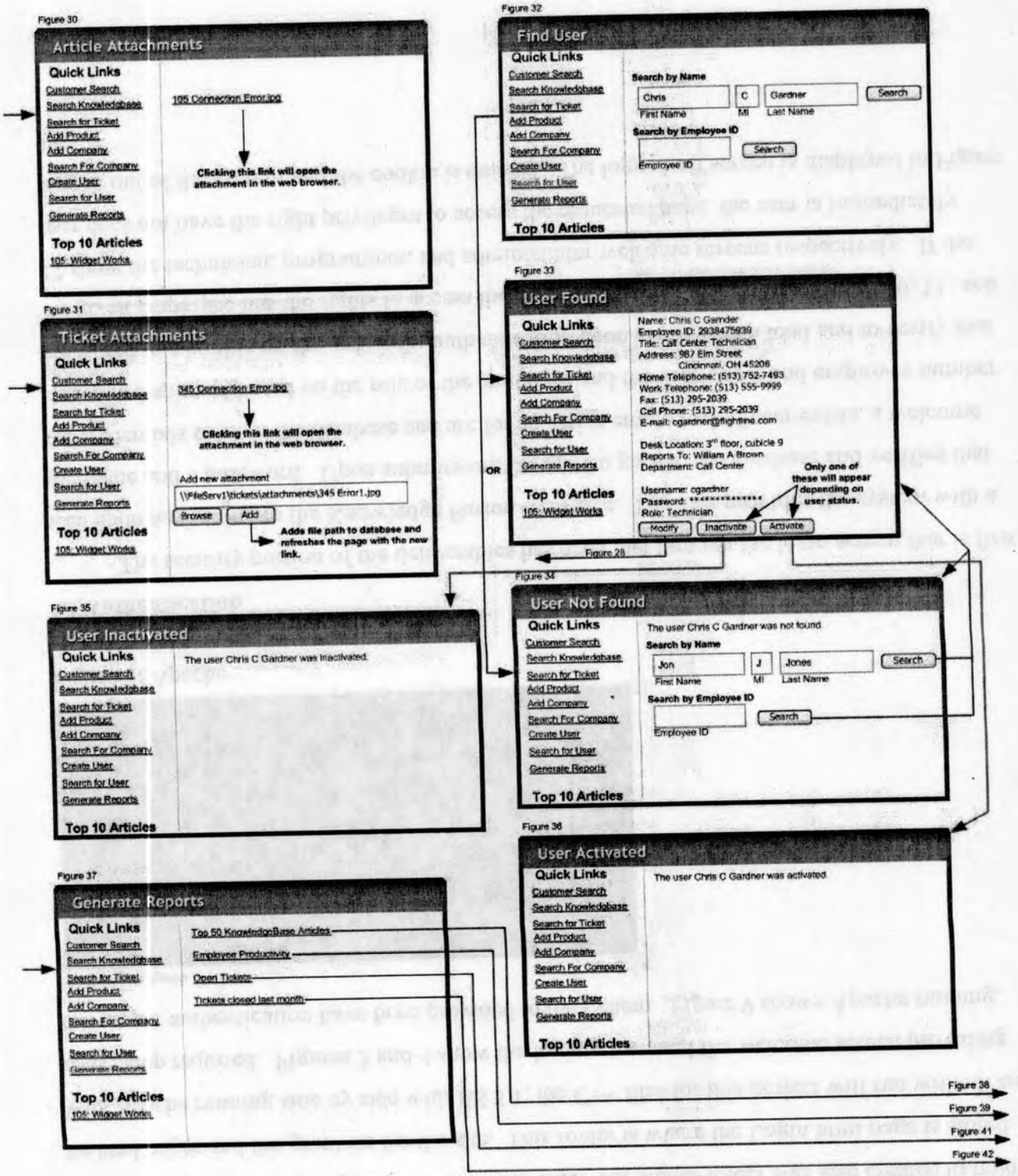


Figure 30
Article Attachments
Quick Links
Customer Search
Search Knowledgebase
Search for Ticket
Add Product
Add Company
Search For Company
Create User
Search for User
Generate Reports
Top 10 Articles
105: Widget Works
105 Connection Error.jpg
Clicking this link will open the attachment in the web browser.
Figure 31
Ticket Attachments
345 Connection Error.jpg
Clicking this link will open the attachment in the web browser.
Add new attachment
\\FileServ1\tickets\attachments\345 Error1.jpg
Browse
Add
Adds file path to database and refreshes the page with the new link.
Figure 32
Find User
Search by Name
Chris
C
Gardner
Search
First Name
MI
Last Name
Search by Employee ID
Employee ID
Figure 33
User Found
Name: Chris C Gamder
Employee ID: 2938475939
Title: Call Center Technician
Address: 987 Elm Street
Cincinnati, OH 45206
Home Telephone: (513) 752-7493
Work Telephone: (513) 555-9999
Fax: (513) 295-2039
Cell Phone: (513) 295-2039
E-mail: cgardner@flightline.com
Desk Location: 3rd floor, cubicle 9
Reports To: William A Brown
Department: Call Center
Username: cgardner
Password: **********
Role: Technician
Only one of these will appear depending on user status.
Modify
Inactivate
Activate
OR
Figure 28
Figure 34
User Not Found
The user Chris C Gardner was not found.
Search by Name
Jon
J
Jones
Search
First Name
MI
Last Name
Search by Employee ID
Employee ID
Figure 35
User Inactivated
The user Chris C Gardner was inactivated.
Figure 36
User Activated
The user Chris C Gardner was activated.
Figure 37
Generate Reports
Top 50 KnowledgeBase Articles
Employee Productivity
Open Tickets
Tickets closed last month
Figure 38
Figure 39
Figure 41
Figure 42

Figure 38

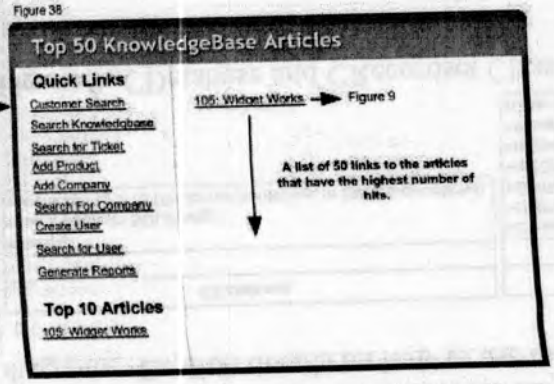


Figure 39

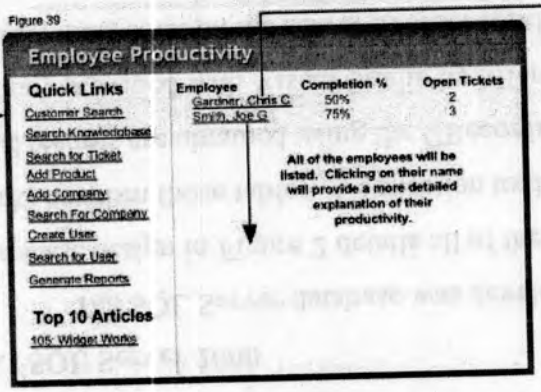


Figure 40

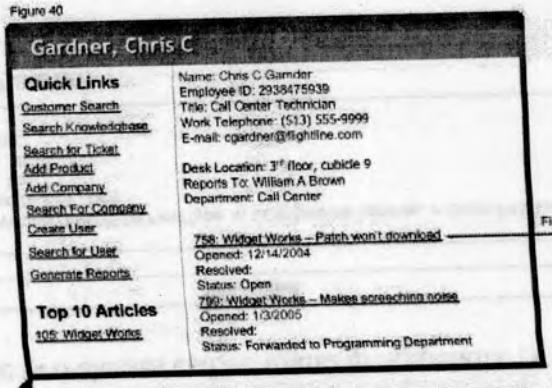


Figure 41

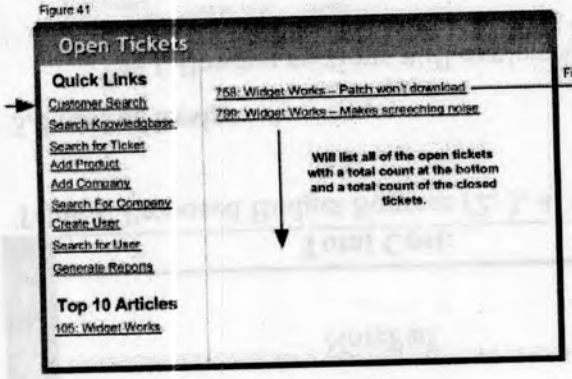


Figure 42

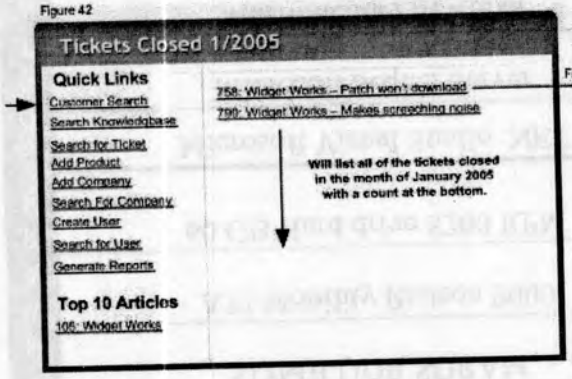


Figure 43

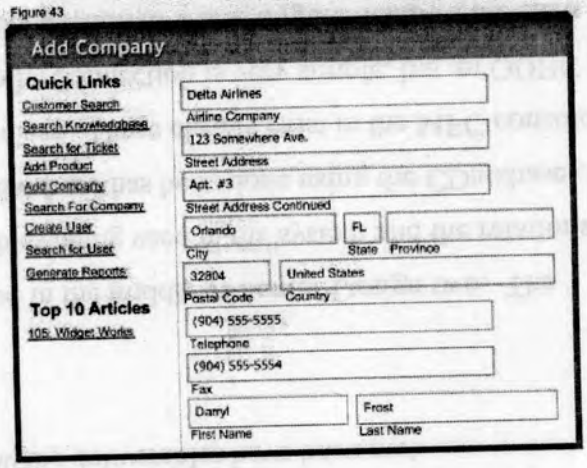


Figure 44

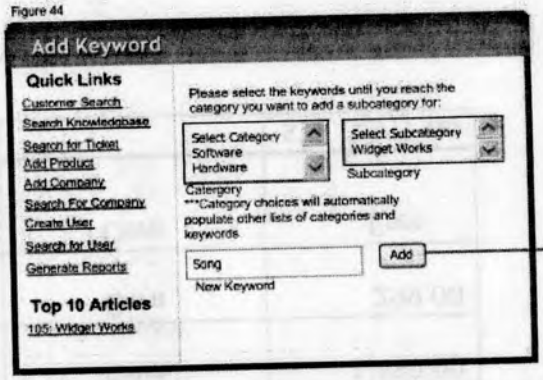


Figure 45

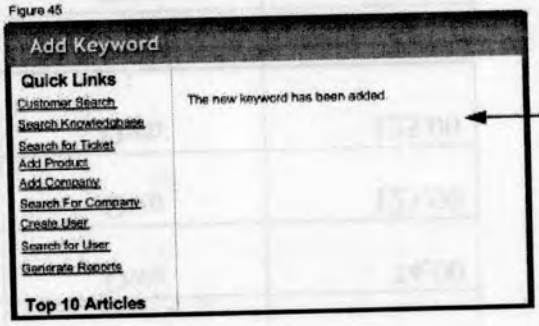


Figure 46

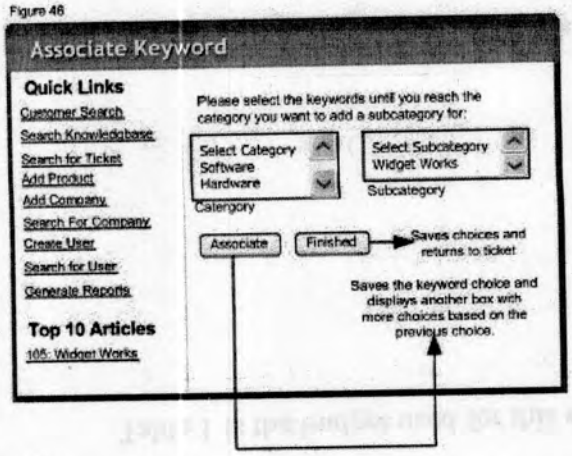


Figure 47

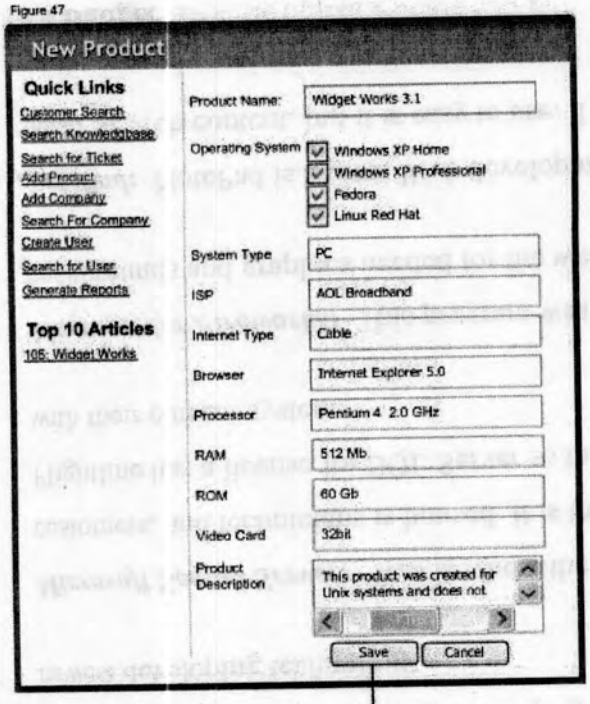


Figure 48

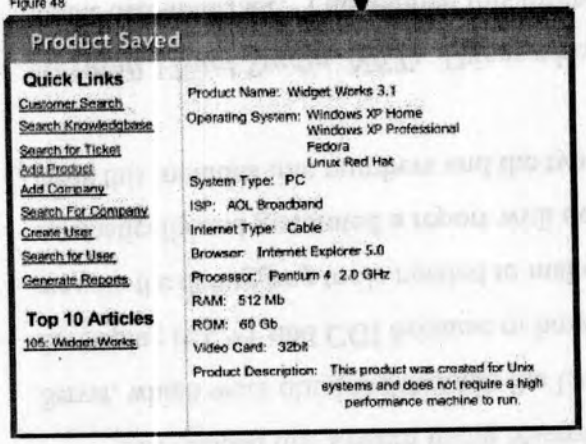


Figure 49

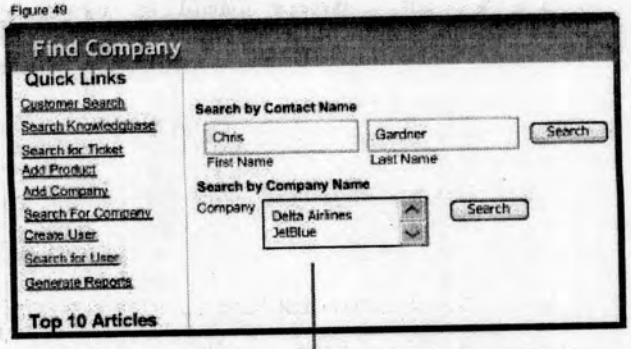
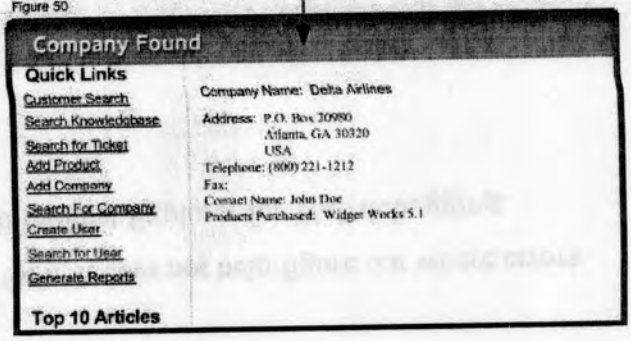


Figure 50



Appendix D – Connecting to MS SQL Server 2000 using C++

In order to use the CDatabase class, there must be an ODBC object created on the server for the connection. C++ does not have a way to connect to MS SQL Server using a DSNless connection, so the ODBC object must be created. The creation of such an object can be done by going to the computer's ODBC Manager located in the computer's administrative tools. Then you click the add button. This will give some options as to what type of object you are creating, choose SQL Server. SQL Server must be installed on the machine in order to do this. Following the prompts, give the object a name and tell it what server to connect to. For added security, change the log in to SQL server authentication and specify the username and password. Then locate the database, and continue following the prompts changing information as necessary. Once this has been done, you are ready to start using the CDatabase class.

The CDatabase class requires the afxdb.h to be included in the project. Below is some sample code for a very simple database connection.

```
#include <afxdb.h>
#include <string>
Int Main()
{
  CDatabase db;
  try
  {
    db.OpenEx("DSN=ODBCName;Server=ServerName;Database=DBName;Uid=Username;Pwd=password;",0);
  }
  catch(exception &error)
  {
    cerr << "Error: " << error.what();
  };
  string strSQLStatement = "INSERT INTO Products (Product, ProductType) VALUES (\"Dove\", \"Soap\")";
  LPCTSTR strSQL = (LPCTSTR) strSQLStatement.c_str();
  //perform the update
  try
  {
    db.ExecuteSQL(strSQL);
  }
  catch(exception &error)
  {
    cerr << "Error: " << error.what();
  };
  db.Close();
  return 0;
}
```

The CRecordset class is used in conjunction with the CDatabase class for select statements. From the code listed above, a recordset can be created and the data from the database can be displayed to the screen or used in other calculations/functions. In order to get the data from the recordset, there must be a CDBVariant object instantiated to hold the data. All of the information the recordset will have a type of variant and must be stored in a variant object. The variant object in turn must be told what form the data should take. This is done in the GetFieldValue() member of the CRecordset class. This function takes three parameters, the column name in the select statement, the variant object to store the data, and finally the data type of the data in the form of a DWord type. The DWord type is a built-in structure in the MFC library that translates from SQL data type to C++ data type. The structure can be seen below.

C data type	SQL data type
SQL_C_BIT	SQL_BIT
SQL_C_UTINYINT	SQL_TINYINT
SQL_C_SSHORT	SQL_SMALLINT
SQL_C_SLONG	SQL_INTEGER
SQL_C_FLOAT	SQL_REAL
SQL_C_DOUBLE	SQL_FLOAT SQL_DOUBLE
SQL_C_TIMESTAMP	SQL_DATE SQL_TIME SQL_TIMESTAMP
SQL_C_CHAR	SQL_NUMERIC SQL_DECIMAL SQL_BIGINT SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR
SQL_C_BINARY	SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY

Table 2: DWord Options for CRecordset Class

On the following page is a short example of how to create a database connection and select records from a table in the database. The results of the select statement will be displayed to the screen. The example shows the use of IsOpen() to test if the recordset object was opened for data, GetRecordCount() to make sure the recordset contains records, IsEOF() to see when we have cycled through all the records, and MoveNext() to move through the records.

```
#include <afxdb.h>
#include <string>
Int Main()
{
   CDBVariant varProductID, varProduct;
   CDatabase db;

   //Open the database
   try
   {
      db.OpenEx("DSN=ODBCName;Server=ServerName;Database=DBName;Uid=username;Pwd=password;",0);
   }
   catch(exception &error)
   {
      cerr << "Error: " << error.what();
   };
   //create the recordset
   CRecordset rsProducts(&db);
   //Create the SQL string
   string strTemp = "SELECT ProductID, ProductName FROM Product";
   LPCTSTR strSQL = (LPCTSTR)strTemp.c_str();
   //Open the recordset
   rsProducts.Open(CRecordset::dynaset, _T(strSQL), CRecordset::none);

   //check to see if the recordset is open
   if(rsProducts.IsOpen())
   {
      //see if any records were returned
      if(rsProducts.GetRecordCount() != 0)
      {
         while(!rsProducts.IsEOF())
         {
            //get the data to display to the screen
            rsProducts.GetFieldValue("ProductID", varProductID, SQL_C_SLONG);
            rsProducts.GetFieldValue("ProductName", varProduct, SQL_C_CHAR);

            cout << varProductID.m_lVal << " - " << *varProduct.m_pstring;

            //move to the next record
            rsProducts.MoveNext();
         }
      }
   }
   //close the objects
   rsProducts.Close();
   db.Close();
   return 0;
}
```

Appendix E – HTML Form Validation using JavaScript

```
function IsNumeric(strString)
{
        var strValidChars = "0123456789.-";
        var strChar;
        var blnResult = true;
        if (strString.length == 0) return false;
        for (i = 0; i < strString.length && blnResult == true; i++)
        {
                strChar = strString.charAt(i);
                if (strValidChars.indexOf(strChar) == -1)
                {
                        blnResult = false;
                }
        }
        return blnResult;
}

function IsValidEmail(str)
{
        return (str.indexOf(".") > 2) && (str.indexOf("@") > 0);
}

function IsEmpty(strField)
{
        if ((strField.value.length==0) || (strField.value==null))
        {
                return true;
        }
        else
        {
                return false;
        }
}

function Validate()
{
        var myForm = document.form1;
        if (!IsEmpty(myForm.txtFirstName) && !IsEmpty(myForm.txtLastName) &&
myForm.selCompany.value!=0 && !IsEmpty(myForm.txtEmployeeNum) && !IsEmpty(myForm.txtStreet1) &&
!IsEmpty(myForm.txtCity) && !IsEmpty(myForm.txtPostalCode) && !IsEmpty(myForm.txtTele1) &&
!IsEmpty(myForm.txtTele2) && !IsEmpty(myForm.txtEmail) && !IsEmpty(myForm.txtOS) &&
!IsEmpty(myForm.txtSystemType) && !IsEmpty(myForm.txtISP) && !IsEmpty(myForm.txtInternetType) &&
!IsEmpty(myForm.txtBrowser) && !IsEmpty(myForm.txtProcessor) && !IsEmpty(myForm.txtRAM) &&
!IsEmpty(myForm.txtROM) && !IsEmpty(myForm.txtVideoCard))
        {
                if ((IsNumeric(myForm.txtCountryTele.value) || IsEmpty(myForm.txtCountryTele)) &&
(IsNumeric(myForm.txtCityTele.value) || IsEmpty(myForm.txtCityTele)) && IsNumeric(myForm.txtTele1.value)
&& IsNumeric(myForm.txtTele2.value))
                {
                        if ((IsEmpty(myForm.txtCountryCell) || IsNumeric(myForm.txtCountryCell.value)) &&
(IsEmpty(myForm.txtCityCell) || IsNumeric(myForm.txtCityCell.value)) && (IsEmpty(myForm.txtCell1) ||
IsNumeric(myForm.txtCell1.value)) && (IsEmpty(myForm.txtCell2) || IsNumeric(myForm.txtCell2.value)))
                        {
```

```
                                if ((IsEmpty(myForm.txtCountryFax) ||
IsNumeric(myForm.txtCountryFax.value)) && (IsEmpty(myForm.txtCityFax) ||
IsNumeric(myForm.txtCityFax.value)) && (IsEmpty(myForm.txtFax1) || IsNumeric(myForm.txtFax1.value)) &&
(IsEmpty(myForm.txtFax2) || IsNumeric(myForm.txtFax2.value)))
                                {
                                        if (IsValidEmail(myForm.txtEmail.value))
                                        {
                                                myForm.submit();
                                        }
                                }
                        }
                }
        }
        if (IsEmpty(myForm.txtFirstName))
        {
                txtValFirst.innerText = "You must enter a First name";
        }
        else
        {
                txtValFirst.innerText = "";
        }
        if (IsEmpty(myForm.txtLastName))
        {
                txtValLast.innerText = "You must enter a Last name";
        }
        else
        {
                txtValLast.innerText = "";
        }
        if (myForm.selCompany.value == 0)
        {
                txtValCompany.innerText = "You must select an employing company.";
        }
        else
        {
                txtValCompany.innerText = "";
        }
        if (IsEmpty(myForm.txtEmployeeNum))
        {
                txtValEmpNum.innerText = "You must enter an Employee Number.";
        }
        else
        {
                txtValEmpNum.innerText = "";
        }
        if (IsEmpty(myForm.txtStreet1))
        {
                txtValStreet1.innerText = "You must enter a street address.";
        }
        else
        {
                txtValStreet1.innerText = "";
        }
        if (IsEmpty(myForm.txtCity))
        {
                txtValCity.innerText = "You must enter a city.";
```

```
        }
        else
        {
                txtValCity.innerText = "";
        }
        if (!IsNumeric(myForm.txtPostalCode.value))
        {
                txtValZip.innerText = "The postal code must be numeric.";
        }
        else
        {
                if (IsEmpty(myForm.txtPostalCode))
                {
                        txtValZip.innerText = "You must enter a postal code.";
                }
                else
                {
                        txtValZip.innerText = "";
                }
        }
        if ((!IsEmpty(myForm.txtCountryTele) && !IsNumeric(myForm.txtCountryTele.value))||
(!IsEmpty(myForm.txtCityTele) && !IsNumeric(myForm.txtCityTele.value)) ||
!IsNumeric(myForm.txtTele1.value) || !IsNumeric(myForm.txtTele2.value))
        {
                txtValTele.innerText = "The phone number must be numeric.";
        }
        else
        {
                if (IsEmpty(myForm.txtTele1) || IsEmpty(myForm.txtTele2))
                {
                        txtValTele.innerText = "You must enter at least the last 7 digits of the telephone
number.";
                }
                else
                {
                        txtValTele.innerText = "";
                }
        }
        if ((!IsEmpty(myForm.txtCountryCell) && !IsNumeric(myForm.txtCountryCell.value))||
(!IsEmpty(myForm.txtCityCell) && !IsNumeric(myForm.txtCityCell.value)) || (!IsEmpty(myForm.txtCell1) &&
!IsNumeric(myForm.txtCell1.value)) || (!IsEmpty(myForm.txtCell2) && !IsNumeric(myForm.txtCell2.value)))
        {
                txtValCell.innerText = "The phone number must be numeric.";
        }
        else
        {
                txtValCell.innerText = "";
        }
        if ((!IsEmpty(myForm.txtCountryFax) && !IsNumeric(myForm.txtCountryFax.value))||
(!IsEmpty(myForm.txtCityFax) && !IsNumeric(myForm.txtCityFax.value)) || (!IsEmpty(myForm.txtFax1) &&
!IsNumeric(myForm.txtFax1.value)) || (!IsEmpty(myForm.txtFax2) && !IsNumeric(myForm.txtFax2.value)))
        {
                txtValFax.innerText = "The phone number must be numeric.";
        }
        else
        {
```

```
        txtValFax.innerText = "";
}
if (IsEmpty(myForm.txtEmail) || IsValidEmail(myForm.txtEmail.value))
{
        txtValEmail.innerText = "You must enter a valid email address.";
}
else
{
        txtValEmail.innerText = "";
}
if (IsEmpty(myForm.txtOS))
{
        txtValOS.innerText = "You must enter an operating system.";
}
else
{
        txtValOS.innerText = "";
}
if (IsEmpty(myForm.txtSystemType))
{
        txtValSystemType.innerText = "You must enter a system type.";
}
else
{
        txtValSystemType.innerText = "";
}
if (IsEmpty(myForm.txtISP))
{
        txtValISP.innerText = "You must enter an internet service provider.";
}
else
{
        txtValISP.innerText = "";
}
if (IsEmpty(myForm.txtInternetType))
{
        txtValInternetType.innerText = "You must enter an internet type.";
}
else
{
        txtValInternetType.innerText = "";
}
if (IsEmpty(myForm.txtBrowser))
{
        txtValBrowser.innerText = "You must enter an internet browser.";
}
else
{
        txtValBrowser.innerText = "";
}
if (IsEmpty(myForm.txtProcessor))
{
        txtValProcessor.innerText = "You must enter a processor.";
}
else
{
```

```
            txtValProcessor.innerText = "";
        }
        if (IsEmpty(myForm.txtRAM))
        {
            txtValRAM.innerText = "You must enter the system's RAM.";
        }
        else
        {
            txtValRAM.innerText = "";
        }
        if (IsEmpty(myForm.txtROM))
        {
            txtValROM.innerText = "You must enter the system's ROM.";
        }
        else
        {
            txtValROM.innerText = "";
        }
        if (IsEmpty(myForm.txtVideoCard))
        {
            txtValVideoCard.innerText = "You must enter a video card.";
        }
        else
        {
            txtValVideoCard.innerText = "";
        }
}
```

Appendix F – Sending Emails with C++

When tickets are opened, forwarded, reassigned, and closed, an email is generated and sent to the email addresses assigned in the database. This is done using the CFastSMTP class which was obtained from The Code Project website, see reference one. The following code is the header file.

```
#include <winsock2.h>
#include <assert.h>
#include <vector>
const int DEFAULT_PROTOCOL = 0;
const int NO_FLAGS = 0;

class CFastSmtp
{
public:
   CFastSmtp();
   virtual ~CFastSmtp();
   bool   AddRecipient(const char email[], const char name[]="");
   bool   AddBCCRecipient(const char email[], const char name[]="");
   bool   AddCCRecipient(const char email[], const char name[]="");
   bool   ConnectServer(const char server[], const unsigned short port=NULL);
   bool   Disconnect();
   bool   GetConnectStatus();
   const unsigned int   GetBCCRecipientCount();
   const unsigned int   GetCCRecipientCount();
   const unsigned int   GetRecipientCount();
   const unsigned int   GetSocket();
   const char*   const   GetLocalHostIp();
   const char*   const   GetLocalHostname();
   const char*   const   GetMessageBody();
   const char*   const   GetReplyTo();
   const char*   const   GetSenderEmail();
   const char*   const   GetSenderName();
   const char*   const   GetSubject();
   const char*   const   GetXMailer();
   bool   Send();
   void   SetMessageBody(const char body[]);
   void   SetSubject(const char subject[]);
   void   SetSenderName(const char name[]);
   void   SetSenderEmail(const char email[]);
   void   SetReplyTo(const char replyto[]);
   void   SetXMailer(const char xmailer[]);

private:
   class CRecipient
   {
   public:
      CRecipient()
      {
         m_pcEmail = NULL;
      };
```

```
  CRecipient& operator=(const CRecipient& src)
  {
    if (&src != this)
    {
      if (m_pcEmail)
        delete [] m_pcEmail;
      int s = strlen(src.m_pcEmail);
      m_pcEmail = new char[s+1];
      strcpy(m_pcEmail, src.m_pcEmail);
    }
    return (*this);
  };
  virtual ~CRecipient()
  {
    if (m_pcEmail)
      delete [] m_pcEmail;
  };
  char* GetEmail()
  {
    return m_pcEmail;
  };
  void SetEmail(const char email[])
  {
    assert(email);
    int s=strlen(email);
    if (s > 0)
    {
      m_pcEmail = new char[s+1];
      strcpy(m_pcEmail, email);
    }
  };
private:
  char *m_pcEmail;
};
bool bCon;
char m_cHostName[MAX_PATH];
char* m_pcFromEmail;
char* m_pcFromName;
char* m_pcSubject;
char* m_pcMsgBody;
char* m_pcXMailer;
char* m_pcReplyTo;
char* m_pcIPAddr;

WSADATA wsaData;
SOCKET hSocket;

std::vector<CRecipient*> Recipients;
std::vector<CRecipient*> CCRecipients;
std::vector<CRecipient*> BCCRecipients;

char* _formatHeader();
bool  _formatMessage();
SOCKET _connectServerSocket(const char* server, const unsigned short port=NULL);
};
```

This class defines a second class, CRecipient, within it for use in the functions of the CFastSMTP class. The following functions are the .cpp file associated with the class and show how the functions work.

```
//**********************************************************************
/*
 Fast & Simple SMTP Class

 Author:
  christopher w. backen <immortal@cox.net>
 Purpose:
  Simple smtp class with handy local ip and hostname functions
 ToDo:
  Attachments & UUEncode/Decode
*/
//**********************************************************************
//*** Change History
//*** DEVELOPER          DATE      MODIFICATION
//***
//*** christopher w. backen 09.05.2002   Misc. updates and corrections
//*** christopher w. backen   08.28.2001   Bug fixes & code changes
//*** christopher w. backen   04.12.2001   Release
//***
//**********************************************************************
#include "stdafx.h"
#include "FastSmtp.h"

//////////////////////////////////////////////////////////////////////
// Construction Destruction
//////////////////////////////////////////////////////////////////////
CFastSmtp::CFastSmtp()
{
   // Initialize variables
   bCon        = false;

   m_pcFromEmail   = NULL;
   m_pcFromName    = NULL;
   m_pcSubject     = NULL;
   m_pcMsgBody     = NULL;
   m_pcXMailer     = NULL;
   m_pcReplyTo     = NULL;
   m_pcIPAddr      = NULL;

   // Initialize WinSock
   WORD wVer   = MAKEWORD(2,2);
   if (WSAStartup(wVer,&wsaData) != NO_ERROR) {
      printf("WSAStartup %d\r\n", WSAGetLastError());
      throw;
   }
   if (LOBYTE( wsaData.wVersion ) != 2 || HIBYTE( wsaData.wVersion ) != 2 ) {
      printf("Can't find a useable WinSock DLL\r\n");
      WSACleanup();
      throw;
   }
```

```
}

CFastSmtp::~CFastSmtp()
{
  // Free recipient lists
  for (unsigned int n = 0; n < Recipients.size(); n++)
    delete Recipients[n];
  for (unsigned int n = 0; n < CCRecipients.size(); n++)
    delete CCRecipients[n];
  for (unsigned int n = 0; n < BCCRecipients.size(); n++)
    delete BCCRecipients[n];

  // Free variables
  if (m_pcFromEmail)
    delete m_pcFromEmail;
  if (m_pcFromName)
    delete m_pcFromName;
  if (m_pcSubject)
    delete m_pcSubject;
  if (m_pcMsgBody)
    delete m_pcMsgBody;
  if (m_pcXMailer)
    delete m_pcXMailer;
  if (m_pcReplyTo)
    delete m_pcReplyTo;

  // Close connection
  if (bCon)
    Disconnect();

  // Cleanup
  WSACleanup();
}

bool CFastSmtp::AddRecipient(const char email[], const char name[])
{
  assert(email);

  int s=strlen(email);
  if (s==0)
    return false;

  CRecipient *pRec = new CRecipient();

  char *pcBuf = new char[s+strlen(name)+4];
  sprintf(pcBuf,"%s<%s>",name,email);
  pRec->SetEmail(pcBuf);
  Recipients.insert(Recipients.end(), pRec);
  delete pcBuf;

  return (true);
}

bool CFastSmtp::AddCCRecipient(const char email[], const char name[])
{
  assert(email);
```

```
  int s=strlen(email);
  if (s==0)
     return false;

  CRecipient *pRec = new CRecipient();

  char *pcBuf = new char[s+strlen(name)+4];
   sprintf(pcBuf,"%s<%s>",name,email);
   pRec->SetEmail(pcBuf);
   CCRecipients.insert(CCRecipients.end(), pRec);
  delete pcBuf;

  return (true);
}

bool CFastSmtp::AddBCCRecipient(const char email[], const char name[])
{
  assert(email);

  int s=strlen(email);
  if (s==0)
     return false;
  CRecipient *pRec = new CRecipient();

  char *pcBuf = new char[s+strlen(name)+4];
   sprintf(pcBuf,"%s<%s>",name,email);
   pRec->SetEmail(pcBuf);
   BCCRecipients.insert(BCCRecipients.end(), pRec);
  delete pcBuf;

  return (true);
}

bool CFastSmtp::Disconnect()
{
  if (!bCon) {
     printf("Not connected to server!\r\n");
     return (false);
   }

  BYTE      sReceiveBuffer[4096];
  int       iLength = 0;
  int       iEnd = 0;

  if (send(hSocket, (LPSTR)"QUIT\r\n", strlen("QUIT\r\n"),
        NO_FLAGS) == SOCKET_ERROR) {
     printf("Socket send error: %d\r\n", WSAGetLastError());
     return (false);
   }
  iLength = recv(hSocket, (LPSTR)sReceiveBuffer+iEnd,sizeof(sReceiveBuffer)-iEnd,
            NO_FLAGS);
  iEnd += iLength;
  sReceiveBuffer[iEnd] = '\0';
```

```
    bCon=false;

    hSocket=NULL;

    return (true);
}

bool CFastSmtp::Send()
{
// verify sender email
    if (m_pcFromEmail == NULL) {
      printf("Please specifiy a sender email address\r\n");
      return (false);
    }

    BYTE      sReceiveBuffer[4096];
    int       iLength = 0;
    int       iEnd = 0;
    char      buf[4096];
// get proper header
    char* msgHeader = _formatHeader();

    if (msgHeader == NULL) {
      delete [] msgHeader;
      printf("Failed to format message header\r\n");
      return (false);
    }
// start
    strcpy(buf, "MAIL FROM: <");
    strcat(buf, m_pcFromEmail);
    strcat(buf, ">\r\n");
    if (send(hSocket, (LPSTR)buf, strlen(buf), NO_FLAGS) == SOCKET_ERROR) {
      printf("Socket send error: %d\r\n", WSAGetLastError());
      return (false);
    }
    iLength = recv(hSocket, (LPSTR)sReceiveBuffer+iEnd,sizeof(sReceiveBuffer)-iEnd,
              NO_FLAGS);
    iEnd += iLength;
    sReceiveBuffer[iEnd] = '\0';
// create message receipts
    char *token;
    for(unsigned int i=0;i<Recipients.size();i++) {
      token = strtok(Recipients.at(i)->GetEmail(),"<");
      token = strtok(NULL,"<");
      if (token == NULL)
        token = strtok(Recipients.at(i)->GetEmail(),"<");
      strcpy(buf, "RCPT TO: <");
      strcat(buf, token);
      strcat(buf, "\r\n");
      if (send(hSocket, (LPSTR)buf, strlen(buf), NO_FLAGS) == SOCKET_ERROR) {
        printf("Socket send error: %d\r\n", WSAGetLastError());
        return (false);
      }
      iLength = recv(hSocket,
                (LPSTR)sReceiveBuffer+iEnd,sizeof(sReceiveBuffer)-iEnd,
                 NO_FLAGS);
```

```
    iEnd += iLength;
    sReceiveBuffer[iEnd] = '\0';
  }
  for(unsigned int i=0;i<CCRecipients.size();i++) {
    token = strtok(CCRecipients.at(i)->GetEmail(),"<");
    token = strtok(NULL,"<");
    if (token == NULL)
      token = strtok(Recipients.at(i)->GetEmail(),"<");
    strcpy(buf, "RCPT TO: <");
    strcat(buf, token);
    strcat(buf, "\r\n");
    if (send(hSocket, (LPSTR)buf, strlen(buf), NO_FLAGS) == SOCKET_ERROR) {
      printf("Socket send error: %d\r\n", WSAGetLastError());
      return (false);
    }
    iLength = recv(hSocket,
              (LPSTR)sReceiveBuffer+iEnd,sizeof(sReceiveBuffer)-iEnd,
               NO_FLAGS);
    iEnd += iLength;
    sReceiveBuffer[iEnd] = '\0';
  }
  for(unsigned int i=0;i<BCCRecipients.size();i++) {
    token = strtok(BCCRecipients.at(i)->GetEmail(),"<");
    token = strtok(NULL,"<");
    if (token == NULL)
      token = strtok(Recipients.at(i)->GetEmail(),"<");
    strcpy(buf, "RCPT TO: <");
    strcat(buf, token);
    strcat(buf, "\r\n");
    if (send(hSocket, (LPSTR)buf, strlen(buf), NO_FLAGS) == SOCKET_ERROR) {
      printf("Socket send error: %d\r\n", WSAGetLastError());
      return (false);
    }
    iLength = recv(hSocket, (LPSTR)sReceiveBuffer+iEnd,
              sizeof(sReceiveBuffer)-iEnd,
              NO_FLAGS);
    iEnd += iLength;
    sReceiveBuffer[iEnd] = '\0';
  }
// init data
  if (send(hSocket, (LPSTR)"DATA\r\n", strlen("DATA\r\n"), NO_FLAGS) == SOCKET_ERROR) {
    printf("Socket send error: %d\r\n", WSAGetLastError());
    return (false);
  }
  iLength = recv(hSocket, (LPSTR)sReceiveBuffer+iEnd,sizeof(sReceiveBuffer)-iEnd,
            NO_FLAGS);
  iEnd += iLength;
  sReceiveBuffer[iEnd] = '\0';
// send header
  if (send(hSocket,
        (LPSTR)msgHeader, strlen(msgHeader), NO_FLAGS) == SOCKET_ERROR) {
    printf("Socket send error: %d\r\n", WSAGetLastError());
    delete [] msgHeader;
    return (false);
  }
// send body
```

```
    if (send(hSocket,
          (LPSTR)m_pcMsgBody, strlen(m_pcMsgBody), NO_FLAGS) == SOCKET_ERROR) {
       printf("Socket send error: %d\r\n", WSAGetLastError());
       return (false);
    }
// signal end
    if (send(hSocket,
          (LPSTR)"\r\n.\r\n", strlen("\r\n.\r\n"), NO_FLAGS) == SOCKET_ERROR) {
       printf("Socket send error: %d\r\n", WSAGetLastError());
       return (false);
    }
    iLength = recv(hSocket, (LPSTR)sReceiveBuffer+iEnd,sizeof(sReceiveBuffer)-iEnd,
                NO_FLAGS);
    iEnd += iLength;
    sReceiveBuffer[iEnd] = '\0';
    delete [] msgHeader;
    return (true);

}

bool CFastSmtp::ConnectServer(const char server[], const unsigned short port)
{
    assert(server);

    if (bCon)
       Disconnect();
    bCon=false;
    hSocket = INVALID_SOCKET;

    hSocket = _connectServerSocket(server, port);
    if (hSocket != INVALID_SOCKET) {
       BYTE      sReceiveBuffer[4096];
       int       iLength = 0;
       int       iEnd = 0;
       char      buf[4096];

       strcpy(buf, "HELO ");
       strcat(buf, server);
       strcat(buf, "\r\n");
       if (send(hSocket, (LPSTR)buf, strlen(buf), NO_FLAGS) == SOCKET_ERROR) {
          printf("Socket send error: %d\r\n", WSAGetLastError());
          return (false);
       }
       iLength = recv(hSocket,
                   (LPSTR)sReceiveBuffer+iEnd,sizeof(sReceiveBuffer)-iEnd,
                    NO_FLAGS);
       iEnd += iLength;
       sReceiveBuffer[iEnd] = '\0';
    } else {
       printf("Invalid socket\r\n");
       return (false);
    }

    bCon=true;
    return (true);
}
```

```
SOCKET CFastSmtp::_connectServerSocket(const char server[],
                        const unsigned short port)
{
  int           nConnect;
  short         nProtocolPort;
  LPHOSTENT     lpHostEnt;
  LPSERVENT     lpServEnt;
  SOCKADDR_IN       sockAddr;

  SOCKET        hServerSocket = INVALID_SOCKET;

  lpHostEnt = gethostbyname(server);
  if (lpHostEnt) {
    hServerSocket = socket(PF_INET, SOCK_STREAM,DEFAULT_PROTOCOL);
    if (hServerSocket != INVALID_SOCKET) {
      if (port != NULL) {
        nProtocolPort = port;
      }else{
        lpServEnt = getservbyname("mail", DEFAULT_PROTOCOL);
        if (lpServEnt == NULL)
          nProtocolPort = htons(IPPORT_SMTP);
        else
          nProtocolPort = lpServEnt->s_port;
      }
      sockAddr.sin_family = AF_INET;
      sockAddr.sin_port = nProtocolPort;
      sockAddr.sin_addr = *((LPIN_ADDR)*lpHostEnt->h_addr_list);
      nConnect = connect(hServerSocket, (PSOCKADDR)&sockAddr,
                 sizeof(sockAddr));
      if(nConnect)
        hServerSocket = INVALID_SOCKET;
    } else {
      printf("Invalid socket\r\n");
      throw;
    }
  }
  return(hServerSocket);
}

char* CFastSmtp::_formatHeader()
{
// check for at least one recipient
  if(Recipients.size() <= 0 )  {
    printf("Please add a message recipient!\r\n");
    return NULL;
  }
  int s=0;
  char *msgHeader = new char[16385];
  //char to[1024];
  for (unsigned int i=s=0;i<Recipients.size();i++) {
    s+=strlen(Recipients.at(i)->GetEmail())+1;
  } if (s==0) s=1; char *to = new char[s];
  //char cc[1024];
  for (i=s=0;i<CCRecipients.size();i++) {
    s+=strlen(CCRecipients.at(i)->GetEmail())+1;
```

```
} if (s==0) s=1; char *cc = new char[s];
//char bcc[1024];
for (i=s=0;i<BCCRecipients.size();i++) {
  s+=strlen(BCCRecipients.at(i)->GetEmail())+1;
} if (s==0) s=1; char *bcc = new char[s];

TCHAR szDate[500];
TCHAR sztTime[500];

// create the recipient string, cc string, and bcc string
to[0] = '\0';
for (i=0;i<Recipients.size();i++) {
  i > 0 ? strcat(to,","):strcat(to,"");
  strcat(to,Recipients.at(i)->GetEmail());
}

cc[0] = '\0';
for (i=0;i<CCRecipients.size();i++) {
  i > 0 ? strcat(cc,","):strcat(cc,"");
  strcat(cc,CCRecipients.at(i)->GetEmail());
}

bcc[0] = '\0';
for (i=0;i<BCCRecipients.size();i++) {
  i > 0 ? strcat(bcc,","):strcat(bcc,"");
  strcat(bcc,BCCRecipients.at(i)->GetEmail());
}
// get the current date and time
SYSTEMTIME st={0};
::GetSystemTime(&st);
::GetDateFormat(LOCALE_SYSTEM_DEFAULT,0,&st,"ddd',' dd MMM yyyy",szDate,sizeof(szDate));
::GetTimeFormat(LOCALE_SYSTEM_DEFAULT,TIME_FORCE24HOURFORMAT,&st,"HH':'mm':'ss
tt",sztTime,sizeof(sztTime));
// here it is...the main data of the message
wsprintf(msgHeader,"DATE: %s %s\r\n", szDate, sztTime);
if (m_pcFromName != NULL)
{
  strcat(msgHeader,"FROM: ");
  strcat(msgHeader, m_pcFromName);
  strcat(msgHeader, "\r\n");
}
strcat(msgHeader,"To: ");
strcat(msgHeader, to);
strcat(msgHeader, "\r\n");
strcat(msgHeader,"Cc: ");
strcat(msgHeader, cc);
strcat(msgHeader, "\r\n");
if (m_pcSubject != NULL)
{
  strcat(msgHeader, "Subject: ");
  strcat(msgHeader, m_pcSubject);
  strcat(msgHeader, "\r\n");
}
if (m_pcXMailer != NULL)
{
  strcat(msgHeader,"X-Mailer: ");
```

```
    strcat(msgHeader, m_pcXMailer);
    strcat(msgHeader, "\r\n");
  }
// start optional fields
  if (m_pcReplyTo != NULL)
  {
    strcat(msgHeader, "Reply-To: ");
    strcat(msgHeader, m_pcReplyTo);
    strcat(msgHeader, "\r\n");
  }
// start MIME versions
  strcat(msgHeader, "MIME-Version: 1.0\r\nContent-type: text/plain; charset=US-ASCII\r\n");
// send header finish command
  strcat(msgHeader, "\r\n");
// clean up
  delete to;
  delete cc;
  delete bcc;
// done
  return msgHeader;
}

const char* const CFastSmtp::GetLocalHostIp()
{
  in_addr   *iaHost;

  if (gethostname(m_cHostName,255) != SOCKET_ERROR) {
    HOSTENT *pHe = NULL;
    pHe = gethostbyname(m_cHostName);
    if (pHe != NULL) {
      for (int i=0;pHe->h_addr_list[i] != 0;i++) {
        iaHost = (LPIN_ADDR)pHe->h_addr_list[i];
        m_pcIPAddr = inet_ntoa(*iaHost);
      }
    } else {
      DWORD dErr = WSAGetLastError();
      printf("Failed to get the local ip address\r\n");
    }

  return m_pcIPAddr;
}

const char* const CFastSmtp::GetLocalHostname()
{
  if (gethostname((char FAR*)m_cHostName,255) == SOCKET_ERROR)
    printf("Failed to get the local hostname\r\n");
  return m_cHostName;
}

//*****************************************************************************
//*** Properties
//*****************************************************************************
bool CFastSmtp::GetConnectStatus()
{
```

```
    return (bCon);
}

unsigned const int CFastSmtp::GetBCCRecipientCount()
{
    return (BCCRecipients.size());
}

unsigned const int CFastSmtp::GetCCRecipientCount()
{
    return (CCRecipients.size());
}

unsigned const int CFastSmtp::GetSocket()
{
    return (hSocket);
}

const char* const CFastSmtp::GetMessageBody()
{
    return (m_pcMsgBody);
}

unsigned const int CFastSmtp::GetRecipientCount()
{
    return (Recipients.size());
}

const char* const CFastSmtp::GetReplyTo()
{
    return (m_pcReplyTo);
}

const char* const CFastSmtp::GetSenderEmail()
{
    return (m_pcFromEmail);
}

const char* const CFastSmtp::GetSenderName()
{
    return (m_pcFromName);
}

const char* const CFastSmtp::GetSubject()
{
    return (m_pcSubject);
}

const char* const CFastSmtp::GetXMailer()
{
    return (m_pcXMailer);
}

void CFastSmtp::SetMessageBody(const char body[])
{
    assert(body);
```

```
    int s=strlen(body);
    if (m_pcMsgBody)
       delete [] m_pcMsgBody;
    m_pcMsgBody = new char[s+1];
    strcpy(m_pcMsgBody, body);
}

void CFastSmtp::SetReplyTo(const char replyto[])
{
    assert(replyto);
    int s=strlen(replyto);
    if (m_pcReplyTo)
       delete [] m_pcReplyTo;
    m_pcReplyTo = new char[s+1];
    strcpy(m_pcReplyTo, replyto);
}

void CFastSmtp::SetSenderEmail(const char email[])
{
    assert(email);
    int s=strlen(email);
    if (m_pcFromEmail)
       delete [] m_pcFromEmail;
    m_pcFromEmail = new char[s+1];
    strcpy(m_pcFromEmail, email);
}

void CFastSmtp::SetSenderName(const char name[])
{
    assert(name);
    int s=strlen(name);
    if (m_pcFromName)
       delete [] m_pcFromName;
    m_pcFromName = new char[s+1];
    strcpy(m_pcFromName, name);
}

void CFastSmtp::SetSubject(const char subject[])
{
    assert(subject);
    int s=strlen(subject);
    if (m_pcSubject)
       delete [] m_pcSubject;
    m_pcSubject = new char[s+1];
    strcpy(m_pcSubject, subject);
}

void CFastSmtp::SetXMailer(const char xmailer[])
{
    assert(xmailer);
    int s=strlen(xmailer);
    if (m_pcXMailer)
       delete [] m_pcXMailer;
    m_pcXMailer = new char[s+1];
    strcpy(m_pcXMailer, xmailer);
}
```

Now that the code has been displayed, here is how it is used. The class must be added into the project and an instance of the class must be created. Just to create and send a very simple email, the server for the email to go through must be set. This is done with the ConnectServer() function. If the connection is established, the email must be given a sender name and address, recipient address, subject, and message body. A carbon copy can be sent and a blind carbon copy as well, but those option are optional. After the email has been completely set up, a connection to the server is created. If this is successful, the email is sent and then the connection is disconnected. The following code demonstrates exactly how to set up and send the email.

```
//send email to client
CFastSmtp mail;

if (mail.ConnectServer("mail.cinci.rr.com"))
{
        mail.SetSenderName("John Doe");
        mail.SetSenderEmail("john.doe@cinci.rr.com");
        mail.SetSubject("Test Email");
        mail.AddRecipient("jane.doe@cinci.rr.com");
        mail.AddCCRecipient("");
        mail.AddBCCRecipient("");
        mail.SetMessageBody("This is the body of the test email.");

        if (mail.GetConnectStatus())
        {
                mail.Send();
                mail.Disconnect();
        }
}
```

Appendix G – Uploading Files using C++

Uploading files to a remote server using C++ requires the use of two classes: CInternetSession and CFtpConnection. The CInternetSession class is required to create an internet connection which is used in getting an FTP connection for the file upload. This is done with the function GetFtpConnection(). After the FTP connection has been established, it is wise to check to see what folder is the root directory the FTP connection has connected to for uploading. If the connection is being made to a UNIX server, this root directory will probably be the "/" which typically represents "root" on a UNIX operating system. After getting the the current directory, GetCurrentDirectory(), you can set the directory where you want to upload files by calling the SetCurrentDirectory() function. This function will return a true if the directory change was successful. If the directory does not exist it will return false. If you wish to create a directory, perhaps in the event that the SetCurrentDirectory() function returns false, call the CreateDirectory() function. If SetCurrentDirectory() returns false, the current directory will remain as the root directory and calling CreateDirectory() should be typed out as a full path. It will accept a partial path. After you have changed to the directory where the files will be uploaded, calling the PutFile() function will perform the actual upload. Below is some sample code on how to actually upload a file with all of the parameters for these functions. The key thing to remember is that an absolute path on a UNIX machine will look something like this: "/Attachment/Ticket." The "/" for the root directory is also recognized to mean "look in the current directory" so from a UNIX machine if the current directory has been set to "/Attachment/Ticket," uploading the file with the absolute path of /Attachment/Ticket/sometext.txt" will cause the function to look within the Ticket folder for another folder called Attachment. You must be in the root directory in order to upload the file with the absolute path. The following code is designed for a UNIX server, but could be used with any operating system as long as absolute paths are supplied to the functions.

The GetCurrentDirectory() function requires a CString parameter to store the directory returned. The SetCurrentDirectory() function requires a CString parameter telling it the folder to change to the current directory. The CreateDirectory() function requires a CString parameter containing the name for the new directory. Finally, the PutFile() function requires the local file name, remote filename, upload type, and a boolean for simple file transfer.

```
//set up the internet session object pointer and initialize it
CInternetSession *m_pInetSession;
m_pInetSession = new CInternetSession("My internet");

//create an ftp object and initialize it with the session object
CFtpConnection* m_pFtpConnection;
m_pFtpConnection = m_pInetSession->GetFtpConnection((LPCTSTR)ServerName, UserID, Password);

CString strRoot;
string strDirectory = "/Attachment/Ticket";
string strFile = "C:\Temp\sometext.txt";
string strFileName = strFile.substr(strFile.rfind("\\"), strFile.length());
string strNewFile = strDirectory + "/" + strFileName;

m_pFtpConnection->GetCurrentDirectory(strRoot);

// Do connection to the FTP server
boolDir = m_pFtpConnection->SetCurrentDirectory(strDirectory.c_str());
if (boolDir == false)
{
        m_pFtpConnection->CreateDirectory(strDirectory.c_str());
}
else
{
        m_pFtpConnection->SetCurrentDirectory(strRoot);
}

if ( m_pFtpConnection->PutFile(strFile.c_str(), strNewFile.c_str(), FTP_TRANSFER_TYPE_BINARY, 1) == 1 )
{
        strMessage = "File :" + strFileName + " uploaded successfully.";
        MessageBox(NULL, strMessage.c_str(), "Success", MB_OK);
        boolSuccess = true;
}
else
{
        strMessage = "Error in file :" + strNewFile + " upload.";
        MessageBox(NULL, strMessage.c_str(), "Error", MB_OK);
        boolSuccess = false;
}
```

References

1. Backen, Christopher W. "CFastSmtp - Fast and easy SMTP class..." The Code Project. http://www.codeproject.com/internet/zsmtp.asp. April 16, 2005.

2. "BridgeTrak Help Desk Software Solution." Kemma Software. 2004. http://www.kemma.com/bridgetrak.htm. November 1, 2004.

3. "CGIScripter – Write Perl CGI scripts for MySQL, Oracle, Access, SQL Server, Sybase, and DB2 data." .COM Solutions Inc. http://www.cgiscripter.net/products/cgiscripter/. November 2, 2004.

4. "Help Desk Software World – Helpdesk Software Information." Help Desk Management Software World. 2003. http://www.help-desk-world.com/. October 20, 2004.

5. "HelpSTAR- Help Desk Software and Asset Management." Help Desk Technology International Corporation. 2004. http://www.helpstar.com/. October 20, 2004.

6. "Macromedia." Macromedia, Inc. 2004. http://www.macromedia.com/. November 2, 2004.

7. Microsoft Corporate Sales. Microsoft Corporation. Personal Interview. November 2, 2004.

8. "Microsoft Product Information Center." Microsoft Corporation. 2004. http://www.microsoft.com/products/info/list.aspx?view=22&type=all. November 2, 2004.

9. Microsoft Training Department. Microsoft Corporation. Personal Interview. October 15, 2004.

10. "Price Grabber." PriceGrabber.com. 2004. http://www.pricegrabber.com/. November 2, 2004.

11. "Price Watch." PriceWatch.com. 1995. http://www.pricewatch.com/. November 2, 2004.

12. "Shop Smart, Shop Fast, Shopzilla!" Shopzilla International. 2004. http://www.shopzilla.com/. November 2, 2004.

13. Struempf, Ron. Director of Programming, Flightline Data Services, Inc. Personal interview. October 7, 2004.

14. Stone, Julie. Flightline Customer Service, Flightline Data Services, Inc. Personal interview. October 18, 2004.