Web Parts at Work: Healthcare Provider's Dashboard

Doug Forst
Computing and Information Systems
University of Wisconsin Stevens Point
Stevens Point, WI 54481
Douglas.J.Forst@uwsp.edu

Josh Eide
Computing and Information Systems
University of Wisconsin Stevens Point
Stevens Point, WI 54481
joshua.t.eide@uwsp.edu

Robert Dollinger
Computing and Information Systems
University of Wisconsin Stevens Point
Stevens Point, WI 54481
rdolling@uwsp.edu

Abstract

Two technological breakthroughs seem to get an edge in providing a completely new type of user experience for Web applications: the first one is the family of Web Parts controls, and the second one consists of the set of technologies generally known as AJAX (Asynchronous JavaScript And XML). This paper elaborates on the development experience related to the creation of a customizable and dynamic **Healthcare Provider's Dashboard.** This is a custom application used to display and manage important information and tasks associated with a healthcare provider. The main objective of the project was to fit as much information as possible into the interface, while making the pages customizable and dynamic such that each user would be able to customize the interface to his/her own liking. Web Parts, available in ASP.NET 2.0 within Visual Studio 2005, have been identified as the right tool to achieve this kind of functionality.

1 Introduction

Web application developers face new challenges in providing innovative features that insure an improved user experience. Customizable, dynamic, flexible and responsive Web pages become the norm for the modern applications. Two technological breakthroughs seem to get an edge in providing such features: the first one is the family of Web Parts controls, introduced by Microsoft with ASP.NET 2.0, and the second one consists of the set of technologies generally known as AJAX (Asynchronous JavaScript And XML). This paper elaborates on the development experience related to the creation of a customizable and dynamic **Healthcare Provider's Dashboard.** This is a custom application used to display and manage important information and tasks associated with a healthcare provider. The main objective of the project was to fit as much information as possible into the interface, while making the pages customizable and dynamic such that each user would be able to customize the interface to his/her own liking. This means the user can move around and reconfigure items on the interface, dynamically add/remove items, and save these personal settings to be retrieved the next time the user logged in again. Web Parts, available in ASP.NET 2.0 within Visual Studio 2005, have been identified as the right tool to achieve this kind of functionality. In a second paper, we describe how the use of AJAX provides flexibility and responsiveness to our application.

ASP.NET Web Parts enable you to build Web pages with modular layouts in which users can modify the appearance and content to suit their preferences. A key Web Parts feature, known as personalization, lets you save user-specific settings for each page and reuse those settings in future browser sessions. Other features include drag-and-drop, connections between Web Parts, custom verbs (verbs are represented in the UI as buttons, links, or menu items), themes, and add/remove capabilities. The Web Parts are all managed by a WebPartManager, and implement many different controls and classes. Each Web Part contains a Web User Control that can be personalized through an associated EditorZone, which has been specifically created and customized for that control. Four custom Web User Controls have been created, each of which allows the user to customize what they view on the page. Our first User Control, Table Viewer, allows the user to view information from a database in an ASP.NET GridView control. This Table Viewer Web Part allows editing in the following ways: displayed table, items displayed, width, columns viewed, and popup style, which displays more details. Three other Web User Controls that we have created include a RSS Feed, a Search Engine, and a Calendar. In addition to the ability to customize each Web Part, the user can also add new and previously closed Web Parts onto the page. More than one instance of each Web Part can be added to the Web page and each Web Part instance can be individually customized. Themes and skin files are also used to change the color scheme of the page.

1.1 What is a Provider Dashboard?

A Provider Dashboard is an application that condenses a provider's most important information into one single application. The application features an interface from which providers can quickly and easily have access to a large variety of information like:

appointments they may have, patient history, drugs that they are able to prescribe, hospital rounds list, “to do list”, list of documents to sign and many other. The dashboard also includes a calendar and RSS feeds displaying the day’s top healthcare headlines. The screen is sectioned off into parts, which contain the different pieces of information. Each part can be customized or drilled into for even more information. Given that in the healthcare industry, efficiency and timing is critical, the main purpose of a provider dashboard is to quickly and easily visualize information important to a healthcare provider; as opposed to using multiple applications to get the same information, in an inefficient and time consuming manner. In addition, all this information needs to fit into a small interface, since the providers use a tablet PC with a15 inches screen to retrieve all of their information.

The objective of this project was to develop a user interface to display this much needed information in a compact and user efficient way that is also dynamically customizable for/by each and every user. User customization means that users are able to reconfigure the interface to their own liking and have the configuration settings saved for the next time they login.

The current Patient Dashboard in production is very standardized and it’s not configurable or customizable. It does not optimize space and is dominated by user controls, not the information. The current application lacks efficiency and cannot assemble provider tasks and responsibilities. A new dashboard named Provider Dashboard needed to be designed as a prototype that will solve the majority of the problems related to the Patient Dashboard.

The Marshfield Clinic IS department Marshfield Wisconsin identified the need to give their providers a strong and effective way of viewing information needed for patients as well as their own tasks. Basically, design a new user friendly and customizable version of the Provider Dashboard (GUI).

The proposed tool for this task was to use ASP.NET 2.0 within Visual Studio 2005. ASP.NET 2.0 has many new features, like the new Web Parts controls, that all allow for personalization and customization. Web Parts are very dynamic and can compact a lot of information into a limited screen. The components used to create this application are described in greater detail throughout this document.

A sample view of the final prototype developed in this project is given in figure 1.

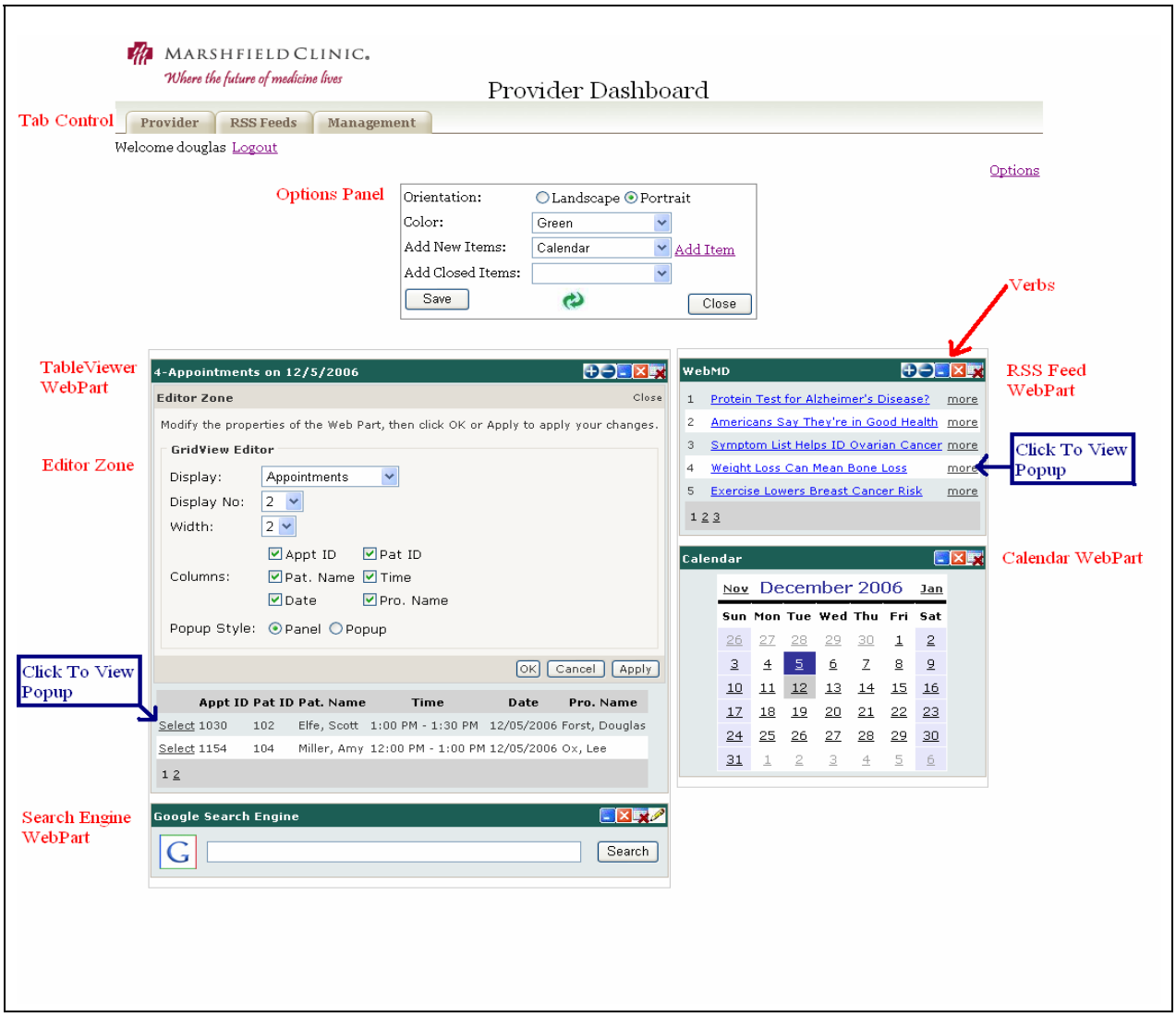

Figure 1: Final Prototype Sample View

1.2 What is a Web Part?

ASP.NET Web Parts controls are an integrated set of controls for creating Web sites that enable end users to modify the content, appearance, and behavior of Web pages directly in a browser. A Web Part can be a web user control or a simple user control. There are two or three columns that can contain controls. These columns in Web Parts terminology are called zones, i.e. regions on a page that contain Web Parts controls. Zones exist to lay out Web Parts controls on a page, and to provide a common UI for the controls. There can be one or many zones on a page, each zone can contain one or many Web Parts controls, and each zone can have a vertical or horizontal orientation for page layout [1]. Once a web user control or an ASP.NET user control is placed within a Web Part Zone it becomes a Web Part.

2 Requirements

Before starting to build a provider dashboard, using Web Parts, a couple of components need to be set up and configured.

For the application to be customizable per user, Personalization (new feature in ASP.NET 2.0) needed to be implemented. Once Personalization is setup, the Web Site Administration Tool can be used to configure it. This includes allowing or denying specific user(s) or group(s) access to a Web page or Web site.

ASP.NET Web Parts enable you to build Web pages with modular layouts in which users can modify the appearance and content to suit their preferences. A key Web Parts feature known as personalization lets you save user-specific settings for each page and reuse those settings in future browser sessions [2]. ASP.NET typically saves these user-specific settings into a MSSQL database. There are a couple of steps to follow in order to create a database where to store the user-specific information related to personalization. By default ASP.NET tries to create a SQLExpress database on the local machine when a Web page is ran for the first time with the WebPartManager. However, most machines will not have SQLExpress installed, in which case the alternative solution at hand is to create a dedicated database on any SQL server at hand. .NET Framework 2.0 provides a configuration tool, aspnet_regsql.exe, which allows a personalization database to be setup on a remote SQL server. This tool is located in a special windows directory at: C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\.

Web site configuration settings are stored in an XML file named Web.config, which is located in the root folder of the Web site. The Web Site Administration Tool lets you change your site configuration without having to manually edit the Web.config file. The first time that you use the Web Site Administration Tool to administer a specific Web site, if no Web.config file exists, the Web Site Administration Tool creates one. By default, the Web Site Administration Tool also creates a database in the App_Data folder of the Web site to store application services data, such as membership and roles information. For most settings, changes that are made in the Web Site Administration Tool take effect immediately and are reflected in the Web.config file [3].

Use the Security tab of the Web Site Administration Tool to manage rules for securing specific resources in the Web application. ASP.NET uses a security system that lets you restrict access to specific user accounts or the roles to which the user accounts belong. With the Security tab, you manage user accounts, roles, and access rules for the Web site. Before using the Security tab for the first time, use the Security Setup Wizard to configure basic security settings for the Web site [4].

3 Web Part Manager

ASP.NET provides a control called the WebPartManager, which managers all of the components related to Web Parts on the page. These components include Web Part Zones, Editor Zone, Catalog Zone, and Connection Zone.

The WebPartManager control acts as the hub or control center of a Web Parts application, which is responsible for managing and coordinating all controls inside WebZones. There must be one and only one WebPartManager control instance on every page that uses Web Parts controls. As with most aspects of Web Parts applications, the WebPartManager control works only with authenticated users. Furthermore, its functionality works almost entirely with server controls that reside within Web Parts zones that inherit from the WebZone class. Server controls that reside on a page outside of these zones can have very little Web Parts functionality or interaction with the WebPartManager control [5]. We have placed the WebPartManager on the master page of the Healthcare Provider's Dashboard application. This WebPartManager will control the Web Parts for all of the pages, so it is not necessary to add one to each Web page of the application.

The WebPartManager is used to set the Display Mode for the page. Display modes enable end users to perform certain tasks to modify or personalize a page, such as editing Web Parts controls, changing the layout of a page, or adding new controls from a catalog of available controls[6]. Within the page load method of the master page the WebPartManager display mode is set to EditDisplayMode. This enables the user to always have the option to edit each Web Part and also have the option to drag-and-drop each. Table 1 lists each display mode that the WebPartManager can use.

Value	Description
BrowserDisplayMode	"Normal" display mode; no editing (default)
DesignDisplayMode	Permits drag-and-drop layout editing.
EditDisplayMode	Permits editing of Web Parts' appearance and behavior.
CatalogDisplayMode	Permits Web Parts to be added to the page
ConnectDisplayMode	Permits connections to be established between Web parts

Table 1**:** WebPartManager Display Modes

Generic Zones

At run time, the Web Part Zone(s) wraps both controls (server controls or web user controls) with a GenericWebPart control. When a GenericWebPart control wraps a server control, the generic part control is the parent control, and you can access the server control through the parent controls ChildControl property. This use of generic part controls enable standard web server controls to have the same basic behavior and attribute as Web Parts controls that derive from the WebPart class.

EditorZone

The EditorZone control is one of the fundamental controls in the Web Parts control set. A key feature of Web Parts is the ability of end users to modify (or personalize) Web pages

according to their preferences, and save their personalized settings for future browser sessions. One aspect of modifying Web Parts pages includes editing. Within the Web Page users can edit the appearance, layout, behavior, and other properties of the visible WebPart controls. Several controls in the Web Parts control set provide the editing features, including the EditorZone control. An EditorZone control becomes visible when a Web Parts page enters edit mode, and it makes available various EditorPart controls that can be used to personalize WebPart controls [7].

There are four Editor Parts that can be added to the EditorZone. Table 2 lists each one of these parts. In the Provider Dashboard none of these default editor zones are used, instead we have created our own Editor Part for each Web Part. The Editor Part that pertains to each Web Part is specified within each Web Part's user control. The Editor Part is specified in the IWebEditable region of each user control.

Value	Description
AppearanceEditorPart	Provides UI for editing titles and other UI-related properties
BehaviorEditorPart	Provides UI for editing behavioral properties (e.g., can the Web Part be closed?)
LayoutEditorPart	Provides UI for editing Web Part's display state (minimized or restored), zone, and zone index
PropertyGridEditorPart	Provides property grid for editing custom properties

Table 2: EditorZone EditorParts

CatalogZone

The CatalogZoneserves as the primary control in the Web Parts control set for hosting ASP.NET server controls of type CatalogPart on a Web page. A CatalogZone control becomes visible only when a user switches a Web page to catalog display mode (CatalogDisplayMode). A catalog can contain several types of CatalogPart controls [8]. The CatalogZone can be drag-and-dropped on to the Web page from the Web Parts server controls. There are three Catalog Parts that can be added to the CatalogZone then. These parts are listed in Table 3. These server controls provide their own user interface but we have created our own CatalogZone interface for the Provide Dashboard.

Value	Description
PageCatalogPart	Lists Web parts that have been removed from the page
DeclarativeCatalogPart	Lists Web parts declared in a <WebPartsTemplate>
ImportCatalogPart	Permits Web parts to be imported from .WebPart files

Table 3: Catalog Zone Catalog Parts

Within the Provider Dashboard the PageCatalogPart and the DeclarativeCatalogPart are implemented. The PageCatalogPart is a list of the Web Parts that have been closed on the Web page. When a Web Part is closed it populates the Closed Web Parts drop down list within the options panel. The DeclarativeCatalogPart is a list of the available Web Parts that can be added to the page. They are declared within the <WebPartsTemplate> tag in

the source code of the site.Master page. A listing of these Web Parts is listed in the Add New Web Parts drop down list, located in the options panel on the site's Master page.

Web Part Connections

The WebPart connection is used to allow two WebParts to communicate with each other. Using Web Part connections, you can make it easier for your users to visualize the relationships that exist between items of data. For example, Web Part connections can model a master-detail scenario where one Web Part displaying a customer list is connected to another Web Part that displays the details of the currently selected customer. Web Part connections can also be used to model one-to-many relationships. As an example, one Web Part that displays a customer list can be connected to another Web Part that displays all the orders for the currently selected customer [9].

4 User Controls as Web Parts

Web user controls are derived from the System.Web.UI.UserControl namespace. These controls once created, can be added to the aspx page either at design time or programmatically during run time. But they lack the design time support of setting the properties created along with the control. They cannot run on their own and they need to stand on another platform like an aspx page [10].

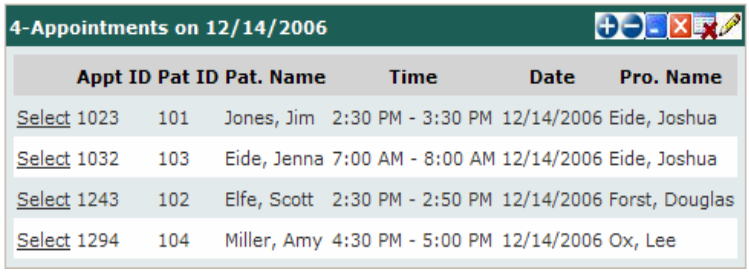

Figure 2: Table Viewer

Table Viewer

Our first web user control is the Table Viewer, which uses the ASP.NET tool GridView in order to display the user requested information from tables. The GridView automatically generates a column for every field and allows for paging and selection. The Table Viewer user control was created to allow for flexibility in what information was displayed, the amount displayed, items displayed, and the width. This again was derived from our focus on personalizing the information. We have also implemented a custom editor zone within the Table Viewer. This allows the user to view any given query of

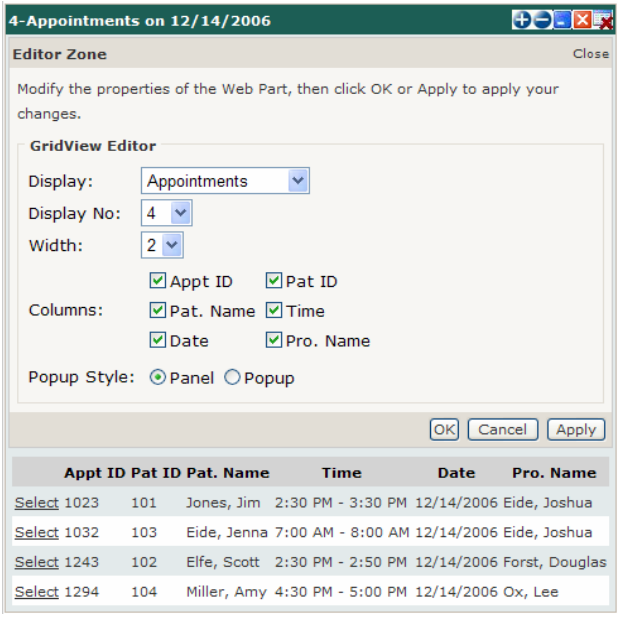

Figure 3: Table Viewer Editing

information from the DB, which is displayed in a table. Next we then allow the user to change how many rows are displayed (1-10) by using the editor zone. The verbs contained at the top of the user control can also change the number of rows to be displayed. Another option allows the user to choose which columns to display. The last customizable attribute of this user control is the type of popup they will view. The user will choose either a panel or a popup to display the selected information.

Figure 2 shows the table viewer, while figure 3 gives a sample view of the table viewer editing process.

RSS Feed

The RSS Feed web user control displays information from any RSS Feed. Within this user control we allow the user to change the displayed number, title, RSS feed choices, and the ability to remove feeds. We found a RSS Toolkit created by a programmer named Dmitry on the ASP.NET Team blog. Using this toolkit we access the RSS Feed, and then bind it to a GridView. We store the RSS Feed list in an ArrayList and the current RSS Feed for the Web Part is stored in a separate string. Each user's list of RSS feeds are user specific. Within the editor zone the user is able to add, remove, and rename any RSS Feed. Also, as in the Table Viewer the user is also able to change the type of popup style that they view.

Figure 4: RSS Feed

Figure 4 shows an RSS feed, while figure 5 gives a sample view of the RSS feed editing process.

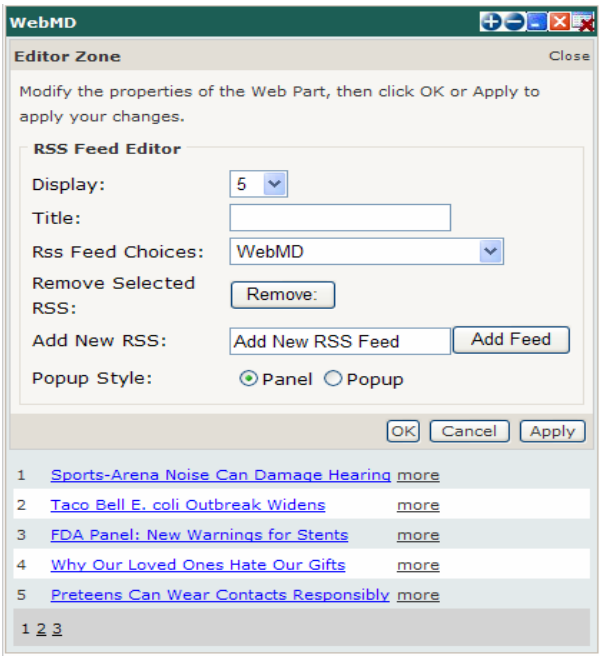

Figure 5: RSS Feed Editor

Calendar Web Part

The Calendar web user control is used to select the date that the user wants to see. The selected date can then be used in other Web Parts on the Web page. For example, it can be used to see the appointments on a specific day. The selected date is a session variable that is initially set to today's date in the Global.asax file. Then the "sessionDate" is set to the selected date when the Calendar's selected date

changed.

The selected date can also be passed through an Interface IMessage which can be used to connect Web Parts. However, using a session variable was much more convenient and it can be used anywhere in the application without creating an IMessage first. The Calendar Web Part does not implement a custom EditorZone. Therefore, clicking edit on the Web Part will not list any options to change (figure 6).

Figure 6: Web Part Calendar

Search Engine Web Part

The Search Engine Web Part allows the user do a Web search at four of most popular search engines on the Web. In the Search Engine editor zone the user can choose whether to search at Google, Ask.com, Yahoo, or Live (MSN). The search results are then displayed in a new window. Figure 7 shows the Search Engine Web Part and the Search Engine Editor.

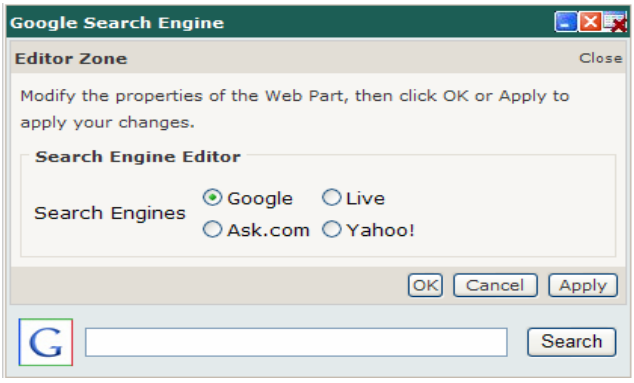

Figure 7: The Search Engine Web Part and Its Associated Editor Zone

Verbs

A custom Web Part control allows you to customize verbs, which provides user interface actions that users can perform on a Web Part control. Usually verbs are represented as buttons, links, or menu items. By default, common Web Part verbs appear on a drop-down list in each Web Part control's title bar. The IWebActionable method is responsible for showing verbs for a Web Part. You can customize the verbs by overriding and returning the WebPartVerbCollection. The two far left verbs allow the user to add or subtract rows to be displayed. The middle two verbs allow the Web Part to be minimized or closed respectively. The next verb allows the user to permanently delete the Web Part. The far right verb opens the editor zone to edit/customize the Web Part. Figure 7 shows a sample Verbs panel.

Figure 7: Verbs

Popup and Panel

We use the popup and panel control to display information from the RSS Feed or Table Viewer when a row is selected. We found a popup toolkit at CodeProject.com by Tomas Petricek [10]. When using a popup we use JavaScript to set two session variables to the x and y coordinates of the mouse click which is then used to display the popup. Information can be set to popup in a popup window or a panel as shown in figure 8.

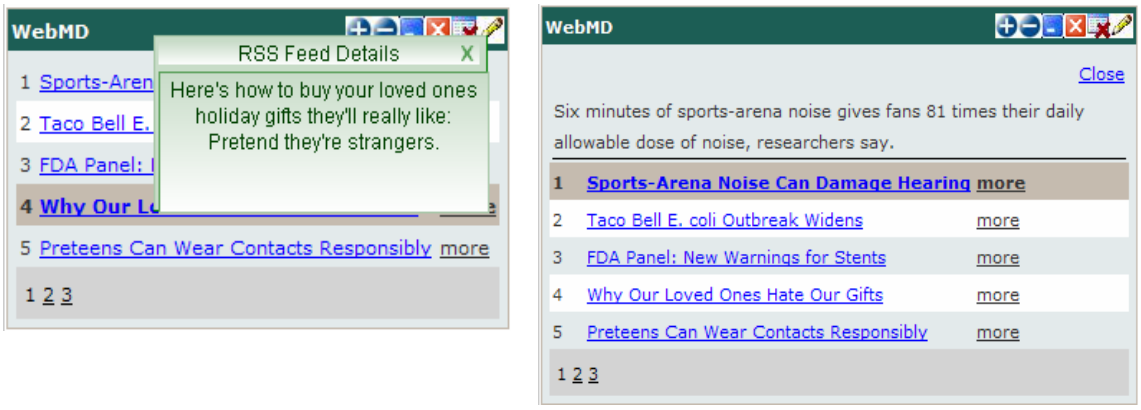

Figure 8: Popup and Panel

5 Classes

BaseUserControl Class

This class inherits the System.Web.UI.UserControl and IWebPart. All changes made to an EditorZone are stored within the declared variables in the BaseUserControl class. Therefore, this class will have all the properties/variables which we would like to edit and save for each user control. Every Web user control inherits the BaseUserControl.

DataAccessClass

This class is used to connect to a SQL database and run a query. The "ConnectToSQLDatabase" method within the class connects to an SQL database specified in the connection string. It is passed the SQL select command and returns a dataset with the results. The connection strings are located in the web.config file. All of the information used in the Provider Dashboard comes from one database. Each user's personalized settings are saved in a separate database.

Web.config

The web.config file is where all of the connection strings are located. The "LocalSQLServer" connection is for Personalization database. **Code Example 1** in figure 9 shows how the user's "Theme" and "Orientation" are declared and saved for each user. The "Theme" stores a string for the user's selected color theme. For example: "Blue", "Red", "Grey", or "Green". The Theme and Orientation are set in the site.Master page load method. The Orientation also stores a string variable, which is either "Landscape" (shows 3 columns/zones) or "Portrait" (shows 2 columns/zones).

```
<profile>
      <properties>
            <add name="Theme" type="System.String" />
            <add name="Orientation" type="System.String"/>
      </properties>
</profile>
```

Figure 9: Code Example 1: Profile properties (Theme and Orientation)

The web.config file is also where the authorization settings are stored. They can also be set through the Web Site Administration Tool. **Code Example 2** (figure 10) is an example of the application's permissions. In the Provider Dashboard each user must have a user account with the application in order to login.

```
<authorization>
      <deny users="?"/>
      <allow users="?"/>
</authorization>
<authentication mode="Forms"/>
```

Figure 10: Code Example 2: web.config Authentication

6 Other Features

Three other features that we used to create more functionality and also simplicity to our application are Themes, Orientation, and also the Master Page. Themes are used to change the color scheme of the Web Page, while Orientation is meant to help in assisting the user with changing from landscape to portrait, and finally the Master Page is used for simplicity and functionality.

Themes

Themes, a new feature of Microsoft ASP.NET 2.0, enable you to define the appearance of a set of controls once and apply the appearance to your entire Web application. For example, by taking advantage of themes, you can define a common appearance for all of the GridView controls in your application, such as the background and foreground color, in one central location. Themes enable you to easily create and maintain a consistent look throughout your Web site.

Themes are not the same thing as cascading style sheets. Cascading style sheets enable you to control the appearance of HTML tags on the browser. Themes, on the other hand, are applied on the server and they apply to the properties of ASP.NET controls. For example, you can use a theme, but not a cascading style sheet, to specify whether or not a GridView control displays a header or footer [11].

Within the Provider Dashboard themes are used to set the properties of the WebPartZones and GridView controls. When the theme color is changed in the Options panel, located on the site.Master page, a different skin can be applied, which changes WebPartZones and GridView controls. A theme or skin can be applied to an individual control or to all similar controls, such as a textbox. In the skin file (located in App_Themes folder) a control's html source code can be placed to specify the properties of that control. The controls within the skin file can not have an "id" associated with them. If you want a skin to be applied only to an individual control a "skinid" must be specified in the skin file. Then the control on the page must specify the "skinid" that it is to associated with in the skin file.

Orientation

The Provider Dashboard provides an option to set the Web page to "Landscape" or "Portrait" mode. Landscape mode displays three zones or columns to have Web Parts. Portrait mode displays two zones or columns. If the page is in landscape mode and then switched to portrait mode the Web Parts that were in the far right zone are moved to the bottom of the left most zone. This was an important feature because this application is going to be run on a tablet PC. When providers are carrying the tablet PC will most likely want to use the portrait mode so they can fit all of the Web Parts vertically on the screen without having to scroll horizontally. However if the provider has the tablet docked they will most likely want to switch to landscape mode to view more Web Parts horizontally.

Master Page

ASP.NET 2.0 introduces a new concept known as Master Pages, in which you create a common base master file that provides a consistent layout for multiple pages in your application. To create a Master Page, you identify common appearance and behavior factors for the pages in your application, and move those to a master page. In the master page, you add placeholders called ContentPlaceHolders where the content (child) pages will insert their custom content. When users request the content pages, ASP.NET merges the output of the content pages with the output of the master page, resulting in a page that combines the master page layout with the output of the content page. As mentioned before the site.master contains the EditorZone and CatalogZone. It also contains the Options panel. This allows us to greatly reduce redundancy of code between these three controls. The EditorZone is placed above the current selected Web Part. This is done through JavaScript (MoveEditorPart) in the source code of the site.master. An example of moving Editor Part can be found at the following web site: http://www.codeproject.com/aspnet/UserControlWebparts.asp

7 Conclusions

Web Parts have been identified as a solution to build a flexible and customizable Healthcare Provider Dashboard. ASP.NET's Web Parts enabled us to build Web pages with layouts in which users can modify the appearance and content to suit their preferences. A key Web Parts feature is personalization, which lets users to save their "user-specific" settings for each page and reuse those settings in future browser sessions. Web Parts can be customized through a custom EditorZone. Along with many other capabilities, Web Parts allow for drag-and-drop, minimize, close, theme setting (color change), and the ability to add multiple instances of a single Web Part.

On Healthcare Provider Dashboard four Web Parts have been created representing possible areas that will be used by a healthcare provider. One of these Web Parts is a flexible Table Viewer, which allows editing in order to be reconfigured in several ways allowing changes of the displayed table, items displayed, width, columns viewed, and popup style that facilitates displaying more details. Three other Web User Controls that we have created include a RSS Feed, a Search Engine, and a Calendar. With these Web

Parts a Provider can easily perform every day tasks like: checking what appointments he/she has today or tomorrow; previous patients visited, hospital rounds list, and the drugs they are able to prescribe and so on. They can also quickly see the latest healthcare news headlines through the RSS Web Part. Also, new Web Parts can easily be created and added to the list of available parts to expand the Provider's Dashboard extensibility.

In conclusion, using ASP.NET's new Web Parts features allowed for the development of a flexible and user friendly Healthcare Provider Dashboard, where the interface becomes dominated by the useful information and not the user controls.

References

[1] "ASP.NET Web Parts Overview" *MSDN*. 2007. Microsoft Corporation. 12 Jan 2007 <http://msdn2.microsoft.com/en-us/library/hhy9ewf1(VS.80).aspx>.

[2] "Walkthrough: Implementing Web Parts Personalization with a User Control" *MSDN*. 2007. Microsoft Corporation. 6 Dec 2006, <http://msdn2.microsoft.com /en-us/library/784d8z92.aspx>.

[3] "Web Site Administration Tool Overview" *MSDN*. 2007. Microsoft Corporation. 6 Dec 2006 <http://msdn2.microsoft.com/en-us/library/yy40ytx0.aspx>.

[4] "Web Site Administration Tool Security Tab" *MSDN*. 2007. Microsoft Corporation. 6 Dec 2006, <http://msdn2.microsoft.com/en-us/library/ ssa0wsyf.aspx>.

[5] "WebPartManager Class" *MSDN*. 2007. Microsoft Corporation. 6 Dec 2006 <http://msdn2.microsoft.com/en-us/library/system.web.ui.webcontrols.webparts. webpart manager.aspx>.

[6] "Web Parts Page Display Modes" *MSDN*. 2007. Microsoft Corporation. <http://msdn2.microsoft.com/en-us/library/f887s4cy.aspx>.

[7] "EditorZone" *ASP.NET Quickstart Tutorials.* 2005. Microsoft Corporation. 13 Dec 2006, <http://quickstarts.asp.net/QuickStartv20/aspnet/doc/ctrlref/webparts /editorzone.aspx>.

[8] "CatalogZone" *ASP.NET Quickstart Tutorials.* 2005. Microsoft Corporation. 13 Dec 2006, <http://quickstarts.asp.net/QuickStartv20/aspnet/doc/ctrlref/webparts /editorzone.aspx>.

[9] Pattison, Ted. "Introducing ASP.NET Web Part Connections" *MSDN*. Feb 2006. Basic Instincts. 13 Dec 2006, <http://msdn.microsoft.com/msdnmag/issues /06/02/BasicInstincts/>.

[10] "Creating a Web User Control in C#". 3 Jan 2005. *CoderSource.NET* 12 Jan 2007 <http://www.codersource.net/asp_net_web_user_controls.html>.

[10] Petricek, Tomas. "ASP.NET Popup Control" 22 April 2004. *The Code Project.* 12 Jan 2007 <http://www.codeproject.com/aspnet/asppopup.asp>.

[11] Walther, Stephen. "Creating Web Application Themes in ASP.NET 2.0" March 2005. *Microsoft Corporation*. 10 Dec 2006 <http://msdn2.microsoft.com/en-us/library/ms379601(VS.8 0).aspx>.