Reconfiguration of Graphical User Interface

Kasi Periyasamy1, Vinoth Perkinian2
1Department of Computer Science
University of Wisconsin-La Crosse
La Crosse, WI, U.S.A.
periyasa.kas2@uwlax.edu
2CareVoyant, Inc.
Schaumburg
IL, U.S.A.
Vinoth.Perkinian@CareVoyant.com

ABSTRACT: The usability and the prime value of a software product increases proportionately with the number of different types of users supported and the number of different types of services provided at any one time by the product. The user interface for the product plays a critical role in this context. In this paper, the authors describe a dynamically reconfigurable graphical user interface for a web-based application. The user interface for this application is designed as a collection of controls. The users will not only be able to view selective information from the user interface, but also will be able to rearrange the controls on the user interface. In addition, users will also be able to introduce new controls and modify existing controls on the user interface without manually recompiling the original code thus achieving greater flexibility and saving time. The paper illustrates this dynamic reconfiguration of user interface through a case study on a health care application.

Categories and Subject Descriptors
H.5.2 [User Interfaces]: Graphical user interfaces (GUI); D.2.9 [Software Management]; Software configuration management

General Terms: User interface, Software management, Software reconfiguration

Keywords: Graphical user interface, Dynamic reconfiguration

Received: 1 June 2011, Revised 4 July 2011, Accepted 13 July 2011

1. Introduction
The advancement in both hardware and software, enabled user interface designers coming up with several different styles of user interfaces1 for software products. User interface of an application, often called the front end of the application, is a prime factor in determining the usability of the product and hence it drives the marketing value of the product. According to Shneiderman and Plaisant [1] “User interfaces help produce business success stories and Wall Street sensations”. In many cases, a poor user interface makes the product not usable even if the product has a rich set of features. As an example, the LaTeX word processing system [2] is well known for its extensive collection of word formatting primitives including a rich set of mathematical symbols. Besides, it is an open source product and is freely available on all platforms. However, there is no proper tool with a good user interface for type setting LaTeX documents. Some editors for LaTeX such as TeXnic Center [3] are available in the market but they provide limited functionalities. Users have to still type a lot of LaTeX commands on their own in order to use the most advanced features of LaTeX. In contrast, most users prefer the WORD package developed by Microsoft [4]. The winning factor here is the simple, easy-to-use, and user friendly interface provided by WORD. It is therefore essential to pay equal attention to the design of a user interface for the product compared to the efforts put on implementing the functional requirements for the same product.

The traditional approach to the design of a graphical user interface (GUI) uses one or more screens that are navigable through a set of controls such as buttons, links and menu items. While this is quite common in desktop applications, there are some inherent difficulties in implementing this feature in a web-based application. This is because, in a web-based application, each window becomes a page. Updating a page on a browser requires the client machine send a request and the page to the server. The server, in turn, updates the entries and resends the page back to the client. If the web-application has several pages, then there will be considerable delay in communication between the client and the server despite the speed or efficiency of the client and/or the server. Technologies such as AJAX [5] have been created to offset some of these communication delays. Yet, many people prefer to have a few pages, possibly one or two, from which all information can be accessed. This, in turn, requires that the space in each page must be efficiently used and should accommodate as much information from the application domain as possible. At the same time, the pages must not be cluttered; otherwise, it will decrease the usability of the product. Therefore, the designers of the GUI for web pages face the challenge and trade-off between the number of pages and information content. It is this challenge that motivated the authors to provide a dynamically reconfigurable user interface so that users will be able to accommodate most information on the same page but they do not need to visualize everything at the same time. In short, most functionalities of the application should be accessible from only a few pages.

How would it be possible to accommodate all required functionalities within a few pages? For example, in a typical health care application, there will be several groups of information for a patient such as demographic information, insurance related information and the patient’s medical history. It is hard to provide all such information in one or two pages unless the user has the freedom to selectively display the group of information he/she wants.

---

1 In this paper, the term user interface refers to graphical user interface only.
To accomplish this, the user should be able to adjust the screen space according to the user's convenience but at the same time should also be able to access all information. One way to achieve this is to divide the information content on the screen into several groups and each group of information is displayed on its own window. Each of these smaller windows is called a control in the rest of the paper. The controls are similar to Google gadgets. The goal of this paper is to describe how user controls can be personalized and so maximum flexibility can be provided to the user.

In order to illustrate the concept of controls and their design, the paper describes the design of a GUI for a web application. The application is the patient portal for a health care information management system. It was developed as part of a joint work between the University of Wisconsin-La Crosse and CareVoyant, Inc. [6]. The product was developed using Microsoft's .NET platform. The design approach exploited the most recently available technologies in the .NET framework and made the user interface dynamically reconfigurable. It thus increased the market value of the product for the company.

The rest of the paper is organized as follows: Section 2 briefly outlines the application domain. The user interface and its features are described in section 3. This is followed by section 4 which includes a detailed discussion on the dynamic reconfiguration of the user interface. The paper includes a brief discussion on continuing work in section 5, which is followed by the conclusion in section 6.

2. Application Domain

The application used as a case study in this paper is a health care information management system developed by CareVoyant, Inc. [6]. This company provides software solutions for health care providers by extensive consolidation of clinical, financial, point of care and business intelligence functions of the health care domain. Generally, big enterprises such as Nursing home, Home care, and Multi-disciplinary physician offices that provide health care support to their patients. They may have one or more companies (called Facilities) physically located in different places across the world. Therefore, managing patients' health care information as supported by the companies belonging to a particular enterprise is a tedious task. It is at this situation, CareVoyant comes into picture. Typically, CareVoyant takes care of all health care management information of all the patients belonging to an enterprise, even if they are located in different places. A detailed discussion of the operations of CareVoyant is beyond the scope of this paper.

One of the services provided by CareVoyant is the Patient Portal (the word 'system' is also used interchangeably throughout the paper to refer to the patient portal). The users of the Patient Portal belong to one of the three categories: patients whose health care information is maintained by the system, administrators who are responsible for adding, updating and maintaining the health care information and guarantors who are financially responsible for patient health care activities such as paying for clinical visits, medications, etc. The contents of information displayed on the Patient Portal may consist of demographic, clinical and financial information. Because of the number and different groups of people accessing the patient portal, and the nature of information accessed by each group, the user interface for the patient portal was expected to be easily navigable, and at the same time, to provide all information in a concise manner. The functionalities that can be invoked by the three types of users are listed in Table 1:

<table>
<thead>
<tr>
<th>Administrator</th>
<th>Patient</th>
<th>Guarantor</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Application configuration</td>
<td>- User registration</td>
<td>- All functionalities a patient can do</td>
</tr>
<tr>
<td>- Database configuration</td>
<td>- Homepage customization</td>
<td>PLUS</td>
</tr>
<tr>
<td>- User administration</td>
<td>- Add/remove controls</td>
<td>- Lock/unlock patient screen</td>
</tr>
<tr>
<td>- User access rights setting</td>
<td>- View/modify clinical and demographic information</td>
<td>- Patient access level settings</td>
</tr>
<tr>
<td>- Application customization</td>
<td>- Make clinical requests</td>
<td>- Patient approval</td>
</tr>
<tr>
<td>- Control management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Registration approval</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Functional requirements for different groups of users

Table 1 indicates only a high level description of the functionalities. In the implementation of the Patient Portal, there is a corresponding page of information for every one of the functionalities listed in Table 1. In addition, the table also gives a hint on the complexity of relationships between these functionalities. For example, a patient needs to register with the system before start accessing any information (indicated by 'User registration') which must be approved by his/her guarantor (indicated by 'Patient approval') and finally must also be approved by an administrator.

The patient portal uses three different databases: The first one is called 'System Database' that includes the information pertaining to a particular enterprise such as application settings and company information. The second one is called 'Company Database' which includes information such as patient demographic information and medical records of the patients belonging to all the companies (facilities) of an enterprise. The last one is a third party database that is mainly included to provide additional information on the portal when needed. Typically, information about drugs, their interactions and drug education is provided by a third-party.

The relationships between entities across databases provide a challenge in the design of the user interface because a user may want to view on a single display control (to be discussed in the in the next section) with entities of information that come from different databases. For example, a user may want to view his/her medication information (that is extracted from the patient’s medical history) on a display control along with the information on the medication (that comes from the third-party database). When this display control is newly created (as discussed in subsequent sections) on-the-fly, the displayed information must be consistent which, in turn, requires that the two different databases must be accessible at the same time.

3. User Interface For Patient Portal

This section describes the design information for the Patient Portal User Interface. The developers used the approach similar to Google gadgets such that the user interface consists of several smaller components (called 'controls in this paper) that can be individually manipulated. For illustration, consider the home page for a guarantor which is shown in Figure 1. There are three divisions in the home page - the top part contains the logo and company information, the left panel consists of a series of menu items that can be invoked by the logged in user, and the rest of the page contains a series of independent rectangles. Each of these rectangles is called a 'control' which displays a specific piece of information. For example, the top control in the guarantor home page is called the 'My Patients' which lists the details of all patients for whom the logged in guarantor is responsible.

---
Figure 2 shows an expanded view of the 'My Patients' control, and Figure 3 shows an expanded view of the 'Demographic Information' control. A number of flexibilities have been implemented in manipulating these controls:

- Each control is resizable and hence the user has the freedom to have an expanded or shrunken view of the information displayed by the control.
- Each control can be moved within the display area (leaving the banner and the left panel untouched). This will automatically reposition other controls on the window so that information displayed on the other controls will not be lost.
- Each control can be removed / added to his/her homepage based on the individual’s preference. The user has the freedom to view and change information they are interested to see in their homepage.

Thus, the basic structure of the GUI itself is designed to provide maximum flexibility to the users. Such flexibility deemed to be important for the users of this application, especially when they want to focus on the information displayed by a fewer controls only.

The Patient Portal was implemented using the advanced technologies provided by ASP.NET framework and C# programming language, and by a third-party tool called Telerik Radcontrols for ASP.NET [7]. While the development of the portal could have been achieved by using the ASP.NET framework alone, the Telerik controls with their AJAX support made the programming easier and manageable. Besides, the developers considered these controls as some sort of design patterns for the web-based applications and hence reusing these controls for another project would also be easier.

4. Dynamic Reconfiguration

The controls discussed in the previous section are called built-in controls (controls shipped with the product). These controls are read from the application source when the system is started. Although the user can shrink / hide / close some of these controls in order to make room for other controls, they cannot be modified or deleted permanently from the application by the users. Any changes to be made on the built-in controls require changes in the code for the application, and hence these controls are considered to be built into the application. Most of such built-in controls are decided by the developers of this System. In addition to these built-in controls, the system provides user definable controls. They can be introduced into the system in three different ways:

Simple controls: These controls can be newly introduced into the application during run-time. Templates for creating such controls will be provided by the developers of this System and will be utilized by Administrators of the System. Once such simple controls are introduced, they will be available to all users of the
However, an administrator can remove these controls from the application permanently at any time, so that it will be no longer available to any user. The main purpose of a simple control is to extract and to display some additional information from the databases which is not included in any of the built-in controls. For example, the demographic information of a patient may include another contact address which can be displayed separately using a simple control, when needed. An important characteristic of a simple custom control is that it is built at run-time but will be destroyed automatically when the system terminates. So, users will not be able to save simple custom controls.

For a simple control, the administrator defines the data object (i.e., the stored procedure) associated with the control and stores it as the control's Meta data in the database. The algorithm for the creation of a simple control is shown below:

```csharp
Create Control (Control_Name, Control_Title, Data_Source) {
    Set<Control>X (Set Control_Name, Set Control_Title, Set Data_Source)
    If <X not in Database> Add X
    Else display "Control Already Exists"
}
```

The name of the control must be unique. Notice that the addition of a simple control does not mean that the control will be permanently added to the system. It will be deleted when the system terminates.

To add the control to the user interface, a .NET User Control is created with a data grid object; the columns of the data grid are generated based on the stored procedure associated with the control. Since the template for a simple control is defined already, it is easier to define the space for the new control. Finally, the simple control is placed inside a dock which acts like a placeholder for the control. The algorithm for adding the control is given below:

```csharp
Add Control (Control_Name) {
    Control X = Retrieve Control (Control_Name)
    If <X not in Database> Add X
    Else display "Control Already Exists"
}
```

Custom built-in controls: A control in this category can be built using the control creation functionality embedded in the application. Unlike a simple custom control, the user has the freedom to design his/her style of the control. Typically, the control creation functionality will let the user select pieces of information from the databases and the order in which these values are to be displayed. The major difference between a custom-built control and a simple control is that the former can be saved along with the application source and can be re-loaded at the user's convenience without the need to dynamically re-create it every time when the system is started. Like a simple control, a custom built-in control can also be deleted by an administrator at any time.

The custom control management screen is shown in Figure 4. The user must provide a unique control name, title for the new control and a data source. The data source represents the set of fields from the database that the user wants to display through the custom control. Currently, ensuring the uniqueness of control name is left to the user. This is because the user may want to keep track of the custom controls created by him/her, and is not required to know the names of other controls in the system. The DataView provides different ways of ordering the information on the control. Currently, it supports only the GridView which enables users view the information on a table-like structure. Work is in progress to provide multiple data views.

The creation algorithm for custom control is given below:

```csharp
Create Custom Control (Control_Name, Control_Title, Data_Source) {
    Set<Control>X (Set Control_Name, Set Control_Title, Set Data_Source)
    If <X not in Built-in_Controls> {
        Create <.NET Data Grid> Data_Grid
        Read (X.Data_Source)
        Create <.NET User Control File> UCFile
        Set UCFile = Read Template
        UCFile.Write (Data_Grid.SourceCode)
    }
    Else display "Control Already Exists in Homepage"
}
```

To add the control to the user interface, a .NET User Control is created with a data grid object; the columns of the data grid are generated based on the stored procedure associated with the control. Since the template for a simple control is defined already, it is easier to define the space for the new control. Finally, the simple control is placed inside a dock which acts like a placeholder for the control. The algorithm for adding the control is given below:

```csharp
Add New Custom Control {
    Control X = Retrieve Control (Control_Name)
    If <X not in Homepage> {
        Create <.NET User Control> Temp
        Control_Dock.Add (Temp)
        Control_Dock.Add (Temp)
    }
    Else display "Control Already Exists in Homepage"
}
```
The algorithm to add the custom control is similar to that of adding a simple control and is defined as follows:

Add Control (UCFile) {
    Read Database (Control_Name for UCFile)
    If (<Control_Name not in Database> {
        Create <.NET Dock> Control_Dock
        Control_Dock.Add (UCFile)
        Homepage.Add (Control_Dock)
    } Else display “Control Already Exists in Homepage”
}

Imported controls: The patient portal system is designed to be so flexible that it not only allows a user to create and save his/her own control, but it also allows importing controls from another system. Realizing the benefits of custom-tailored controls, the developers of this system decided to introduce some additional controls which can be imported by a user. This will be more useful if a user does not want to mess up with the controls or the GUI. The developers can take care of the request for customized control based on the needs of their clients. After importing, a user can save the control and include it on the GUI whenever needed. Currently, users cannot modify an imported control. However, an imported control can be deleted by an administrator. Figure 5 shows the screenshot for imported control management.

Like the other two types of controls, each imported control comes with a unique ID but this is assigned by the developers of this system. Each control requires an ASCX file and a C# file. The ASCX file includes the display code for the control while the business logic for the same control is included in the C# file. Proper naming conventions are used to identify the pairs of the ASCX file and the corresponding C# file so that users can easily track of these controls. While importing, a user must select these files as shown in Figure 5. Current implementation does not verify the relationship between these files. Instead, the same file prefix is used for both types of files so that users can easily identify the type of files.

Following is the algorithm to create an imported control.

Import Control (Control_Name, Control_Title, UCFile_Path) {
    Read UCFile (UCFile_Path)
    Set UCFile.Name = Control_Name
    Set UCFile.Title = Control_Title
    If <UCFile not in Built-in_Controls> {
        UCFile.Save (Built-in_Controls)
        Update Database (Control_Name for UCFile)
    } Else display “Control File Already Exists in Built-in Controls”
}

The algorithm to add an import control to the database follows:

Add Control (UCFile) {
    Read Database (Control_Name for UCFile)
    If <Control_Name not in Homepage> {
        Create <.NET Dock> Control_Dock
        Control_Dock.Add (UCFile)
        Homepage.Add (Control_Dock)
    } Else display “Control Already Exists in Homepage”
}

A summary of the features supported by the three different types of controls is given in Table 2.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Simple Control</th>
<th>Custom Built-in Control</th>
<th>Imported Built-in Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available for reuse in other CareVoyant applications</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Sorting</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Column reordering</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Dynamically added to the application</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Additional data views support</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Additional data objects support</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Incorporating business logic</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Occupies resources only on demand/usage</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Collapse</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Moving controls anywhere on the screen</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Dynamically adjusting screen size</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Allow property change (e.g., Title, Skin etc.)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Allow data object changes</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Table 2. Features supported by the user defined controls

5. Continuing Work

The current implementation of the patient portal has several limitations. First of all, the system is designed as a Microsoft application taking extensive advantages of the Microsoft...
Realizing the benefits of the reconfiguration technique, CareVoyant, the sponsor of this project, has developed a tool called ‘CareVoyant Modifier’ that allows dynamic changes to any GUI (web or standalone applications) developed using Microsoft's Windows Presentation Foundation (WPF). Using this modifier, a GUI designer can reposition a control on a GUI, change its size, remove or disable/enable the control without changing the code. The dynamic reconfiguration is achieved by taking the WPF schema which uses the Extensive Application Markup Language (XAML) to define and link various user interface elements to generate the modified user interface. Figure 6 shows a flow chart describing how the modifier works.

The major advantage of the modifier includes end user customization, and the ability to create different versions of the same user interface for different users. The development of this tool is a significant step forward in this direction because it is now possible that any user interface that has been developed using WPF can be dynamically reconfigured without recompiling the code.

The basic approach used in this work is dynamic recompilation of source code. This technique was also exploited and used in other applications. For example, He and Periyasamy [8] used this approach in designing an object-oriented database system in which user defined objects can be added to the database dynamically. The object-oriented database was implemented in Java. Wilson [9] used a similar approach using C# in dynamically introducing user defined keywords in test scripts. He built an automated test case generator that allows users to introduce new test scripts that are dynamically compiled and linked to existing test cases. Sain and others [10] has developed context-aware middleware architecture for a personal healthcare application. Though, from application point of view, the product is somewhat similar to the one described in this paper, the two approaches are quite different.

6. Conclusion

Dynamic compilation is a facility that enables users to compile part or whole of a source code while the original code is executed. Recent versions of both Java and C# provide the feature of dynamic compilation. This feature has been exploited in three different applications by one of these authors. In this paper, the authors have applied this notion to reconfigure a user interface by the end users themselves without assisted by the developers of the original code. The paper presents a novel, user interface design architecture. The user interface is designed with the goal to minimize navigations to other pages in a web application. This is done through plugged-in components called ‘controls’ which can be minimized, resized or repositioned by the users. The end result is a customized user interface, controlled by the client. In addition, the design architecture also allows users to create their own controls, save them and load them at their convenience, and also allows them to import other built-in controls from another system. This provides the maximum flexibility in customization of user interface.

References