Acta Universitaria
DIRECCIÓN DE INVESTIGACIÓN Y POSGRADO

A Method to Ease the Deployment of Web Applications that Involve Database Systems

Sergio Ledesma*, Donato Hernández Fusilier*, Antonio Vega Corona* y Juan Gabriel Cervantes Aviña*.

ABSTRACT

The continuous growth of the Internet has driven people, all around the globe, to perform transactions on-line, search information or navigate using a browser. As more people feel comfortable using a Web browser, more software companies are trying to alternatively offer Web interfaces to provide access to their applications. The consequent nature of the Web connection and the restrictions imposed by the available bandwidth make the successful integration of Web applications and database systems critical. Because popular database applications provide a user interface to edit and maintain the information in the database and because each column in the database table maps to a graphic user interface control, the deployment of these applications can be time consuming; appropriate field validation and referential integrity rules must be observed. Thus, an object-oriented approach is proposed to ease the development of applications that involve database systems.

RESUMEN

El crecimiento continuo de la Internet ha permitido a las personas, alrededor de todo mundo, realizar transacciones en línea, buscar información o navegar usando el explorador de la Web. A medida que más gente se siente cómoda usando los exploradores de Web, más empresas productoras de software tratan de ofrecer interfaces Web como una forma alternativa para proporcionar acceso a sus aplicaciones. La naturaleza de la conexión Web y las restricciones impuestas por el ancho de banda disponible, hacen la integración de aplicaciones Web y los sistemas de bases de datos críticas. Debido a que las aplicaciones que usan bases de datos proporcionan una interfase gráfica para editar la información en la base de datos y debido a que cada columna en una tabla de una base de datos corresponde a un control en una interfase gráfica, el desarrollo de estas aplicaciones puede consumir un tiempo considerable, ya que la validación de campos y reglas de integridad referencial deben ser respetadas. Se propone un diseño orientado a objetos para así facilitar el desarrollo de aplicaciones que usan sistemas de bases de datos.

Recibido: 29 de Junio de 2006
Aceptado: 26 de Febrero de 2007

INTRODUCTION

The *World Wide Web* (WWW) is a distributed system of computers spread all over the Internet. Its enormous popularity stems from the fact that it has a graphical user-friendly interface (Tanenbaum, 2002). To warranty competitiveness, corporations have tried to slightly enhance the Web browser by adding non-standard functionality, causing a nightmare for Web developers during the last decade.

The *WWW* provides an enormous means of information on roughly all subjects through linked documents, search engine services, and transaction services. All these services are known as resources. To easily find and use these services, *Uniform Resource Locators* (URL) were conveniently defined. From a user perspective, an URL is a simple text string that is typed at the top of the Web browser in the address field. This string specifies the exact location of an Internet resource.

Keywords:
Graphic user interface; Integration; Database; Web application deployment; SQL.

Palabras clave:
Interfase gráfica del usuario; Integración; Base de datos; Desarrollo de aplicaciones Web; SQL.

* Facultad de Ingeniería Mecánica Eléctrica y Electrónica (FIMEE) de la Universidad de Guanajuato. Tampico #912. Col. Bellavista. C. P. 36730. Salamanca, Guanajuato. Correos electrónicos: selo@salamanca.ugto.mx; donato@salamanca.ugto.mx; tono@salamanca.ugto.mx y avina@salamanca.ugto.mx.

Web pages are written in a special format known as Hyper Text Markup Language (HTML); see the HTTP and HTML home pages in the Internet. Because Web browsers are capable of displaying HTML documents, they transform the raw data sent from a Web server to graphical information in the user's display. Unfortunately, the HTML specification establishes a very simple structure, making very difficult to deploy Web applications. Several vendors have created several Web technologies to simplify Web application deployment. Unfortunately, Web programming is not easy. The proposed method targets this problem, simplifying the deployment of Web applications through the use of the **IStorable** interface that will be discussed in detail later.

In recent years, the cutting edge of software development has shifted from traditional "fat clients" apps to Web applications. The integration of back-end systems and seamless data sharing, once the holy grail of corporate IT departments, have given way to concerns over lower total cost of ownership, zero-footprint installs, and the ability to run applications from anywhere an Internet connection is available (Prosise, 2002). Web applications like eBay or Amazon.com are complex Web applications that involve database systems. The deployment of this kind of applications can be time consuming because current Web technologies do not offer a simple integration between the user interface and the database system. Generally, they offer some graphical tools or proprietary Web controls to simplify this process. The proposed method provides an Object-Oriented programming solution to make simpler the deployment of Web applications.

The Web server

Web pages can be static or dynamic. They are almost always stored in a Web server, although they can be stored and edited in a personal computer. Static Web pages are text files that are written using the HTML language; a HTML editor or a simple text editor can be used to write these pages. In general, static pages are not useful to deploy Web applications because their content does not change unless the HTML file is modified. Pages with dynamic-content are HTML pages that are typically written or modify in the fly; that is they are written when the user requests the specified URL. Most programming languages can be used to create dynamic Web pages. Some popular languages are C/C++, Java, Perl, C#, PHP, Visual Basic (VB) and ColdFusion. Each language is associated with a specific Web technology; i.e., C# is used on ASP.NET technology, and Java is used on Java Servlet technology (Liang, 2001).

It would be impossible to talk about the Web without mentioning the Extensible Markup Language, XML. In a few short years, XML has grown from an obscure specification into the world's de facto data language. Whereas HTML is designed to express appearance, XML is designed to express raw information absent any implied notion about how the data should be rendered (Prosise, 2002). It is a simple language that is entirely text based, and has no predefined tags as HTML does. XML documents can be converted into HTML on the fly using Extensible Stylesheet Language Transformations, XSLT, impacting considerably the deployment of Web applications.

Basically, the Web server technology and the programming language determine the type of database access to be used. Most programming languages support *Open Data Base Connectivity* (ODBC). ODBC is a widely accepted application programming interface (API) for database access. Most vendors support the ODBC 3.0 API in addition to their own native SQL APIs. A driver is a translation code that offers integration between two modules. In this case, the ODBC driver is one that accepts a call and translates it into the native database language.

The Java language supports *Java Data Base Connectivity* (JDBC) technology. It is an API that provides cross-DBMS connectivity to a wide range of SQL databases; see the JDBC page at http://java.sun.com/products/jdbc/ (Hall and Brown, 2001).

OLE DB is a data access technology that originated in the zenith of COM. MSDASQL was a generic solution that permitted databases without an OLE DB provider of their own but that had ODBC drivers available to be accessed using the OLE DB API. ADO.NET is an easy database API of the .NET Framework. Other database systems, like MySQL, Oracle, dBase, Microsoft Excel, Microsoft Access, FoxPro, etc., provide their own drivers, or these can usually be downloaded from the Internet.

Object-Oriented Programming

For dynamic-content Web pages, a computer program writes a Web document each time a user requests the specified page (Felton, 1997). Programs are built using computer instructions. Traditionally, a program is a list of computer instructions that execute in sequence; this list of instructions is called code. Unfortunately, for large applications a list of instructions is difficult to read and maintain. In the last years, many programmers have been using *Object-Oriented*

Programming (OOP) instead of a list of instructions, because OOP allows them to easily read and maintain large programs (see Abiteboul and Beeri, 1995).

OOP organizes the program in a comprehensive set of individual units. Units are called objects and can directly interact with each other. OOP instantly brings the domain of real-world applications to the programming domain; i.e., a management school system may be composed of student, faculty and classroom objects. Software objects live in the programming domain, and are used to model objects from the real-world domain. An object has variables (properties) and methods; i.e., an object of type **student** has an **expected graduation date** property and a **learn()** method. Objects can interact with each other through their methods. All objects are purposely created, used and utterly destroyed. A class is an object specification; i.e., the **student** class specifies that all **student** objects have an **expected graduation date**, and all of them learn. A class is a technical blueprint or prototype that clearly defines the variables and the methods common to all objects of a particular kind (see the on-line Java Tutorial). Several objects of the same class can be created; i.e., a University may use the same class to create several objects of type student to fill a classroom, each student has an **expected graduation date** even though each student has a particular **expected graduation date** value.

To guarantee reusability, a base class should be as generic as possible. Reusability is the probability of using a class more than once with slight or no modification. Because implementing a method may expressly restrict the generic behavior of a base class, sometimes it is appropriate to have a class that declares a method without implementing it. These are known as abstract classes, and cannot be used directly to build objects. Abstract classes represent abstract concepts, and they are used through derivation to create classes that are not abstract; i.e., an **abstract box** class may specify that a box can be open without specifying expressly how to open it; that is, all derived classes from the **box** class need to implement the **open()** method to become non-abstract; i.e., a cardboard box may be open with a different procedure (method implementation) than an aluminum box, both of them are boxes and can be opened but they are open with a different technique.

Finally, an interface is a formal agreement of services by providing a comprehensive set of methods and constant declarations. A class may implement an interface by implementing all the methods declared in that interface. A class may implement zero, one or more interfaces as required. A class that implements a contractual interface is able to communicate with objects that required that interface. In this paper, the **IStorable** interface will be proposed as a bilateral contract of service between a Web element and a master data provider called **Storer**. The **IStorable** interface is used to integrate the displaying behavior of a Web control with its data editing facility. For example an edit box may implement the **IStorable** interface, if it wants to participate on a database transaction. Thus, the edit box control is responsible of validation and the **IStorable** interface allows this control to load data from the database as well as update the database with very little additional code.

UML Notation

The Object Management Group (OMG) has created the technical specification for the Unified Modeling Language (UML), see (Wieringa, 1998). This technical specification can be retrieved using the URL http://www.uml.org. OMG is a not-for-profit computer industry specifications consortium. UML is used to model the structure and functionality of a system (Roques, 2004 and Holt, 2004). UML notation will be used to describe the integration process between web applications and database systems.

PROPOSED METHOD

To illustrate the significant interest of the integration between Web interfaces and database systems, two commercial products will be briefly discussed. The first family of products to be discussed is *PeopleSoft* Enterprise applications; they are designed for meeting complex business requirements (see the *PeopleSoft* web site). These products kindly provide web services integration with multi-vendor and homegrown applications and can be easily configured and adapted to meet the customer requirements. *PeopleSoft* products were originally designed as Desktop applications and have been on the market for several years. To access the *PeopleSoft* database, users needed to install the client software. For big corporations, the Information Technology (IT) team had to typically spend several hours installing and maintaining the client program in all their users' computers; this installation process is time and money consuming. Some years ago, *PeopleSoft* created a Web interface to access its database, eliminating the need of the client software. Despite the fact that the *PeopleSoft* Web interface may be slower than the traditional Desktop client, it provides direct access from any computer to the*PeopleSoft* database with no special software installed.

Another good example is *Microsoft Outlook*, the popular tool that comes with *Microsoft Internet Explorer* to read and send e-mail. *Microsoft Corporation* created *Outlook Web Access* (OWA) as part of *Microsoft Exchange* (Microsoft Mail Server.) OWA provides e-mail services using the Web browser, removing completely the need to install any e-mail client software in the user's computer. Finally, note that the performance of these two products depends directly on the technology that brings together the Graphic User Interface (GUI) and the database (see Barga *et al.,* 2004 and Conn, 2002).

OOP description of Web elements

Figure 1 shows the UML diagram of some of the most common Web elements. Consider that an appropriate namespace, called **Web**, has been clearly defined and that all the classes in Figure 1 are inside this namespace. Starting at the root of the diagram, the Web **Object** is the most generic class to describe Web elements. The **Object** class is abstract because the **getTagName()** method is abstract; this basically implies that a derived non-abstract class must implement this method. The **getTagName()** method allows the object itself to retrieve the name of the respective HTML tag for rendering purposes, i.e., html, div, span, table, form, hr, etc. As it is clearly shown in the UML diagram of Figure 1, any Web object is capable of rendering through its **render()** method, which uses the HTML format to successfully write the appropriate data utilizing the static Server property, see Figure 4 (b), of a Site object. Usually, the Web site itself calls the **render()** method of the main Web page each time a user's request arrives to the Web server.

From left to right, the second column of Figure 1 shows some of the simplest Web elements: **Br**, **Hr** and **Page**. The br and hr tags do not have a consequent closing tag; the Br object renders as <br />, and is used to create a new line by stopping more Web elements to display on the current line. Additionally, **Br** objects can be used to increase the space between two consecutive paragraphs. **Br** and **Hr** are non-abstract classes because they properly implement the **getTagName()** method, which returns the text string "br" for the **Br** class, and "hr" for the **Hr** class. The **Hr** class allows creating a horizontal separator in a Web page. Essentially, these two basic classes will help understanding more complex classes, such as **Node** and **Input**, which will be used to seamlessly integrate Web elements with data providers. Note that a Web tag that does not have a consequent closing tag, may optionally follow the Extensible Markup Language (XML) notation; i.e., use <br /> instead of <br> or <hr /> instead or <hr> (see Rusty, 2003).

Each Web page has a head variable and a unique body as it can be seen from Figure 1. The **Body** and **Head** classes are derived directly from the abstract class **Node**. The **render()** method of the Page class is called by the Web site; in turn, each Web page calls the **render()** methods of its head and its body. Surprisingly, the **getTagName()** method of the **Page** class must not return "page" but the text string "html", which is the standard HTML tag for a page.

Derived directly from the **Object** class, there are two abstract classes: **Element** and **DoubleTagElement**. The **Element** class has a comprehensive set of properties to control outward appearance through the use of Cascade Style Sheets (CSS) and programmed behavior by using Scripts. The **DoubleTagElement** is used to describe simple tags that require a consequent closing tag; i.e., title, option, script, etc. The classes Title, **Option** and **Script** are non-abstract classes because they implement the abstract method **getTagName()** officially declared in the base class.

In general, Web pages are built following a hierarchy structure, that is, a Web element may have children and its children may have children as well. A good example of this hierarchy structure is a Web table (which is represented by the table tag). Tables have one, two or more rows, and each row has at least one column. This tree-structure of Web pages is described through the abstract **Node** class of Figure 1. The **Node** class has an array of Web **Objects** to represent its children. A **Node** object may have zero or more children; i.e., a Web page has two children, head and body. When a **Node** object is rendered, the Node object itself calls the **render()** method of its children. Most popular Web elements are directly derived from the Node class as it can be seen from Figure 1.

The abstract **Input** class is one of the most interesting classes when deploying Web applications. This class represents the input HTML tag and is useful for user input. Most Web applications frequently use the input tag to appropriately collect information from the user. Generally, the **Input** class symbolizes the abstract concept of user input interface without restricting the fundamental nature of the input. To successfully create a non-abstract class from the base Input class, the derived class must implement the **getInputType()** method to specify its input type: text, radio, checkbox, submit, button or password. Figure 1 shows the UML diagram of the most popular Web controls. Note that this diagram conveniently includes complex controls

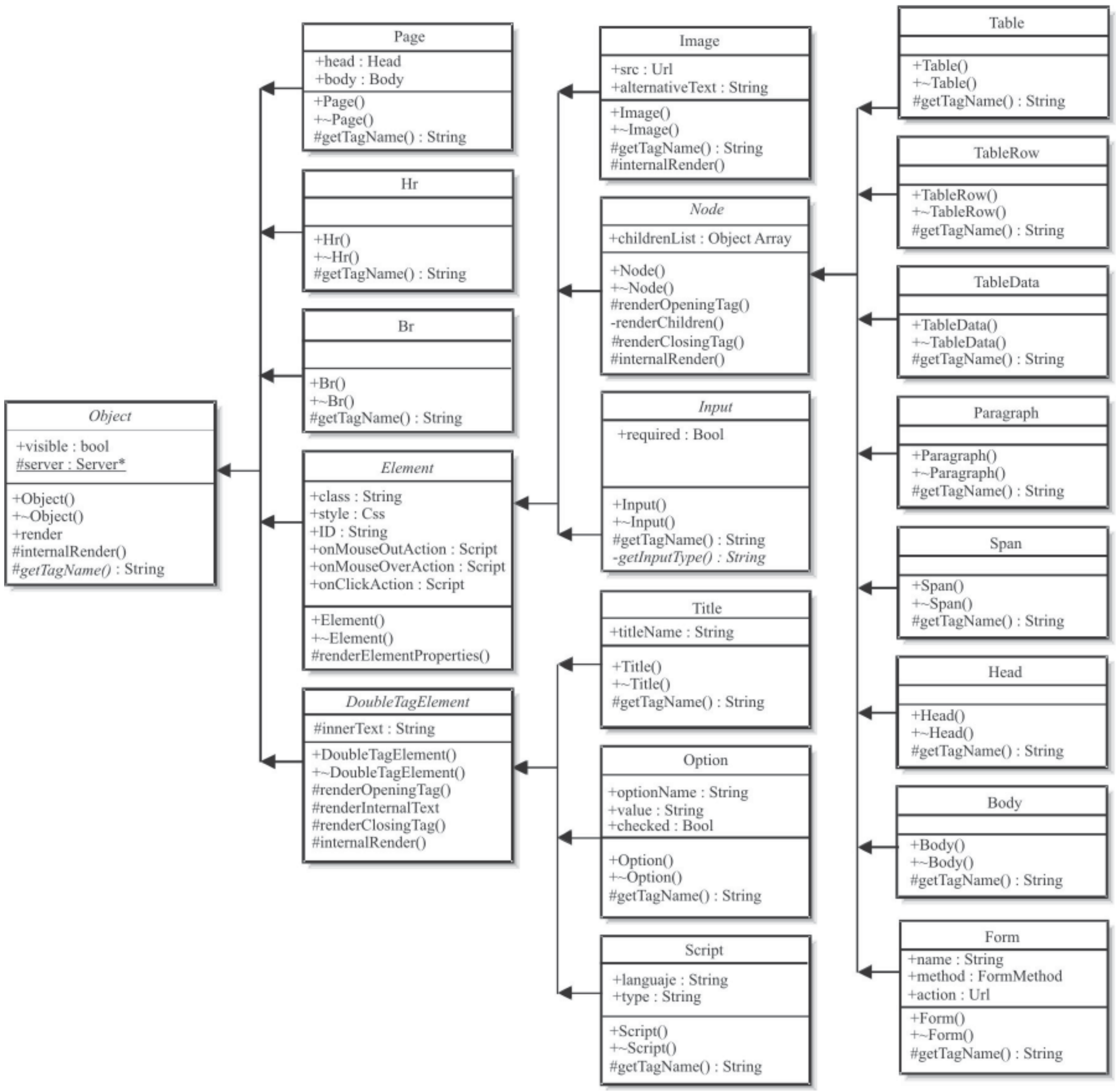


Figure 1. UML diagram of some typical Web objects.

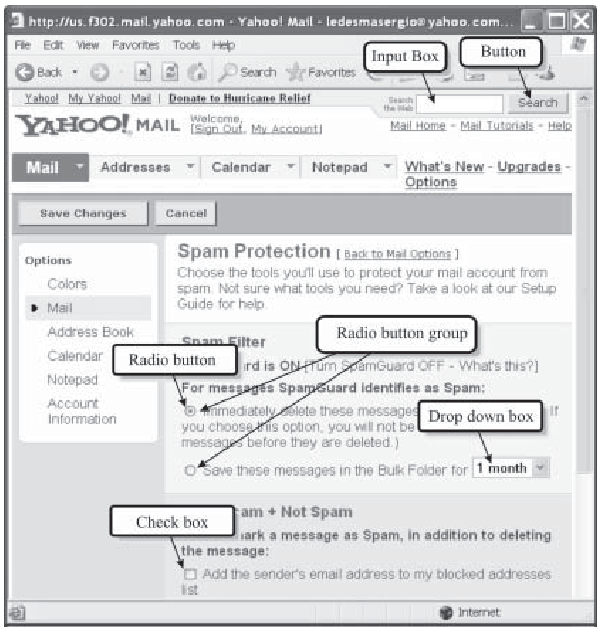
like the drop down box and the radio button group instead of the raw controls: combo box option and radio button. The adequately use of complex controls instead of the simple ones can greatly reduce the code when integrating Web interfaces with database systems.

The **Input** class has only one property, **required**, which is used for non-null values. This property is very important because an Input object is responsible for checking if a required value is not present. The object may optionally display a message informing the user or notify other Web or data objects of this event.

OOP description of Web form elements

In general, dynamic Web pages require the use of Web forms to collect information using a Web browser. These forms are composed of individual Web controls. Unfortunately, the deployment of Web or Windows forms can be time consuming; each control in the form must usually be synchronized before opening the form with information a database. When the user clicks on the OK button of the form to accept the transaction, the program must validate and collect data from all controls and update appropriately the database. The deployment of this kind of forms can be time consuming because typical database applications have several tables and these tables may have several columns (see Liu, 1999), the proposed method reduces this time by making typical form controls to implement the **IStorable** interface as it will be described next.

Figure 2 shows the Yahoo spam protection page; this page was selected because it clearly shows the most popular Web controls. At the top of Figure 2 the most popular Web controls, the **Input Box** and **Button** controls, are shown. The **Input Box** provides a two-way communication between the user and a Web browser; some text may optionally be initially displayed and utterly modified. The **Button** control is used to execute actions; i.e. in the Yahoo page shown in Figure 2, the button at the top initiates a Web search.


http://us.f302.mail.yahoo.com - Yahoo! Mail - ledesmasergio@yahoo.com...
File Edit View Favorites Tools Help
Back Search Favorites
Input Box
Button
Yahoo! My Yahoo! Mail | Donate to Hurricane Relief
Search the Web
Search
YAHOO! MAIL Welcome, [Sign Out, My Account]
Mail Home - Mail Tutorials - Help
Mail Addresses Calendar Notepad What's New - Upgrades - Options
Save Changes Cancel
Options
Colors
Mail
Address Book
Calendar
Notepad
Account Information
Spam Protection [Back to Mail Options]
Radio button group
Radio button
Spam Filter
Drop down box
1 month
Check box
Internet

Choose the tools you'll use to protect your mail account from spam. Not sure what tools you need? Take a look at our Setup Guide for help.

...rd is ON [Turn SpamGuard OFF - What's this?]

For messages SpamGuard identifies as Spam:

Immediately delete these message... If you choose this option, you will not be... messages before they are deleted.)

Save these messages in the Bulk Folder for 1 month

...am + Not Spam

...ark a message as Spam, in addition to deleting the message:

Add the sender's email address to my blocked addresses list

Figure 2. The Yahoo spam protection page showing most popular Web controls.

Figure 2 shows other frequently used Web controls, the radio button, the check box and the drop down box (also known as combo box). The radio button control receives its name from the old car radios that used mechanical buttons to store radio stations; i.e., only one button could be pushed at the same time. Radio buttons allow a user to choose one alternative from a given set of options; i.e. a Web trip planner offers the feature of selecting the prefer departure day: Monday, Tuesday, Wednesday, etc. The combo box provides the same functionally as a group of radio buttons. However, the combo box takes less space at the price of hiding all available options; to display this set options the user must click the combo box control with a mouse. The check box control is used for yes/no questions; i.e. a Web trip planner may have a check box to indicate whether or not the traveler would like to have his meals included. In the Yahoo span protection page, Figure 2, the radio button group is used to specify what action should be automatically taken when a spam message arrives. In the same page, there is one check box at the bottom of the page; it asks whether or not the user would like to automatically add the sender's e-mail address to his blocked address list. There are other Web controls that are not discussed in this paper because they are not relevant to the deployment of database Web applications.

The **InputBox** control is an input text control and its UML diagram is shown in Figure 3. All **InputBox** objects have three properties: **value**, **readOnly** and **regularExpression**. The **value** property represents the actual text or value stored in the control, the **readOnly** property disables the control (preventing text changes), and the **regularExpression** property allows data checking through the use of **regular expressions**. The class implementation of the **RadioButtonGroup** and the **DropDownBox** is pretty similar; both classes have a **selectedIndex** property to specify the index of the option that is selected. For most popular database applications, the drop down box and a radio button group map directly into a lookup table; i.e., a database table stores the letters 'I' and 'D' instead of storing the text International and Domestic. A look up table provides full text description for each letter in this table. The **CheckBox** class is used to manage Boolean values and usually maps into a yes/no field (known as bit or bool) in a database system.

The major problem when deploying Web applications that involve database systems is proper integration. Most database systems contain data tables that are organized following the normal forms of the relational databases (see Arenas and Libkin, 2004); however, tables and table columns must map to GUI

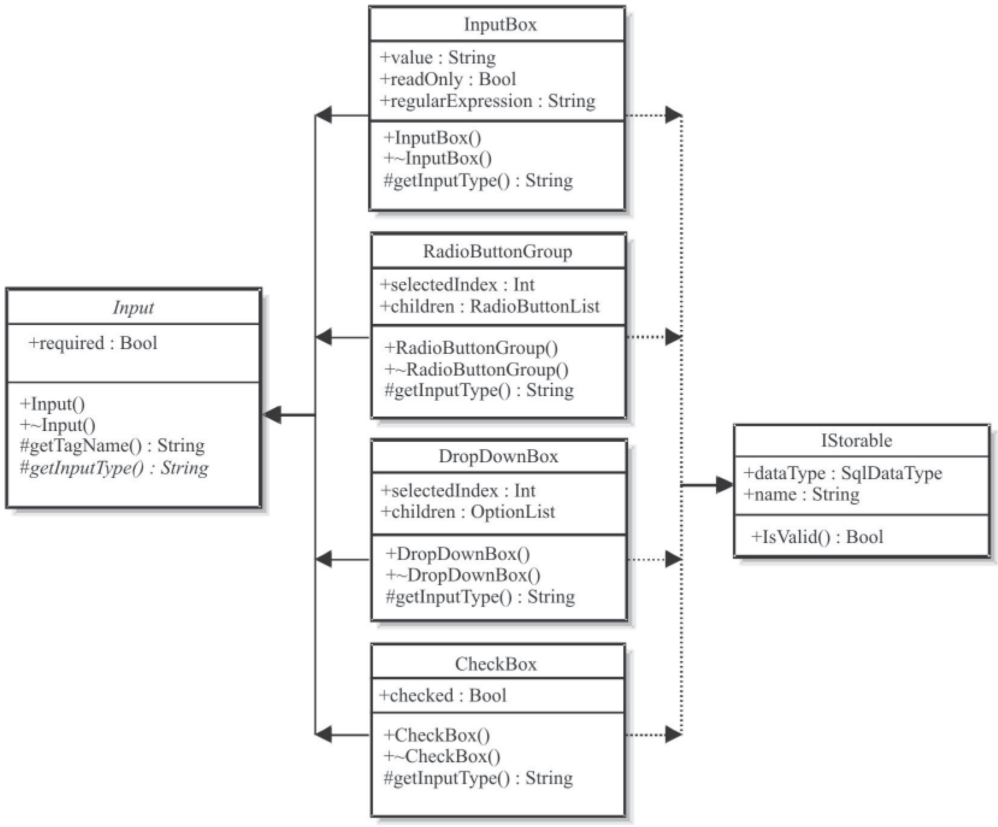


Figure 3. The UML diagram of most common Web controls.

controls making the deployment of web applications difficult. While the Web controls, input box, combo box, radio button and check box, require a simple HTML tag to be properly created, they are not designed to perform complex tasks. To facilitate the successful integration of these Web controls with a database system, the interface **IStorable** is proposed. Figure 3 shows the UML diagram of the **IStorable interface**. This interface provides a direct link between the database system and a Web form. **IStorable** is pretty simple and has only two properties and one method. The **dataType** property is used to expressly indicate the data type stored by the control. For example, a check box control will have a Boolean data type; that is the only valid values are true and false (yes or no). The second property of the **IStorable** interface is **name** of type string; this value stores the column name of the corresponding column in the database table. Consider, for example, an employee database table with a column to store the employee's age, in this case the property **name** of the **IStorable** interface establishes the name of the age column as declared inside the database system. **IsValid()** is the only method of the **IStorable** interface and returns true if the control has valid data or false otherwise. In essence, each Web control is derived from the **Input** class, and should implement the **IStorable** interface when the control actively participates on data transactions.

RESULTS

The **IStorable** interface provides a contractual interaction between the database system and a Web server. The **Input** class provides HTML rendering information, while the **IStorable** interface adequately provides rules for storing and retrieving data. The major advantage of separating these two functions of a Web control is isolation between the control appearance and its behavior (see Noble et *al.,* 2002).

Figure 4 (a), shows the UML diagram of the **Storer class**, which brings together the Web controls of

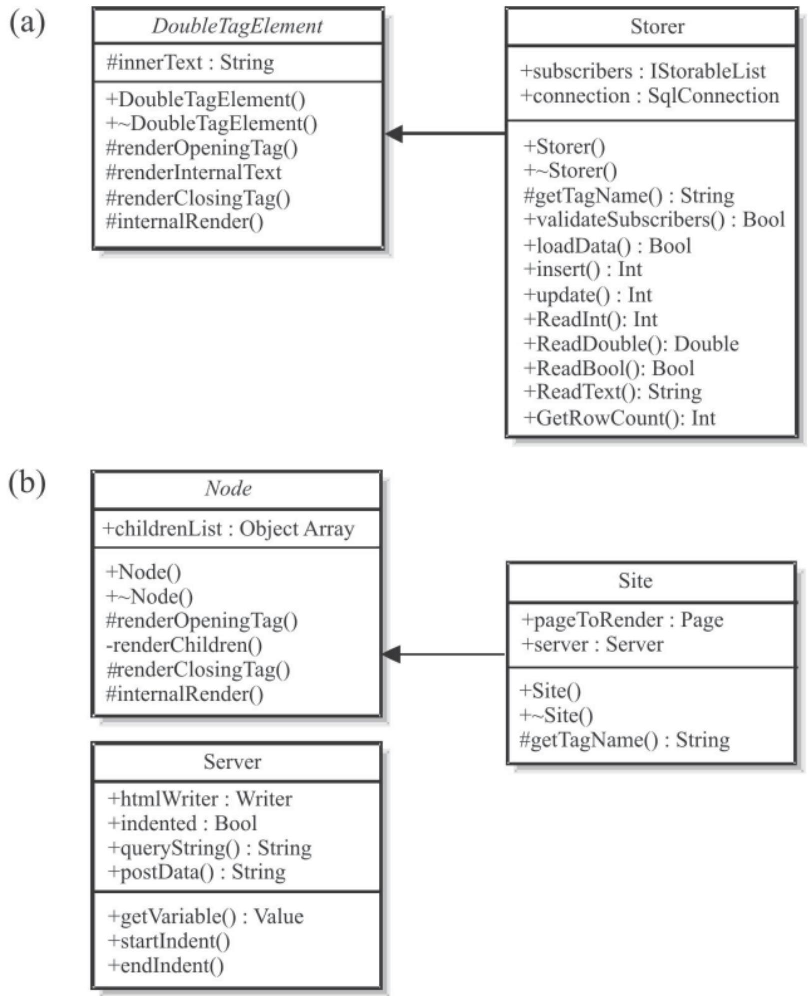


Figure 4. (a) The Storer UML diagram, (b) The Web Site UML diagram.

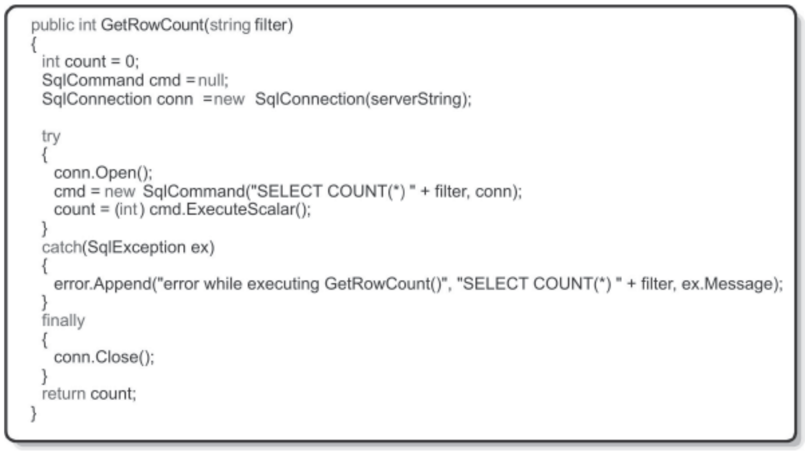
a Web page and a database system. The **Storer** class is derived directly from a **DoubleTagElement**, meaning that a **Storer** object can be placed as element of a Web page, and has the required ability of rendering. The class Storer offers an elegant technique to provide data services to Web applications, because it is responsible of controlling and displaying errors by managing all data transactions. The **Storer** class has an administered collection of subscribers; each subscriber must be a Web object that implements the **IStorable** interface. A Web page that provides database access must include a **Storer** object, and all its input controls must be subscribers of the Web page **Storer** object; which is responsible of negotiating any transaction with the database system.

When a Web page is first open, the unique **Storer** object in that page is responsible of extracting the pertinent information from the database and notifying its subscribers of update value events; this functionality is conveniently provided by the **loadData()** method of the **Storer** class, as it can be seen from Figure 4 (a). The **validateSubscriber()** method of the **Storer** class provides a means to know whether or not all its subscribers have a valid value. If a Web page provides the capability to create records in the respective database system, the **Storer** object will eventually call the **validateSubscribers()** method before inserting or updating data to the database system. If the **validateSubscribers()** method reportedly returns true, then the **insert()** method can be safely called to introduce a new record into the database appropriately. For those Web pages that provide editing capabilities, the Storer **update()** method can be used after a probative value of true is returned by the **validateSubscribers()** method.

The **Storer** object will consistently render nothing, if no errors have been reportedly occurred. For those cases when a call to l**oadData()**, **insert()** or **update()** generate errors, the **Storer** object will render the error description using standard HTML through its **DoubleTagElement** behavior as shown in Figure 4 (a).

Figure 4 (b), shows the UML diagram of a typical Web Server and a Web Site. The Web server and Web site were tested using Microsoft Internet Information Services, but other Web server software can be used. The Web site has the responsibility to attend the user's request by finding out what page to render while the Web server provides environment variables for appropriate rendering.

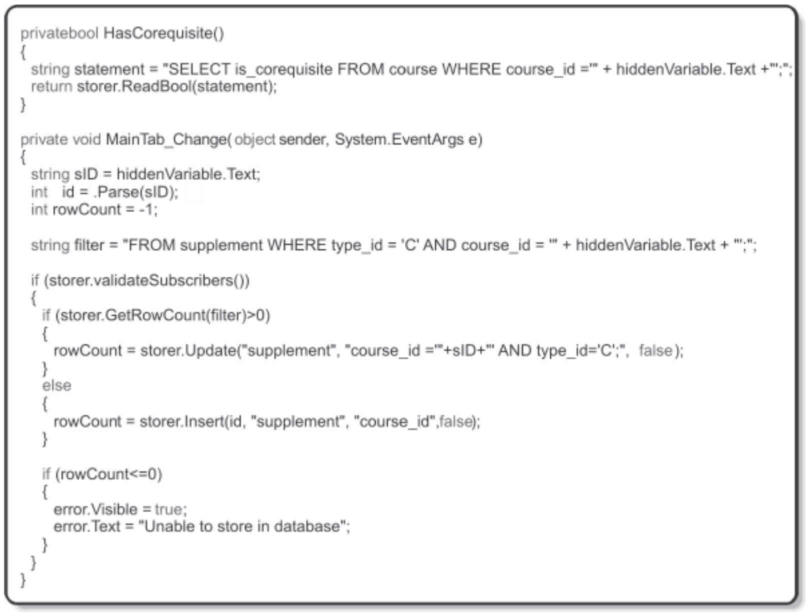
When deploying Web applications, the **Storer** class can reduce considerably the deploying time



```
public int GetRowCount(string filter)
{
  int count = 0;
  SqlCommand cmd = null;
  SqlConnection conn  = new  SqlConnection(serverString);

  try
  {
    conn.Open();
    cmd = new  SqlCommand("SELECT COUNT(*) " + filter, conn);
    count = (int) cmd.ExecuteScalar();
  }
  catch(SqlException ex)
  {
    error.Append("error while executing GetRowCount()", "SELECT COUNT(*) " + filter, ex.Message);
  }
  finally
  {
    conn.Close();
  }
  return count;
}
```

Figure 5. Typical C# code to read from a database.



```
privatebool HasCorequisite()
{
  string statement = "SELECT is_corequisite FROM course WHERE course_id ='" + hiddenVariable.Text +"';";
  return storer.ReadBool(statement);
}

private void MainTab_Change( object sender, System.EventArgs e)
{
  string sID = hiddenVariable.Text;
  int   id = .Parse(sID);
  int rowCount = -1;

  string filter = "FROM supplement WHERE type_id = 'C' AND course_id = '" + hiddenVariable.Text + "';";

  if (storer.validateSubscribers())
  {
    if (storer.GetRowCount(filter)>0)
    {
      rowCount = storer.Update("supplement", "course_id ='"+sID+"' AND type_id='C';",  false );
    }
    else
    {
      rowCount = storer.Insert(id, "supplement", "course_id",false);
    }

    if (rowCount<=0)
    {
      error.Visible = true;
      error.Text = "Unable to store in database";
    }
  }
}
```

Figure 6. Code showing the Storer class in a typical web application.

because most of the database access coding and error handling is inside the **Storer** class. Moving the database transaction coding to one place (the **Storer** class) not only reduces the code size, but also makes the code easy to read and maintain. For those cases where the **Storer** class cannot be used because the database organization does not map directly into the Web interface, typical database access code might be used. The **Storer** class and the **IStorable** interface can be easily implemented using any Object-Oriented Programming language (C++, C#, Java, or other). The use of the **Storer** class and the **IStorable** interface provides a reduction of code deploying time as it will be explained next.

Figure 5 shows the function **GetRowCount()** of the **Storer** class. This function allows retrieving the number of rows returned by the SQL statement SELECT; in general, this function can be pretty handy on application deployment. The implementation of **GetRowCount()** is straightforward but not simple; as it can be seen from this figure executing an SQL statement for retrieving information must be done using a try and catch block. Without the use of the **Storer** class the programmer must frequently type code similar to the one shown in Figure 5.

To clearly illustrate how the **Storer** class can dramatically reduce the code size, consider Figure 6. This figure shows part of the code of a web application deployed using ASP.NET to manage university courses. The first function shown is used to find out if a specific course is a co-requisite for other course(s). As it can be seen the **Storer** class simplifies the code substantially; only two lines of code are basically required compared with 18 lines that are required if the Storer class were not used. The second function in Figure 6 is executed whenever the user wants to intentionally leave the current web page. By taking a look at the code, it can be noticed that the **Storer** class validates user input and decides to perform a SQL statement: UPDATE or INSERT using its helpful function **GetRowCount()**. Thus, a pretty compact code can be writing by using the **Storer** class. If the same function would have been written without the assistance of the **Storer** class, a very much complicated code must be required; several SQL statements should have been built manually using user input, and appropriate validation/database-access code should have been required. Finally, it is important to mention that most of the error handling functionality is implemented directly by the **Storer** class. For example, during deployment time, the **Storer** class will provide immediate feedback to the developer with detail information to accurately detect and correct an error, i.e. the exact SQL statement that caused the error and why.

Finally, it is important to mention other Web technologies that ease the integration between database systems and Web interfaces. One popular technology is ADO.NET, which is the successor of ADO and OLE DB. This technology integrates effortlessly with XML, bridging the gap between relational data and XML and simplifying the task of moving back and forth between them (Prosise, 2002). ADO.NET works bests with Web forms (a Microsoft Corporation technology). Both, ADO.NET and Web from offer a set of tools and Web controls to simplify the deployment of Web applications with databases. However, they required proprietary Web controls and they do not offer a complete integration between the Web application and the database system. Other popular Web technology is Java Server Pages (JSP); similar to ASP.NET is it tries to simplify the deployment of Web applications; however, a more easy to use OOP solution is required.

DISCUSSION AND CONCLUSIONS

Despite the fact that OOP can be used to considerably simplify the integration of GUIs and database systems, many Web applications make use of structured programming for database access coding. In this case, structured programming is used on languages that support OOP; that is, they do not use an object to negotiate all database transactions. We propose the **Storer** class and the **IStorable** interface to ease the proper integration between GUIs and database systems. A **Storer** object readily maintains a list of objects that implement the **IStorable** interface. Each subscriber keeps a bilateral contract with the **Storer** object to store and retrieve data. The **Storer** object is responsible of inserting, updating and deleting any data from the database; it is also responsible of reporting of any results to other objects and the user. We showed that the **Storer** class can noticeably simplify the deployment of applications that involve database systems; the resulting code is easy to read and maintain, not to be mentioned much more compact.

REFERENCES

Abiteboul, S. and Beeri, C. (1995). The Power of Languages for the Manipulation of Complex Values. *The VLDB Journal — The International Journal on Very Large Data Bases* 4(4) 727-794.

Arenas, M. and Libkin L. (2004). A Normal Form for XML Documents. *ACM Transactions on Database Systems* (TODS) 29(1) 195-232.

Barga, R., Lomet, D., Shegalov, G. and Weikum G. (2004). Recovery Guarantees for Internet Applications. *ACM Transactions on Internet Technology (TOIT)* 4(3) 289-328.

Conn, R. (2002). Software Systems Requirements. *Journal on Educational Resources in Computing* (JERIC) 2(4).

Felton, M. (1997). *CGI Internet Programming with C++ and C.* Lucent Technologies. Inc. An Alan R. Apt Book Prentice Hall Upper Saddler River New Jersey USA.

Hall, M. and Brown L. (2001). *Core Web Programming.* The Sun Microsystems Press, Java Series, Second Edition, Prentice Hall PTR, Upper Saddler River, NJ USA. p. 903.

Holt, J. (2004). *UML for Systems Engineering.* Peter Peregrinus Ltd. London England.

HTTP - Hypertext Transfer Protocol Page (2006). World Wide Web Consortium. http://www.w3.org/Protocols/

HyperText Markup Language (HTML) Home Page (2006). World Wide Web Consortium. http://www.w3.org/MarkUp/

JavaTM Tutorial. (2006). Trail: Learning the Java Language Lesson: Classes and Inheritance. http://java.sun.com/docs/books/tutorial/java/javaOO/abstract.html

Liang Y. D. (2001). *Introduction to Java Programming, Second Edition.* Prentice Hall Upper Saddle River, New Jersey USA. 899 –902.

Liu, M. (1999). Deductive Database Languages: Problems and Solutions. *ACM Computing Surveys* (CSUR) 31(1) 27-62.

Noble, J., Biddle, R. and Tempero E. (2002). Methaphor and Metonymy in Object-Oriented Design Patterns. *Proceedings of the twenty-fifth Australasian conference on Computer* 187-195.

PeopleSoft Web Site. (2007). *Oracle and PeopleSoft Corporation.* http://www.peoplesoft.com

Prosise, J. (2002). *Programming Microsoft .NET,* Microsoft Press, Redmond, Washington USA. 177–190.

Roques, P. (2004). *UML in Practice.* John Wiley & Sons Ltd. The Atrium, Southern Gate, Chichester, West Sussex, England.

Rusty, H. E., Scott W. M. (2003). *XML in a Nutshell.* 3a ed. O'Reilly. Sebastopol, CA, USA.

Tanenbaum A. S. (2002). *Computer Networks* 4a ed. Vrije University, Amsterdam, The Netherlands. Prentice Hall PTR.

Wieringa R. (1998). A Survey of Structured and Object-Oriented Software Specification Methods and Techniques. *ACM Computing Surveys (CSUR)* 30(4) 459-527.