Web Application Framework Design Document

Andre Michel Gauthier, Gauthier Services Limited
<nomad@gserv.co.uk>

Revision History

Revision 0.5	25 April 2002	
	Initial Draft	

The Web Application Framework is an aim to provide an easy to use tool set to develop scalable Web based applications that are easy to develop and maintain.

The approach has been to place function before form and to enable developers to build all the required functionality of an application and then customise the front end using simple templates. Applications can be built from small simple components providing direct access to database functions and/or providing output routines. Simple components can be linked together to produce complex behavior required of an application.

Table of Contents

Introduction

The web application framework has been designed to provide an extensible framework for rapid web application development. The aim has been to provide a means of developing applications by placing function before form to enable the rapid deployment of workable solutions in the quickest time possible. The developed application can then be tweaked using an output templating layer to create the desired appearance once the required features have been implemented.

Many of the current systems for developing web based applications (e.g. ASP, JSP and other web scripting languages) are based on a two tier architecture where a back end database is accessed for information by a server processed script which then presents the results to the end user. Two tier architectures tend to mix form and function requiring developers to code the business logic and display logic concurrently. This mix also makes maintenance of the code rather more difficult as changes to form can affect function and vice versa. In an attempt to address this concern three tier architectures (e.g. EJBs) have been developed placing the business logic in an abstract container and then allowing the display logic to be developed separately and call functions of the intermediate tier. The added code complexity of three tier architectures often extends the development time and costs for a web application. The intermediate tier will also require some container application to hold the business logic adding additional cost and complexity to the deployed application. Three tier architectures also tend to require components to be developed in a high level language requiring a further investment in skills over simple scripting languages (which are still required for the display layer).

The web application framework enables developers to build applications out of simple components and then use a simple template language to produce the display. The aim has been to take the best aspects of two and three tier architectures and then reduce the complexity of application development as much as possible whilst retaining flexibility. Based on a two tier architecture (web server front end and database at the back end) the business logic and display logic have been clearly separated in the first tier (web server). Although other applications have been developed to provide this separation in the form of display template routines they can still only be accessed through the scripting language. This is not enforced in the scripting languages so developers can (and probably do) sidestep the separation. In the web application framework the separation is performed by the framework and can't be bypassed providing a strict environment to develop applications.

Features

Enhanced Two Tier Architecture. Providing a clear separation between the data layer and the interaction layer without the added complexity of three tier systems.

Connection Pooling. Enabling rapid response to requests by reducing the need to renegotiate connections with the database. In multi site installations this can also prevent high levels of activity on one site from depriving other sites of access.

Multiple database configurations. The framework supports multiple database configurations allowing one consistent application front end to access many different back-end databases.

Simple component architecture. Allows the development and debugging of applications one step at a time and facilitates easy customization of applications to end-user requirements post installation.

Simple development. Development of applications has been simplified by removing the need for any additional Java development. Entire applications can be created through the database configuration and creation of simple SQL templates to interact with the database.

Complex Behavior possible. Simple components can be chained together to produce complex behavior even interacting with different databases at each stage.

Configuration Database. All of the configuration is held in a database providing centralised configuration management. Database replication enables configuration to be maintained centrally and propagated to many sites transparently.

Web based interface. The View Manager Application provides a Web interface which can be used to develop and maintain applications remotely.

Wide range of supported databases. The Web application framework leverages on standard JDBC drivers providing access to a wide range of backend databases.

Portable Java Servlets. The framework has been built using Java Servlets which can be installed in many off the shelf web servers without the need for complex and expensive application servers.

Pull down Menus. Values in the database can be easily referenced to create pull down menus to ease user interaction.

Simple Templating system. Someone with basic knowledge of HTML can easily add the required additional tags into their code to create customised output without affecting the underlying business logic.

Abstracted Security. The access control is handled by the framework and therefore does not rely on any access control to be present in web server on which the system is to be deployed.

View Level Access Control. Users can be granted access only to specific views based on the access groups which they are members of.

Record Level Access. User and group id details can be used to restrict access to specific records within a view providing even greater control on who can access/modify data.

Technical

The Design has been broken down into three Layers: Data Interaction, Component Linking and Display. This strict delineation allows development of each layer to be focused on separately and if desired by separate developers.

Data Interaction Layer

The data interaction layer can prepare a number of different view types to process and display. The simplest is the input view which simply specified fields to be used in a form to get information from the user. A rudimentary view type for editing text files on the web server is also available. The most utilised view types are results and update views which both use SQL templates to either update the database or select records from the database.

The data interaction layer utilises a template processor to parse an SQL template. Special tags similar to HTML markup are substituted for required values in the SQL and then this is run on the database server. The templates allow for default values to be placed into the SQL output when no data is input by the user. An example of an SQL template is Example 1. Each %prop tag specifies the name of a variable from the form data posted to the web server the default attribute specifies what value should be substituted in the absence of input. The suffix and prefix attributes specify additional text that should be placed before and after the substituted value. If no value is present and no default is specified then nothing is substituted.

Example 1. Example SQL template.

```
select * from announcement
where
title like <%prop name="title" prefix="'" suffix="%'" default="">
<%prop name="announcementid" prefix="and announcementid=">
order by posted desc
```

Component Linking

There are three ways in which views can be linked to other view providing a flexible means to construct an application. Views can be linked either by a standard link, an in-line link or a chained link and there is no real limit to the number or combination of these link types.

Standard Link. Standard links provide a means to submit data collected or displayed on one view to another view. The most common example is to use a standard link to link to a view which updates that record. These links generate submit buttons on the HTML form.

In-line Link. In-line Links work in a similar way to standard links as data collected in one view is submitted to another view but in this case the results are displayed in the current view. A simple example of this would be to display all the groups a user is a member of when you view the users data.

Chained Link. Chained links allow for data submitted to one view to be passed on to another view once this view has finished processing. This can be used to update data in more than one database at a time, re-display a record after updating it or displaying a new record after inserting. This last example utilises the ability of update views to interrogate the database and obtain the unique record identifier of a newly inserted record. This feature can also be used to interrogate the database for other values such as the number of records remaining after a delete.

Display Layer

The display layer uses the template engine to parse simple headers and footers to enable easy generation of a consistent front end to your application. There are default output methods which will then present the results of your data interaction as a HTML form which can then be modified and/or used as input to another view. The behavior of the default display engine can be tweaked allowing the type of form element to be changed as required. Most form elements can be created in this way including text boxes, password fields, large text areas, hidden fields and pull down menus. Any linked views are also displayed by the default routines.The default display routines can optionally be replaced with even more customised HTML templates which can also make use of form elements generated by the default display routines. E.g. the pull down menus and any linked views.

Demo Applications

As a proof of concept and a test of the frameworks capabilities a number of demonstration applications have been developed. These applications include a simple intranet system comprising of an announcements board, a discussion board and a company directory.

View Manager

The View Manager was developed to provide a Web interface to the Web Application Framework. All components of the framework can be created modified and deleted via a simple to use front-end including the editing of SQL and HTML templates.

Intranet System

The intranet system was developed to provide some commonly required applications that could be used to kick start the development of an integrated company intranet system. Additional applications can be added into the system to provide a unified working environment.

Project Manager

The project manager was developed as a simple project manager allowing projects to be created tasks created and resources assigned to tasks. Expenses can also be filed against projects and budgets can be calculated based on resource costs and expenses incurred by a project.

The Project manager has also been linked to the discussion board component of the intranet system enabling access to discussions related to specific projects.

Deployment

There are various possible ways of deploying the framework depending on your requirements.

Single Site Install

A simple single site install is the most basic deployment of the Web Application Framework (Figure 1). All user requests are handled by the web server which verifies the user, looks up the view configuration from the database server. The view configuration is used to establish which SQL templates to use. The web server then parses the SQL template and submits it to the Database. The database will then process the request and send a response back to the web server. The Web server will then process this response performing any link operations and obtaining further responses from the Database if required. Once a complete set of data is collected the data is fed back to the user.

Variations on this deployment could involve the users being based at many sites and accessing the web server using secure HTTPS communication over the internet. Client certificates could also be used if the web server supported it to prevent any unauthorised access. In order to simplify the deployment the database and web server may also reside on the same physical machine although there are some security implications for this if the web server is to be visible on the internet. There may also be more than one database server present and the web server will determine which database connection to use when it loads the configuration for the view. The databases may also be of different types i.e. Oracle, SQL Server or PostgreSQL.

Multi-Site Install

A multi-site install can be done a number of ways one of which is outlined in (Figure 2). In this case these databases may be of heterogenous nature with each web server able to communicate with the other sites databases (preferably through a VPN over the internet or using SSH port forwarding). This allows the same consistent interface to be available to all sites and for all sites to be able to access each other's data when required. As their own database is local, a site would not be reliant on high bandwidth connection to the internet for the majority of it's functions.

If all sites need access to the same data which may not need updating often then one database could be configured as the master database and the others as slaves. The framework could then be set up to read data from the local slave database and only write to the master database. The master would then replicate any updates down to the slaves. This would provide the same functionality of a single site install that could be accessed from may sites using HTTPS but local data reads should be significantly faster and the load on the master database lower.

Future Plans

Due to the current trend for greater and greater integration of information systems into larger systems it has become necessary to consider the ability to integrate the framework into larger systems. The rise of XML as a mechanism for communication between systems has been gathering pace over the last few years and seems to be the way that the industry will go.

Within the current framework it is possible to write output templates that could output data as an XML document. Future extensions to the framework will be to provide default XML output methods for views as there currently exists for HTML output. Due to the component architecture that has been employed throughout the development of the framework this will be a seamless upgrade that would allow all existing views on the data to be output optionally as XML with little or no additional development of an application.

The next phase of development will be to provide an additional request handling component that could process XML requests and forward them on to the underlying view manager component to produce an XML request. There are a number of currently emerging technologies for integrating applications that handle XML requests and responses (e.g. SOAP) that can provide a larger framework for these applications to work within. Where possible compatibility with as many of these systems as possible will be maintained. The ability to handle XML requests will also enable easy integration with many existing e-hub application servers on the market e.g BEA Tuxedo.

Glossary

API	**Application Programmer Interface.** A structured way of forcing programmers to write code that can communicate with some other code.
ASP	**Application Server Pages.** A web scripting system produced by Microsoft.
Client Certificates	**Client Certificates.** A mechanism for a user to be validated by a certificate that has been signed by a certificate authority. If a user does not have a valid certificate signed by the correct authority then access can be denied.
E-Hub	**E-Hub.** A large scale application server designed to handle transactions and requests for data over a large number of different resources.
EJB	**Enterprise Java Beans.** A set of programming APIs to enable creation of Java components that can be distributed across a network developed by by Sun Microsystems. See Also API.

HTML	**Hyper-Text Markup Language.** The most common way of presenting information over the Internet using simple tags to convey formating information.
HTTPS	**Hyper-Text Transfer Protocol (Secure).** A method of transferring data between a web server and web browser that utilises public private key encryption. See Also Public Private Key Encryption.
JDBC	**Java DataBase Connectivity.** An API for communicating with databases. There are a large number of database servers that have drivers which support this API. See Also API.
JSP	**Java Server Pages.** A web scripting system produced by Sun Microsystems.
Public Private Key Encryption	**Public Private Key Encryption.** Public Private Key Encryption is a method of encrypting information in such a way as to only enable one person to read the data. A public key and a private key are generated. Any data encrypted by the public key can only be decrypted by the private key and vice verse. If A wants to communicate with B then A would encrypt the data with B's public key (which any one can have) and also with A's Private key. When B receives this data he can use A's public key to verify that the data has come from A and then use B's private key to actually read the data. As no one else should have B's private key then the data should not be intercepted.
SQL	**Structured Query Language.** The standard query language used in the majority of database systems
SSH	**Secure SHell.** Secure shell is a means of encrypting specific traffic between sites using public private key encryption. It is normally used to provide an encrypted remote login to machines with sensitive data on. The port forwarding facility allows for any specific traffic intended for a machine (e.g. just database access) to be encrypted between the two sites. See Also Public Private Key Encryption.
VPN	**Virtual Private Network.** A system usually configured using specialised firewall routers and public private key encryption to enable sites to be linked securely over the internet. See Also Public Private Key Encryption.
XML	**eXtensible Markup Language.** A mechanism for marking up data usually indicating the meaning of the data.