GenericServ, a Generic Server for Web Application Development

Samar TAWBI
PHD student
tawbi@irit.fr

Bilal CHEBARO
Assistant professor
bchebaro@ul.edu.lb

Abstract

Web application developers face currently several challenges to design their web sites. In this paper, we describe a generic server for web-based applications called GenericServ. GenericServ is an abstract nucleus offering an easy to use interface that web application developers can use to create sophisticated web sites. Depending on the applications' domain, developers can generate specific web design servers for this domain. As an example of the use of GenericServ, we developed a publishing server called PubliWeb.

1. Introduction

The Internet became an essential media, especially the World Wide Web that is recognized and used by enterprises, government agencies and the wide public. The information flowing using the World Wide Web represents today more than two thirds of the overall Internet traffic [1]. The media types used in the Web communications are very diverse and are utilized for a wide array of applications. This diversity of media, user types and applications for the Web pushes for the need of sophisticated web sites. It is not enough to present the information to the web users as a bulk in an artistic manner, but it is essential to provide a great level of interactivity and easy access to sophisticated databases. Hence, web applications, like client-server applications, have generally two requirements, functionality (development and deployment) and graphical interfaces (layout).

Covering these two requirements for the web applications induces the need for different types of skills. Several persons are then required for a web site development and the cooperation between these persons is not always easy to manage and coordinate given the complexity of the interactions between the services provided by each group (development and deployment group and layout group).

Our approach to deal with the above issues is to free the web application designers from the need to have development expertise and a special working environment. This is achieved by clearly separating the tasks related to the deployment of the functionality and services that have to be supported by the web site under construction, and those related to the layout and graphical design.

We propose GenericServ as a generic tool or server that offers a platform providing the web site designers, the storage, the access and the processing of the information. Using this server, every individual will be able to focus on his area of expertise and integrate it in the whole project. GenericServ is an abstract service generator that defines a general guideline. It can be extended to deploy concrete service generators by specialized layers to it. Its architecture is independent of the offered services and is abstract enough for future service extensions. These extensions are of course dependent on the required server. For example, it is possible to deploy as many servers as the domains to be covered (publishing, e-commerce, email servers, etc.).

The rest of this paper is organized as follows. The next section discusses the motivation for the creation of such a server. It is followed by a description of the GenericServ operation mode. Then we present the three-layer architecture of the server that includes a core, basic utilities and specific layer (PubliWeb). The service architecture is then presented before our conclusions.

2. Motivation

The experience has shown that it is very difficult to achieve the right goals if any of the two groups developing a Web application (developers and graphical designers) tries to bypass the other group’s services. Hence, the designers taking care of the layout of a web site have great difficulties in integrating the services required for their applications. This paper addresses this particular issue of coordinating the efforts of the two groups using a generic server tools and services to simplify interactions between them by providing a virtual independency of their tasks.

Any development expert always tries to reuse existing solutions for the design and deployment of a new application. This will allow gaining time and achieving quality by adding improvements or services to existing software and design instead of reinventing the whole work.
Reusability has motivated several research efforts as, for example, the class libraries and APIs for programming languages, inheritance and abstract classes. Currently, software designers have introduced the concept of Design Pattern, defined by [2] as ‘A Solution to a Problem in a certain Context’. Another definition is given by [6] ‘A design pattern explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences’. Design Patterns have dramatically changed the way applications are designed.

Different approaches have been taken in the distributed data communications and the Internet development areas. These approaches take into account the multitude of platforms, programming languages and communication protocols, like the Java “Aglets” and The Java RMI, which offers the possibility to create objects on a server and allows their remote invocation [5]. On another side, the intermediate software or middleware becomes more and more important, like CORBA, DCOM from Microsoft and EJB [4]. All these techniques offer the complex means for developers to achieve complex applications in an easier way. They can be used to solve the complex application development tasks. However, they face two main obstacles for web application developers: They still require a high level of expertise and they impose a very specialized development environment.

Our GenericServ aims at providing a simpler and reusable way to achieve the goal of implementing Web sites covering several services and domains. It proposes a generic server that defines a core and a set of basic services that can be used to develop more sophisticated services. Even though these services can be very diverse, the servers hosting them will have the same operating way. This is the case, for example, of our publishing server PubliWeb that offers for publishing web site creators services such as editing, submissions, consulting and management (e.g. accepting, rejecting and deleting articles) of articles to publish.

GenericServ defines as well the cooperation means between the developers and the graphical designers. The developers implement the functionality and services of the web application without the need to deal with the issues related to the layout if these applications, while the graphical designers do not need to be involved in the technical details of the development. GenericServ is responsible of merging the two efforts through a common interface. The web applications permanently evolve to integrate new functionality to answer the new users needs like the need for a new login page, a database, a chat room, etc. Our system also deals with this situation by offering the tools to easily extend the already developed applications.

3. GenericServ operation

Using GenericServ in a given domain, developers implement the functions that need to be supported by the web application in this domain as “Services”. These Services are made available to web site designers who are specialized in graphical interfaces. The latter integrate to their site the required functionality by accessing the Services dynamically on the network.

The site designers hence avoid the difficulties related to the deployment of the functionality of their applications. They only need to know the list of available services and their possible parameters. They will then need only to include the references to the services in their pages that will be executed when the end user (site navigator) tries to download these pages. The service is then offered with its latest version to the site visitor while every service update is taken into account without the need to make any change on the web page accessible by the visitor.

A web site that uses our server is deployed in a way that the pages containing the service calls are first directed to the server, for example PubliWeb (figure 1). This server analyses and augments the page by the results of the requested services. The hypertext links to these pages will have the following format:

url_server ? URL_page_html

Where url_server is the server’s URL, which is PubliWeb in our case and URL_page_html is the URL of the web page to access. Obviously there can be direct links to simple pages that do not contain any service invocation. The navigating end users will be connected to our server when they click on a given link.
A Java servlet that is the link between the server and the end user is responsible for managing this communication. It searches the corresponding page on the server hosting the site, then analyses it and sends the response to the end user. All this process is totally transparent to the end user.

We have chosen to separate the proposed services by the GenericServ-based servers in two categories: basic and specific. Basic services are common services that can exist in any web application regardless of the domain to which it applies, such as login pages. The extended services are specific to a given domain, like editing articles for publishing web sites. This separation eases the design of the server on one hand and on the other hand it allows a better extensibility of the server and adaptability to different domains. In fact, only the specific part needs to be changed (extended services) to obtain a publishing server, an email server, an e-commerce server, etc. while the common part remains unchanged. Any of these servers can be shared by several web sites (Figure 2).

The functionality provided by these servers, whether it is basic or extended, has been grouped into service classes or simply services. Hence, for example, the authentication service contains the different functions to construct the login web page where every field of this page may correspond to a call of the service with a different parameter (Figure 3). The available functions for a service should be invoked in a logic that is proper to this service. This induces the need for verification at the server if the reference to these functions has been accomplished according to the rules of this service.

Figure 3. Web page using the server

We use the keyword serverkeyword to indicate a service invocation with possible parameters. The expression will be replaced by the result of the processing of the requested function of the service. This result is usually an html code included in the page presented to the end user. In the example in figure 3, serverkeyword.login means that we call the service related to the end user authentication.

This service prepares the form where the user will be identified, verifies the login and password and allow the access to the user with a given welcome web page. The example shows four service calls with different parameters for every call. Every call is interpreted differently based on these parameters.

In general, a service call in a web page has the following format:

serverkeyword.service_name ([attribute, …]).

4. Three-layer Architecture

Every server compliant with the above-described model conforms to the Layered Architecture pattern described in [7] and to Reflection pattern in [3]. It contains a three-layer hierarchy that allows easy extensibility:

- A generic part called the core layer. It contains the processing and analysis modules. To be generic, this level is completely independent of the internal behavior and the application domain to which the server is applied.
- A set of basic services common to all the servers that constitute the basic layer of utilities [7].
- A specific part that contains the services used in a specific domain. These services have to be created in accordance to a skeleton that conforms to the server’s nucleus. This layer is called Useful Systems [7].

Our GenericServ covers the two first layers and constitutes the base level of the Reflection pattern described in [3]. This is because it contains the stable part
of the system including a user and new service development interface.

4.1. GenericServ Core

The GenericServ core is the analysis module. It receives the end user requests and processes them, then redirects them the appropriate service. It creates with the basic services the common part around which the different servers are formed, for example the PubliWeb publishing server. The core alone is inactive from the user’s point of view, but it offers a platform on which domain-specific services can be added. These services make the keywords that will be provided for users or for publishing web site creators in the case of PubliWeb. The GenericServ core consists of four modules (figure 4):

Figure 4. Server architecture

- Connection Manager: It is the entry point to the server. It plays the role of the barrier that hides the processing modules of GenericServ.
- HTML page analyzer: It detects the service calls, verifies their validity (the format of the calls) and extracts the required attributes for the processing of the service.
- Service Manager: It handles the assignment of requests to corresponding services. During a service call, the page analyzer sends a request to the service manager with the corresponding service name. The service manager verifies the validity of the invoked service before launching the execution.
- The Session Manager: It manages the end user sessions and the user-related information that needs to be stored during these sessions such as login name and password.

4.2. Basic utilities

It contains the basic services that can be used independently of the implemented server type. Some examples are authentication services, database creation and access.

4.3. Specific layer: PubliWeb

This is a specific layer that we have created on top of the core and the basic utilities (GenericServ). It groups the classes that constitute the publishing services as well as all the functionality required for the proper operation of these services. These services have to be created in accordance to a skeleton that conforms to the server’s nucleus. It is this layer that renders the server visible and ready to receive requests and analyze them and respond to them.

This layer has mainly two modules (figure 4):

- Extended services: that cover the functionality that the PubliWeb offers to the publishing web page developers. In addition to these services, the management and control functions have to be added. They contain the semantic verification of the calls, which is the attribute type, the calls order, etc. Several services have been created for PubliWeb like article submission, published articles viewing, article addition and deletion, acceptance and rejection of articles by the article administrator, etc.
- Utilities: Every server requires a set of internal functions to facilitate the operation of the services. This module handles these tasks almost independent of the system that are essential for the system and the server. This module is composed of classes containing the recurrent functions, such as the HTML fields generation or specific function to the server’s domain such as certain tests for article publishing or default system parameters.

5. Services structure

The current services have a well-defined skeleton that has to be enforced for any future service. The skeleton is flexible enough to allow the development of most of the functionalities that can be considered in the future. All services have to have the same interface in order to simplify their use. Without this property, it will become difficult if not impossible to automatically load any new service for future invocation. For every service, some parameters have to be specified before its implementation: Is it a simple service? Do we need to keep a trace for the service? Is this trace needed for the whole duration of the session? Does it contain several calls for the same service? Is it dependent of other services? Etc. The answers to these
questions determine the need or not for certain objects that define the service.

The implemented services should follow implementation guidelines and have to be grouped in specific packages in order for the server to integrate them automatically in its available services.

The structure of a service is presented in figure 5. It is composed of several objects of the following classes:

- Service: provides the result of the service invocation in the form of an HTML text. This result is sent to the user. All the classes of this type have the same interface.
- Parameters: These are the parameters and tools that this service uses.
- ServiceRedThread: This is the class that saves the traces used during the lifetime of the analysis of a given web page.
- ServiceTrack: This class gathers the traces that should be saved for the whole lifetime of a user’s session.
- ServiceRedirection: This class handles the processing and the choice of the redirection page based on the results obtained from the service execution.

![Service Structure Diagram]

Figure 5. Service Structure

6. Conclusion

The GenericServ architecture we propose provides a solution to the web design issues encountered by the non-specialized developers by separating the development and implementation from the layout of web pages. This is achieved by a flexible system that gives the web designers and the service developers an easy to use interface. Service semantics can change dynamically and can be updated without the need to change the existing web pages code. The three-layer architecture of GenericServ allows an easy creation of specific services, such as the PubliWeb service based on a generic core and a set of tools provided in the form of basic services.

Although, we are talking about services called within HTML pages but the architecture of the server is not restricted to deal with HTML. In fact, we could work with an XML analyzer rather than an HTML analyzer. This will give the server a new perspective as it will no more be restricted to web development, it could cover also distributed applications especially if we use the SOAP protocol for service calls.

7. References

[6] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns : Elements of reusable Object-Oriented Software, Addison Wesley, 1995