Chapter 22

A WEB SERVICES PROVIDER

Jean-Paul BAHSOUN *[1], Bilal CHEBARO [2], Samar TAWBI ♦[2]

Abstract: In this paper, we define a generic tool 'GenericServ' that offers a 'service providers' platform which facilitates the programming tasks of web application development. The system's architecture is generalized to propose three patterns for business applications' development. The paper is divided into two major parts. In the first one, we expose the motivation for the definition of the service provider, where we emphasize the architecture of the system and the arguments to choose such architecture. In the second part, we define the patterns based on this generic service provider.

Key words: Web development, Web services, Genericity, Patterns.

1. THE WEB EVOLUTION

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 flow that uses the World Wide Web represents today more than two thirds of the overall Internet traffic [2]. In the last five years, the Online Computer Library Center research determined that the public web has more than doubled in size, increasing from 1,450,000 sites in 1998 to over three million in 2002 [15]. More than 55% of web site creators are not programming specialists. The web has become now a requirement for everyone in all domains. User types are multiplying, the requirements are increasing and the web sites are more and more complex. For this reason, a web services provider, facilitating the web applications development,

♦ Scholar from the CNRSL.
* Names are in alphabetical order.

becomes critical, especially if it is a generic one that can be deployed in many areas on the web. This paper presents a solution to simplify the development of web applications.

The paper is divided into two major parts: in the first one (section 2) we will outline the motivation for this generic service provider's development. Then, we will propose a general architecture for our system, taking into consideration the different existing architectural styles that can be used in this application. In the following part (section 3), we define a set of patterns, based on this tool, and generalized to the development of business applications.

2. GENERICSERV: A GENERIC SERVICE PROVIDER FOR WEB DEVELOPMENT

2.1 Presentation

Since the needs for sophisticated web sites is more and more relevant, especially for the functionality that is related to business domains like commerce, education and banking, a generic service provider 'GenericServ' was proposed in [23]. It is a generic tool or server that offers a platform providing the web site designers the ability to store, access and process the information. Its generic nature makes it independent of business domains. It defines a protocol for service creation and a general architecture that can be extended to deploy concrete servers in Web domains as article publishing, e-commerce, e-mailing and education, by defining the set of functionalities needed in these areas.

Many tools were proposed to facilitate the web site development, from the HTML editors to content managers. But the first category of tools appears to be too basic for sophisticated needs. This is especially true for the functionalities that are related to business domains. On another hand, even though the content managers are very helpful for easy web development, the users of such tools should have the product on their working environment. So their update and upgrade are not automatically accessible by the users. Meanwhile, approaches that have been taken in the distributed data communications and the Internet development areas like [5][8][9][10] face two main obstacles for web application developers: They still require deep programming skills and they impose a very specialized development environment. In this context, we propose a generic service provider as a platform used to create servers offering complex functionalities that Web application creators could call from their pages (HTML or XML). The

services offered are remotely called from the web pages, and are executed in real time, when the page is accessed. Such server has a simple interface to use as it is dedicated to non-specialists.

Thereby, the generic service provider constitutes an abstract architecture that proposes the management of client sessions, services and data storage as well as a template for the definition and the interface to use the services. In fact, this nucleus serves as a base for concrete servers extension. Instantiating a server will consist of implementing the functionalities needed in a certain domain, as services according to a template imposed by the generic service provider. These services always follow the same interface so that the provider could take them into consideration automatically. In addition, this allows easy evolution and maintenance of the system and complete transparency to end-users. This tool defines also the cooperation means between the developers and the graphical designers. It is responsible of merging the two efforts through a common interface.

We should note that although it could; in general, the concrete service provider does not provide hosting services for web sites. In fact, it constitutes an intermediate layer between the end-user (i.e. the navigator) and the hosting server of the web site (see figure 1).

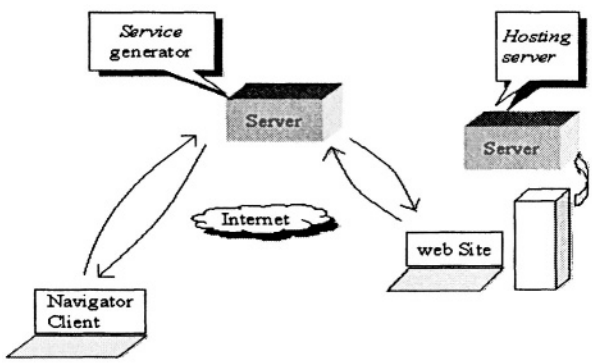

Figure 1. **The concrete provider operation constitutes an intermediate layer between a navigator and a hosting server**

2.2 The System's General Architecture

The system explained above aims at providing a simpler and reusable way to achieve the goal of implementing Web sites covering several services and domains. Therefore, the architecture design should support the required

software system qualities such as robustness, adaptability, reusability and maintainability [3][4]. So in the case of this services provider, the architecture has to guarantee its flexibility, easy reuse and evolution.

Patterns have become now a must that software designers and developers should use or at least try to apply systematically in all phases of their work. There have been patterns applied to the analysis phase [11]. Patterns have also been applied at the architectural analysis phase of the software development process. In this level, patterns propose an abstract representation of the system's architecture [1] [4]. They are also called architectural styles [21] [20]. A number of architectural styles [12] was defined serving certain categories of software systems like repository models [6], layered architectures [19], client server architectures [17] and others.

The major concern in 'GenericServ' was to choose the architectural style that is the most suitable for this system. Since each one of the known styles [12] [18] has some drawbacks to apply to our system, we have opted for a heterogeneous architecture using a combination of the three architectural styles listed above.

2.2.1 A heterogeneous architecture

The provider will be deployed on a three-tier client/server architecture that has the advantage to split the deployment to three levels, which are appropriate to the deployment of the system.

In this architecture, every server based on GenericServ (the generic service provider) will consist of the three tiers: client, processing and database (Fig.2). The other styles are used to define the internal architecture of the different tiers.

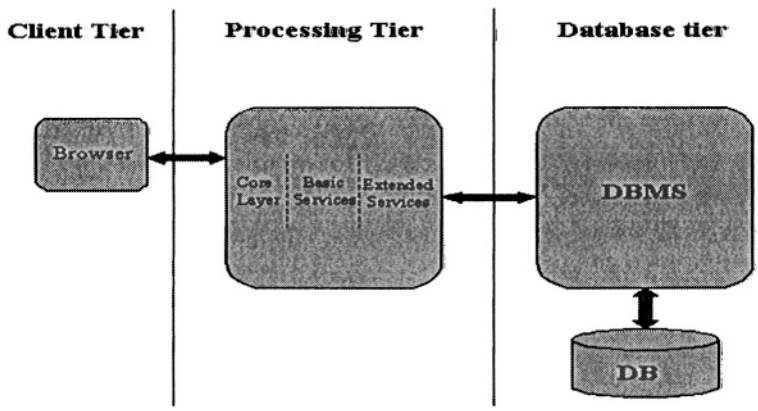

Figure 2. The 3-tiers architecture of the server

In the following we provide more details on each one of the tiers forming the architecture.

- The ***Client tier*** represents the interface that the web applications' creator uses for accessing the services offered by the provider. This is a thin client with almost no processing responsibilities to free the web application developers from programming tasks. So, it does not restrict any special software platform in order to free the users from any programming effort. Thus, the web site creators will just need the list of the services available with their parameters, if any.
- The ***Database tier*** centralizes the data related to the applications in a repository database managed by a DBMS. It is accessed through the services offered to users, but stays completely transparent in order to allow independent updates and changes. It is accessed through an interface implementing the DAO pattern [13]. This makes the deployment of the Database server changeable and upgradeable independently of the other parts of the system.
- The ***Processing tier*** constitutes the services provider itself. It is deployed as the middle-tier containing all processing, client management, testing and security. Its internal architecture consists of three opaque layers designed in a strongly modular object-oriented architecture in order to take the advantages of the independence that the layered style offers.

'GenericServ' standardizes the management of the services and defines a communication protocol between the different actors of the application. Thus, around this core, a set of services could be defined for a specific domain to form a useful server, like the one created for online article publishing needs 'PubliWeb' [23]. In the next paragraph, we will focus on the structure of this tier as it is the central one and it contains all the processing tasks of the system.

2.3 The processing tier architecture – three layers –

In the processing tier of every concrete server, we distinguish three layers (Fig. 3):

1. The **core layer** It constitutes the kernel of the system that includes general processing. This core layer defines and implements the management of the users and services in a generic way, in order to be independent from the specified servers business domain. It proposes a skeleton for all services that will be defined on this server. It contains four modules:
   - **Connection manager:** It constitutes the entry point of the server (its interface of use). It manages the connection requests, the verification,

if the requests include the necessary parameters, and the interaction between the client and the server.

- **Analyzer:** It analyses users' requests. It must be transparent in order not to put too many constraints on the users. Its role is to capture the service calls, to verify them syntactically and to extract their attributes. Then it sends these calls to the "Service Manager" that will delegate them to the corresponding services.
- **Service manager:** It dispatches the calls, arriving from the analyzer, to the appropriate services after verifying their existence and their dependencies with other services. This is achieved by keeping a temporary trace of the running services that will be consulted for any dependency tests.
- **Session Manager:** It handles the client session's management. It also safeguards the state of the client's work during the session.

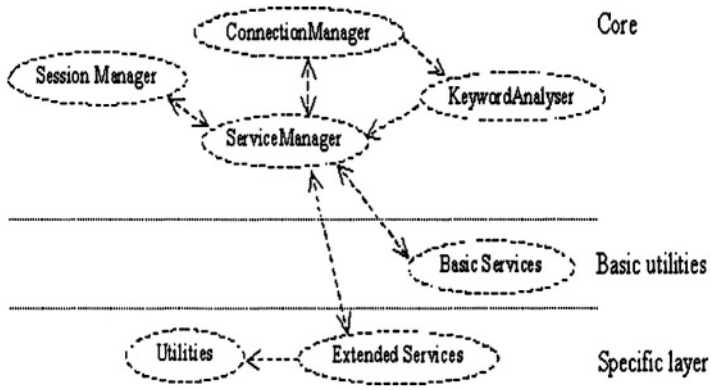

Figure 3. GenericServ based services providers' architecture.

2. The **Basic Utilities:** this layer contains basic services that can be present in all service providers independently of their domains. It contains, for example, services to communicate with the database tier. Other common services like client authentification, security or any other domainless functionnality can be created in this layer in order to encourage reuse and avoid redundancy.
3. The **Specific Layer** represents the set of services proposed in a precise domain like article publishing. Some examples are online article submission, article viewing and article acceptance or rejection. A module 'Utilities' is defined in this layer covering redundant parameters and functions that are necessary for the services' processing.

The processing tier uses also the Reflection pattern defined in [6] as a way to divide the system into two levels: a Base level and a Meta level. The base level is responsible for the computation that stays stable through the

concrete server's lifetime. It contains the core and the basic utilities layers. The Meta level contains the varying part of the system. It is the services layer. The separation of the server into these three layers allows an instantiation of the generic provider into concrete servers with a minimum of effort, since this task consists of implementing services according to the prototype imposed by GenericServ.

3. A CATALOGUE OF PATTERNS

In this section, we define a set of patterns inspired from GenericServ and generalized to business applications. The first pattern proposes a *service provider* to help the non-experimented programmers to perform sophisticated development in a certain domain. In the second pattern *'Generic Server',* we propose genericity in order to respond to the permanent evolution of users' needs and to make the server functionality independent from the business domain of the services it provides. And finally, the last pattern defines an opaque *'modular layered architecture'* for this type of applications. These patterns are discussed in the following sections.

3.1 The Services provider pattern

Context

Applications in a same domain may have lots of common functionalities with each other, but different designs and user interfaces. Redoing the same development several times is a waste of time and effort. In addition, an application needs different type of actors (analyzers, graphic designers, developers...). And sometimes the cooperation between these actors may cause certain problems.

One would like to create an application in a certain domain. Therefore, the creator of the application may not be very well experienced in programming, although he wants to conceive a robust and performant application.

Problem

You would like to minimize the redundancy of the functionalities in order to reduce the development efforts. So, you are trying to reuse functions that have been developed in other applications. On another hand, you have to guarantee a good cooperation between the different actors of the application without affecting the robustness and the efficiency of the applications. And

essentially, you want to offer some functions created by experienced developers to non-specialists or simply less experienced people.

Solution

Create a service provider offering the common functionalities needed by the applications in a certain domain, in the form of services that could be used by the applications' creators (Fig.4). This services provider could be used by different applications in the same domain. It has to have a simple interface for the users. This interface must be transparent to the user services in order to allow the developer to call the services without the need to know the structure of the provider. The system could be deployed as a server in a client/server structure, where the applications using the services will act as clients of this server; or, for example, as an application where the users will choose their services in some kind of graphical interface.

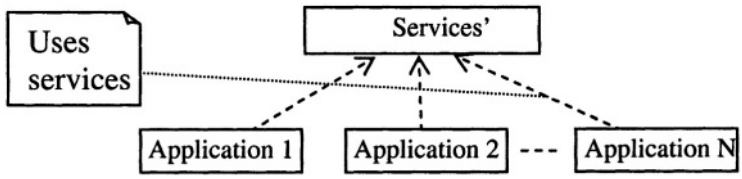

Figure 4. *The Service provider structure*

Known uses

The html page editors, the IDLs (Interface Definition Languages), the development environments for programming languages like Borland C++, Symantec Visual Studio, etc.

Consequences

- The Services Provider pattern will free the application creator from most of the development tasks, so it makes it possible for a large group of non well-experienced programmers to implement relatively sophisticated applications.
- There is a complete separation between the different actors of an application while keeping an efficient communication between them.

Related patterns

Generic Server (see next section) is the generic version of this pattern. Our pattern is inspired from Technical Infrastructure [14] that proposes a solution to system complexity by encapsulating the computing tasks from the application developer.

3.2 The Generic Server pattern

Context

You are designing a services provider application, but you need different types of services in different domains. In addition, not all of the services are foreseen at the time the application is created. You want your system to be extensible during its lifetime, and to easily support updates. New services will continuously be added responding to new requirements. Realizing a service provider will soon be insufficient because of the continuously increasing needs. On another hand, the extensions and updates of the services built on the system must not affect the system's users. These users do not want to redo their work each time the provider's developers make changes to the system.

Problem

How do you design such an application in a way to ensure the flexibility and extensibility of the system?

What could guarantee the harmony of the work between all actors of the application? They must stay independent from each other without affecting the performance of the system.

Solution

Conceive a generic application in such a way that it standardizes the handling of its services and the communication with users. This could be achieved by defining from one side, the protocol of communication with the users, and from the other side, a prototype of services definition. As for the generic service provider 'GenericServ' (Fig.5)

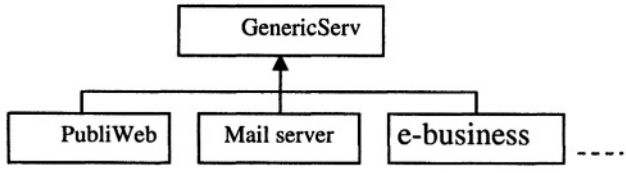

Figure 5. GenericServ, a generic web server

Known uses

In the domain of distributed applications development, the distributed object norms have produced a communication layer as in CORBA [8] and DCOM [5], in order to free the developers of such applications from the implementation of communication functionality. Another known use of this

pattern is libraries, like CASTOR [7], that illustrates a generic source generator offering services to map between Java objects and XML schemas.

Consequences

- Applying the Generic Server pattern makes the application more extensible and more flexible. The standardization of the services allows the developers work independently from the control and management of the services.
- The generic nature of this pattern makes it domain independent, so it will be applicable to more categories of applications.
- Future and unexpected needs could be added automatically thanks to the standard manipulation of the services.

Related patterns

The Generic Server could be structured using several architectural patterns like the Three Layered Architecture [18] and Pedestal [16]. But we suggest that it will be structured according to the 'Encapsulated modular layers' pattern explained below.

3.3 The Encapsulated Modular Layers pattern

Context

You want to conceive a generic application. But now you need to build a good architecture. This architecture should reinforce the genericity. You should keep the actors of the application independent from each other in order to maintain a generic domain independent system.

Problem

You must find the best way to realize the genericity of your system in a flexible architecture.

Solution

Create a four-layer architecture where you separate the processing core from the services offered by the application. Therefore, conceive each layer in a modular object-oriented structure. The layers must be opaque, so that they can evolve independently from each other. Simplify the interfaces between the layers in order to simplify modules updates.

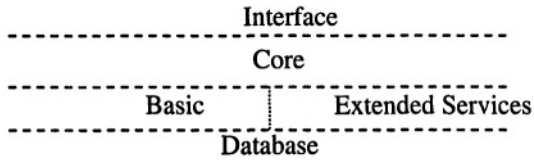

Figure 6. **Encapsulated modular layers structure**

The first layer contains the interface that is visible to users, and encapsulates all layers behind. The second layer is the core of the server. It includes general processing, like clients management and services control. In this layer, the generic aspect of the application will be defined, because it will contain service management and manipulation, but it is defined independently from their business domains. It defines a standard way to deal with services. The communication with the users and other layers is done via interfaces encapsulating its internal functionnality. So, it will be completely opaque to the user and to the other layers. A third layer that contains the services offered by the application is defined. It is divided into two modules: 'basic services' that are common to all domains (can be called domainless services), and 'extended' services with their utilities that are domain dependant. The two central layers are the common layers to all types of servers since they are domain independent. And finally, the last layer contains the database management (Fig.6).

Known uses

The Amoeba operating system [22] consists of a kernel providing basic services for processes and network communications, memory management and I/O services. But the difference is that in the 'Encapsulated opaque Layers' pattern, we restrict the services to have a predefined skeleton in order to generalize their management.

Consequences

- The opaque Layers with the encapsulating interfaces emphasize the reusability of the system.
- The evolutions of the four layers are independent from each other due to their opacity.

Related patterns

This pattern gives the development structure of an application of the type 'Generic Server'. The services offered by an application specifying the 'Encapsulated Opaque Layers' can follow the structure of the Service Prototype pattern [24]. It is a specialization of 'Layered Architecture' pattern [19].

4. CONCLUSION

We have presented in this paper an approach to solve the issues encountered in web applications development that are related to the complexity of the new sophisticated functionalities needed in this space. We have focused on the web development domains since it involves a large variety of persons with very different programming expertise. So, we have proposed a generic tool for developing web applications. The architectural style used for this tool is a heterogeneous one that combines advantages gained from different basic styles. Then, based on this tool we have defined three patterns for business applications' development. The first proposes a service provider to avoid redundant functionalities and to facilitate the development. The second gives a generic aspect to the provider when used for several or evolving domains. And the third defines architecture for this type of tools emphasizing modularity, reuse and easy update.

REFERENCES

1. Abowd, G.; Bass, L.; Kazman, R.; & Webb, M. SAAM: A Method for Analyzing the Properties of Software Architectures, 81-90. Proceedings of the 16th International Conference on Software Engineering. Sorrento, Italy. CA: IEEE Computer Society Press, 1994.
2. Abramatic, J.F., Développement technique de l'Internet, W3C, http://www.w3c.org, 1999.
3. Aksit, M., Bergmans, L., Berg van den K., Broek, van den P., Rensink, A., Noutash, A., & Tekinerdogan, B., Towards Quality-Oriented Software Engineering, to be published in Software Architectures and Component Technology: The State of the Art in Research and Practice, M. Aksit (Ed.), Kluwer Academic Publishers, January 2000.
4. Bass, L., Clements, P., & Kazman, R. Software Architecture in Practice, Addison-Wesley 1998.
5. Brown, N. & Kindel, C., Distributed Computing Object Model Protocol – DCOM/1.0, www.grimes.demon.co.uk/DCOM/DCOMspec.htm, 2002
6. Castro, J. & Mylopoulos, J., Information Systems Analysis and Design, 2001.
7. castor.exolab.org, 2002.
8. Daniel, J., Au coeur de CORBA (avec Java), Vuibert 2000.
9. Enterprise Java Bean 2.1 Specification, java.sun.com, 2002.
10. Farly, J., Java Distributed Computing, O'REILLY, 1998.
11. Fowler, M.., Analysis Patterns: Reusable Object Models, Addison-Wesley, 1996.
12. Garlan, D., "An Introduction to Software Architecture," Advances in Software Engineering and Knowledge Engineering, Volume I, edited by V.Ambriola and G.Tortora, World Scientific Publishing Company, New Jersey, 1993.
13. http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html, 2002.
14. Meszaros, G., Archi Patterns: A process pattern language for defining architectures, Pattern Language Of Program Design conference, 1997.
15. The Online Computer Library Center, http://www.oclc.org, 2002.

16. Rubel, B., "Patterns for Generating a Layered Architecture", Pattern Language of Program Design, Vol.1, Addison Wesley, 1995.
17. Sadoski, D. & Comella-Dorda, S., Three Tier Software Architecture, URL : http://www.sei.cmu.edu/str/descriptions/, 2000.
18. Shaw, M., Making Choices: A Comparison of Styles for Software Architecture. IEEE Software 12, 6 27-41, November, 1995.
19. Shaw, M., 'Some Patterns for Software Architectures', Pattern Language Of Program Design, Addison Wesley, 1996.
20. Shaw, M. & Clements, P. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems, Proc. COMPSAC97, 1st Int'l Computer Software and Applications Conference, August, 1997.
21. Shaw, M. & Garlan, D. Software Architectures: Perspectives on an Emerging Discipline, Englewood Cliffs, NJ: Prentice-Hall, 1996.
22. Tanenbaum, A.S., Modern Operating Systems, Prentice Hall, 1992.
23. Tawbi, S. & Chebaro, B., GenericServ: A generic server for web application development, web requirements & e-services workshop of the 1st EURASIA conference for Advances in information & communication technology, workshop proceedings, Austrian computer society, 2002.
24. Tawbi, S. & Chebaro, B., Service Providers patterns for client/server applications. Poster in the proceedings of the ICEIS 2003, 5th International Conference On Enterprise Information Systems, IEEE computer society press, 2003.