ATG Press Appetizer

Volume 1, Number 2, December 2001

Important Note: The information, which is provided by this document, is devoted to the entire world for improving creativity, and therefore developing new job opportunities around the globe, in the field of computer science and engineering. So, no part of the proposed information is protected by any intellectual property right. However, the document itself, is copyrighted by Aria Technology Group. No part of this document could be published, in any form of media, without written permission of Aria Technology Group. This is a delayed publication. The actual release date is December 2004.

Globe Server

Farnad Laleh
Aria Technology Group

ABSTRACT

The popularity of a technology does not always help to its improvement. Among different technologies, application server is a good example. The Internet boom of the 1990s absorbed many attentions and resources to the development of more advanced application servers, which are the building blocks of the modern Internet. Since most servers on the Internet are information centric (versus computation centric), the result of those developments has been a big generation of standards for data and information representation. The standards by themselves, cannot improve the core technologies of application servers, either information or computation centric. In this paper, we present a software architecture for a typical application server which covers both information and computation centric paradigms. We call it *globe server* and the goal of this architecture is the improvement of application server core technology. It is a quite different concept from the standards for data and information representation.

1. INTRODUCTION

When a technology becomes popular and a growing demand backs it, introducing more advanced form of it in a minimum possible timeframe is inevitable and the natural trend of market driven technologies. If adequate basic research does not back that technology, it will be misdirected to a path that will be somehow dispraising if not an actual dead end. It is exactly what happened to application server technology in recent years. A historical outlook could help to understand this better.

The popularity of the Internet and its dominant data transmission protocol (HTTP) and information representation format (HTML), motivated the industry to develop new technologies, which were based on or derived from these two mainstreams. This derivation was not problematic by itself, but the problem came from confusing two strategies for these developments: *generality and universality*.

A concept (subject or object) is *general* if it is usable in different domains, in a same manner. For example, *byte* is a general concept in computer science and can be used in different computer science domains (artificial intelligence, computer graphics, databases, and many mores) in a same manner. But a concept is *universal* if it covers all aspects of a special domain while it would be useless in other domains. For example, Unicode is a universal format for text representation because it can represent text in all of languages (aspects) while it is useless in computer graphics or artificial intelligence.

Industrial improvement usually depends on the definition of new universal concepts not general ones. The reason is very clear: general concepts have different interpretations among different parties and it is hard and expensive to converge these interpretations into a single viewpoint, but universal concepts bring previous diverged concepts into a single viewpoint, which consequently make the future developments coherent and cost efficient.

HTML is a universal format for text and image representation. Its universality helped to its fast and growing popularity. This popularity motivated the industry to develop something that would be more universal than HTML. Due to the HTML popularity, there were many HTML-based tools around and for saving development cost and time, the industry speculated to reuse them in the new developing formats. So the new major formats, SGML and XML, were derived from HTML. Unfortunately, the lack of enough

basic research in the field cast these new formats as the general ones rather than the universal. However, XML is wrongfully introduced as a universal data format in some documents.

As a result, everyone can see divergence in the field. There are many vocabularies for XML, which try to standardize themselves as the future mainstreams. More interesting, we can see different viewpoint to the XML itself among academia and business sector. In business sector, XML is the base tool for service integration (leveraging XML-RPC, SOAP, and WSDL) while in academic world it is the base of the so-called Semantic Web (leveraging RDF and DAML+OIL) and autonomous machine-based activities on the net. In addition to these, the handling of large XML files has been remaining a great challenge.

In this paper, we establish a new universal software architecture for application servers, which can cover both of information and computation centric models. In section 2, we present the basic concept of the proposed architecture in an analogical explanation. In section 3, we present the exact technical model of the architecture. Finally, in section 4, we draw a roadmap for the future developments of globe server.

2. THE CONCEPT BEHIND OF GLOBE SERVER

In this section, we try to give a very simple analogical argument to introduce the concept of globe server. We suppose that the reader of this paper is familiar with the concepts like client-server architecture, two-tier, three-tier, and n-tier server architectures.

Today's application servers are much like a grocery. A client goes to a grocery for obtaining some stuffs. The grocer (presentation logic tier) is behind the counter and is in direct contact with the client and gets the client's order and gets back his/her required stuffs. The available stuffs (information materials) are categorized in the shelves (database) behind the grocer and are only accessible to him/her. If the grocery is small enough then the grocer can get the client's order and fetch the required stuffs from the shelves, lonely (a two-tier architecture). If the grocery is so large that one grocer cannot do the both jobs at a same time, then two grocers are needed (a three-tier architecture). The first is behind the counter and gets orders (gets back stuffs) from the client (presentation logic tier) and the second processes the orders and fetches the required stuffs from the shelves (business logic tier). If the grocery is much larger so that two grocers are still inadequate, then several grocers (an n-tier architecture) could share the jobs in their own manner. More than one are behind the counter and more than one process the orders and fetch the stuffs.

Now suppose that our client goes to a supermarket for obtaining the required stuffs. At a supermarket, our client can see all the shelves and stuffs and walk among them and select what he/she wants without ordering to another person. However, some salespersons may stand around the shelves to help the client in his/her decision and also control possible unauthorized access to the shelves. At the supermarket, the client has his/her logic (business and presentation logic) in walking among the shelves and accessing to the stuffs, while some supervisors may monitor this access. G-*server* (abbreviated form of globe server) is quite like a supermarket.

Technically speaking, when our client goes to a grocery and call the grocer, the client makes a *procedure call* with some parameters (his/her order) passes to a procedure. The grocer activity for processing the order, is the body of procedure and is not defined or affected by the client. In contrast, the client makes no procedure call at a supermarket; rather he/she does what it wants (at a controlled environment) to obtain the required stuffs. We call this *procedure definition*. So the key concept behind of g-server is procedure definition, which we explain it technically in the next section.

3. GLOBE SERVER ARCHITECTURE

Traditional application servers are the host of several procedures or services (as called these days), which are remotely callable by clients. The client job is to send some parameters to remote procedure(s) and gets back the result(s) instantly or after a period of time. These procedure calls vary from the simple HTTP Get

and Post to the more advanced SOAP and XML-RPC methods. In traditional application server architecture, it is the duty of server architects to predict the client needs and design the required procedures, accordingly. But as the volume of information on the server grows, it is very hard to predict the client needs and behavior if not impossible.

A recent approach is to design a group of different procedures (services) for different needs and then describe and introduce them to client using WSDL and UDDI. It is quite evident that this approach is expensive (because of the number of the procedures should be developed) and not efficient in highly dynamic or very large information spaces. Remembering the previous section, this approach is still like a grocery but a large one with a few grocers.

In g-server architecture, the server gives the ability of request description to its client. In this mechanism, which is called *remote procedure definition*, the client can describe for the server, the information it has, the information it needs, the format and style of the response, and a few other queries. It is performed employing a remote server-side scripting language that is called *globe server script language* or *g-script*, as abbreviated by us. G-server accepts the g-scripts that are generated (written) by client, so it can produce the exact response, at the right time, and in a customized form for its client.

We should now clarify two new concepts, the g-server architecture and the g-script language structure. The following figure shows the architectures of a traditional three-tier and a basic g-server. A few comparative notes would help to understand the g-server architecture.

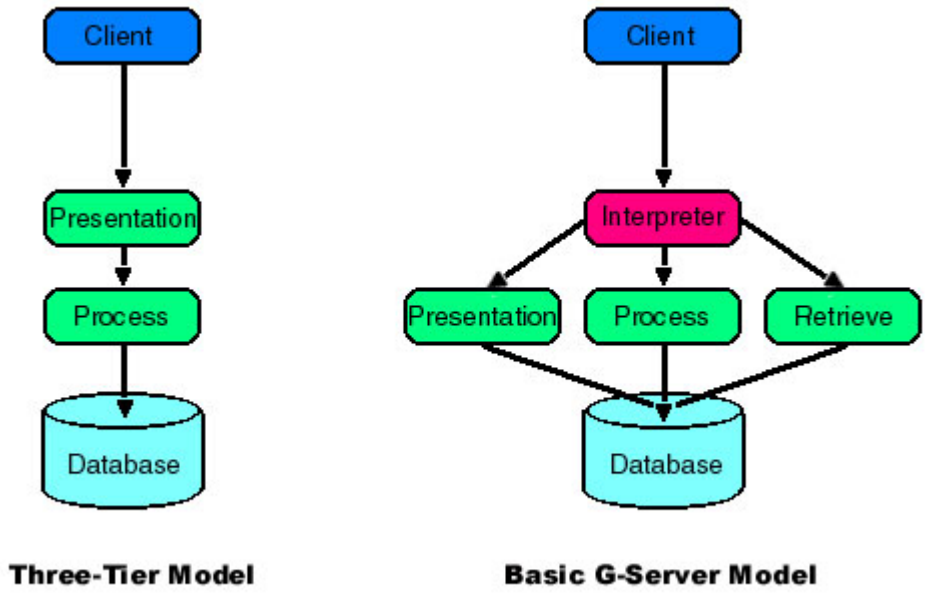

Fig. 1. Traditional three-tier architecture versus basic g-server architecture (the arrows would be bi-directional)

As can be seen, the traditional three-tier model has a serial architecture, i.e., the client is in touch with the presentation tier while the process logic (or business logic as stated these days) tier is an intermediate between the database and the presentation tier. The client can access the process logic only through a set of *design-time* embedded interfaces in the presentation tier. The process tier has the same role between the presentation tier and database.

In the basic g-server model, the presentation and process tier are not connected to each other serially. So in this architecture, we do not reckon them as tiers where they are *component packages*. Each component package consists of a library of different object-oriented components, which are recognizable for the interpreter and can be accessed by it.

The interpreter is in direct touch with the client and interprets the coming g-script from the client-side. Accordingly, the interpreter calls the required methods at the corresponding components to serve the client's request. There is a new component package in the basic g-server model that has not a corresponding tier in the traditional three-tier architecture. The retrieve component package is for searching among the existing information in the database and gives an extended ability to the client to retrieve its required information. This component package has a key role in large information centric servers while the role of the process component package is more important in computation centric servers.

In traditional application server architectures when a client makes a simple call (empty call) to the server, it gets back the required basic information from the presentation tier. But in g-server, client is not in direct touch with the presentation component package and should tell to the interpreter what it wants. This issue could be resolved by a very simple method, which is called *default scripting*. If a client sends an empty g-script to a g-server, the interpreter uses a few pre-defined and reserved g-scripts to serve the client.

The basic g-server as shown in figure 1, has three component packages. It should be mentioned that there are more complex models of the g-server that consists of more component packages. So we categorize the g-severs into three main models: basic, standard, and advanced.

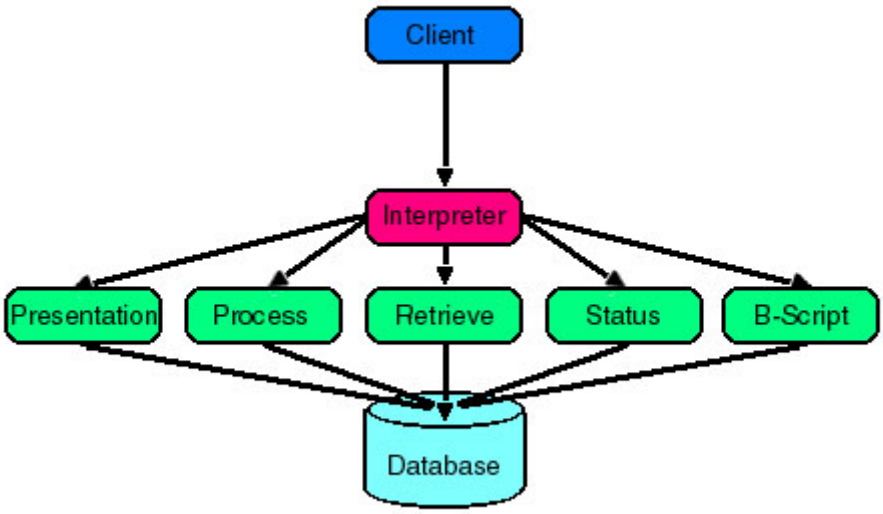

Fig. 2. Advanced g-server architecture (the arrows would be bi-directional)

Here is how the g-server is categorized in terms of its component packages. The minimum number of component packages for a g-server is three, which is its basic model.

1. The basic model includes the presentation, process, and retrieve component packages.
2. The standard model includes the basic g-server model plus the status component package.
3. The advanced model includes the standard g-server model plus the b-script component package.

There are two new component packages, see figure 2, which should be explained here. The status component package gives the ability of inter-relationship between the clients who use a same g-server. It includes some status flags (simple or complex string-based flags) that are set and read by the client, but under a predefined authorization policy. This inter-relationship has many applications in collaborative or competitive environments, e.g., financial markets, game nets, and many mores.

The b-script or *backward script* component package is a more complex concept. Backward script is generated in the server-side and implemented on the client-side. It should not be mistaken with the former familiar concepts like Java and VB scripts. B-script is the key concept behind of *globe client*, which is the subject of the next paper of our Appetizer series. So we do not explain it more in this paper except this useful note that b-script is the main road for autonomous and intelligent machine based activities on the net. In fact, b-script in conjunction with g-script can cover all of the goals of the Semantic Web.

4. ROADMAP FOR GLOBE SERVER DEVELOPERS

As g-script is a unique way of communication between different clients and g-severs, it should be standardized while the component packages and the interpreter can be developed on different platforms. The interfaces between the interpreter and component packages may or may not be standardized. It is a development strategy and is not a universal issue.

The only universal issue that should be addressed is the standardization of g-script. This is one of the two goals of ATG Caspian Hours Project. In this project, we try to establish a universal formats for g-script in collaboration with other institutions. In another part of the project, we do the same thing for b-script.

In addition to the powerful technical aspects of the g-server, which covered in the previous section, it is really interesting to mention that it has many useful applications in business. A party who holds a g-server can sell some g-scripts to the third-party clients for some special services. For example, a stock brokerage company can sell some special g-scripts to its best clients for giving them some special offers, whenever they want. In fact, a g-script can be customized for a client according to its character. This customization is a business itself!

Finally, it should be mentioned that the XML-based technologies, as the general technologies, may be used in g-server related developments. The only criteria for doing (or not doing) this, is the efficiency.

REFERENCES

It should be mentioned that most papers of our Appetizer series are reference-less. As our Appetizer papers are more a critique of current industrial trends, we eliminate the references for not pointing to a few names in the industry as devil. We also suppose that the reader of these series is among the industrial activists.