ATG Press Appetizer

Volume 1, Number 3, December 2002

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 Client

Farnad Laleh
Aria Technology Group

ABSTRACT

Current application client technologies (both of thin and rich) are immature when compared to application server technologies. The immaturity is originated in different technical and commercial issues. Technically, application clients have not strong and universal definitions and standards (e.g. something like HTML) as application servers have. Commercially, business sector represents itself through its application servers, so the development efforts are diverted to the server-side projects. From the engineering point of view, since the overall performance on a network depends on both of client and server, the client-side developments are as important as the server sides. In this paper, we try to categorize technically plausible application clients into a few comprehensive classes and put a definition upon each class. Afterward, we show a way in which each class can be standardized for the future coherent developments. Finally, we highlight the classes that have a remarkable potential for the current and the future commercial applications.

1. INTRODUCTION

The concept of application client is tied with the concept of distributed computing systems. In traditional distributed computing, application client is an application program that uses the resources of both the underlying computer and other accessible computers on a network. In recent years, this concept is referred to as the *rich application client.*

The advent of the Internet and its explosive growth pushed the usage of a new group of clients. These clients were targeted to show the information which was cast in HTML format, i.e., the dominant information representation format on the Internet. Since the HTML-based representations could not exploit many features of the ready-to-use application program interfaces (APIs) on the client-side, some new technologies (like the Java virtual machine) were added to the proposed clients. This could help more usage of the client-side resources but is still far from the optimal usage of the proposed resources. So this group of client is referred to as the *browser-based application client* or the *thin application client.*

This traditional categorization of application clients is apparently based on the usage degree of the resources on the client underlying system. This sort of categorization can only address the designing of *development tools* for application clients not the development itself. In fact, it does not draw a roadmap for the development of different possible application clients and leaves it as a quite general issue to the developer.

We think that application clients could be categorized based on their own functions. This categorization not only opens some new ways for the universal development of application clients, but also establishes a strong base for the development of autonomous and intelligent application clients. The recent issue (intelligent clients) has been followed in different forms in the past 10 years. We can point to agent-based systems and the Semantic Web as the two major efforts toward this, but both of them have been remaining as the general concepts (see Globe Server in Appetizer series) rather than the universal ones.

In this paper, we categorize application clients based on their functions, in section 2. Afterward in section 3, we present a universal software architecture for application clients. This universal model, which is called *globe client*, covers all of the categorized classes. Finally, we draw a roadmap for the future developments of the globe client and highlight its commercial applications in section 4.

2. CATEGORIZATION OF APPLICATION CLIENTS

There are different types of application client around for years, ranging from the older FTP and SMTP clients to the recent video conferencing and messenger systems. Although these clients seem substantially different from the application point of view, but there are based on a few basic functions from the engineering point of view. We use these functions as criteria for our categorization as the following:

1. **Information representation:** client represents to its user the information which are presented by server. For example, a Web browser represents images and text, which are cast by the server in an HTML file.
2. **File transfer:** client sends and receives file to/from server. In addition, we put video and audio streaming into this category. FTP clients are a simple example for this category.
3. **Message transfer:** client sends and receives message to/from server. The messages could be text, voice, and video. Chat clients are a good example for this category.
4. **Information process:** client processes the information it receives from server and produces a result, accordingly. The result may be sent back to server or used by the client itself. This is the most general category and entails many activities from simple form processing to the more complex scientific computing and artificial intelligence procedures.
5. **Network traversal:** client may traverse the network space for finding its required resources on a real or a virtual server. A virtual server is a client that can share some resources with a limited number of other clients. This category is not widely in use these days, but it is an important part in the future applications, especially in aggregated and intelligent environments.

It should be mentioned that the user interface of a client is not considered as a function of it, i.e. we can change the client user interface while its function does not change. In this paper, what we call as a *client function* is one of the above five categories. A client can have more than one function or may have all of the five functions at once. In addition, these functions may have inter-relationship inside a client. *Globe client* is an application client that contains at least three of the above functions. In the next section, we explain its architecture and its different models.

3. GLOBE CLIENT ARCHITECTURE

Currently, we are familiar with the possible functions of a typical application client, so we can propose a universal software architecture for it. As it was mentioned before, we do not consider user interface as a part of the client functions, although it is a part of client architecture. In addition to the well-known concept of user interface, g-client (abbreviated form of globe client) uses a new sort of interface, which we call it *server interface*. This new interface is a powerful technology that allows a server to talk to its client in order to help it for performing its functions.
A simple example would clear the concept of server interface. Suppose that a client represents to its user an identification form it receives from a server. The user should fill and send it to the server but forgets to fill it completely and sends it incompletely. In traditional client-server applications, the server returns an error message to the client, which is represented consequently to the user. Employing the server interface technology, the server tells to the client the incomplete part of the form and asks if the client itself can complete it for the server. If the client knows the incomplete part and has the authorization for sending it to the third party, then it completes it for the server without the user interference. If not, it shows an error message to the user.

The server interface technology is the base of automatic client activities. We should mention that globe client is substantially different from *intelligent client*. An intelligent client is based on some inference

engines and may do some functions in its favor. Contrary, a g-client does exactly the functions that its server asks it and these functions are toward the user requests and policies. G-client does not use inference and other AI-based mechanism in its server interface technology while it can use these methods as a part of its information processing (the 4th category) function.

The key concept behind of server interface is the *remote procedure definition*, which we explained it before in Globe Server (see Appetizer series). We use this concept same as the way we use it in g-server. The only difference is the substitution of client and server, i.e., g-client accepts remote scripts that are generated by server so it can produce the exact response, at the right time, and in a customized form for its server. Since this sort of script comes from the server-side after the client request to server, we call it *backward script* or *b-script*. The following picture shows the architecture of g-client.

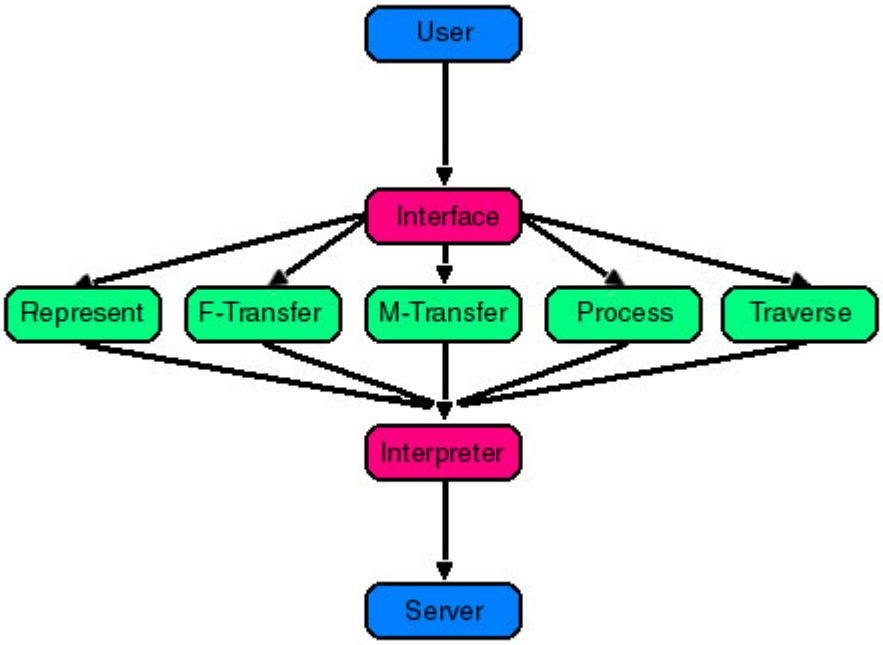

Fig. 1. Advanced g-client architecture (the arrows would be bi-directional)

The picture shows that the user is in direct touch with the interface (user interface) as the server is with the interpreter. The interpreter has the role of server interface and interprets the coming b-script from the server-side. There are five *component packages*, which correspond to the five mentioned functions in the previous section. Each component package consists of a library of different object-oriented components, which are recognizable and accessible by both of the interpreter and the interface. The represent, f-transfer, m-transfer, process, and traverse component packages correspond to the information representation, file transfer, message transfer, information process, and network traversal client functions, respectively. We categorize the g-client into three main models (as we did for the g-server before) as the following:

1. The basic model includes the represent, f-transfer, and m-transfer component packages.
2. The standard model includes the basic g-client model plus the process component package.
3. The advanced model includes the standard g-client model plus the traverse component package.

If you remember the advanced g-server architecture, it has a component package named b-script, which is supposed to generate the required b-scripts for g-client. Contrary to this, we have no corresponding components package in the g-client for generating g-script. It does not mean that g-client cannot generate any g-script, rather it does it using the process component package and the generated g-script is based on the result(s) of information processing function.

Another important issue which should be addressed is the position of former technologies like the Java Virtual Machine (JVM) in g-client architecture. Fortunately, their positions are very clear. These technologies are embedded in their corresponding component packages, e.g., the JVM is embedded in the represent component package. Again, a concept like the Java script should not be confused with the concept of b-script. They are in two different separate layers.

4. ROADMAP FOR GLOBE CLIENT DEVELOPERS

As b-script is a unique way of communication between different servers and g-clients, 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 b-script. This is one of the two goals of ATG Caspian Hours Project. In this project, we try to establish a universal formats for b-script in collaboration with other institutions. In another part of the project, we do the same thing for g-script.

In addition to the powerful technical aspects of the g-client, which covered in the previous section, it is really interesting to mention that it has many useful applications in business. For example, a consulting firm can prepare different b-scripts for its different clients. The b-scripts assist the clients in interaction with the firms. In addition, the firm can give some information about other firms to its clients through the b-scripts.

Suppose that a client decides to buy 10,000 barrels of 40-degree oil from its long time partner, broker A, so the client (using its g-client) sends a request (g-script) to the broker g-server. Since the broker has not more than 6,000 barrels available for now, it sends the address of broker B and C g-servers along with an estimated price through a b-script to the g-client to find the remaining 4,000 barrels at the most competitive price. Afterward, the g-client sends two separate g-scripts to the broker B and C g-servers according to the received b-script from broker A. If one or both of the B and C offer a price bellow the estimated price of broker A, the g-client reports the situation to its user to decide for a buy.

Scenarios like the above are the major goal of the Semantic Web and the agent-based system researchers. While these fields highly depend on artificial intelligence and other bottlenecks in computer science, we can reach the same goal with the old-fashion and easy to use scripting systems. People sometimes come to trouble, because of not facing a hard problem, but not knowing what is the problem!

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.