WPM 14.7

A JAVA BASED MULTI-TIER ARCHITECTURE FOR ENTERPRISE COMPUTING: A CASE STUDY FROM AN UNIVERSITY ACADEMIC INFORMATION SYSTEM

Shiuh-Sheng Yu[1,2] and Wen-Chin Chen[2]
[1]Graduate Institute of Information Management
National ChiNan University, Puli, Nantou, Taiwan 545, R.O.C.
[2]Communications & Multimedia Laboratory
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan 106, R.O.C.

ABSTRACT

We present in this paper a Java based multi-tier architecture for enterprise computing. The architecture has been used successfully in developing the academic information system of the National ChiNan University. The system shows that the functionality, performance, and total cost of the proposed architecture are all quite well for enterprise application development. For performance of the system, its response time is 4 times faster than the IDC version, and 6 times faster than the CGI version using Visual Basic.

1. INTRODUCTION

The booming Internet industry has made current WWW a major channel for publishing enterprise information. Most of the early implementations of the WWW publishing are done using the CGI technique. However, due to the poor performance and the lacks of interaction, the CGI is rarely adopted in crucial enterprise applications. Instead, the Java language, with its ability to build cross-platform business logic into the browser, gives WWW better performance to meet the needs of enterprise application [1]. We present in this paper a Java-based multi-tier architecture for intranet/enterprise application. The architecture is used to develop a university academic information system using JDBC (Java Database Connectivity) and RMI (Remote Method Invocation) [2].

2. SYSTEM ARCHITECTURE AND IMPLEMENTATION

Our study aims at designing and implementing an academic information system to be used by the university to handle all daily academic transactions [3]. The development results in a Java based 3-tier architecture shown in Figure 1. Under this architecture, once a client is connected to the system, the authenticator first checks the user's account and password. If the password is valid, it then creates a corresponding user object to serve the client. Instead of building a database connection for each user object, all SQL commands are passed through the database connection pool. This method has the following advantages over the CGI approach:

- Java applet has the full power of programming language to build complex interactive applications;
- RMI can pass and return any Java objects instead of just string as in CGI;
- RMI server uses thread to serve incoming request;
- UI updates are handled by applet;
- The traffic between client and middleware is further reduced;
- The pool can pre-allocate database connections to reduce response time;
- The pool monitors the status of database connections for error recovery;
- The pool increases the utilization of database connections, thus reduces the license fee.

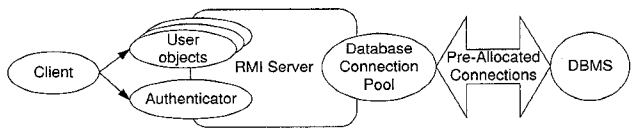

Figure 1. The architecture of an university academic information system.

Currently, our system provides the following features:

- Query the information of the courses opened for current and previous semesters;
- Maintain personal information and password;
- A student can select and drop courses, view transcripts and curriculum schedules;
- A teacher can query the information of the students who have selected his/her courses, and input scores and syllabus.

3. PERFORMANCE ANALYSIS

Performance is the most critical concern of current Java implementation. To compare the performance between CGI, ISAPI and our system, we setup two servers for DBMS and RMI servers with the configurations shown in Table 1.

Our database is about 5MB in SQL Server. The RMI Server has a footprint of 4.9MB. Three different types of

0-7803-4357-3/98/ $10.00 © 1998 IEEE

queries are used to evaluate the system performance. Query type A includes a SQL select command and three prepared statements each executes 19 times. The returned result set is about 1.5KB. Query type B has a SQL select command and a result set of 2KB. Query type C has a result set about 100 Bytes, and has been cached by RMI server. Table 2 shows the response times and the CPU times spent in each tier.

CPU	Pentium 133
RAM	64MB
Hard disk	EIDE 4.0GB*1
Network adapter	10Base-T Ethernet
OS	Windows NT 4.0
DBMS	SQL Server 6.5
Language	JDK1.1.4 + JPP1.1.4
WWW server	IIS 3.0
Database connection software	JDBC-ODBC Bridge

Table 1. Server configuration.

Query	Response Time	Database CPU Time	RMI Server CPU Time	Client CPU Time
A	530ms	42%=223ms	49%=260ms	10%=53ms
B	80ms	6%=4.8ms	49%=39ms	48%=38ms
C	12ms	0%	32%=3.8ms	62%=7.4ms

Table 2. Response and CPU times of different queries.

Query C shows that the ratio of the CPU time spent for RMI marshaling to that of de-marshaling is about 1:2. If we ignore the marshaling overhead in query A, the middle tier spent 235ms. This time cost includes a 60 ms overhead of fixing a bug of storing non-ASCII code to database in JDK1.1. Without the bug, query A should have spent 175ms. We implemented an optimized Visual C++ 5.0 version of query A, and found that it costs 70ms (ignoring the time spent for process creation and database connection setup). The 105ms difference contributes 30% of the response time.

Compared to the other two systems, our system's response time is 4 times (2190 ms vs. 530ms) faster than the IDC version, and 6 times (3320ms vs. 530ms) faster than the CGI version using Visual Basic[4]. The reasons for our system to outperform the others two can be explained as follows:

- The HTML based systems require the middle tier to generate HTML files for user interface;
- Without database connection pool, every client request results in new database connections. Our experiments showed that SQL Server spends 90ms in connection setup;
- IDC limits the optimization of SQL commands, such as using prepared statements;
- IDC is an interpret language, which is even slower than Java;
- Process creation costs several hundreds milliseconds on Windows NT.

4. DEVELOPMENT COST

We spent three man-month to develop the whole system. The system consists of about 5500 lines of Java code. The cost of the deployment is quite low, since the only work is to setup two Windows NT servers. We estimate that the total cost of the system is about $10,000 to $20,000 US dollars, including the server hardware and software cost.

As the cost of hardware and software package is expected to be further reduced in the future, the percentage of the cost spent on software development in enterprise information system will become higher. In our system, half of the investment has been spent on the programming cost. With the promise of "Write Once Run Anywhere" provided by Java language to reduce the maintenance cost, it is possible to build a very low cost enterprise information system using our architecture.

5. CONCLUSIONS

Our implementation shows that a Java based 3-tier architecture is adequate for developing the high performance, low cost, fast deployment, and easy to use enterprise information systems. We expect that the new Java virtual machine with JDK1.2 will further boost up our system's performance and make our architecture more promising for developing intranet/enterprise applications.

REFERENCES

[1] P. Ram, R. Abarbanel, "Enterprise Computing: The Java Factor," *IEEE Computer*, June 1997, pp. 115-117.

[2] A. Wollrath, J. Waldo, R. Riggs, "Java-Centric Distributed Computing," *IEEE Micro*, May/June 1997, pp. 44-53.

[3] *http://admwww1.cc.ncnu.edu.tw.*

[4] F.-P. Lai etc. "The integration of enterprise information—Intranet," *Communications of IICM*, vol. 1, No. 2, April 1997, pp. 43-54.