Evaluating Distributed Graphical User Interface Communication

Thomas Tilley [HREF1], School of Information Technology, Griffith University, Parklands Drive, Southport, QLD 4214, Australia. T.Tilley@mailbox.gu.edu.au

Simon Pollitt [HREF2], Department of Computer Science, The University of Adelaide, North Terrace, Adelaide, SA 5005, Australia. sepollit@gisca.edu.au

Peter Eklund [HREF3], School of Information Technology, Griffith University, Parklands Drive, Southport, QLD 4214, Australia. P.Eklund@mailbox.gu.edu.au

Keywords

World Wide Web, GUI, Java, CORBA, client/server, sockets, distributed object

Abstract

Two main options exist to implement an Internet front-end for an application with a graphical user interface (GUI). The first is to "hard-code" a remote interface specifically for the application and distribute it to client computers. The problem with this approach is the inflexibility of the resulting interface. An alternative approach that minimises the need for client re-distribution is to provide a GUI framework from which an application's interface can be built dynamically. This paper presents experimental results arising from a project to implement a distributed platform-independent GUI framework.

Introduction

This paper presents experimental results arising from a project to provide a distributed platform-independent graphical user interface (GUI). Two main options exist for the implementation of an Internet front-end for applications with GUIs. The first is to ‘‘hard-code’’ a remote interface specifically for the application and distribute it to client computers. The problem with this approach is the inflexibility of the resulting interface. If the application is modified then the GUI software needs to be changed and re-deployed to clients. World Wide Web based deployment options include: FTP or HTTP down-loading of the client software; using "push" technology to update clients; and distribution of the front-end as an applet.

An alternative approach which minimises the need for client re-distribution is to provide a GUI framework (Schmidt and Fayad 1997a; Schmidt and Fayad 1997b). Using the framework an application's interface can be built dynamically directly from the application server. If the appropriate set of GUI components are implemented on the client the remote GUI can provide the same ''look and feel'' as the program running locally on the application server. An additional advantage is that changes to the back-end (server-side) application do not necessitate modification to the front-end (client-side). Any re-arrangement of the application's user interface (UI) is reflected dynamically by the remote client.

Implementing a Distributed Platform-Independent GUI

"Distributed computing" typically refers to the transparent distribution of processes or data over an unspecified number of machines. In this paper the term "distributed" indicates the simple separation of an application and its user interface via the Internet. This two-tier client-server approach has obvious parallels with both the X-Windows System and the "thin client" Network PC paradigm.

The server-side application for the project was a Spatial Database Management System (SDBMS) providing real-time 3D data visualisation. The SDBMS was implemented in C++ on the Silicon Graphics IRIX platform. The application's UI was built using the Qt GUI toolkit [HREF4]. One important aspect of the project was to provide platform independent access to the SDBMS while presenting users with the products standard "look and feel". A Silicon Graphics server could render the applications GUI on a Windows NT machine, for example. This platform independence requirement made Java an obvious client implementation choice. While Java is only as portable as the platforms that support it (Curtin 1998), this now represents a large number of platforms. The incorporation of the Java Virtual Machine (VM) into Web browsers also provides a convenient distribution mechanism for Java applets.

The Swing GUI toolkit from the Java Foundation Classes [HREF6] was used to implement the GUI components, known as widgets, on the client. Swing was chosen for two reasons. Firstly, because Swing is implemented in Java it met the project's portability requirement. Secondly, it mirrors the functionality of the Qt toolkit used by the target application.

Distributed Computing

Providing front-ends for legacy systems presents a number of engineering challenges. One of these challenges is how to best integrate components written in different languages. In this project the need for a Java client to interact with a server application written in C++ limited the available communication options. Four implementation techniques were explored. The first was to communicate directly between Java and C++ using BSD style sockets.

Sockets provide a fast, low-level, file-like means of communication between processes or machines. Transmitting data via a socket is analogous with writing to a file and the reception of data from a socket with file reading. This character based approach means that complex objects must be "flattened" or serialised before they can be sent "over the wire". The efficiency of the serialisation affects the length of the resulting representation - typically a string. The longer the string the longer it takes to transmit. Serialisation therefore has some bearing on the time taken to transmit an object via sockets. There is also an associated engineering cost in terms of the design and implementation of a suitably complex and robust communication protocol (Waldo et al. 1994). Communication between different languages must also provide compatibility and consistency between base types such as integers.

One obvious way to solve the type compatibility problem is to use Java sockets for communication on both the client and the server. This was the second technique explored. While this technique increases the complexity of the server and appears to merely shift the location of the problem it actually facilitates a solution. Using the Java Native Interface (JNI) [HREF7] Java programs can communicate with C and C++ via shared libraries. JNI provides type mappings and methods for inter-operation between the languages. This resolves the compatibility problem at least for simple types. Java sockets provide the client-server communication while JNI handles Java/C++ interaction within the server.

The third technique used CORBA - the Common Object Request Broker Architecture (Vinoski 1997; Vogell and Duddy 1997; Mowbray and Malveau 1997). As with the first option it facilitates "direct" communication between the Java client and the C++ server. CORBA is a high-level distributed object technology that largely abstracts over protocol and serialisation issues. It provides base types for inter-language operation as well as the ability to handle exceptions. With the incorporation of CORBA object request brokers (ORBs) into Web browsers CORBA is an increasingly flexible deployment option.

The final implementation technique again required Java to run on both the client and server. While JNI was again used within the server to facilitate Java/C++ interaction the distributed communication was implemented using Java's Remote Method Invocation (RMI) [HREF8]. RMI is a high-level, CORBA-like technology providing communication between Java VMs running on different hosts.

Experimental Design

An experiment to evaluate the suitability of the four techniques described above was devised. The experiment is designed to simulate the transmission of "callbacks" between a remote GUI client and a server application. There are two main aims. The first was to measure the comparative performance in terms of string echo time for each of the communication techniques. The second aim is to gain insight into the engineering costs associated with implementing each method.

GUIs are assembled using sub-components called widgets. Widgets represent items such as buttons, sliders and menu items. In response to user actions widgets generate callbacks. For example a user clicking on a button generates a callback which then invokes the appropriate response in the application - perhaps to open a dialog box. A moderately complex GUI may incorporate hundreds of callbacks (Myers 1996). In a distributed GUI these callbacks need to be forwarded to the server.

Callbacks from some widgets can also contain parameters reflecting their current state. A slider's callback may for example pass an integer representing the current position of the slider. To simulate a simple single-parameter callback a randomly generated string was passed from a client to a server. This string was then echoed back to the client and the elapsed time was recorded. Simple implementations for each of the four techniques were written and the experiment was conducted. A program was used to generated test files containing random strings with lengths between 1 and 256 characters. Twelve files were generated - 10 containing 10 strings, one containing 20 strings and one composed of 40 strings. To remove the effect of disk input/output on the experiment these files were pre-loaded by each client into a convenient data-structure before transmission. The JNI server implementations were also required to convert the test string from Java to C++ and then back to Java before echoing it to the client. Network effects were reduced by running both the client and server on the same host in loop-back mode and connection establishment times were ignored.

The Java code was implemented using version 1.1.6 of the Java Development Kit for the IRIX platform. The CORBA code was implemented using Visibroker version 3.3 for Java and the IRIX version of Visibroker 3.2 for C++. The test machine was a Silicon Graphics Origin 200 running under IRIX 6.4.

Results and Interpretation

The results over 10 runs for each of the files containing 10 strings are presented in Table 1. The times are in mS and the results are the average over the 10 runs. The data exhibits few surprises however the results may be influenced by the choice of a particular VM or CORBA implementation.

Test	C++ Sockets	Java Sockets	CORBA	RMI
1	8.1	17.3	32.5	32.8
2	7.0	14.7	29.5	34.4
3	6.4	13.7	22.8	27.7
4	8.6	15.4	25.9	31.1
5	6.1	16.6	22.4	30.5
6	5.2	13.2	23.5	24.4
7	5.8	14.2	26.1	30.5
8	5.8	15.3	23.3	33.2
9	6.1	13.7	22.4	27.5
10	6.2	15.3	25.6	35.7

Table 1: Average echo times (mS) for 10 strings.

The Java to C++ sockets implementation provided the fastest echo times. With the additional JNI conversion overhead the Java to Java sockets took approximately twice as long. The CORBA implementation was approximately 4 times slower than the C++ sockets and RMI provided the slowest times overall. Test file 6 was the smallest of these files and file 1 was the largest. This is reflected in Figure 1. The results obtained are consistent with the expectation that smaller files take less time to transmit and larger files take more.

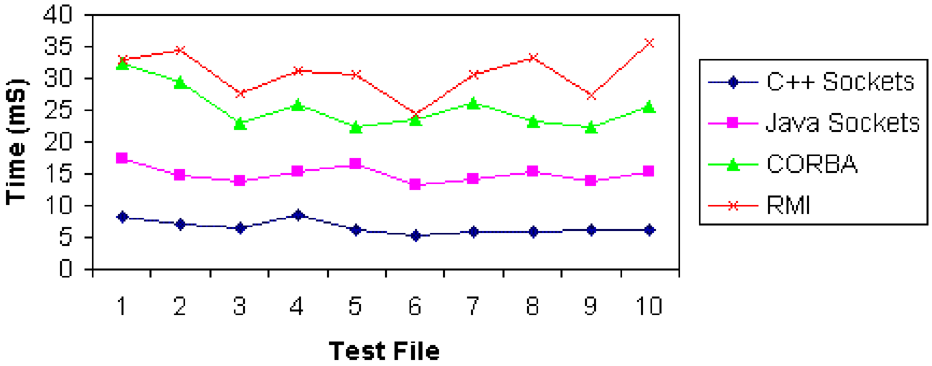

Figure1: Comparing the average echo times of the four implementations.

Table 2 represents the average echo times in mS over 10 runs for files containing 10, 20 and 40 strings. The data appears to indicate an approximately linear increase in the average echo times.

This can be seen in Figure 2. Some initial experiments with 10, 100 and 1000 strings also reinforce this observation.

Strings	C++ Sockets	Java Sockets	CORBA	RMI
10	8.1	17.3	32.5	32.8
20	12.5	35.7	55.9	61.6
40	27.3	59.9	113.4	117.4

Table 2: Average echo times (mS) for 10, 20 and 40 strings.

While these results provide a simple timing comparison for the implementations the other significant issue is the engineering cost of each. The distributed GUI project had strict time constraints imposed. Although sockets provide the fastest implementations the significant engineering cost to develop a robust protocol for a modestly complex GUI in the available time was too large. RMI and CORBA are high-level distributed object technologies that largely abstract over protocol and serialisation issues. While this comes at the expense of speed it simplifies some of the costs associated with implementing distributed systems (Schmidt and Fayad 1997a).

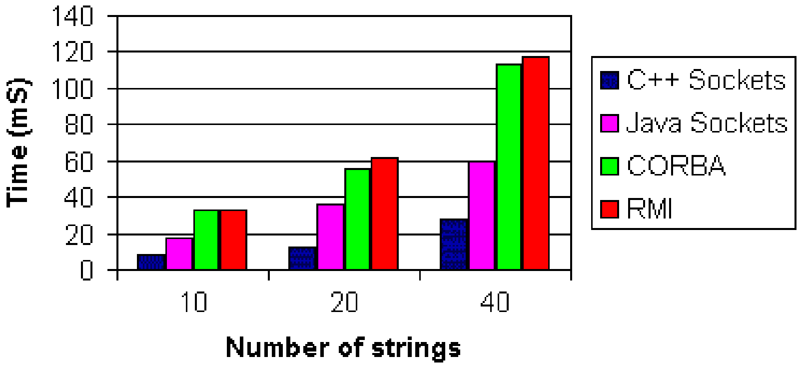

Figure2: The average echo times for 10, 20 and 40 strings.

Although the RMI method provided the slowest callback times it is simpler to use than CORBA and integrates transparently into Java code. RMI also automatically provides distributed garbage collection. This needs to be handled manually under CORBA and is an important consideration when dealing with C++. On the basis of its comparatively low engineering cost RMI was used to implement communication within the distributed GUI framework.

While the GUI framework itself did not implement remote data visualisation the experimental results suggest that an alternative communication method would be more suitable. Remote visualisation typically requires significant volumes of data to be transmitted. Speed is of the essence to reduce waiting time for the user and a less complex protocol is required (Shneiderman 1992). With a lower associated protocol engineering cost and a greater speed requirement a sockets based implementation would be more appropriate.

Conclusion

The choice of an appropriate communication mechanism for Web-deployed applications depends on a number of factors. RMI was used within the distributed GUI despite the experimental results obtained. The time available in which to implement a complex distributed communication mechanism made low engineering cost the most significant consideration.

With the inclusion of CORBA ORBs and Java VMs in Web browsers distributed object technologies are now widely accessible. The engineering advantages they provide, however, come at the cost of performance. Sockets based implementations provide the best performance but the cost of developing complex protocols has to be addressed.

References

Schmidt, D.C., & Fayad, M.E. (1997a). Lessons learned building reusable OO frameworks for distributed software. Communications of the ACM, 40(10), 85-87.

Schmidt, D.C., & Fayad, M.E. (1997b). Object-oriented application frameworks. Communications of the ACM, 40(10), 32-38.

Curtin, M. (1998). Write once, run anywhere: Why it matters [HREF5].

Vinoski, S. (1997). CORBA: Integrating diverse applications within distributed heterogeneous environments. IEEE Communications Magazine, 35(2), 46-55.

Vogell, A., & Duddy, K. (1997). Java programming with CORBA. New York: John Wiley and Sons.

Mowbray, T.J., & Malveau, R. (1997). CORBA design patterns. New York: John Wiley and Sons.

Waldo, J., Wyant, G., Wollrath, A., & Kendall, S. (1994). A note on distributed computing [HREF9]. (Report No. SMLI TR-94-29). Sun Microsystems Laboratories.

Myers, B.A. (1996). UIMSs, toolkits, interface builders. Handbook of UI Design.

Shneiderman, B. (1992). Designing the user interface: Strategies for effective human-computer interaction (2nd ed.). Reading, Mass.: Addison-Wesley.

Hypertext References

HREF1
: http://www.int.gu.edu.au/~ttilley/

HREF2
: http://www.gisca.adelaide.edu.au/~sepollit/

HREF3
: http://www.int.gu.edu.au/~peklund/

HREF4
: http://www.troll.no/qt/

HREF5
: http://www.java.sun.com/features/1998/01/wora.html

HREF6
: http://www.java.sun.com/products/jfc/

HREF7
http://java.sun.com:80/products/jdk/1.1/docs/guide/jni/spec/jniTOC.doc.html
HREF8
http://java.sun.com:80/products/jdk/rmi/index.html
HREF9
http://www.smli.com/techrep/1994/abstract-29.html

Copyright

[Proceedings]

AusWeb99*, Fifth Australian World Wide Web Conference, Southern Cross University, PO Box 157, Lismore NSW 2480, Australia Email: "AusWeb99@scu.edu.au"*