Client/Server Configuration in a Next Generation Internet Environment: End-User, Developer, and Service Provider Perspectives

Caspar Ryan and Stephen Perry

RMIT University, School of CS & IT
PO Box 71, Bundoora VIC, Australia, 3083
{caspar,sperry}@cs.rmit.edu.au

***Abstract*—This paper discusses variations in client/server configuration, for current and Next Generation Internet applications, from the perspective of end-users, application developers and service providers. An empirical study, which evaluates the performance of a typical Internet based business scenario, provides support for the argument that delegating functionality to clients offers a significant benefit to end users and service providers, thereby potentially justifying additional development effort.**

***Index Terms*— Client-Server, Developer, End-User, Next Generation Internet, Service Provider, XML, XSL**

I. Rationale

In traditional client-server systems, the majority of computation and data management is performed by the server, with a thin client providing a presentation layer to the user. This has occurred because client devices, especially those that are mobile or embedded, have lacked the computational power required to support client-side application services [1]. Furthermore, managing a centralised server is simpler and more convenient from an administrative perspective, than dealing with clients that provide application services.

However, with advancing miniaturisation and the improvement in communications technology, client device limitations are becoming less of a consideration when architecting the distribution of responsibility between client and server in a distributed application. For example, mobile devices such as Intel XScale based PDA's have CPU's running at hundreds of megahertz with 64MB or more of program memory. Current generation mobile phones are able to run distributed application code using standardised platforms such as Java 2 Micro Edition (J2ME) [2], Symbian OS [3], and .NET Compact Framework [4].

Nevertheless, the move to richer client functionality has been slow, with tension existing between the needs of various stakeholders such as application users, developers and service providers. This is exacerbated by the increasing diversity of the Next Generation Internet (NGI) [5], where devices are ever varying along a number of dimensions such as screen size, form factor, processing speed, permanent storage capacity, and power consumption.

This work is part of the research program of the Australian Telecommunications Cooperative Research Centre (ATCRC) http://www.telecommunications.crc.org.au.

Given this changing communication landscape, the rest of this paper discusses the resultant effect on the configuration of client-server applications. Sections II, III and IV begin by studying the perspective of the various stakeholders (user, service provider and developer), with section V providing support in the form of an empirical study of specific client-server configurations. Based on these findings, this paper goes on to argue that the benefits to application users and service providers are substantial enough that the increased development effort associated with heavier client solutions is justified in some circumstances. Consequently, further research should explore techniques for supporting the dynamic configuration of client-server applications, in order to utilise more effectively the emerging technologies of the Next Generation Internet.

II. Application Developer

Targeting a broad range of devices currently requires the use of a number of disparate APIs. For example even if a developer chooses a language such as Java that promises 'write once run anywhere' behaviour, the developer may use J2ME for a Java enabled phone such as the Nokia 7650; Java 2 Standard Edition (J2SE) [6] version 1.1 for a PDA such as a Compaq iPAQ, and J2SE 1.4 for a desktop machine. Furthermore, the same developer may use yet another Java technology, such as Java 2 Enterprise Edition (J2EE) [7], for the development of server side components [8].

In cases where a common platform such as Java is not available, the development effort can become even more fragmented, requiring a mix of programming languages, implementation platforms, and design paradigms.

Consequently, from an application developer's perspective, the thin client approach is simpler because only a single codebase is required for the server software, with only the user interface differing between application clients. In many cases, the use of a declarative user interface representation such as XHTML and WML can be achieved using style sheet transformation, thus localising and minimising the amount of customisation required for different types of client [9]. Thin client development effort can be further reduced if dedicated commercial solutions such as the Bullant JRAP environment [10] are used.

III. Application or Internet Service Provider

If we consider Internet Service Provision[1] to be the delivery of network infrastructure and data bandwidth, then introducing different client/server configurations does not have dramatic consequences. This is so, because the amount of data transferred between the client and server, in thin, fat or medium configurations, is largely application dependent.

However, from the perspective of the Application Service Provider (ASP), who must provide hardware and software infrastructure to host such applications, the effect can be substantial. With the growing convergence of the public switched telephone network (PSTN) and Internet, there exists the potential for a much larger pool of clients [5]. Assuming the currently dominant model of 'big iron' servers and thin clients, a rapid increase in clients necessitates a proportional increase is server power. This in turn poses scalability and cost issues, which must eventually be passed on to the service user.

There is also the issue of system administration, which is again likely to be the responsibility of the ASP and thus any possible cost savings to be had by delegating processing responsibility to the client must factor in any potential increase in system administration cost. Nevertheless, from the perspective of the ASP, any technology that effectively offloads processing to the client is likely to be welcomed.

IV. Application End User

Since end-user acceptance is the ultimate deciding factor in application success [11], the impact of any application development or deployment decision must also be considered from the perspective of the application user.

Doing so finds that more fully utilising client side resources has a number of potential benefits in terms of perceived user experience, provided these factors are considered at design time.

Firstly, client side performance is more consistent and immediate, particularly in increasingly prevalent wireless environments where variations in network conditions substantially affect the performance of thin client applications that must connect to the server for every user interaction [12].

Secondly, applications located on the client have the ability to provide a richer and more responsive user interface. For example in the case of Java, a J2ME midlet using MIDP UI components, and especially a browser based Java Applet using the Swing Toolkit [13] can have a functionally richer set of user interface components and modes of user interaction than can be achieved with a mark-up language such as WML or XHTML.

In light of the above two points, it appears that economic and administrative concerns are driving application development as much as the desire to provide a responsive and highly useable client interface. As a result, the end user has much to gain if a more flexible approach to client/server configuration is taken.

[1] This paper distinguishes *Internet service providers* (ISPs), who provide basic network infrastructure, from *application service providers* (ASPs) who provide specific hardware and software to host distributed applications. In some cases, the ISP and ASP may be the same commercial entity.

V. Empirical Testing of Different Client/Server Configurations

A. Introduction

Although the benefits of utilising client resources are immediately compelling from the perspective of the end-user (section IV), the perspective of the application service provider (section III) warrants further examination. The case in point is whether the potential benefits offset the additional work required to develop heavier client applications targeting multiple devices in a heterogeneous networking environment. Consequently, this section describes a series of laboratory tests that were designed and executed to simulate the performance of a typical business application use-case in a number of specific client/server configurations.

In order to be broadly representative of typical business applications, the custom client and server programs, which were developed specifically for this study, perform an XML transformation process using XSLT to provide presentation output to the user [9]. Not only is such a technique relevant to current generation internet applications which transform application specific XML data into XHTML [14], but also future NGI applications since emerging presentation standards such as SVG [15] and XSL-FO [16] are also based upon XML transformation.

The execution process used by the test applications is as follows: 1) The client connects and makes a request to the web server for some data 2) The web server retrieves some data in an intermediate XML format that satisfies the request[2] 3) The data is transformed for client presentation in one of two ways:

Thin Client - The server applies a cached style sheet, appropriate for the client type that made the request; performs an XML transformation using a suitable engine, and sends the result of the transformation as marked up text for presentation on the client.

Medium Client - The server retrieves a style sheet appropriate for the client type that made the request, and sends the stylesheet, along with the source XML data, to the client for transformation. The client then performs the transformation locally and presents the transformed output to the user.

Note that the client is designated *medium* since it does not provide a local user interface (since it is still using a mark-up based presentation style), but does perform substantial processing in order to complete the XML transformation and thus cannot be considered a thin client.

B. Configuration

The server consisted of a Java Servlet running on Apache Tomcat version 4.1 [17], deployed on a 2GHZ Pentium 4 machine with 512MB RAM. Both client programs were also

[2] The origin of this data is assumed to be an external application server or enterprise information system (EIS), as would generally be the case in large scale applications such as those using J2EE.

written in Java using J2SE 1.4, and issued requests to the servlet using a standard HTTP GET request. Both client- and server-side parsing was done using the Java API for XML Processing (JAXP) API provided with J2SE 1.4, in order to provide identical client and server conditions.

Tomcat was configured for a maximum of 50 concurrent threads, which subsequently limited the number of simultaneous client requests to 50, since Tomcat can only process a single connection per thread. In production systems, this type of tuning parameter is usually set based on server capability and application behaviour. Tuning the test application on the aforementioned hardware found that running too many threads adversely affected application performance by introducing various exceptions, particularly when Tomcat was forced to perform a large number of concurrent XML transformations.

C. Methodology

Nine different data sets were used for testing, with *small*, *medium* and *large* variations for each of the following components of an XML transformation: XML source file, XSL stylesheet file, and transformation output. This resulted in 27 possible test combinations, allowing the analysis of performance under a wide range of application scenarios. The data sizes for the small, medium and large XML files were < 1kB, ~20kB, and > 100kB respectively.

For completeness, all size combinations were attempted, even though some appeared unlikely in practice, such as the case where the output produced by the transformation was larger than the source XML and XSL files combined. Achieving this required the repetition of XML source data, using computationally expensive recursive template matching in the XSL stylesheet. Thus, when testing the scenario of a small XML file, no medium or large output could be created without the XML parser running out of memory due to the extensive recursion required. Consequently, only 25 of the theoretically possible 27 combinations were executed.

To provide the final test suite, the 25 test scenarios were executed at both 50 and 500 simultaneous server requests to represent different load conditions, and in both client- and server-side transformation modes.

Since Tomcat's maximum concurrent thread count had been set to 50, 50 simultaneous requests caused little request queuing at the server end, whereas 500 requests meant that all requests following the first 50 were distributed to the individual request queues of each server thread. Since Tomcat processes queued requests sequentially, testing with larger numbers of simultaneous requests gave relatively linear increases in response time, providing no further information.

When performing client-side XML transformation, two requests were made for each client connection, since both the XML source file and XSL stylesheet were needed locally. This differed from the server-side transform client, which only requested the single transformed output stream. Note that this approach simulates the worst-case scenario of a traditional web client, since in practice both files could be marshalled into a single response. Furthermore, as only a single test client was used to launch all of the requests, XML processing could not actually take place on the client. Instead, transformation times for each scenario were pre-recorded and added to the average time taken to process the client side transformation requests.
The output variables of interest were server-side CPU usage measured using the Microsoft Management Console's Performance Monitor [18], and average time per client request using internal instrumentation in the client code. These results are summarised in tabular form and discussed further in the following sections.

D. Results

Fig. 1, Fig. 3 and Fig. 4 show the average client request times in milliseconds for combinations of small, medium and large XML source files respectively. Each figure provides separate tabular results for 50 and 500 simultaneous requests. Additionally, within each table, results for both client and server XML transformation times are given.

Fig. 2 shows the server CPU usage for a typical request requiring XML transformation of a large source file, using a large XSL style sheet producing a large output.

E. Discussion

The overall results of Fig. 1, Fig. 3 and Fig. 4 show that in most cases, client-side XML processing yields lower average request times, in some cases up to a third of the equivalent server side transformation. Furthermore, Fig. 2 shows that server-side CPU usage is significantly higher during server-side XML transformation, demonstrating that XML processing is considerably more CPU intensive than simply reading and sending XML and XSL files for client-side transformation.
In order to provide a context for further comparison of client- and server-side XML transformation, the following paragraphs single out a number of interesting examples for discussion. Particular emphasis is placed on how these test scenarios relate to specific use cases in real life applications.

50 Simultaneous Requests

	XSL File Size					
	Small		Medium		Large	
Output Size	Client	Server	Client	Server	Client	Server
Small	1307	1829	565	1457	555	1670
Medium			567	1670	569	1577
Large			404	1577	452	2192

500 Simultaneous Requests

	XSL File Size					
	Small		Medium		Large	
Output Size	Client	Server	Client	Server	Client	Server
Small	3804	7779	3226	7233	3346	9187
Medium			3779	9187	4728	9640
Large			2969	9640	2936	14964

Note: *Small XML transformations with medium or large results were not possible due to the excessive recursive template matching required.*

Fig. 1. Average response times in milliseconds for *small* XML source file requests.

50 Simultaneous Requests

	XSL File Size					
	Small		Medium		Large	
Output Size	Client	Server	Client	Server	Client	Server
Small	650	891	667	1187	685	1975
Medium	1148	238	945	1580	1067	1968
Large	700	11430	603	10919	927	11182

500 Simultaneous Requests

	XSL File size					
	Small		Medium		Large	
Output Size	Client	Server	Client	Server	Client	Server
Small	4430	6215	3881	7176	6253	14880
Medium	3467	2269	5397	8836	6726	16812
Large	3178	85257	5354	78568	6164	94016

Note: *Medium XML transformations with large results were slow due to recursive template matching.*

Fig. 3. Average response times in milliseconds for *medium* XML source file requests.

50 Simultaneous Requests

	XSL File Size					
	Small		Medium		Large	
Output Size	Client	Server	Client	Server	Client	Server
Small	1495	2035	1437	1626	1741	2665
Medium	1649	2105	1482	3017	1792	3545
Large	1274	3016	1418	3219	1759	3961

500 Simultaneous Requests

	XSL File size					
	Small		Medium		Large	
Output Size	Client	Server	Client	Server	Client	Server
Small	10697	11269	11450	12485	12298	23039
Medium	10504	18377	10916	18463	12390	31416
Large	8980	27557	9992	28860	12639	38807

Note: *Large XML transformations with large results were slow due to recursive template matching.*

Fig. 4. Average response times in milliseconds for *large* XML source file requests.

1) World Wide Web Scenario

A common use-case on the World Wide Web (WWW) sees a browser-based client requesting information requiring transformation into XHTML for client side viewing. This scenario is well approximated by a medium sized XML file, a medium sized XSL file, producing a medium (page) sized XHTML output (see highlighted cells in Fig. 3). The server-based transformation requires similar output bandwidth to that of a standard HTML web page but has much greater CPU and memory requirements. Processing XML on the client requires the downloading of both XML source file and XSL stylesheet, resulting in a data transfer increase in the order of double that required by the server-side transformation. Nevertheless, this is largely irrelevant in performance terms for clients with high bandwidth connections. Furthermore, since the total data size often remains quite small (under 20 kb), even devices with limited networking resources may be capable of efficiently performing the XML transformation locally. Therefore, this common example provides a substantial benefit in terms of reduced server load and increased client-side utilisation without any other significant performance drawback, provided the following condition is met. Namely, clients must be powerful enough to produce the transformation in an acceptable timeframe from the perspective of the user.

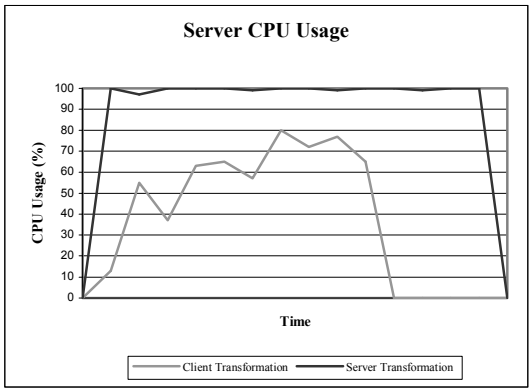

Fig. 2. Comparison of CPU usage for client- and server-side transformation with large input and output.

2) Business to Business Scenario

A typical business-to-business (B2B) scenario involves the conversion of a large data source from one format into another. This can be represented by a large XML file, a medium XSL stylesheet, producing large transformation output. Fig. 4 shows that for this use-case, performing the transformation on the client is up to three times faster than the same operation done on the server. This is because server-side parsing of the large XML source file requires considerable processing effort, and the large transformed result requires a significant amount of output bandwidth to the client. Therefore, provided the client is powerful enough to perform the transformation, the networking overhead of receiving both the XML and XSL file has even less relative impact than the WWW client scenario described above. Nevertheless, given the larger XML source file, some limited clients may not be capable of performing the transformation due to the increased processing and memory requirements, but modern desktop clients could easily perform this duty and relieve the server of a significant processing load.

3) Query Scenario

The final use-case involves a query on a large data set satisfied by a small result set. This translates into a large XML source, a medium sized style sheet, and a small XML output stream (see highlighted cells in Fig. 4). Performing the transform operation on the server requires significant CPU resources, but only a small amount of output bandwidth given the small result size. Server-side CPU load would be especially high where there were either, a large number of independent queries, or server-side caching was ineffective or unavailable. In the case of client-side transformation, a large amount of data must be transferred to the client, thereby limiting this option to devices with the necessary communications bandwidth. However, where sufficient bandwidth is available, client-side transformation relieves the server of a significant CPU load. Client-side transformation is also potentially advantageous when many small queries are made on the same data set, since the XML source data could

be cached on the client, thus negating the impact of increased bandwidth usage. Consequently, whether or not this query is best performed on the server or the client is largely device and application dependent, providing support for continuing research into dynamic client-server configuration, as discussed in the section VI of this paper.

4) Limitations

Given the test configuration described in section B, the physical data transfer limit for the network was never reached and thus bandwidth related issues are not reflected in the results of section D. Nevertheless, the effect of limited bandwidth has been addressed via the typical application scenarios of sections 1) to 3).

Although not strictly speaking a limitation, the financial cost impact of bandwidth usage has not been individually considered in the application scenarios above, because bandwidth cost is highly dependant upon the communication medium and service provider. For example, commercial 3G wireless networks have a much higher cost per kilobyte than broadband cable networks. As such, application users must ultimately decide the importance of speed versus cost in cases where service providers offer flexible cost models based on quality of service (QoS).

VI. Summary and Future Work

This paper has discussed variations in client/server configuration, for current and Next Generation Internet applications, from the perspective of application end-users, developers and service providers. The analysis found potential benefits for end-users and service providers but identified difficulties for developers who would need to target an increasingly diverse range of devices in order to provide more extensive client-side application services.

The tests discussed in section V, particularly the specific application use-case scenarios of V.E.1) to V.E.3), demonstrated empirically that in many cases, client-side processing can produce faster results and reduce server load, allowing the server to spend more time responding to other requests. The net result is that service providers can offer current QoS levels at reduced cost, or higher QoS without cost increase. Provided such initiatives are accompanied by flexible billing models, such an outcome can help offset the increased development cost and effort required to delegate application functionality to client devices.

Another notable finding was that whether a given application operation is better performed on the client or server is contextually dependent upon device capabilities and current levels of resource utilisation. This provides a strong rationale for further research into the area of dynamic client-server configuration and the development of frameworks providing greater support for device heterogeneity and more seamless development processes than current efforts such as J2ME [2] and .NET compact framework [4].

References

1. Baldi, M. and G.P. Picco. *Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications*. in *Proceedings of the 20th International Conference on Software Engineering*. 1998. Kyoto, Japan.
2. Sun Microsystems, *URL: http://java.sun.com/j2me/*. 2003.
3. Symbian Ltd, *URL: http://www.symbian.com/*. 2003.
4. Microsoft Corporation, *URL: http://msdn.microsoft.com/vstudio/device/compact.aspx*. 2003.
5. Moyer, S. and A. Umar, *The Impact of Network Convergence on Telecommunications Software.* IEEE Communications, 2001. **January**: p. 78-84.
6. Sun Microsystems, *URL: http://java.sun.com/j2se/*. 2003.
7. Sun Microsystems, *URL: http://java.sun.com/j2ee/*. 2003.
8. Singh, I., et al., *Designing Enterprise Applications with the J2EE Platform*. 2nd ed. 2002, Boston: Addison-Wesley.
9. Kinno, A.Y., Y.; Morioka, M.; Etoh, M. *Environment Adaptive XML Transformation and its Application to Content Delivery*. in *Proceedings of the 2003 Symposium on Applications and the Internet*. 2003.
10. Gravana Pty Ltd, *URL: http://www.bullant.com.au/Products/JRAP/*. 2003.
11. Nielson, J., *Designing Web Usability*. 2000, Indianapolis, Indiana: New Riders.
12. Jing, J., A. Helal, and A. Elmagarmid, *Client-Server Computing in Mobile Environments.* ACM Computing Surveys, 1999. **31**(2): p. 118-157.
13. Geary, D.M., *Graphic Java 2 Mastering the JFC Volume II Swing*. 1999, CA: Sun Microsystems.
14. Deitel, H., P.J. Deitel, and S.E. Santry, *Advanced Java 2 Platform How To Program*. 2001, NJ: Prentice Hall.
15. W3C, *URL: http://www.w3.org/TR/SVG/*. 2003.
16. W3C, *URL: http://www.w3.org/Style/XSL/*. 2003.
17. Apache Software Foundation, *URL: http://jakarta.apache.org/tomcat/*. 2003.
18. Microsoft, *Performance - Microsoft (r) Management Console 1.2*. 1999.