Performance Issues in Integrating a Capital Market Surveillance System Using Web Services

Feras T. Dabous, Fethi A. Rabhi and Hairong Yu
School of Information Systems Technology and Management
University of New South Wales, Sydney NSW 2052 Australia
{f.dabous,f.rabhi,hairong.yu}@unsw.edu.au

Abstract

Internet-based technologies have opened new opportunities for conducting business within and across enterprises that were never possible a few years ago. This paper presents our experience in using Web services for prototyping a service-oriented architecture for Capital Market Systems (CMSs). Our work exposes a world-class surveillance system's functionality into a number of Web services. Our work also includes benchmarking the performance of this legacy system and investigating the associated overheads of using SOAP as a wire format for Web services. Even though other research studies have tried to explain SOAP's performance inefficiency, there is lack of studies that evaluate SOAP in the context of a realistic business application. This initial investigation shows that system's integration opportunities introduced by Web services can outweigh the performance overheads. This occurs in some aspects of real-time CMSs that are not performance-demanding such as the dissemination of market alerts to the analysts.

1. Introduction

Distributed computing over the Internet has been recognised since the introduction of distributed object technologies such as CORBA and Java RMI. Object technologies are tightly coupled and believed to be suitable as frameworks for the interaction between related components or applications (i.e. within an enterprise). For few years, tremendous effort has been made to drive businesses into utilising such technologies in so-called e-businesses integration environments. However, e-businesses integration requires an interoperable loosely coupled infrastructure. In addition, integration over the Internet has gained momentum since the introduction of the Web services concept. The Web services architecture has been endorsed by both middleware technology vendors and distributed applications development communities especially in the e-businesses domain for its simplicity in integrating arbitrary applications within and across enterprises.

Many studies have emphasised on the performance degradation of Web services which could significantly affect performance in real-time applications. However, none of these studies has pointed out that this degradation in performance should be measured in the context of realistic applications. For instance, the throughput of a Web service can be acceptable if every possible realistic usage of the web service would still be within the required throughput.

Based on an initial a service-based software architecture for Capital Market Systems (CMSs) that has been presented in [4], this paper presents ongoing research work in investigates the applicability and usability of Web services in integrating a commercial CMS.

2. Background in CMs Surveillance

The primary objective of any surveillance department in a Capital Market (CM) is to detect illegal trading behaviour such as *insider trading* and *market manipulation* so that a fair and efficient market is maintained [7]. In order to achieve its objectives, the surveillance department correlates real-time trading data against historic trading data, company announcements and market news. Proactive surveillance systems such as SMARTS [7] which is the case study used in this paper are employed in many capital markets. The main objective of those systems is to generate market alerts as they occur in real-time and then to deliver them to the market analysts for further investigation. SMARTS employs a special purpose built-in alerting and analysing language called ALICE and therefore can adapt itself to new market rules and regulations by simply updating the corresponding ALICE program.

SMARTS is a Unix/Linux based system that receives real-time transactions (orders and trades) from an exchange trading engine, processes them as they arrive, manages a lo-

Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03)
0-7695-1999-7/03 $17.00 © 2003 IEEE

cal copy of *orderbooks*, checks against illegal trading behaviour rules, and then issues and disseminates alerts (if any) to market analysts. Analysts can then utilise a number of associated tools to investigate further actions.

Figure 1 presents a conceptual view of a proposed surveillance service and its interactions with other CMS services in the context of a service-oriented architecture for capital markets. This paper focuses on the order dissemination part where interactions with the other services in the CMS architecture are being investigated.

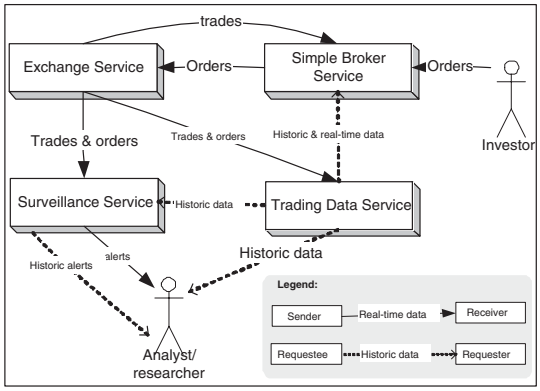

Figure 1. Conceptual overview of market data flow between CMSs

3. Related Work in SOAP Performance

Many studies have been conducted on evaluating Web services technologies, in particular the wire protocol SOAP, against performance requirements [3, 1, 2, 5]. In these studies, the performance of different implementations of SOAP has been compared in addition to performance comparison with other technologies such as CORBA and JavaRMI. These studies have focused on two major performance metrics: latency (i.e. roundtrip time) and throughput (transactions rate) correlated with SOAP message size. The major Web services performance factors that have been identified by such studies are:

- The variation of different SOAP implementations in marshalling and unmarshalling the SOAP message [2] specially for scientific numeric numbers such as float point values [1]. Therefore, object technologies show a better performance than SOAP [2][3].
- The complexity (the size and the presence of nesting within tags) of the XML formatted SOAP message [3].
- HTTP is a request-response protocol, and therefore naturally suits synchronous RPC-based Web Services but is ill-suited for document-based exchanges [3]. The reliable messaging protocols such as MQSeries better suit asynchronous document-based services.
- The protocol of interaction: synchronous that uses RPC style shows a better performance than asynchronous that uses document-based style [5][3].
- The text-based nature of SOAP contents as opposed to a compact equivalent of binary representation of the same data. However, [3] shows that the well-established text-based FIX protocol is outperforming an equivalent binary protocol.
- Current SOAP implementations are not suitable for performance demanding real-time applications [1].

These studies have emphasised the need to modify or extend emerging Web services standards in order to improve their performance.

4. Surveillance Web Services Design

We have designed two Web services that wrap the SMARTS surveillance functionality to expose services similar to two of its client applications (see figure 2): ALDIT (Alice Editor) and ALMAS (Alerts Management). ALDIT corresponds to posting and executing an analyst's ALICE program over a selected period of time and collecting the results that could be in the form of alerts and/or market metrics computed values. ALMAS corresponds to retrieving/editing/management of real-time alerts based on real-time market data or historic alerts based on historic market data in addition to performing calibration runs. Figure 2 presents the use cases for both services. The next two sections discuss two of the most important and surveillance demanding use cases as Web services: retrieving real-time alerts and calibration alerting.

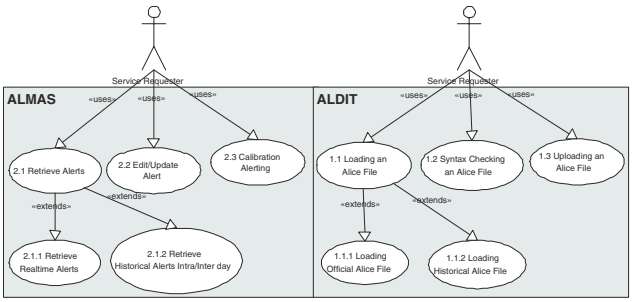

Figure 2. Use cases for ALMAS vs. ALDIT corresponding Web services

- **Real-time alerting service** invokes legacy code related to retrieving alerts from the SMARTS alerts database for the current day up to the moment the service is invoked. This is a very fast retrieval process and depends only on the number of available alerts and not on the market real-time trading data,
- **Calibration run service** invokes legacy code (called the `alertserver` program) for the period (interday or intraday) specified in the request. Our preliminary study shows that alertserver complexity depends heavily on the size of market data (i.e. number of transaction) for the requested period and on the complexity of the Alice program submitted by the service.

5. Implementation

In our implementation, we endorse the use of the gSOAP [5] toolkit for a number of reasons: gSOAP is implemented in C++, provides a C/C++ binding, and therefore is best suited to integrate the SMARTS system as it is fully written in C. gSOAP does not suffer from the Java-based implementations' inherent performance problems. gSOAP provides a transparent API as its compiler generates stubs interfaces and thus allows more interoperability [5].

We use RPC-style synchronous communication which is currently supported in every SOAP implementation and best suited for the real-time alerts retrieval. However, we are considering the document-based asynchronous style which is better suited for calibration runs.

6. Experiments and Performance Evaluation

Existing SMARTS client applications run on the same server and communicate with the server and other client applications using the concept of inter-process communication protocols and shared memory[1]. Therefore, the evaluation approach adopted in this study focuses on determining the overheads of incorporating Web services in terms of latency and throughput while ignoring the network traffic.

6.1. SOAP message size

This experiment aims at determining the size of a SOAP message compared to the size of the binary or text data generated by SMARTS. It shows that the size of a SOAP message (which is about 130 bytes) is about 10.7 times bigger in average than the size of the original alerts generated by SMARTS (which is about 1392 bytes). This result has been generated using the calibration run. This is conformant with related studies on the SOAP message size.

[1]SMARTS has just released Windows-based "fat-client" application using a private communication protocol built over TCP/IP.

6.2. Latency

This experiment aims at determining the roundtrip time for a request. This time involves marshalling the SOAP message and binding it to the HTTP protocol at the requester side, and vice-versa on the service side. We also determine the latency of the legacy code invocation and then determine the Web services overheads percentage.

6.2.1. Calibration run service

This experiment has been conducted on Australian Stock Exchange (ASX) historic trading data for two months in 2000. Table 1 shows that the overall Web service average latency is about 8.5 seconds where 99% of this time is consumed by the legacy code invocation that processes the trading data for a whole trading day. Another experiment on few days of trading data in 2002, which has over a million transactions per day, has shown that the legacy code consumes even more than 99% of the total execution time. This is a natural result as the legacy code processing for larger volume of trading data takes more time and the SOAP request and response message sizes do not affect performance.

Table 1. Average latency time of SMARTS server vs. Web services

	Overall Avg Latency	Legacy Avg Latency	SOAP Latency
Calibration Service	8.51 sec	8.38 sec 98.5%	0.13 sec 1.5%
Real-time Service	1.01 ms/alert	0.26 ms/alert 26%	0.75 ms/alert 74%

6.2.2. Real-time alerting service

This experiment is conducted on a selected ASX historic trading data period that has a high alerting rate. It should be noted here that this experiment shows that the number of daily official real-time alerts of any market rarely exceed 200 alerts [7]. The worst case scenario is when an analyst makes a request just before a market closing time at the end of a trading day for all available alerts. This experiment shows that the overall average latency of requesting a single alert is about 1 ms. In this case, the legacy code invocation is about 26% of the overall Web service latency (see table 1). This is because the legacy code process involves a simple retrieval operation from a database. Figure 3 shows the latency when requesting large number of alerts in the worst case of a realistic scenario.

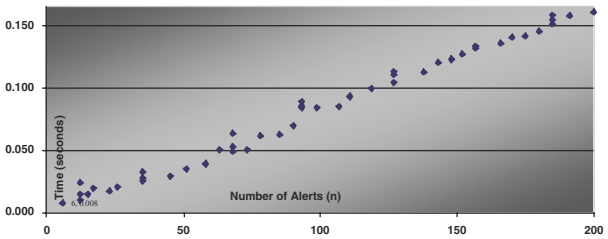

Figure 3. Latency of requesting `n` alerts in real-time

6.3. Throughput

Throughput analysis for surveillance services is an ongoing work. However, we have identified that every invocation of a calibration run service is measured in seconds due to the heavy process involved when the legacy code processes market data for the requested period. Preliminary experiments show that the real-time alert service throughput is about 1043 messages/second.

7. Conclusion and Future Work

In this paper we have presented some Web services based on a realistic surveillance system that is currently operational in more that 10 securities exchanges world-wide. These services are in the context of prototyping a service-oriented architecture for CMSs. We have presented two Web services which represent typical scenarios in surveillance. We have conducted performance experiments for these two services, which use the gSOAP RPC-style, on realistic market data from the ASX.

There are four main results from these experiments: firstly,in the calibration run service, the legacy code invocation latency depends mostly on the size of trading data that needs to be processed. This latency measure is about 99% of the overall Web service. Therefore, the SOAP marshalling, binding to HTTP at the requester side, and vice versa on the service side is insignificant when the legacy code invokes a computationally-intensive process. Secondly, in the real-time alerting service, the legacy code latency is only 26% of the overall Web service. However, in a realistic scenario the number of daily alerts rarely exceeds 200 alerts which only take 156 ms. Therefore, the SOAP performance overheads should not be considered to be a bottleneck for services that are not demanding such as the real-time alerting service. Thirdly, the SOAP message size is about 10 times its equivalent binary representation. This size can be reduced in three different ways: using compact XML tags which may affect readability of the message, using a compression algorithm which adds extra cost in compressing and decompressing, and using a binary SOAP message which is still in research stage and has not evolved as a standard yet. Finally, exposing the functionality of this surveillance system as Web services can significantly facilitate the integration opportunities with other systems in the capital markets area and thus enabling more interoperability.

Ongoing and future work for this project involves considering document-style Web services which facilitate asynchronous loosely-coupled message-oriented integration and provide better performance results. This style of integration is suitable for integrating different CMSs. For example, a surveillance service can invoke an *Exchange* service to receive trading data, a *News* service to receive market announcements, and a *Trading Data* service to retrieve historic market data. On the other hand, other services such as *Settlement* and *Registry* can invoke the surveillance service to check the compliance and legality of a conducted trade. We will also investigate the RPC-Literal style performance and its implications on interoperability when compared with RPC-style performance.

Acknowledgments

The authors would like to thank Glen Eric Tan and Robert Kai Chu for their effort in prototyping and conducting the performance experiments. This work is supported by the CMCRC [6] and its industrial partners[2].

References

[1] K. Chiu, M. Govindaraju, and R. Bramley. Investigating the limits of SOAP performance for scientific computing. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, pages 246–254, 2002.

[2] D. Davis and M. Parashar. Latency performance of SOAP implementations. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and Grid*, pages 407–412, 2002.

[3] C. Kohlhoff and R. Steele. Evaluating SOAP for high performance business applications: Real-time trading systems. In *Proceedings of WWW2003*, Budapest, Hungary, 2003.

[4] F. Rabhi and B. Benatalla. An integrated service architecture for managing capital market systems. *IEEE Networks*, 1, 2002.

[5] R. A. van Engelen and K. A. Gallivan. The gSOAP toolkit for web services and peer-to-peer computing networks. In *Proceedings of IEEE CC Grid Conference*, 2002.

[6] Capital market cooperative research centre (CMCRC). `http://www.cmcrc.com`.

[7] Securities markets automated research trading surveillance SMARTS. `http://www.smarts.com.au`.

[2]SMARTS (www.smarts.com.au), SIRCA (www.sirca.com.au), and ComputerShare (www.computershare.com.au).