Integrating Web Applications and Web Services

Nicholas L. Carroll and Rafael A. Calvo

Web Engineering Group
School of Electrical and Information Engineering
University of Sydney, NSW 2006
Australia
{ncarroll,rafa}@ee.usyd.edu.au

Abstract. Database systems are an essential component in multi-tiered web applications. These applications increasingly interact with others using web services. In this paper, we describe and compare two architectures for integrating web services into web applications, and perform performance benchmarks using the Google web service. For one of the architectures we contribute a SOAP interface to the PostgreSQL Relational Database Management System, implemented as a user-defined function that allows developers to make service calls from within the database. We show that SQL can be used to easily query data provided by web services, and therefore as a way of accessing and using web services in a database-driven web application.

1 Introduction

Database-driven web applications store, access, process and serve information using a database and application server. Recently, these applications have been able to access data stored in other systems using web services. An understanding of how different service oriented architectures affect system performance and development effort is important to support the right design decisions in web development. Integration of web services into a web application usually occurs within the application's business logic. However, in recent releases of databases such as Oracle and DB2, it is now possible to request web services directly from the database system. Due to these new possibilities we should reassess the best architectures for integrating web services. This paper contributes to this reassessment by comparing two architectures.

We will use a reference scenario that defines a need for integrating web services into a web application: integrating a student information portal with a search engine web service, and using contextual information to improve the accuracy of search results [1]. In practical terms this can be achieved by integrating the search engine with the business logic of an application. For example, a student searching for third year electives from her University's course catalogue, might simply enter a general search for "third year courses" into the search engine. The results of the search would list third year courses provided by all faculties at the University.

D. Lowe and M. Gaedke (Eds.): ICWE 2005, LNCS 3579, pp. 328–333, 2005.

Depending on the number of results returned, narrowing the results down to third year courses may prove to be time consuming. If the search engine is integrated into the application as a web service as described in Section 2, it could implicitly use such facts as the student's prior course history to improve the accuracy of the search results. The integration could provide a more complete query without the student needing to describe the context, thereby limiting the search results to courses that the student qualifies for enrolment. This type of integration is not commonly used despite its potential benefit [2]. Two key issues need to be addressed to improve the uptake: 1) the performance of such systems cannot be degraded, and 2) developers must find it easy to integrate these data sources into their applications. We address the first in Section 3 and the latter in Section 4, and Section 5 concludes.

2 Integrating Web Service Requesters

The two architectures described here are both distributed, multi-tier architectures consisting of: a presentation component, business logic, database, and a web services requester that binds to a web service provider. The differing factor between the two architectures is the positioning of the web services requester component for integration with the web application. This difference is conveyed in Fig. 1, where Architecture 1, the most commonly used, integrates web services into the business logic component [3], and Architecture 2 integrates web services into the database component. The last architecture is not very popular but recent releases of Oracle and DB2 are now capable of using SOAP [4], as the XML messaging specification for communications between a web service requester and web service provider.

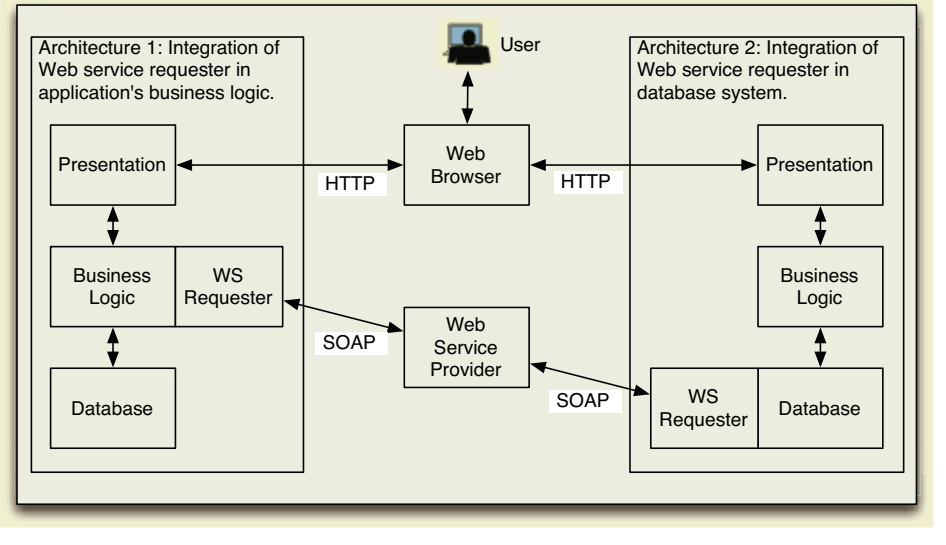


Fig. 1. Two architectures for web services integration

Both architectures were implemented using the same technologies wherever possible. We used Java to implement two simple applications using each architecture. Both web applications provide the functionality described in our reference scenario. For Architecture 1, we integrated the Google web service requester directly into the business logic component, which was as trivial as calling specific methods supplied by the Google Web API. In Architecture 2, we integrated a "SOAP client into the database through user-defined functions" [5]. Most database systems support user-defined functions, which means this approach for integrating a SOAP client into a database system can be generally applied. The SOAP client functioned as the web service requester component within the database system. We used PostgreSQL in both cases.

3 Performance Analysis

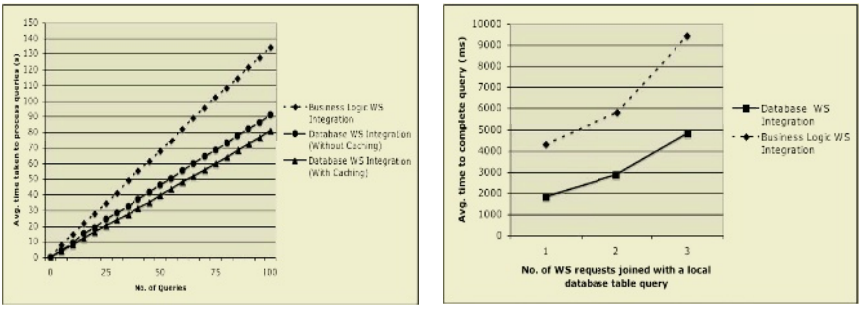
The performance analysis that we conducted was limited to the time taken to access and combine information from multiple data sources. In our scenario, we use the provided student ID to query the local database system for information about the student, which can then be used in conjunction with an external data source, to produce more accurate search results. Analysing the accuracy of the search results is beyond the scope of this paper. For this scenario, we assume that associating context with a search engine query improves the accuracy of search results based on efforts in [1].

3.1 Experiment One: Benchmarking with Google

Our first objective was to compare the performance of accessing a web service from within a database system, to accessing the same web service directly from the business logic of a web application. An advantage of requesting web services directly from the business logic is the convenience of accessing web service APIs provided by the programming language used to implement the business logic. In the first test (business logic integration) we used Google's Web API implemented in Java. A drawback of this architecture is that if the information returned from a web service call is connected with the data stored in the database in some way, then there would be an inefficiency caused by the overhead of communications between the database and web service via the business logic. It was therefore necessary to determine if there was in fact a performance gain for requesting web services directly from within the database.

For the second test, we chose to query the PostgreSQL database for a list of search terms, which would then be used to forward to the search engine web service. The results from the search engine web service would then be displayed. The goal was to determine which architecture could perform this task in the least amount of time. This task was performed ten times each for both web applications, and the times for all ten runs to process 100 queries were logged.

Fig. 2(a) shows the average times taken to process 100 queries by both architectures. The graph indicates that requesting a web service from within the



(a) (b)

Fig. 2. a) Average time taken to execute web service requests. b) Average time taken to join data from different data sources

database is more efficient than consuming web services directly from the business logic component of a web application. The business logic web service integration architecture achieved on average 0.75 queries/sec, while the database web service architecture achieved on average 1.23 queries/sec when PostgreSQL was allowed to use its internal default caching mechanism, and 1.10 queries/sec when it was disabled.

3.2 Experiment Two: Adding Context to Google's Queries

Our second objective for our performance analysis was to compare the efficiency of both architectures in joining information from multiple data sources. To evaluate this, we created a table in the database that matches a student ID to the URL of the faculty that they are enrolled in. The goal was to query the database table for the URL of the faculty that the student is enrolled in. Then use the retrieved URL to filter the results from the search engine to limit the results for the student's search query to a faculty web site. This task was designed to simulate our reference scenario of associating context to a search query.

For the database web service, the task can be summarised as one SQL SELECT statement as shown below. In fact, joining additional web services together to query from was as trivial as listing the user-defined functions to the respective web service requesters after the FROM keyword in the SQL SELECT statement. Implementing this task was not as straight-forward in the business logic web services architecture, especially when joining information from more than one data source. Our implementation required loops to iterate through the search engine results to find web pages that match the faculty URL. It was observed that joining information from multiple data sources from within the database required considerably less code to implement than doing the equivalent in the business logic. The benefits of which are reduced complexity and improved maintainability of the source code.

```
SELECT summary, url, snippet, title, directorytitle
    FROM google_search('third year courses') gs
    WHERE EXISTS (SELECT 1 FROM faculty_urls fac
    WHERE fac.sid = 200517790 AND gs.url LIKE fac.url || '%' )
```

We measured the time taken to complete the task of accessing and joining data from a local database table with a web service for a single search query. We then repeated the task joining a second, and third web service in addition to the existing data sources. These tasks were completed ten times each, and the average times for each task are shown in Fig. 2(b). The results show that the database web services architecture performed the tasks up to 50 percent quicker than when the web services requester was integrated directly into the business logic. Also the performance of the database web services architecture did not degrade as much as the business logic web services architecture when additional web services were joined to the query.

4 Scenario-Based Evaluation

It is difficult to define "optimum architectures" since comparisons cannot be abstracted from the circumstances the systems are used in [6]. In fact, according to the Architecture Tradeoff Analysis Method (ATAM) described by Bass et al [6], the architectures must be evaluated based on a set of quality goals. These quality goals can only be based on specific scenarios and on the perceptions that different stakeholders have of them. This method is out of the scope of this paper but we will address three of the ATAM quality goals: usability, performance and scalability.

Performance and scalability are important since the system would perform a large number of interactions with the web services provider. A drawback of the common architecture for integrating web services directly into the business logic is that, if data returned from a web service is connected with the data stored in the database in some way, then there will be an inefficiency caused by the overhead of communications between the database and web service via the business logic component. The database web service architecture does away with this overhead, leaving the database to obtain data from web services directly.

An important limitation of the database web service architecture is that it can only be used to query data-provisioning web services. Data-provisioning web services are services that use SOAP RPC, as oppose to the SOAP Document style. The SOAP Document style is used for "complex web services that deal with the composition of other web services to formulate workflows for business processes" [7]. Business composition web services rely on meta-data for their composition with other web services. A more suitable query language for business composition web services is XQuery, which is "XML based and can process meta-data" [8]. XQuery is suited more towards architectures that integrate web services into the business logic component of a web application.

Most database systems provide support for user-defined functions where developers can add their own implementations to the function catalog of a database

system. The advantage of this approach is that the integration of web services is not dependent on extending SQL to support web services. Secondly, access to web services is totally transparent to the application developer, as the invocation of a web service is only a call to an SQL compliant database function.

5 Conclusion

We studied two architectures for integrating web services into database driven web applications. The first architecture presented, integrated a web service requester directly into the business logic component. The second architecture, with the web service requester in the database, consolidates access to multiple data sources through a single data access point. These architectures allow web developers to implement systems that can produce queries to external data sources with contextual information provided by the business logic and internal data sources, as was demonstrated through the use of a reference scenario. We have shown that the database integration is more efficient (Section 3) due to a better coupling of the query and contextual information. This architecture therefore provides an alternative for web developers looking at integrating web services into their database-driven web applications. The main disadvantage (Section 4) is the lack of support, but this is changing due to new products.

References

1. Leake, D.B., Scherle, R.: Towards context-based search engine selection. In: IUI '01: Proceedings of the 6th international conference on Intelligent user interfaces, Santa Fe, New Mexico, United States, ACM Press (2001) 109–112
2. Barros, F.A., Goncaolves, P.F., Santos, T.L.: Providing context to web searches: The use of ontologies to enhance search engine's accuracy. Journal of the Brazilian Computer Society **5** (1998)
3. Geetanjali, A., Kishore, S.: XML Web Services – Professional Projects. Premier Press (2002)
4. Malaika, S., Nelin, C.J., Qu, R., Reinwald, B., Wolfson, D.C.: Db2 and web services. IBM Systems Journal **41** (2002) 666–685
5. Carroll, N., Calvo, R.: Querying data from distributed heterogeneous database systems through web services. In: The Tenth Australian World Wide Web Conference (AUSWEB'04). (2004)
6. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2/E. Addison Wesley Professional (2003)
7. Curbera, F., Khalef, R., Mukhi, N., Tai, S., Weerawarana, S.: The next step in web services. Communications of the ACM **46** (2003) 29 – 34
8. Hoschek, W.: A unified peer-to-peer database framework for scalable service and resource discovery. In: Lecture Notes in Computer Science - Proceedings of the Third International Workshop on Grid Computing. (2002) 126–144