Abstract

Software development using the Service Oriented Architecture (SOA) model can be an effective commercial solution in regards to simplicity and flexibility. Using the XML format, specifically Web Services Description Language (WSDL), a single structure is used for all communication between the client and server software. This single communication method, then allows for platform and language independence between the client and the server applications.

In this paper, we present an open source Service Oriented Architecture implementation. The reader will be presented with a prototype web service to use as a basis to provide performance testing results for this Service Oriented Architecture solution developed from open source tools.

1. Introduction

Recent development trends toward software development using the Service Oriented Architecture has proven productive, with the potential for high profitability and efficiency when used in industry. The simplest forms are completely web-based, and allow limited software reuse, but in a more robust form, this architecture could allow a single commercial application to be used by multiple, language and platform independent, client interfaces. This Service Oriented Architecture proves to be a highly effective solution with possible implementation through many commercial software applications.

Providing a single server-side solution for multiple client-side applications allows for a high degree of software reuse. This ability is achieved through a single XML structured format for all data transferred between client and server applications called WSDL. Since the WSDL passes a single data design between all software components, the only commonality required is the names and formats of the parameters to be utilized. To add even more flexibility, a well thought out and structured WSDL design could incorporate many enhancements before any significant rework of the client or server applications would be required.

In our work, we implemented a telephone directory as a web enabled service using this Service Oriented Architecture model, and the WSDL format. As conditions to our development, our prototype was developed using totally Open Source compliant software, and all activities associated with the final product were required to be invoked through a web browser.

2. Related work

Software interoperability is a major concern for developing heterogeneous applications in enterprise computing. Interoperability of heterogeneous applications is defined as the ability for multiple software applications written in different programming languages running on different platforms with different operating systems to communicate and interact with one another over different computer networks.

Web services have been widely considered to be a valuable solution to enterprise application integration for software interoperability. Web services use open standards including XML, Simple Object Access Protocol (SOAP), and WSDL. Clients and servers follow the Web service protocol for message exchanges. Figure 1 illustrates a sample scenario.

![Figure 1. Web services](image-url)
Service Oriented Architecture. The development of the prototype on SOA follows alongside Zilora’s [16] work as it relates to the implementation of a Web Service from the perspective of project lifecycle. Furthering Zilora’s research, our project team takes a real-world perspective on a single project implementation. In our case this was an open source SOA prototype that we named Access Point - Telephone Directory Prototype Web Service.

3. Telephone directory prototype overview

As it relates to our project, the team developed a prototype of a Telephone Directory Web Service to provide a tool to use for performance testing of the open source SOA model. In the development of this prototype, or team was given a series of initial requirements. First the solution should be developed entirely using Open Source compliant tools. Secondly, the service was required to transfer data via SOAP (Simple Object Access Protocol). Also, the final product should use WSDL for defining all parameters.

After taking into consideration all of the requirements presented to the project team, the Access Point Telephone Directory Prototype began to take shape. Figure 2 presents a component diagram of the final Access Point product developed by our project team.

![Figure 2: Access point telephone directory prototype components diagram](image)

4. Service Oriented Architecture Components

For development purposes our team considered all components required for development of an open source SOA prototype. These choices centered around the available hardware, the tools around the storage and implementation of the client and server components, the open source tools for development, and tools to be utilized during our testing phase of the system.

From the perspective of developing a SOA modeled solution, four open source components were heavily utilized: Apache HTTP Server, Apache Tomcat, Apache Axis, and Eclipse.

5. Development of the Telephone Directory Prototype on the SOA

Our final product, Access Point Telephone Directory Prototype, was developed through the Service Oriented Architecture (SOA) design model. As an outline of our solution, three methods were implemented as the business rules for our tool, Login, CustomerSearch, and Logout. Figure 3 presents a data flow diagram representing all data actions within the entire Access Point Telephone Directory Prototype.

![Figure 3: Access point telephone directory data flow diagram](image)

The Eclipse tool used the business rules we established through our functionality and provided a major amount of assistance in developing the core classes used by our prototype. These three classes, the PortBindingImpl, the PortBindingStub, and the PortBindingSkeleton worked together to manage all of the functionality for the Access Point Telephone Directory Prototype. The Stub provides a communicational interface between the client and the Skeleton, while the Skeleton takes the information from the Stub and formats it for the implementation to use through the Web Service itself, all using the WSDL format to manage the data transfers.

From the perspective of the classes within the Access Point Telephone Directory Prototype, three methods are implemented for the entire solution. As noted before these three methods comprise the functionality of login, logout and the actual telephone query itself. As a note, two private methods are also included in the Implementation class. These two methods serve specific purposes in the final solution. First the InitializeJDBC method is used to create a connection for the Web Service to the actual database.
housing information relating to the user authentication tasks from login and the customer search tasks associated with the CustomerSearch methods. The other private method involved in this process, searchTotal, is called by CustomerSearch in order to perform validation on the SQL query and to detail out the number of results that will be presented to the user after the search is performed.

6. Testing results

With the design completed and the coding performed, the next aspect to review is the testing of the Access Point Telephone Directory Prototype.

During the coding phase of our project, a series of System test cases were derived using a Black-Box, Top-Down approach. These tests were generated based on these four design components: Login, CustomerSearch, Logout, and Tracking.

This set of tests did provide a series of unanticipated results, instead of getting an expected error message when Zero results were returned, the output presented a default ‘No Users’ return from the database call. After the team discussed this it was determined that this was caused by the test cases being created in a Black-Box approach. The code was written to return the default entry instead of an error message. Outside of this issue, all other system tests performed as expected.

The final type of testing performed involved load testing the system using an open source tool called Http_Load. This tool allowed the project team to set the number of connections to attempt to make every second, and the length of time to keep these connections established. The results of these tests allowed our team to complete a series of charts. Figure 4 and 5 display the results of this testing against the web service.

Looking at the overall results, in regard to the number of successful fetches and the number of kilobytes transferred during the web service testing, the service appears to be able to handle a maximum number of 250 KB per second. These results were consistent as a maximum setting to the service.

However for the web server itself, the time to connect did vary once we began to exceed the number of connections per second that the service could handle. Adding our extended testing for the web server, the connections performed in a similar fashion, with a large range based on the timeframe.

Further review of the results presented a concern in regards to the time necessary to establish a connection to the web server, but not during testing of the web service itself. During web server testing, the http_load tool returned results showing more connections than we were attempting to achieve. In a real application setting, this result would have been enough to cause the team to return to the beginning of the performance testing cycle using a different load testing tool.

7. Experience learned

With the purpose of this project and the experience of working in a industrially modeled project structure, all members of the project team left the project feeling like we have taken on a task that will become a valuable component in the business world. Knowing this experience will help us grow our skills, we did finish the project with a large set of experiences to guide us in the future. As for our project experiences, we broke them down into the following categories: General Project experiences, Hardware related experiences, and Development experiences.

From the view of General Project experience, overall our team felt that this was a great overall experience, however, we did see many areas that we could improve. First from the project itself, we as a team did not feel like our estimates in regards to Project phase planning were very accurate. For example, our Design phase of the project was
estimated to occur over a two week period, but in the end we actually were performing Design tasks for almost two months, this caused our Design and Coding phases to essentially overlap. Our team felt at several points along the way that we had a complete grasp on the concept only to find that we were just opening a new door. The General Project experience does include a few items that could assist future work in this type of setting to get a better start. The major example our team noted relates to the initial status of the project. As we look back, as a team we felt that all components that we expected could have been included with more definition around the project at the beginning of our work. Our team also realizes that having this type of information thrown out at the onset of the project could become an overwhelming experience.

Hardware problems were the largest single problem our team worked through during this project. Our team as a whole believes that the project would have been better served by performing all of the software installation tasks on a server that does not contain older versions of the software. Even taking this position, the Access Point Telephone Directory prototype project does bring a feel of real world situations because we were required to work within the system in place. It gave us a chance to see SOA model developed from beginning to end, and get a solid understanding of how these types of tools will perform.

The final aspect to consider from a perspective of Project experience is the Development aspect. One major hurdle to be cleared within our project was the development language itself. Due to the limited options available to the team, and the choices around the SOA components, Java was chosen as the development language. This choice came from the literature used in researching the topic. For our team only one member had a background using Java, so the actual coding aspects were hampered by this limitation.

Our team also experiences a large set of development delays. These delays ranged from hardware to team knowledge issues. In the end our final prototype did not meet all of the intended requirements outlined at the beginning of the project. Most of the features that were removed focused on security aspects we wanted to test in our solution. Because of these limitations and the fact that our team did not see the UDDI as a necessary step because the client side required the Stub code in hand, we eliminated UDDI from our web service implementation.

Any future work in this direction would encompass many considerations we were not able to include due to our project timeline. The secondary features we would incorporate into the prototype would center around client-server security, client billing, and enhanced error handling features. Beyond this our project team’s ideal implementation would have incorporated thorough performance testing along with benchmarking against other SOA designs.

Acknowledgements

As a team we are indebted to all reviewers, the valuable comments have contributed greatly to the quality of this paper. Our project team is grateful for the assistance given by Mr. Chris Morris. His knowledge allowed this project to achieve a successful conclusion.

References