A Framework for the Distribution of Data using Web Services and XML

Robert Nack
Computer Science Department
University of Wisconsin – Eau Claire
nackrm@uwec.edu

Abstract

While Web Services are becoming important pieces of today's Internet-driven businesses, they may also have useful roles in the realm of research. We have identified a need within a local group of student researchers for an easy way to access data that is generated as a result of test simulations. We have developed a framework that is easy to set up and allows for easy access to data.

Introduction

Information is the key to success. This axiom is what has pretty much defined the "Information Age". With the value of information being so high in our society, finding ways of sharing information at the lowest cost possible is very advantageous for obtaining more information. If we can lessen the burden of sharing the information we already have and maintaining its access to the others from the outside world, then we can shift those resources to the actual research that is being done in the first place.

The Reason

This project was developed for several reasons. The first reason was the formation of a group of students dedicated to sponsoring research in the field of computer science. The Special Interest Group for Research in the Department of Computer Science at the University of Wisconsin – Eau Claire, in its first few months, found itself involved with a handful of projects. One of these projects requires that large result sets generated by the simulations be stored for later use. How much data will need to be stored? Thousands of processes will be spawned; each of them adding hundreds of rows of data, with each row being ten to twenty fields when the simulation runs.

All of this will be stored in a database on a server, which is enclosed within a private network with limited access to the outside. The simulation application is run on a group of computers, which are also inside this private network. Once the data has been collected, users will need some kind of access to it so they will be able to evaluate the results. One issue we have to worry about is network congestion within this private network. The applications running in this environment communicate across nodes to each other while they are running. The more traffic congestion, the slower the applications run. With this in mind, we decided it would be in our best interests if we kept unnecessary traffic outside of this private network.

Also, we wanted to ensure the integrity of the data. If we prevent everyone on the outside from being able to modify any of the data, then we can ensure that the only changes are from within the small private network. While we could have allowed for a direct connection to the database, with different user accounts having different access, we wanted to try to incorporate these restrictions into a higher level of abstraction. In our case, we wanted to allow access to the database only from within the private network, and then allow a set of restricted commands on the database from outside. Web Services is a technology that allows for remote commands to be accessed by clients. With this in mind, we went with a design where outside users could access the restricted set of commands via a Web Service and access to the database was limited only to those computers within the private network.

Background

Web Services and XML are at the core of this project. Web Services is the technology that allows us to access the information we are requesting, while XML is a key part of

every step. Before going into the details of the project, we'll briefly go over some of the ideas behind both Web Services and XML.

XML

Extensible Markup Language (XML) has blown onto the scene of computers and information over the past few years as the "next big thing". More and more, different companies, entire industries, and even average computer science students are looking into its uses. What exactly is XML? XML is a way to store data, using tags that are similar to those in HTML, so that it may be retrieved later without any information loss [1]. While a file may somehow be lost or part of it becomes missing, XML will ensure the integrity of the data. If the end of the file is truncated, then it will become obvious to an XML parser that the file is missing at least part of an ending tag, and therefore is no longer well formed.

One thing that XML provides is a means to share data across multiple platforms and programming languages. Most popular languages supply a library for parsing information from XML documents. Even if one isn't provided, the rigid structure of XML allows the user to write their own if needed, but chances are someone out there already has.

Most modern operating systems encode files in either UTF-8 or UTF-16 encoding schemes, which is the default for XML documents. However, if your document is encoded in something else, most parsers should try and figure it out, until they can read the first tag, an <xml> tag, that has an attribute describing the encoding of that file. Once the parser tries an encoding that allows them to read that line, then it can confirm that it has selected the correct encoding type, and will go on its way to parsing the rest of the document. Thus the file can cross platforms, as well as programming languages, as an extremely flexible means of transferring data.

Web Services

Web Services can simply be described as a useful way for a client application to retrieve data from a service application. Typically each service is defined and published, and then a client can contact the service by reading the definition for details of how to connect and retrieve its requested data. XML is the markup language that underlies most of the specifications in Web Services because it is a generic language that can be used to describe any kind of content in a structured way [2].

Project Accomplishments

While Web Services provides for an easy way to standardize connections between clients and services, we were looking for an even easier way to define new services. Since most of our work will involve retrieving information from databases behind the private

network, then to make defining a new service should just require us to fill in a few pieces of information about the database and query to retrieve the data in it.

At this time, we have implemented a simple web service application that reads command definition files, parses out all of the configuration information, and then is able to accept connections from clients requesting the specific commands. Figure 1 gives a visual description of what happens in the standard case. In this standard case, there are five actions that occur in this series.

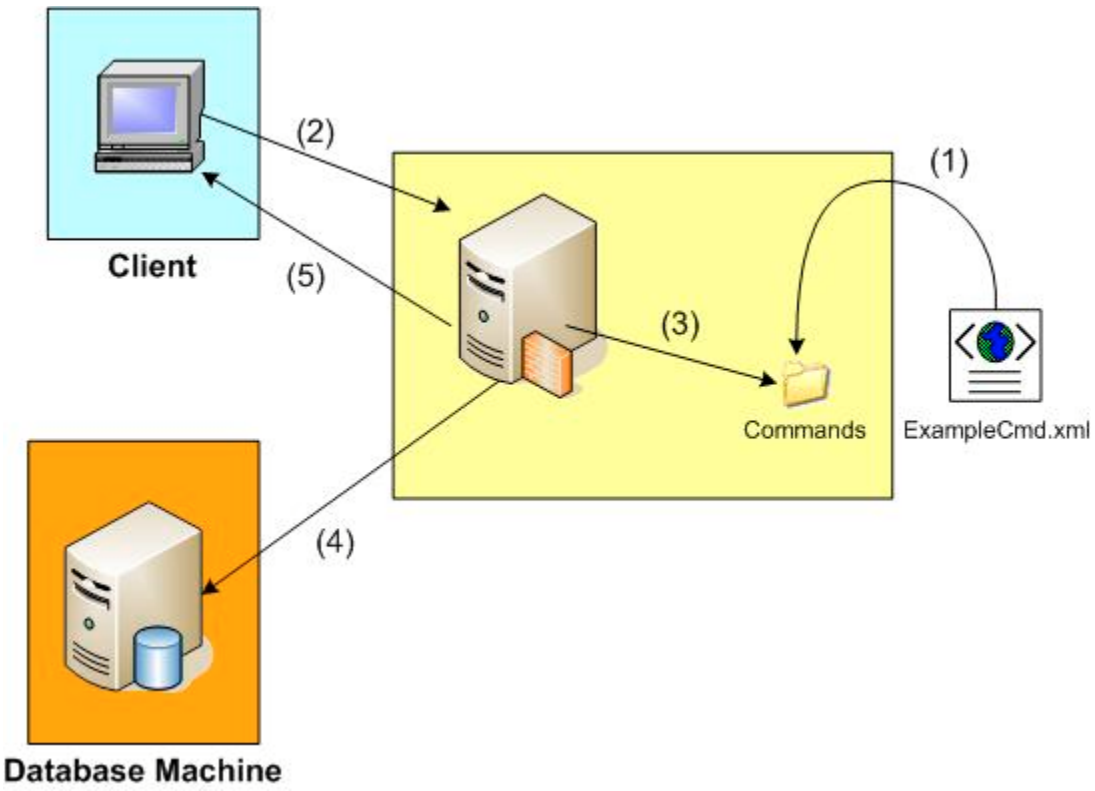

Figure 1: Standard Use Case of Application

First, a command, "ExampleCmd.xml", is defined by creating an XML file following the format of CommandsSchema.xsd, and is then placed into the "Commands" directory. CommandsSchema.xsd is a Schema specifically designed for this application. It allows the user to define security restrictions as well as the command itself. Later, I will describe more about what a command file actually looks like.

Second, a user can call a remote method called "runCommand" using XML-RPC. By passing the parameters as an array of arguments, where each argument is a key/value pair such as "command = ExampleCmd", then it becomes a simple task for the server to interpret the incoming request, no matter the order of arguments, just as long as the required ones are there.[1]

[1] As a side note, the current version throws an exception if an error occurs, the user doesn't fulfill the required security restrictions, or if the command is not defined.

Third, the server application receives the request for information. After parsing out all of the parameters, the server then checks to see if the command name exists in the "Commands" directory. If it exists, then the file is read in and parsed. Security requirements are checked, and if they are satisfied then the request continues. If they fail, an exception is thrown.

Fourth, the server executes the query. The command file describes for the server where, how, and what to connect and retrieve the information from its storage location. More likely than not, this will be some kind of query on a database. This is the case in the example from Figure 1. Results are returned to the application, where they are prepared for return to the client.

Fifth and last, the results are returned to the client application. They are packaged in the form of a two-dimensional array, a sufficient representation of a table of data. From here the client application can use this information in any way it could ever want to.

Figure 2 refers to a simple test xml file, "ExampleCmd.xml", which defines a command, called ExampleCmd, for the web service. This example command describes a simple command that gets a list of users from a database. It describes the database type, connection information, and query string as basic functionality under the <function> tag, but at the top under the <security> there are some optional elements. These elements can describe where a person can connect from, as well as if they are required to have a username/password combination to make a connection.

As soon as this file is placed into the commands directory of the application, the command becomes available to the Web Service users. The main difference between our system versus those using traditional Web Service technologies is that we are not actually adding a "new" Web Service. The information is still retrieved by calling runCommand(). If a user would like the information from this example query, they just pass a list of parameters, one of which being "command=getusers". That is what identifies which service is actually being requested to the server, so it can in turn execute and send back the correct results to the user.

```
<?xml version="1.0" ?>
<command="getusers">
 <version="1.0">
 <security>
  <iprange name="UWEC">
   <start="137.28.0.1" />
   <stop="137.28.255.255" />
  </iprange>
 </security>
 <function>
  <sql>
   <servertype="MySQL" />
   <connection>
       <address="localhost" />
       <port="3306" />
       <username='testname' />
       <password='testpassword' />
     <database='testdb' />
   </connection>
   <query='SELECT username FROM users'/>
  </sql>
 </function>
</command>
```

Figure 2: ExampleCmd Definition

Future Work

The goal of any future work is to make the process of setting up a Web Service for our particular needs easier. The current CommandSchema is not very strict on the form of data in its fields. More work should be put towards revising the schema, so it only accepts legal values in fields. That way, users can better identify when there is a problem with the syntax of their definition.

With more use of the framework we will be able to better identify more features and requirements, while dropping the ones that seem out of place. The implementation of our project will certainly undergo many changes as issues are found in its current state.

Summary

The main goal of creating a platform-independent, simple and easy to set up way of retrieving information from within a private network has been achieved in this project. We are able to connect to the implemented server, which in turn retrieves the data we are requesting. Performance, while not a top-level requirement at the time, has been considerably excellent. With sample test data, we have been able to retrieve over two thousand rows of results in less than a tenth of a second. While the implemented application was originally developed and tested in a Windows environment, it is currently

running on a Linux machine, with test clients connecting from both Windows and Linux. While this may not cover the total spectrum of platforms, it does lead us to believe that few modifications would be required, if any, for the majority of other environments to be able to connect to our server.

References

1. Ray, Erik T. (2003). *Learning XML, Second Edition.* Sebastopol: O'Reilly.

2. Whali, et al. (2004). *Web Services Handbook*. IBM Redbooks.

Acknowledgments

I would like to acknowledge the assistance of Dr. Paul Wagner in the preparation of these materials. Also, I would like to acknowledge the help of Gabe Munoz, who spent countless hours setting up an environment for our simulations to run.