# Developing Web Services for Existing Java Applications

*Mindfire Solutions*

*www.mindfiresolutions.com*

*March 12, 2003*

***Abstract:***

*This paper discusses a RMI driven approach to building Web Services for an existing Java Application. We deal with the basics of JAX-RPC and steps in using Sun's Java Web Services Developer Pack (JWSDP). We describe the architecture and process used to implement a Web Services wrapper layer over an existing RMI-enabled Java application.*

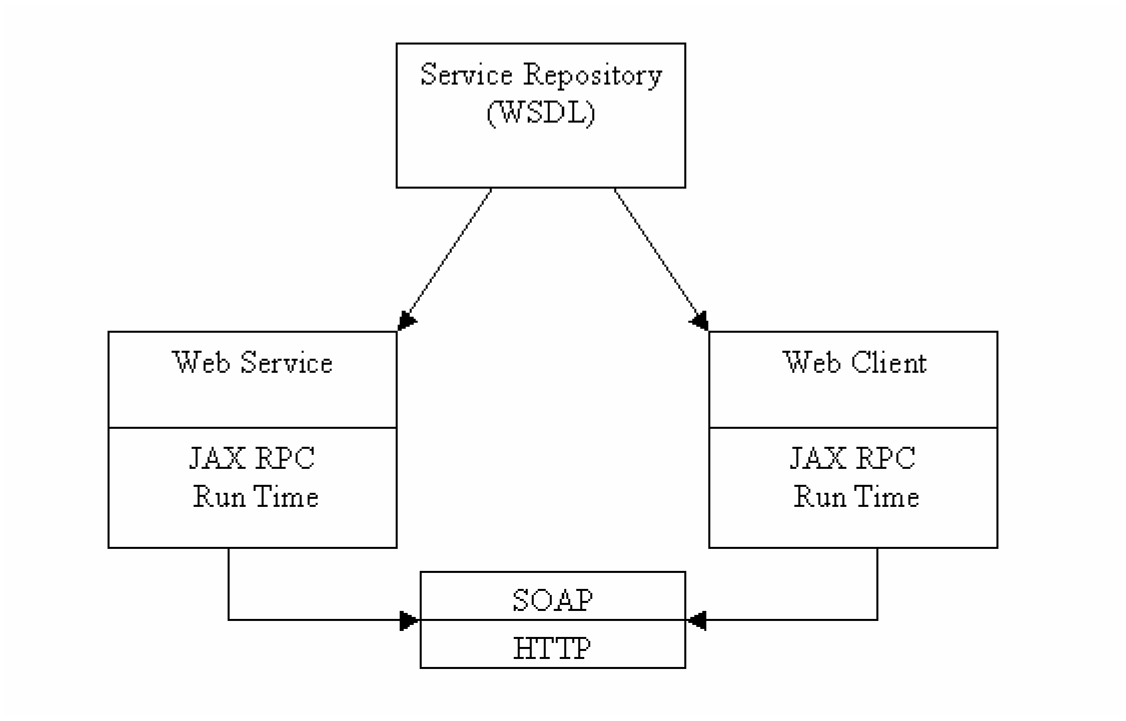
## Introduction

The purpose of this article is to share our experience of building a Web Service for an existing Java application, making it accessible over the Internet. In the process, we also discuss various aspects of JAX-RPC, a useful tool for deploying Java based Web Services.

## Deploying a Java based Web Service using JAX-RPC

There are a number of options available for deploying Java based Web Services. These include JAX-M, Apache SOAP, JAX-RPC etc. However JAX-RPC stands out as the best of these alternatives as it provides a number of useful features like the built-in WSDL-to-Java and Java-to-WSDL mapping tools, the "Deploy Tool" (a useful utility for managing Web Services) etc. It also provides automatic handling of low-level details like marshalling and un-marshalling of SOAP messages, creation of service end-points etc.

As expected, the JAX-RPC architecture is very similar to the Web Services model.



The JAX-RPC runtime system runs on both the client and the server. It automatically takes care of marshalling/un-marshalling messages between the client and the server. These messages (basically SOAP messages) are sent using the HTTP protocol.
JAX-RPC also provides both Java-to-WSDL and WSDL-to-Java mapping tools. The former generates a WSDL description of the service from the service's definition classes. However, we found that this tool cannot handle overloaded methods. The tool automatically renames an overloaded method by appending some characters to it. For example, two versions of some overloaded method "hello" might be called "hello" and "hello_1" in the WSDL document. The WSDL-to-Java mapping tool works on the client side to generate references (called stubs) to the service's methods from the WSDL document of the service. These stubs are used by the client program to call the service's

methods. Please note that unlike RMI (Java's version of RPC), the stubs are generated on the client side (and not on the server) and thus are not downloaded at run time.

## Using JAX-RPC, the First Step

The first task is to install the JAX-RPC runtime environment on both the server and the client machines (if the client is using JAX-RPC). You may get away without installing JAX-RPC on the machine on which the client application is executed (by supplying all the necessary class files), but JAX-RPC is necessary to compile the client with the Stubs (explained shortly). The JAX-RPC runtime along with the necessary tools for developing and deploying Web Services (and client applications) is available with the Java Web Services Developer Pack (The JWSDP download version is available at www.sun.com).

## Writing and Compiling the Service Classes

Once you have written the service's Interface and the Implementation classes, use the "ant" tool to compile the service and generate the WSDL document. This WSDL document describes the Web Service giving details of the methods available to the client applications. Although the JAX-RPC documentation also describes the "xrpcc" tool, the "ant" tool must be preferred as "ant" has deprecated "xrpcc". Also some of the Schemas used with the configuration files of "xrpccc" have been deprecated. To use the "ant" tool, certain configuration files are needed (these are XML files describing the behavior of "ant") and this is where your troubles begin. These files are not provided with JWSDP, but samples of these files are available with examples given in the JAX-RPC documentation. But again, these sample config files have been written specifically for these examples and you will have to adapt them for use with your Web Service. Alternatively, you could write these files yourself.

## Deploying your Web Service

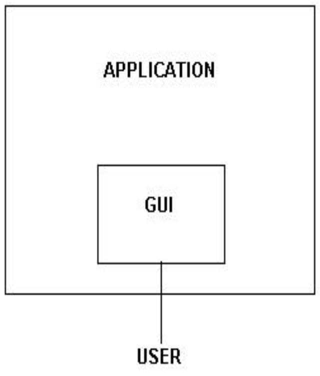
Once the deployable WAR(Web Archive) file has been generated, use the "DeployTool" provided with JWSDP to deploy your Web Service. Although this could have been done using "ant", we prefer the "DeployTool" because it is faster, and its GUI makes it more convenient to start and stop services.

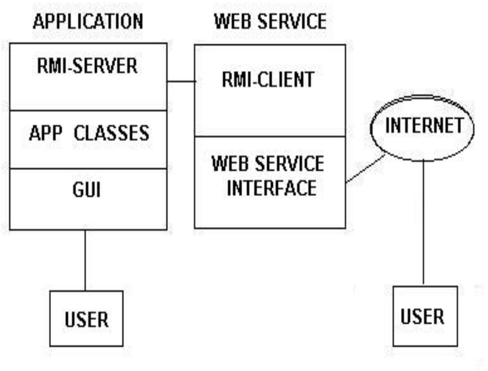
## JAX-RPC and the Client Side

The ant tool is used at the client side to generate the stubs from the Web Service's WSDL document. Again, you must have the appropriate config files ("Config.xml", in this case), specifying the WSDL's location. Once the stubs have been generated, compile the client-application with the stubs to generate the final application. The generated class files may be packaged into a .jar file if necessary. Note that having JAX-RPC runtime installed on the machine on which the client runs, is not necessary, as long as the class files used by the client application are provided.

## Discussion

Consider a simple Java based application, in which the user interacts with the application using the GUI. Our goal is to build a Web Service over it making the application available over the Internet. We adopt the following approach:





**Original Model** **The Web Service Model**

Although there are other alternatives available, we look at a three-layered architecture, as shown in the figure above. A new class(es) is added to our existing application, which provides a RMI (Remote Method Invocation) interface to our Web Service. The Web Service uses this RMI interface to communicate with the application. The Web Service itself consists of two layers. The first is the Web Service interface-it consists of the classes whose methods are called by the client applications. This represents the functionality of the application exposed to the remote users. The second is the RMI layer, which basically is the client for the RMI server built into the application. Whenever the Web Service receives a request, it is forwarded to the application using RMI. The RMI server fires appropriate events (or calls appropriate methods) within the application as if some input had been received from the GUI. The RMI server now responds to the Web Service by returning the response of the application, or by throwing appropriate exceptions.

The main advantage of using this approach is that it makes the Web Service and the application independent of each other, making the development and maintenance simpler. Another interesting side effect is that we get the flexibility of running the Web Service and the application on different hosts. This may be helpful for security critical applications as the end user never gets to know the actual location of application.

However, this approach also raises a few issues worth considering. The first one being that introducing a layer of RMI between the application and the Web Service would increase the response time of the system, as the delay caused by the RMI calls would be included in the response time. Secondly, you will have a HTTP server running from within your application for the automatic downloading of the Stub class. This will further increase the overhead. (This problem may be avoided by manually placing the Stub file in the deployable WAR file of the Web Service. But now, you will have to re-deploy the Web Service with the new stub class, every time some changes are made to the RMI layer of the application.)

Another aspect of providing a Web Service for an existing application is security/authentication. When the application is being made accessible over the Internet, some form of authentication, depending upon the nature of the application, needs to be provided. If the application provides some form of an authentication procedure, then the Web Service must use it before servicing any requests. However, if no such facility is available, then a security mechanism must be provided in the Web Service. This could be provided from within the Web Service module or by extending the application. The choice should depend upon the nature of the application and the desired level of security.

## Conclusion

JAX-RPC is a useful technology for building and deploying Java based Web Services. It combines the portability and robustness of Java with the convenience and flexibility of Web Services, giving developers a useful platform for providing Internet based services.

Building Web Services for existing applications with RMI as an interface between the application and the Web Service gives a flexible yet simple implementation of the server side system.