Paper 005-2009

# Building Analytic Services with SAS® Business Intelligence

John Leveille, d-Wise Technologies, Raleigh, NC
David Kratz, d-Wise Technologies, Raleigh, NC

## ABSTRACT

The rapid rise of the Internet has caused successive waves of change through the information industry. One of the latest overarching concepts is known as Service-oriented Architecture. Web services are one popular technology for creating a Service-oriented Architecture. This paper shows how one can use SAS BI Web Services to deploy SAS analytics through web services. Specific steps for developing a SAS Stored Process and deploying it as a web service are shown. Scenarios discuss the deployment of services within a corporate network as well as via an intercompany configuration. The audience for this paper is assumed to have a basic level of familiarity with XML and HTTP technologies.

## INTRODUCTION

Information systems and I.T. infrastructure are evolving, both within the enterprise and across enterprises via the public Internet. Organizations are seeking increasing levels of automation, real-time operations, and up-to-the-minute decision making. You can see the ripple effects of this accelerated change within your own projects as system requirements change during development. This rapid evolution has combined with a desire for flexible integration to give rise to the concept of Service-oriented Architecture (SOA). The web site JavaWorld describes SOA as, "an evolution of distributed computing based on the request/reply design paradigm for synchronous and asynchronous applications."

There are several legacy computing concepts which one can point to as forerunners to SOA – EDI, Message Queues, distributed computing, etc. However, for the purpose of this paper, it makes sense to think about SOA as an evolved form of data-driven web pages. Starting in the mid to late 1990's, web pages began appearing that allowed people to interact with databases behind those web pages. These data-driven web pages helped propel the Internet beyond marketing into full blown e-commerce, next-level information access, data sharing, and business-to-business (B2B) integration. These were welcome advances, however, the data-driven web pages have a problem. They are designed almost exclusively for a human to interact with the web application. SOA is intended for computer-to-computer interaction. It represents the evolutionary path for applications to step beyond data-driven web pages and point-and-click shopping carts.

The SAS Business Intelligence Platform is an excellent technology for bringing analytic services to the enterprise. The BI Platform leverages SAS Stored Processes for *both* traditional, data-driven applications and web services in the new SOA. Using a single set of SAS programs, you can provide analytics to traditional and next-generation systems, simultaneously. Web services technology compliments the BI Platform and enables you to develop computer-to-computer applications that harness the unique analytic power available through SAS products and procedures.

# SERVICE-ORIENTED ARCHITECTURE

SOA offers a variety of advantages to organizations that embrace it.

## ADVANTAGE: INTEROPERABILITY

Interoperability means that systems which were never specifically designed to "talk" to one another can be made to do so in a rich, effective, yet adaptable manner. Web services, the most popular technology for implementing SOA, follow W3C standards and have been adopted by a long list of software vendors. This large scale adoption promotes interoperability across systems, vendors, and organizations.

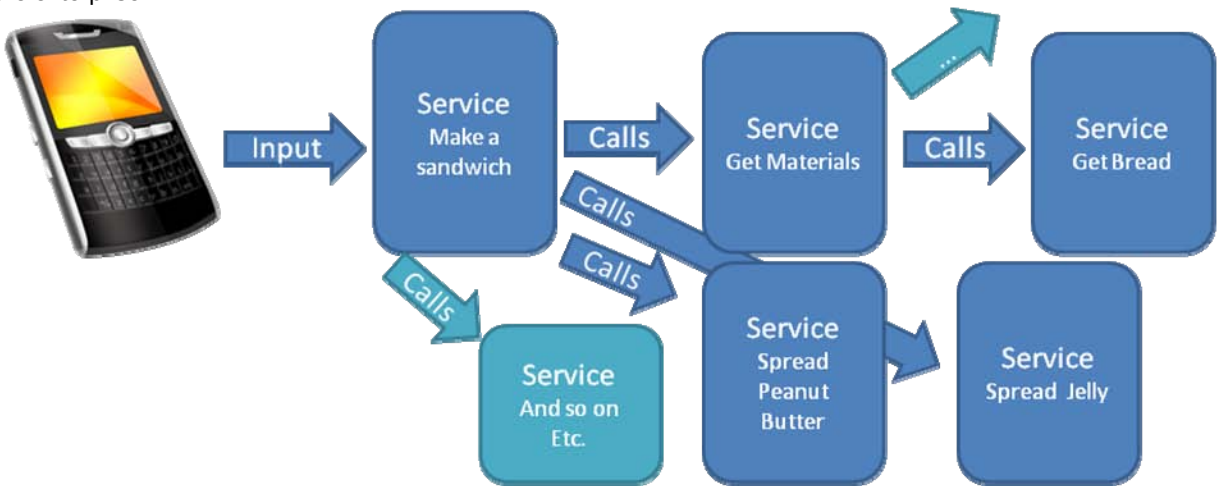
## ADVANTAGE: FIREWALL-FRIENDLY PROTOCOLS

At present, most SOA implementations are implemented using web services. These services are exposed through either the HTTP or the HTTPS protocols. Because of the rise of web-based applications, corporate networks allow much of the HTTP and the HTTPS traffic to pass through firewalls and security infrastructure.

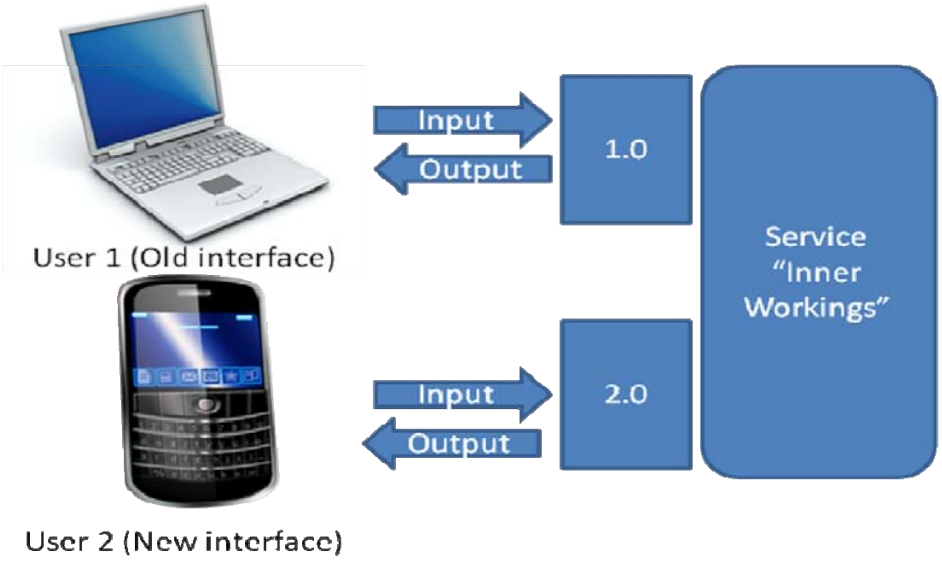
## ADVANTAGE: DELEGATION AND ENCAPSULATION

Just as objects within a computer application are designed for a specific purpose, each service typically provides a

discrete function. Services can delegate specific tasks to other services because it is assumed that all services are ready and available for use somewhere on the network. Delegation of tasks helps to reduce redundant programming in the enterprise.




Furthermore, each service provides an interface to systems that use the service. Over time, consumers of the service can utilize new interfaces while the older interface is maintained for support of older clients. The inner workings of the service are encapsulated behind these interfaces. You know how to use the service, but you don't know how it is implemented. This results in a weak coupling of systems easing the burden of maintenance and future integration changes.




### ADVANTAGE: MAXIMIZE THE USE OF SENSITIVE DATA

Every organization has some sensitive data. In the past it has been problematic to exploit these data across the enterprise and simultaneously maintain the security around them. A web service provides a functional interface to these data including authentication of client systems and fine grained access control. A single service can customize the security for each client that accesses it.

## PROTOCOLS, STANDARDS, AND IMPLEMENTATION

XML (e**x**tensible **m**arkup **l**anguage) is a plain text markup language, similar to html, which is used to transport data. Unlike html, XML's tags are user-defined, allowing for custom formatting. At present, XML forms the basis of the communication protocols used by SAS BI Web Services.

SOAP is a simple XML-based communication protocol which allows applications to communicate over HTTP. It provides a user-extensible way for applications running on different systems or languages to communicate with each other. Applications can make SOAP requests and get SOAP responses. This is the basic means by which a web

service operates. Here is an example of a very simple SOAP request for a fictitious web service that provides baseball scores

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
   <m:GetScore xmlns:m="http://www.d-wise.com/services/baseball">
      <m:Date>2008-07-01</m:Date>
      <m:Teams>
          <m:Team>Philadelphia</m:Team>
          <m:Team>Atlanta</m:Team>
      </m:Teams>
   </m:GetScore>
</soap:Body>
</soap:Envelope>
```

When the SOAP request is sent to the web service, it returns a response in XML format containing the result of the request. For example,

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
   <m:Score xmlns:m="http://www.d-wise.com/services/baseball">
      <m:Date>2008-07-01</m:Date>
      <m:Teams>
          <m:Team>
              <m:Name>Philadelphia</m:Name>
              <m:Runs>8</m:Runs>
          </m:Team>
          <m:Team>
              <m:Name>Atlanta</m:Name>
              <m:Runs>3</m:Runs>
          </m:Team>
      </m:Teams>
   </m:Score>
</soap:Body>
</soap:Envelope>
```

So XML and SOAP allow web services to communicate, but, given how free form XML can be, how does the requestor know the format for the request? That is where WSDL comes in. WSDL (web services description language) is a XML format for abstractly describing network services and binding them to a concrete protocol and message format. It is extensible to allow for intercommunication between dissimilar message formats and network protocols. Put simply, it tells you how to create the SOAP request and where to send it.

There are many proprietary WSDL specifications floating around the Internet for proprietary web services. There is nothing that prevents an organization or individual from simply writing a WSDL file and publishing a web service. However, there are also groups that have formed to create standards in WSDL to promote better harmony for similar services, usually focused on a specific industry such as banking, healthcare, energy, etc. SAS has chosen to use a standard for BI Web Services called XML for Analysis. XML for Analysis (XMLA) is a communications standard for client applications to call web services that interface with multidimensional or OLAP databases. Though the standard was established for OLAP applications, SAS has implemented it in a very generic way, making it suitable for just about any application.

XML and HTTP provide a means by which a client can use a web service, but these two protocols and the WSDL interface do not declare how the web service is implemented. They do not tell you what kinds of programs are running behind the scenes after the web service request is made. Web services can be implemented using any of a variety of languages and technologies. A SAS BI Web Service is implemented using a SAS Stored Process. This is a SAS program which is stored on a server and can be launched by a request external to that server. The process itself can (with few exceptions) do anything that a normal SAS session can do. To make use of SAS Stored processes, it is necessary to have some key components of the SAS BI platform installed. However, a full install of the BI platform is not required.

## A SAS BI WEB SERVICE EXAMPLE – PRODUCT CONFIGURATION SERVICE

### SCOPE AND FUNCTION OF THE SERVICE

The following example is a collection of web services in use at a fictitious audio electronics manufacturer. Together, these services comprise a system that is referred to as the "product configuration service." The manufacturer has a partially-outsourced sales and supplier business model. This model includes trusted partners who require access to inventory and product configuration details that are not accessible to the generate public. In addition, most of the partners have networks and computer systems which are isolated from the manufacturer's network and only connected to one another via the public Internet. The product configuration service was devised as a computer-to-computer solution that enables efficient function of both internal and external sales and supplier organizations while keeping centralized data and business logic within the primary corporate databases.

Suppliers interact with the electronics manufacturer by providing the delivery of product components, inventory availability plans, and options for alternative components to accommodate variability in pricing, demand, and marketing initiatives. Sales and customer service organizations interact with the manufacturer in the typical paradigm: checking inventory, creating customized orders, tracking orders, etc.

### REQUIRED SOFTWARE

In order to use SAS BI Web Services, as outlined in this example, you must have SAS Business Intelligence installed. However, you do not have to install all of the components in SAS Business Intelligence. For BI Web Services to function you must have a Metadata Server, a Stored Process Server, and SAS BI Web Services for Java or SAS BI Web Services for .NET. You also need an appropriate web application server that supports the flavor of BI Web Services you choose to install. For Java this would be Apache Tomcat or other equivalent, supported application server. For .NET this would be Microsoft Internet Information Server. You do not need the SAS Information Delivery Portal, SAS Web Report Studio, or other SAS web application in order to use BI Web Services.

### SPECIFIC SERVICES

For the purposes of this example, the business will have three functional services: *DescribeSystemComponents*, *CreateOrder*, and *TrackOrder*. Descriptively named, these will list the components in a given product, create and order, and track an order respectively. In order to illustrate the process of constructing a web service, the SAS code for the DescribeSystemComponents service is shown here:

```
/*DescribeSystemComponents.  Input = productID.  Output = list of System components*/

%let errorText=;
%validate; /*Adds to errorText if input is invalid*/

libname instream xml; /*For input*/
libname _WEBOUT xml xmlmeta=&_XMLSCHEMA; /*For output*/

%MACRO describesystemcomponents;
    %if %length(&errorText) = 0  %then %do;
            /*First perform a simple query for components.*/
            proc sql;
                    create table output as
                    select &productID as productID, b.ID, b.name,
                    b.description hifi.component b where b.productID = &productID;
            quit;

            /*Apply up-sale/cross-sale heuristics.*/
            %if %length(&suggest) > 0  %then %do;

               %upsell
               %crosssell

            %end;

            /*Now "print" the results back to the web service client.*/
            /*The copy proc sends the data and the XML libname engine*/
            /*will "print" it as XML.  The client receives this over */
            /*the network.                                           */
            proc copy in=work out=_webout;
                    select output;
            run;

    %end;
%MEND describesystemcomponents;

%describesystemcomponents
%errorOutput;/*Outputs an errorstatement if an error has occured.*/
```

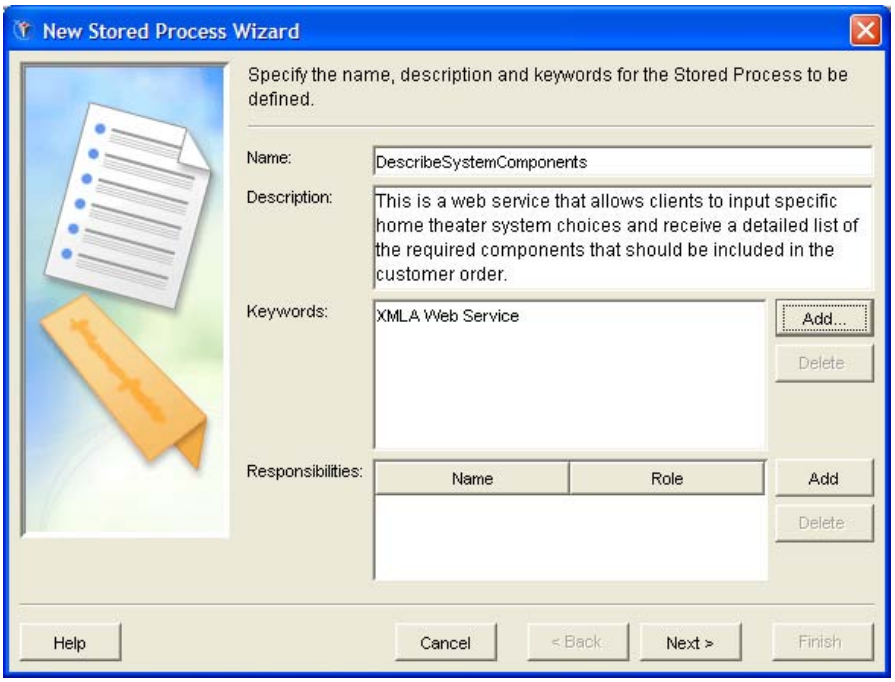
```
libname _WEBOUT clear;
libname instream clear;
```

The program displayed here is a SAS Stored Process. It is constructed as a simple SAS program that adheres to the documented specification of SAS Stored Processes. Please see the references section of this paper for more information on how to write a SAS Stored Process program. In the case of SAS BI Web Services, the SAS Stored Process accepts input from two sources: parameters and input XML. The product id is one of the parameters for the process, and the value passed by the client is stored in the macro variable productID. The libref instream is assigned as a mechanism for receiving XML data sent by the client.

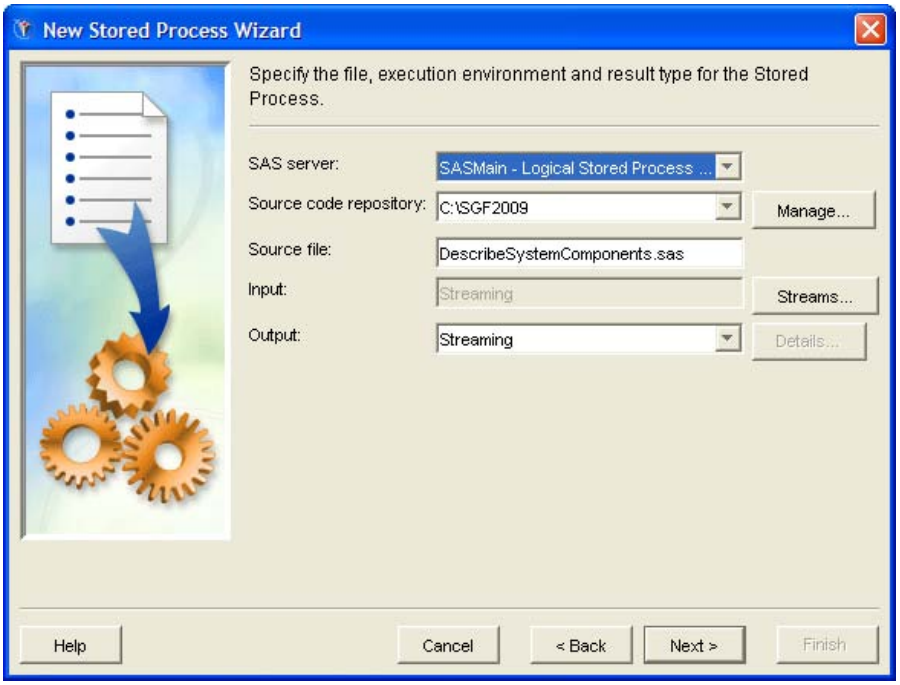
When the process receives a request it will use the product id to perform an SQL query against product and component tables to return the list of components which will be purchased and assembled into the requested product. In addition to the product id, this service also accepts a parameter which controls the invocation of upsell and crosssell heuristics. The upsell and crosssell macros contain the business logic and SAS analytics that score and match the customer and the request to the appropriate suggested, additional products and components.

After creating the SAS Stored Process code for the web service, you have to register the stored process and supply specific metadata that lets SAS know that this stored process is also a web service. This can be done using the SAS Management Console (SMC). The SAS BI Manager plug-in is required to register stored processes, so ensure that it is installed before attempting to follow the SMC steps below.

After launching the SMC and connecting to the appropriate metadata profile, expand the BI Manager node located in the left panel, under the Environment Manager node. Right Click on the BIP Tree folder and select the New Stored Process option. The following screen should appear:



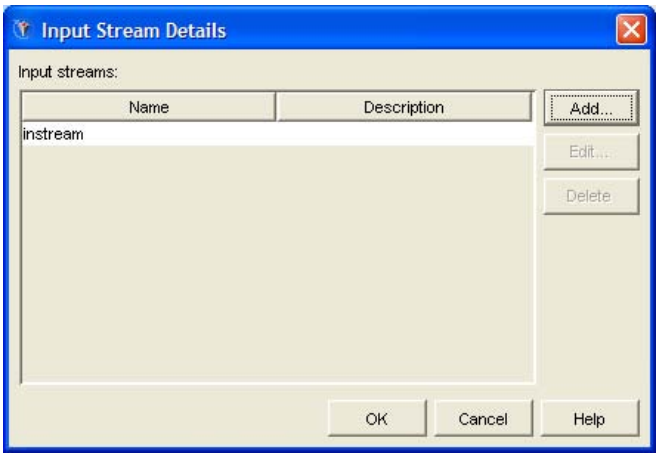

The description field is entirely optional, and the name should reflect the process being registered. The keywords field, however, must contain the entry XMLA Web Service. Without this keyword, the stored process will not be registered as a web service. Pressing Next will bring up the following screen:




When choosing the SAS Server, it is important to choose a stored process server. By default, the workspace server will appear an option, however it is not suitable to the task at hand. If no stored process server is available for selection, one will need to be created. Setting up a Stored Process Server is a topic outside the scope of this paper.

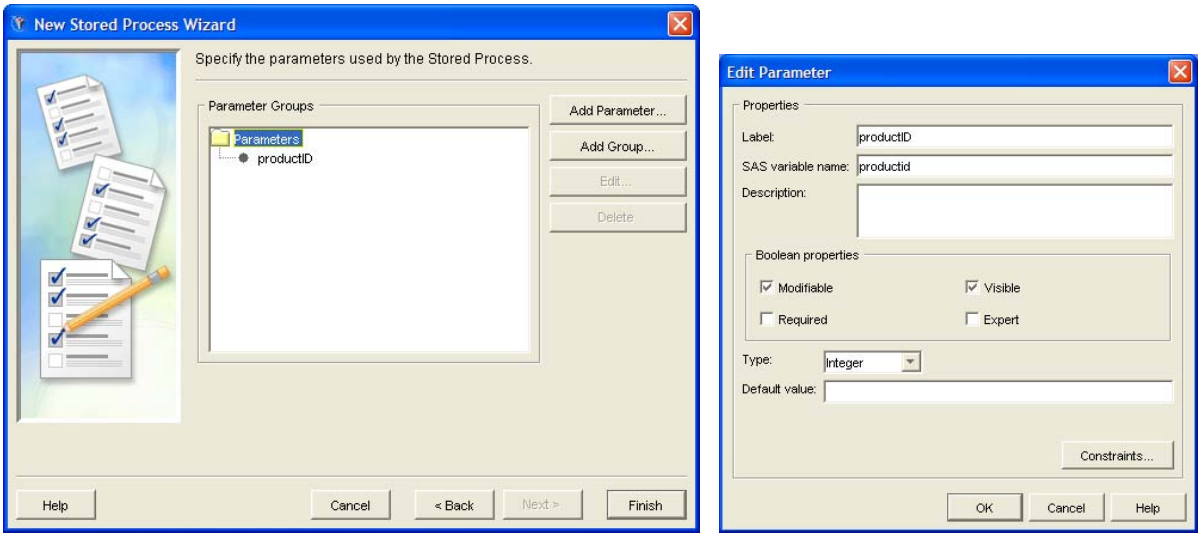
The source code repository option refers to the physical location where the source file resides on the server. The source file name provided must correspond to the exact name of a file located in the source code repository.

The input choice is particularly important. While the stored process does not have to accept any input, it won't be taking parameters without it. Clicking on the streams button will bring up the following dialog box:




Take particular note of the name associated with the input stream: instream. This corresponds to the input libname we set up in the earlier DescribeSystemComponents code. The name must be the same in both instances, or the web service will not function properly. There is a further complication: when adding the instream input stream the dialog box will feature a checkbox entitled "Supports multi-pass reads". This option needs to be checked.

Since it is desired that information be passed back when DescribeSystemComponents is run, select Streaming from the Output box. Clicking Next will bring up the following screen:




Click Add Parameter to reveal a dialog where you can edit the parameter details. Input "productID" for both label and SAS variable name. The label corresponds to what the value will be termed in the XML request, while the SAS variable name is the macro variable to which the value will be assigned. The other important option here is the Type field. For the purposes of this example it should be set to Integer. The parameter definition step is repeated for the "suggest" parameter.

Clicking finish will complete the process of registering the stored process as Web Service. At this point you can start your web application server with SAS BI Web Services and you have a working web service. There is nothing more for you to do. If the client programs know how to discover and invoke web services then they are able to begin using your service immediately. The remainder of the example code in this paper walks through the creation of a Java client for the DescribeSystemComponents web service. This detail is provided for those new to the topic, but it also of great value to experienced web service programmers who are new to SAS BI Web Services. There are nuances within the implementation of the client code that experienced programmers need to understand for a successful implementation using this platform.

## CLIENT PROGRAMS

There are many frameworks for developing web services and web service clients. The Java and .NET communities are both heavily invested in web service tools and frameworks, therefore, you have a broad spectrum of options for client implementation. For the purpose of this paper, we will show only a single web service client implementation. The example uses the open source Apache AXIS framework to develop a Java client for the DescribeSystemComponents web service. The code presented here was compiled using Apache AXIS 1.4 and Java 1.4.2.

The first step in creating the Java client for your web service is to get the WSDL file for the Web Service. In SAS BI Web Services, you can find the WSDL in the file name xmla.wsdl which is located in "C:\Program Files\SAS\Web\WebServicesforJava\1.0\SASXMLA" for the typical PC installation. Here is a preview of that file.

```
<?xml version="1.0" encoding="utf-8"?>
<q1:definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
                xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" … >

…

    <q1:operation name="Execute">
      <soap:operation soapAction="urn:schemas-microsoft-com:xml-analysis:Execute"
                      style="document" />
      <q1:input>
        <soap:body use="literal" />
      </q1:input>
      <q1:output>
        <soap:body use="literal" />
      </q1:output>
    </q1:operation>
  </q1:binding>

…
```

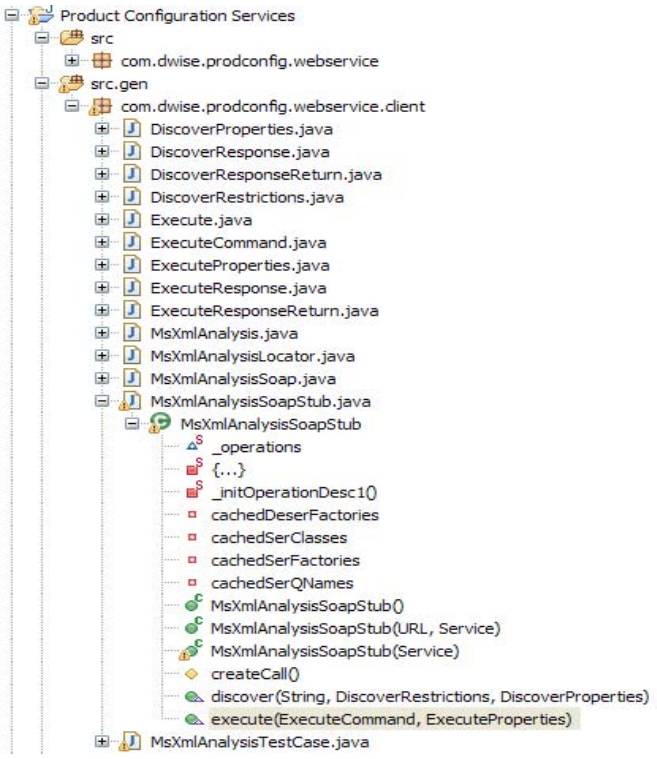
```
  <q1:service name="MsXmlAnalysis">
    <q1:port name="MsXmlAnalysisSoap" binding="s0:MsXmlAnalysisSoap">
      <soap:address location="http://localhost:8080/SASXMLA/SASXMLA" />
    </q1:port>
    <q1:port name="MsXmlAnalysisHttpGet" binding="s0:MsXmlAnalysisHttpGet">
      <http:address location="http://localhost:8080/SASXMLA/SASXMLA" />
    </q1:port>
    <q1:port name="MsXmlAnalysisHttpPost" binding="s0:MsXmlAnalysisHttpPost">
      <http:address location="http://localhost:8080/SASXMLA/SASXMLA" />
    </q1:port>
  </q1:service>
</q1:definitions>
```

In addition to the AXIS framework for web services, the Apache open source project offers an excellent build script tool called ANT. Using ANT you can script AXIS to quickly build out Java stub classes. The resulting Java programs form a client that can use the XMLA web service and call the Discover and Execute methods.

```
<!-- use axis to generate stub files -->
<target name="generate" depends="init">
   <axis-wsdl2java output="${src.gen}"
                   testcase="true"
                   verbose="true"
                   failonnetworkerrors="true"
                   printstacktraceonfailure="true"
                   url="${service.wsdl.sasStoredProcess}">
      <mapping namespace="urn:schemas-microsoft-com:xml-analysis"
               package="com.dwise.prodconfig.webservice.client" />
   </axis-wsdl2java>
</target>
```

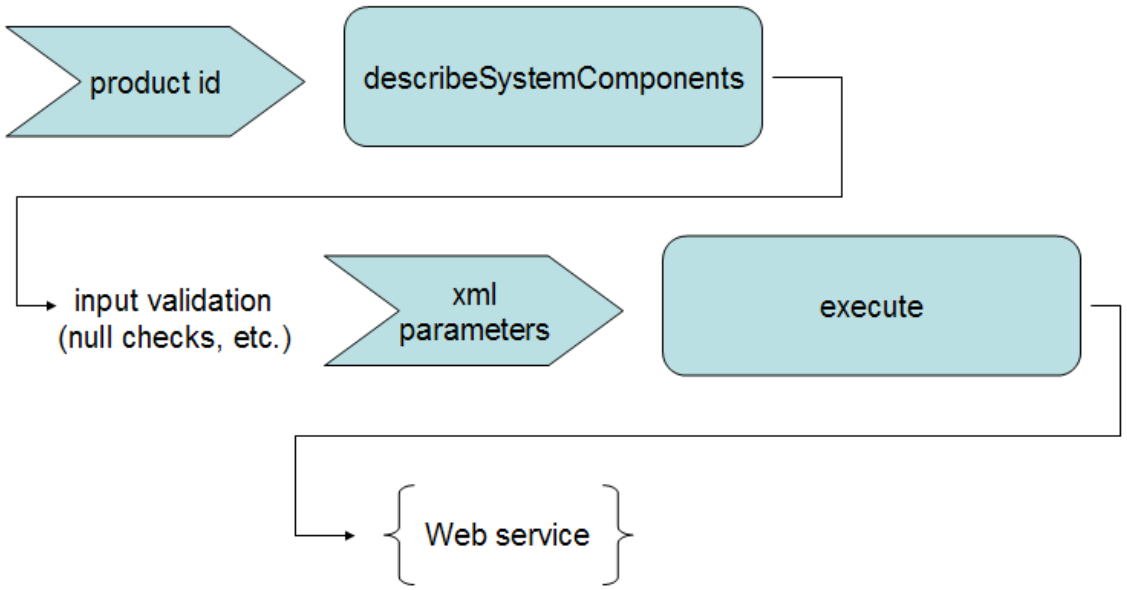
Within the build script the ANT task “wsdl2java” is invoked. This task reads the WSDL file from the url parameter in the task invocation. The wsdl2java task generates Java stub code and places it in the folder indicated by the output parameter. The result is a set of code that does a lot of the heavy lifting of talking http and SOAP, two protocols which must work together for the client program to invoke the web service.

After the client stubs are generated, the generated Java classes can be brought into a Java development environment like Eclipse and used for programming.



The figure here shows a listing of the source files contained in the Java project. Each java file within the src.gen folder is a program that was generated by wsdl2java. The MsXmlAnalysisSoapStub class is expanded to illustrate that there are several methods within that one program. The execute method is highlighted because it is the Java method that connects most directly to the SAS Stored Process within the BI Platform. This method name, “execute”, is generated from the operation name “Execute” in the WSDL file shown previously. The test case class generated by AXIS even shows you some example code for calling the web service through the generated Java code. You can open the program MsXmlAnalysisTestCase.java and read it to understand how to write your own client code.

Unfortunately, the WSDL specification that is provided by the SAS 9.1.3 BI Platform is a generic WSDL file for all analytic web services. SAS 9.2 is supposed to include features which support custom WSDL specification, but until that is released you have to adapt client code to a one-size-fits-all interface. This means that if you write a variety of web services and develop client code for each, you will always be calling an operation named “execute”. By the same token, the input parameters to the execute method are generic. Therefore, the calling program must create XML request data and stuff it into the execute method in a specific format which the web service is expecting. This extra bit of Java programming can be accomplished in your own Java class which calls the generated code. This custom class also gives you the opportunity to tailor the name of the execute method to something more appropriate for your application. For example, the method “execute” which accepts an undefined list of parameters can be wrapped by a method named “describeSystemComponents” that accepts a product id parameter and a flag for upsell/crosssell features.




Here is a partial listing of the describeSystemComponents method.

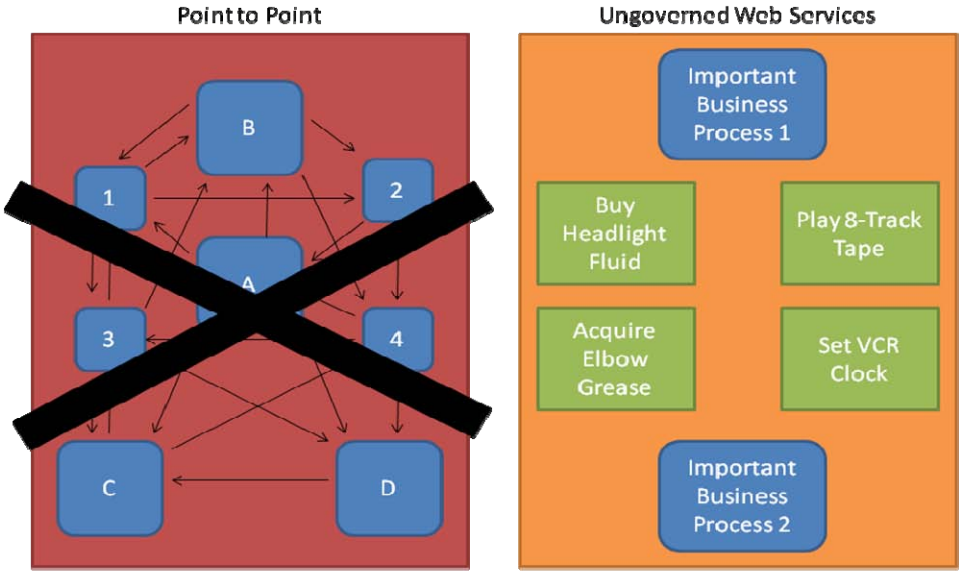
```
public List describeSystemComponents(String productId, String suggest)
    throws MalformedURLException, ServiceException,
           SOAPException, RemoteException {

   if (productId == null || productId.trim().length() == 0)
      throw new IllegalStateException("productId is null or blank.");

   String serviceUrl = "http://localhost:8080/SASXMLA/SASXMLA";
   String stpUrl = "SBIP://Foundation/DescribeSystemComponents(StoredProcess)";

   //get an instance of the client stub for the service
   MsXmlAnalysis serviceLocator = new MsXmlAnalysisLocator();
   MsXmlAnalysisSoap service =
      serviceLocator.getMsXmlAnalysisSoap(new URL(serviceUrl));

   //create XML element denoting the path to the Stored Process in metadata
   MessageElement me = new MessageElement("StoredProcess", "", "");
   me.addAttribute("", "name", stpUrl);

   //create XML element for each parameter being passed
   MessageElement me2 = new MessageElement("Parameter", "", "");
   me2.addAttribute("", "name", "productID");
   me2.setValue(productId);
   me.addChild(me2);
```

```
MessageElement me3 = new MessageElement("Parameter", "", "");
me3.addAttribute("", "name", "suggest");
me3.setValue(suggest);
me.addChild(me3);

//build the XML for the Execute message
ExecuteCommand command = new ExecuteCommand(new MessageElement[] { me });

//supply some additional properties required by SAS (see BI Web Service doc)
MessageElement pme =
   new MessageElement("PropertyList", "", "urn:schemas-microsoft-com:xml-analysis");

MessageElement pme2 = new MessageElement("DataSourceInfo", "",
                          "urn:schemas-microsoft-com:xml-analysis");
pme2.setValue("Provider=SASSPS;");
pme.addChild(pme2);

// build the properties using the message that was just formed
ExecuteProperties properties = new ExecuteProperties(new MessageElement[] { pme });

// run the web service!
ExecuteResponseReturn response = service.execute(command, properties);

//now upack the response XML to determine the result...
```

The remainder of the code for this method uses the Apache AXIS MessageElement class to deconstruct the web service response and place the information into a list that the calling program can easily consume. The method that packs and unpacks the input and output for the web service call may seem complex, but you only have to write this code one time. Once it is completed, the complexity is encapsulated, insulating client applications from any knowledge that they are calling a SAS BI Web Service.

## SOA CHALLENGES

Service-oriented architecture is not without its challenges. One challenge is governance. When the emphasis is shifted from point-to-point system integrations to development of services, there is a potential for the organization to develop a hodge-podge of services and no clear definition about what they are or how to use them. This does not tend to happen in a point-to-point approach because all facets are strongly defined and narrowly scoped. The point-to-point integration is nothing without connecting its endpoints. The web service, on the other hand, can be something without a concrete client application. However, that something could be: a useless service.




Another reason for governance is to enforce some standards and manage the technical complexity. Although web services are touted as being interoperable, programmers will find that differences exist because vendors are pushing the existing specifications to the limit. As a result, it is possible to develop services that cannot be used by all clients

or do not easily integrate with all languages and frameworks. SOA governance can keep the organization away from the "bleeding edge" of technology and produce a more effective end result.

Web services are currently the most popular strategy for implementing SOA. Unfortunately, web services have specific technical challenges, as well. One major limitation of web services is the lack of server outcalls. In a traditional client-server application the client connects to the server and maintains the network connection for the duration of the client-server interaction. This is called a stateful connection. In contrast, web services are built on top of http, a stateless protocol. The typical client-server architecture allows the server to send spontaneous messages to the client such as "please refresh your display", however, the stateless protocol means that web services do not have this capability. If the web service were configured to monitor events in a database and an event occurred, the web service cannot make an http connection out to the client to send the message.

Another difficult task in a web service is to deliver graphics or other binary data as part of the web service response. Since XML is a text language, inserting a binary data stream inside the XML text can break the integrity of the message. To get around this you can use an escaping or encoding scheme for the binary data which adds complexity and a proprietary aspect to your web service.

## CONCLUSION

Web services are a powerful technology for implementing a Service-oriented Architecture within your organization. SAS BI Web Services enable you to exploit the power of SAS analytics by easily turning your SAS Stored Processes into web services. With SAS behind the web service, you can provide powerful information by running statistical models, querying OLAP cubes, computing complex measures, and more. The same programs you have used in the past to drive web pages and batch jobs can lead the way in evolving your corporate systems with a modern Service-oriented Architecture.

## REFERENCES

Christensen, Erik, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. World Wide Web Consortium, "Web Services Description Language (WSDL) 1.1." http://www.w3.org/TR/wsdl

Kodali, Raghu R., JavaWorld.com, "What is Service-Oriented Architecture?" http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa.html

*Opensource Authorship.* Apache Software Foundation. "Apache Web Services Project." http://ws.apache.org/axis/

*Opensource Authorship.* Apache Software Foundation. "The Apache ANT Project." http://ant.apache.org

SAS Institute, Inc. SAS Product Documentation. "SAS BI Web Services." http://support.sas.com/rnd/itech/doc9/dev_guide/websrvcs/index.html

SAS Institute, Inc. SAS Product Documentation. "SAS Stored Processes." http://support.sas.com/rnd/itech/doc9/dev_guide/stprocess/index.html

*Undetermined Author*. W3Schools.com. "Introduction to SOAP." http://www.w3schools.com/soap/soap_intro.asp

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

John Leveille
d-Wise Technologies, Inc.
3115 Belspring Ln
Raleigh, NC 27612
Tel: 888-563-0931 x101
jleveille@d-wise.com
http://www.d-wise.com

David Kratz
d-Wise Technologies, Inc.
3115 Belspring Ln
Raleigh, NC 27612
Tel: 888-563-0931 x107
dkratz@d-wise.com
http://www.d-wise.com