



Virtuoso Web Services

Prologue

Today's clamor over Web Services and the SOA buzz should be viewed as a 'promise' from the working groups of the W3C made to you, the IT professional. These dedicated folk work unstintingly in the vineyards of the committees, fostering vast bodies of technical guidance that should eventually, make your life easier.

The extent to which this promise is delivered upon is highly dependent on the tool smiths who adopt these far reaching standards, and provide servers, programming languages, and other tools *yet to be conceived*. The contemporary tools and infrastructure market is indeed crowded with good tools and bad, by vendors large and small, in a noisy Web Services world.

The SOA model is a new way to think about internal and external services composition and invocation. Connecting, combining, reusing, and collaborating - all of these things *should* be easier in the open world of XML based Web Services.

Many IT professionals have heard of Web Services, read about SOA, and toyed with the idea of the Enterprise Services Bus. However, though the time seems ripe for implementing these new technologies in a working IT division, many hesitate before venturing forward, and understandably so.

The central issue should be 'rollout trauma'; the best solution is a comprehensive server and *services composition environment* that creates the least havoc during pilot testing and production rollout.

Virtuoso Universal Server Web Services fills the Web Services Promise

Abstract

This Whitepaper examines OpenLink Software's Virtuoso Universal Server. Virtuoso provides a comprehensive single server SOA (Service Oriented Architecture) based on standard Web Services protocols. With a potent focus on 'minimal disruption' of

existing IT operations, OpenLink delivers a viable solution for those stewards ready to take steps towards a Service Oriented Architecture, without pulling the entire enterprise down around one's ears.

Virtuoso Universal Server publishes existing application logic as standards compliant Web Services. This 'exposure of services' can be executed incrementally and tested on a module by module basis, all within a production or staged environment. Virtuoso Web Services are published from existing program modules written in Java or .Net languages, or the Virtuoso SQL procedure language. Virtuoso is a complete Web Services platform, and a consumer of externally published Web Services from trading partners or external data/services providers.

Making the most of 'time-tested' code is a key feature of Virtuoso Universal Server. Code re-use dramatically improves productivity, quality and speed of delivery. Virtuoso also provides data transformation services to liberate relational data into the XML domain - a prerequisite for deploying Service Oriented Architecture using Web Services.

XML Based Web Services

'Web Services', describes a set of XML based protocols for exchanging messages between applications. A general description of such services might read as follows:

- Message Envelopes
- Message formats
- Message Attachments
- Message Routing
- Services Description
- Service Invocation
- Service Advertising

Web Services provide a standard framework for reusing and sharing services from an IT organization's pool of existing applications. These standards assist in the modular *assemblage* of new services or from working code packages and collections of services, and allow the exposure of these proven functions through a layer of XML protocols.

Orchestration of these processes over the Internet - by a partner or subscriber for instance, is the justification of Web Services. These voluminous standards comprise a universal architecture for simplifying application to application communication, and integration of data across the internet/intranet/extranet. Web Services enable an IT organization to build, extend and expose applications such as publicly available web-based stock quotes, to complex business-to-business partner integrations.

All of the foregoing issues are common problem areas, and there are several

well-worn and traditional ways of addressing these integration and accessibility issues. Web Services are simply the latest act in the drama of industry convergence standards.

Web Services are central to interoperability?.

As stated in the Frontispiece prologue , Web Services are merely a 'promise of potential' in attaining the Real-Time Enterprise (RTE). The attributes of this seductive vision - the ability to rapidly respond to changes in business climate, or to quickly shape and sculpt technical partner relations in a timely manner, is our lofty goal.

Roadblocks abound in the form of scattered and incompatible data sources, applications, and processes. These inevitable and confounding artifacts of daily life in the IT department make it nigh impossible for any organization to reach this RTE vision. There is a hard truth in realizing the fact that this data and systems cacophony is a fact of 'corporate IT life?'. With the proliferation of Databases, Middleware, Languages, and operating systems, even single vendor solutions are a dead end in realizing the vision of a real time enterprise.

Conceptualizing the Promise of Web Services

The RTE vision requires clear-headed principles for identifying and reusing services from existing LOB applications. In addition, we need standards for interface definition, invocation, and message exchange. All of the foregoing is needed to share application functions and create partner processes - whether within or without the corporate firewall.

Service Oriented Architecture (SOA) is the catch-all term for creating accessible services from previously hard-wired monolithic application logic. Similar principles should also be applied when developing new business solutions in a brave new web services world.

Applications or fractional code modules exposed or composed within a Services Oriented Architecture do not exist within a vacuum; SOA merely provides a conceptual framework for service assembly. SOA = Web services - a standardized protocol stack and interface specification expressed in XML.

Today's business applications have already become increasingly modular with the advent of object languages. Such code lends itself to decomposition into component parts that may be individually or collectively invoked using Web Services protocols. Such a transformation implies that our tried and true application logic is no longer bound within our monolithic corporate application stew, and is now available as an executable function via the ubiquitous URI (http://codebase1.an_api_function.com). Some revel in this magic.

SOA and therefore Web Services provide a mechanism allowing the public (or secure trading partners) to consume a company's services over the public internet. This commonly takes the form of a machine to machine (web service) transaction sans browser - invoked over HTTP.

Web Services provide rich soil for growing new types of composite service offerings, potentially extending any vendor's reach through its network of partners. Any company, whether in B2B or B2C, can expand and streamline its value chain through Web Services.

Delivering on the Promise

The Promise of SOA is quite literally fulfilled by your tools and server infrastructure vendor. To the extent that a Web Services vendor knows the problems inherent in data and application integration, such will be the level of your IT organization's success in implementing a new paradigm like SOA.

As a Universal data access middleware vendor with a twelve year lineage, OpenLink Software understands that each business runs on time tested applications. Core LOB systems run on relational databases, often implementing business logic as stored procedures or as code at the application server layer.

Virtuoso Universal Server makes connecting to applications, modular functions, and diverse data sources (independent of location), a reality. Employing a virtual database that connects and transforms relational data to XML in real-time, Virtuoso seems the natural connector between the Web Services world and existing IT infrastructure..

Virtuoso Web Services Protocol Suite

Overview of Web Services in Virtuoso

Virtuosos Universal Server implements the entire Web Services (WS) protocol suite. All modern aspects of SOA WS-based inter-application communication and data access are supported. Each protocol layer is a foundation for higher abstract process functions. The following is a bottom-up exposition of the Virtuoso WS stack.

SOAP

Simple Object Access Protocol, is the lingua franca of the WS suite. SOAP is an XML format for transferring complex data and context information between computing

machinery and processes.

Designed to be operating system and programming language neutral, SOAP provides an extensible framework for representing data, character encodings, meta-information, and remote services requests.

SOAP is the messaging transport and conveyance method, which includes transaction contexts, encryption, security credentials, error conditions, message routing and other information.

Virtuoso Universal Server was designed to keep the CIO, IT Admin, and programmer as far away from SOAP as one cares to be - except in the bath.

WSDL = the Dope on SOAP

Web Services Description Language is the XML format for describing SOAP messaging interfaces.

A SOAP method, message, or procedure, is optionally bound to a WSDL description. Another way of saying this is that an instance of WSDL must be generated for each set of one or more SOAP objects that we intend to expose.

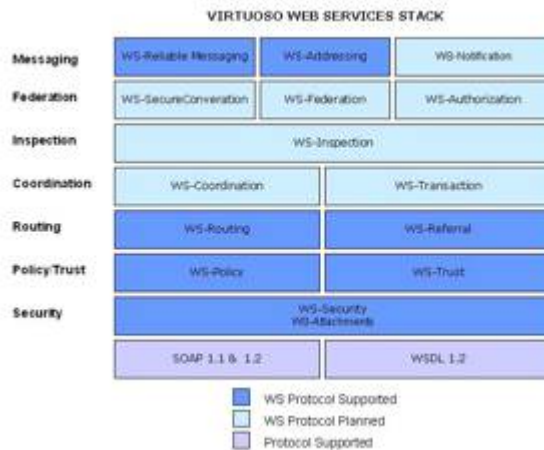
WSDL defines data types, parameters and return values for incoming services traffic. The WSDL file is automatically generated based on the function prototype and data types exposed at a SOAP endpoint. The nomenclature of 'end-point', is colorful, however its simple meaning is defined as, 'the last item to be executed as the result of a URI invocation'.

The WSDL file also makes it possible to generate stubs in most programming languages for invoking procedures exposed by a SOAP server.

WS Protocol Stack

Diagram 1 depicts the current WS protocols that are supported in Virtuoso and the protocols that are planned in the upcoming releases.

Diagram - Virtuoso Web Services Protocols [\[1\]](#)



Security

Virtuoso provides an environment for secure Web Services by implementing WS-Security, which provides a methodology for digitally signing and encrypting SOAP messages.

WS-Trust complements WS-Security by providing a standard for requesting and validating certificates and other security tokens used in SOAP messages secured with WS-Security.

Messaging

WS-Reliable Messaging (WSRM) is a protocol for guaranteed asynchronous message delivery between endpoints. This standard captures the essential functions of a message queue, allowing for guaranteed reception and guaranteed order of reception in a stream of one-way messages between peers.

WS-Routing allows specifying a complex chain of message delivery intermediaries for SOAP service invocation.

WS-Referral provides a way for notifying users of Web Services of changes in service providers, redirecting a client to an alternate service provider. It is a stateless protocol for inserting, delete and query routing entries in a SOAP router.

Together these protocols form a comprehensive whole suited for almost any application-to-application interface situation.

Bringing it all Together with Virtuoso Universal Server

What is a SOAP Endpoint?

A SOAP endpoint is the location, described by a URL, where a Web Service is invoked (consumer) and executed (publisher). Every Service action requires at least one endpoint on which to run.

How Does it Work?

A Virtuoso server may listen at multiple port addresses for HTTP clients. Each HTTP listener may contain multiple SOAP URI end points. A WSDL description file is automatically associated to each end point.

Each SOAP end point specifies a SQL user account. Stored procedures associated with this account will be exposed as services of this end point URI. These SQL accounts comprise the first line of security, and should only be used for the granting of necessary privileges.

The end point can additionally declare that it requires incoming calls to comply with different levels of WS-Security, for example accepting only signed and/or encrypted messages.

Procedures that are published may declare a precise XML Schema type for their return value and each of their parameter calls, as well as a call style, which may be either RPC or document. A special SQL declaration is reserved for mapping a user defined type (SQL declaration similar to a subroutine) to a SOAP structure type. The SQL user defined type can additionally declare explicit SOAP data types for its members.

Thus it is possible for an application to comply with a given WSDL file by declaring the exact SOAP types of all operations. In the absence of these declarations, default SOAP types are inferred from the SQL types.

Example: Setting Up an End Point

A SOAP end point is created using Virtuoso's virtual directory- essentially a web server executable services function. Endpoints may also be created programmatically using SQL procedures. End points will specify: URL, SQL account credentials, and WS-Security requirements of the end point.

Virtuoso Web Services Examples

Publishing a Stored Procedure and Structure Type

The following exhibit in Figures 1 and 2 shows an example of a Web Service for stock quotes. The example returns a structure with the days high and low as well as the quote and the day's trading volume.

Figure 1 details the SQL type for holding the data, which includes the quote, the day's high, the day's low, weeks high and low as well as volume.

Figure 1 - SQL Type for storing the Stock Quote Data

```
create type "quote_data"
as (
    "quote" real,
    "day_low" real,
    "day_high" real,
    "week_low" real,
    "week_high" real,
    "volume" real
)
;
```

The Soap representation is seen below in Figure 2.

Figure 2 - SOAP representation for the Stock Quote Structure

```

soap_dt_define (',
  '<complexType name="SOAPQuoteStruct"
    xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
    xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="services.wSDL"
    xmlns:tns="services.wSDL">

    <all>
      <element name="quote" type="float" nillable="true"/>
      <element name="day_low" type="float" nillable="true"/>
      <element name="day_high" type="float" nillable="true"/>
      <element name="week_low" type="float" nillable="true"/>
      <element name="week_high" type="float" nillable="true"/>
      <element name="volume" type="float" nillable="true"/>
    </all>
  </complexType>', 'quote_data'
)
;

```

In Figure 3, a stored procedure is created, which will return, a stock quote object given the ticker. Note the SOAP type declarations. These are needed if we wish to define a SOAP type bound to a WSDL file.

Figure 3 - Stock Quote Stored Procedure

```

create procedure "getQuote (in ticker nvarchar)
  returns quote_data __soap_type 'services.wSDL:SOAPQuoteStruct'
{
  declare sst quote_data;
  sst := new quote_data();
  select REC_DAY_LOW, REC_DAY_HIGH, REC_WEEK_LOW, REC_WEEK_HIGH, REC_VOLU

  into sst."day_low", sst."day_high", sst."week_low", sst."week_high", ss
  from REC_QUOTE_DATA
  where REC_QUOTE = ticker  ;

  return sst;
}
;

```

Figure 4 - Resulting WSDL file for the above operations

```

<types>
  <schema targetNamespace="services.wsdl"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <complexType name="SOAPQuoteStruct">
      <all>
        <element name="quote" type="float" nillable="true"/>
        <element name="day_low" type="float" nillable="true"/>
        <element name="day_high" type="float" nillable="true"/>
        <element name="week_low" type="float" nillable="true"/>
        <element name="week_high" type="float" nillable="true"/>
        <element name="volume" type="float" nillable="true"/>
      </all>
    </complexType>
    .....
  </types>
  .....
  <message name="getQuoteRequest">
    <part name="quote_name" type="xsd:string" />
  </message>
  <message name="getQuoteResponse">
    <part name="CallReturn" type="ns0:SOAPQuoteStruct" />
  </message>
  .....
  <portType name="SOAPPortType">
    <operation name="getQuote" parameterOrder="quote_name">
      <input message="tns:getQuoteRequest" />
      <output message="tns:getQuoteResponse" />
    </operation>
    .....
  <binding name="SOAPBinding" type="tns:SOAPPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/ht
    <operation name="getQuote">
      <soap:operation soapAction="http://openlinksw.com/virtuoso/soap/schem
      <input>
        <soap:body use="encoded" namespace="http://openlinksw.com/virtuoso/
      </input>
      <output>
        <soap:body use="encoded" namespace="http://openlinksw.com/virtuoso/
      </output>
    </operation>
    .....

```

Publishing Services in Hosted Languages

Virtuoso may host programs coded in Java and the .Net languages. This logic may also be exposed as a Web Service. This is done by having Virtuoso automatically create a procedure wrapper invoking the hosted language.

If structures or variables of the hosted method or class are to be transferred to persistent storage, they will first be mapped to a SQL type, which declares the precise SOAP types for the members.

The following exhibits show the procedure for hosting a stock quote in Virtuoso. The first step is to create an object method or class in the Java. Figure 5 shows the definition of our stock quote class written in Java.

Figure 5 - Defining the stock quote class in Java

```
public class SOAPJavaQuote implements java.io.Serializable
{
    public float quote, day_low, day_high, week_low, week_high, volume;
    public SOAPJavaQuote()
    {
        quote = 1;
        day_low = 2;
        day_high = 3;
        week_low = 4;
        week_high = 5;
        volume = 6;
    }
}
```

In Figure 6 and 7, the commands used to import the class into Virtuoso is illustrated as a hosted type and the SOAP definition of the Java object.

Figure 6 - Importing the class into Virtuoso as a hosted type

```
import_jar (NULL, vector('SOAPJavaQuote'));
```

Figure 7 - Defining a SOAP rendition of the Java object.

```
soap_dt_define ('',
    '<complexType name="SOAPJavaQuoteStruct"
        xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
        xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="services.wSDL"
        xmlns:tns="services.wSDL">
    <all>
        <element name="quote" type="float" nillable="true"/>
        <element name="day_low" type="float" nillable="true"/>
        <element name="day_high" type="float" nillable="true"/>
        <element name="week_low" type="float" nillable="true"/>
        <element name="week_high" type="float" nillable="true"/>
        <element name="volume" type="float" nillable="true"/>
    </all>
</complexType>', 'SOAPJavaQuote');
```

Figure 8 details the stored procedure, which will retrieve the Java quote object from a table and return it to the SOAP client. Note that we could also call a Java method as a user defined type for making a quote object.

Figure 8 - Stored procedure for retrieving the Java Stock Quote object

```

create procedure "getJavaQuote" (in quote_name nvarchar)
    returns "SOAPJavaQuote" __soap_type 'services.wsdl:SOAPJavaQuoteStruct'
{
    declare sst "SOAPJavaQuote";
    select JQ_DATA into sst from JAVA_QUOTES where JQ_QUOTE = quote_name and

    JQ_DATE = now();

    return sst;
}
;

```

Consuming Web Services

WS-Security

Virtuoso stored procedures can invoke SOAP services published and saved in Virtuoso or by other sites.

In the following example shown in Figure 9, the quote service 'getJavaQuote' described in Figure 8 is included along with using WS-Security for encrypting the communication.

Figure 9 - Example using WS-Security

```

xenc_key_3DES_rand_create ('wss-3des', '!secret!');

resp := SOAP_CLIENT (url=>targetUrl,
                    operation=>'getJavaQuote',
                    parameters=>vector ('quote_name'. 'YHOO'),
                    auth_type=>'key',
                    ticket=>xenc_key_inst_create ('wss-3des', xenc_key_i
                    security_type=>'encrypt'
                    template=>'[secret key name]');

```

So far, we have been interested in SOAP and its WS-Security extension.

WS-Routing also provides a means to pass messages through a complex chain of intermediaries. Virtuoso will automatically recognize the routing headers in incoming messages. If it is not the final recipient of a message it will forward the message, after optional security checking, to the next recipient in the chain. It will likewise forward replies back to the original requester.

As seen in Figure 10 below, WS-Referral allows indicating that the provider of a service has changed. All requests matching certain criteria can be flagged as now supplied by another server.

Figure 10 - Configure the end point to refer the stock quote message to a second server (this is done by client's request setting up a routing rules WS-Referral).

```

declare result any;
declare str varchar;
declare header any;
declare action varchar;
declare register any;
declare resp, router, endpoint any;

router := 'http://localhost:'||server_http_port()||'/router';
action := 'http://schemas.xmlsoap.org/ws/2001/10/referral#register';
header := xml_tree_doc (
  '<m:path xmlns:m="http://schemas.xmlsoap.org/rp">' ||
  ' <m:action>' || action || '</m:action>' ||
  ' <m:rev>' ||
  ' <m:via/>' ||
  '</m:rev>' ||
  '<m:id>uuid:' || uuid () || '</m:id>' ||
  '</m:path>'
);

register := xml_tree_doc (
  '<r:register xmlns:r="http://schemas.xmlsoap.org/ws/2001/10/referral"
  <r:ref>' ||
  ' <r:for>' ||
  ' <r:exact>' || endpoint || '</r:exact>' ||
  '</r:for>' ||
  '<r:if>' ||
  ' <r:ttl>36000000</r:ttl>' ||
  '</r:if>' ||
  '<r:go>' ||
  ' <r:via>' || mirror || '</r:via>' ||
  '</r:go>' ||
  ' <r:refId>uuid:E64134EE-1F0D-11D7-9F95-FD3FB2DAD2D7</r:refId>' ||
  '</r:ref>' ||
  '</r:register>'
);

resp := soap_client (url=>router,
                      operation=>'register',
                      parameters=>vector ('register', register),
                      style=> 1, -- Document Literal Encoding
                      headers => vector ('path', header),
                      soap_action=>action
                      );

```

End points may be configured to examine en-route WS-Security routing message headers, forwarding messages to designated recipients.

WS-Reliable Messaging (WSRM) guarantees that transmissions arrive in numbered order, and provides a recovery mechanism for a series of one-way transmissions via persistent logging of incoming and outgoing messages in the Virtuoso database.

WSRM is useful in situations involving lengthy WS conversations, and may provide asynchronous replies transferred by connections established at a later time.

The following example in Figure 11, uses WSRM for sending purchase orders. The receiving server processes these in a batch and uses WSRM for sending an order status confirmation.

Figure 11 - Sending an Order via WS-Reliable Messaging

```
declare addr wsa_cli;
declare wsrcli wsr_cli;
declare req soap_client_req;

addr := new wsa_cli ();
addr."to" := _to; -- the ultimate destination; newOrder service endpoint
addr."from" := _from;
addr.action := 'http://temp.uri:newOrder';

req := new soap_client_req ();
req.url := _to;
req.operation := 'newOrder';
req.parameters := vector ('OrderID', order_id,
                          'ProductId', product_id,
                          'Quantity', quantity,
                          'CustomerId', customer_id);

wsrcli := new wsr_cli (addr, _to);
wsrcli.send_message (req);
wsrcli.finish (req);
```

Hosting .Net Web Services

Virtuoso provides no-code hosting of Microsoft .Net Web Services .asmx files.

Inserting an .asmx file in a Virtuoso virtual directory will make asmx Web Services accessible to clients. The mechanism used for exposing these files is similar to ASP .Net hosting in Virtuoso.

Thus, Virtuoso is a viable alternative to Microsoft's IIS for .asmx hosting, and .asmx services may be cross-platform hosted on Unix through Virtuoso's integration of the Mono CLR run time.

Conclusion

For the busy but curious IT professional, OpenLink's Virtuoso Universal Server covers the SOA services spectrum with aplomb. Virtuoso operates as a data and application API junction box, mediating between clients and servers using Web Services standards.

Virtuoso's local storage and processing capabilities make it well suited for intelligent

data conversion, logging and message dispatching. With the move to SOA made up of XML's whole cloth, the world of Corporate databases needs products like OpenLink's Virtuoso to transmute the relational data driving our LOB applications into SOAP objects for universal and standards-based access.. The real-time Enterprise is now just a stone's throw away.

[\[1\]](#) See table 1 in the Appendix for details on the industry specification status