Roger's Rules For Web Services

If you’re not using standards for design, definitions and transport, you’ve got something, but it’s not a Web service

By Roger Sippl

November 1, 2005 — Ted Codd, the father of the relational model and the database products based upon it, published in 1985 a set of rules for what defined a relational database. Usually you will find 12 rules in this list, but Codd was a pretty strict guy, and there were actually 13. Rule 0 was that you had to use the database for everything you did with the data in it.

Rules like this can be a great help because sometimes people don't know what they “should” do when a new technology comes out. Back in Codd’s day, it was relational DBMS, but today it is Web services and the concept of service-oriented architecture (SOA). We all need an old guy (me in this case) to give us some clues. The new notion of a “composite application” especially begs the immediate questions of “What is it?” and “What isn’t one?” and also “How should we go about building a 'good' one?”

Of course, “goodness” has implicit within it the concept of “right” and “wrong,” or at the very least “OK” and “better.” So here I will list Roger’s Rules for Web Services, where I explore the boundaries of “good” and “bad” with regard to this new modeling clay of information technology. I will use, as the basis of value, the appropriateness of a design of a Web service with regard to its usability by others in the new age of SOA. The standard of value will be how easy or hard it is for another person to use this Web service in a new breed of tool to build a higher-level solution such as a composite Web service, an entire composite application or a workflow.

In short, how will this Web service will be handled by the new breed of application development tools, and even end-user information usage tools, which are arriving. This is quite analogous to the issues of database design and Codd’s more primitive database structuring advice regarding “normal forms” (first normal form, second normal form and so on, that he defined for databases). By those guidelines, if you didn’t structure your database well, then you would have trouble building solutions on top of it using fourth-generation languages (in the 1980s and 1990s) and object libraries and business intelligence tools (in the 1990s and in this decade).

Put yet another way, when the Macintosh came out, we had menus and windows, but there were lots of ways they could be used. Some would be intuitive to most users; some would not. We needed style guides. Web services and SOA are the same way. So, you can call them rules or a style guide or patterns or just a semi-strict old guy’s opinion, but whatever you call them, here they are.

1. Use WSDL (including definitions for operations and the XML schema needed to define the data structures that are going to go into and out of those operations) to the best of your ability, injecting the highest amount of semantic knowledge that is known about what this piece of business logic is going to do.

If WSDL is not used, or its UDDI entry equivalent, in my opinion it is not really a Web service. It is something else. Maybe it is XML over http or some other document exchange over the Internet mechanism, but a Web service has metadata that describes the operation signatures and the types of data that go in and out of those operation signatures. Tools that expect Web services to have this metadata explaining or self-describing them will not work well when it isn’t there.

2. Use SOAP for your communication. It can be over http, or https or even some other protocol if there is a reason to use a special carrier medium, but use SOAP.

Non-SOAP XML payloads over simpler protocols aren’t a dumb thing to do; they just aren’t what tools expect. The new breed of tools that are coming to market want to send a SOAP message, receive one, analyze one, reroute one and so forth. If you don’t use that part of the standard, in my mind it isn’t a Web service. You won’t get the benefits of the standards work or the benefits of the tools that use those standards and expect them.

3. Provide at least semi-obvious and right-sized semantics in the design of the business logic, whose inputs and outputs are described by the WSDL.

People and tools will look at the structure of your business logic and the inputs and outputs, and they will try to assume a usage pattern for these services. Try to make it as obvious as possible, and try to make the operations the best modeling clay possible.

Also, consider giving the user or tool some hint, perhaps in the name of the Web services operation, whether this is a “query-purposed” operation, or has an “update,” “insert” or “delete” intent in its heart or something
that can’t be described with such simplistic notions. But use good names for the schemas and operations. I am
not sure “comments” are going to be read much in this metadata medium, but names are quite visible with these
XML standards. Operations with names FindCustomer, DeleteOrder or PromoteToPlatinumStatus are much
better than DoIt or PostIn.

Sometimes there are many services that always have to be used in combination when some obvious higher-level
services could have been provided. Sometimes you can only retrieve a “customer with his orders and shipments
of those orders and returns of parts of those shipments” but can’t separately read the customer records or the
order records. Give consideration to the “right-sizing” of the services built. When services must serve multiple
masters, provide good modeling clay so that composition or refactoring tools that might be derived to more
devour specially purposed services from yours can be built.

4 Judiciously, even rarely, use "more definition needed at runtime": data structuring mechanisms
(such as the data type "any" or recursively nested data structures of arbitrary depth or "nested
compositors", and then only when you absolutely have to.

There just isn’t as much information about what to do with this business logic when this “runtime” discovery or
“must know in advance” type of business logic is built. Again, sometimes multiple masters must be served, and
this is the best modeling clay that can be built. But be aware that only the best “modeling up” or “refactoring”
tools will be able to deal with such runtime ambiguities and provide more obvious Web services, on top of these
more “raw” services, which could be used by end-user tools or less-initiated developers.

5 Make authentication as straightforward and obvious as possible.

There is no “one standard” for Web services security in practice right now, but
there are several “patterns” that are working out anyway. There are standards
coming, but even when they get used, our history dooms us to diversity in this
regard. But it works anyway, because tools have been built to make use of services
by automatically authenticating, as long as a fairly obvious pattern for
authentication is used. Some obvious patterns are the “session id style” where there
is a “login” call that returns the session id that is used in all subsequent calls. This
is good for stateful session families of services. There is also the “user name and
password in every call” style, which is also obvious. Windows basic authentication
or “error 404 http basic” authentication is also fairly straightforward. But don’t get
too creative and stuff names and passwords into cute spaces in headers and such.
The more you get inventive here, the harder it is for others to use the services.

6 Supply auxiliary metadata that provides additional semantics that aren’t
yet in the standards via additional Web services calls or via WSDL
extensions, and make them well known. Don’t hide the semantics.

There is the ability to pass a fair amount of semantic metadata within a WSDL file to a tool or human user, but
the WSDL and XML schema standards don’t contain everything that a user would like to know about how to
use a particular Web service. There are tools and repositories coming out that collect more semantic knowledge
than the standard metadata documents currently hold. Sometimes this can be made available to those tools with
calls that “describe” how to use the other calls—for example, the names of operations to call to get the always
changing list of legal values for an element, or alternate “display names” or validation rules for an element.
Perhaps even the desired font size and color or position on a layout page might be important in how to use a
Web service’s inputs and outputs. Whatever the additional metadata that could be offered, try to offer it using
another Web service that lets the user discover information that has to be given to the consumer of the service.
There is no place to pass those semantics in the standard XML metadata documents. Another way to pass such
additional metadata would be to put it in the WSDL extension mechanism. However, putting it in a document
that cannot be found or must be read by a human, or putting this information in an inaccessible database makes
the service, again, hard to use.

7 Be cognizant of the notion of a NULL value. Make these semantics as clear in your WSDL as possible,
and be as consistent as possible across your Web services.

Codd did have a rule about NULLs in his database algebra. He required that they be dealt with. He required that
there be a value that means “nothing is here,” which was distinct from an empty string (for character data) and
distinct from a zero (for numeric data). Web services have the notion of a nil value, which is the same, and a
WSDL can say whether these NULL values are allowed or not for a data element.

So, first of all, don’t cross up your favorite tools by saying they aren’t allowed, and then deliver them in a SOAP
message. I suppose this goes without saying. Moreover, be aware that there are other ways, new ways, to deliver
“nothing” in the SOAP message or a certain element. The code that is sending the SOAP payload can leave an
element out, tags and all, or can put in the tags, but have nothing between them, for example. This is confusing
for tools, to say the least, but a good design that is cognizant of this issue can help.

In my writing on this topic, I will use the term “unavailable” to describe the case where the element just didn’t
show up in the return values from a “query purposed” operation. The receiving end doesn’t know whether it is a
NULL or not; it just didn’t show up. If you execute what appears to be a corresponding “update” purposed
operation with that data, you can’t send a NULL for that value when you didn’t get anything for that element
when it was retrieved, because you might overwrite some data inadvertently.

8 Build applications using metadata-driven application development principles. (Use a repository for
semantic knowledge that has an existence of its own apart from any particular application, and use
metadata to define user interfaces as much as possible.)
Use the standards, and tools that are built to take advantage of the standards, to create the “semantic Web” of your enterprise. Build composite applications to find the data, and transact with it, where it sits. Don’t move data around and make uncontrolled duplicate copies just to get access to it. (Don’t violate Codd’s first normal form rule at the “enterprise silo” level.) Be an agile business that knows the 360-degree view of your customers, and every entity in your enterprise, because you have successfully implemented the above rules.

http://www.sdtimes.com/article/special-20051101-01.html

Copyright © 2005 BZ Media. All rights reserved.