<http://www.soapui.org/The-World-Of-API-Testing/soap-vs-rest-challenges.html>

<http://docs.spring.io/spring-ws/site/reference/html/tutorial.html>

<http://tomee.apache.org/examples-trunk/simple-webservice/README.html>

SOA

How is the service consumer (the part of software that makes use of the service, for example, on the User Interface side) expected to invoke the service and receive the list of requested objects?

- **With platform native calls**: As far as Java is concerned this may be implemented through **Remote Method Invocation** (**RMI**), Sockets, Servlets, or JMS

-**With a distributed object communication middleware**: CORBA or DCOM are just some examples.

**With a text-based communication protocol**: This can be done by sending the request as a text stream and obtaining a textual response containing the data. This is the approach on which web services are based.

The first way is straightforward, but it has some drawbacks. It is tied to a common language (the service and its consumer must share the same technology and language—for example Java, .NET). Furthermore, the exchanged object's classes must be the same version otherwise the communication will not happen.  Distributed object communication middleware have been a successful answer for a considerable time span. CORBA in particular, thanks to its cross-platform nature, has offered an evident asset where interoperability was needed.  The text-based approach implies, on the other side, a process of serialization (conversion from object to a textual form) when the client sends the request, and a process of deserialization (conversion from text to object) when received by the server. A similar double process must happen then for the response flow.  The serialization and deserialization processes seem to add complexity to the communication. But consider the advantage, a complete independence from technologies and loose coupling between parts.

* 1. History of SOA

The term SOA was first coined by Gartner analyst Yefim V. Natis in one of the research papers in 1994. According to Yefim:

*SOA is a software architecture that starts with an interface definition and builds the entire application topology as a topology of interfaces, interface implementations, and interface calls...*

Despite being coined much earlier, SOA started to become a buzzword only in early 2000. With the advent of web services and WSDL compliant business process, SOA started to become popular among technology enthusiasts.

* 1. Objectives of using SOA

**Loose coupling** The business process being decomposed into independent services will help in bringing down the dependencies on a single process. This in turn will help in faster processing time

**Platform-neutrality** ML-based message information flow enhances the capability to achieve platform neutrality. These XML messages are based on agreed XML schema, eliminating the need to set up other messaging standards that can differ across platforms.

**Standards** The message flow across the enterprise is in the form of globally accepted standards. The service only has to depend on the service descriptions without worrying about the target standards and removing the dependencies.

**Reusability** The business logic being divided into smaller logical units, the services can easily be re-used. These enhance the utilization of SOA-based solution, which has a cascading affect on service delivery and execution.

**Scalability** As the business processes are decomposed into smaller units, adding new business logic is easy to accomplish. The new logic could either be added as an extended unit of the current service, or it can also be constructed as a new service.

**XML**

We've all been staring a form of self-describing data in the face every time we use a web browser. HTML is a good example of a standard data format that is quite flexible due to its provision for self-describing elements. For example, the color and font to be applied to a particular section of text are described right along with the text itself. This kind of self- describing data is commonly referred to as a markup language. The content is "marked-up" with instructions for its own presentation. This is very nice, and it obviously has gained an incredible level of industry acceptance. But HTML is not flexible enough to accommodate content that was not anticipated by its designers. That's not a knock on HTML; it's just the truth. HTML is not extensible.

The Extensible Markup Language (XML) is just what we're looking for. XML is a hierarchical, tag-based language much like HTML. The important difference for us is that it is fully extensible. It allows us to describe content that is specific to our own applications in a standard way, without the designers of the language having anticipated that content. For example, XML would allow me to create content to represent the stock quote response message from the previous section

**SOAP**

The **Simple Object Access Protocol** (**SOAP**) is a web service standard communication protocol defined by the W3C. It basically defines the structure of the exchanged message, which is composed of an "envelope" with a "header" and a "body". As you will see next, this protocol adds various levels of complexity.

The protocol is made up of a number of distinct parts. The first is the envelope, used to describe the content of a message and some clues on how to go about processing it. The second part consists of the rules for encoding instances of custom data types. This is one of the most critical parts of SOAP: its extensibility. The last part describes the application of the envelope and the data encoding rules for representing RPC calls and responses, including the use of HTTP as the underlying transport.

But it also offers a wide range of powerful features, among which are:

* Automatic generation of classes involved in the communication process
* Automatic generation of the web service descriptor (WSDL)
* Automatic generation of client classes starting from the service WSDL
* Ability to be used with network protocols other than HTTP (for example, SMTP or JMS)
* Ability to encapsulate authentication mechanisms

Ability to establish a stateful conversation

All SOAP messages are packaged in an XML document called an envelope, which is a structured container that holds one SOAP message. The metaphor is appropriate because you stuff everything you need to perform an operation into an envelope and send it to a recipient, who opens the envelope and reconstructs the original contents so that it can perform the operation you requested. The contents of the SOAP envelope conform to the SOAP specification,1 allowing the sender and the recipient to exchange messages in a language-neutral way: for example, the sender can be written in Python and the recipient can be written in Java or C#. Neither side cares how the other side is implemented because they agree on how to interpret the envelope. In this chapter we'll get inside the SOAP envelope.

The SOAP HTTP request uses the HTTP POST method. Although a SOAP payload could be transported using some other method such as an HTTP GET, the HTTP binding defined in the SOAP specification requires the use of the POST method. The POST also specifies the name of the service being accessed

The Host: header field specifies the address of the server to which we're sending this request, www.mindstrm.com. The next header field, Content-Type:, tells the server that we're sending data using the text/xml media type. All SOAP messages must be sent using text/xml. The content type in the example also specifies that the data is encoded using the UTF-8 character set. The SOAP standard doesn't require any particular encoding. Content-Length: tells the server the character count of the POSTed SOAP XML payload data to follow.

The next one, however, is SOAP specific. The SOAPAction: header field is required for all SOAP request messages transported using HTTP.2 It provides some information to the HTTP server in the form of a URI that indicates the intent of the message.

An RPC-style request message usually results in a corresponding HTTP response. Of course, if the server can't get past the information in the HTTP headers, it can reply with an HTTP error of some kind. But assuming that the headers are processed correctly, the system is expected to respond with a SOAP response. Here's the HTTP response to the RPC-style request from the previous example:

HTTP/1.0 200 OK

Content-Type: text/xml; charset="utf-8"

Content-Length: 359

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

<SOAP-ENV:Body>

<m:GetCurrentTemperatureResponse xmlns:m="WeatherStation">

<m:temperature>26.6</m:temperature>

</m:GetCurrentTemperatureResponse>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

The response's HTTP header fields are, for the most part, similar to those of the request. The response code of 200 in the first line of the header indicates that the server was able to process the SOAP XML payload. The Content-Type: and Content-Length: fields have the same meanings as they did in the request message. No other HTTP header fields are needed; the correlation between the request and response is implied by the fact that the HTTP POST is inherently a request/response mechanism. You send the request and get the response as part of a single transaction.

When sending data over a network, the data must comply with the underlying transmission protocol, and be formatted in such a way that both the sending and receiving parties understand its meaning. This is what we refer to as data encoding. Data encoding encompasses the organization of the data structure, the type of data transferred, and of course the data's value. Just like in Java, it is the data that gets serialized, not the behavior. Data encoding and serialization rules help the parties involved in a SOAP transaction to understand the meaning and content of the message. The model for SOAP encoding is based on XML data encoding, but the encoding constrains or alters those rules to fit the intended purpose of SOAP.

One way to avoid interoperability problems with SOAP-based services is to use a structured language to describe the service, its location, the service methods, parameters, data types, and so on. The Web Services Definition Language (WSDL) does just that. WSDL is an XML grammar for describing web services. Systems can determine the programmatic interface of a web service by looking at the WSDL document associated with that service. The document describes the service methods along with their parameters and return types, and may also include the address, or endpoint, of the service. One of the greatest benefits of WSDL is that it is a single, accepted standard1 for describing web services, which among other things motivates SOAP developers to avoid using their own mechanism for that task

According to the WSDL specification:

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure- oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

A WSDL document is a collection of one or more service definitions. The document contains a root XML element named definitions; this element contains the service definitions. The definitions element can also contain an optional targetNamespace attribute, which specifies the URI associated with the service definitions. WSDL uses namespaces and namespace IDs in the same way we've been using them in SOAP envelopes, so it's common to find a number of namespaces declared at the definitions level of the document.

Goals We Can Achieve with SOA

**Loose Coupling**

**Seamless Integration**

**Reusability**

**Return on Investment (ROI)**