

Essential Guide: When and How to use REST

Where does the software architecture stand among SOAP, APIs?



In this e-guide

■ Section 1: REST vs. SOAP p. 2

■ Section 2: REST and APIs p. 7

■ Section 3: More on REST p.29

■ Further Reading p.41

In this e-guide:

Representational state transfer (REST) is a stateless software architecture that reads webpages containing XML. REST, which some architects view as a simpler alternative to Simple Object Access Protocol (SOAP) and Web Services Description Language Web services, has become a popular Web application program interface (API) model over the years. A RESTful API, or RESTful Web service, uses both HTTP and REST.

This REST guide outlines a range of stories that highlight when it's best to implement REST, especially in comparison with SOAP or an API.

In this e-guide

■ Section 1: REST vs. SOAP p. 2

■ Section 2: REST and APIs p. 7

■ Section 3: More on REST p.29

■ Further Reading p.41

■ Section 1: REST vs. SOAP

When to use REST instead of SOAP

SOA and REST each have favorable qualities and have been compared to each other from the start. Maybe the question shouldn't be whether SOA or REST is best, but rather when they can be combined in order to meet an organization's goals.

➤ **Continue reading**

In this e-guide

■ Section 1: REST vs. SOAP p. 2

■ Section 2: REST and APIs p. 7

■ Section 3: More on REST p.29

■ Further Reading p.41

■ SOA and RESTful interfaces: When, why they should be combined

Tom Nolle, President, CIMI Corporation

Since the dawn of the [service-oriented architecture \(SOA\)](#) evolution, SOA has been compared and contrasted with the Web model of the "RESTful" interface. While a lot has been said about the differences, most of the discussions have been arcane, and enterprises still report they're confused about what makes one model better than the other.

Most application architects realize it's possible to combine [representational state transfer \(REST\)](#) and SOA under at least some situations and gain advantage by doing so, if the applications are suitable and the development practices reflect the goals and benefits of both approaches. The biggest question is whether the goal is to develop a RESTful interface throughout while meeting most SOA goals, or to create a [hybrid of REST and SOA](#).

Breaking down SOA, REST

SOA is an architecture designed to facilitate the development of flexible, agile applications by reusing common components within a general model of workflow management. The components are loosely coupled, meaning components are located via a publish/subscribe registry process, and they are linked using a general object access mechanism (typically the [simple](#)

In this e-guide

■ Section 1: REST vs. SOAP p. 2

■ Section 2: REST and APIs p. 7

■ Section 3: More on REST p.29

■ Further Reading p.41

[object access protocol or SOAP](#)) that uses a definition language ([Web services description language or WSDL](#)) to describe the features and interfaces that link user and provider. The model supports standard mechanisms for identity, security and recovery processes and so is functionally rich in supporting complex business-critical applications.

The [RESTful model](#) was designed for simple user access via a browser. While this [hypertext markup language \(HTML\)](#) look-and-click navigation approach has been expanded to allow for more structured ([extensible markup language or XML](#)) information exchanges between program elements and not just with users, the basic interface is the same; the components are represented as [uniform resource locators \(URLs\)](#) and decoded using Internet-compatible [domain name system \(DNS\)](#) mechanisms.

Component coupling, arguably, is beyond loose -- it's nonexistent except as the users themselves might select among links visually. [RESTful services](#) are easy to develop and deploy, lightweight, inexpensive to host and maintain, and ideal for typical online applications.

The classic approach to creating a REST/SOA symbiosis is to add a Web server front-end to a SOA application, which makes SOA applications directly Internet-ready and lets them be accessed by browser/thin-client tools. This is fine where it's determined that the value of SOA's application-composition flexibility is high, but the growing importance of thin clients, mobility and Web access has prompted some architects to look at applying more RESTful concepts within the applications themselves.

In this e-guide

■ Section 1: REST vs. SOAP p. 2

■ Section 2: REST and APIs p. 7

■ Section 3: More on REST p.29

■ Further Reading p.41

Keys to RESTful SOA

For application architects, the key in creating a kind of RESTful SOA is the definition of the objects that will represent processing elements/resources. In REST, each object is represented by a URL and the user of the object is responsible for including state information in each message so that the object's processing is always stateless. Objects that are too complex, in that they represent multi-stage processes, will be difficult to code in RESTful form and their use will require transformational design patterns or other adapting mechanisms.

The next-biggest issue in RESTful SOA is managing the composability and binding process. One of the biggest objections of enterprises to formal (SOAP-based) SOA is that this level of discovery and binding flexibility isn't useful enough to justify the complexity. They report most applications have relatively static components, and that where components are introduced dynamically, they're typically representing functionality exposed by another application or even another user via an [application program interface \(API\)](#) -- which ironically is often RESTful!

It is possible to provide directory-based component discovery processes for RESTful interfaces if this level of dynamism is needed, but such needs may mean that a hybrid approach that uses REST at the front-end portion of the application and SOA/SOAP at the back end is justified.

In this e-guide

■ Section 1: REST vs. SOAP p. 2

■ Section 2: REST and APIs p. 7

■ Section 3: More on REST p.29

■ Further Reading p.41

Overcoming concerns

Some application architects may have concerns about the security of RESTful interfaces. In a strict sense, the same encryption options are available for both, but in SOA/SOAP, the developer has more explicit control over security. However, RESTful applications can mandate secure connections and the developer can ensure this is done. The issues with auditing whether security has been imposed can be resolved by testing insecure HTTP calls to the interfaces to ensure no connection is permitted in that mode.

Orchestration and workflow threading are also sometimes cited as concerns when considering implementing SOA via RESTful interfaces. Of course, many message-bus protocols will support the use of RESTful interfaces (the specific support will depend on both the message bus and the selected programming language). What may be challenging is ensuring that backout procedures can be implemented if a transaction fails because of a component failure.

No explicit intermediary processes in REST and no specific mechanisms exist to ensure that a given RESTful process, having been successfully completed, can be reversed, much less what information is required to do that reversing. It is relatively easy to build a process map into a RESTful message and pass it from stage to stage so that a failure can be backtracked, but it may be necessary to develop this capability rather than to simply exploit a standard tool or capability.

In this e-guide

Section 1: REST vs. SOAP p. 2

Section 2: REST and APIs p. 7

Section 3: More on REST p.29

Further Reading p.41

The great majority of "services" implemented today are based on [REST and not on SOA/SOAP](#), but enterprises have come to trust the formal SOA architecture for mission-critical applications. In applications where security/identity management, composability and compliance are absolutely critical, it may be difficult to reframe in RESTful terms. But for the average enterprise, the real question is whether the current application components demand SOAP or can be accessed in RESTful form.

Where REST is available, it likely offers a faster path to deployment, easier integration with customers, suppliers, and mobile workers, and more flexible GUI tuning for worker support. REST, as even its proponents will agree, is not a perfect solution, but for many SOA applications, it's the best solution.

 **Next article**

In this e-guide

■ Section 1: REST vs. SOAP p. 2

■ Section 2: REST and APIs p. 7

■ Section 3: More on REST p.29

■ Further Reading p.41

■ REST vs. SOAP: Choosing the best web service

Swati Dhingra, Associate Project Manager, FirstRain Software Centre

"I need to update the local inventory database with the inventory information from multiple suppliers. The suppliers provide a web service-based interface. As the application does not have any server side component (the application is a fat client talking directly to the database), is it possible to consume these web services directly from my application database?"

Do similar questions trouble you? Ever wondered what could be the solution to such an issue?

For a different perspective, the databases can act as the service consumer in contrast to the norm of acting as a service provider. Here is how you can invoke a web service via database-stored procedures -- and in explaining so, I'll address the differences of REST vs. SOAP.

Web services overview

A Web service, in very broad terms, is a method of communication between two applications or electronic devices over the World Wide Web (WWW). Web services are of two kinds: Simple Object Access Protocol (**SOAP**) and Representational State Transfer (**REST**).

In this e-guide

■ Section 1: REST vs. SOAP p. 2

■ Section 2: REST and APIs p. 7

■ Section 3: More on REST p.29

■ Further Reading p.41

SOAP defines a standard communication protocol (set of rules) specification for [XML](#)-based message exchange. SOAP uses different transport protocols, such as [HTTP](#) and [SMTP](#). The standard protocol HTTP makes it easier for SOAP model to tunnel across [firewalls](#) and [proxies](#) without any modifications to the SOAP protocol. SOAP can sometimes be slower than middleware technologies like [CORBA](#) or [ICE](#) due to its verbose XML format.

REST describes a set of architectural principles by which data can be transmitted over a standardized interface (such as HTTP). REST does not contain an additional messaging layer and focuses on design rules for creating stateless services. A client can access the resource using the unique [URI](#) and a representation of the resource is returned. With each new resource representation, the client is said to transfer state. While accessing RESTful resources with HTTP protocol, the URL of the resource serves as the resource identifier and GET, PUT, DELETE, POST and HEAD are the standard HTTP operations to be performed on that resource.

REST vs. SOAP

There are significant differences between SOAP and RESTful web services. The bullets below break down the features of each web service based on personal experience.

In this e-guide

■ Section 1: REST vs. SOAP p. 2

■ Section 2: REST and APIs p. 7

■ Section 3: More on REST p.29

■ Further Reading p.41

REST

- RESTful web services are stateless. You can test this condition by restarting the server and checking if interactions survive.
- For most servers, RESTful web services provide a good caching infrastructure over an HTTP GET method. This can improve the performance if the information the service returns is not altered frequently and is not dynamic.
- Service producers and consumers must understand the context and content being passed along as there is no standard set of rules to describe the REST web services interface.
- REST is useful for restricted-profile devices, [such as mobile](#), for which the overhead of additional parameters are less (e.g., headers).
- REST services are easy to integrate with existing websites and are exposed with XML so the HTML pages can consume the same with ease. There is little need to refactor the existing site architecture. As such, developers are more productive because they don't need to rewrite everything from scratch; instead, they just need to add on the existing functionality.
- A REST-based implementation is simple compared to SOAP.

SOAP

- The Web Services Description Language ([WSDL](#)) describes a common set of rules to define the messages, bindings, operations and location of the service. WSDL is akin to a contract to define the interface that the service offers.

In this e-guide

■ Section 1: REST vs. SOAP p. 2

■ Section 2: REST and APIs p. 7

■ Section 3: More on REST p.29

■ Further Reading p.41

- SOAP requires less plumbing code than REST services design (e.g., transactions, security, coordination, addressing and trust). Most real-world applications are not simple and support complex operations, which require conversational state and contextual information to be maintained. With the [SOAP approach](#), developers don't need to write plumbing code into the application layer.
- SOAP web services, such as JAX-WS, are useful for asynchronous processing and invocation.
- SOAP supports several protocols and technologies, including WSDL, XSDs and WS-Addressing.

Consuming a web service via a database stored procedure allows users to straight away update a database with information from different sources. Users can also schedule a job at regular intervals to get data updated periodically in the database.

REST or SOAP: Which is best for me?

Proponents on both sides of the REST vs. SOAP discussion can be fervent in their advocacy for their web service architecture of choice. Both SOAP and RESTful architectures have proven themselves to be reliable, successful and capable of scaling infinitely, so the decision to use REST or SOAP has less to do with their efficacy and more to do with how either approach fits in with an organization's software development culture and project needs.

In this e-guide

■ Section 1: REST vs. SOAP p. 2

■ Section 2: REST and APIs p. 7

■ Section 3: More on REST p.29

■ Further Reading p.41

Both SOAP web services and RESTful web services have proven their ability to meet the demands of the largest enterprise organizations in the world, while at the same time being able to service the smallest internet of things devices or embedded applications in production.

When choosing between REST and SOAP, two of the key topics to factor into the decision are:

1. The types of clients that will be supported. For example, REST services offer an effective way for interacting with lightweight clients, such as smartphones.
2. How varying degrees of flexibility and standardization will be either rebuffed or accepted by the organization's corporate culture.

Keeping these factors in mind will go a long way in helping organizations to choose between SOAP and RESTful web services.

Common problems faced when invoking Web services and solutions

Sometimes, even after doing everything as expected in the stored procedure to call the Web service, the procedure doesn't get compiled. The following is a compilation of runtime errors faced during stored procedure execution to invoke a Web service and their solutions.

In this e-guide

Section 1: REST vs. SOAP p. 2

Section 2: REST and APIs p. 7

Section 3: More on REST p.29

Further Reading p.41

Problem 1: Getting "ORA-25293 : HTTP request failed" error during procedure compilation.

Solution: Follow the steps below to rectify this.

- Log in through sys user as sysdba.
- View the privileges to the selected schema to use the utl_HTTP package by using the command as follows:
 - `select grantee, table_name, privilege from dba_tab_privs where table_name = 'UTL_HTTP';`
- The grant will be provided to all the public users by default.
- Revoke execute grant on utl_http from public and provide it explicitly to the specific schema from which the Web service needs to be invoked.
 - `revoke execute on utl_http from public;`
 - `grant execute on utl_http to BTFT2;`
 - `select grantee, table_name, privilege from dba_tab_privs where table_name = 'UTL_HTTP';`

Problem 2: If the stored procedure calling the Web service gives "Network access denied" during the call of Web service.

Solution: Add the Web service url to the access control list by following the steps below.

- Login through sys as sysdba.
- Execute the following procedure to create ACL.

In this e-guide

Section 1: REST vs. SOAP p. 2

Section 2: REST and APIs p. 7

Section 3: More on REST p.29

Further Reading p.41

- **BEGIN**
DBMS_NETWORK_ACL_ADMIN.CREATE_ACL (
 acl => '<name of the acl file>.xml',
 description => 'Permissions to access the web service url',
 principal => '<Schema name>',
 is_grant => TRUE,
 privilege => 'connect');
COMMIT;
END;
 /
- Create a role and then grant connect to this role on the ACL by using the steps below:
 - **create role role1;**
 - **BEGIN** **DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE** (
 acl => '<name of the acl file>.xml',
 principal => 'role1',
 is_grant => TRUE,
 privilege => 'connect',
 position => null);
COMMIT;
END;
- Assign the host names to the ACL to open all the related links in the host.
 - **BEGIN**
DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL (
 acl => '<name of the acl file>.xml',
 host => '.*<host name of the webservice url>');
COMMIT;

In this e-guide

Section 1: REST vs. SOAP p. 2

Section 2: REST and APIs p. 7

Section 3: More on REST p.29

Further Reading p.41

Or

- ```
END;
/
```
- **BEGIN**  
**DBMS\_NETWORK\_ACL\_ADMIN.ASSIGN\_ACL** (  
acl => '<name of the acl file>.xml',  
host => '<ip>');  
**COMMIT;**  
**END;**
  - Confirm whether the domain has been added in the ACL using ACL\_UTILITY package.
    - **SELECT \* FROM**  
**TABLE(DBMS\_NETWORK\_ACL\_UTILITY.DOMAINS('www.ajax.googleapis.com'));**
    - **select**acl , **host** , lower\_port , upper\_port **from**  
DBA\_NETWORK\_ACLS;
    - **select**acl , principal , **privilege** , is\_grant **from**  
BA\_NETWORK\_ACL\_PRIVILEGES;

➤ Next article

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

---

## ■ REST or SOAP: Which offers the most benefits for mobile applications

**Todd Biske**, Contributor, SearchMicroservices

If you've ever read or been involved in a discussion with a true "RESTafarian," you've probably heard that [representational state transfer](#) (REST) is the architecture of the web, and there's certainly a lot of truth to that.

There is no arguing with it: REST is a very good fit for [browser-based interactions](#). Browsers and the engines behind them are built to traverse the links in the content they process, as well as providing an array of handlers for the various response types that are possible. The dominant programming language for browsers is [JavaScript](#) and, let's face it, trying to program a [simple object access protocol](#) (SOAP) client in JavaScript is not something anyone wants to do. We've even moved away from using [extensible markup language](#) (XML) and toward [Javascript object notation](#) (JSON).

Why am I talking about web browsers when the question is about [mobile applications](#)? The reason I bring up the web browser first is two-fold.

First, there are really two major models for mobile applications: native and wrapped mobile-web. [Native applications](#) are built from the ground up using the languages, native libraries and toolkits of the mobile OS platform involved. For example, for iOS devices, it's Objective C and the libraries and

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

toolkits provided by Apple for iOS. The presentation logic is installed on the device as part of the application installation.

[Wrapped mobile web applications](#) merely use a native component that wraps a web browser but also provides a bridge between the device capabilities and the web browser, as when interacting with the photo library of the device. The bulk of the presentation logic isn't installed with the application installation, but downloaded via the web when the application is run. When this style is used, we're back to a typical web approach using JavaScript, and SOAP simply isn't an option.

But what about native applications? Why couldn't SOAP be used? There are third party libraries for both Android and iOS that make calling SOAP Web services easier, but out of the box there isn't native support for it. This isn't terribly surprising for iOS as Objective-C has its roots in desktop programming for the Mac, and SOAP has its roots in enterprise integration. It's a bit more surprising for Android, given that Java was the first supported language for application development.

What this really comes down to is a matter of ubiquity and simplicity. First, while native platforms don't support SOAP out of the box, they do support HTTP and XML, so the ubiquity exists. On the simplicity side of things, XML is certainly better than SOAP and, in the case of wrapped mobile web applications, JavaScript is better than XML.

REST isn't just about JSON or XML though, but any of the media types that the browser or platform can natively handle. If I can get back a byte stream and just hand it off the capabilities of the browser or a native presentation

## In this e-guide

Section 1: REST vs. SOAP p. 2

Section 2: REST and APIs p. 7

Section 3: More on REST p.29

Further Reading p.41

component, rather than having to tunnel that data through something like a SOAP envelope, things are not only simpler, but they're also going to be less processor intensive, which means better battery life for the end user.

## Next section

## In this e-guide

Section 1: REST vs. SOAP p. 2

Section 2: REST and APIs p. 7

Section 3: More on REST p.29

Further Reading p.41

## Section 2: REST and APIs

### The relationship between REST and APIs

Mobile and cloud applications, social networking websites, and automated business processes are among the drivers fueling the need for RESTful APIs. Application architects can find themselves with a good headache if they don't completely understand the technology.

 **Continue reading**

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

---

---

## ■ Software developers demanding resource based, RESTful APIs

Jason Tee, Contributor, TheServerSide.com

Well-seasoned enterprise architects know that there is no silver bullet when it comes to application integration, and no single middleware solution will ever [meet every integration requirement criteria](#) to the tee, but when it comes to modularization and [web service development](#), a REST based approach is becoming more and more popular. REST may not be the best approach for every situation, but it does have enough virtues to put it on the short list for any organizations that is comparing the various different approaches to developing web service, creating OSGi components and deploying micro-services. And the fact is, [a REST based approach](#) can be effective in even the most challenging corner-cases, where other options like Java messaging or SOAP based web services run out of steam.

---

## The effective application of RESTful techniques

One of the key strengths of REST is its simplicity, but that simplicity sometimes raises objections from developers who question whether such a simple approach to application integration can deal with the complex challenges enterprises must deal with on a daily basis. For example, software architects often question the veracity of REST when it comes to handling complex, [real world, transactional systems](#) that involve coordinating

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

multiple resources whose interactions need to be synchronized. Jerome Louvel (@jlouvel), coauthor of [Restlet in Action](#) and the creator of the [Restlet framework](#), addresses the topic of transactional, e-commerce systems by framing it as a simple use case with a straightforward solution. For example, to make a traditionally developed online shopping application into a RESTful experience, an architect simply needs to break down the associated business transactions into all of their underlying components.

In a transactional, online e-commerce type of system, people need to be able to browse products, select products, view and edit their cart, enter payment and shipping information and then make a final decision to pay. At any point up to the point where the user completes their checkout, users need to be able to navigate back to a previous step. From a [pure API standpoint](#), this means mapping all the important parts of the transaction into resources via URIs. At the end, the transaction is completed by sending a final HTTP request that coordinates the state of each of the resources involved in the transaction. “It’s just viewing your transaction as a set of one or multiple resources depending on the complexity of your business processes. Technically, nothing prevents you from doing that. It’s just not trying to map what you’re [used to doing with RPC](#) or putting too much context in one request,” says Louvel.

Of course, a [RESTful approach](#) to software design is only one piece of the puzzle when it comes to creating integration middleware that will tie together back-end services with mobile applications, browser based web sites and even embedded systems. Creating application integration middleware also means creating modular and reusable components, with the most proven approach for achieving these goals being the creation of micro-



---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

---

services using OSGi. Unfortunately, the manner in which JAX-RS, REST based applications connect with OSGi based components hasn't always been clear. "People always struggle to connect OSGi concepts with JAX-RS concepts," said [Holger Staudacher](#), a software developer and consultant at EclipseSource. However, the integration path between RESTful web services and modular, OSGi components has become increasingly more clear through projects such the [OSGi JSX-RS connector](#). These software tools help fill in the gaps, simplify the development of REST based micro-services, and make modularity easier to achieve by integrating OSGi with REST-based, JAX-RS frameworks such as Jersey and Wink. "We provide the glue" said Staudacher about the various software projects that help simplify the development of application that use REST and OSGi. Needless to say, using a REST based approach for [developing a distributed architecture](#) effectively becomes a whole lot easier when there is a proven and effective structure within which applications can be built.





---

## Easing into a resource based approach to APIs

A RESTful approach to software development can greatly simplify the task of application integration, but this new approach can often cause troubles for those with more of a legacy background in [distributed computing](#). That's when developers find themselves creating more problems than they fix. Louvel's advice is not to take the usual, traditional approach when developing RESTful APIs. Forget what you learned from the service-oriented, [RPC based world](#). Then, you'll discover that you are working without limitations and you can really start to solve your business problems.

////////////////////////////////////

**In this e-guide**

- 
[Section 1: REST vs. SOAP](#) p. 2
- 
[Section 2: REST and APIs](#) p. 7
- 
[Section 3: More on REST](#) p.29
- 
[Further Reading](#) p.41

Furthermore, the modularity involved in developing a RESTful architecture means developers have smaller, more agile parts that provide greater flexibility when designing solutions that really work.

The main takeaway Louvel has for Java developers is this: apply as many principles from the tenants of RESTful development and design as possible, but at the same time, use only as much as is needed. See everything as a resource. Use calls from the HTTP protocol, especially GETs, PUTs, POSTs and DELETES to effectively create and update your resources. Eventually, approaching the problem domain in a RESTful manner becomes second hand. By applying these foundational concepts, developers get a lot of built-in features that are common to the world-wide-web such as caching and automatic compression for free. And once the development team has got the basics down, kick it up a notch and try to see every problem as having a RESTful solution.

////////////////////////////////////

 **Next article**

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

---

## ■ Effective Naming Strategies for RESTful Web Services

**Lukas Stewart**, Contributor, TheServerSide.com

One of the major shortcomings of RESTful web service design is the historic lack of any strong technology that can be used to effectively describe the service contract interface between the server and the client. RESTful APIs are flexible, sensible and provide for a great deal of loose coupling, but if an application architect is looking for an effective way to describe the RESTful API in a manner similar to the way WSDL describes traditional SOAP based web services, well the architect is sadly out of luck.

Fortunately, the description languages are catching up with the popularity of REST, and WSDL 2.0 and WADL are two technologies that are filling the RESTful need. Of course, these specifications are not perfect, as their ability to describe JSON strings and complex attributes is still lacking, but nevertheless, progress is being made.

One caveat to using these technologies, especially if software engineers are hoping to use code generators to create WADL and WSDL files is the fact that resource elements in a RESTful URL need to be separated with constants. It's actually a good idea in general, as it makes URLs more meaningful and easier to read. But new standards are making the use of interjected constants less about what's good, and more about what's required.

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

When you see an example, it's fairly easy to comprehend. Take a look at the following URL that might be used for a product rating service:

<http://www.example.com/services/ratings/plunger/5>

In this example, you can see quite clearly that a plunger is being given a five out of five product rating. But the URL is not separated with constants, which causes problems with the description languages. Fortunately, solving the problem is easy, as you can see from the following, corrected URL:

<http://www.example.com/services/ratings/product/plunger/rating/5>

The words product and rating are constants in the URL, and assign meaning to the corresponding resource values. As you can see, it's not a big change in terms of URL design, but it is significant, as it will make your RESTful APIs more compatible with the various description languages on the market today, and in turn, clients will have fewer problems when interacting with them.

---

➤ **Next article**

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

---

## ■ Gartner analyst: REST APIs gain added importance in application integration design

**Jack Vaughan**, Senior News Writer, SearchDataManagement

Major software changes underway today revolve around the [design of the Application Programming Interface](#) (API), according to Daniel Sholler, Gartner vice president and analyst specializing in application architecture, integration and development issues. The nature of these "integration interfaces" is subtly changing.

"What we are seeing is that [the design of APIs](#) has become as important as the design of the user interface," Sholler told an audience at last month's Gartner AADI conference in Las Vegas, Nev. The APIs, he said, increasingly support [REST interfaces](#), which call for a more general [approach to design](#).

REST, for REpresentational State Transfer, is a simple stateless architecture generally running over HTTP. The Web itself is often cited as the exemplary [REST system](#). [Among big REST drivers](#) today are mobile applications, social networking Web sites, mashup tools and automated business processes.

Sholler uses the term WOA, for Web-oriented architecture, to describe the application of [REST principles](#) to the design of Web services. Because it is more general, [the REST style](#) shows "a bit more tolerance to changes,"

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

according to Sholler. That is particularly good with third-party [cloud and mobile applications](#) that are frequently altered with little forewarning.

The REST architecture's generality allows it to work in a wide variety of situations, and it has proved popular with a wide set of programmers. Sholler pointed to simplified REST identifiers as illustrations – he cited examples from a BestBuy application as well as Facebook and Amazon APIs - of the changes REST as brought to middleware and API development.

The forward march of the Web – and HTTP – has had a hand in [driving REST forward](#) as well. It was a different scenario, said Sholler, in the early day of Web services. Around that time, SOA built with the main goal of connecting existing application portfolios, often using SOAP (for Simple Object Access Protocol). Those traditional enterprise portfolios have now been joined by more Web and mobile application types.

"The style of SOAP was meant for the classic middleware design," said Sholler. That is different, he suggested, than the Web, where redundancy to failed communications is more or less built into the system.

---

## WOA criteria

"The design criteria for WOA use cases are to build something that is absolutely application neutral," he said. "You want to design it so it is as general as possible ... [to] make it only as specific as you have to."

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

////////////////////////////////////

"You have to think about going from the general to the specific, not from the specific to the general," he said.

Sholler cautioned that it was important for software architects to understand where WOA design works best. Among several apt REST cases he cited: instances of unknown or inconsistent infrastructure capabilities, limited out-of-band communications and cases where there are large numbers of counterparties.

"REST comes up where you don't have control, when there are open-ended use cases, where you want to collaborate with users [and] where you need to be tolerant of change that comes along. Where we see 'REST' most is in cloud, mobile and [areas where you need] access to information."

////////////////////////////////////

➤ **Next section**



## In this e-guide

Section 1: REST vs. SOAP p. 2

Section 2: REST and APIs p. 7

Section 3: More on REST p.29

Further Reading p.41

## Section 3: More on REST

### More on REST uses

Everything from enterprise service buses (ESBs) to health care can have a part in RESTful integration. Here are some other stories involving the platform.

[Continue reading](#)

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

---

---

## ■ RESTful services take on a role in health IT infrastructure

**Stephanie Mann and Jack Vaughan**, Editor and Senior News Writer, TechTarget

In recent years, there has been a federal push for a national health IT infrastructure, but a nationwide standard for health information exchange has remained elusive, hindering efforts. SearchSOA.com recently spoke with Gerald Beuchelt, project software systems engineer at The MITRE Corporation, about how [RESTful services](#) might play a role in creating a nationally-recognized health information exchange, charting the way for a REST architecture in health IT.

In particular, Beuchelt gave insight into [hData](#), a REST-based standards framework for health information exchange ([HIE](#)) that has already gone through its first round of standardization. The hData project began back in 2009 when MITRE, a federally funded research and development nonprofit, conceived of the idea in response to government needs. According to Beuchelt, HIE standards at the time were complex, and the health IT industry recognized it had a problem. From the beginning, hData reflected a departure from the way HIE had been approached in the past, Beuchelt said, noting:

We got to the problem and consciously said, 'Let's start with a clean slate, and [think about] how we would really rebuild all this, having learned a lot of things from our adventures with [SOAP](#) and SOA.' One of the things we came

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

up with is that we probably want to use a RESTful approach instead of a fairly XML-centric, SOAP approach.

A RESTful approach would allow hData to be coding-neutral in many ways, while still supporting [Javascript Object Notation](#), native files like PDFs and other common medical imaging formats, Beuchelt explained. This new RESTful service "exchange paradigm" was an alternative to the message and document exchange paradigms that already existed for HIE.

According to Beuchelt, health IT has gone through a number of changes with regards to the exchange paradigm over the past few decades:

In the '80s, it was very much message oriented. But such message architecture is not necessarily the most effective for [interoperability](#), and it also does not necessarily address all use cases that you would like to address. [Then,] one of the things that came up in the late '90s and early 2000s was the creation of the CDA -- [clinical document architecture](#). It is effectively an XML framework for writing complex medical notes. ... But one of the issues that the industry was facing with the CDA specification was that it was a very, very complex XML. ... A service exchange paradigm allows you to create a much more agile way of passing things around.

The potential of service-oriented architecture ([SOA as a paradigm](#)) for exchanging health information wasn't recognized until the mid-2000s, he said. Since then, groups like [Health Layer 7 \(HL7\)](#) and the Object Management Group have been working closely together to create services that deal with specific health IT needs. For example, Beuchelt is co-chair of

## In this e-guide

Section 1: REST vs. SOAP p. 2

Section 2: REST and APIs p. 7

Section 3: More on REST p.29

Further Reading p.41

the SOA Working Group, an HL7 group that works to enable HL7 standards to be fully usable within systems that are designed using SOA principles.

The hData work follows recent trends supporting REST. Beuchelt explained:

We saw that a lot of RESTful developers were very successful in terms of creating effective exchange solutions -- not necessarily based on formal languages like [WSDL](#) or WADL [Web Application Description Language], but by using "code by example" and good documentation of the exchange semantics and syntax that are required for a RESTful exchange.

REST is sometimes something of a Wild West, at least compared to more highly standardized approaches like WSDL. But Beuchelt and his colleagues are intent on bringing a well-described hData REST methodology. "That is the reason why we actually took hData and went forward to some standards organizations to effectively write a profile of how we want to use a [RESTful architecture](#) for health-IT exchange," he said.

➤ **Next article**

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

---

---

## ■ What role does an ESB play in RESTful integration?

**Todd Biske**, Contributor, SearchMicroservices

While there may be some that compare REST with XML, they really aren't competing approaches. In reality, you can probably group most approaches these days into three categories:

1. SOAP-based Web Services
2. SOAP-less Web Services implemented using XML/HTTP, with or without a formal schema
3. True REST-style services utilizing the HTTP methods, MIME types and HATEOAS (Hypermedia as the engine of application state)

From what I've seen, there are more and more services being placed into category #2. What makes things confusing is that this category is almost always referred to as [RESTful services](#), when in reality, they aren't. They're just XML-based services sent using the POST HTTP operation, without the SOAP envelope and WSDL. These services are more easily consumed by a browser-based consumer using JavaScript than a SOAP-based service. These services may also be exposed using JSON/HTTP, rather than XML, for even easier consumption by browser-based consumers; however, they still tend to rely on the POST operation for most, if not all, functions. The use of SOAP-based Web Services is still quite common in the enterprise. Many packaged products expose their services using SOAP-

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

based services with WSDL for the excellent tool integration that a well-defined interface provides, but it is increasingly common to see these services also exposed as XML/HTTP, or even true REST-style services.

In my experience, category 3 is still the least common approach, at least in the enterprise, but with use trending upwards.

---

## The role of the ESB





So how do these trends impact the role of the ESB? If anything, the increased emphasis on HTTP actually encourages more use of an intermediary-based approach. Think of how many intermediaries already exist for the HTTP space. You can have caching appliances, load balancers, web security products, and more. The real question is whether you still need an ESB, or you can get by with the existing HTTP-based intermediaries that already exist in most enterprises.

The challenge is that the closer you get to option 3, the less likely that you will have a formal specification of the messages involved. Option 1 clearly has WSDL files, option 2 may still have XML schemas, but option 3 may nothing more than developer documentation on what to expect in response to a GET request, or what HTTP parameters will be accepted on a PUT or POST. This doesn't prevent the use of an intermediary such as an ESB, but it may require a lot more work to get things done.

The most likely scenario is still to rely on an ESB for bridging the gap from systems that fall into categories 1 and 2 (SOAP/HTTP, XML/HTTP) to



**In this e-guide**

- 
[Section 1: REST vs. SOAP](#) p. 2
- 
[Section 2: REST and APIs](#) p. 7
- 
[Section 3: More on REST](#) p.29
- 
[Further Reading](#) p.41

consumers that want something from category 3. For example, there's no reason that you can't take a SOAP Web Service that has a getOrder() operation, and then use an ESB to expose that as a REST service accessed via a GET request to http://rest-services/order?id=xxxxx. This puts the ESB squarely into the integration space where you want to expose REST services from a system that is incapable of doing it, rather than as a general purpose intermediary through which all traffic flows.

So, in short, there's certainly a place for intermediaries in any of the three approaches that I describe. One only has to look at the use of an HTTP load balancer in virtually any Web-based system to understand that. As for ESBs, their role is likely to be relegated to mediating between the REST-style interfaces desired and the interfaces exposed by the underlying systems involved. If those systems can't be modified, you're going to need something in the middle to bridge the gap, and an ESB might just fit the bill.



 **Next article**



---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

---

## ■ Thinking like a developer for better REST interfaces

**George Lawton**, Contributor, SearchMicroservices

Organizations need to think about a developer's needs in the development or [deployment of RESTful services](#). Some of the best practices for this process include keeping the REST interfaces simple and consistent as well as providing good error feedback when exceptions occur. In addition, organizations also need to think about the infrastructure required to maintain service level agreements (SLA) after the application program interface (API) has been deployed.

Chris Haddad, vice president of Technology Evangelism at WSO2, said that applying governance to [RESTful APIs](#) is important for business processes that depend on them. A managed API needs to be available with SLAs in a manner that is secure, authenticated, authorized and protected. There are many challenges [facing good RESTful API management](#). Potential consumers may not trust API stability, reliability or performance. There are also security risks that can prevent publishing and offering open access.

There are two levels of governance with [respect to RESTful APIs](#). Design time governance looks at how to create a resource that is stateless, has a resource oriented URL convention and which supports security. Operational governance looks at the average and peak throughput needed for the API in order to ensure a guaranteed SLA. This form of governance looks at the

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

---

consuming clients, and the hardware and software configurations required to support their needs. It also necessitates a system for monitoring the APIs with the ability to alert managers via SMS or email when a fault is detected.

Haddad recommends sticking with the methods provided by the HTTP protocol including GET, POST, PUT and DELETE for RESTful services. Organizations need to ensure that these are not implemented with side effects that change the state of the system in unexpected ways. For example, PUT should update the resource in place. The GET, HEAD, OPTIONS and TRACE methods should not modify anything.

Haddad also recommends using State Chart (SCXML), which provides a state machine notation for API control that can be incorporated into an approval model. A variety of tools support SCXML including IBM Rational Software Architect, SCXMLgui and in the future WSO2 Carbon.

---

## Keeping it simple

In a recent Webcast, Brian Mulloy, VP of Products at Apigee, said a good place to start is with a virtualization layer on top of the API, which insulates developers from any of the logic underneath. This allows organizations to fine tune the logic underneath the API, without burdening the developers with these changes. He noted that some APIs, such as those for Salesforce.com have up to 20 versions, which can confuse developers.

He also recommends keeping the version names simple, and to refer to them with a “v” followed by an integer. Incorporating a decimal point into the

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

version number can cause the developer anxiety at the implication that it could be constantly updated.

While it is easy to get lost in the [abstract principals of REST design](#), it is more important to simplify life for developers. This includes reducing the number of URIs required to access services. Web applications can reduce the number resources by making extensive use of the POST, GET, PUT and DELETE commands.

As a matter of convention, Mulloy also recommends that resources be specified as nouns because these better match a developer's expectations and thought flow. However, verbs are useful when submitting requests for an algorithmic response, such as currency conversion.

Mulloy also takes a stand for more concrete naming of resources. For example, he points out that the BBC uses the high level abstraction of “item” to make requests across all of its different resources. But it would be more compelling for a developer to see resources for blogs, news, reports and video clips, because it would make it more visceral and easier to program with.

Effective error reporting for REST interfaces is an important aspect in helping developers write working code and in finding underlying bugs in the Web application. “The errors are one of the key pieces of visibility into the API,” said Mulloy. Not only will the developer access the API during the development process, it will also help them find problems after the program has been deployed in the wild.

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

Mulloy recommends that the API translate error messages into standard HTTP error codes. For example, Facebook returns an arbitrary number with no additional context which makes life confusing. In contrast, Twilio plugs the codes into HTTP status codes, and provides a more granular error message.

This upfront work reduces the amount of time a developer spends interpreting codes. In the cases where more complicated codes are generated, Twilio also provides a URL for the message back to its developer community site. This helps provide a forum for developers to identify and correct programming errors more quickly.

There are a wide variety of authentication methods for RESTful services. While some websites, like PayPal have developed their own proprietary method, Mulloy recommends that companies adopt OAUTH 2.0, which simplifies lives for developers in accessing restricted services.

---

## Simplify resources with better messaging

While the common practice in designing RESTful APIs has been to focus on more comprehensive URIs, Mike Amundsen, Layer 7's principal API architect argues that it makes more sense to focus on better messaging rather than direct protocol methods. He wrote, "The key to creating a powerful API, it turns out, is in the design of the messages sent back and forth between parties. Reliable and evolvable API design is based not on function calls and shared objects but on hypermedia-style messages."

## In this e-guide

Section 1: REST vs. SOAP p. 2

Section 2: REST and APIs p. 7

Section 3: More on REST p.29

Further Reading p.41

To address this need, Amundsen has been working on a framework for understanding hyper media types, using a concept called H-Factors. These can be used to create an API with the flexibility, usability and longevity of Web pages. In this case, the URIs can be kept simple, and new features can be added by extending the types of messages supported.

[Next article](#)

---

## In this e-guide

---

■ Section 1: REST vs. SOAP p. 2

---

■ Section 2: REST and APIs p. 7

---

■ Section 3: More on REST p.29

---

■ Further Reading p.41

---

## ■ About SearchMicroservices

Over 4 million programmers, architects, IT managers, and developers turn to our site for industry news, expert advice and peer-to-peer learning opportunities around managing microservices and service-oriented architecture (SOA), application modernization and digital transformation, Business Process Management (BPM), application integration and APIs, software containers and microservices design and development, DevOps, and more.

Whether you're looking to solve a specific application architecture problem or just trying to stay on top of recent industry developments, our site is your online portal for in-depth and relevant information.

---

**For further reading, visit us at**  
**<http://SearchMicroservices.com/>**

Images; Fotolia

©2017 TechTarget. No part of this publication may be transmitted or reproduced in any form or by any means without written permission from the publisher.