When to use SOAP and REST

If your consumers are more likely to be RIAs or Ajax clients, you will probably want something simpler than SOAP, and more native to the client (notably JSON).

Limited bandwidth and resources; remember the return structure is really in any format (developer defined). Plus, any browser can be used because the REST approach uses the standard GET, PUT, POST, and DELETE verbs. Again, remember that REST can also use the XMLHttpRequest object that most modern browsers support today, which adds an extra bonus of AJAX.

Totally stateless operations; if an operation needs to be continued, then REST is not the best approach and SOAP may fit it better. However, if you need stateless CRUD (Create, Read, Update, and Delete) operations, then REST is it.

Caching situations; if the information can be cached because of the totally stateless operation of the REST approach, this is perfect. That covers a lot of solutions in the above three.

So why would I even consider SOAP? Again, SOAP is fairly mature and well-defined and does come with a complete specification. The REST approach is just that, an approach and is wide open for development, so if you have the following then SOAP is a great solution:

Asynchronous processing and invocation; if your application needs a guaranteed level of reliability and security then SOAP 1.2 offers additional standards to ensure this type of operation. Things like WSRM – WS-Reliable Messaging.

Formal contracts; if both sides (provider and consumer) have to agree on the exchange format then SOAP 1.2 gives the rigid specifications for this type of interaction.

Stateful operations; if the application needs contextual information and conversational state management then SOAP 1.2 has the additional specification in the WS* structure to support those things (Security, Transactions, Coordination, etc). Comparatively, the REST approach would make the developers build this custom plumbing.

SOAP: A better choice for integration between legacy/critical systems and a web/web-service system, on the foundation layer, where WS-* make sense (security, policy, etc.).

RESTful: A better choice for integration between websites, with public API, on the TOP of layer (VIEW, ie, javascripts taking calls to URIs).

Parallel heavy processing (see "Context/Foundations" section below): with bigger and/or slower processes, no matter with a bit more complexity of SOAP, reliability and stability are the best investments.

Need more security: when more than HTTPS is required, and you really need additional features for protection, SOAP is a better choice (see @Bell, 32 votes). "Sending the message along a path more complicated than request/response or over a transport that does not involve HTTP", S. Seely. XML is a core issue, offering standards for XML Encryption, XML Signature, and XML Canonicalization, and, only with SOAP you can to embed these mechanisms into a message by a well-accepted standard as WS-Security.

It is important to note that one of the advantages of SOAP is the use of the “generic” transport. While REST today uses HTTP/HTTPS, SOAP can use almost any transport to send the request, using everything from the afore mentioned to SMTP (Simple Mail Transfer Protocol) and even JMS (Java Messaging Service). However, one perceived disadvantage is the use of XML because of the verboseness of it and the time it takes to parse.

areas that REST works really well for are:

Limited bandwidth and resources; remember the return structure is really in any format (developer defined). Plus, any browser can be used because the REST approach uses the standard GET, PUT, POST, and DELETE verbs. Again, remember that REST can also use the XMLHttpRequest object that most modern browsers support today, which adds an extra bonus of AJAX.
Totally stateless operations; if an operation needs to be continued, then REST is not the best approach and SOAP may fit it better. However, if you need stateless CRUD (Create, Read, Update, and Delete) operations, then REST is it.

Caching situations; if the information can be cached because of the totally stateless operation of the REST approach, this is perfect.

That covers a lot of solutions in the above three. So why would I even consider SOAP? Again, SOAP is fairly mature and well-defined and does come with a complete specification. The REST approach is just that, an approach and is wide open for development, so if you have the following then SOAP is a great solution:

Asynchronous processing and invocation; if your application needs a guaranteed level of reliability and security then SOAP 1.2 offers additional standards to ensure this type of operation. Things like WSRM – WS-Reliable Messaging.
Formal contracts; if both sides (provider and consumer) have to agree on the exchange format then SOAP 1.2 gives the rigid specifications for this type of interaction.
Stateful operations; if the application needs contextual information and conversational state management then SOAP 1.2 has the additional specification in the WS* structure to support those things (Security, Transactions, Coordination, etc). Comparatively, the REST approach would make the developers build this custom plumbing.

REST is almost always going to be faster. The main advantage of SOAP is that it provides a mechanism for services to describe themselves to clients, and to advertise their existence.

REST is much more lightweight and can be implemented using almost any tool, leading to lower bandwidth and shorter learning curve. However, the clients have to know what to send and what to expect.

In general, When you're publishing an API to the outside world that is either complex or likely to change, SOAP will be more useful. Other than that, REST is usually the better option.

Why REST?
Here are a few reasons why REST is almost always the right answer. Since REST uses standard HTTP it is much simpler in just about ever way. Creating clients, developing APIs, the documentation is much easier to understand and there aren't very many things that REST doesn't do easier/better than SOAP. REST permits many different data formats where as SOAP only permits XML. While this may seem like it adds complexity to REST because you need to handle multiple formats, in my experience it has actually been quite beneficial. JSON usually is a better fit for data and parses much faster. REST allows better support for browser clients due to it's support for JSON. REST has better performance and scalability. REST reads can be cached, SOAP based reads cannot be cached. It's a bad argument (by authority), but it's worth mentioning that Yahoo uses REST for all their services including Flickr and del.ici.ous. Amazon and Ebay provide both though Amazon's internal usage is nearly all REST source. Google used to provide only SOAP for all their services, but in 2006 they deprecated in favor of REST source. It's interesting how there has been an internal battle between rest vs soap at amazon. For the most part REST dominates their architecture for web services.

Why SOAP?
Here are a few reasons you may want to use SOAP. WS-Security

While SOAP supports SSL (just like REST) it also supports WS-Security which adds some enterprise security features. Supports identity through intermediaries, not just point to point (SSL). It also provides a standard implementation of data integrity and data privacy. Calling it "Enterprise" isn't to say it's more secure, it simply supports some security tools that typical internet services have no need for, in fact they are really only needed in a few "enterprise" scenarios. WS-AtomicTransaction

Need ACID Transactions over a service, you're going to need SOAP. While REST supports transactions, it isn't as comprehensive and isn't ACID compliant. Fortunately ACID transactions almost never make sense over the internet. REST is limited by HTTP itself which can't provide two-phase commit across distributed transactional resources, but SOAP can. Internet apps generally don't need this level of transactional reliability, enterprise apps sometimes do. WS-ReliableMessaging.

Rest doesn't have a standard messaging system and expects clients to deal with communication failures by retrying. SOAP has successful/retry logic built in and provides end-to-end reliability even through SOAP intermediaries.
REST is an architecture. REST will give human-readable results. REST is stateless. REST services are easily cacheable.

SOAP is a protocol. It can run on top of JMS, FTP, Http. REST is over HTTP, but SOAP can be over any transport protocols such HTTP, FTP, STMP, JMS etc.

Soap Web-services :

If your application needs a guaranteed level of reliability and security then SOAP offers additional standards to ensure this type of operation.

If both sides (service provider and service consumer) have to agree on the exchange format then SOAP gives the rigid specifications for this type of interaction.

RestWeb-Services :

Totally stateless operations: for stateless CRUD (Create, Read, Update, and Delete) operations.

Caching situations: If the information needs to be cached.

To summarize their strengths and weaknesses:

***** SOAP *****

Pros:

Langauge, platform, and transport agnostic

Designed to handle distributed computing environments

Is the prevailing standard for web services, and hence has better support from other standards (WSDL, WS-*) and tooling from vendors

Built-in error handling (faults)

Extensibility

Cons:

Conceptually more difficult, more "heavy-weight" than REST

More verbose

Harder to develop, requires tools

***** REST *****

Pros:

Language and platform agnostic

Much simpler to develop than SOAP

Small learning curve, less reliance on tools

Concise, no need for additional messaging layer

Closer in design and philosophy to the Web

Cons:

Assumes a point-to-point communication model--not usable for distributed computing environment where message may go through one or more intermediaries

Lack of standards support for security, policy, reliable messaging, etc., so services that have more sophisticated requirements are harder to develop ("roll your own")

Tied to the HTTP transport model

http://searchsoa.techtarget.com/tip/REST-vs-SOAP-How-to-choose-the-best-Web-service

REST vs. SOAP

Multiple factors need to be considered when choosing a particular type of Web service, that is between REST and SOAP. The table below breaks down the features of each Web service based on personal experience.

REST

The RESTful Web services are completely stateless. This can be tested by restarting the server and checking if the interactions are able to survive.

Restful services provide a good caching infrastructure over HTTP GET method (for most servers). This can improve the performance, if the data the Web service returns is not altered frequently and not dynamic in nature.

The service producer and service consumer need to have a common understanding of the context as well as the content being passed along as there is no standard set of rules to describe the REST Web services interface.

REST is particularly useful for restricted-profile devices such as mobile and PDAs for which the overhead of additional parameters like headers and other SOAP elements are less.

REST services are easy to integrate with the existing websites and are exposed with XML so the HTML pages can consume the same with ease. There is hardly any need to refactor the existing website architecture. This makes developers more productive and comfortable as they will not have to rewrite everything from scratch and just need to add on the existing functionality.

REST-based implementation is simple compared to SOAP.

SOAP

The Web Services Description Language (WSDL) contains and describes the common set of rules to define the messages, bindings, operations and location of the Web service. WSDL is a sort of formal contract to define the interface that the Web service offers.
SOAP requires less plumbing code than REST services design, (i.e., transactions, security, coordination, addressing, trust, etc.)
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 need not worry about writing this plumbing code into the application layer themselves.
SOAP Web services (such as JAX-WS) are useful in handling asynchronous processing and invocation.
SOAP supports several protocols and technologies, including WSDL, XSDs, SOAP, WS-Addressing
In a nutshell, when you're publishing a complex application program interface (API) to the outside world, SOAP will be more useful. But when something with a lower learning curve, and with lightweight and faster results and simple transactions (i.e., CRUD operations) is needed, my vote goes to REST.

http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/
Deciding Between SOAP and REST
Before you spend hours fretting over the choice between SOAP and REST, consider that some Web services support one and some the other. Unless you plan to create your own Web service, the decision of which protocol to use may already be made for you. Extremely few Web services, such as Amazon, support both. The focus of your decision often centers on which Web service best meets your needs, rather than which protocol to use.
SOAP is definitely the heavyweight choice for Web service access. It provides the following advantages when compared to REST:

Language, platform, and transport independent (REST requires use of HTTP)
Works well in distributed enterprise environments (REST assumes direct point-to-point communication)
Standardized
Provides significant pre-build extensibility in the form of the WS* standards
Built-in error handling
Automation when used with certain language products
REST is easier to use for the most part and is more flexible. It has the following advantages when compared to SOAP:

No expensive tools require to interact with the Web service
Smaller learning curve
Efficient (SOAP uses XML for all messages, REST can use smaller message formats)
Fast (no extensive processing required)
Closer to other Web technologies in design philosophy

http://thinketg.com/finding-the-right-web-service-xmlsoap-vs-httprest/
Use SOAP when:
All you need is simple operations, like read only methods
Implementing a one-way or one-object service for something like data exchange or transfer
You want finer control over the specific transport of data, or can't always use HTTP
Rigid specifications need to be enforced on incoming requests, and you want to minimize additional documentation needed for use
You can rely on client ability to parse XML, or more preferably SOAP itself
You need built-in error handling when things go wrong with requests

Use REST when:
Operations are complex, like create/read/update/delete on objects
Implementing a multi-faceted service for a variety of different objects
You want to easily and quickly target a variety of consumer end user devices as clients
Requests are generally stateless, like call and response compared to a conversation
Your clients may have limited bandwidth or processing power
You can leave it up to the client to get their requests correct and to deal with problems