Qualitatively

The rules in SOAP are important because without these rules, you can’t achieve any level of standardization. REST as an architecture style does not require processing and is naturally more flexible. Both SOAP and REST rely on well-established rules that everyone has agreed to abide by in the interest of exchanging information.

SOAP uses WSDL for communication between consumer and provider, whereas REST just uses XML or JSON to send and receive data

In both SOA and REST, interoperability is achieved by using common data types (usually structured), either by sharing a schema (i.e., WSDL files) or by using a previously agreed data type (typical in RESTful applications).

SOAP is language, platform, and transport independent 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), but REST requires use of HTTP/HTTPS. Our implementation takes advantage of SOAP so it is also language, platform, and transport independent.

SOAP works well in distributed enterprise environments, but REST assumes direct point-to-point communication and our implementation works well with distributed enterprise environments [CHECK ARTICLES]

SOAP is standardized REST \*\*

SOAP spends a lot of bandwidth communicating metadata so it is not good when your bandwidth is very limited. 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.

SOAP is hard to implement and is unpopular among Web and mobile developers

SOAP provides significant pre-build extensibility in the form of the WS\* standards

SOAP has automation when used with certain language products. For example if you know WSDL file REST has no automation

* REST is easier to use for the most part and is more flexible and no expensive tools require interacting with the Web service. Although REST provides structure, it supports a single syntax interface and only a set of previously agreed data types.   Also, its apparent simplicity hides the functionality that the interlocutors need to be endowed with.
* REST is more efficient because SOAP uses XML for all messages, REST can use smaller message formats so it is better to use when information about objects doesn’t need to be communicated to the client. It is beeter to use • When clients need to have access to objects available on servers. both SOA and REST rely, in practice and in most cases, on HTTP and text based data (XML or JSON).

But Our implementation uses binary [Article]

REST is faster than SOAP because no extensive processing required. In case of our implementation

if the information can be cached because of the totally stateless operation of the REST approach

The simplicity of JSON is combined with operation descriptions in a service-oriented distributed programming language that provides design time self-description without the need for a separate schema language.

There is usually no support for partial interoperability and polymorphism in distributed systems. Compliance [21] and conformance [22] are concepts that support this, but they are not used in the SOA and REST contexts.

* good characteristics of the SOA and REST models can be combined by using a better designed interoperability model, based on active resources instead of passive document descriptions;
* Description of resources need also to be adequate for people, the developers. JSON and REST are simpler than XML and SOA but less equipped for design-time support. The ideal would be to combine the simplicity of the former with the design-time support provided by the latter;
* *Decoupling*. Web interaction at the service level is based on passive data resources (e.g., XML and JSON), with interoperability based on schema sharing (SOA, with WSDL) or pre-agreed media types (REST). This implies interoperability for all possible variants of service interaction and entails a higher level of coupling than required. We propose to use *partial interoperability*, based on the concepts of *compliance* and *conformance* (defined below).
* A systematization of the resource interoperability problem, by providing a resource interaction model (Fig. 19.1) and an interoperability abstraction scale (Table 19.1);
* 􏰀  A new architectural style, Structural Services (Table 19.2), which combines the structural capability of REST with the functional variability of SOA, with a platform-independent execution model and a language well matched to this style;
* 􏰀  New ways of supporting Big Data in the IoT, namely in terms of efficiency (for volume and velocity) and decoupling (for variety, or heterogeneity), with:
  1. –  Native support for binary data;
  2. –  Variable source information (which can be omitted in frequent cases);
  3. –  Coupling reduced to the bare minimum required by the application, due to  the use of structural interoperability (compliance and conformance) instead  of schema sharing;
  4. –  Resource-based contextual information, which provides a better decoupling  than the traditional approach of activating context layers since the context sent as part of a message is data also subjected to structural interoperability.