| **SOAP** | | **REST** | |
| --- | --- | --- | --- |
| **Pro** | **Con** | **Pro** | **Con** |
|  |  | Many REST applications understand XML and those that don’t can and do use a conversion kit. |  |
| **2.  SOAP works best wherever the (REST examples) ~~above~~ is not the only communications alternative (i.e. where some of the cases below would have to be considered):**   * Service to Service communication rather than client to Service * All communication through a 3rd party message broker (mostly provided by an Enterprise Service Bus) * Guaranteed message delivery * Service discovery * Publish / Subscribe functionality with asynchronous event arrival * Service Governance (featuring version control and automatic “version conversion” of messages in transit when necessary) * Automatic content filtering (for security) * Authorization (ACL lists, SAML assertions, open ID, Shibboleth, ...) * Asynchronous IO * A communication framework exists which can be leveraged (supported by standardized technologies) * Security Managers  (WS-Security) * Workflow generators (BPEL) – allows standardization of process behavior as well as process data * Non-hierarchical Service Registry (UDDI or private XML Registry schemes) in which services from multiple organizations are represented * A single session is established once (via a Session Broker), and many different services can then be accessed directly (without an intermediary) |  | **REST works best for a series of Request / Response exchanges over a direct connection between a client and a RESTful Service.**   * User utilizing a mobile device communicating directly with a REST service (over the internet or in a “cloud”) * User at a browser using downloaded JavaScript to communicate with a REST service * Little prior connection between client and service “organizations” * Dynamic discovery of service capabilities (a very nice feature) * No interoperability framework to leverage * Minimal security (user ID, password) * Minimum security (user / password) * Best Practices to support “detection of changes since a specified timestamp” (effectively a polled event) * No legacy applications |  |
| to leverage existing service deployments and / or scalability is important |  | simplicity and quick (synchronous) response are key |  |
| WebService's WSDL is a contract between two-protocol exchanging entities.  Banks, Government agencies, stock markets, all use WebServices to establish a connection and submit remote operations.  The payload of a Webservice call is typically well protected in terms of security and encryption and can be large. |  | REST is developed for quick transfer of small data in an ad-hoc way, just like the fact that any http client can talk to any http server with no further configuration. This is not true of Webservices where you must know the names, addressing models and procedural conventions of the particular service.*REST describes a world in which hypermedia is the engine of application state*. You download a document, and it contains some links, which the user (or application) follows with more REST requests to obtain more fragments. |  |
| Webservices allows for numerous (and increasingly more complete) security models such as single sign-ons, expiring tokens and username/passwords.  The security models are independent of the specification, so that we can evolve our security based on what is needed. |  |  | JSON and REST do not have any security mechanisms and need to rely on whatever HTTP provides, so the solutions end up being vendor-based rather that standards-based. |

| **XML** | | **JSON** | |
| --- | --- | --- | --- |
| **Pro** | **Con** | **Pro** | **Con** |
|  |  |  | Almost no SOAP applications understand JSON. |
| XML can be standardized because there is a common way to express that standard:  the XML Schema Description (XSD).  This is universally supported by multiple WS developer kits which can generate code to automatically validate the XML and marshal / unmarshal it into Java or C# objects. |  |  | …not aware of anything even vaguely equivalent on the JSON side. |
| XML / SOAP bindings are supported by WSDL 1.1 (by far the most widely deployed version) which allows web client stubs to be auto-generated. |  |  | The JSON alternative (WADL) is new, and relatively rarely adopted. |
| All other educational standards (SIF, IMS, PESC) have XML mappings. |  |  | A JSON decision for CDS would fragment this space. |
|  |  |  | an empty list in XML really confuses a JSON converter |
| XML as the semantically complete way to provide the formal specifications. |  | JSON is a point solution rendering that satisfies specific needs - such as mobile and REST deployments. |  |
| Tools (are available to) simply down-shift the XML into JSON at (certain) ~~that~~ delivery point(s) |  |  |  |
|  |  | JSON-LD … makes some sense for publishing applications (e.g.  a spreadsheet of student enrollment demographics as JSON-LD content to augment Web page and PDF documents.) | - but not for formal transaction exchanges. |
| current PESC standard which specifies an e-transcript as XML. |  |  |  |
|  | heavy operational system overhead of processing XML vs. JSON |  |  |
| XML and in particular XML Schema is a contract between two document-exchanging entities.  XML is document oriented, |  | JSON is data oriented. JSON was developed to closely match the exchange of objects. it maps more directly onto a programming data structure. |  |
| XML is a W3C standard. |  |  | JSON is an evolving consortium. |
| By making the Webservice interfaces data agnostic (their payload is the XML documents), we create a loosely coupled architecture where the data you receive is parsed into an object. |  | rather than mapped (as in JSON) |  |
| (As a loosely coupled XML-based system) XML … will allow for more flexible handling. |  |  | In a tightly coupled (JSON-based) system, all parties must be at the exact version at the exact time. |
| Webservices allows for numerous (and increasingly more complete) security models such as single sign-ons, expiring tokens and username/passwords.  The security models are independent of the specification, so that we can evolve our security based on what is needed. |  |  |  |

**Other**

On the question of JSON vs. XML: **It should be one or the other and not both**

·         Standardizing on both means you are effectively determining what a JSON equivalent is for every possible XML message ... a matter best left to conversion software because sometimes there is no clear answer (ex: an empty list in XML really confuses a JSON converter).

·         Standardizing on one, means you standardize “the wire”.  Any application then can be assured that if it meets the standard it will interoperate with any other conformant application ... without trying to match up on which of the standardized formats are supported.

·         This approach can be carried further. The choice of transport should either be SOAP or REST.  If the decision is taken to standardize on SOAP for example, the standard should specify whether it is SOAP v1.1 or SOAP v1.2 and whether WS-Addressing is being used).  The tighter the standard is from the beginning, the greater the interoperability between conformant applications.  And of course, it is always much easier to loosen a standard later than to tighten it.

=================================

The original idea was to use JSON as a mechanism for the exchange of payloads because of the heavy operational system overhead of processing XML vs. JSON.  This was deemed to be too far from the current PESC standard which specifies an e-transcript as XML.  There was an additional thought that the transactional interchange parameters be passed as JSON and not XML, but still deliver the payload as XML.  I think Ron does a great job at making a case for one vs. two and Parchment’s opinion aligns with Ron’s in this case.

I do disagree with his opinion that it should be XML in that if we choose REST over SOAP, then defining the transaction parameters as JSON would be preferable to a broader set of developers.  Again, this presume that the payload of the transaction is still an XML eTranscript.

With respect to SOAP vs REST, we remain neutral on the topic.  Our only requirement is that we do not revert back to a single WSDL model as originally proposed.  Our preferences:

-          SOAP based api with multiple WSDLs.  Use of XML or JSON for transaction semantics and XML for payload delivery.

-          Restful api with JSON for transaction semantics and XML for payload delivery.

=================================

1. XML and in particular XML Schema is a contract between two document-exchanging entities.  XML is document oriented, JSON is data oriented. JSON was developed to closely match the exchange of objects. it maps more directly onto a programming data structure.  XML is a W3C standard. JSON is an evolving consortium.

2. WebService's WSDL is a contract between two-protocol exchanging entities.  Banks, Government agencies, stock markets, all use WebServices to establish a connection and submit remote operations.  The payload of a Webservice call is typically well protected in terms of security and encryption and can be large.  REST is developed for quick transfer of small data in an ad-hoc way, just like the fact that any http client can talk to any http server with no further configuration. This is not true of Webservices where you must know the names, addressing models and procedural conventions of the particular service.*REST describes a world in which hypermedia is the engine of application state*. You download a document, and it contains some links, which the user (or application) follows with more REST requests to obtain more fragments.

3. We, at CDS, are architecting a Service that must be as formal as possible, as secure as possible, and as dependable as possible to exchange documents between Institutions.  We are not developing a service that's open to the public and can be adapted to perform novel acts with minimal effort.  A good analogy might be fax vs email.  There needs to be a level of trust between the exchanging parties.  We can guarantee this trust with two contracts, one for describing the data exchanged (XML schema) and the other for the sending and receiving (WebService WSDL).

4. By making the Webservice interfaces data agnostic (their payload is the XML documents), we create a loosely coupled architecture where the data you receive is parsed rather than mapped (as in JSON) into an object.  By parsing, institutions can perform their own validations on the documents they can handle, for example by making sure that their minimal requirements are met.  This will make versioning nightmares manageable.  In a tightly coupled system, all parties must be at the exact version at the exact time.  Say you are unrolling a new version of CDS.  The entire system will be broken until all members have upgraded.  They will not be able to process any documents at all because the direct mapping will fail.  To coordinate upgrades in vastly disparate systems will be a considerable challenge.  XML doesn't solve this problem, but will allow for more flexible handling.  If a student's email/contact can be extracted from a transcript payload, the institution can still process this transcript, even if only manually (or semi-manually).  This will not be the case for a JSON exception.

5. Webservices allows for numerous (and increasingly more complete) security models such as single sign-ons, expiring tokens and username/passwords.  The security models are independent of the specification, so that we can evolve our security based on what is needed.  JSON and REST do not have any security mechanisms and need to rely on whatever HTTP provides, so the solutions end up being vendor-based rather that standards-based.