NISO RP-2006-01

Best Practices for Designing Web Services in the Library Context

by the NISO Web Services and Practices Working Group

A Recommended Practice of the National Information Standards Organization

July 2006

Published by the National Information Standards Organization
Bethesda, Maryland

About NISO Recommended Practices

A NISO Recommended Practice is a recommended "best practice" or "guideline" for methods, materials, or practices in order to give guidance to the user. Such documents usually represent a leading edge, exceptional model, or proven industry practice. All elements of Recommended Practices are discretionary and may be used as stated or modified by the user to meet specific needs.

Contents

Foreword ii
Purpose and Scope ii
Small Interfaces 1
Document Service Interface 3
Information Model 3
Behavior Model 4
Action Model 4
Process Model 4
Description Limits 4
Policies and Contracts 5
Enable HTTP Caching 5
Filter User Input 6
Reuse Existing Output Formats 6
Document Output Formats 7
DTD (Document Type Definition) 7
XML Schema 8
RDF Schema (Resource Description Framework Schema) 8
Relax NG (Regular Language for XML Next Generation) 8
DSD (Document Structure Description) 9
Security 9
Throttling 9
Conclusion 9
Appendix A: Web Services in the Library Context 10
Types of Web Services 10
Discovery Services 10
Locate Services 10
Requesting Services 11
Delivery Services 11
Common Services 11
Interoperability 12
Link Between Interoperability and Web Services 12
Need to Discover Services 12
Appendix B: Glossary 13

Foreword

“Web services” is a broad term variously defined. For the purposes of this paper, we are working in the sense of the IBM definition:

"Web services is a technology that allows applications to communicate with each other in a platform- and programming language-independent manner. A Web service is a software interface that describes a collection of operations that can be accessed over the network through standardized XML messaging. It uses protocols based on the XML language to describe an operation to execute or data to exchange with another Web service."

Purpose and Scope

This effort outlines the actual and potential uses of web services in a library context. As the nature of web services is changing rapidly, this is only a snapshot in time. The intent is to be helpful, not limiting. Web services are seen as an alternative to fully developed application programming interfaces (API) for circumstances in which the additional overhead is not warranted. Many works exist to explain technical features of web service interactions.

The intended audiences are people from both the vendor and user community who seek to understand the role and potential of web services in aspects of library work.

NISO Web Services and Practices Working Group Members

This Recommended Practice was developed by the following members of the Web Services and Practices working group:

- Candy Zemon, Polaris Library Systems, co-chair
- Ian Davis, Talis Information Ltd., co-chair

- Kerry Blinco, IMS Australia
- Robert Bull, Crossnet
- Ruth Castillo, Auto-Graphics, Inc.
- Matthew Dovey, Oxford University Computing Services
- Emily Fayen, MuseGlobal, Inc.
- Jeremy Frumpkin, Oregon State University
- Steve Griffin, The IMS Global Learning Consortium
- Sebastian Hammer, Index Data
- Paul Harvey, Fretwell-Downing Informatics
- Ted Koppel, ExLibris, Inc.
- Ross McCarthy, Endeavor Information Systems, Inc.
- Ken Poore, Sirsi/Dynix
- Christopher Rennie, ProQuest
- Pat Stevens, NISO Standards Development Committee liaison
- Michael Teets, OCLC, Inc.
- Sean Thomas, VTLS, Inc.
- Tom Wilson, University of Maryland

Best Practices for Designing Web Services in the Library Context

In order to create an interoperable "ecology" for digital library systems, some standard practices need to be adopted, and some standards and standard technologies need to be provided in order to best support these practices. This document will recommend a set of best practices in support of interoperable digital library services.

An overview of web services in the library context, including types of web services and interoperability issues is discussed in Appendix A. Those new to web services may want to review the appendix before reading this recommended best practice.

Small Interfaces

In general, simplicity and small interfaces are to be desired in the design of web services. Small and simple interfaces are easier to document, test, maintain, and understand. Small and simple may also prove to be more robust. Length of time to implementation may be reduced if the service interface is small.

The service interface is the summary abstract level describing the web service sufficiently for another party to determine what the service does and how to interact with that service. The interface defines message types and message exchanges that the service expects and can use. The interface also specifies any conditions or requirements associated with the messages. A message exchange pattern may be included, which serves as a generic template for the actual exchange of the defined messages. It is in this pattern definition that any conditions and relationships between messages can be described. This is also the spot for defining how message exchanges are both initiated and terminated. Error handling or abnormal termination should also be considered and described here.

The service interface and its associated operations are mapped to a concrete message format and transmission protocol through a binding.

Given that the service interface is the formal exposure of what the service contains, how it expects to receive and respond to messages, and requirements around how those messages are structured and interact, the interface itself contains a lot of carefully structured information. The simpler or smaller the service interface is, the more readily understood and implemented the related service can be.

The tradeoff, of course, is that should it become desirable to expand the task addressed with further options or dependent tasks, the initial web service interface is not structurally predisposed to absorb the new complexity.

One school of thought prefers using separate small services in conjunction or in succession to building an overly complex single message. Examples of complex protocols abound in the ANSI/NISO family of Z standards. The design philosophy followed in those cases is to create an infrastructure as complex as necessary to accommodate all known and reasonably imagined use cases in the general problem space. In contrast, the design philosophy of simple web service interfaces is that the actual task to be addressed informs the extent of the interface, with small and simple being preferred both as initially easier to understand and implement and in the long term more amenable to combination with other (small interface) web services as the need arises. The issue of versioning becomes important should a web service interface get changed or expanded.

Complexity Levels	**Expandability**	
	High	**Low**
High	X	
Low		X

Figure 1: Complexity levels vs. expandability

Another tradeoff that must be considered revolves around the latency inherent in web services. Although focused simple services that move only the necessary data are desirable in terms of ease of implementation and clarity of mission, performance for long or complex tasks may be better for fewer large services each returning a lot of data.

Amount of Data Returned Levels	**Performance Speed**	
	High	**Low**
High	X	
Low		X

Figure 2: Amount of data returned levels vs. performance speed

A third tradeoff involved revolves around the intended audience or consumers of the web service. If the intent is to be able to use the service with "anyone" interested in consuming it, more forgiveness is built into the interface, more data is probably going to be received, and the service interface will most likely be larger. If the intent is to focus on an audience with specific awareness about the task to be done, the service interface can be much simpler and the data received much more focused.

Audience Levels	**Service Interface Size**	
	High	**Low**
High	X	
Low		X

Figure 3: Audience levels vs. service interface size

A fourth tradeoff, touched on above, is whether the service interface should contain a small number of operations or a large number of operations. A large number tends to imply fewer parameters needed for each operation. The tradeoff is likely to be felt most in terms of performance.

Performance Levels	Number of Operations	
	High	Low
High		X
Low	X	

Figure 4: Performance levels vs. number of operations

The service interface is generally expressed in Web Services Description Language (WSDL), a W3C specification. (At the time of writing, version 2.0 of the specification was in draft). It is an XML-based service description that includes how to use the described service (the protocol bindings and message formats) and the supported operations and messages.

Specific examples of WSDLs can be found on the W3C site. Version 1.1 examples are at http://www.w3.org/TR/2001/NOTE-wsdl-20010315. Version 2.0 can be found here: http://www.w3.org/TR/2006/CR-wsdl20-20060106/#wsdllocation.

Document Service Interface

There are several different models involved in fully describing or documenting a web service interface. Depending on the service in question, one, some, or all of the following models may be needed to adequately and unambiguously describe a service interface.

Information Model

The information model of a service includes the type of information that may be exchanged, the format in which that information can be presented, definitions of any terms used, and a consistent interpretation of strings and tokens used in the information exchange. Information models for exchanges within a single information domain may be able to leave some details unexplained. Information models for exchange across information ownership boundaries will need to be more exhaustive in all areas of the model.

The information model should also include structure (syntax) and meaning (semantics) descriptions.

Structure

It is necessary to understand the structure of the information to be exchanged. This includes several levels: the encoding of character data, the format of the data and the structural data types associated with elements of the information.

It is important to consider the use or content of the information as well as the type when describing the structure of an information model. For instance, within a street address structure, the city name and the street name are typically given the same data type—some variant of the string type. However, city names and street names are not really the same type of thing at all. Distinguishing the correct interpretation of a city name string and a street name string is not possible using type-based techniques—it requires additional information that cannot be expressed purely in terms of the structure of data.

Semantics

This centers on the meaning of the information to be exchanged. Information may be intent as well as content. For example, a purchase order combines the description of the items being purchased and the fact that one party intends to purchase those items from another party.

When information is exchanged across ownership boundaries, consistent interpretation of that information is essential. Such interpretations might be explicitly described in the information model. Existing industry-standard descriptions might be included in the information model by reference. The point is that the information itself, as well as any tokens representing information, must be interpreted consistently in order for interoperation to occur.

Behavior Model

The behavior model concerns itself with descriptions of actions, responses, and dependencies between actions included in the service.

This description will include what behavior is expected of each party including: what (if any) response is required from either party when an action is invoked by the other, what dependencies might be involved (such as a sequence of actions that cannot meaningfully occur in a different order), what error conditions exist, and what to do upon encountering them. A security-based service is an example needing a behavior model in that certain services (such as access to secure information) can only be supplied after the authentication of the queryer is successful.

Action Model

The action model lists the actions possible within the service, as well as implied effects of the actions. For example, in a service managing a bank account, beyond knowing how to access a bank account and how to issue commands (service requests), to be successful one must also understand that using the service may actually affect the state of the account (for example, withdrawing cash).

Process Model

The process model is not fully defined and the need for it may vary depending on the service. It describes the temporal relationships between and temporal properties of actions and events associated with interacting with the service. In other words, any required time-ordering of events and actions is covered.

Description Limits

Despite the models and the best intentions of those describing the service, there are limits to descriptions which will often leave unstated assumptions undocumented. Another limit to description is what happens in certain multi-branched possible responses where private choice may be involved. For example, in the case that there is more than one response, this set of responses has to be converted into a single choice. This is a private choice that must be made by the consumer of the search information.

Policies and Contracts

Beyond the models described above, there may be policies (constraints) or contracts (agreements) affecting the use of a service. Whatever method is most effective for the parties and situation involved should be used to express those policies and agreements.

Enable HTTP Caching

A significant factor in the growth of the Web has been the ability for clients and intermediaries to efficiently cache transmitted content. Caches retain copies of content that can be reused by later requests instead of asking the service for it again. Caching can reduce bandwidth consumption and also decrease the latency of requests making the service appear faster and more responsive. Caching of content can also reduce load on the web service by reducing unnecessary requests.

It is recommended that Web service implementations should make use of HTTP caching mechanisms to integrate into the existing Web infrastructure. Content accessed via HTTP GET is particularly amenable to caching and the service provider can take advantage of a number of standard HTTP headers to assist caching intermediaries.

Most caching systems will not cache HTTP POST requests since this method is designed to effect a change of state on the origin server. If the web service implementer has a free choice of HTTP methods then GET should be preferred for requests that do not modify the state of the server (idempotent requests). Most query and search services fall into this category, whereas a login or item checkout service typically will not. Enabling query or search responses to be cached can drastically reduce the load on back-end systems such as databases.

Following is a list of best practices around HTTP caching behavior as it applies to web services.

- **Best Practice: Prefer HTTP GET for Requests That Do Not Modify the State of the Service**

  POST requests will not usually be cached by intermediaries so use GET where the request will not modify the state of the service.

  A particularly efficient mechanism provided by HTTP is *conditional GET*. This mechanism allows clients to issue a single GET request to the server and receive either a short response indicating that the content has not changed since the last request or a copy of the new content if it has.

  Conditional GETs are controlled by two pairs of HTTP headers. The first pair is ETag which is sent by the server and *If-None-Match* which is sent by the client. The content of the ETag header is a string enclosed in quotes that uniquely identifies the content being sent in the response. This string can be stored by clients who may then use it when performing subsequent requests. If the server has generated an E Tag for some content a client may at a later date re-request the content and specify an *If-None-Match* header containing the value of the ETag previously supplied. If the server determines that the content to be returned would have the same ETag it can return a 304 response code with no content. If the content is different, the server should return a 200 response code and include the new content.

- **Best Practice: Responses Should Include an ETag HTTP Header**

  Supplying an ETag response header enables clients and caches to save bandwidth and processing time by using conditional GET requests with the If-None-Match request header.

- **Best Practice: Responses Should Include a Last-Modified HTTP Header**

  Supplying a Last-Modified response header enables clients and caches to save bandwidth and processing time by using conditional GET requests with the If-Modified-Since request header.

- **Best Practice: Prefer Conditional GET Headers over the Expires HTTP Header**

  To be effective the Expires header requires clocks to be synchronized between client and server. Prefer ETag and Last-Modified headers instead since all comparisons are performed by the server alone.

- **Best Practice: Avoid User Specific Information in URLs**

  Using URLs without user identifiers for content that is not user specific increases the chance that a particular request can be satisfied via a cache.

  Many web services provide content that is not user specific such as search results or data conversions. This kind of content is ideal for caching by intermediaries such as proxy caches which may be serving entire institutions or Internet service providers. However if user-specific information such as session identifiers are included in the URL the efficiency of the caches are drastically reduced. The cache cannot assume that the content accessed by slightly different URLs is the same and therefore must forward many more requests to the origin web service. If possible, web services should avoid user information in URLs for content that is suitable for multiple users.

Filter User Input

It is a fact of life that web-based applications are subject to web-based threats to security or integrity of operation. The end user may be a legitimate consumer of the service. Or it may be malware or some other security threat. For that reason, all user input to a web service needs to be intercepted and vetted in some sense before being allowed to initiate the service or consume its results. Such attention to verifying the integrity of the input will go a long way toward protecting the web service from unintended use.

Reuse Existing Output Formats

When designing a web service careful thought must be given to the format of the service requests and responses. These formats must be machine processable but still expressive enough to convey all the information necessary for the service to function correctly. Typically XML is used to structure output formats, although there are many other options depending on the specific situation. Often the needs of a particular service can be met by reusing an existing well known format. It is recommended that web service designers look for and use an existing format rather than invent a new one specifically for their service.

Reuse of an existing format can lower development costs for the client and can increase the potential for interoperability. Clients may be able to reuse existing parsing code, data structures or frameworks when building in support for the service. In many situations the client may already be able to process the format if it is widely known and so the incremental development costs for supporting an additional service becomes very low. The client developer may also benefit from the community's experience with deployed formats. Problems with the format such as ambiguities or inconsistencies are usually well-known and publicized either through formal errata or informal articles, papers and discussions. This community knowledge can become enshrined as interoperability profiles, or simply as a set of best practice rules for consuming and producing the format.

Compare this with the situation where the service uses a new format for its output: the client developer has to discover and understand the documentation of the format; map this to any existing data structures, resolving conceptual overlaps, omissions or uncertainties; write parsing code to populate the data structures and then embark on a cycle of testing the boundaries and edge cases of the format. All of this development has to be conducted either in isolation or with a lot of potentially costly support from the service provider.

In the library domain there are many existing formats that are well understood and have good client support. The classic example is, of course, MARC, and more recently MARC XML. Other formats commonly used for various segments of the library domain include Dublin Core, MODS, PRISM, and ONIX.

Service developers should also consider support for data formats with standardized data models such as RDF. Using a format like RDF eliminates the need for the client developer to write a specific parser since the rules for parsing XML versions of RDF are well specified and widely implemented. If the client adopts the RDF data model then the need for separate data structures representing each format is removed along with much of the data mapping costs. The client application still requires logic to interpret the data model but this task is simplified by many existing frameworks. By supporting RDF the service provider's responsibilities move from deciding on a particular format to encapsulate the service data to determining how best to represent that data within the standardized data model.

Sometimes clients of the service will have access to existing parsers for known formats. For example if the web service is primarily designed for access by rich web clients running within a web browser then consider using XHTML as an output format since the browser will have an efficient parser specifically designed for the purpose. It may even be possible in this situation for the client to simply incorporate the XHTML directly into its own user interface, bypassing any specific parsing. Another option in this situation would be to output JSON (Java Script Object Notation) data structures which can be parsed directly by the script calling the service. This is an example of a non-XML output format that may be appropriate for the situation.

Document Output Formats

Web services, once developed, need to be described in a way that is understandable to humans and directly accessible to machines. There are several commonly-used methods for expressing/codifying a web service. A brief description of some of these follows, including some differentiating features of the various methods.

DTD (Document Type Definition)

This method of documenting output formats depends on declarations of elements and attribute lists. An element declaration names the allowable set of elements within the document. Part of this is to also specify whether declared elements may be contained within each element. Rules about how such inclusions occur are also stated. Attribute-list declarations name the allowable set of attributes for each declared element, including the type of each attribute value, or an explicit set of valid values. A DTD is separate from the XML document it is meant to "explain" or define. A Document Type declaration in the target document will associate it with its relevant DTD. There is a DTD syntax that must be adhered to in order to produce a valid DTD.

Among the strengths of the DTD are the facts that support for DTD is widespread in XML tools and that DTD is included in the XML 1.0 standard.

Among the weakness of the DTD are the facts that the DTD does not support some important XML features like namespaces and that there are some limits to what DTD can express—it is more limited than XML in general

XML Schema

XML Schema, also referred to as XML Schema Definition (XSD), has achieved Recommendation status within the W3C. It is a schema language that also supports validation, collecting information about the document structure during the act of validation. XML Schema is associated with Microsoft support and is particularly apt to support object oriented programming.

Among the strengths of the XML Schema are the Microsoft connection, the easy linkage with object oriented programming, and the W3C Recommendation status.

Among the weaknesses of the XML Schema are the Microsoft connection which leads to a charge of lack of openness, and some restrictions arising from datatype dependencies on other W3C specifications.

RDF Schema (Resource Description Framework Schema)

Resource Description Framework (RDF) refers to a group of specifications for a metadata model that is often implemented in XML. RDF specifications are maintained by the World Wide Web Consortium (W3C).

RDF Schema is an extension of RDF. It can describe groups of related resources and their interrelationships. RDF Schema descriptions are written in RDF.

Among the strengths of the RDF Schema is the fact that it is a favored method for Semantic Web and knowledge management applications and it is part of a set of specifications maintained by the W3C.

Among the weaknesses of the RDF Schema are its very association with the Semantic Web and the slow uptake of this method by very simple web services.

Relax NG (Regular Language for XML Next Generation)

RELAX NG is a schema language for XML that specifies a pattern for the structure of the XML document. There is also a non-XML syntax of Relax NG. An OASIS technical committee maintains the specification. Relax NG is also part of the ISO Document Schema Definition Languages (DSDL) standard (ISO/IEC 19757).

Among the strengths of Relax NG are the simplicity of use and the leverage of nesting Russian-doll structure as well as support for data typing, regular expressions, and namespaces. It also supports interleaving.

Among the weaknesses of Relax NG are its difficulty in handling recursive elements in its nested version, and the fact that the W3C XML Schema specifications are better known and more widely adopted, though that is slowly changing. Relax NG also lacks the range of datatypes supported by XML Schema.

DSD (Document Structure Description)

DSD is a schema language for XML, meaning that it is a language used to describe valid XML documents. DSD is an alternative to DTD and to the XML Schema. It is reputed to be an extremely flexible easy to use language. But its differences with and separateness from the W3C set of specifications can be viewed as a barrier to common use.

Another aspect of documenting existing web services is the whole "phone book" question. How does an interested entity discover whether a web service is already defined that might meet a current need? The same issue comes up in every communication situation. How do I find others to communicate with? In some venues, this need has been answered by a registry in which entities list their capabilities. In some instances, a directory, either maintained centrally or distributed, can answer the "who can I talk to" question. NCIP (National Circulation Interchange Protocol) has long been discussing a policies directory to meet a similar need. The Australian ILL community has instituted a centralized directory of providers. The current situation in web services is not very different. Various lists of web services, categorized or not, exist. These are generally sponsored either by major interested parties such as Microsoft's UDDI directory listings or by technical groups such as W3C's Web Services Activity.

Security

Other NISO groups have worked on security in terms of authentication methods. Of particular aptness here is the NISO Metasearch Initiative's paper on authentication methods. For web services in general, secure transport in a web environment (usually meaning https) plus attention to authentication methods will help ensure that the web service is used for its intended purpose by its intended audience while safeguarding any personal information it may use.

Throttling

Web applications can be swamped by too much traffic or too heavy demands. Just as user input is inspected and accepted with caution, so queries for the service may have to be intercepted, examined, and treated either in separate streams depending on the results to be delivered, or meted out in a measured way so as not to overwhelm the resources of the service.

Conclusion

This paper has given an overview of the issues involved in implementing and designing web services that may be of use in the library environment. As web services become a more common tool for communication between applications, unforeseen library-specific uses may arise. The intent of this paper is to explain briefly some of the decisions involved in finding, designing, implementing, and using web services.

Appendix A: Web Services in the Library Context

Types of Web Services

The following list of categories is neither exhaustive nor exclusive of one another, and particular implementations of web services are likely to be able to fall into more than one category. The following categories and examples are intended as guiding examples and to provide some application of commonly used web service categorizations to library web services

Discovery Services

discover metadata, full text, or a service

1. Web services to search and retrieve results from a data repository (such as a library catalog or a licensed database)

    Examples: searching a library's catalog holdings from a book vendor web site, searching full text database for a particular citation, or metasearching several remote resources with a single query.

2. Web services to create and maintain a directory (such as a directory of services, a directory of policies, or a directory of members)

    Examples: the Talis directory of services and the OCLC directory of ILL policies.

Locate Services

resolve an object to its location, either physically or in a process

1. Web services to communicate inventory information from standalone units to the circulation system (such as a storage facility or a separate library location).

    Examples: communication between inventory management systems or closed storage locations and central circulation systems.

2. Web services to communicate requests and circulation transactions between peer circulation systems (such as is common in direct consortial borrowing scenarios).

    Examples: bookmobile transactions, e-book circulations occurring on a vendor web site

3. Web services to communicate library holdings to third party service providers (such as reading reference services).

    Examples: communication of library holdings to courseware applications, communication of holdings from e-journal aggregators to library applications

4. Web services to communicate acquisition status and metadata from an ordering system to a repository (such as between the library acquisitions system and a governing body).

Examples: Updating funding authority with current fund status, updating acquisitions system with web-placed vendor orders

Requesting Services

requesting information, object, etc.

1. Web services to communicate orders to publishers.

   Examples: services that allow a library to select and order materials from a publisher's web site and services that return appropriate information for the library's acquisitions process

2. Web services to communicate hold requests between circulation systems and third party electronic materials suppliers.

   Examples: communication between library public catalogs and hosted content sites of many types: e-book vendor sites, or e-journal or document supply sites.

3. Web services to retrieve and manipulate non-text materials such as images and sound files

   Examples: access to downloadable audio, access to downloadable images

Delivery Services

facilitating delivery of objects, information, formatting, transactional information, etc.

1. Web services to communicate circulation information between circulation systems and third party services.

   Examples: self-check stations, electronic materials suppliers or payment system stations.

2. Web services to reformat and normalize metadata (similar to crosswalks).

   Example: data migration transformations

Common Services

administrative, back office, systems integration services, etc.

1. Web services to communicate with financial entities

   Examples: the institution's financial system or credit card payment management services

2. Web services to transmit and receive software changes and updates

   Examples: remote system updates, remote system configuration

3. Web services to support interoperability between ID management systems and transactional systems

   Examples: remote authentication and authorization systems

Interoperability

At the core of web services is the idea of interoperability. Interoperability is defined in the Wikipedia as "The ability of systems, units, or forces to provide services to and accept services from other systems, units, or forces and to use the services so exchanged to enable them to operate effectively together."

Link Between Interoperability and Web Services

While various definitions of web services do not utilize the actual word "interoperability", they speak or infer the functionality described in the definition above. In the context of library (and digital library) systems, interoperability, if not ***the*** key factor, is ***a*** key factor for designing information systems that operate in a greater context. Libraries use standards such as MARC, Dublin Core, OpenURL, etc. because they allow a level of portability between systems and institutions. However, until recently the amount of dynamic interchange between different library systems has been limited at best. However, more recent trends with digital library systems and technology, as well as within the web technology community as a whole, point to a rapid scaling of new interconnected information systems. Companies such as Google are advertising API's to new services (such as Google Maps). The trend to open systems promotes the ability to "hook" new tools and services together.

Need to Discover Services

Registry efforts, such as UDDI, IESR, Ockham, and OCLC's OpenURL registry provide a discovery mechanism which is vital to auto-enabling service interoperability. These registry efforts organize the web of available services so that a set of services can be put together to support a particular information function.

Appendix B: Glossary

This glossary defines key terms used in the document.

HTTP – an acronym for Hypertext Transfer Protocol, which is a protocol for distributed hypermedia information systems. HTTP is characterized as a small but extensible set of operations that can be used to fetch, update, create, and remove remote resources identified by URIs. HTTP is the protocol used for the vast majority of the World Wide Web.

REST – an acronym for Representational State Transfer, an architectural style for large scale hypermedia systems. Key to the REST style is the notion that the state of the system is described by a set of uniquely identified resources which can be manipulated by a restricted set of ubiquitous operations. In terms of Web Services the resources are usually data items identified by URIs which respond to standard HTTP methods such as GET, POST, PUT and DELETE.

SOA – an acronym for Service Oriented Architecture, an architectural style characterized by the system components being loosely coupled via well defined interfaces. This term is most commonly used when describing a system consisting of components implemented as Web Services.

SOAP – a message exchange protocol that uses XML to encode structured data exchanged between peers. Most SOAP based Web Services operate using HTTP, typically by accepting SOAP-encoded XML documents sent using the POST method. However the SOAP protocol can also be operated over other transports which may provide benefits such as guaranteed delivery or queuing of messages.

URI – an acronym for Uniform Resource Identifier, a sequence of characters that denotes a name for a single resource. The term URI encompasses other related terms such as URL (Uniform Resource Locator) and URN (Uniform Resource Name).

Web Service – a programmatic interface used for application to application communication via the World Wide Web. Generally web services utilize HTTP to transport XML requests and responses. Protocols such as Z39.50 are not typically considered to be Web Services.

WSDL – an acronym for Web Service Description Language, which is an XML format for describing the interface and expected data types for a Web Service.

XML – an acronym for Extensible Markup Language, a textual format for representing structured information. XML was designed to be a simpler form of SGML (ISO 8879) for use on the World Wide Web and remains a subset of that language.