Messaging Systems

Scott Seely
http://www.pluralsight.com/

Outline

- Classes of Messaging Systems: Enterprise and Internet
- XML
- JSON
- SOAP and WS-*
- REST and Resource Oriented Architecture

Classes of Messaging Systems

- **Internet**
  - Protocol: HTTP|S
- **Enterprise**
  - Protocol: HTTP|S, TCP/IP, email, UDP, MSMQ, etc.
- **Common**
  - WS-*
  - REST: XML, JSON, Atom, etc.

XML Technologies

- **Extensible Markup Language XML**

```
<root>
  <child>Some Value</child>
</root>
```

- **XML Infoset**
- **XML Namespace (XMLNS)**

```
<a:root xmlns:a="http://www.pluralsight.com">
  <a:child>Some Value</child>
</a:root>
```

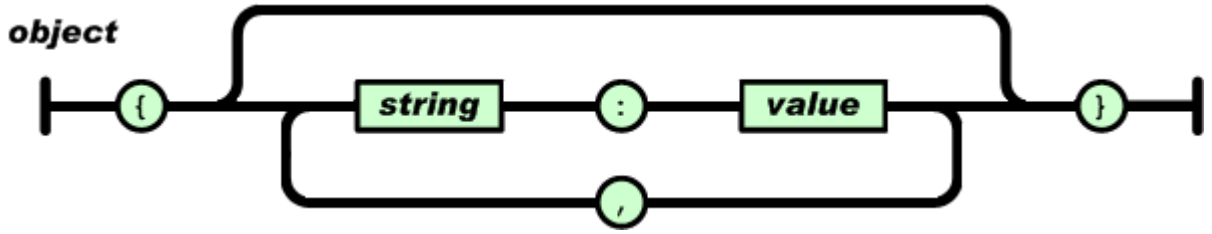
- **XML Schema Document (XSD)**

XSD Sample

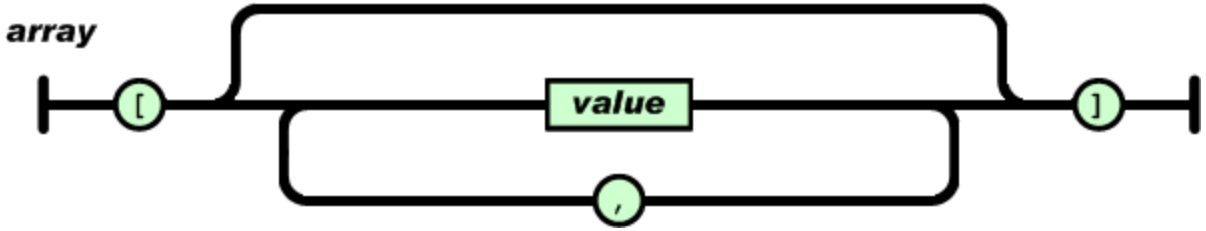
```
<xs:schema targetNamespace="http://www.pluralsight.com/"
       elementFormDefault="qualified"
       xmlns="http://www.pluralsight.com/"
       xmlns:tns="http://www.pluralsight.com/"
       xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:complexType name="root">
     <xs:sequence>
       <xs:element name="child" type="xs:string"/>
     </xs:sequence>
   </xs:complexType>
</xs:schema>
```

JavaScript Object Notation (JSON)

- **Builds on two structures:**
  - Name/value pairs
  - Ordered lists of values, *aka arrays*
- **Object:**

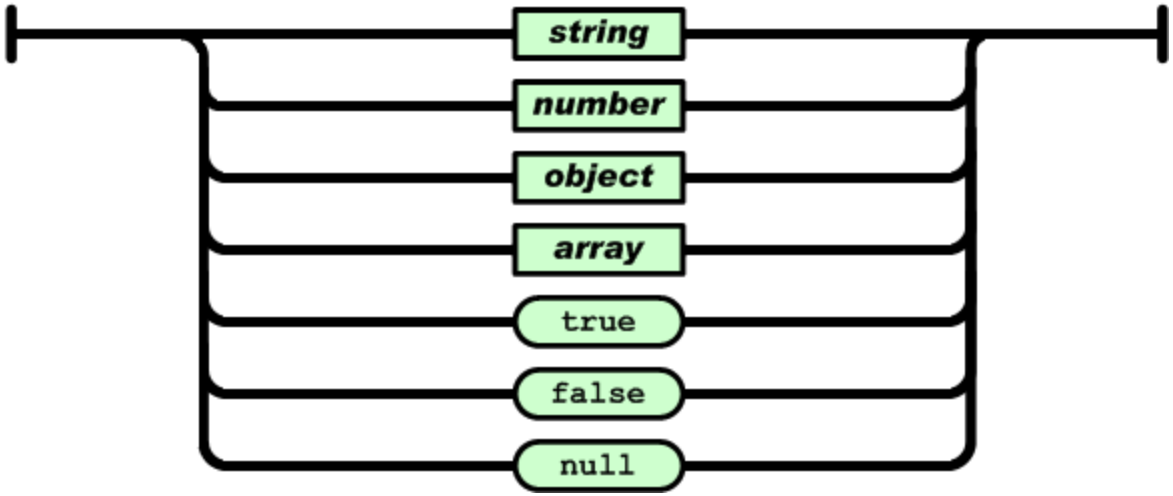


- **Array:**



JavaScript Object Notation (JSON)

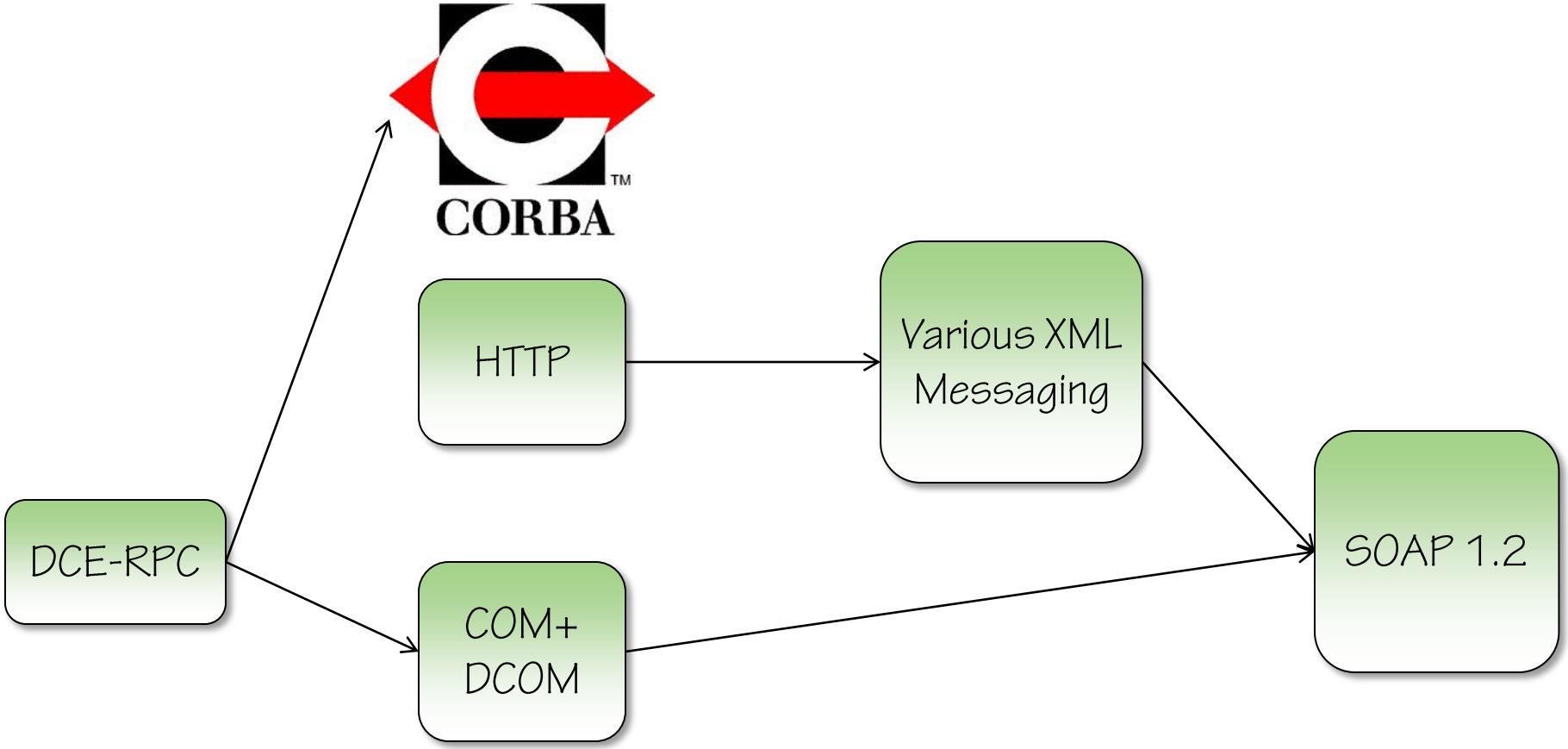
- **Value:** *value*



- **String: C-style strings.**
  - Escape character: \
  - \b, \f, \r, \n, \t, \\, \/, \", \uXXXX all have usual meanings
- **Numbers are signed, support digits [0 .. 9], decimal, and exponent**
  - 9, -10, 3.14, 6.022E+23

SOAP

RPC Evolution



SOAP Specification

- XML Infoset
- Root node: Envelope
- Metadata: Header
- Actionable data: Body
- Errors: Fault

WS-* Foundation Specs

- **Foundational Specs**
  - SOAP 1.2
  - Web Services Description Language (WSDL)
  - WS-Addressing
  - WS-Policy
  - WS-PolicyAssertions/WS-PolicyAttachment
  - WS-MetadataExchange
  - WS-Discovery

WS-* Specs (cont'd)

- **Security**
  - WS-Security
  - WS-SecureConversation
  - WS-Trust
  - WS-Federation
- **Transactions**
  - WS-Coordination
  - WS-AtomicTransaction
- **Others**
  - WS-ReliableMessaging

REST

- **Representational State Transfer**
- **Relies on Uniform Interface**
  - URI identifies Resource
  - HTTP Methods state action: GET|HEAD, POST, PUT, DELETE
    - PUT/DELETE are idempotent
    - GET is cacheable based on cache control headers
- **Resources are of two forms**
  - Data resource
  - Algorithmic resource
- **Two schools of implementation of REST**
  - Hypermedia as the engine of application state (HATEOAS)
  - The URL is King (TUK)

HATEOAS

- **Key concept expressed in Chapter 5 of Fielding Dissertation**
- **Each URI maps to a single resource**
- **A resource may have more than one URI**
  - http://www.pluralsight.com/modules/messaging/current
  - http://www.pluralsight.com/modules/messaging/2.9
- **Desired MIME type representation of resource expressed through Content-Type**
  - Accept: image/jpeg;q=0.9, image/gif;q=0.8, image/*;q=0.5
  - Accept-Encoding: gzip, deflate
  - Accept-Language: en-US

The URL is King (TUK)

- **Encode important stuff in URL**
- **Indicate canonical representation in the Content-Location HTTP header.**
- **What is the 'important stuff'?**
  - Things that change the representation. Example: .doc, .docx, .pdf, and .txt representations.
  - Things that change the language. Example: Spanish, French, English, German, Japanese, etc. translations.
- **Canonical resource looks at Accept-Type, Accept-Language.**
- **Resource specific looks at URL only for content to deliver.**
  - Example: http://www.pluralsight.com/instructors/ScottSeely/bio
  - http://www.pluralsight.com/instructors/ScottSeely/bio.es.doc
  - http://www.pluralsight.com/instructors/ScottSeely/bio.en-US.pdf

HTTP Response Codes

- 100-199: Provisional Information Response
- 200-299: Request Succeeded
- 300-399: Request needs to be redirected
- 400-499: Client error, do not repeat same request
- 500-599: Server error, better response later?

Resource Oriented Architecture

- **REST is a set of architectural principles**
- **ROA is an architectural design patterns for creating resource oriented applications**
  - Design the data set
  - Split the data set into resources
  - For each resource type:
    - Name the resources with URIs
    - Expose a subset of the uniform interface
    - Design the representations(s) accepted and served
    - Integrate the resource into existing resources with links
    - Consider the typical course of events
    - Consider error conditions

Resource Oriented Architecture

- **Resources are Addressable**
- **Resource state lives on the server**
- **Application state lives on the client until a POST/PUT/DELETE causes the resource to change.**
- **Resource states are connected. Links and forms can be used to navigate state.**
- **Uniform Interface: all interaction mediated through GET, HEAD, PUT, DELETE, POST**
  - Remember, PUT and DELETE are idempotent
  - GET and HEAD are safe (no side effects)
- **Resource Types**
  - Predefined resources (finite)
  - Data items (potentially unbounded)
  - Algorithm output (potentially unbounded)

Possible Representations

- **Already discussed XML and JSON**
- **XHTML: Embed resources as Microformats. (list of accepted ones at microformats.org). Popular ones:**
  - hCalendar: Calendar entries
  - hCard: Contact information
  - And more are in the pipeline (resume/CV, news, geo, address, etc.)
- **Atom**
  - XML vocabulary for describing lists of time-stamped entries
- **Atom Publishing Protocol (AtomPub)**
  - Adopted by OData, nice way to build your own Atom feeds!

ROA and Long Running Operations

- Submit long running task: POST Vacation Request submitted on a Sunday
- Response: 202 Accepted, Location: http://vacation.example.com/requests/1b4ca
- Update the request: PUT to http://vacation.example.com/requests/1b4ca
- GET /requests/1b4ca returns current state of request.
- To cancel request, send a DELETE

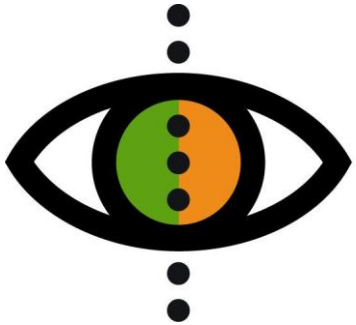
Summary

- **Classes of Messaging Systems: Enterprise and Internet**
- **XML**
  - Helpful for sending messages over non-HTTP transports
  - Gives structure, security to messages
- **JSON**
  - Easy to interpret by most languages
  - Simple form to read/write
- **SOAP and WS-***
  - Primarily useful for sending structured messages
  - Provides addressing, security, discovery, reliability
- **REST and Resource Oriented Architecture**
  - Builds apps on Web protocols/techniques
  - REST is built to scale

Resources

- Roy Fielding, Architectural Styles and the Design of Network-based Software Architectures: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (Chapter 5 introduces REST)
- Richardson and Ruby, RESTful Web Services. ISBN: 0-596-52926-0
- Scribner and Seely, Effective REST Services via .NET. ISBN: 0-321-61352-2
- Microformats: http://microformats.org
- JSON: http://json.org

For more in-depth online developer training visit



pluralsight

see what you can learn

on-demand content from authors you trust