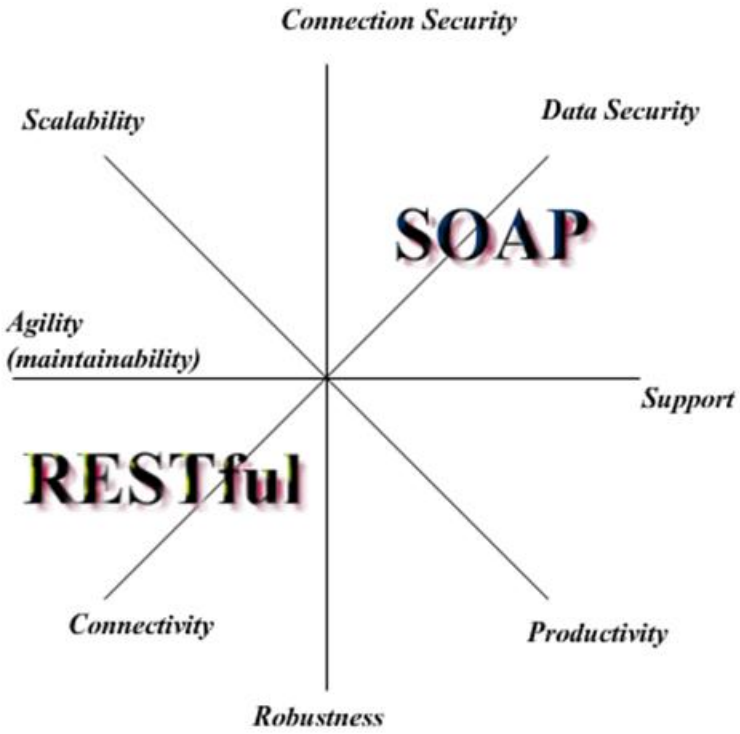
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

4. Web Services

Vilmos Bilicki PhD
University of Szeged
Department of Software Engineering

Different approaches

- Web Service
- REST
- RMI, ….

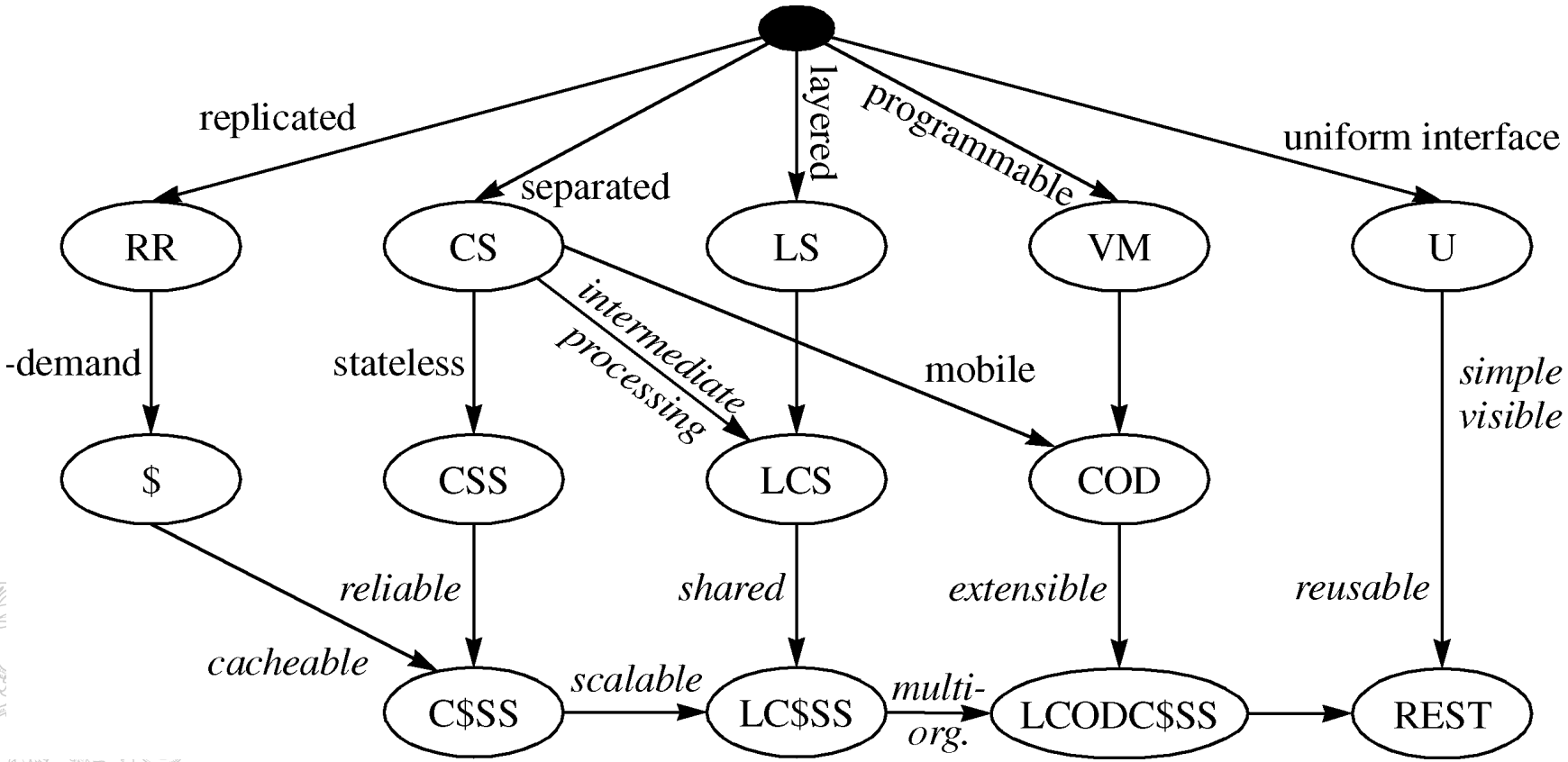


Conceptual Overview

Requirements

- Requirements supported by REST-enabled systems stem from the requirements addressed by any system following Web architecture[1]:
  - Simplicity
    - Low barrier of entry, fast adoption of Web APIs.
  - Extensibility
    - Allowing growth and flexibility.
  - Distributed hypermedia
    - Relying on the established concepts of hyperlinked content.
  - Scalability at the Web level
    - Should rely on technologies/protocols supporting scalable solutions.
  - Independent deployment
    - Coexistence of old and new

WWW


replicated
separated
layered
programmable
uniform interface
RR
CS
LS
VM
U
-demand
stateless
intermediate processing
mobile
simple visible
$
CSS
LCS
COD
reliable
shared
extensible
reusable
cacheable
scalable
multi-org.
C$SS
LC$SS
LCODC$SS
REST

Conceptual Overview

Requirements - Simplicity

- Participation in the creation of information is voluntary
  - Low entry-barrier is necessary
- Hypermedia has simple and general user interface
  - The same interface is used for all information sources
  - Hypermedia relationships are flexible – unlimited structuring
  - Users can be guided through reading by manipulating links
  - Simple queries are incorporated for searching purposes
- Partial availability of the overall system doesn't prevent the authoring of the content
  - Hypermedia authoring language is simple and capable of using existing tools
  - Unavailability of referenced information allows further authoring
  - References to the content are easily exchanged
- Communication can be viewed and interactively tested by developers

Conceptual Overview

Requirements - Extensibility

- User requirements change over time just as society does
- The system must avoid locking to the deployed solutions
  - The limitations must be easily resolvable
- A system with the goal to be long-lived as the Web must be prepared for change.

Conceptual Overview

Requirements – Distributed Hypermedia

- Hypermedia includes application control information embedded within the presentation of information.
- Distributed hypermedia allows the content and control information to be stored at remote locations.
  - Transfer of large amounts of data is needed while a user interacts with content.
- Users are quite sensitive to perceived latency
  - Time between link selection and information rendering
  - Information is distributed across the global network
  - Network interactions must be minimized.

Conceptual Overview

Requirements – Internet Scale

- The Web is Internet-scale distributed hypermedia system
- The Web must answer to to the problem of anarchic scalability
  - The constituent systems are not centrally managed neither have a common goal
  - Parts must continue to operate even under unanticipated load, or when given malformed or maliciously constructed data.
- Security becomes a significant concern
  - Multiple trust boundaries may be present in any communication
  - Additional authentication must be in place before trust can be given
  - Authentication may degrade scalability

Conceptual Overview

Requirements – Independent Deployment

- Systems must be prepared for gradual and fragmented change
  - Old and new implementations may co-exist without preventing the new implementations to achieve their full potential.
- Existing design decisions must acknowledge future extensions.
- Old systems must be easily identifiable
  - Legacy behavior can be encapsulated without impacting newly deployed subsystems
- The architecture must allow deployment of new elements in a partial and iterative fashion
  - Not possible to enforce deployment order.

Conceptual Overview

Representational State Transfer (REST)

- Representational State Transfer (REST)
  - A style of software architecture for distributed hypermedia systems such as the World Wide Web.
- REST is basically client/server architectural style
  - Requests and responses are built around the transfer of "representations" of "resources".
- Architectural style means
  - Set of architectural constraints.
  - Not a concrete architecture.
  - An architecture may adopt REST constraints.
- HTTP is the main and the best example of a REST style implementation
  - But it should not be confused with REST

Conceptual Overview

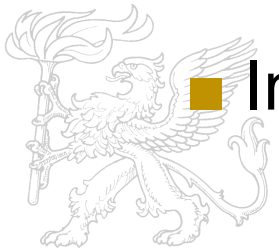
Major REST principles

- Information is organized in the form of resources
  - Sources of specific information,
  - Referenced with a global identifier (e.g., a URI in HTTP).
- Components of the network (user agents and origin servers) communicate via a standardized interface (e.g., HTTP)
  - exchange representations of these resources (the actual documents conveying the information).
- Any number of connectors (e.g., clients, servers, caches, tunnels, etc.) can mediate the request, but each does so without being concern about anything but its own request
  - an application can interact with a resource by knowing two things: the identifier of the resource and the action required
  - no need to know whether there are caches, proxies, gateways, firewalls, tunnels, or anything else between it and resource
  - The application needs to understand the format of the information (representation) returned.

Conceptual Overview

REST Architectural Constrains (1)

- Client-server
  - Separation of concerns
    - Clients are separated from servers by a uniform interface.
  - Networking
    - Clients are not concerned with data storage, which remains internal to each server, so that the portability of client code is improved. Servers are not concerned with the user interface or user state, so that servers can be simpler and more scalable.
  - Independent evolution
    - Servers and clients may also be replaced and developed independently, as long as the interface is not altered.



Conceptual Overview

REST Architectural Constrains (2)

- Stateless communication
  - Scalability, reliability
    - No client context being stored on the server between requests. Each request from any client contains all of the information necessary to service the request.
  - Resources are conversationally stateless
    - Any conversational state is held in the client.

- Uniform Interface
  - Simplicity (vs. efficiency)
  - Large-grained hypermedia data transfer
  - Example: Create, Retrieve, Update, Delete

Conceptual Overview

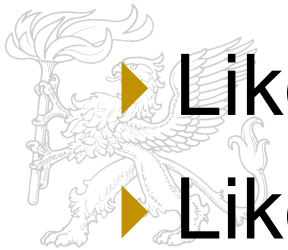
REST Architectural Constrains (3)

- Caching
  - Efficiency, scalability
    - Well-managed caching partially or completely eliminates some client-server interactions, further improving scalability and performance.
  - Consistency issues
    - As on the World Wide Web, clients are able to cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable or not, to prevent clients reusing stale or inappropriate data in response to further requests.
- Code-on-demand
  - Extending client functionality
    - Servers are able to temporarily extend or customize the functionality of a client by transferring to it logic that it can execute. Examples of this may include compiled components such as Java applets and client-side scripts such as JavaScript.

Conceptual Overview

RESTful Web Service definition

- A RESTful Web service is:
  - A set of Web resources.
  - Interlinked.
  - Data-centric, not functionality-centric.
  - Machine-oriented.

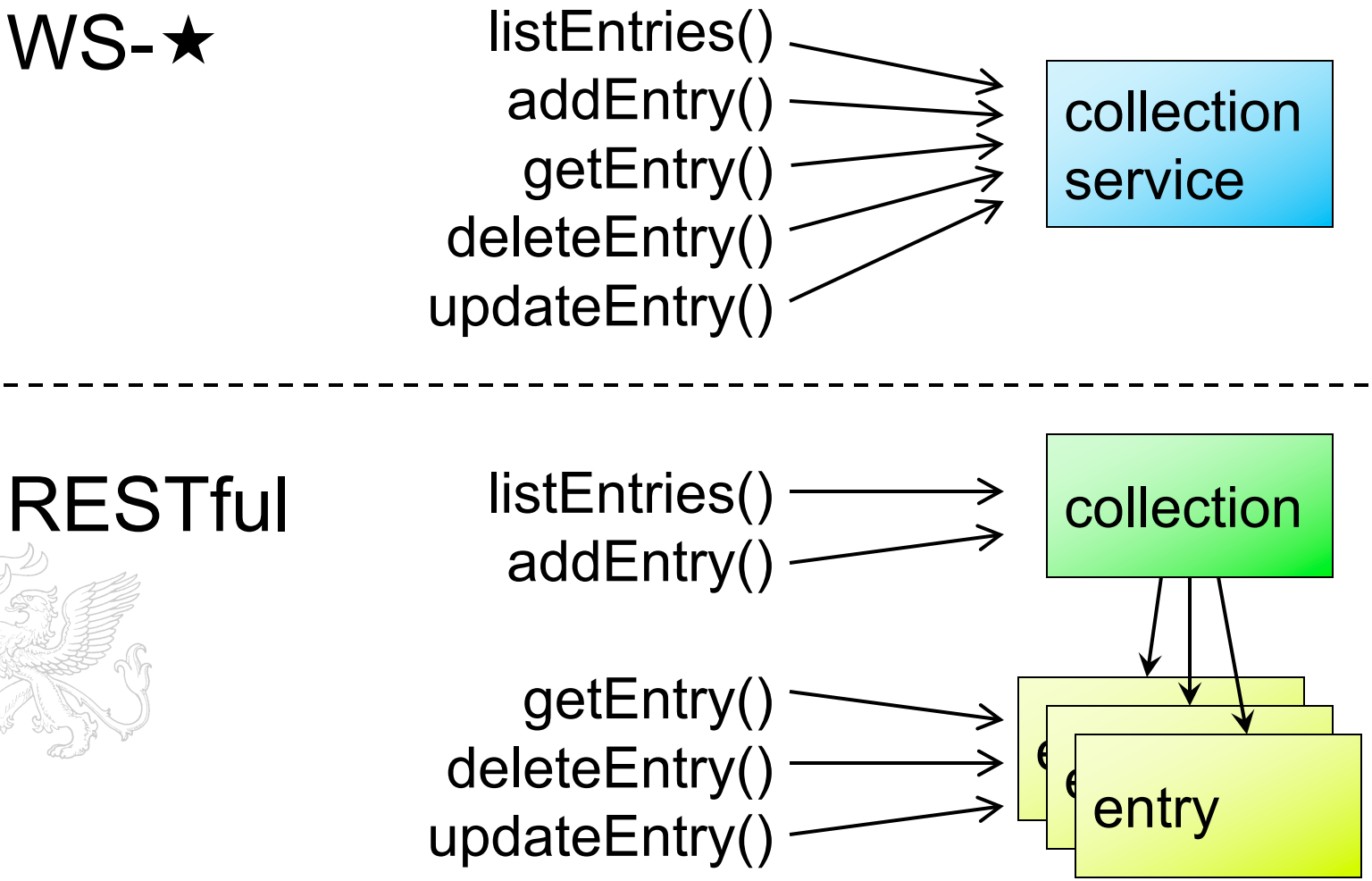


- Like Web applications, but for machines.
- Like WS-*, but with more Web resources.

WS-* stands for a variety of specifications related to SOAP-based Web Services.

Conceptual Overview

WS-★ vs REST: A quick comparison



- A SOAP service (WS-★) has a single endpoint that handles all the operations – therefore it has to have an application-specific interface.

- A RESTful service has a number of resources (the collection, each entry), so the operations can be distributed onto the resources and mapped to a small uniform set of operations.

Technologies

- Todays's set of technologies, protocols and languages used to apply RESTful paradigm:
  - HTTP as the basis
  - XML and JSON for data exchange
  - AJAX for client-side programming (e.g. browser)

- There exists an attempt to develop WSDL-like definition language for describing RESTful services
  - Web Application Description Language (WADL)

HTTP

Overview

- Hypertext Transfer Protocol (HTTP)
  - A protocol for distributed, collaborative, hypermedia information systems.
  - A request/response standard typical of client-server computing.
  - Currently dominant version is `HTTP/1.1`.
- Massively used to deliver content over the Web
  - Web browsers and spiders are relying on HTTP.
- The protocol is not constrained to TPC/IP
  - It only presumes a reliable transport.
- Resources accessed by HTTP are identified by URIs (more specifically URLs), using the `http` URI schemes.

HTTP

Request-response format

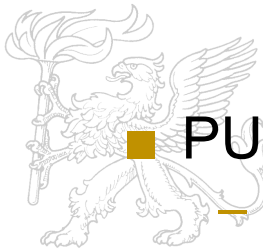
- Request consists of
  - Request line, such as `GET /images/logo.gif HTTP/1.1`, which requests a resource called `/images/logo.gif` from server.
  - Headers, such as `Accept-Language: en`
  - An empty line
  - An optional message body

- Response consists of
  - Status line which includes `numeric status code` and `textual reason phrase`
  - Response headers
  - An empty line
  - The requested content

HTTP

Request methods

- HTTP request methods indicate the desired action to be performed on the identified resource:
  - GET
    - Requests a representation of the specified resource. GET should not be used for operations that cause side-effects (problematic with robots and crawlers). Those operations are called safe operations.
  - POST
    - Submits data to be processed (e.g., from an HTML form) to the identified resource. The data is included in the body of the request.
  - PUT
    - Uploads a representation of the specified resource.
  - DELETE
    - Deletes the specified resource.



XML

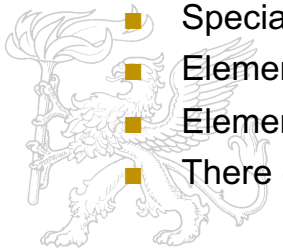
Overview

- eXtensible Markup Language (XML)
  - A set of rules for encoding documents electronically.
  - De-facto standard (W3C Recommendation).
- Ubiquitous presence on the Web and the Semantic Web
  - Storage and transportation of data (RDF/XML and SOAP),
  - Visualization of data (XHTML),
  - Application configuration (XML configuration files), etc.
- As such it can not be avoided as a possible data format for Web 2.0 Web Services.

XML

Characteristics

- As opposed to JSON XML can be verified against a schema expressed in a number of languages such as Document Type Definition (DTD), and XML Schema:
  - the vocabulary (element and attribute names),
  - the content model (relationships and structure), and
  - the data types.
- Founded on the standards laying in the core of Web
  - Uniform Resource Identifiers (URI)
  - Unicode
- Well-formedness an XML document
  - Properly encoded legal Unicode characters,
  - Special syntax characters such as < and & are used only as markup delineation,
  - Element tags are correctly nested,
  - Element tags are case sensitive,
  - There exists a single “root” element.



XML

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<Person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumber type="home">212 555-1234</phoneNumber>
  <phoneNumber type="fax">646 555-4567</phoneNumber>
  <newSubscription>false</newSubscription>
  <companyName />
</Person>
```

JSON

Overview

- JavaScript Object Notation (JSON)
  - A lightweight computer data interchange format.
  - Specified in Request For Comment (RFC) 4627.
- Represents a simple alternative to XML
  - A text-based, human-readable format for representing simple data structures and associative arrays (called objects).
- Used by a growing number of services
- JavaScript-friendly notation
  - Its main application is in Ajax Web application programming.
- A serialized object or array
- No namespaces, attributes etc.
- No schema language (for description, verification)

JSON

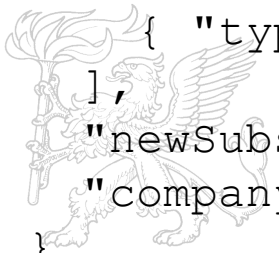
Data types

- JSON basic data types are
  - Number (integer, real, or floating point)
  - String (double-quoted Unicode with backslash escaping)
  - Boolean (true and false)
  - Array (an ordered sequence of values, comma-separated and enclosed in square brackets)
  - Object (collection of key:value pairs, comma-separated and enclosed in curly braces)
  - null

JSON

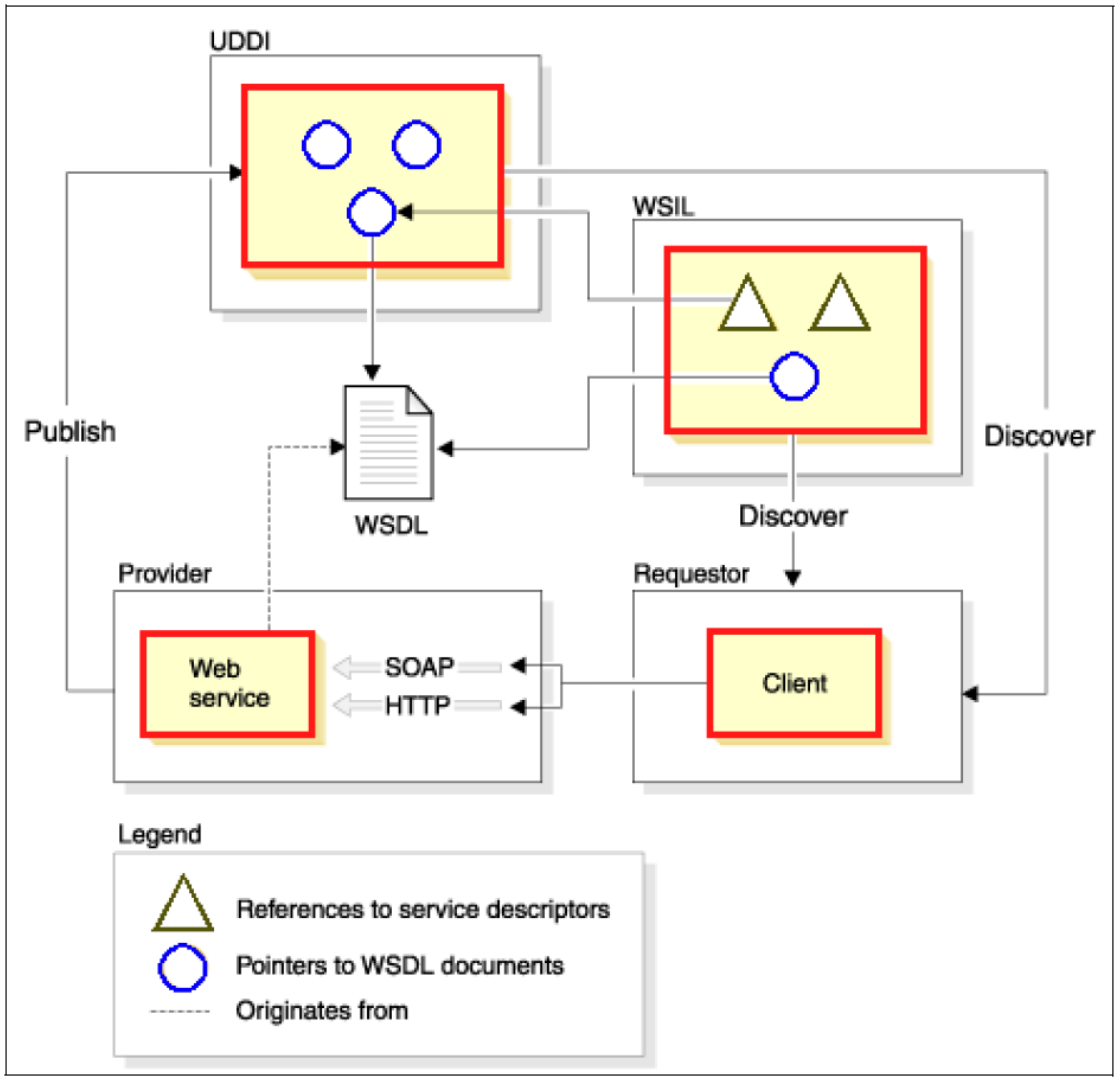
Example

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumbers": [
    { "type": "home", "number": "212 555-1234" },
    { "type": "fax", "number": "646 555-4567" }
  ],
  "newSubscription": false,
  "companyName": null
}
```

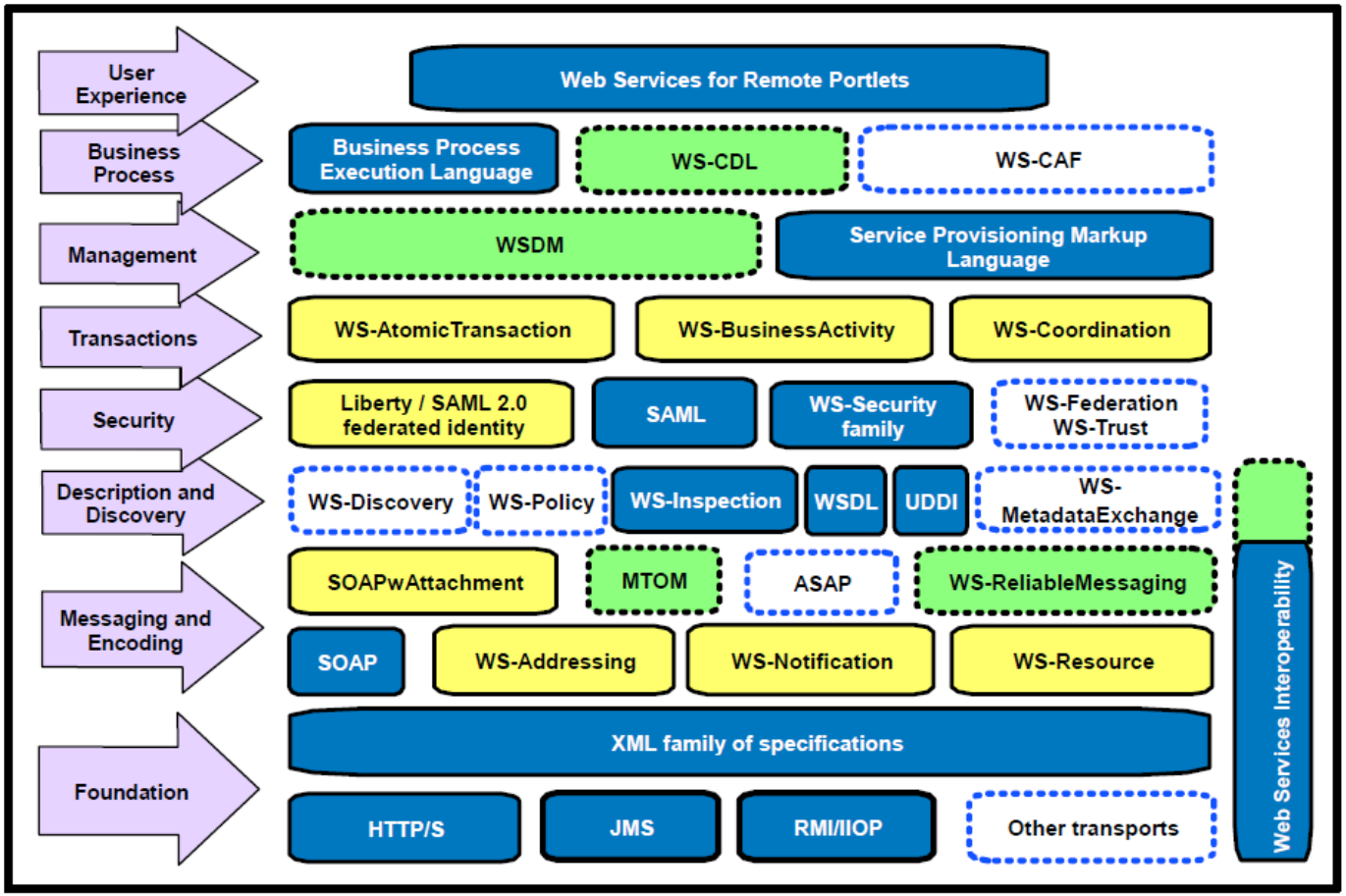


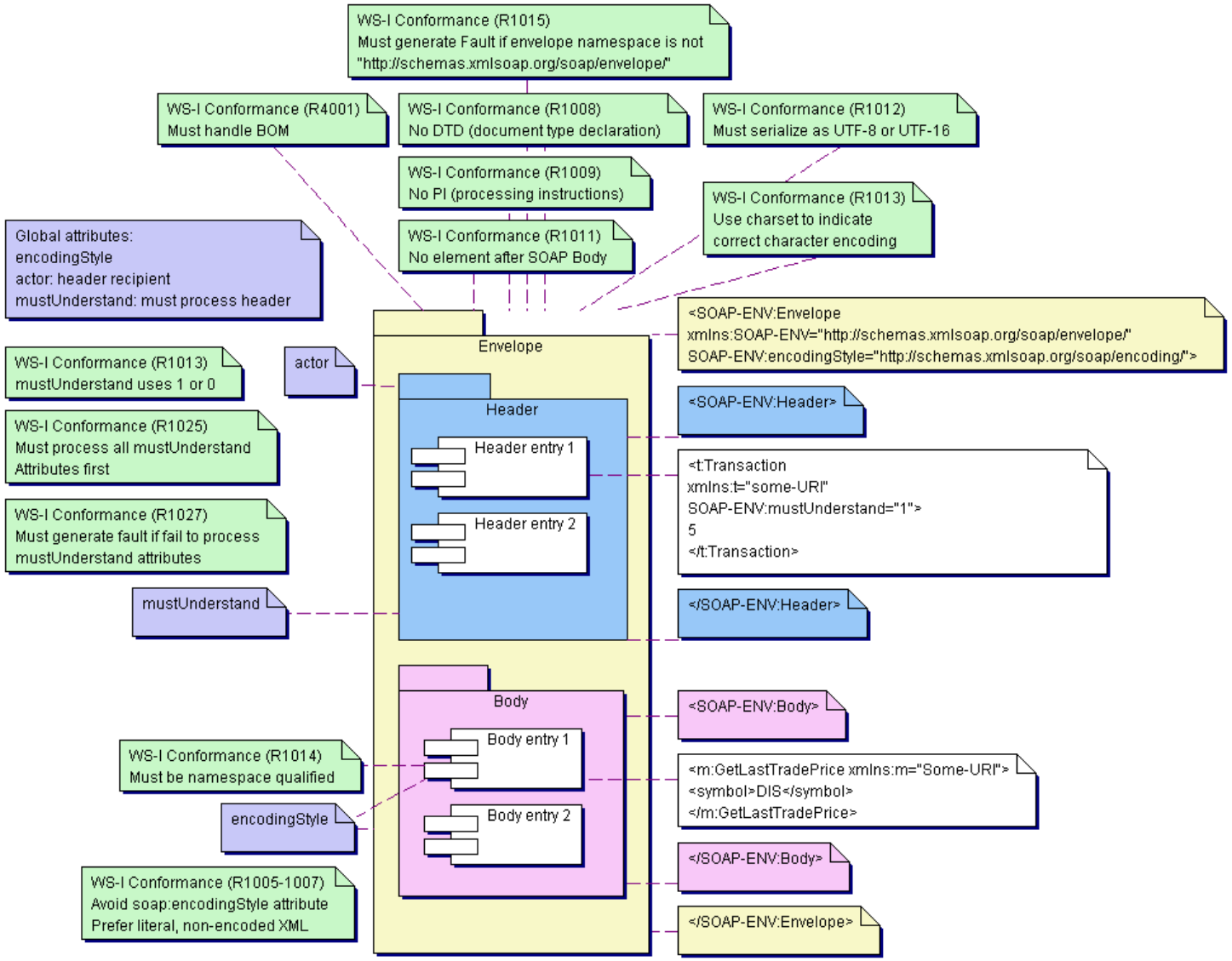
Elements of SOA

- XML
- SOAP
- WSDL
- WSIL
- UDDI



Web Services




WS-I Conformance (R1015)
Must generate Fault if envelope namespace is not "http://schemas.xmlsoap.org/soap/envelope/"
WS-I Conformance (R4001)
Must handle BOM
WS-I Conformance (R1008)
No DTD (document type declaration)
WS-I Conformance (R1012)
Must serialize as UTF-8 or UTF-16
WS-I Conformance (R1009)
No PI (processing instructions)
WS-I Conformance (R1013)
Use charset to indicate correct character encoding
WS-I Conformance (R1011)
No element after SOAP Body
Global attributes:
encodingStyle
actor: header recipient
mustUnderstand: must process header
Envelope
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
WS-I Conformance (R1013)
mustUnderstand uses 1 or 0
actor
Header
<SOAP-ENV:Header>
WS-I Conformance (R1025)
Must process all mustUnderstand Attributes first
Header entry 1
<t:Transaction
xmlns:t="some-URI"
SOAP-ENV:mustUnderstand="1">
5
</t:Transaction>
Header entry 2
WS-I Conformance (R1027)
Must generate fault if fail to process mustUnderstand attributes
mustUnderstand
</SOAP-ENV:Header>
Body
<SOAP-ENV:Body>
WS-I Conformance (R1014)
Must be namespace qualified
Body entry 1
<m:GetLastTradePrice xmlns:m="Some-URI">
<symbol>DIS</symbol>
</m:GetLastTradePrice>
encodingStyle
Body entry 2
</SOAP-ENV:Body>
WS-I Conformance (R1005-1007)
Avoid soap:encodingStyle attribute
Prefer literal, non-encoded XML
</SOAP-ENV:Envelope>

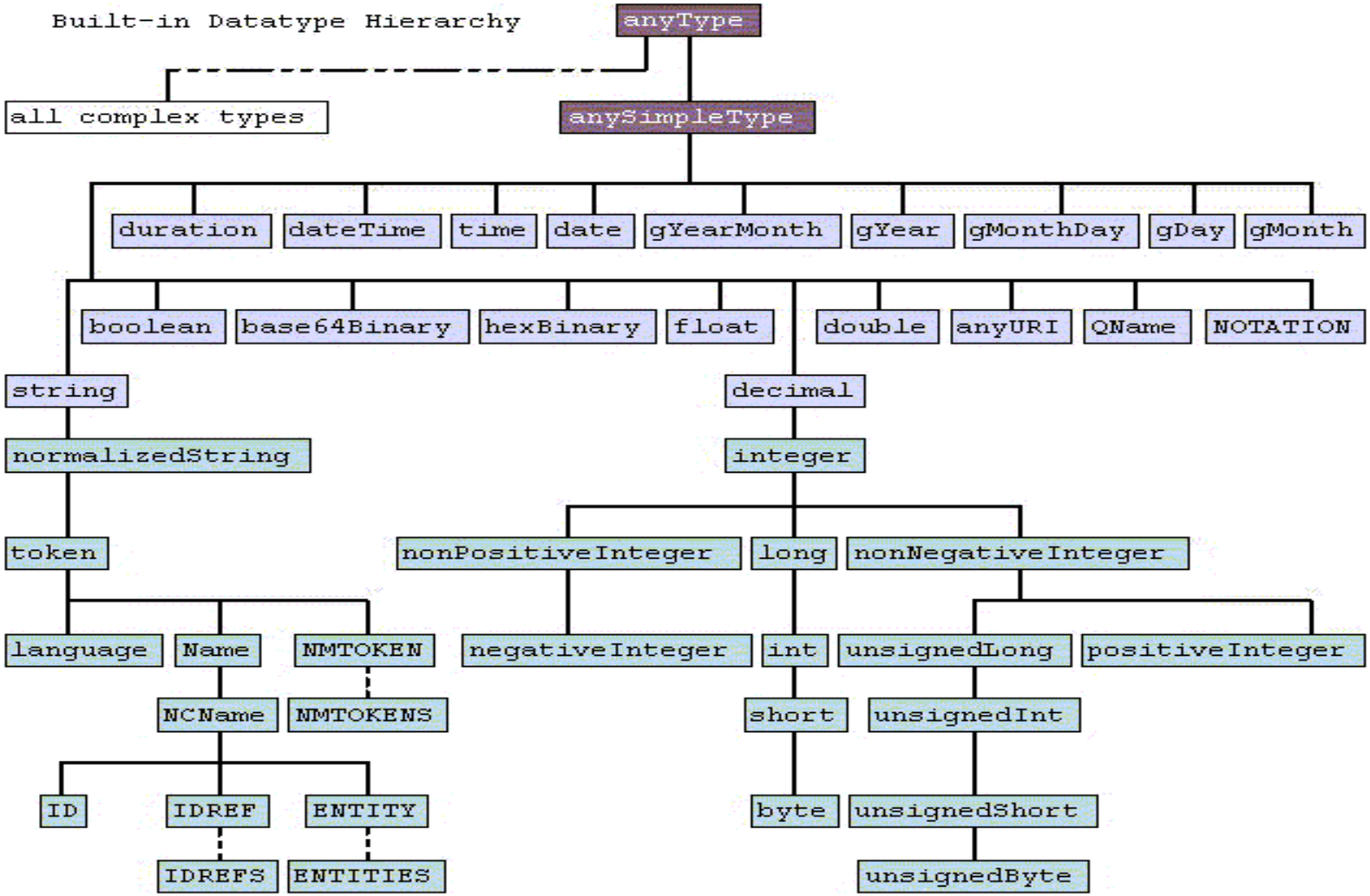
WSDL Parts

- Types
  - Used to define custom message types
- Messages
  - Abstraction of request and response messages that my client and service need to communicate.
- PortTypes
  - Contains a set of operations.
  - Operations organize WSDL messages.
  - Operation->method name, portType->java interface
- Bindings
  - Binds the portType to a specific protocol (typically SOAP over http).
  - You can bind one portType to several different protocols by using more than one port.
- Services
  - Gives you one or more URLs for the service.
  - Go here to execute “echo”.

Namespaces

- The WSDL document begins with several XML namespace definitions.
- Namespaces allow you to compose a single XML document from several XML schemas.
- Namespaces allow you to identify which schema an XML tag comes from.
  - Avoids name conflicts.

Schema Built In Types



WSDL Messages

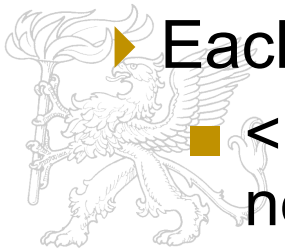
- The “message” section specifies communications that will go on between endpoints.
  - Gives each message a name (to be used later for reference).
  - Specifies the type of message
    - Can be primitive types, like strings
    - Can be defined types, as we saw previously.

The echoServiceInterface messages

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions>
  <wsdl:types />
<wsdl:message name="echoResponse">
    <wsdl:part name="echoReturn" type="xsd:string" />
</wsdl:message>
<wsdl:message name="echoRequest">
    <wsdl:part name="in0" type="xsd:string" />
</wsdl:message>
<wsdl:portType name="Echo">
    <wsdl:operation name="echo" parameterOrder="in0">
      <wsdl:input message="impl:echoRequest" name="echoRequest" />
      <wsdl:output message="impl:echoResponse"
    name="echoResponse" />
    </wsdl:operation>
  </wsdl:portType>
  …
</wsdl:definitions>
```

Structure of a Message

- WSDL <message> elements have name attributes and one or more *parts*.
  - The message name should be unique for the document.
  - <operation> elements will refer to messages by name.
- I need one <part> for each piece of data I need to send in that message.
- Each <part> is given a name and specifies its type.
  - <part> types can point to <wsdl:type> definitions if necessary.



WSDL portTypes

- WSDL messages are only abstract messages.
  - We bind them to *operations* within the portType.
- The structure of the portType specifies (still abstractly) how the messages are to be used.
  - Think of operations->java methods and portTypes->java interfaces.

The echoServiceInterface portType

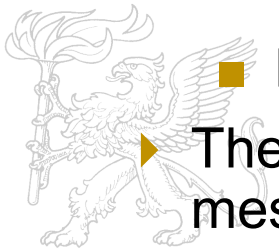
```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions>
  <wsdl:types />
<wsdl:message name="echoResponse">
    <wsdl:part name="echoReturn" type="xsd:string" />
  </wsdl:message>
    <wsdl:message name="echoRequest">
    <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:portType name="Echo">
    <wsdl:operation name="echo" parameterOrder="in0">
      <wsdl:input message="impl:echoRequest" name="echoRequest"
    />
      <wsdl:output message="impl:echoResponse"
    name="echoResponse" />
    </wsdl:operation>
  </wsdl:portType>
…
</wsdl:definition>
```

EchoService portType

```
<wsdl:portType name="Echo">
    <wsdl:operation name="echo" parameterOrder="in0">
      <wsdl:input
      message="impl:echoRequest"        name="echoRequest" />
      <wsdl:output
      message="impl:echoResponse"       name="echoResponse"
    />
      </wsdl:operation>
</wsdl:portType>
```

portType Message Patterns

- PortTypes support four types of messaging:
  - One way: Client send a message to the service and doesn't want a response.
    - <input> only.
  - Request-Response: Client sends a message and waits for a response.
    - <input>, then <output>
  - Solicit-Response: Service sends a message to the client first, then the client responds.
    - <output>, then <input>
  - Notification: <output> only.
- These still are abstract. We must implement them using some message protocol.
  - HTTP units of transmission are request and response, so mapping Solicit-Response to HTTP will take some work.



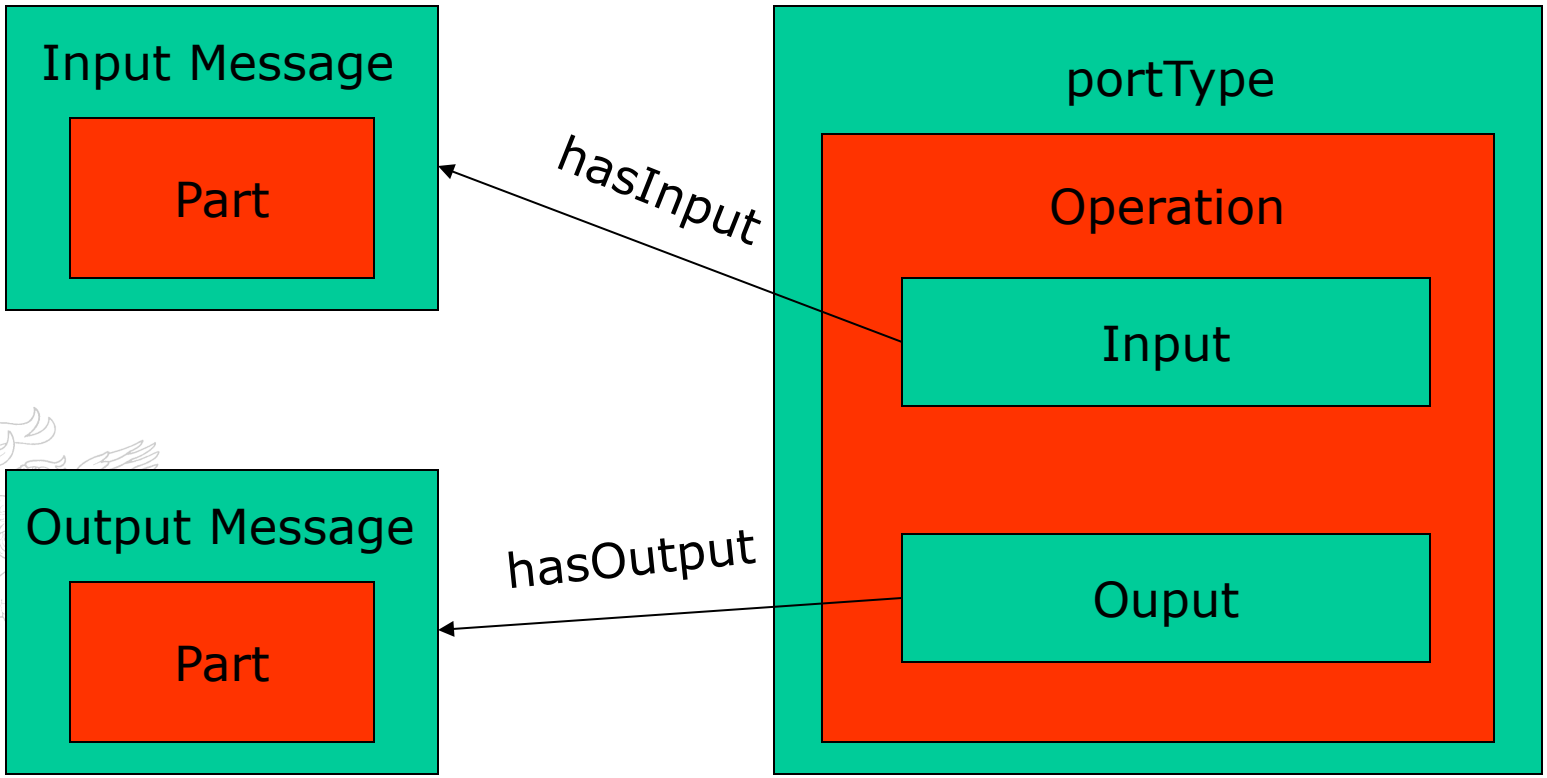
portType for EchoService

- The echo service has one method, echo.
- It takes one string argument and returns one string.
- In WSDL, the portType is “Echo”, the operation is “echo”.
- The messages are organized into input and output.
  - Messages are placed here as appropriate.
  - That is, <input> takes the <echoRequest> message.

Parameter Order

- This attribute of operation is used to specify zero or more space-separated values.
- The values give the order that the input messages must be sent.

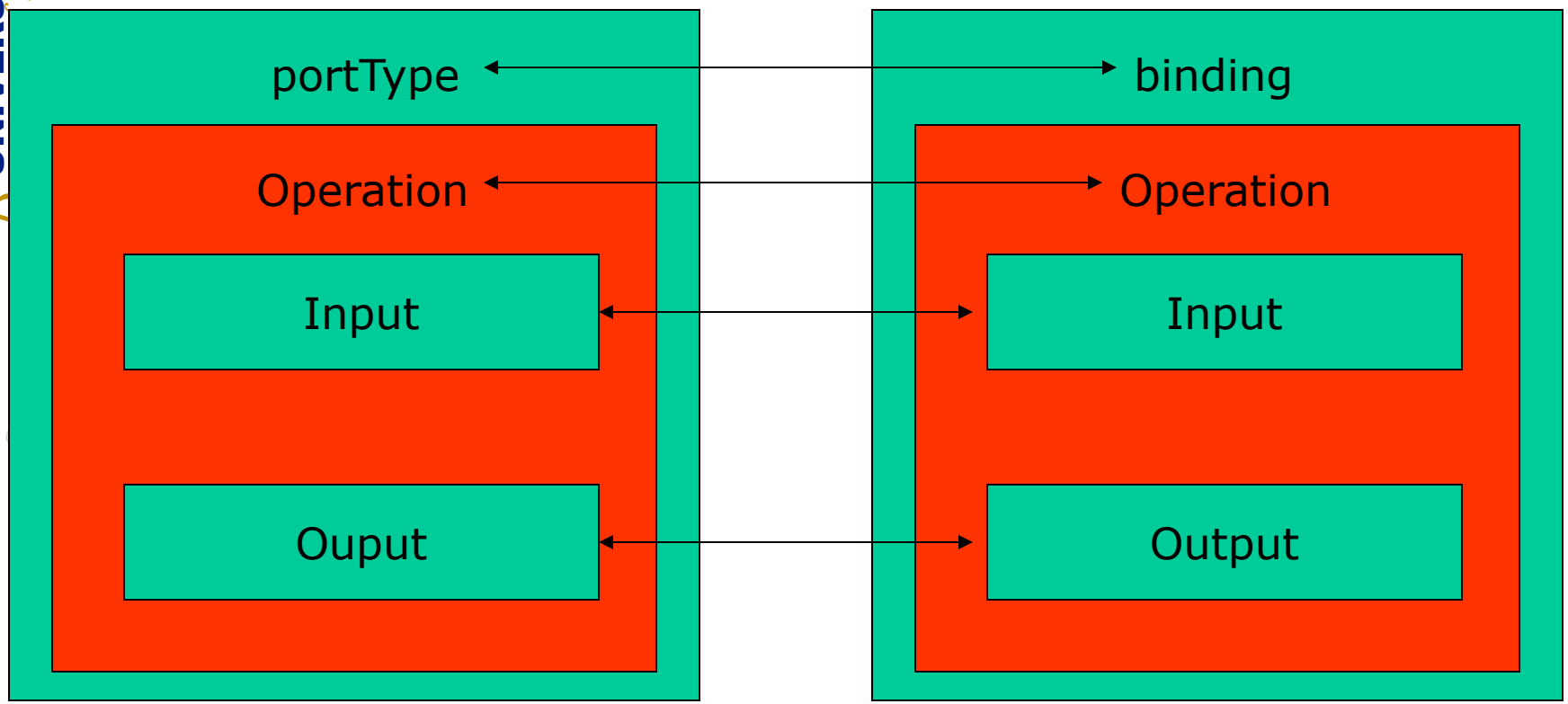
The Picture So Far…



Binding tags

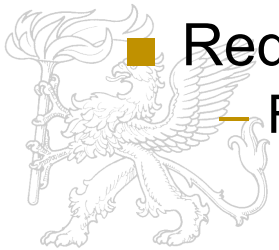
- Binding tags are meant to bind the parts of portTypes to sections of specific protocols.
  - SOAP, HTTP GET/POST, and MIME are provided in the WSDL specification.
- Bindings refer back to portTypes by name, just as operations point to messages.
  - They are mirror images of the portTypes.
  - Each part is extended by schema elements for a particular binding protocol (i.e. SOAP).
- In WSDL bindings messages:
  - Each corresponds to SOAP body sections, described later.
  - Additionally, we specify that the body should be encoded.
    - That is, RPC encoded.
    - Alternatively, could also be “literal” (or “document”).

WSDL Internal References

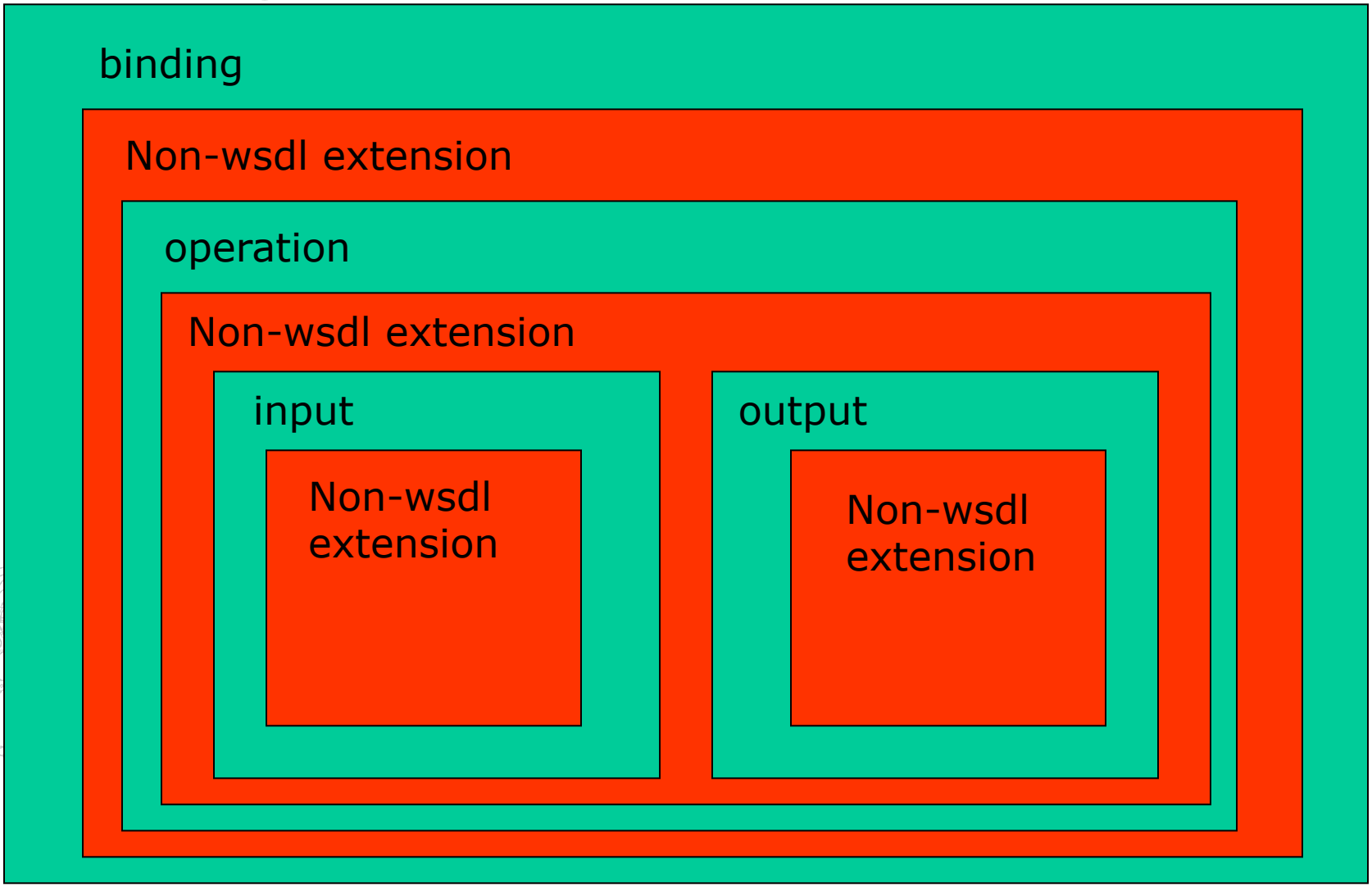


Structure of the Binding

- <binding> tags are really just placeholders.
- They are meant to be extended at specific places by wsdl protocol bindings.
  - These protocol binding rules are defined in supplemental schemas.
- The following box figure summarizes these things
  - Green boxes are part of WSDL
    - From the wsdl namespace, that is.
  - Red boxes are parts of the document from other schemas
    - From wsdlsoap namespace in the echo example.

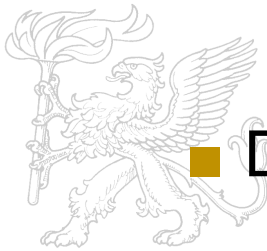


Binding Structure

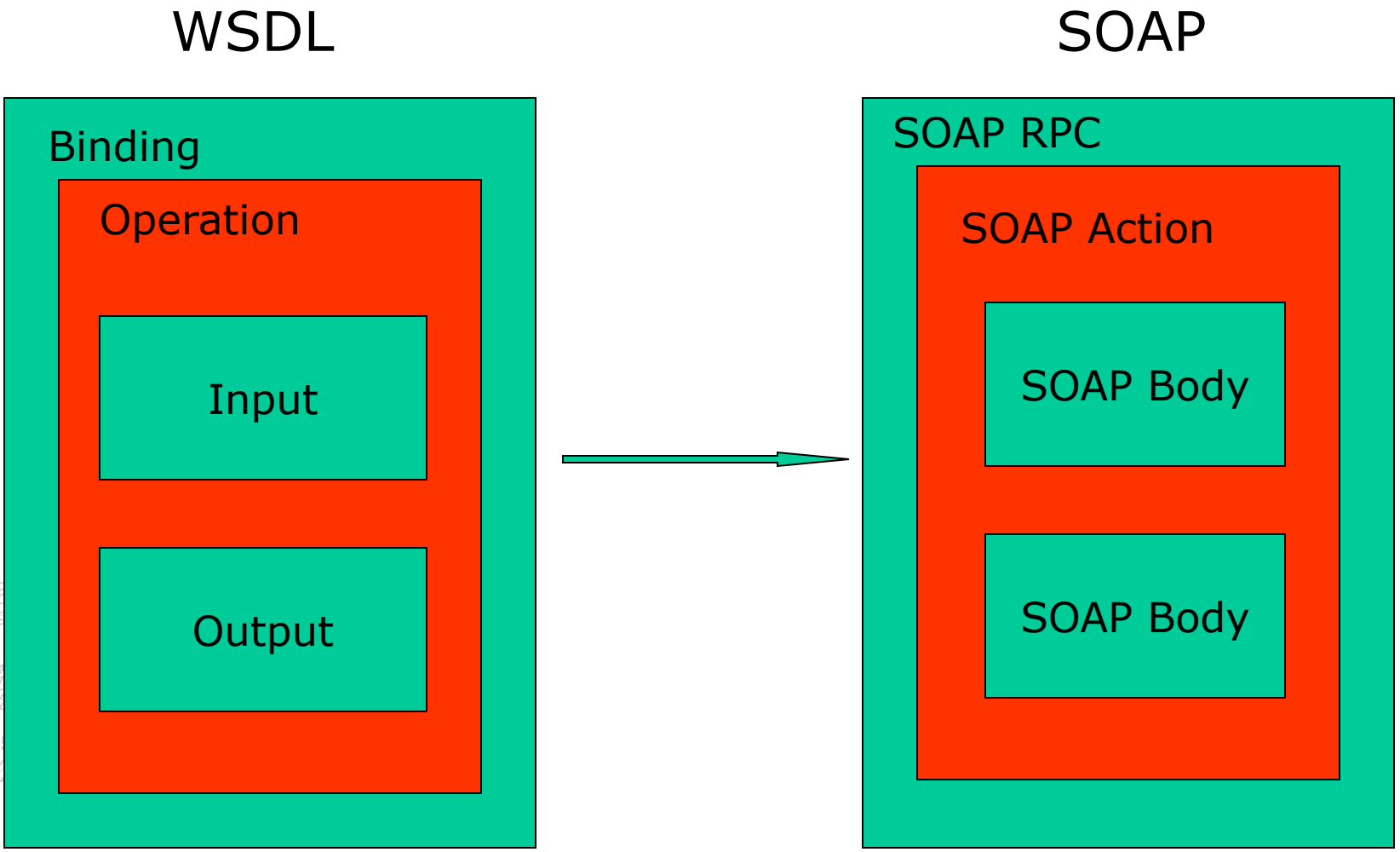


A little more on encoding...

- We specify SOAP encoding
- SOAP is a message format and needs a transport protocol, so we specify HTTP.
- Operation styles may be either “RPC” or “Document”.
- SOAP Body elements will be used to actually convey message payloads.
  - RPC requires “encoded” payloads.
    - Each value (echo strings) is wrapped in an element named after the operation.
    - Useful RPC processing on the server side.
  - Documents are literal (unencoded)
    - Use to just send a payload of XML inside SOAP.



Binding Associations to SOAP



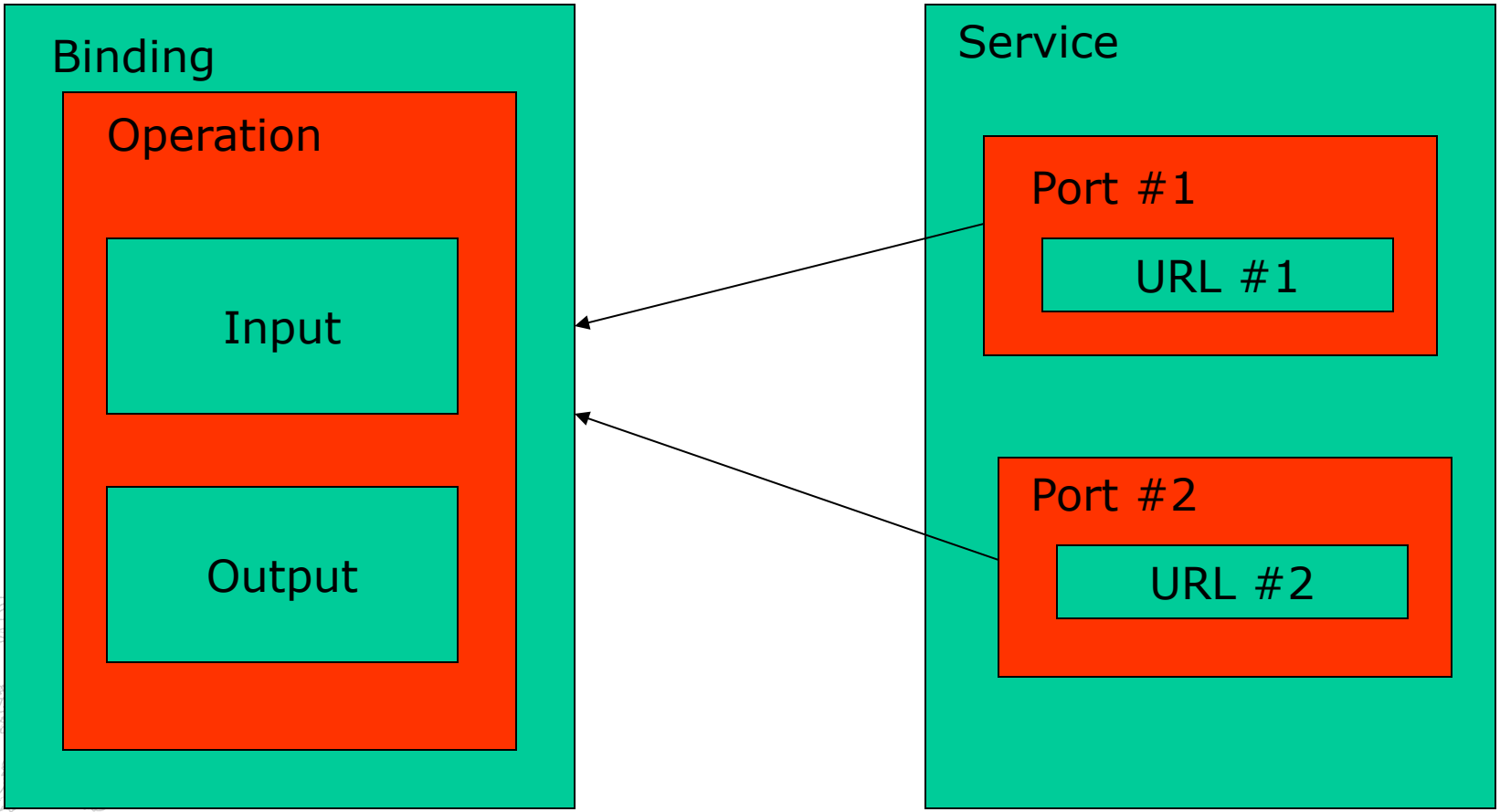
Binding Restrictions

- Binding elements point by name to portTypes.
- WSDL allows more than one binding element to point to the same port type.
  - Why?
  - Because a service may support multiple, alternative protocol bindings.

Port and Service Tags

- The service element is a collection of ports.
  - That's all it is for.
- Ports are intended to point to actual Web service locations
  - The location depends on the binding.
  - For SOAP bindings, this is a URL.

Port Associations to Bindings

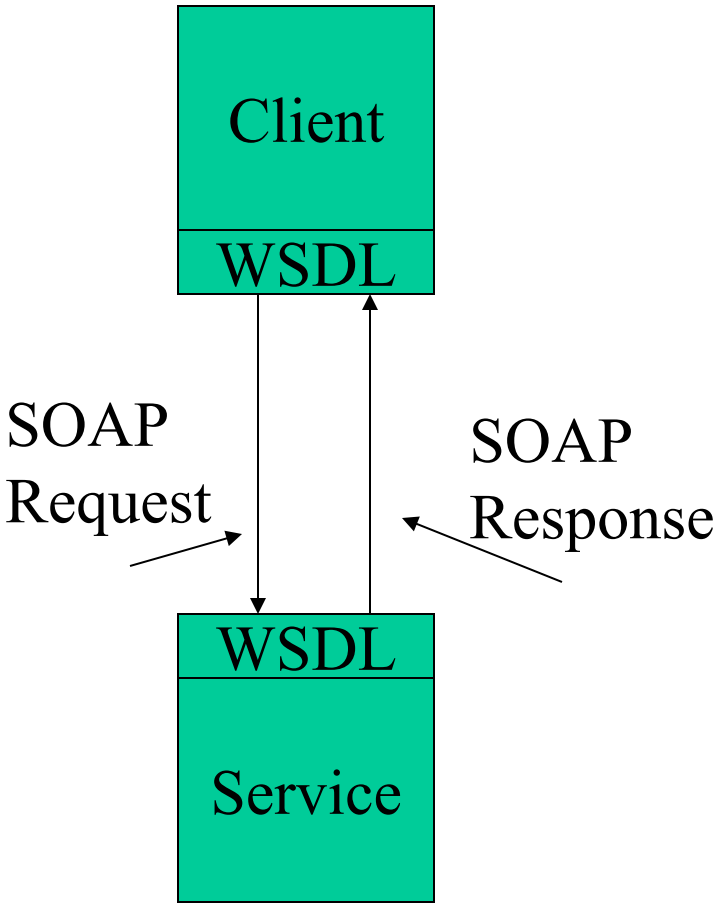


Summary of WSDL

- WSDL decouples remote service operations.
  - Types=custom message definitions.
    - Any data types not in the XML schema.
  - Message=name the messages that must be exchanged and their data types, possibly defined by <type>.
  - PortTypes=service interfaces
    - Operations=remote method signatures.
  - Bindings=mappings of portType operations to real message formats
  - Ports=locations (URLs) of real services.

SOAP and Web Services

- WSDL
  - Defines the interfaces for remote services.
  - Provides guidelines for constructing clients to the service.
  - Tells the client how to communicate with the service.
- The actual communications are encoded with SOAP.
  - Transported by HTTP



SOAP in One Slide

- SOAP is just a message format.
  - Must transport with HTTP, TCP, etc.
- SOAP is independent of but can be connected to WSDL.
- SOAP provides rules for processing the message as it passes through multiple steps.
- SOAP payloads
  - SOAP carries arbitrary XML payloads as a body.
  - SOAP headers contain any additional information
  - These are encoded using optional conventions

SOAP Basics

- SOAP is often thought of as a protocol extension for doing **Remote Procedure Calls** (RPC) over HTTP.
  - This is how it is often used.
- This is not accurate: **SOAP is an XML message format** for exchanging structured, typed data.
  - It may be used for RPC in client-server applications
  - May be used to send XML documents
  - Also suitable for messaging systems (like JMS) that follow one-to-many (or publish-subscribe) models.
- **SOAP is not a transport protocol**. You must attach your message to a transport mechanism like HTTP.

SOAP Request

```
<?xml version='1.0' ?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:echo
          soapenv:encodingStyle="http://schemas.xmlsoap.org/soa
p/encoding/"
          xmlns:ns1="http://.../axis/services/EchoService">
        <in0 xsi:type="xsd:string">Hollow World</in0>
    </ns1:echo>
  </soapenv:Body>
</soapenv:Envelope>
```

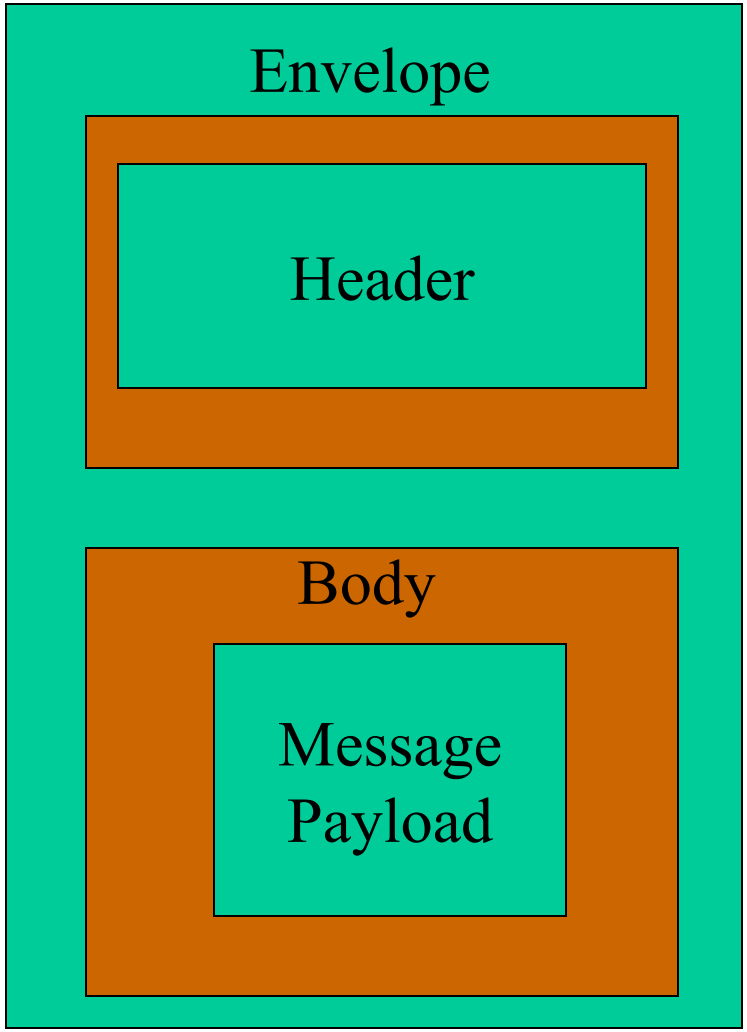
SOAP Response

```
<?xml version='1.0' ?>
<soapenv:Envelope
        xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
        xmlns:xsd=http://www.w3.org/2001/XMLSchema
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
   <ns1:echoResponse

soapenv:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
       xmlns:ns1="http://../axis/services/echoService">
      <echoReturn xsi:type="String">
        Hollow World
      </echoReturn>
   </ns1:echoResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP Structure

- SOAP structure is very simple.
  - 0 or 1 header elements
  - 1 body element
  - Envelop that wraps it all.
- Body contains XML payload.
- Headers are structured the same way.
  - Can contain additional payloads of “metadata”
  - Security information, quality of service, etc.



SOAP Envelop

- The envelop is the root container of the SOAP message.
- Things to put in the envelop:
  - Namespaces you will need.
    - **http://schemas.xmlsoap.org/soap/envelope** is required, so that the recipient knows it has gotten a SOAP message.
    - Others as necessary
  - Encoding rules (optional)
    - Specific rules for deserializing the encoded SOAP data.
    - More later on this.
- Header and body elements.
  - Headers are optional, body is mandatory.
  - Headers come first in the message, but we will look at the body first.

SOAP Headers

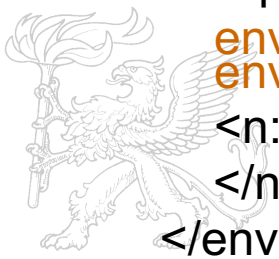
- SOAP Body elements contain the primary message contents.
- Headers are really just **extension points** where you can include elements from other namespaces.
  - i.e., headers can contain arbitrary XML.
- Headers may be processed independently of the body.
- Headers may optionally define **encodingStyle.**
- Headers may optionally have a "**role**" attribute
- Header entries may optionally have a "**mustUnderstand**" attribute.
  - mustUnderstand=1 means the message recipient must process the header element.
  - If mustUnderstand=0 or is missing, the header element is optional.
- Headers may also have a "**relay**" attribute.

Example Uses of Headers

- **Security**: WS-Security and SAML place additional security information (like digital signatures and public keys) in the header.
- **Quality of Service:** SOAP headers can be used if we want to negotiate particular qualities of service such as reliable message delivery and transactions.
- **Session State Support:** Many services require several steps and so will require maintenance of session state.
  - Equivalent to cookies in HTTP.
  - Put session identifier in the header.

Example Header from SOAP Primer

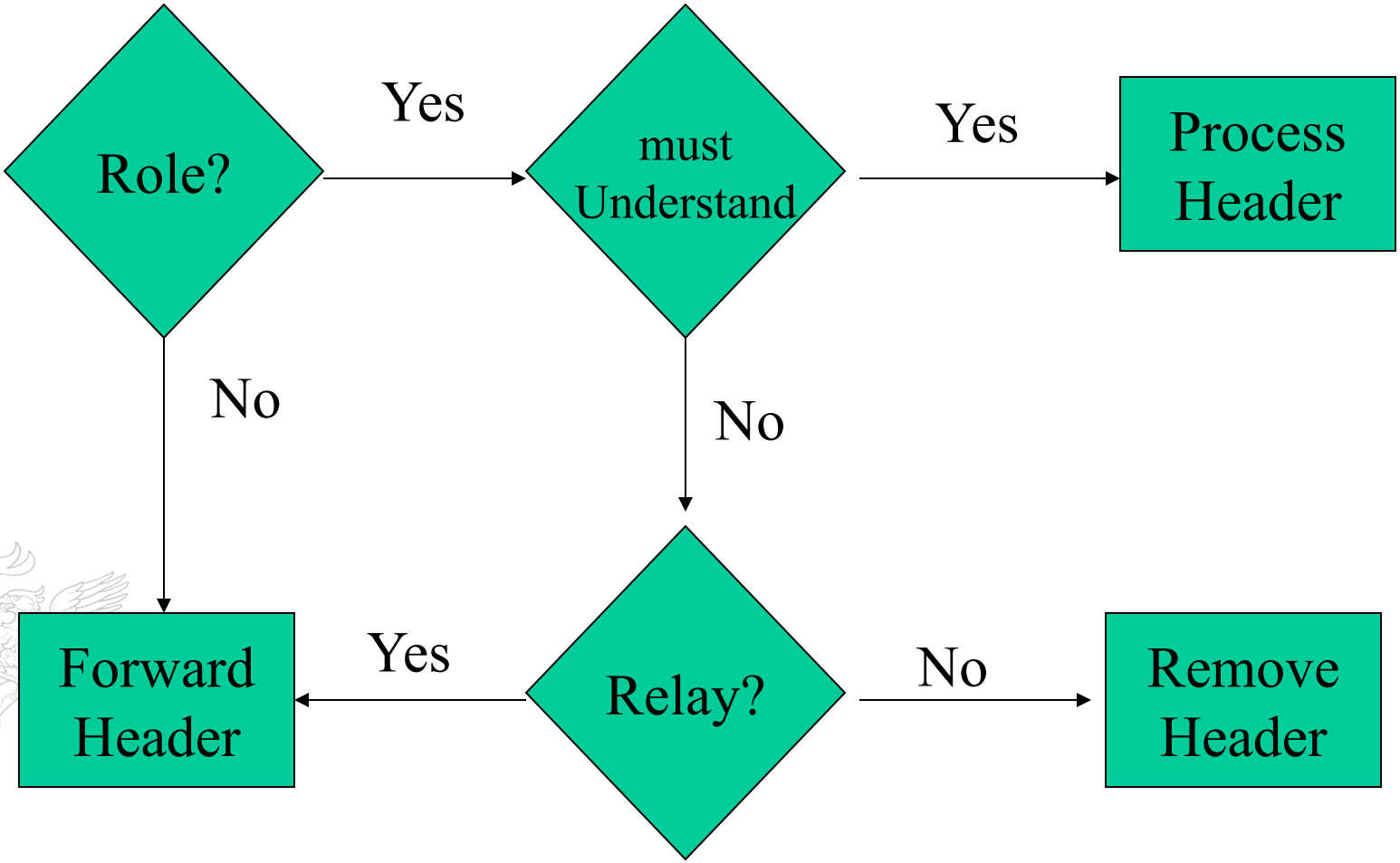
```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://my.example.com/"
     env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
     env:mustUnderstand="true">
       <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d
       </m:reference>
       <m:dateAndTime>2001-11-29T13:20:00.000-05:00
      </m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="…"
     env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
     env:mustUnderstand="true">
      <n:name>Åke Jógvan Øyvind</n:name>
      </n:passenger>
  </env:Header>
```



Header Processing

- SOAP messages are allowed to pass through many intermediaries before reaching their destination.
  - Intermediary=some unspecified routing application.
  - Imagine SOAP messages being passed through many distinct nodes.
  - The final destination processes the body of the message.
- Headers are allowed to be processed independently of the body.
  - May be processed by intermediaries.
- This allows an intermediary application to determine if it can process the body, provide the required security, session, or reliability requirements, etc.

Roles, Understanding, and Relays



Header Roles

- SOAP nodes may be assigned role designations.
- SOAP headers then specify which role or roles should process.
- Standard SOAP roles:
  - **None**: SOAP nodes MUST NOT act in this role.
  - **Next:** Each SOAP intermediary and the ultimate SOAP receiver MUST act in this role.
  - **UltimateReceiver:** The ultimate receiver MUST act in this role.
- In our example, all nodes must process the header entries.

SOAP Body

- Body entries are really just placeholders for XML from some other namespace.
- The body contains the XML message that you are transmitting.
- It may also define encodingStyle, just as the envelop.
- The message format is not specified by SOAP.
  - The <Body></Body> tag pairs are just a way to notify the recipient that the actual XML message is contained therein.
  - The recipient decides what to do with the message.

SOAP Body Element Definition

```
<xs:element name="Body" type="tns:Body" />
<xs:complexType name="Body">
    <xs:sequence>
      <xs:any namespace="##any"
     processContents="lax" minOccurs="0“
   maxOccurs="unbounded" />
    </xs:sequence>
     <xs:anyAttribute namespace="##other"
   processContents="lax" />
</xs:complexType>
```

SOAP Body Example

```
<soapenv:Body>
  <ns1:echo soapenv:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1=
    "http://.../axis/services/EchoService">
       <in0 xsi:type="xsd:string">Hollow
    World</in0>
  </ns1:echo>
 </soapenv:Body.
```

WS architecture