universität wien

Web Services

Faculty of Computer Science
Workflow Systems and Technologies

Interoperability

Distributed Systems

- Various software agents *work together to accomplish some tasks*.
- These agents *do not* necessarily *operate in the same computing environment* - communication must occur over the network.
- Architectural challenges of distributed systems include e.g.:
  - Lack of shared memory between caller and object
  - Concurrent access to remote resources
  - Latency and unreliability caused by underlying transport
  - Issues due to partial failures
  - Issues due to incompatible updates introduced to participants

Web Services

*"... are **self-contained**, **modular** business applications that have **open**, **internet-oriented**, **standards-based interfaces** ... communicate directly with other Web services via standards-based technologies"*
(UDDI Consortium)

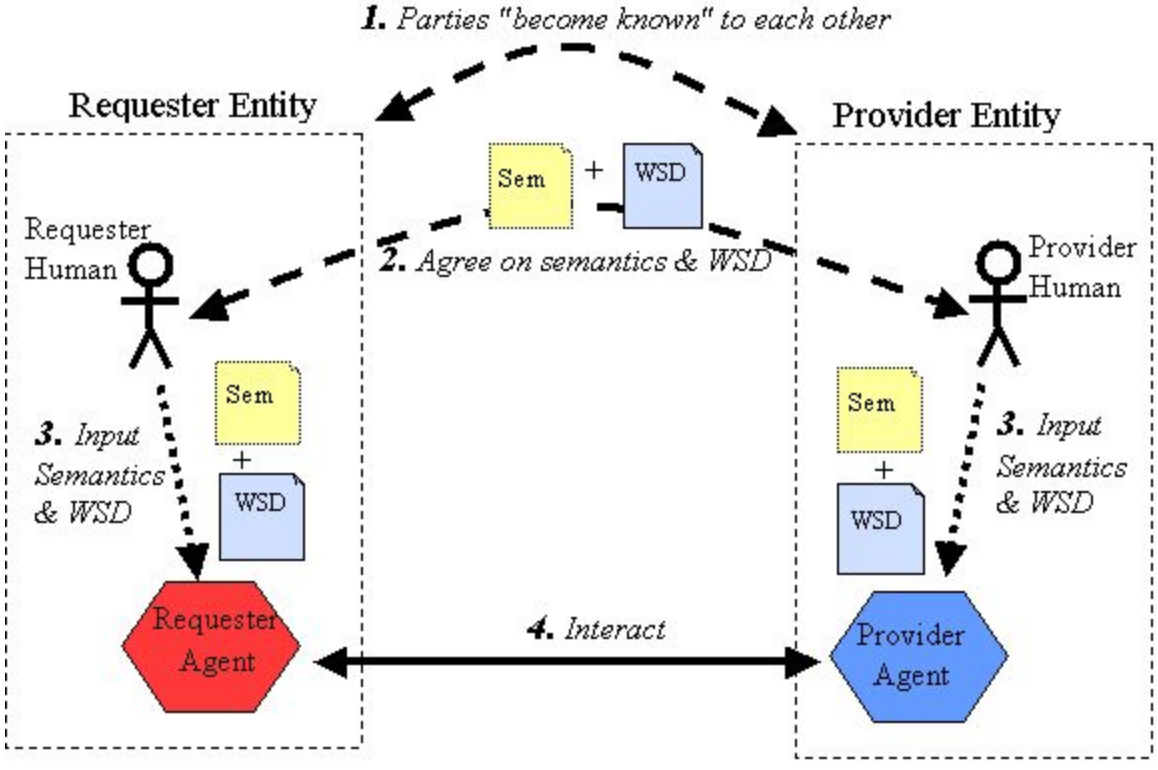
*"... is a software system designed to support **interoperable machine-to-machine interaction** over a network. It has an **interface** described **in a machine-processable format** ... Other **systems interact** with the Web service **in a manner prescribed by its description** ... "*
(W3C Consortium)

Web Service Architecture (W3C)

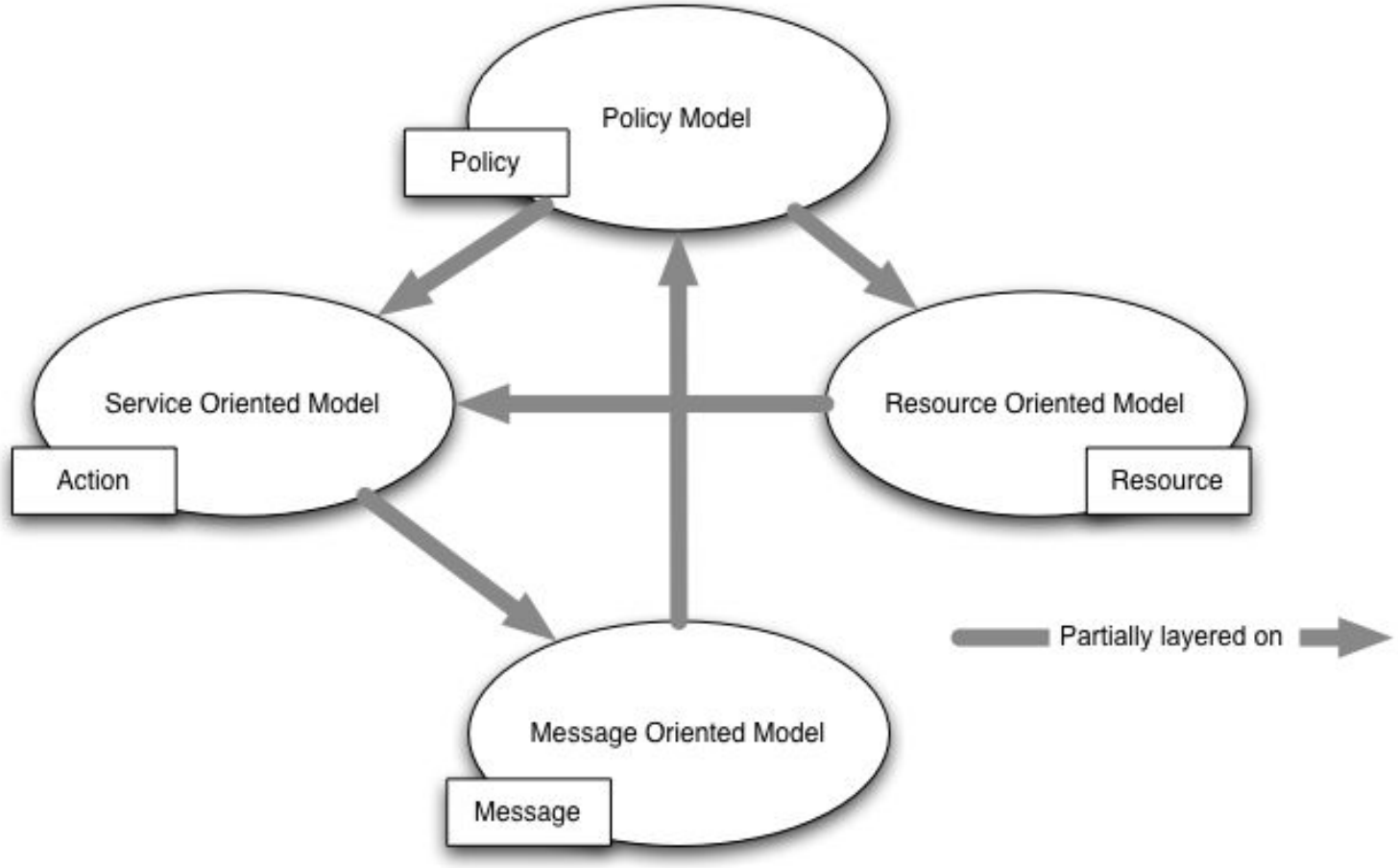
Collection of required *concepts* that enable interoperability between web services:

- ***Service*** Abstract definition of a web service - implementation agnostic
- **Agent** Concrete piece of software that implements a service which send and receives messages.
- **Provider** Person or organization that provides a service via an appropriate agent.
- **Requestor** Person or organization that consumes a provider's service
- **Service Description** Defines the mechanics of the message exchange between two parties.
- **Semantics** Shared expectation about the behavior of the service. It can be a formal or informal agreement - expresses a form of 'contract'.

Web Service Interaction

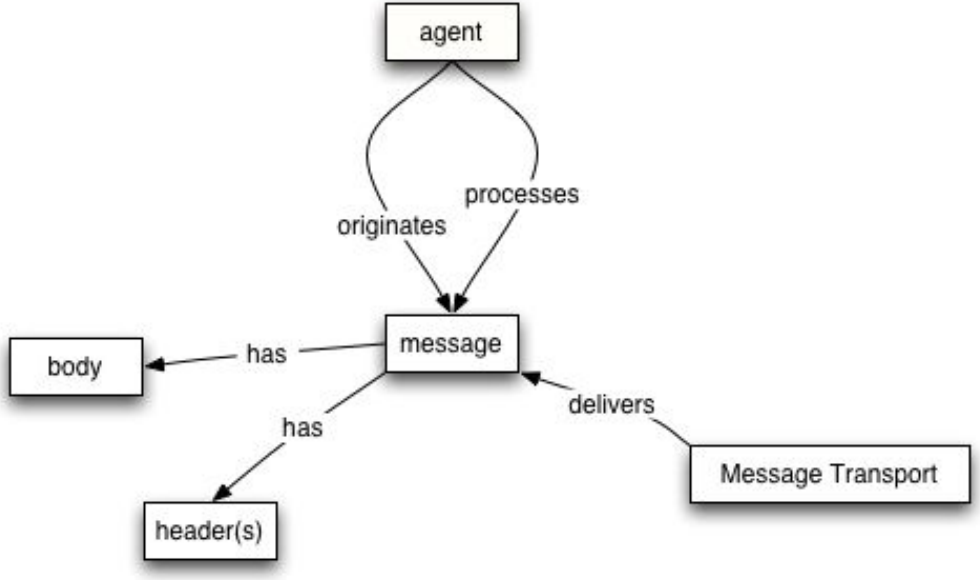


Architectural Models of Web Services



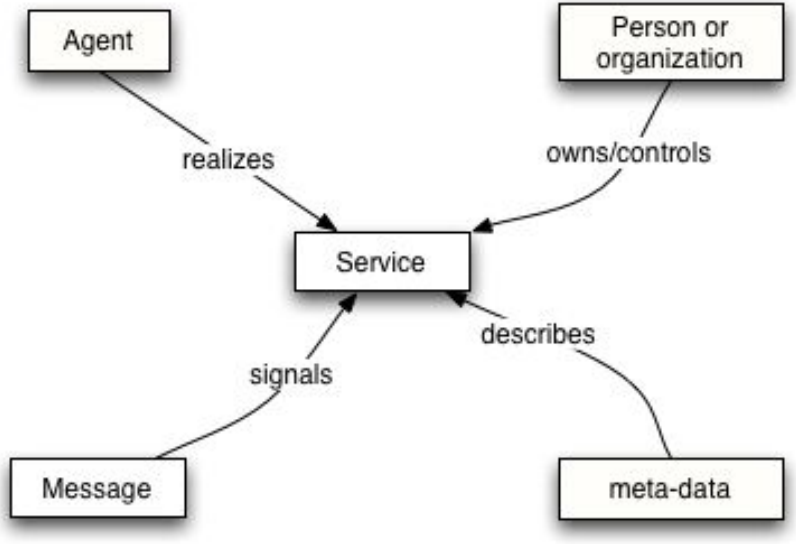
Architectural Models - Message Oriented Model

- Revolves around messages,
- the *structure of messages* with regards to the message headers and bodies,
- and around the *delivery mechanisms* for messages.



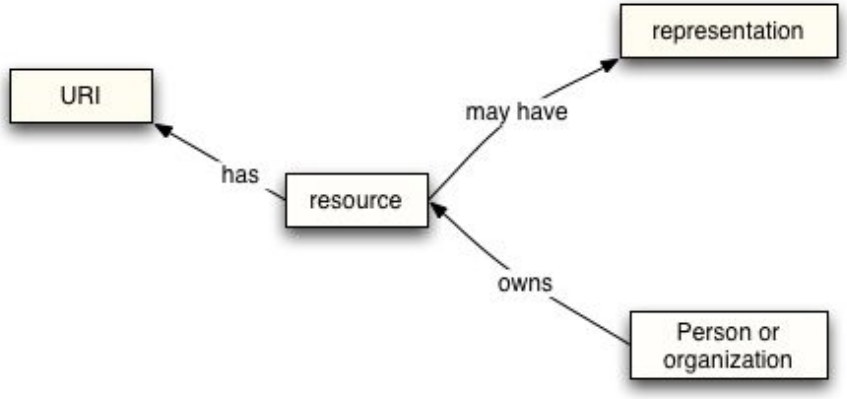
Architectural Models - Service Oriented Model

- Revolves around the concept of services.
- *Meta-data* is an integral part to *document* several aspects of services including interface and transport binding details, semantics and policy restrictions; key for the deployment and use of services.
- *Ownership* expresses the notion of *responsibility* for the provided functionality.



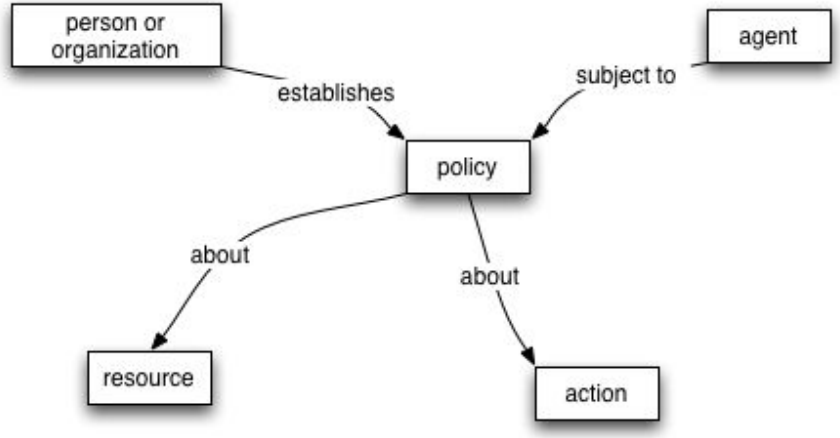
Architectural Models - Resource Oriented Model

- Revolves around the notion of a *resource* similar to the resource concept in the Web Architecture, i.e., *each URI identifies one resource* (e.g., web pages, images, multimedia-files etc.).
- Representations *reflect the state* of resources, however a representation mustn't be the same as the resource.



Architectural Models - Policy Model

- *Policies* are concerned with resources and are enacted to represent security, quality of service, management and application concerns.
- Emphasis on *constraints* on the *behaviour of agents* (and services).
- Constraints are imposed on agents by the entity responsible for the resource.



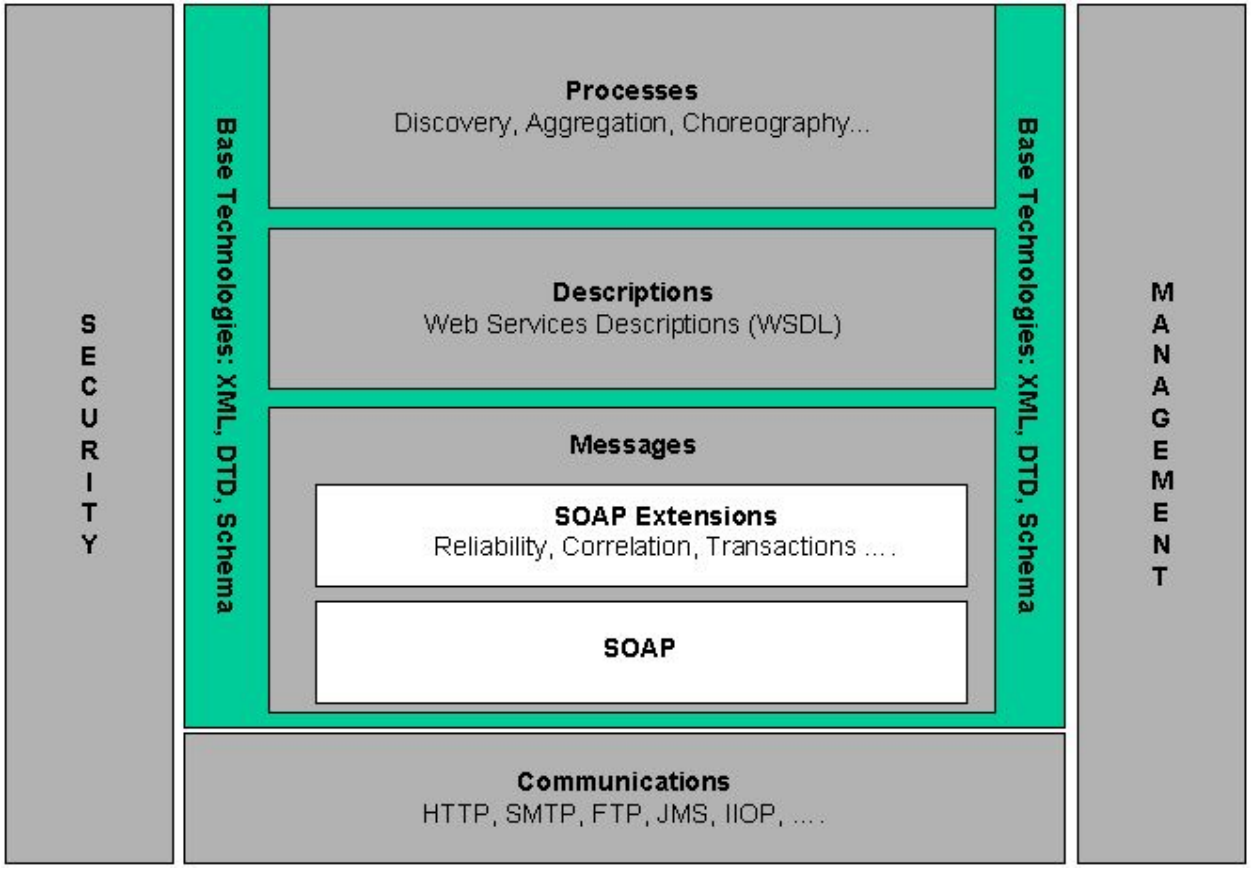
Service Oriented Architecture (SOA)

SOA is one form of distributed systems architecture that is characterized by following properties:

- **Logical View** Services are defined by *what they do*.
- **Message orientation** Services are defined by the messages exchanged - implementation details of the agents must be abstracted away.
- **Description orientation** Description of services by machine-processable meta-data. Semantics should also be included in the description.
- **Granularity** Tendency towards coarse-grained services and thus larger and complex messages.
- **Network orientation** Use of services foremost occurs over the network - is not an absolute requirement.
- **Platform neutrality** Messages are exchanged in platform-neutral and standardised format via interfaces.

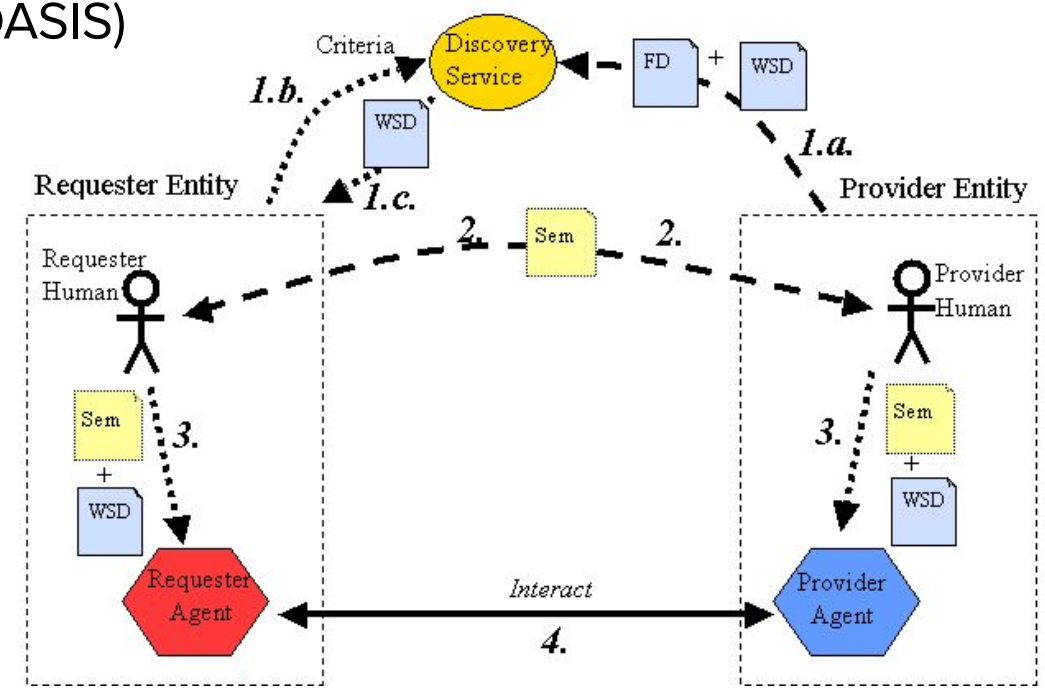
Web Service Perspectives

(Classic) Web Service Architecture Stack



Web Service Discovery

- Location of service (descriptions) one wants to engage with.
- Service location approaches
  - Registry (e.g. UDDI by OASIS)
  - Index
  - Peer-to-Peer



Web Service Semantics

- A successful interaction between systems demands a shared agreement about ***form***, ***structure*** *and* ***meaning*** of messages.
- This shared agreement governs the ***visibility*** of the *message semantics*.
- Use of *standards* facilitates acquiring insights about the *intent* of messages through the inspection of the flow of messages and their content.
    - E.g. SOAP defines the format and structure of the header and bodies of the messages.
    - Meaning expressed via meta-data (e.g. OWL, RDF)

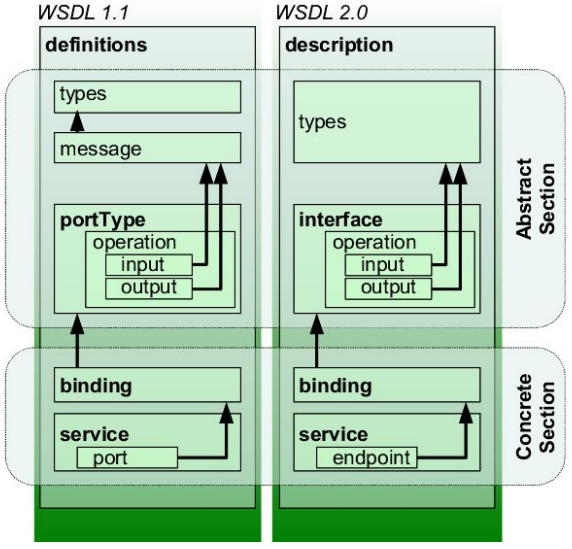
Additional Perspectives

- *Web Services Security* aims at securing components involved in the point-to-point communication (e.g. transport encoding via SOAP, schema validation, message integrity checks, message confidentiality via encryption etc.)
- *Web Services Reliability* revolves around reliable and predictable delivery of messages and interactions of services.
- *Web Service Management* focuses on enabling monitoring, controlling, and reporting of service qualities and usage.

Web Service Description (Language)

Web Services Description Language (WSDL)

- General purpose, platform independent description language for web services in the context of distributed systems.
- Aims to facilitate remote invocation of services solely with the help of the machine processable description.
- Specifies *operations* and *messages* as abstract services.
- *Binds* services to a concrete network protocol (e.g. HTTP) and message format (e.g. SOAP) to define an endpoint.
- WSDL Version 1.1 and 2.0 are W3C standards.



Building Blocks of WSDL. From (Cristcost, 2007).

SOAP

- Represents a general purpose messaging framework.
- Enables exchange of structured information for automatic processing.
- Version 1.1 and 1.2 are W3C standards.
- Messages are wrapped in an Envelope consisting of an optional SOAP Header and a mandatory SOAP Body.
- Various encoding styles include RPC/literal, RPC/encoded and Document/literal. See detailed discussion here.
- Literal style relies on the custom types defined in a WSDL document.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1="urn:ec.europa.eu:taxud:vies:services:checkVat:types">
    <SOAP-ENV:Body>
        <ns1:checkVat>
            <ns1:countryCode>AT</ns1:countryCode>
            <ns1:vatNumber>U37586901</ns1:vatNumber>
        </ns1:checkVat>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example: VIES Service with WSDL and SOAP

- VIES is a service provided by the EU to verify the validity of a VAT number issued by any Member State.
- It provides a WSDL document for the service.

```
<?php //Run on almighty.cs.univie.ac.at via wwwlab.cs.univie.at/~a<student_id>/<vies>.php

$client = new SoapClient("https://ec.europa.eu/taxation_customs/vies/checkVatService.wsdl",
                         array('trace' => 1) ); // Enables us to view the last SOAP request/response

$res =$client->checkVat(
  array("countryCode" => "AT",
        "vatNumber" => "U37586901"));


// To be view the response in the browser in a readable form...
header('content-type: text/plain');

echo "\n\n== Request SOAP Envelope\n";
print_r($client->__getLastRequest());

echo "\n\n== Response SOAP Envelope\n";
print_r($client->__getLastResponse());

echo "\n\n== Response (unmarshalled)\n";
print_r($res);
?>
```

Example: VIES Service with WSDL and SOAP

== Request SOAP Envelope

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1="urn:ec.europa.eu:taxud:vies:services:checkVat:types">
    <SOAP-ENV:Body>
        <ns1:checkVat>
            <ns1:countryCode>AT</ns1:countryCode>
            <ns1:vatNumber>U37586901</ns1:vatNumber>
        </ns1:checkVat>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

== Response SOAP Envelope

```
<env:Envelope
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
    <env:Header/>
    <env:Body>
        <ns2:checkVatResponse
            xmlns:ns2="urn:ec.europa.eu:taxud:vies:services:checkVat:types">
            <ns2:countryCode>AT</ns2:countryCode>
            <ns2:vatNumber>U37586901</ns2:vatNumber>
            <ns2:requestDate>2023-05-04+02:00</ns2:requestDate>
            <ns2:valid>true</ns2:valid>
            <ns2:name>Universität Wien</ns2:name>
            <ns2:address>Universitätsring 1
                         AT-1010 Wien
            </ns2:address>
        </ns2:checkVatResponse>
    </env:Body>
</env:Envelope>
```

WSDL Example with a PHP Client and Server (2/2)

```
<definitions
  name="IOP_WSDL_EXAMPLE"
  targetNamespace="http://interop.wsd"
  xmlns:tns="http://interop.wsd"
  xmlns:iops="http://interop.wsd/xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <schema targetNamespace="http://interop.wsd/xsd"
      xmlns="http://www.w3.org/2000/10/XMLSchema">
        <complexType name="station">
          <sequence>
              <element name="system" type="xsd:string"/>
              <element name="temperature" type="xsd:string"/>
          </sequence>
        </complexType>
    </schema>
  </types>

  <message name="getSensorReadingsRequest">
    <part name="station" type="xsd:string"/>
    <part name="unit" type="xsd:string"/>
  </message>

  <message name="getSensorReadingsResponse">
    <part name="station" element="iops:station"/>
  </message>

  <portType name="WeatherStationPort">
    <operation name="getSensorReadings">
      <input message="tns:getSensorReadingsRequest"/>
      <output message="tns:getSensorReadingsResponse"/>
    </operation>
  </portType>

  <binding name="WeatherStationsBinding" type="tns:WeatherStationPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getSensorReadings">
      <soap:operation soapAction="tns:WeatherService#getWeather"/>
      <input><soap:body  use="literal" /></input>
      <output><soap:body use="literal" /></output>
    </operation>
  </binding>

  <service name='WeatherService'>
    <port name="WeatherStationPort" binding="tns:WeatherStationsBinding">
      <soap:address location="https://wwwlab.cs.univie.ac.at/~.../.../server.php"/>
    </port>
  </service>
</definitions>
```

WSDL Example with a PHP Client and Server (1/2)

```php
<?php // CLIENT
// avoid caching of WSDL file during development
ini_set("soap.wsdl_cache_enabled","0");

// DANGEROUS! - Use next lines only for debugging purposes.
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);


$opts = array('trace'=>1,);
$client = new
SoapClient('https://wwwlab.cs.univie.ac.at/~.../weather.wsdl',
   $opts);

// Treat everything in the response as text from here on.
header('content-type: text/plain');

function pp_soapenvelope($client) {
   echo "==REQUEST\n";
   $doc = new DOMDocument('1.0');
   $doc->formatOutput = true;
   $doc->loadXML($client->__getLastRequest());
   print $doc->saveXML();

   echo "==RESPONSE\n";
   $doc = new DOMDocument('1.0');
   $doc->formatOutput = true;
   $doc->loadXML($client->__getLastResponse());
   print $doc->saveXML();
}


$result = $client->getSensorReadings('vienna','celsius');
pp_soapenvelope($client);
echo "==Runtime Object Deserialised from the SOAP Response\n";
print_r($result);
?>
```

```php
<?php // SERVER
  // Class which provides the implementation of the operations
  // definded in the WSDL.
  class WeatherStations {
    function getSensorReadings($station, $unit) {
      // 'Fetching' station and returning its sensor reading...
      $station = new StdClass;
      $station->system = 'X100';
      $station->temperature= '38.1C';

      return $station;
    }
    // more methods
  }


  // Testing above class.
  if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    $test = new WeatherStations;
    header('content-type: text/plain');
    print_r($test->getSensorReadings('london', 'kelvin'));
    exit;
  }

  // Handle requests of a WSDL-SOAP client.
  if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    ini_set("soap.wsdl_cache_enabled","0");
    $opts = [];
    $server = new
SoapServer('https://wwwlab.cs.univie.ac.at/~.../weather.wsdl', $opts);
    $server->setClass('WeatherStations');
    $server->handle();
    exit;
  }

?>
```

Web Application Description Language (WADL)

- Conceptually similar to WSDL
- Particular focus on describing HTTP

```
<resources base="http://example.com/">
  <resource path="widgets">
    <resource path="reports/stock">
      <param name="instockonly" style="matrix"
        type="xsd:boolean"/>
      ...
    </resource>
    <resource path="{widgetId}">
      ...
    </resource>
    ...
  </resource>
  <resource path="accounts/{accountId}">
    ...
  </resource>
</resources>
```

```
<method name="GET">
  <request>
    <param name="format" style="query">
      <option value="xml" mediaType="application/xml"/>
      <option value="json" mediaType="application/json"/>
    </param>
    ...
  </request>
  <response>
    <representation mediaType="application/xml"/>
    <representation mediaType="application/json"/>
  </response>
</method>
```

```
  <method name="POST" id="addImageCollectionMember">
    <request>
      <representation mediaType="image/*"/>
    </request>
    <response status="201">
      <param name="location" style="header" type="xsd:anyURI"
        required="true">
        <link resource_type="#entry" rel="self"/>
      </param>
      <representation href="#entry"/>
    </response>
  </method>
```

Examples from (Hadely, M., 2009)

Representational State Transfer

Representational State Transfer (REST)

- Represents an architectural style that promotes uniform-interface, hypermedia-driven and scalable API designs.
- Defined in *Architectural styles and the design of network-based software architectures* by Fielding, R. T., (2000).
- Collection of **architectural constraints** for the behaviour of **hypermedia**
- Interfaces that fulfil the REST constraints are considered *"RESTful"*
- Hypermedia as the engine of application state (**HATEOAS)**

Architectural Constraints

- *Client Server Model*
  - Enforces separation of concerns
  - Client and server can evolve independently
- *Stateless*
  - Induces visibility, reliability, and scalability
  - May cause network overhead
- Cache
  - Improved efficiency and scalability
  - At the cost of reliability as the cached state starts deviating from the current state over time

Architectural Constraints

- *Uniform interface*
  - Decoupling of services from implementation
  - Loss in efficiency due to the use of standardised non-application specific representations for transfer
  - Key concepts: *identification of resources* and *HATEOAS*
- Layered-System
  - Composition of hierarchical layers
  - Overhead due to latency
- Code-On-Demand
  - Extension of client capabilities on demand

Architectural Data Elements

Table 5-1: REST Data Elements

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Rest Data Elements Table from (Fielding, R. T, 2000)

HTTP GET Example

Request

```
GET /interop/attendees HTTP/1.1
Host: univie.interop.at
Accept: application/json, application/xml
```

Response

```
HTTP/1.1 200 OK
Content-Length: <???>
Content-Type: application/xml

<attendees/>
```

HTTP POST Example

Request

```
POST /interop/attendees HTTP/1.1
Host: univie.interop.at
Content-Type: application/x-www-form-urlencoded

attendee=Alice
```

Response

```
HTTP/1.1 201 OK
Location: /interop/attendee/256
```

Overview of HTTP Methods

Method Name	Description
GET	Transfer a current representation of the target resource.
HEAD	Same as GET, but do not transfer the response content.
POST	Perform resource-specific processing on the request content.
PUT	Replace all current representations of the target resource with the request content.
DELETE	Remove all current representations of the target resource.
CONNECT	Establish a tunnel to the server identified by the target resource.
OPTIONS	Describe the communication options for the target resource.
TRACE	Perform a message loop-back test along the path to the target resource.

https://www.rfc-editor.org/rfc/rfc9110#table-4

HTTP Methods

- HTTP methods describe the *primary semantics* of a request (see RFC 9110).
- Semantics provide a *uniform interface*, i.e. they are not tied to a specific resource. Headers can be applied for additional semantics (e.g. Accept).
- However, *a resource determines if those semantics are considered.*
- RFC 5789 extends set of methods with PATCH
  - Requests a set of changes to be applied on a resource
  - See the approaches in RFC 6902 and RFC 7386
- Methods can be classified into *safe* and/or *idempotent* methods.
  - *Safe* methods don't cause permanent state altering side effects
  - *Idempotent* methods when repeatedly executed result in the same states

HTTP Status Codes

- HTTP status codes (three-digit integer) describe the result and semantics of a response
- Usually a textual description of the status code is also provided
- Classes of status codes:

  1xx Informational: Request was received, continuing process

  2xx Successful: Request was received, understood and accepted

  3xx Redirect: Additional action required to complete the request

  4xx Client Error: Request is malformed or cannot be fulfilled

  5xx Server Error: Failed to fulfill an apparently valid

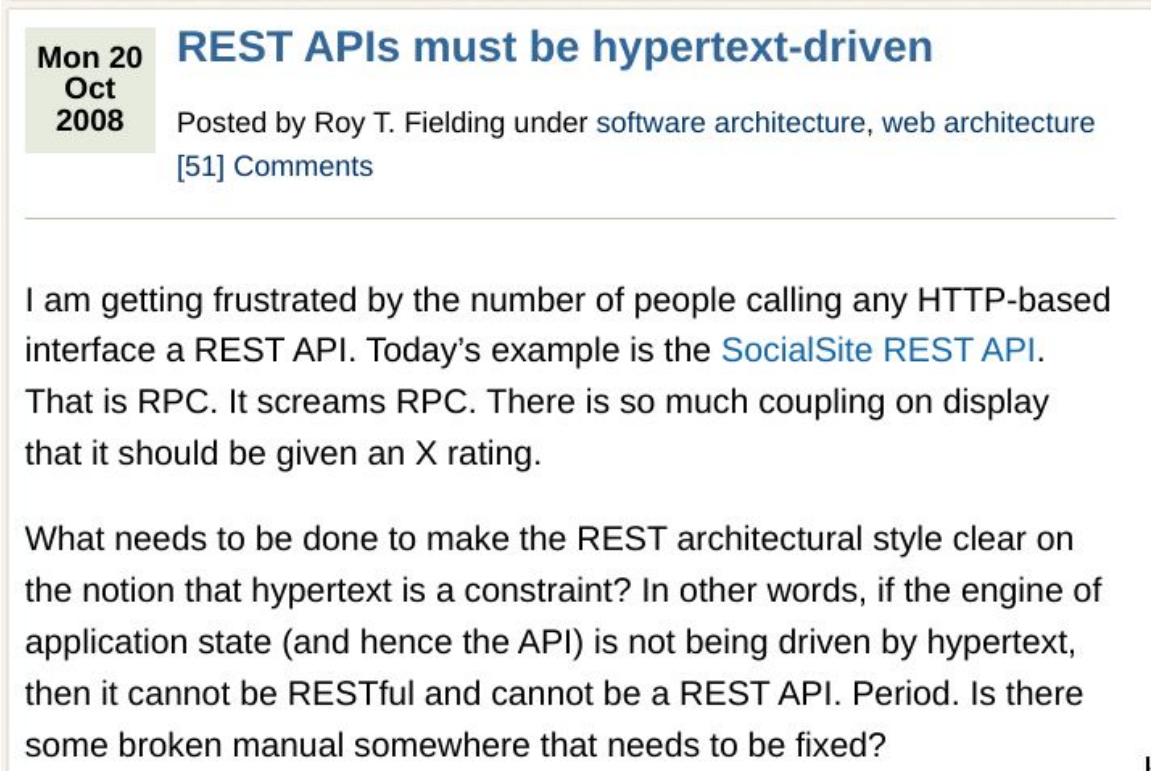
Hypermedia

Wikipedia (2022) *"... an extension of the term hypertext, is a nonlinear medium of information that includes graphics, audio, video, plain text and hyperlinks."*

Cambridge Dictionary *"a combination of videos, images, sounds, text, etc. that are connected together on a website, which you can click on in order to use them or to go to other related videos, websites, etc."*

HATEOAS

Study the blog post:



Mon 20 Oct 2008

REST APIs must be hypertext-driven

Posted by Roy T. Fielding under software architecture, web architecture
[51] Comments

I am getting frustrated by the number of people calling any HTTP-based interface a REST API. Today's example is the SocialSite REST API. That is RPC. It screams RPC. There is so much coupling on display that it should be given an X rating.

What needs to be done to make the REST architectural style clear on the notion that hypertext is a constraint? In other words, if the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period. Is there some broken manual somewhere that needs to be fixed?

(Fielding, R. T., 2008)

HATEOAS

- Use of *hypermedia* is a fundamental *constraint*
  - Server must provide *self-descriptive representations* of resources
  - Client can thus interpret the acquired representations and explore its server-provided options regarding obtaining or manipulating the application state
- *Prior knowledge beyond the initial URI and standardised media types used for the server-returned representations shall not be required.*
- Example:
  - HTML (*Hypertext* Markup Language) uses hyperlinks to provide access to images, web documents etc.
  - Browser (client) understands HTML and knows how to render it and what to do when a hyperlink is triggered
  - Not limited to HTML!

Media Types for Resources

- Multipurpose Internet Mail Extensions (MIME), aka. Media Types
- Indicates format of resources
- Managed by IANA - RFC 6838
- Common standardised Media Types:
  - application/xml
  - application/json
  - text/html
  - application/x-www-form-urlencoded

Other Web Service Approaches

OData

- Open Data Protocol
- Supports building and consuming CRUD-based HTTP APIs
- The uniform interface design is inspired by the REST constraints.
- ISO/IEC and OASIS standard since Version 4.

OData GET Example

```
GET https://services.odata.org/v4/TripPinServiceRW/People('russellwhyte') HTTP/1.1

HTTP/1.1 200 OK
Content-Length: 482
Content-Type: application/json; odata.metadata=minimal
ETag: W/'08D1D5BE987DE78B'
OData-Version: 4.0
{
'@odata.context': 'https://services.odata.org/V4/(S(ak3ckilwx5ajembdktfunu0v))/TripPinServiceRW/$metadata#People/$entity',
'@odata.id': 'https://services.odata.org/V4/(S(ak3ckilwx5ajembdktfunu0v))/TripPinServiceRW/People('russellwhyte')',
'@odata.etag': 'W/'08D1D5BE987DE78B'',
'@odata.editLink': 'https://services.odata.org/V4/(S(ak3ckilwx5ajembdktfunu0v))/TripPinServiceRW/People('russellwhyte')',
'UserName': 'russellwhyte',
'FirstName': 'Russell',
'LastName': 'Whyte',
'Emails': [
    'Russell@example.com',
    'Russell@contoso.com',
    null
]
}
```

OData POST Example

```
POST https://services.odata.org/v4/(S(34wtn2c0hkuk5ekg0pjr513b))/TripPinServiceRW/People HTTP/1.1
OData-Version: 4.0
OData-MaxVersion: 4.0
Content-Length: 428
Content-Type: application/json
{
    'UserName':'lewisblack',
    'FirstName':'Lewis',
    'LastName':'Black',
    'Emails':[
        'lewisblack@example.com'
    ]
}
```

```
HTTP/1.1 201 Created
Content-Length: 652
Content-Type: application/json;odata.metadata=minimal;odata.streaming=true;IEEE754Compatible=false;charset=utf-8
ETag: W/'08D1D3800FC572E3'
Location: https://services.odata.org/V4/(S(34wtn2c0hkuk5ekg0pjr513b))/TripPinServiceRW/People('lewisblack')
OData-Version: 4.0
```

GraphQL

- A querying language approach for APIs exposing graph-centric data.
- Provides an uniform interface by using the same type system to describe and query data structures.
- Enables fine-grained control over which properties to expose and which properties to include in the response (reduced payload size).

GraphQL Example

```
# Type Defintions

type Query {
  human(id: ID!): Human
}

type Human {
  name: String
  appearsIn: [Episode]
  starships: [Starship]
}

enum Episode {
  NEWHOPE
  EMPIRE
  JEDI
}

type Starship {
  name: String
}
```

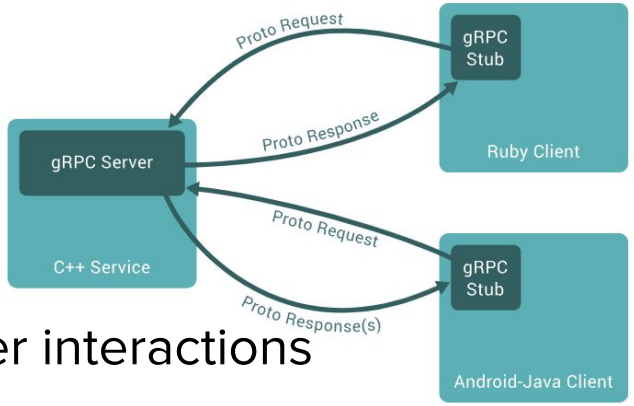
```
# Data Query

{
  human(id: 1002) {
    name
    appearsIn
    starships {
      name
    }
  }
}
```

```
# Query Result

{
  "data": {
    "human": {
      "name": "Han Solo",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "starships": [
        {
          "name": "Millenium Falcon"
        },
        {
          "name": "Imperial shuttle"
        }
      ]
    }
  }
}
```

gRPC



- Remote Procedure Call Framework
- Focus on service oriented design of client server interactions
- Services and their methods, parameters and return types are described using an interface description language (IDL).
- Uses Protocol Buffers by default as IDL that offers a language and platform neutral serialization mechanism for structured data; but formats such as JSON and XML are also supported as IDL.
- Provides a compiler to generated stubs from IDL definitions.
- Supports streaming.

gRPC Example with Protocol Buffers as IDL

```
service HelloService {
  rpc SayHello (HelloRequest) returns (HelloResponse);               # unary call
  rpc LotsOfReplies(HelloRequest) returns (stream HelloResponse);    # server stream
  rpc LotsOfGreetings(stream HelloRequest) returns (HelloResponse);  # client stream
  rpc BidiHello(stream HelloRequest) returns (stream HelloResponse); # bidirectional
}

message HelloRequest {
  string greeting = 1;
}

# The assigned numbers represent field numbers which are
# used to identify the fields in the binary message.
message HelloResponse {
  string reply = 1;
  optional int32 = 2;
}
```

References

- Booth, D. et. al. (2004). Web Services Architecture. https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/
- Cristcost (2007). Representation of concepts defined by WSDL 1.1 and WSDL 2.0 documents [Illustration]. Wikimedia Commons. https://upload.wikimedia.org/wikipedia/commons/c/c2/WSDL_11vs20.png
- Hadely, M. (2009). Web Application Description Language. https://www.w3.org/submissions/wadl/
- Gudgin et. al. (2007). SOAP Version 1.2 Part 1: Messaging Framework. https://www.w3.org/TR/soap12/
- Butek, R. (2005). Which style of WSDL should I use?. https://developer.ibm.com/articles/ws-whichwsdl/
- Christensen, E. et. al. (2001). Web Services Description Language (WSDL) 1.1. https://www.w3.org/TR/wsdl.html
- Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures
- Fielding, R., Nottingham, M. et Reschke., J, (2022). RFC 9110 HTTP Semantics. https://www.rfc-editor.org/rfc/rfc9110
- Fielding, R. T. (2008). REST APIs must be hypertext-driven. https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven
- GraphQL (2024). A query language for your API. https://graphql.org/
- gPRC (2024). Introduction to gRPC. https://grpc.io/docs/what-is-grpc/introduction/