REST

Mohammad Homayoon Fayez
Assistant Professor at ZIBAT

Web Services

- A distributed environment
- Typically hosted by remote systems
- Accessed via HTTP.
- Standard based interfaces separate the applicaion code and the client
- Not tied to a specific set of technologies
- interoperable, extensible, loosely coupled, platform and language agnostic

RPC

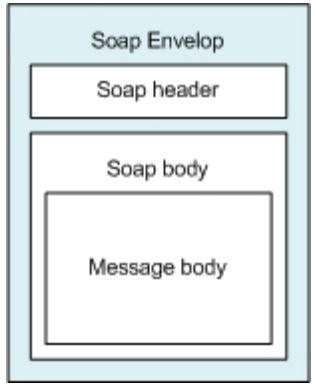
- Remote Procedure Call
  - Presents an Interface for distributed function calls
  - Binds objects of the services to a registry which is used by the client to execute remote procedure calls
    - Example: Remote Method Invocation(RMI)
- XML-RPC
  - The parameters of the remote calls are marshalled as XML
  - HTTP is used as a transport protocol

SOAP

- Simple Object Access Protocol

  - A protocol for exchanging information in a distributed environment
  - Message is the basic unit of communication
  - The message is in XML format
  - Claimed to be loosely coupled
    - Reason : Focus is on the contract provided by WSDL, rather than the implementaion details
  - A W3C standard

SOAP message example



The request

GET /StockPrice HTTP/1.1

Host: example.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:s="http://www.example.org/stock-service">
  <env:Body>
    <s:GetStockQuote>
       <s:TickerSymbol>myCompany</s:TickerSymbol>
    </s:GetStockQuote>
  </env:Body>
</env:Envelope>
```

The response

HTTP/1.1 200 OK

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:s="http://www.example.org/stock-service">
  <env:Body>
       <s:GetStockQuoteResponse>
            <s:StockPrice>45.25</s:StockPrice>
       </s:GetStockQuoteResponse>
  </env:Body>
</env:Envelope>
```

WSDL (Web Service Description Language)

- SOAP Contracts/Interfaces makes a consumer able to easily find a service and bind to it in a platform independent way.
- This Contract is defined as a WSDL document
- WSDL defines what a particular service expects of a consumer and what a service offers.
- WSDL describes operations and messages in an XML format and then bounds them to a network protocol and message format to describe a web Service as an endpoint.

Elements of a WSDL document

Element	Definition
<types>	Data type of the web service
<message>	Definition of the data communicated by the web service
<portType>	The Operations supported by the web service
<operation>	Description of an action supported by the service
<binding>	Communication protocol and data format specification for a particular port type
<port>	A single endpoint defined as a combination of a binding and a network address.
<service>	A collection of related endpoints

A simplified example of WSDL

```
<?xml version="1.0"?>
  <types>
    <schema>
      <element name="AgentDetailsRequest">
        <complexType>
                  <all>
                       <element name=”id” type=“int”/>
                       <element name=”domain” type=“string”/>
                  </all>
        </complexType>
      </element>
    </schema>
  </types>
  <message name="GetAgentDetailsInput">
    <part name="agentInput" element="xsd1: AgentDetailsRequest"/>
  </message>
  <message name="GetAgentDetailsOutput">
    <part name="agentOutput" element="xsd1: AgentDetailsResponse"/>
  </message>
  <portType name=" AgentDetailsPortType">
    <operation name="GetAgetDetails">
      <input message="tns: GetAgentDetailsInput "/>
     <output message="tns: GetAgentDetailsOutput "/>
    </operation>
  </portType>
```

```
<binding name="PguardBinding" type="tns: AgentDetailsPortType ">
    <soap:binding style="document"  transport=" http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetAgentDetails">
            <soap:operation soapAction="http://example.com/GetAgentDetails"/>
             <input>
                    <soap:body use="literal"/>
             </input>
             <output>
                    <soap:body use="literal"/>
             </output>
      </operation>
</binding>
<service name="PguardService">
    <port name="PguardPort" binding="tns:PguardBinding">
                <soap:address location="http://example.com/agent"/>
    </port>
</service>
```

Pros and Cons

Advantages

- Language, platform, and transport independent
- Data integrity
  - SOAP security extension provides the facility to encrypt and digitally sign the message.

Disadvantages

- Complex
- Hard to develop
- Lack of security at SOAP level
- XML payload which consumes a lot CPU power for parsing

REST

- Representational State Transfer
  - Exposes resources as web services through HTTP’s Uniform Interface
  - REST is defined by
    - Identification of resources to be exposed as services
    - URI’s to access the resources
    - Relationship between the resources and the HTTP methods
    - Manipulation of the resources through representaions
    - Example

HTTP Uniform Interface

- REST is an architectural style which applies additional constraints to the usage of Hyper Text Transfer Protocol. It emphasizes on the right use of the http verbs
- HEAD
- GET
- POST
- PUT
- DELETE

Representational State Transfer ?

- RESTful web services are composed of resources
- A University's department with name = IT, for example, is a resource which could be accessed by ttp://www.example.com/departments/IT
- The above query will return a representation of the resource
- The representation could be an image, a XML or HTML document
- This will place the client in a state and yet in a new state if the representation holds a new hyperlink to another resource and is traversed by the client
- These are merely the current states of the resources and not the session states
- Therefore the term Representational State Transfer meaning that the client changes (transfers) state with each resource representation

Stateless

- RESTful communications are stateless where each request from the client to server must contain all the information necessary for the server to understand the request. It means that the client cannot take the advantage of any stored context on the server.
- Session states are kept on the client i.e. client is responsible for tracking its own actions.
- Statelessness improves availability and reliability because recovery from partial failures becomes easy.
- The server can quickly free resources because it does not need to store states between requests which improve scalability

Resource

A resource is the intended target of a hypertext reference i.e. any information like a document or an image which can be accessed via a hypertext reference or URI. A resource could be static, will have the same value whenever it is accessed, or dynamic i.e. with varying values. For example a resource could be a particular article, the list of the products of a company or the photos of the company employees for the year 2010 etc.

Representation

Data and metadata

The representation of a resource is a data entity which is used by REST components to perform an action on e.g. GET or PUT. A representation is composed of data and metadata.

Control data

The purpose of a message, such as the action being requested is defined by Control Data. Control Data is also used to override the default behaviour of some connection elements. For example cache behaviour can be modified by Control Data included in the request or response message. The control data can be used to know the current or the desired state of the requested resource. It can also be used for content negotiation to select the best representation if at a given time multiple representation/versions exist.

Representation

Media type

The representations can have different data formats. These data formats are called media types. A representation can be processed by the recipient according to its media type. For example a XML or JSON representation for automated processing and a HTML or picture format for human view. One important thing to mention is that REST does not impose any constraint on physical representations of the resources, because applications and users could have varied needs. So it is not limited to JSON and XML.

Accessing resources

In RESTful web services resources are uniquely identified by URIs. URI is a string of characters for identifying a resource. URL and URN are described as subsets of URI by IETF (Internet Engineering Task Force) in rfc2396. In this context URI is both the locator and the name of the resource. We can say that a URI is composed of a URL and a URN which can graphically be illustrated as in figure 2.6.

http://www.ietf.org/rfc/rfc2396.txt

Each resource must have a unique name which is accessed via its URI. The resource should be accessed via nouns (the name of the resource) and not verbs like getStudent. The following is not a RESTful way because getStudent is a verb not a noun.

http://www.example.com/teachers/getTeacher?name=Fayez

A RESTful URI will be:

http://www.example.com/teachers/Fayez

URI schemas

To access RESTful resources, specific URI schemas are used. Each URI is uniquely mapped to a resource. One good practice will be to define URIs that makes sense to ordinary users.

For example /{University}/{Department} makes perfect sense, everyone will understand this hierarchy.

But /{University}/{Department}/{Location}/{longitude}/{latitude} don’t make sense because latitude is not part of the hierarchy of long- itude.

Both longitude and latitude are the subset of location. The following example illustrates how to construct a URI for resources of the same level.

URI schemas

A simple path that points to a single resource would look like this:
/{University}/{Department}

If there are multiple resources of the same level then a semicolon is used to separate them.
/{University}/{Department}/{Location}/{longitude};{latitude}

A representation of a resource can be exposed in different formats as shown bellow
/maps.{format}

A user agent can use it to get a maps.xml or maps.json format as follows
http://www.example.com/../../maps.xml
http://www.example.com/../../maps.json

HATEOAS

Resources are accessed through their URIs which could also be implemented as links in html documents. These links are references to some resources which may hold new URIs/links, in their representation, to other resources. This is how we get “the Web” where information is connected together through hyperlinks. Beside hyperlinks the HTML forms can also be used to access a resource. This approach of linking and form submission is called Hypermedia as the engine of application state

(HATEOAS). Clients will get faster responses from the services using this approach because the client will not wait for a complete list of products to be generated. Instead, the service will send for example a list of first 10 products and a hyperlink to the next 10.

Following is an example of embedded hyperlink into an XML representation to the next 10 products.

```
<products>
<link rel="next" href="http://example.com/webstore/products?listSize=10"/>
  <product id="001">
    <name>Harddisk</name>
                <size>1000Gb</price>
    <price>DKK 500 </price>
  </product>
...
</products>
```

WADL

RESTful web services can be described using WADL. WADL is an alternative specification to WSDL with support for RESTful web services. WADL is designed to provide a machine process-able description of HTTP based Web applications.

WADL can describe the resources, the relationships between resources, the HTTP methods that can be applied to each resource, the expected input and output and their supported formats and the resource representation formats.

From the following example one can clearly see that WADL is easy to understand in contrast to WSDL which is basically used for SOAP.

WADL example

```
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://research.sun.com/wadl/2006/10 wadl.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:ex="http://www.example.org/types"
    xmlns="http://research.sun.com/wadl/2006/10">
    <grammars>
                <include href="ticker.xsd"/>
    </grammars>
    <resources base="http://www.example.org/services/">
      <resource path="getStockQuote">
```

```
<method name="GET">
            <request>
                <param name="symbol" style="query" type="xsd:string"/>
            </request>
            <response>
              <representation mediaType="application/xml"
                  element="ex:quoteResponse"/>
              <fault status="400" mediaType="application/xml"
                  element="ex:error"/>
            </response>
          </method>
        </resource>
    </resources>
</application>
```

Goals

Roy Thomas Fielding in the abstract of his dissertation defines the goals of REST as:

- Scalability of component interactions
- Generality of interfaces
- Independent deployment of components
- Intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems

Scalability of component interactions

Scalability can surly be achieved because we know that the Web is growing exponentially without degrading in performance. Different kinds of clients can access servers and other clients through web. The Web based tools are even extended to the hand held devices

Generality of interfaces

We mentioned before that REST uses HTTP's Uniform Interface. It is a generic interface which can be used by any HTTP client to communicate to any server with ease. This is because of its simplicity and well known general methods

Independent deployment of components

This is a very important issue because if the deployment of new components is dependent on old components some servers may not be upgraded and newer clients may not be able to talk to them or vice versa. Under the topic “REST applied to HTTP” in his dissertation Roy states:

“One of the major goals of REST is to support the gradual and fragmented deployment of changes within an already deployed architecture”. Architectural Styles and the Design of Network-based Software Architecture by Roy Thomas Fielding

Intermediary components

The intermediary components that REST requires could be different kinds of Web proxies. We know that some proxies are used to cache information to reduce interaction which will improve performance. Other kinds of proxies enforce security policies.

Pros and Cons

- Manageability
  - Complex business logic is divided in resources which are small units of functionality. This approach makes the application easily manageable.
- Scalability
  - REST uses http as its communication protocol for web services and the web is growing exponentially without degrading in performance.
  - Http is cacheable which provides the ability to serve more clients.
  - Intermediaries like a cache proxy can be introduced.
- Extensibility
  - One of the design goals of REST is ‘independent deployment of components’ i.e. extensibility
  - In a RESTful web application business logic is divided into small autonomous building blocks of functionality which makes the application highly and easily extensible.
  - New content-types can be introduced at any time because the representation of resources is not bound to any particular content-type.
- Availability
  - Benefits from the HTTP’s Uniform Interface
- Reusability
  - Because it is cacheable

Pros and Cons

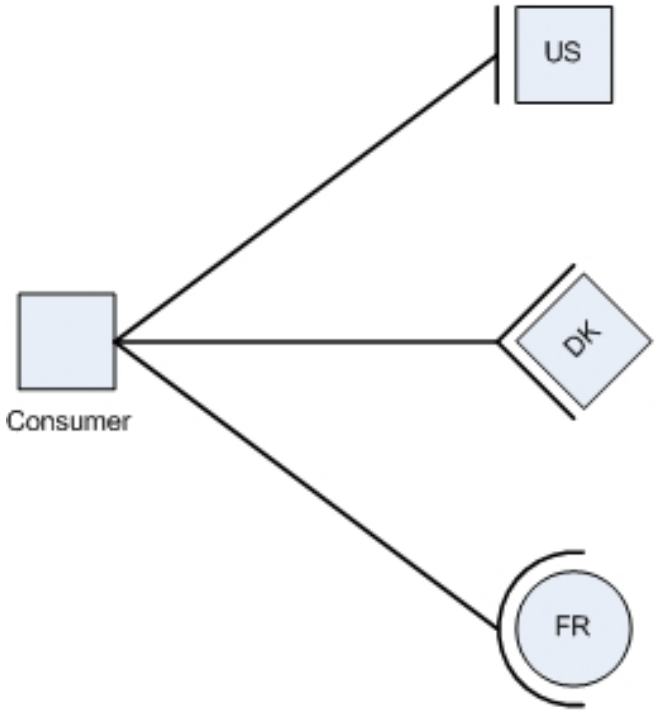
- Security
  - XML signatures and encryption
  - Wire encryption through HTTPS
  - REST has compatibility for intermediaries therefore proxies which enforce security policies can be used
- Lightweight - not a lot of extra xml mark-up
- Easy to build, no toolkits required
- Language and platform independent
- Easy to develop
- Easy to learn,
- Less reliance on tools
- No need for additional messaging layer
- Close in design and philosophy to the Web

Disadvantages

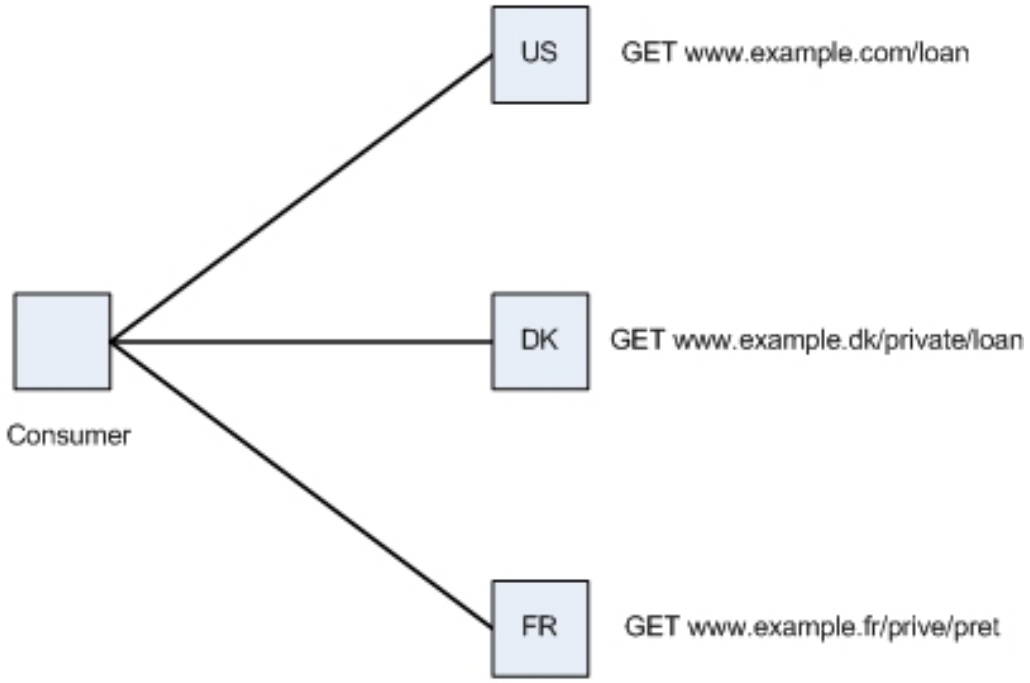
- Security issues for example sending sensitive data as parameters over HTTP.

http://www.taranfx.com/rest-vs-soap-using-http-choosing-the-right-webservice-protocol (Rest ain't perfect)

REST Interface vs service specific contracts



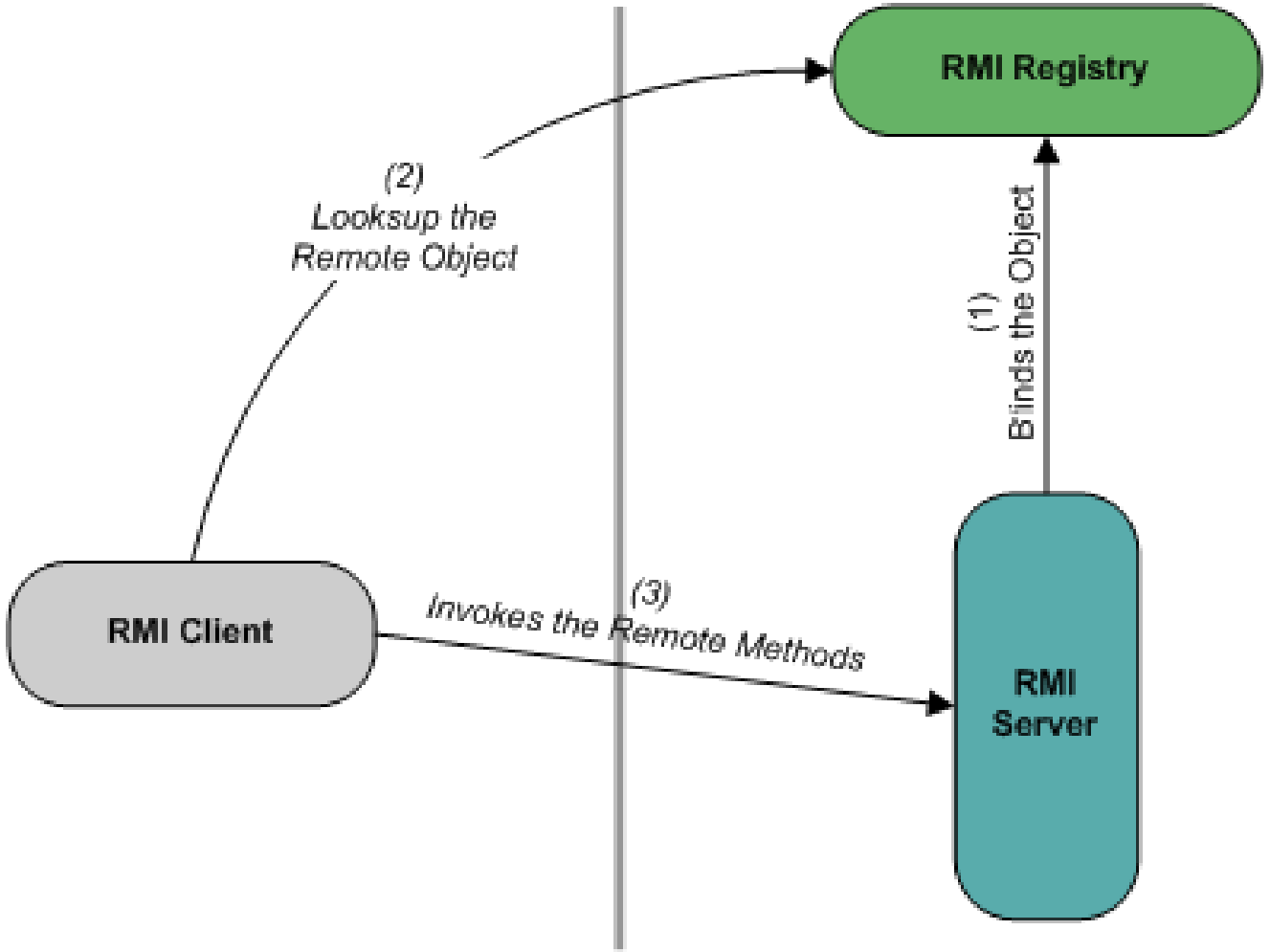
Service specific contracts



REST Uniform Interface

back

RMI Example



back