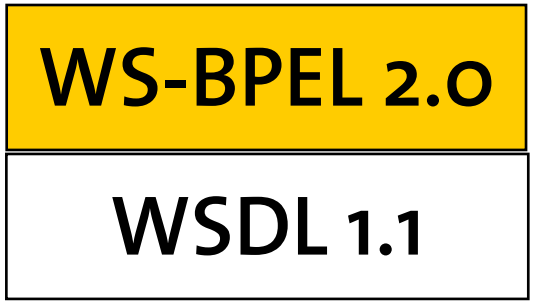
BPEL for REST

Cesare Pautasso
Faculty of Informatics
University of Lugano (USI), Switzerland
http://www.pautasso.info

RESERVOIR

Motivation – Why BPEL for REST?

> "The WS-BPEL process model is layered on top of the service model defined by WSDL 1.1. [...]
> Both the process and its partners are exposed as WSDL services"
>
> [BPEL 2.0 Standard, Section 3]



RESTful Web Services APIs...

...do not use WSDL 1.1

The Goal

- Compose RESTful Web Services
- Compose WSDL Web Services
- Use Business Process Modeling Languages

One solution: JOpera.org

- No need to change the JOpera composition language to compose both kinds of services

The Goal

- Compose RESTful Web Services
- Compose WSDL Web Services
- Use Business Process Modeling Languages

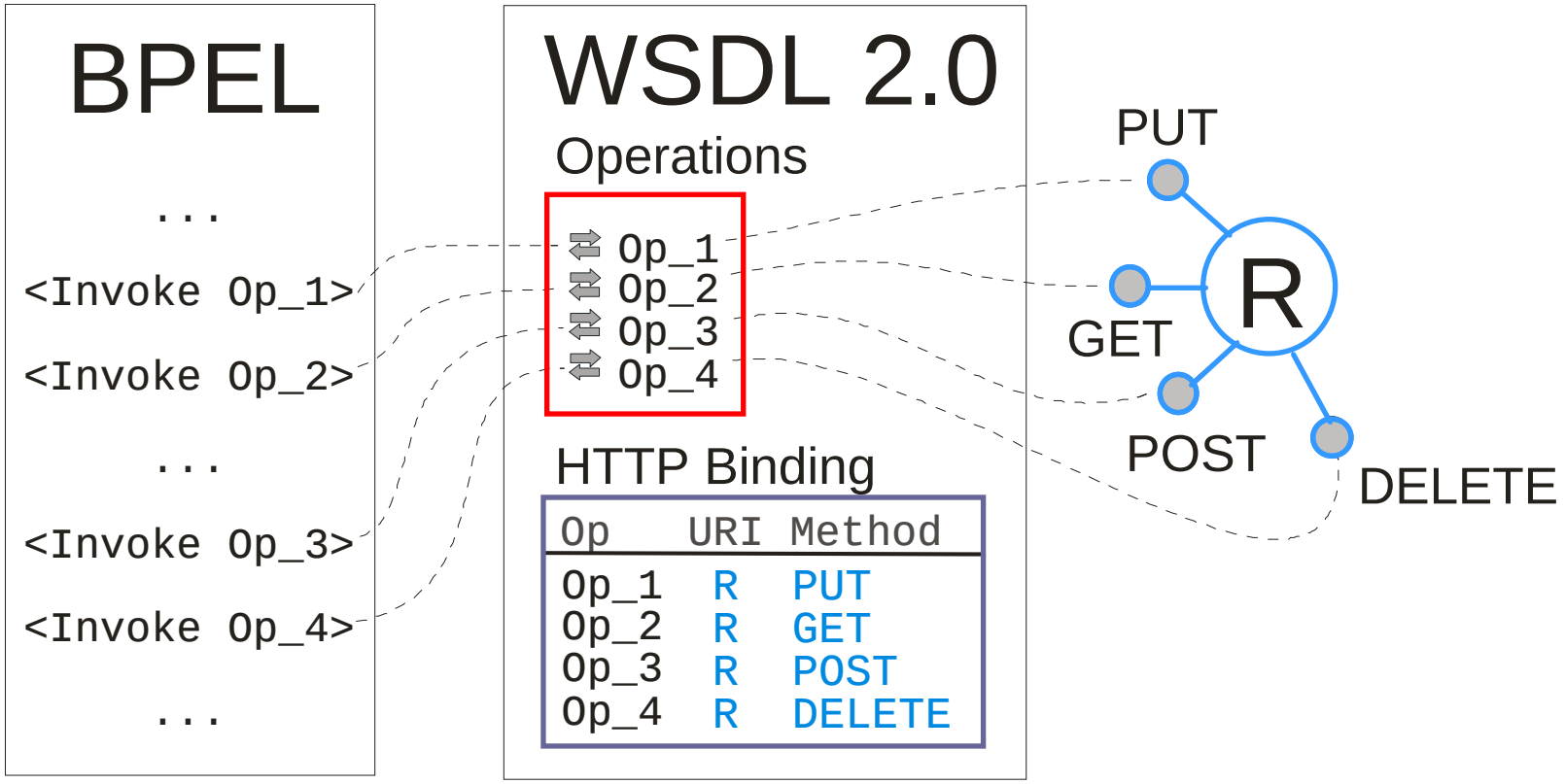
In this paper: BPEL for REST

- Extend BPEL to support RESTful Web Services

The Hack – Without BPEL for REST

WSDL 2.0 HTTP Binding can wrap RESTful Web Services

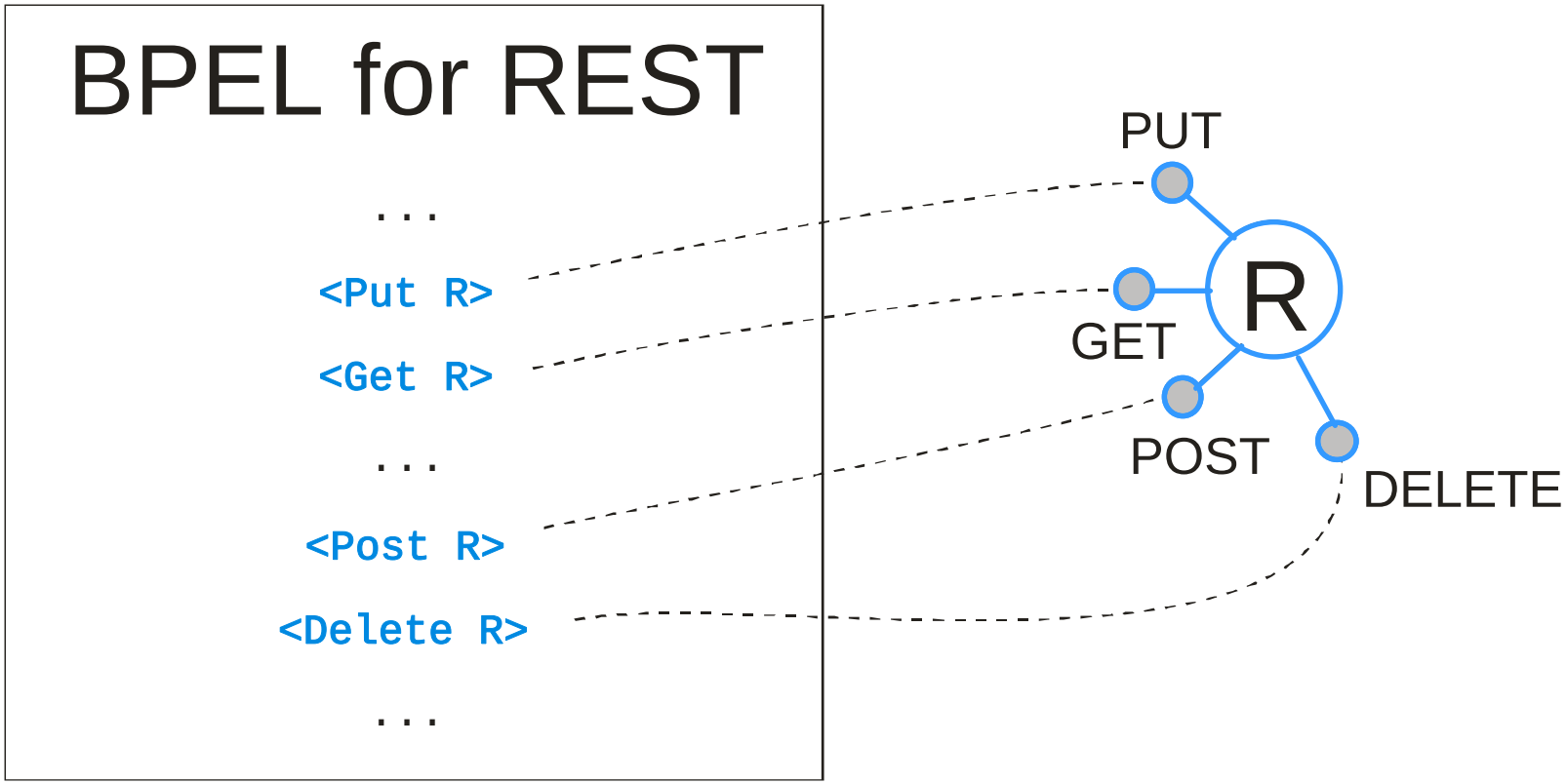
(WS-BPEL 2.0 does not support WSDL 2.0)



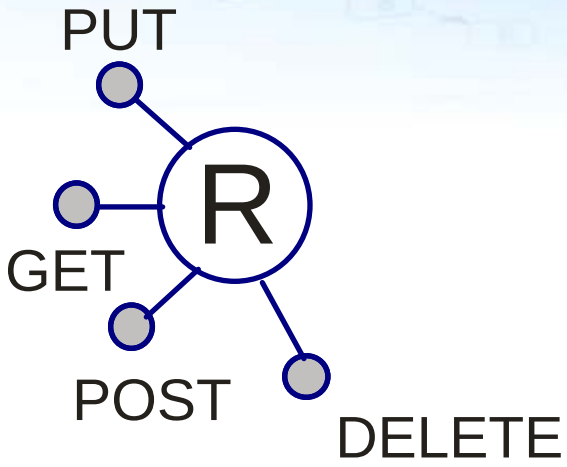
BPEL for REST

Native support for direct invocation of RESTful Web services

REST concepts first-class language constructs



REST in one slide



- Web Services expose their data and functionality trough **resources** identified by **URI**
- **Uniform Interface** Principle: Clients interact with the state of resources through 4 verbs: GET (read), POST (create), PUT (update), DELETE
- **Multiple representations** for the same resource
- **Hyperlinks** model resource relationships and valid state transitions

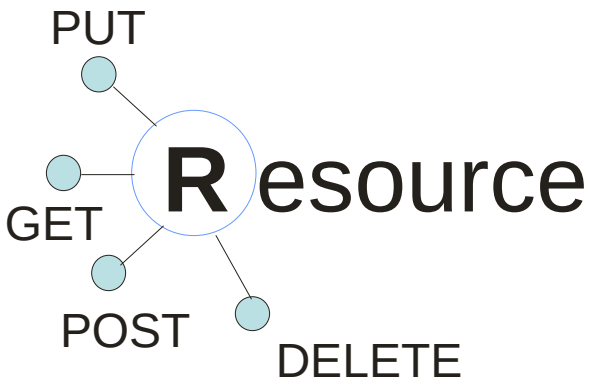
The Challenges

- Resource addressing through URI
  - *Interact with dynamic, variable set of URI*
- Uniform Interface (GET, POST, PUT, DELETE)
  - *Make the 4 verbs explicit in the composition language*
- Multiple resource representations
  - *No static message types*
  - *Negotiate with clients the most appropriate representation*
- Hyperlinks
  - *Implement state transition logic of a resource*
  - *Generate new URIs dynamically as processes runs*

Outline – BPEL for REST

- Motivation
- BPEL for REST Extensions
  - Invoking RESTful Web Services
  - Publishing RESTful Web Services
- Example
- Outlook

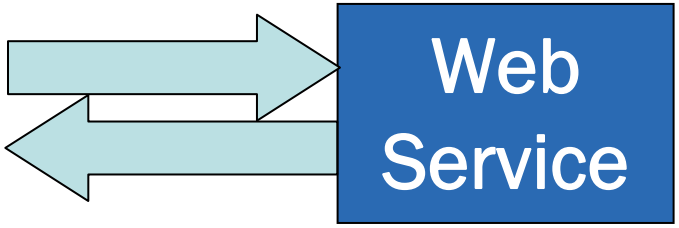
BPEL for REST - invocation primitives




```
<put>
<get>
<post>
<delete>
```

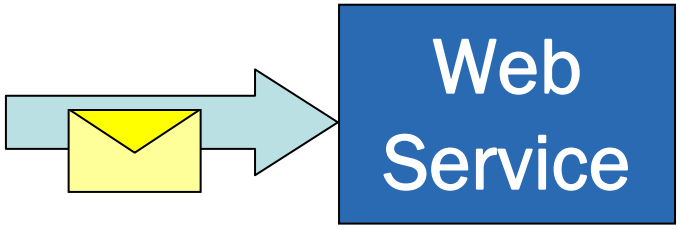
```
<onPut>
<onGet>
<onPost>
<onDelete>
```

```
<invoke>
```




```
<receive>
<reply>
```

```
<invoke>
```




```
<receive>
```

Invoking RESTful Web Services

- 4 new activities (`get`, `post`, `put`, `delete`)

```
<get uri="" response="">
  <header name="">*value</header>
  <catch code="" faultName=""?>*...</catch>
  <catchAll>?...</catchAll>
</get>

<post uri="" request="" response=""> ... </post>
<put uri="" request="" response=""?> ... </put>
<delete uri="" response=""?> ... </delete>
```

Publishing RESTful Web Services - I

- 4 new request handlers (`onGet`, `onPut`, `onDelete`, `onPost`)
- 1 new "scope" (`resource`)

```
<resource uri="">
  <variable>*
  <onGet>? ... </onGet>
  <onPut>? ... </onPut>
  <onDelete>? ... </onDelete>
  <onPost isolated="false"?>? ... </onPost>
</resource>
```

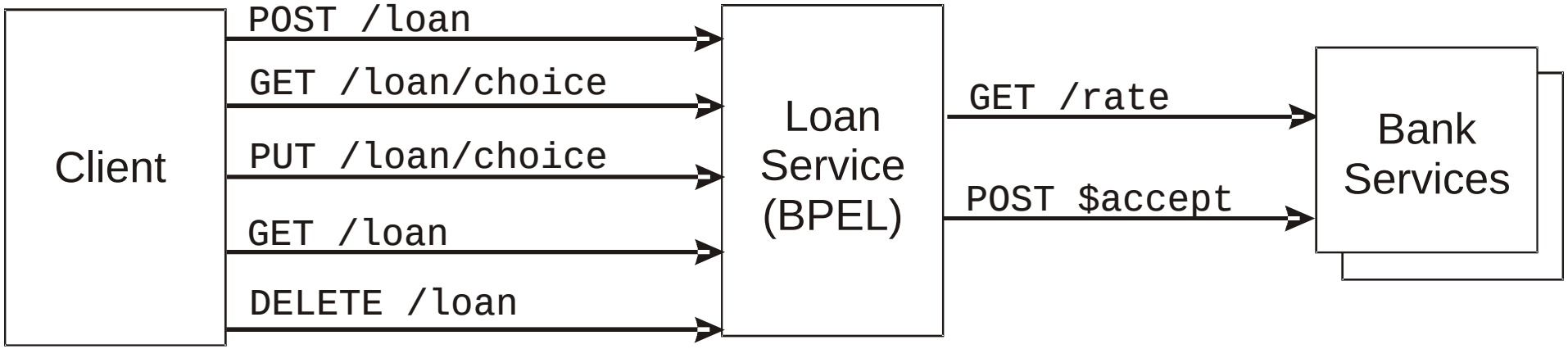
Publishing RESTful Web Services - II

- 1 new activity (`respond`)

```
<respond code=""?>
   <header name="">*value</header>
   payload
</respond>
```

Example

- Loan Application Service Process



1. State of the Resource

- Declarative construct to publish a resource URI

```
<process name="LoanApplication">
  <resource uri="loan">
<!-- State variables of the resource -->
  <variable name="name"/>
  <variable name="amount"/>
  <variable name="rate"/>
  <variable name="bank"/>
  <variable name="start_date"/>
  <variable name="end_date"/>
```

2. Handle POST request - Preconditions

- Create a new loan application resources only if...

```
<onPost>
  <if>
<condition>$request.amount &gt 100000</condition>
  <then>
      <respond code="400">
            Requested amount too large
      </respond>
      <exit/>
  </then>
```

3. Handle POST request - Initialization

- Store initial loan application resource state

```
<else><sequence>
  <assign>
      name = $request.name;
      amount = $request.amount;
      start_date = $request.start_date;
  </assign>
  <respond code="201">
      <header name="Location">/loan/$name</header>
      Processing loan application…
  </respond>
```

4. Invoke RESTful Web Services

- Get rates from banks

```
<scope>
  <variable name="ubs_response"/>
  <variable name="cs_response"/>
  <variable name="url_accept"/>
  <variable name="accept_response"/>
<flow>
<get
  uri="http://www.ubs.ch/rate?chf=$amount&from=$start_date"
  response="ubs_response">
<get
  uri="http://www.cs.ch/rates?amount=$amount&start=$start_date"
  response="cs_response">
</flow>
```

5. Let Client Choose – GET Handler

- Return rates offered by the banks

```
<while>
<condition>TRUE</condition>
<resource uri="choice">
  <onGet>
      <respond code="200">
  <header name="Content-Type">application/json</header>
 [ { bank:"cs",
     rate:"$cs_response.rate",
 end_date:"$cs_response.until" },
   { bank:"ubs",
     rate:"$ubs_response.rate",
 end_date:"$ubs_response.end" } ]
      </respond>
  </onGet>
```

6. Let Client Choose – POST Handler

- Store the client choice and continue

```
<onPost><sequence>
  <assign>bank = $request.choice;</assign>
      <if>
<condition>bank == "cs"</condition>
      <then>
<assign>rate = $cs_response.rate;
  end_date = $cs_response.until;
  url_accept = $cs_response.accept</assign></then>
<else><assign>rate = $ubs_response.rate;
  end_date = $ubs_response.end;
  url_accept = $ubs_response.accept</assign></else>
</if>
<respond code="200"/><activeBPEL:break/>
</sequence></onPost></resource></while>
```

7. Inform Bank

- Accept the loan offered by the chosen bank

```
<post uri="$url_accept" request="$name"
  response="accept_response">
</scope>
</sequence>
</else>
</if>
</onPost>
```

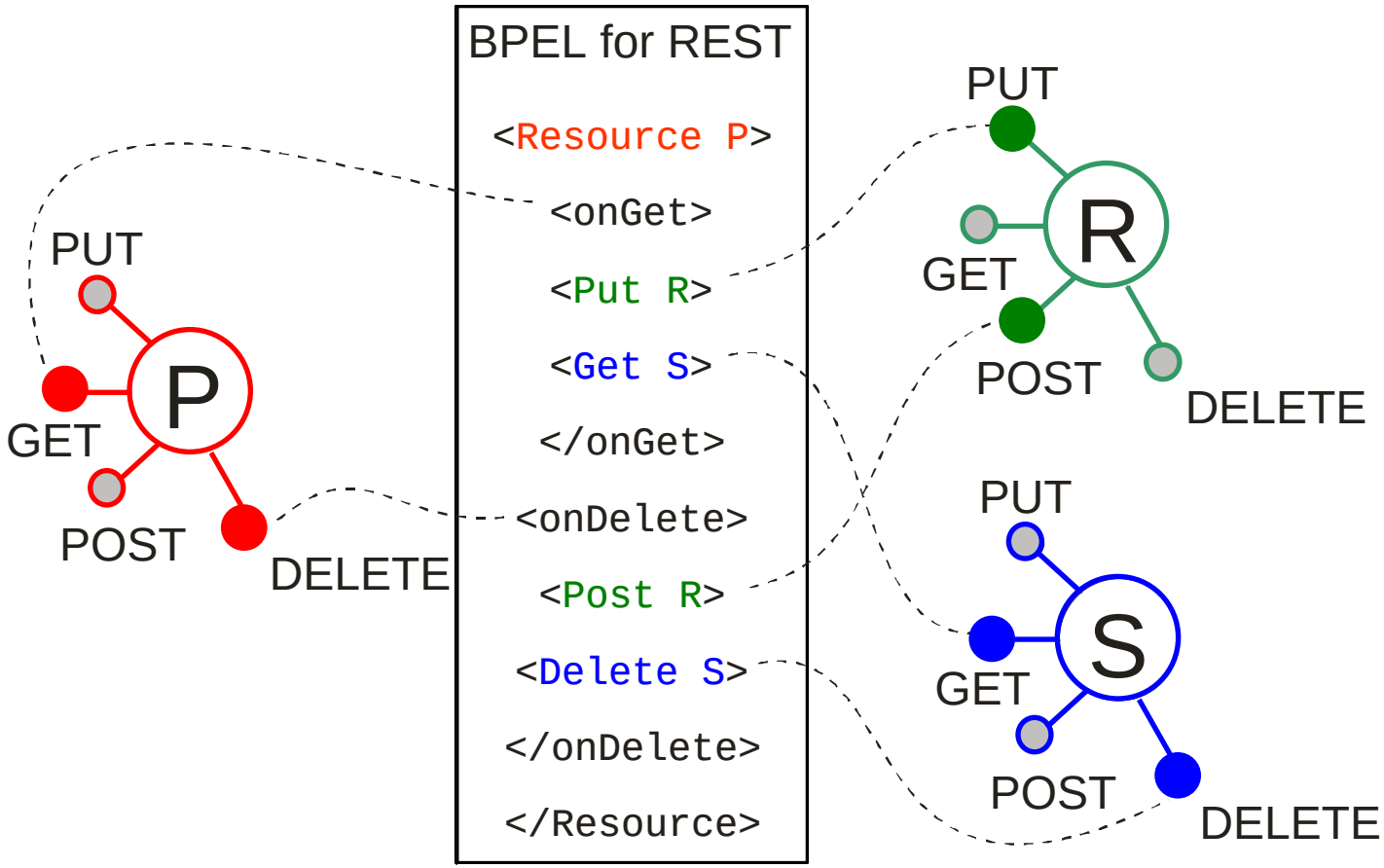
8. Let Client Choose – DELETE Handler

- Cancel the loan application or cancel the loan

```
<onDelete>
  <if>
      <condition>bank == null</condition>
  <then>
      <respond code="200"/>
      <exit/>
  </then>
  <else>
      <!-- Start the loan cancellation process -->
      <invoke...>
  </else>
  </if>
</onDelete>
```

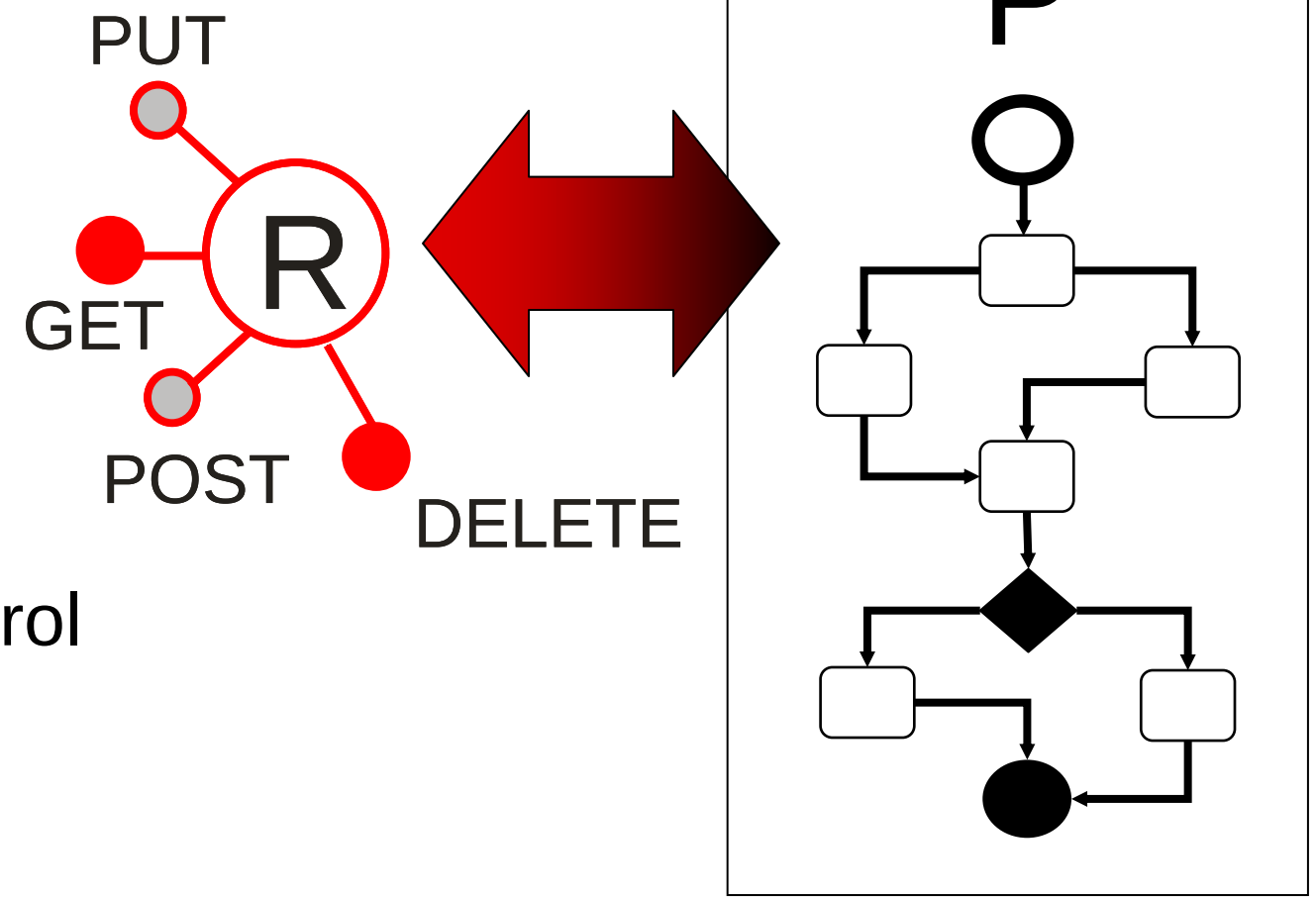
BPEL for REST Extensions - Summary

- Publishing and Invoking RESTful Web Services



Mapping the State of Resources and Processes

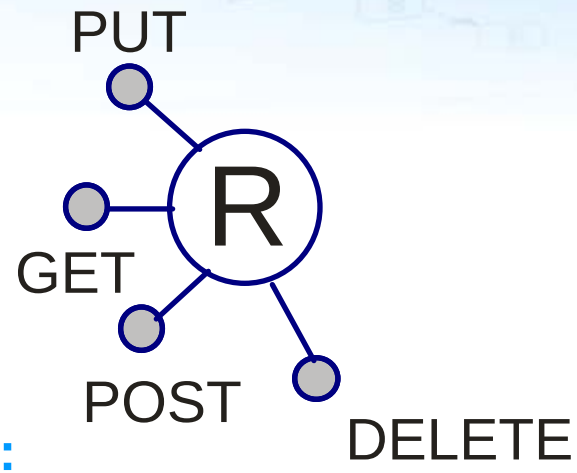
- Correlation
- Lifecycle
- Visibility
- Access Control



Conclusion

- Business Process Modeling Languages have been applied with success to compose “traditional” WS-* Web Services (**BPM = SOA + BPEL**)
- **Business Process Modeling Languages should also be applied to compose RESTful Web Services**
- BPEL for REST is a lightweight WS-BPEL extension for composing both kinds of services

More information



- R. Fielding, **Architectural Styles and the Design of Network-based Software Architectures**, PhD Thesis, University of California, Irvine, 2000
- C. Pautasso, O. Zimmermann, F. Leymann, **RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision**, Proc. of the 17th International World Wide Web Conference (WWW2008), Bejing, China, April 2008.
- H. Overdick, **Towards Resource-Oriented BPEL**, Proc. of the 2nd Workshop on Emerging Web Services Technology (WEWST2007), Halle-Saale, Germany, November 2007.
- C. Pautasso, G. Alonso, **From Web Service Composition to Megaprogramming**, Proc. of the 5th VLDB Workshop on Technologies for E-Services (TES-04), 29/08/2004, Toronto, Canada, (2004)

BPEL for REST

Cesare Pautasso
Faculty of Informatics
University of Lugano (USI), Switzerland
http://www.pautasso.info

RESERVOIR