Unit 5: Design of Web Services (1/2)

Introduction to Web Services. Resource API with REST: RESTful Web Services

Design of Web Services

- Introduction to Web Services
  - Web Service API styles
- Resource API with REST: RESTful Web Services
- RPC API
- Message API
- Flexible Query API

Web Service: Definition

- A Web service is a programmatically available application logic exposed in a well-defined manner over standard Web protocols
- According to W3C Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks

Web Service API Styles

- A **Web Service API Style**
  - Is an architectural pattern that governs the interaction patterns between clients and services over the web.
  - Outlines how requests and responses are structured.
  - Determines the protocols and technologies to be used
- Importance:
  - Determines how clients consume services.
  - Influences scalability, maintainability, and performance.
  - Impacts the service's architecture, development speed, and long-term viability.

Web Service API Styles 1/2

Pattern Name	Problem	Description	Technology
RPC API	How can clients execute remote procedures over HTTP?	Define messages that identify the remote procedures to execute and also include a fixed set of elements that map directly into the parameters of remote procedures	• SOAP • gRPC
Message API	How can clients send commands, notifications, or other information to remote systems over HTTP while avoiding direct coupling to remote procedures?	Define messages that are not derived from the signatures of remote procedures. These messages may carry information on specific topics, tasks to execute, and events	• AMQP • MQTT

Web Service API Styles 2/2

Pattern Name	Problem	Description	Technology
Resource API	How can a client manipulate data managed by a remote system, avoid direct coupling to remote procedures, and minimize the need for domain-specific APIs?	Assign all procedures, instances of domain data, and files a URI. Leverage HTTP as a complete application protocol to define standard service behaviors	• REST
Flexible Query API	How can clients query or manipulate data with high flexibility while minimizing over-fetching and under-fetching of data?	Allow clients to specify the structure of the response data by defining queries in a strongly typed schema. The server resolves these queries based on its schema definition and returns data in a standardized format.	• GraphQL

Design of Web Services

- Introduction Web Services
- Resource API with REST: RESTful Web Services
  - REST
  - Defining the Resource API
  - Client-Service Interaction Style patterns
  - Richardson Maturity Model
- RPC API

REST

- **RE**presentational **S**tate **T**ransfer: **architectural style** for designing networked applications.
  - *Representational*. Interactions between clients and services are made through "representations" of resources.
  - *State.* The representation encodes the current state of a resource.
  - *Transfer*: Representations can be transferred two ways: from server to client for retrieval or vice versa for updates and creations.
- REST typically means REST over HTTP, although it's not protocol-specific.
- “RESTful” is typically used to refer to web services implemented according to REST.

REST Architectural Constraints

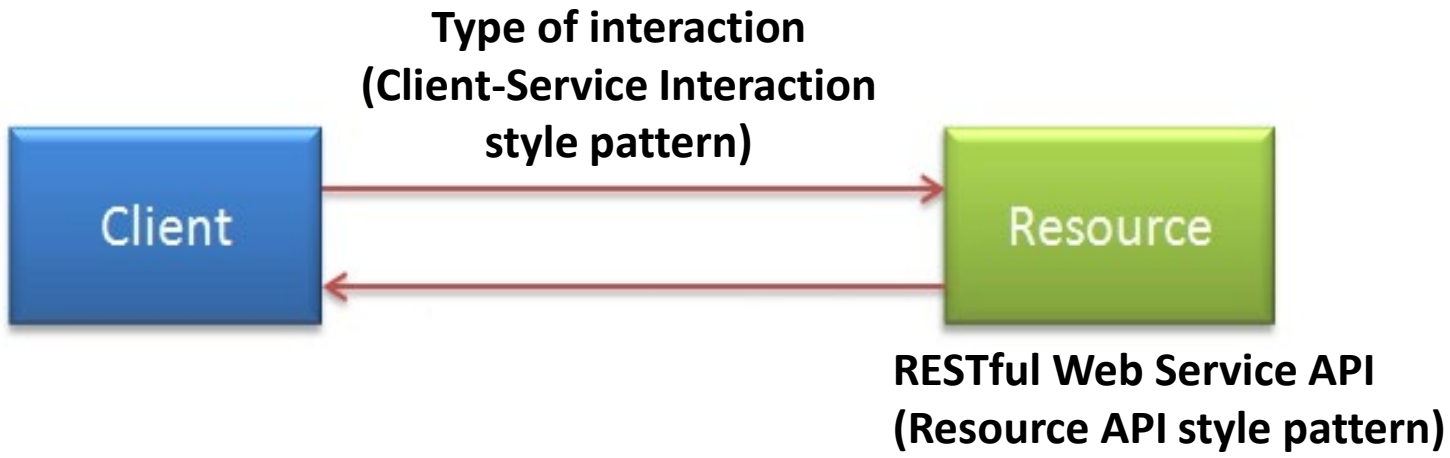
- Uniform interface
  - Each resource have its own specific URI
  - Resources are manipulated using the four basic methods provided by HTTP: PUT, GET, POST, and DELETE
- Client-Server
- Stateless
  - Each HTTP request happens in complete isolation. The server never relies on information from previous request
- Cacheable
  - Responses must be capable of being cached by clients and intermediaries

REST Architectural Constraints (cont.)

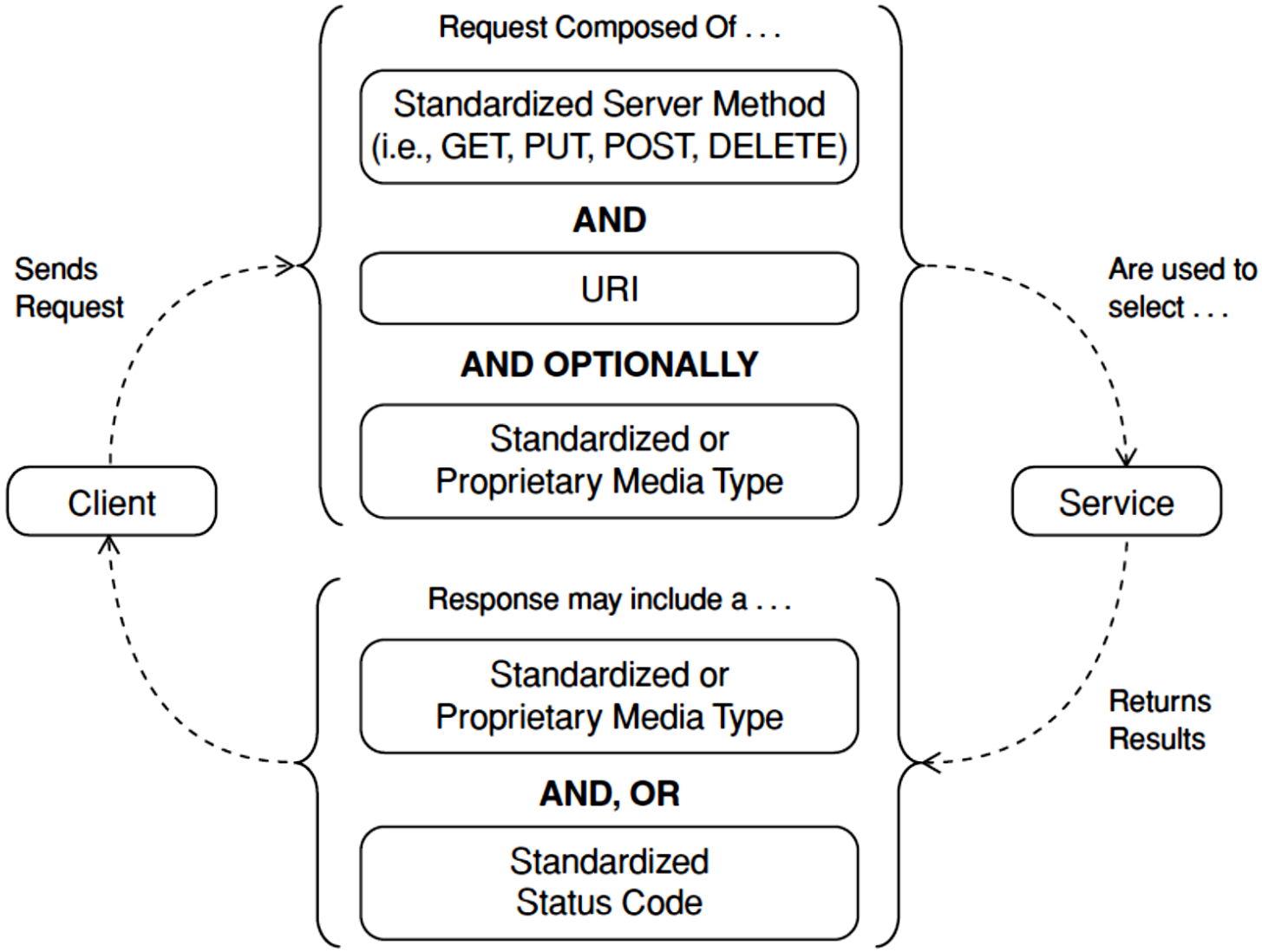
- Layered system
  - A client cannot ordinarily tell whether it is connected directly to the end server or an intermediary along the way.
  - Intermediaries, such as proxy servers, cache servers, etc, can be inserted between clients and resources to support performance, security, etc
- Code on demand (optional)
  - Servers can temporarily extend or customize the functionality of a client by transferring executable code (e.g. JavaScript).

Design of RESTful Web Services

- To design a RESTful Web Service, we have to:
  - Define the Service API
    - Apply the Resource API style pattern
  - Refine the type of interaction between the client and the service
    - Select and apply Client-Service Interaction Style patterns



Resource API



* Extracted from: Robert Daigneau. Service Design Patterns. Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Addison Wesley, 2012

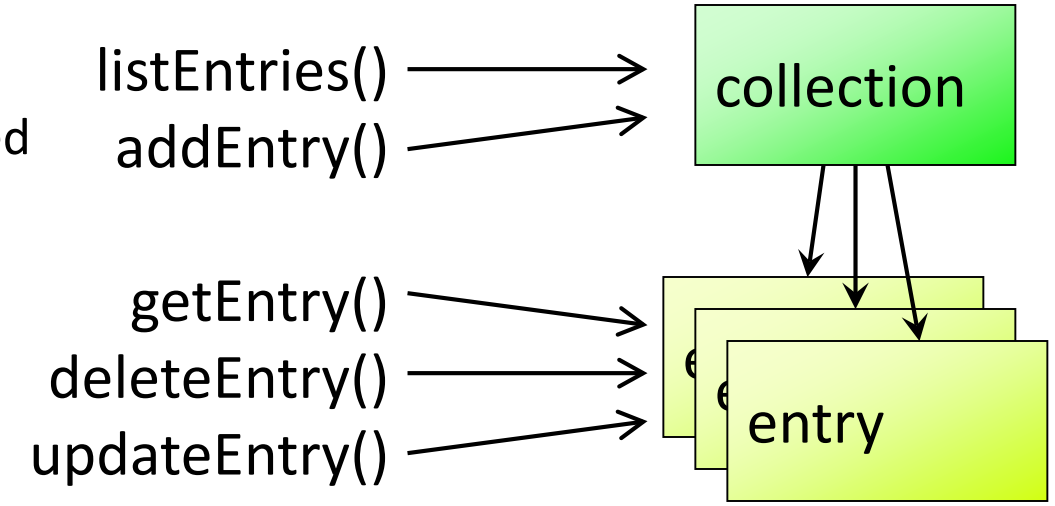
Defining a Resource API: A Blueprint

1. Identify resources to be exposed as services
2. Define “nice” URIs to address the resources
3. Understand what it means to do a GET, POST, PUT, DELETE for each resource
4. Design and document ingoing/outgoing resource representations
5. Use status codes and headers meaningfully in server responses
6. Document the API using a standard notation (e.g. OpenAPI)

1. Identify resources to be exposed

- A resource is anything that is important enough to be referenced as a thing itself
- Usually we have 2 levels of abstraction:
  - A collection: is a resource representing a whole class and/or set of individuals
  - An entry: is a resource representing an instance/individual within a class/set

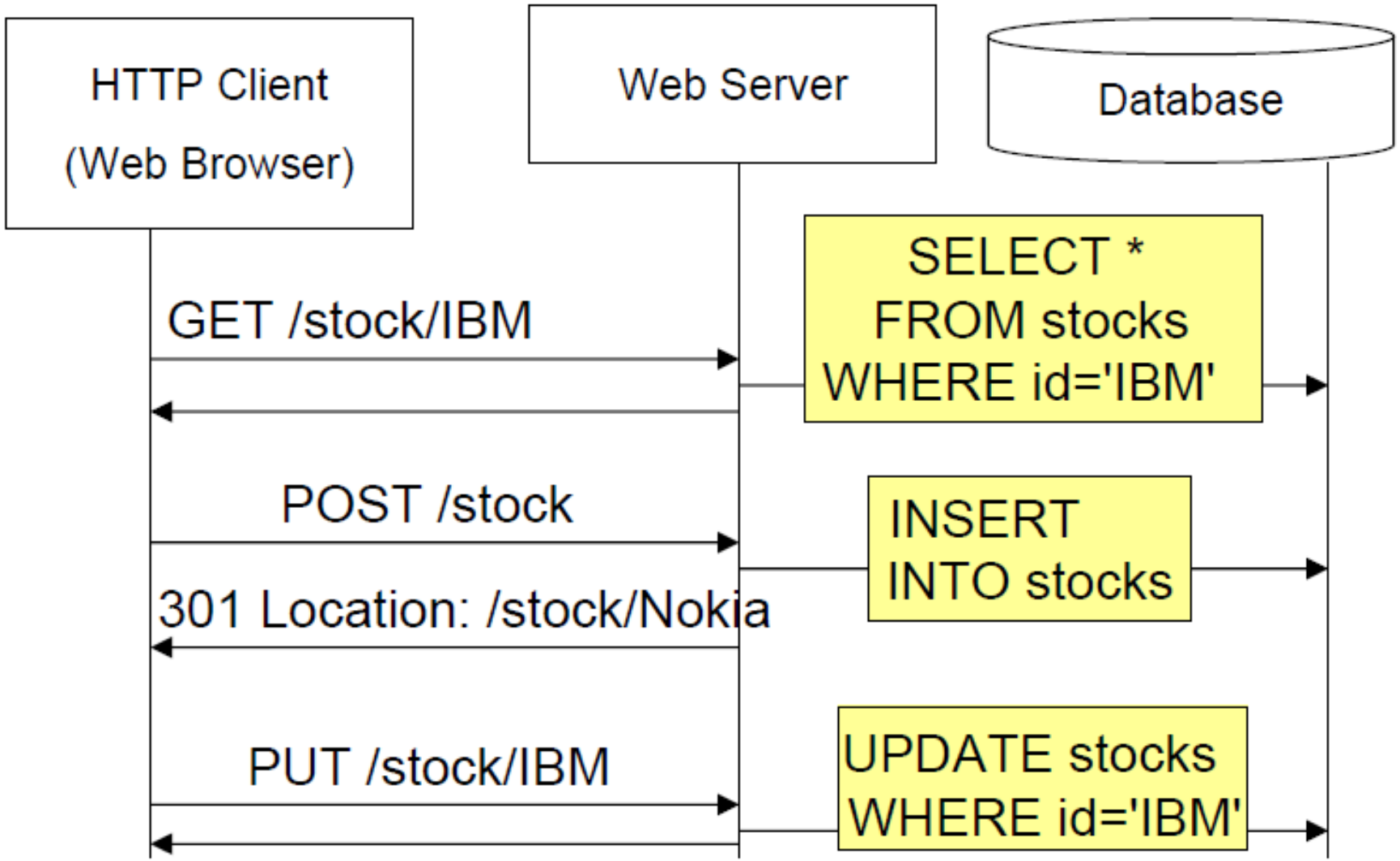
Operations can be distributed onto the resources (the collection, each entry) and mapped to a small uniform set of operations



2. Define “nice” URIs to address the resources

- Only two base URIs per resource:
  - Collection: `/stocks` (plural noun)
  - Entry: `/stocks/:stock_id` (e.g. `/stocks/IBM`)
- Query with specific attributes:
  - `/dogs?color=red&state=running&location=park`
- Versioning:
  - `/v1/stocks`
- Paging:
  - `/stocks?limit=25&offset=50`
- Non a resource response (e.g. calculate, convert, …):
  - `/convert?from=EUR&to=CNY&amount=100` (**verbs**, not nouns)

3. Understanding HTTP methods



3. Understanding HTTP methods (Example)

Resource	GET	PUT	POST	DELETE
.../stocks	**List** the members (URIs and perhaps other details) of the collection. For example list all the stocks.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's ID is assigned automatically and is usually returned by the operation.	**Delete** the entire collection.
.../stocks/IBM	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Update** the addressed member of the collection, or if it doesn't exist, **create** it.	Treat the addressed member as a collection in its own right and **create** a new entry in it.	**Delete** the addressed member of the collection.

3. Understanding HTTP methods : Safeness & Idempotence

- Method GET is ***safe***, which means it is intended only for information retrieval and should not change the state of the server. In other words, it should not have side effects, beyond relatively harmless effects such as logging, caching, etc.
- Methods PUT and DELETE are defined to be ***idempotent***, meaning that multiple identical requests should have the same effect as a single request.
- Method GET, being prescribed as safe, should also be idempotent, as HTTP is a stateless protocol.
- Method POST is not necessarily idempotent, and therefore sending an identical POST request multiple times may further affect state or cause further side effects

4. Design and document ingoing representations

(*Ingoing representations = data sent by the Client to the Server*)

- path parameters
  - Only for resource indentifiers.
- query parameters
  - Usually for GET requests.
- body data
  - Only for POST and PUT requests.
  - Usually in the same format as the outgoing representations (responses from the server).
- header parameters
  - Typically, authentication tokens.

4. Design and document outgoing representations

(*Outgoing representations = data sent back by the Server to the Client*)

- Representations should be human-readable, but computer-oriented
- Representations should be useful, they should expose interesting data.
  - A single representation should contain all relevant information necessary to fulfil a need
  - A client should not have to get several representations of the same resource to perform a single operation
- Nowadays, JSON is the most used format.

5. Use status codes and headers meaningfully: Successes

- Make sure that clients requests are correctly turned into responses
  - Response code
    - Examples: **200** (“OK”), **201** (“Created”)
  - Some HTTP headers
    - Example: **Content-Type** is **image/png** for a map image
  - Representation
- A set of headers should be considered to save client and server time and bandwidth
  - Conditional HTTP GET for requesting resources that are frequently requested and invariable (**Last-Modified and If-Modified-Since** headers)

5. Use status codes and headers meaningfully: Errors

- Make sure that clients receive clear information in case of error
  - Response code
    - Examples: **3xx**, **4xx, 5xx**
  - Some HTTP headers
  - A document describing an error condition

5. Use status codes and headers meaningfully: Errors

Learn to use HTTP Standard Status Codes

```
100 Continue
200 OK
201 Created
202 Accepted
203 Non-Authoritative
204 No Content
205 Reset Content
206 Partial Content
300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
305 Use Proxy
307 Temporary Redirect
```

4xx Client's fault

```
400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Request Entity Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
```

```
500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported
```

5xx Server's fault

5. Use status codes and headers meaningfully: Errors

Most Common Errors in API documentation

Error	When?
400 Bad Request	Missing required parameters Incorrect parameter formats/types Inconsistencies between parameters, e.g. end_date < start_date
401 Unauthorized	Missing or incorrect authentication (tokens, api-keys, etc.)
403 Forbidden	Privilege not granted Example: Remove someone else's comment
404 Not Found	Retrieving/Updating/Deleting and unknown individual, i.e. there is no individual with the provided id GET /tweets/4343
409 Conflict	Business rule violation Example: Trying to vote a comment twice

6. Document the Resource API using a standard notation

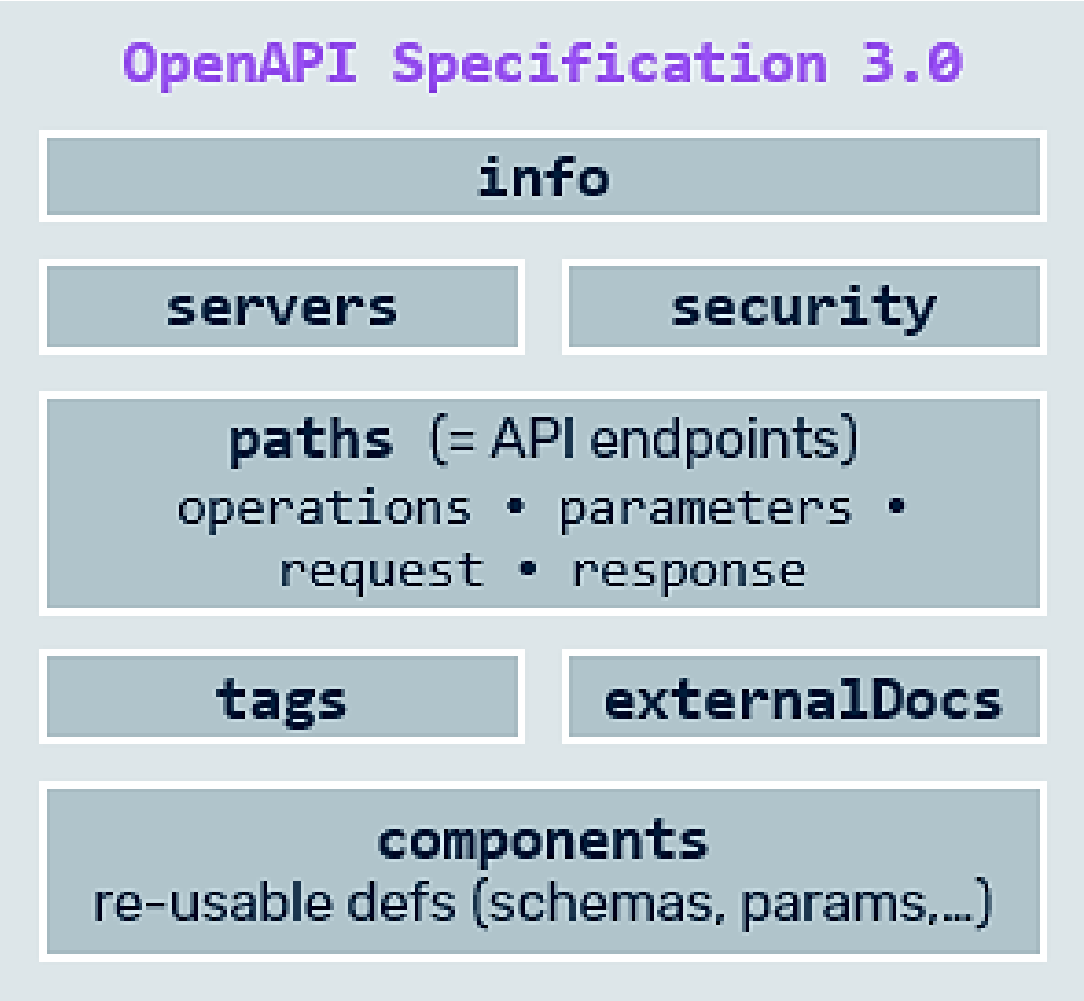


Image from https://techradar.softwareag.com/technology/openapi/

serialized in either
JSON or YAML

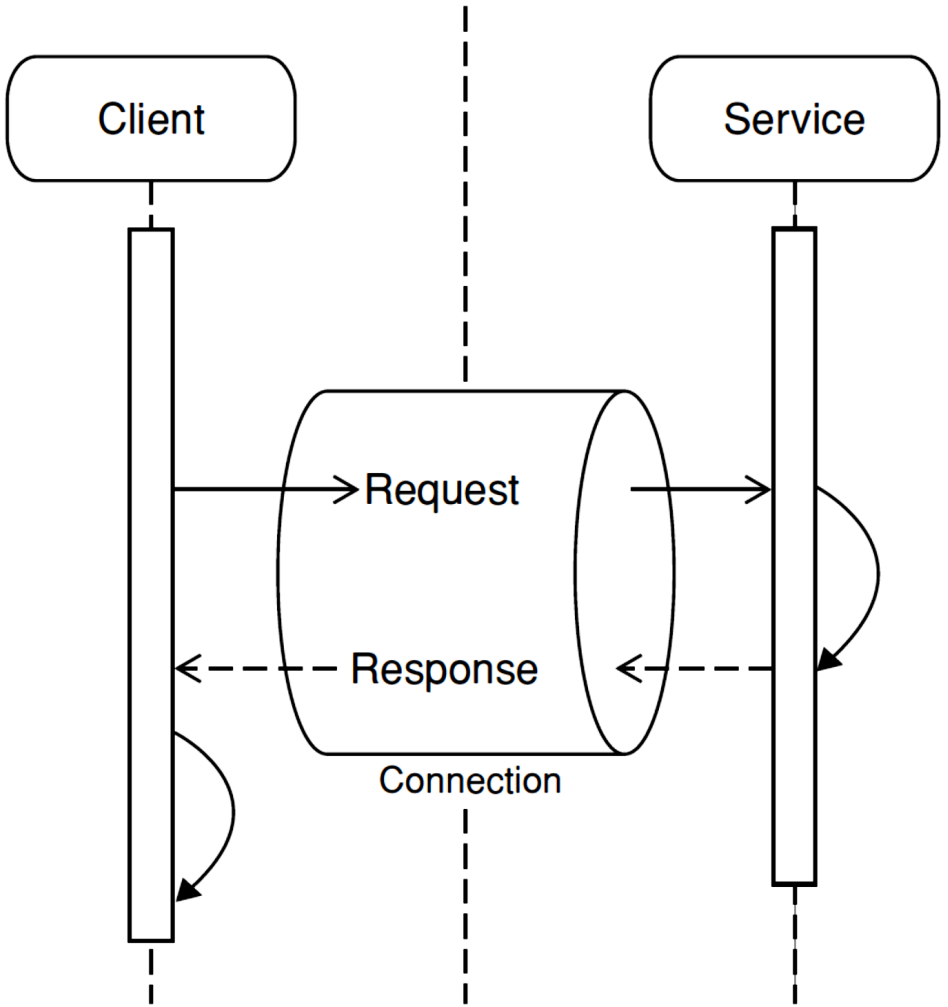
HTTP, OAuth2, JWT[3]

- synchronous calls
- call backs

RESTful Web Services: Refining the interaction

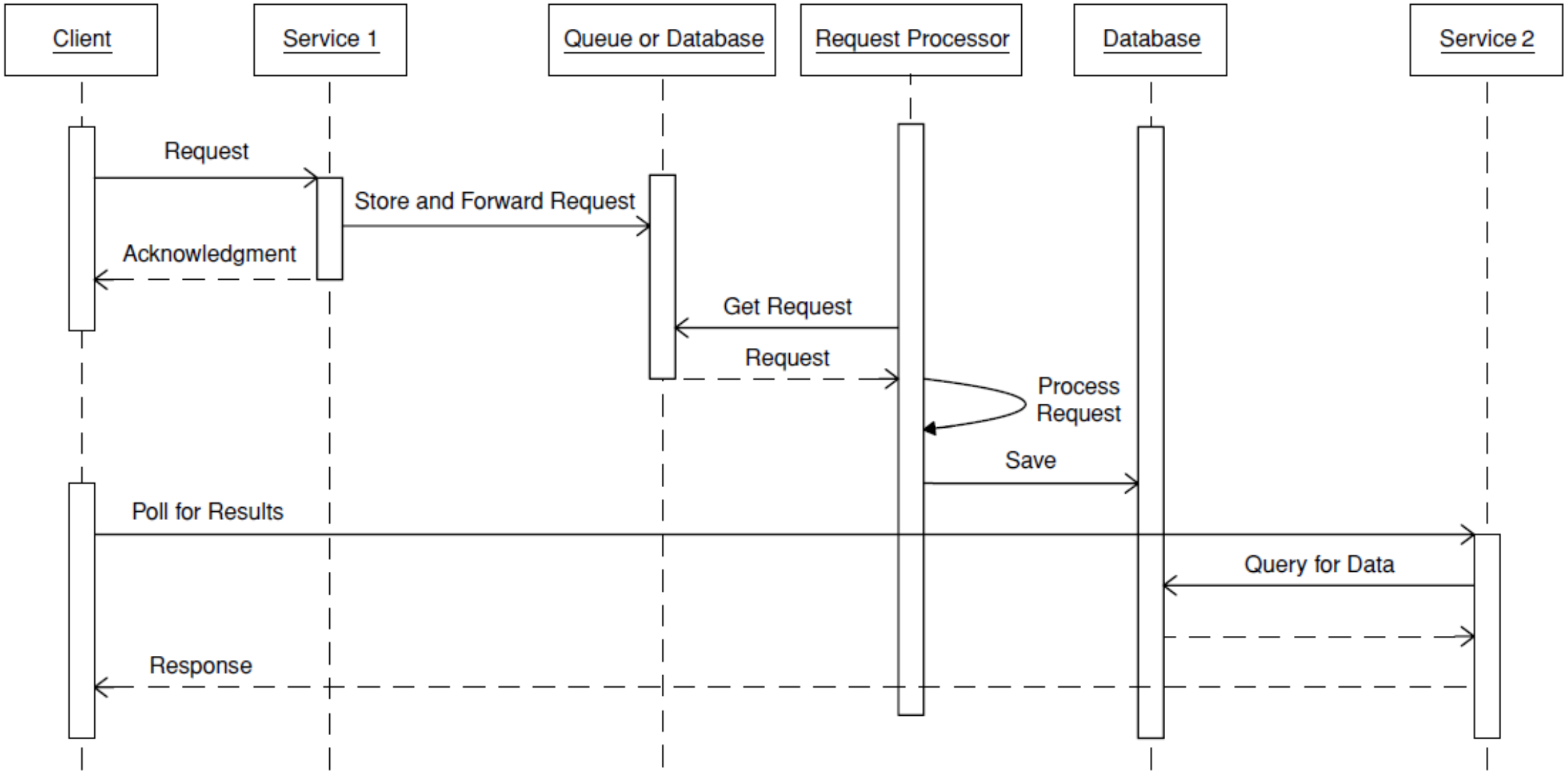
Pattern Name	Problem	Description
Request/ Response	What's the simplest way for a web service to process a request and provide a result?	Process requests when they're received and return results over the same client connection.
Request/ Acknowledge	How can a web service safeguard systems from spikes in request load and ensure that requests are processed even when the underlying systems are unavailable?	When a service receives a request, forward it to a background process, then return an acknowledgment containing a unique request identifier.
Media Type Negotiation	How can a web service provide multiple representations of the same logical resource while minimizing the number of distinct URIs for that resource?	Allow clients to indicate one or more media type preferences in HTTP request headers. Send requests to services capable of producing responses in the desired format.
Linked Service	Once a service has processed a request, how can a client discover the related services that may be called, and also be insulated from changing service locations and URI patterns?	Only publish the addresses of a few root web services. Include the addresses of related services in each response. Let clients parse responses to discover subsequent service URIs.

Refining the interaction: Request/Response Pattern



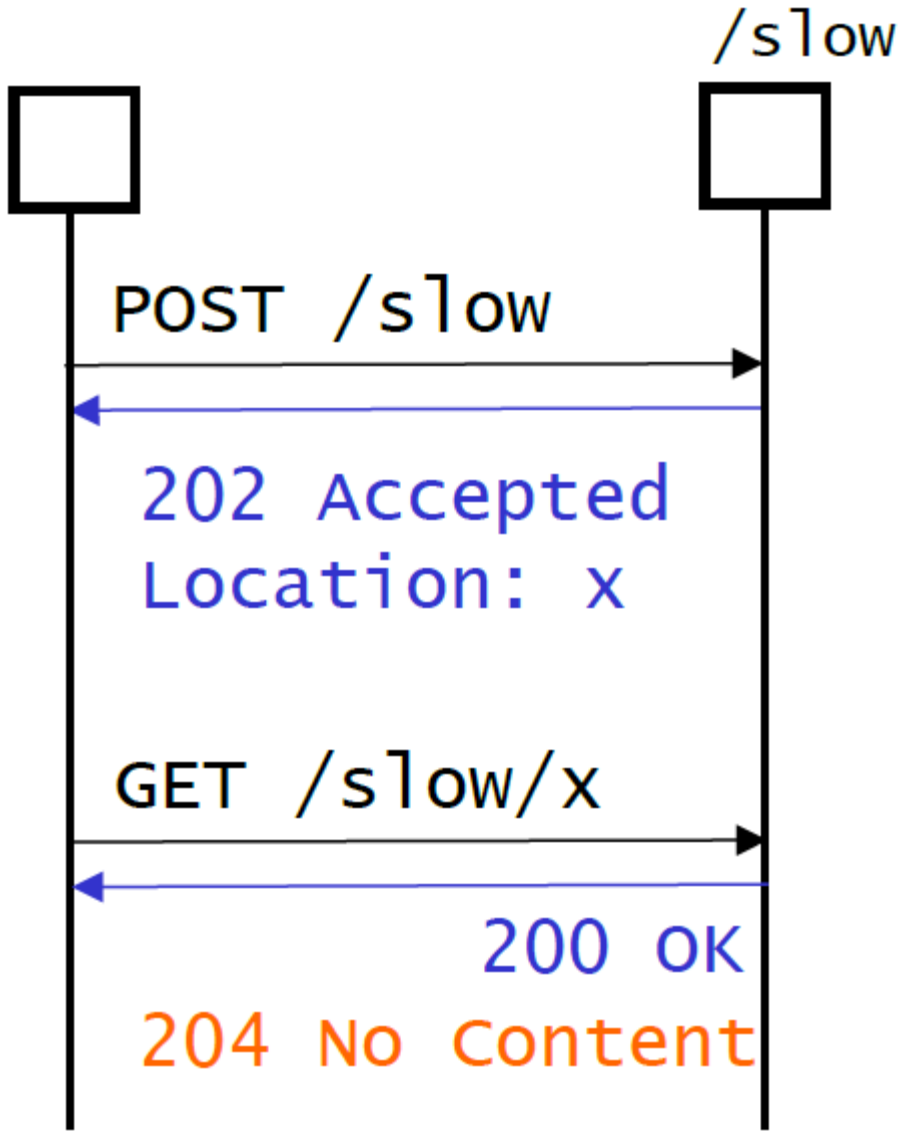
* Extracted from: Robert Daigneau. Service Design Patterns. Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Addison Wesley, 2012

Request/Acknowledge Pattern: Poll Variation

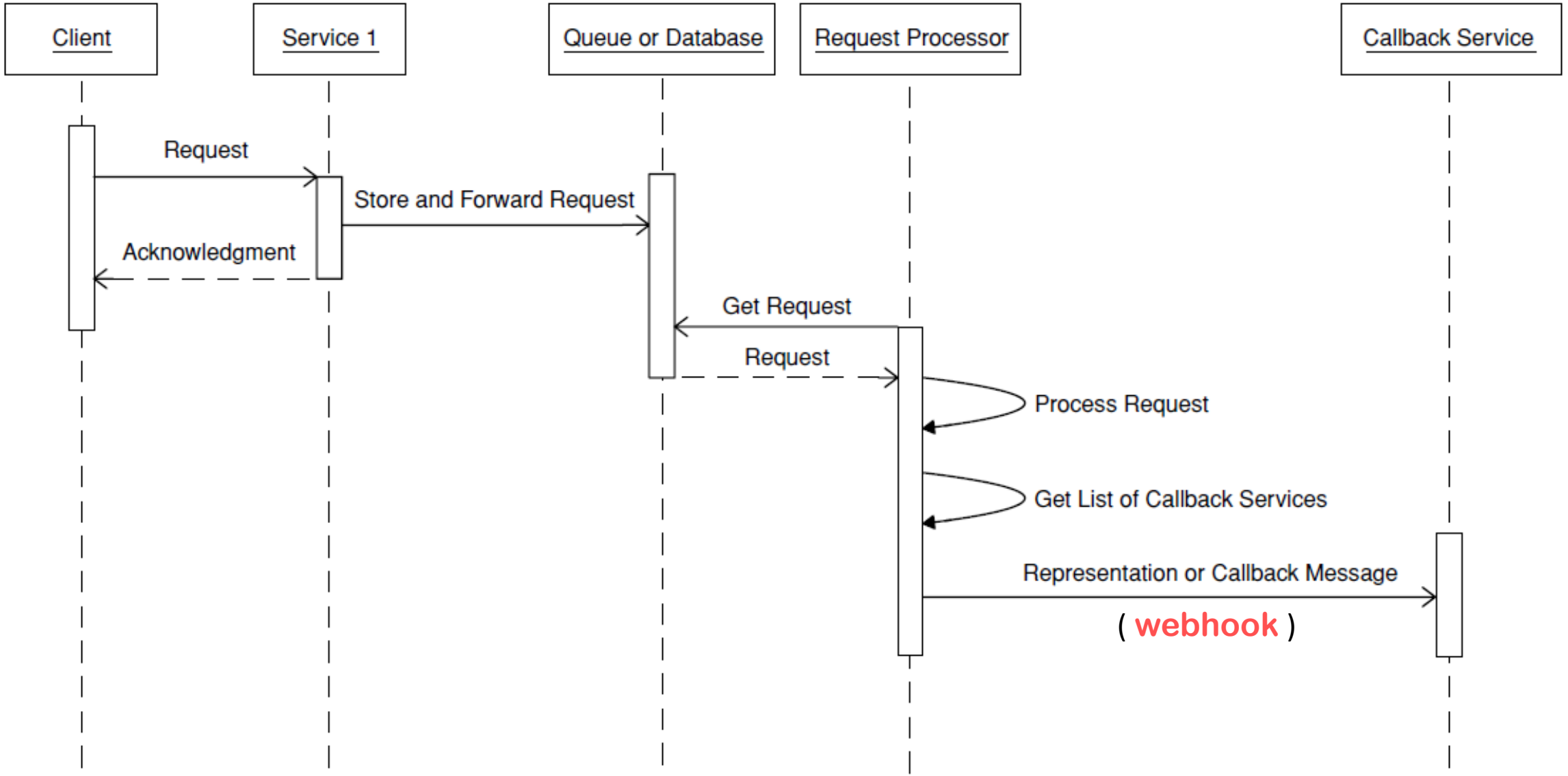


Poll Variation: REST Example

- The server may answer it with 202 Accepted providing a URI from which the response can be retrieved later
- Problem: how often should the client do the polling? /slow/x could include an estimate of the finishing time if not yet completed



Request/Acknowledge Pattern: Callback Variation



Refining the interaction: Media Type Negotiation

- Web services that are used by large and diverse client populations must often accommodate different media type preferences.
- There are many ways for a client to convey its preferences
  - Forced Media Type Negotiation
  - Media Type Negotiation
  - Advanced Media Type Negotiation

Refining the interaction: Forced Media Type Negotiation Pattern

- The specific URI points to a specific representation format using the postfix (extension)
- Warning: This is a conventional practice, not a standard.
- What happens if the resource cannot be represented in the requested format?
- A URI should be used to represent distinct resources (not distinct formats of the same resources).

```
GET /resource.html

GET /resource.xml

GET /resource.json
```

Refining the interaction: Media Type Negotiation Pattern

- Negotiating the message format does not require to send more messages
- The client lists the set of understood formats (MIME types)

```
GET /resource
Accept: text/html,
application/xml,
application/json
```

- The server chooses the most appropriate one for the reply (status 406 if none can be found)

```
200 OK
content-Type:
application/json
```

Refining the interaction: Advanced Media Type Negotiation Pat.

- Quality factors allow the client to indicate the relative degree of preference for each representation. If q=0, then content with this parameter is not acceptable for the client.

```
Media/Type; q=X
```

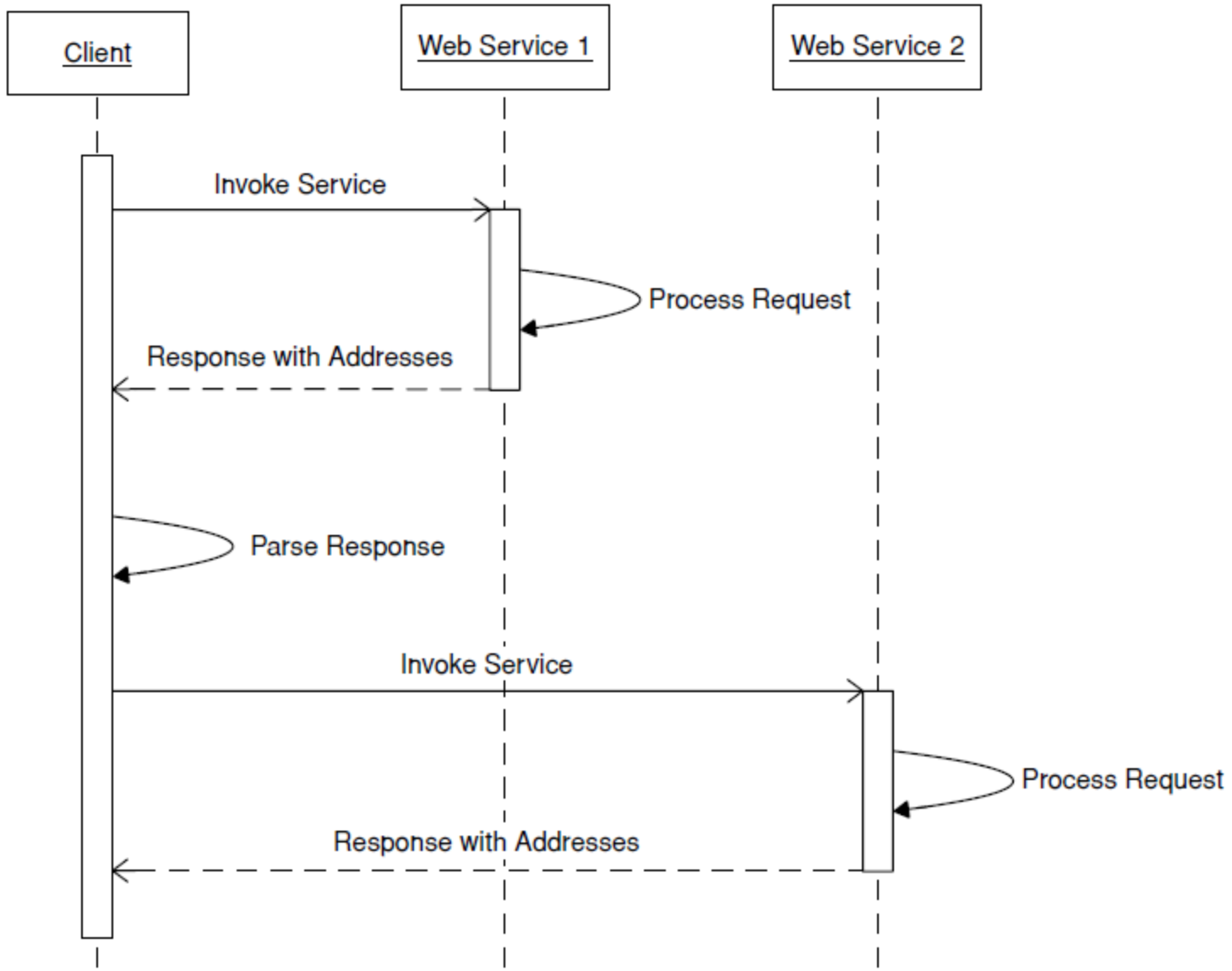
- Ex1: The client prefers to receive HTML (but any other text format will do with lower priority)

```
Accept: text/html, text/*;
q=0.1
```

- Ex2: The client prefers to receive XHTML, or HTML if this is not available and will use Plain Text as a fallback

```
Accept:
application/xhtml+xml;
q=0.9, text/html; q=0.5,
text/plain; q=0.1
```

Refining the interaction: Linked Service Pattern



Refining the interaction: Linked Service Pattern

- Model relationships (e.g., containment, reference, etc) between resources with hyperlinks in their representations that can be followed to get more details
- **H**ypertext **A**s **T**he **E**ngine **O**f **A**pplication **S**tate (HATEOAS): representations may contain links to potential next application states, including direction on how to transition to those states when a transition is selected

Refining the interaction: HATEOAS Example

GET /tweets/391678195147079681

Accept: application/json

200 OK

Content-Type: application/json

{"created_at": "Sat Oct 19 21:32:13 +0000 2013",
"id": 391678195147079700,
"text": "bla, bla, bla",
"actions": [
{ "action": "Retweet",
"method": "post",
"href": "/tweets/391678195147079681/retweets" }
, …], … }

Richardson Maturity Model

- **Level 0: *HTPP Tunneling***
  - Single URI (endpoint). Variants:
    - POST + Payload: SOAP, XML-RPC, POX
    - GET uri?method=method_name&par1=val1&par2=val2 (Flickr)
- **Level 1: *URI Tunneling*** (Twitter v1.1)
  - Many URIs
  - Few verbs: GET, POST
- **Level 2: *CRUD API* ➔ *Resource API***
  - Many URIs: collections and individuals
  - Many verbs: GET, POST, PUT, DELETE
- **Level 3: *Hypermedia API***
  - Level 2 + HATEOAS

References

- Leonard Richardson and Sam Ruby. **RESTful Web Services**. O'Reilly, 2007
- Robert Daigneau. **Service Design Patterns. Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services.** Addison Wesley, 2012

- https://martinfowler.com/articles/richardsonMaturityModel.html