Politecnico di Milano
Department of Electronics, Information
and Bioengineering

HTTP protocol integration in PerLa

Project for Pervasive Data Management course

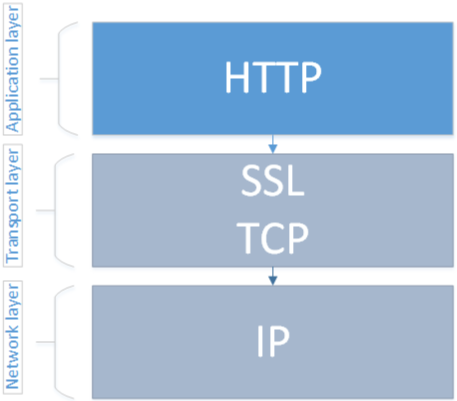
Federico Monterisi

June 9, 2014

Outline

HTTP protocol

*The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext [...] through extension of its **request methods**, **error codes** and **headers**.*



R. Fielding J. Gettys J. Mogul H. Frystyk L. Masinter P. Leach T. Berners-Lee
Hypertext Transfer Protocol – HTTP/1.1 1997: The Internet Society.
http://www.w3.org/Protocols/rfc2616/rfc2616.html

HTTP protocol | Message structure (1)

The request message is composed by **request line** (method, URI and protocol version), **header** (informations about the client) and **body**.

Request message

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: 60
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

first=Zara&last=Ali
```

HTTP protocol | Message structure (2)

The response message is composed by **status line** (protocol version and status code), **header** (informations about the server) and **body**.

Response message

```
HTTP/1.1 200 OK
 Date: Mon, 27 Jul 2009 12:28:53 GMT
 Server: Apache/2.2.14 (Win32)
 Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
 Content-Length: 88
 Content-Type: text/html
 Connection: Closed

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

Outline

REST in HTTP protocol

REST architecture born for every communication protocols, but the major success case is its enforcement with HTTP protocol.
REST services use URI as a pointer to managed **resource** or to **collection of resources**.
For example a resource could be pointed by URI `http://myservice.com/api/resource/id-res` and its collection by `http://myservice.com/api/resource/`.
The performable actions on these resources are specified by HTTP method GET, POST, PUT, DELETE.

REST in HTTP protocol | GET, POST

GET
: Get the resource, so the body of HTTP response contains the model of resource. No content-type is required in HTTP request because all necessary information are retrieved by URI.
Multiple GET operations do not change server state.

POST
: Usually create and add a resource to collection. So body of HTTP request is required and probably also the content-type. HTTP response contains the resource aligned to server state (ex. server add an identifier to resource).

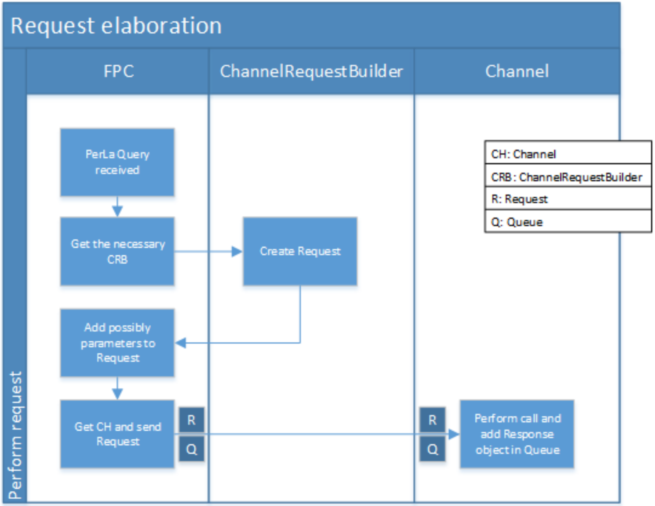
REST in HTTP protocol | PUT, DELETE

PUT Put the resource and replace it. Body in HTTP request is required, but not in HTTP response. Client already known the resource, it needs just if operation ends correctly watching Status Code.

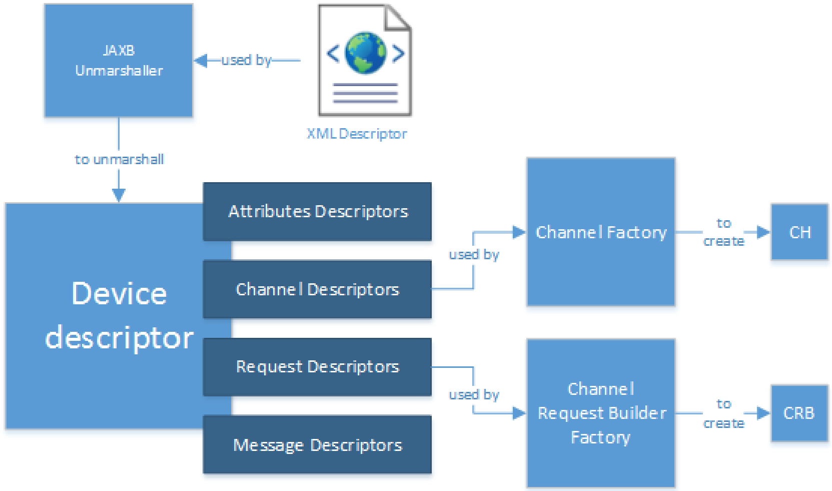
DELETE Delete the resource. So it is necessary just the URI of the resource nor body in the request, nor in the response.

Outline

PerLa channel design | Channels and requests

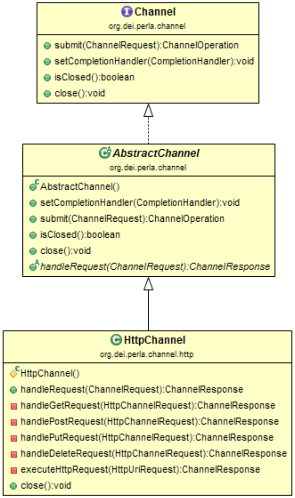


PerLa channel design | Descriptors and factories



Outline

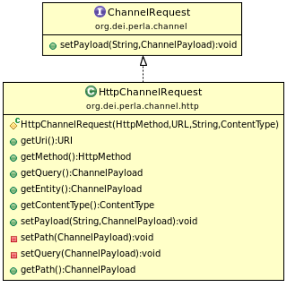
HTTP channel | Channel and request (1)



Channel
org.dei.perla.channel
submit(ChannelRequest):ChannelOperation
setCompletionHandler(CompletionHandler):void
isClosed():boolean
close():void
AbstractChannel
org.dei.perla.channel
AbstractChannel()
setCompletionHandler(CompletionHandler):void
submit(ChannelRequest):ChannelOperation
isClosed():boolean
close():void
handleRequest(ChannelRequest):ChannelResponse
HttpChannel
org.dei.perla.channel.http
HttpChannel()
handleRequest(ChannelRequest):ChannelResponse
handleGetRequest(HttpChannelRequest):ChannelResponse
handlePostRequest(HttpChannelRequest):ChannelResponse
handlePutRequest(HttpChannelRequest):ChannelResponse
handleDeleteRequest(HttpChannelRequest):ChannelResponse
executeHttpRequest(HttpUriRequest):ChannelResponse
close():void

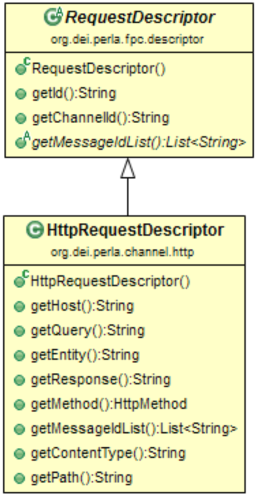
- Invoked by FPC using method `submit` that, consumed `HttpChannelRequest`, returns a `ChannelOperation` containing the logic with call result.
- `handleRequest` method dispatches the GET, POST, PUT and DELETE `HttpChannelRequest`, respectively, to `handleGetRequest`, `handlePostRequest`, `handlePutRequest`, `handleDeleteRequest`
- For a simple and standard implementation it has been used Apache HTTP Component library.

HTTP channel | Channel and request (2)


ChannelRequest
org.dei.perla.channel
setPayload(String,ChannelPayload):void
HttpChannelRequest
org.dei.perla.channel.http
HttpChannelRequest(HttpMethod,URL,String,ContentType)
getUri():URI
getMethod():HttpMethod
getQuery():ChannelPayload
getEntity():ChannelPayload
getContentType():ContentType
setPayload(String,ChannelPayload):void
setPath(ChannelPayload):void
setQuery(ChannelPayload):void
getPath():ChannelPayload

Except HTTP method, a part of Uri and Content-type, defined in the XML descriptor, the others are setted dynamically by FPC using `setPayload` method. It uses a string as identifier, for query url, additional uri path and entity, and `ChannelPayload` object, containing the value.
Query url and uri path are encapsulated in Uri parameter so `HttpChannel` can be used it already formatted.

HTTP channel | Request descriptor (1)



`id`	is a string identifier of request. Required
`channel-id`	is a string identifier of channel sending this request. Required
`host`	is the host for sending HTTP request. It is accepted also a complex url like `http://mysite.com/one/path?q=i` Required
`path`	is the identifier of tag `message` that represent a dynamic path. Optional
`query`	is the identifier of tag `message` that represent a dynamic query. Optional
`entity`	is the identifier of tag `message` that represent the entity (content) of HTTP request Required for POST and PUT request
`method`	is an enumeration value that specifics HTTP method for the request (get, post, put or delete). Optional, default is get
`content-type`	is the content-type specified in HTTP request. Optional, default is */* (Known as *wildcard* content-type)
`response`	is the identifier of tag `message` that represent the response content of HTTP request Required for post and get request

HTTP channel | Request descriptor (2)

Http Request XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<device xmlns:ht=
    "http://perla.dei.org/channel/http">
  <channels>
    <ht:channel id="http_ch_01" />
  </channels>
  <requests>
    <ht:request id="post_req"
      channel-id="http_ch_01"
      method="post"
      host="http://mysite.com"
      path="req_path_id"
      query="req_query_id"
      response="req_response_id"
      content-type=
        "application/x-www-form-urlencoded"
      entity="req_entity_id"/>
  </requests>
</device>
```

HTTP channel | CRB factory

`HttpChannelRequestBuilderFactory` must validates the XML `request` tag an checks its consistency with REST architecture. So it creates `HttpChannelRequestBuilder`.

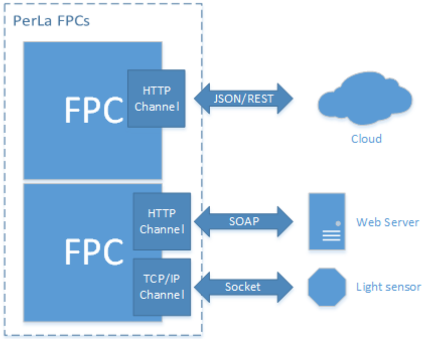
Allowed, not allowed and mandatory attributes

`method`	`response`	`content-type`	`entity`
GET	M	NA	NA
POST	M	M	M
PUT	NA	M	M
DELETE	NA	NA	NA

Notice, attributes `id`, `channel-id` and `host` are always mandatory while attributes `path` and `query` are always allowed.

Outline

Conclusion and a possible scenery



This implementation allows an integration of PerLa with SOAP and JSON/REST services. These technologies could utilized the same Channel despite the REST orientation of HTTP Channel code.

Conclusion and a possible scenery

Thinking about public transport in a city, PerLa takes information about the bus time-table, through a Web Service, and can:

- notify if the next is the last ride
- suggest to use bus (maybe with a touristic ride) or underground depending on weather