Tutorial

WS 16/17

Software Service Engineering 9th Tutorial

Mahda Noura
Technische Universität Chemnitz
Department of Computer Science
Distributed and Self-organizing Systems Group

http://vsr.informatik.tu-chemnitz.de

Exercise 1

- Answer the following questions:
  - What are the goals of RESTful architectures? Which properties do the resulting systems possess?
  - What are the REST constraints? Which properties do they induce?
  - What is the “Hypermedia” constraint and which examples of its application do you know?

REST Architecture style

- What is an architecture style?

  An architectural style is a coordinated **set of architectural constraints** that restricts the roles/features of architectural elements and the **allowed relationships** among those elements within any architecture that conforms to that style.

- Why someone decides to adhere to certain architecture style?

REST Architecture style

- What are the REST constraints?
  - Client-Server
  - Stateless
  - Cache
  - Uniform interface
    - Identification of resources
    - Manipulation of resources through representations
    - Self-descriptive messages
    - Hypermedia as the engine of application state
  - Layered system
  - Code-on-demand

REST Architecture style

- Client-Server
  - Portability (server responses are understood by all clients)
  - Scalability (server outsources part of logic to client)
- Stateless
  - Visibility (request analysis is possible)
  - Reliability (request can be easily repeated in case of failures)
  - Scalability (no persistent storage on the web needed)
  - But: amount of data transferred increases, correct client implementation required
- Cache
  - Scalability (servers are requested not so often)
  - But: reliability can decrease (if cached data differs from original)

REST Architecture style

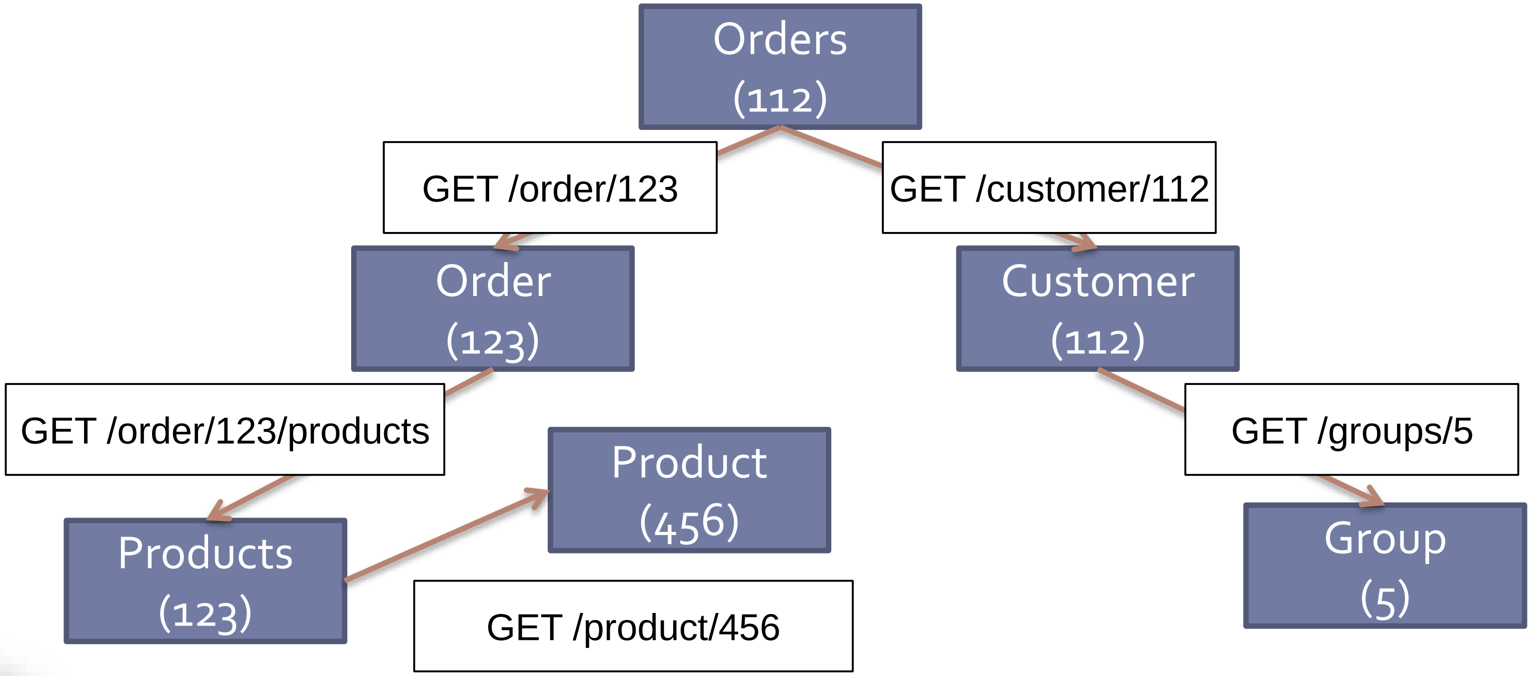
- Uniform interface
  - Simplicity (one protocol to learn)
  - Visibility (everyone understands the protocol)
  - Independent evolvability (implementations behind interface can change)
  - But: performance drawbacks because of interface transformation
- Layered System
  - Simplicity (layering of services, request optimization)
  - Scalability (load balancing)
  - But: additional latency
- Code-on-Demand
  - Simplicity (clients can be enriced with functionality)
  - But: reduced visibility (code can be arbitrary)

Examples

- How data is organized in a RESTful architecture?
  - Resources
    people, cities, countries, Chemnitz, Germany
  - Resource identifiers
    /people, /citites, /countries, /cities/1, /countries/germany
  - Resource metadata
    Chemnitz is located in Germany
  - Resource representation
    <city name=„Chemnitz" url=„/cities/1"><citizens>243.173</citizens></city>
  - Representation metadata
    Format: application/xml, Created: 2012-01-10
  - Control data
    Action: read, Cache: disabled

Hypermedia Constraint

- Part of the uniform interface constraint
- Resource representations should expose further possible interactions:



Hypermedia Constraint

- Content-Negotiation:
  - Clients indicate their preferences on representation format, language, time of the representation:
    - Accept: text/html;
    - Accept-Language: de;
    - Accept-Datetime: Thu, 31 May 2007 20:35:00 GMT
  - Servers responds according to the client request

Exercise 2

- Convert the following fictitious SOAP-based service into a REST/HTTP-based one.

SOAP Operation	HTTP Verb	URL	Request Header	Response Header	Request Body
getAllBooks	GET	/books	Accept: application/xml or Accept: application/json	Content-Type: application/xml Content-Length: 123	-
getBookById(bookId: int; language: {en, de, fr, nl})	GET	/books/{bookId}	Accept-Language: en-US Accept: application/xml	Content-Type: application/xml Content-Length: 123	-
updateBook(bookId: int; book: (title, authors))	PUT	/books/{bookId}	Content-Type: application/xml Accept: application/xml	Content-Type: application/xml Content-Length: 123	<book> <title>…</title> <authors>…</authors> </book>
getAllCategories	GET	/categories	Accept: application/xml	Content-Type: application/xml	-
getBooksInCategory (categoryId: int)	GET	/categories/{categoryId}/books	Accept: application/xml	Content-Type: application/xml	-
addBook(categoryId: int; book: (title, authors))	POST	/categories/{categoryId}/books	Content-Type: application/xml	Location: /books/32	<book> <title>…</title> <authors>…</authors> </book>
getBookImage(bookId: int);	GET	/books/{bookId}	Accept: image/png	Content-Type:image/png	-
searchForBooksByKeyword(keyword: string)	GET	/books?keyword={keyword}	Accept: application/xml	Content-Type: application/xml	-

Exercise 2

SOAP Operation	HTTP Verb	URL	Request Header	Request Body
getAllBooks	GET	/books	Accept: */*	-
getAllBooksAsJson	GET	/books	Accept: application/json	-
getBookById(bookId: int; language: {en, de, fr, nl})	GET	/books/{bookId}	Accept-Language: en-US	-
updateBook(bookId: int; book: (title, authors))	PUT	/books/{bookId}	Content-Type: application/xml	<book> <title>…</title> <authors>…</authors> </book>
getAllCategories	GET	/categories	Accept: */*	-
getBooksInCategory (categoryId: int)	GET	/categories/{categoryId}/books	Accept: */*	-
addBook(categoryId: int; book: (title, authors))	POST	/categories/{categoryId}/books	Content-Type: application/xml	<book> <title>…</title> <authors>…</authors> </book>

Exercise 3

- In the template BookmarkService.zip you will find a REST/HTTP Web service operating on a single resource – User = (int id, string name). Extend the service towards management of user bookmarks.
  1. Extend the service to operate on a resource Bookmark = (int id, string url). Following operations should be implemented:
     - Read all bookmarks / Search bookmarks by keyword
     - Create a bookmark assigned to a user
     - Delete a bookmark
     - Read bookmarks of a given user
  2. Extend the service client from the BookmarkServiceClient subproject to consume the above operations

Exercise 4

- Using the template *LibraryServiceAndClient.zip* implement a console application to borrow a book in a library. The library provides a RESTful Web service to retrieve the information about books, about bookings and functionality to borrow a book. Complete the implementation of the client proxy.
  - Inspect the service behavior using a REST client (e.g. Chrome Advanced REST client or Firefox REST client)
  - Use hypermedia to discover and interact with resources
  - Test your implementation using the pre-built scenario.

Homework

- Extend the service and the client from the task 3 with the following functionality:
  - Bookmarks should be able to be assigned to *Categories* = (***int*** *id*, ***string*** *name)*.
  - All categories can be listed
  - Bookmarks of a given category can be listed
  - Categories can be searched by a keyword