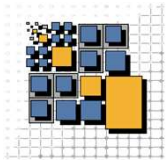
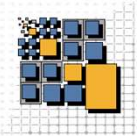


DSG-SOA-M 2024:
- REST and RESTful Services -

Dr. Andreas Schönberger

Distributed Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

What is REST?

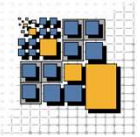


- Roy Thomas Fielding described the “REST architectural style” in his PhD thesis:
  “Architectural Styles and the Design of Network-based Software Architectures”, University of California, Irvine, 2000
- REST = Representational State Transfer
  → *describes the principles of Web-based applications*
- Fielding designed HTTP and URI specifications
  (together with Tim Berners-Lee)
- The REST hype began around 2007:

→ Why then did it take so long (since 2000) until RESTful Web Services got broad attention by industry and academia?

- REST **IS NOT** a programming / services framework or a standard to be implemented.
  There are just many frameworks that claim to be RESTful.
- RESTful Web Services got attention as replacement for WSDL/SOAP Web Services

REST Principles and Misunderstandings



→ *Create a **uniform interface** for accessing web resources by means of*

- Addressable resources
- Representation-oriented manipulation of resources
- Self-Descriptive Messages
- Stateless communication
- HATEOAS
  (Hypermedia as the engine of application state)

nice read:
http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

pronounced either "hideous" or "hate yo' ass“

Things that Fielding did not say in his thesis:

- Mandatory use of HTTP as protocol (although frequently used)
  → and he did not define Lo-REST and Hi-REST
- Use an URI scheme to create an API
- Use REST as a replacement for
  Message Queueing or RPC

REST Overview

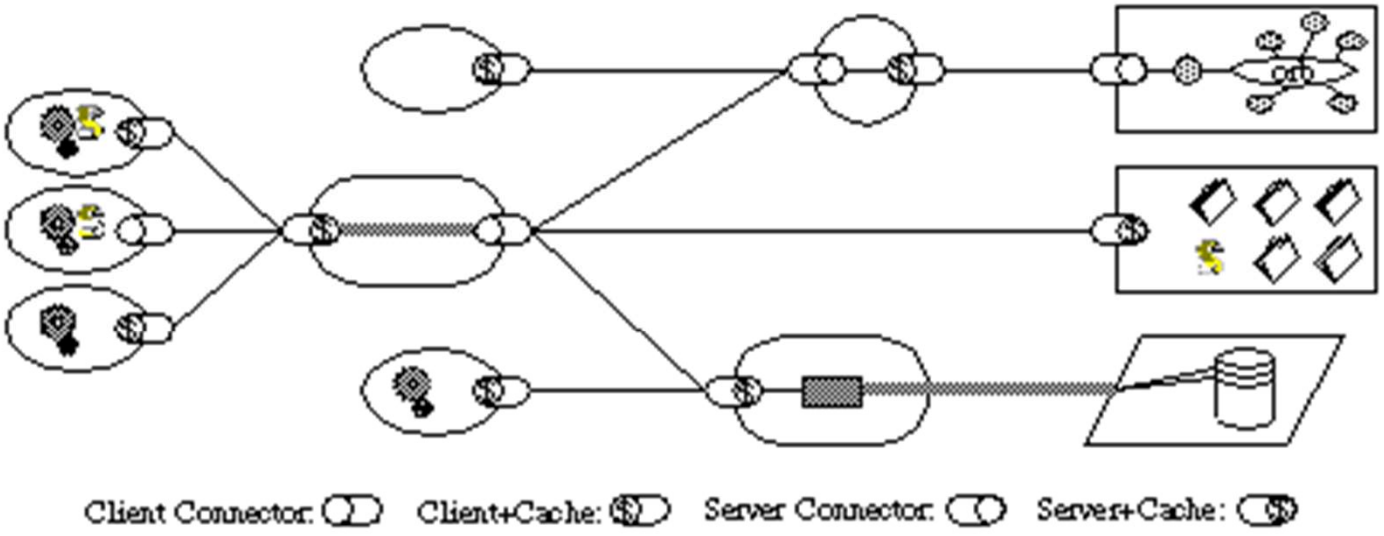
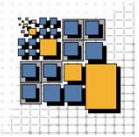


Figure 5-8. REST

Roy T. Fielding: ***"Architectural Styles and the Design of Network-based Software Architectures", University of California, Irvine, 2000***

REST – Addressable Resources



- *Resources* are an abstraction for the entities you interact with
- URIs are used for identifying resources

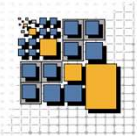
  `protocol://host:port/path?queryStr#anchor`

  - `protocol`: Names the protocol such as HTTP/S, FTP, …
  - `host`: IP address or DNS name
  - `port`: well, the port, default values depend on the protocol
  - `path`: "/"-delimited series of path elements
  - `queryStr`: parameters as "&"-delimited series of name=value pairs
  - `anchor`: reference in the respective resource

Why that?

→ Standardized and universal way of addressing, i.e., resource identification also possible on intermediaries

Non-Example: Addressable Resources

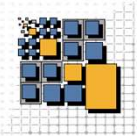


Look at the following link:

doors://mydoorshost.com:36683/?version=2&prodID=0&view=00000002&urn=urn:teleste::1-4dac6b3c0b7e165e-O-3-4a7a383010bc0

What is wrong with this?

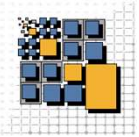
REST – Representation Orientation



Fielding: *"REST components perform actions on a resource by using a representation to capture **the current or intended state of that resource** and transferring that representation between components. A representation is **a sequence of bytes**, plus representation **metadata to describe those bytes**."*

- Metadata means
  - Control data determines the purpose of a message, e.g., an HTTP method (GET, POST, etc.) or a parameter
    → NOT really specific to a user application
  - Data format is defined by a media type, e.g., a mime type
    → NOT really specific to a user application
- This is different from WSDL services where you identify actions by means of portType and operation!

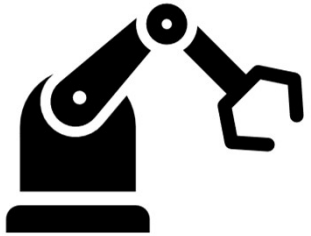
Non-Example: Representation Orientation



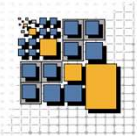
Let’s assume you are designing a remote control service for a machine.

Among other things you can turn on/off an express mode.

Why not offering an operation such as toggleExpress() ?



REST – Self-Descriptive Messages

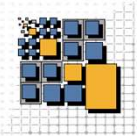


Fielding: *"REST constrains messages between components to be self-descriptive in order to support intermediate processing of interactions."*

This amounts to:

- Complete addressing information
  → do not assume sessions or environments that implicitly identify the receiver of a message
- Explicit information about content encodings
  → by means of content encoding identifiers, e.g., `transfer-encoding`
- Independence of underlying transport protocol
  → provide information about message boundaries in the message
- Cache control
- Content negotiation
  → for selecting from multiple available contents (representations)

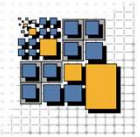
Non-Example: Self-Descriptive Messages



```
UNH+ME0000001+ORDERS:D:93A:UN'
BGM+220+123321'
DTM+137:990312:101'
DTM+2:990503:101'
NAD+BY+++Institute of Software Technology:and Interactive
Systems:Vienna University of Technology+Favoritenstraße 9-11/188-
3+ Vienna++1010+AT'
CTA+PE:HH:Hugo Heuschreck'
NAD+SE+++Hard & Software GmbH+Wiedner Hauptstrasse 12/8+Vienna++
1040+AT'
TAX+7+VAT+++20
CUX+2:ATS:9'
LIN+1++34567892189:EN::9'
QTY+21:3:EA'
PRI+AAA:200000:PE'
LIN+2++98754390211:EN::9'
QTY+21:10:EA'
PRI+AAA:40000:PE'
UNS+S'
MOA+86:1200000'
UNT+18+ME0000001'
```

Sample EDI message for ordering electronics
By courtesy of: Marco Zapletal, TU Vienna

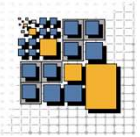
REST – Stateless Communication



Fielding: *"All REST interactions are stateless. That is, each request contains all of the information necessary for a connector to understand the request, independent of any requests that may have preceded it."*

- Thus
  - No (session) state to be maintained on the server's side
    → makes scalability by far easier
  - Less constraints for caching responses
  - Less frequently requires synchronization for concurrent requests
- But what about requests that require shared knowledge between services
  - What about using a merchant id for calling the coupon generation service? #bamberggutschein
  - And what if we hand over a merchant id to an invoice generation service?

REST – HATEOAS

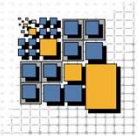


- HATEOAS = Hypermedia as the engine of application state
  - Interactions are data driven and document centric
  - Links are used to connect data

→ Clients discover from the content what they can do with it:

- Content types must be well-known **or**
- Clients have some sort of artificial (or natural) intelligence
- Clients DO NOT rely on a static interface description for determining admissible interactions

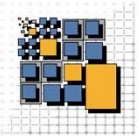
HTTP as a REST Stereotype



- Fielding did not postulate HTTP is the one and only REST implementation in his thesis, but HTTP is frequently used as an example by the REST community
- HTTP just uses four verbs for interaction
  - GET, PUT, POST, DELETE
    (HEAD, OPTIONS, TRACE, CONNECT are rather disregarded)
- However, not every single HTTP detail complies with REST

Fielding: *"An example of where an inappropriate extension has been made to the protocol to support features that contradict the desired properties of the generic interface is the introduction of site-wide state information in the form of HTTP cookies [73]. Cookie interaction fails to match REST's model of application state, often resulting in confusion for the typical browser application."*

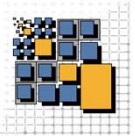
REST as Silver Bullet?



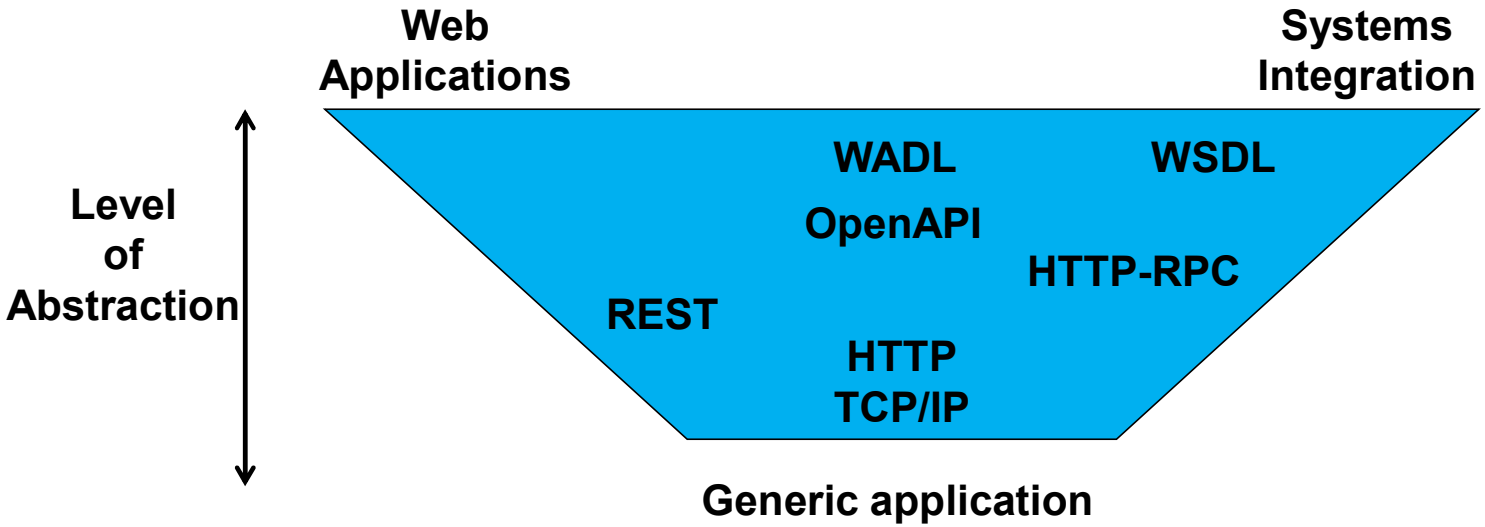
REST is not meant to solve all kinds of distributed system interaction!

Fielding: *"The REST interface is designed to be efficient for large-grain hypermedia data transfer, optimizing for the common case of the Web, but resulting in an interface that is not optimal for other forms of architectural interaction."*

So, what Web Standards Shall we Use?

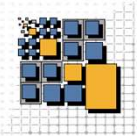


- Depends on the abstraction layer and the application scenario



- Fielding: *"What makes HTTP significantly different from RPC is that the requests are directed to resources using a generic interface with standard semantics that can be interpreted by intermediaries almost as well as by the machines that originate services. The result is an application that allows for layers of transformation and indirection that are independent of the information origin, which is very useful for an Internet-scale, multi-organization, anarchically scalable information system. RPC mechanisms, in contrast, are defined in terms of language APIs, not network-based applications."*

Applying REST: Example I



What information should be in a QR code for a coupon?

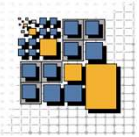
Answer A:
„DFRT-H78N“

Just a secure code that proves the owner of the code is the beneficiary of the code.

Answer B:
„https://portal.bamberg-gutschein.de/public/voucher-scan?voucherCode=DFRTH78N “

A URI that resolves to the actual coupon claim of the code

Applying REST: Example II



Let‘s assume you have to implement a software engineering tool chain that integrates requirements management, issue tracking and test management.

What kind of links would you use for referencing requirements in an issue tracker?

Answer A:

Just use the standard ticket ids, everybody will copy the browser addressline anyway.
https://myredminehost.com/redmine/issues/195029

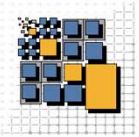
Answer B:

Use a built-in tool filter with semantic query parameters, it is built-in!
https://myredminehost.com/redmine/issues?set_filter=1&status_id=&cf_134=RQ_MYREQUIREMENT&cf_144=MYPROJECT&cf_147=MYSUBSYSTEM*

Answer C:

Use a URL rewriting scheme to clearly communicate requirement information.
https://myredminehost.com/redmine/MYSUBSYSTEM/RQ_MYREQUIREMENT/MYPROJECT

Summary



- REST is a paradigm, is a paradigm, is NOT HTTP
  - Applying REST principles is very helpful in a lot of system architecture / design situations
  - Remember the "*for large-grain hypermedia data transfer*" use case
  - Mind your system context and ask a view what-if questions!
  - Still, the available online resources on how to design RESTful HTTP interactions will help you with getting started
- It's not worthwhile to be religious about applying REST or even achieving a REST maturity level, let's rather talk about how sensible REST principles are for your system, e.g.
  - Session management for web interactions
  - Interactions with scarce resources

  → Come up with scenarios where you want to break REST principles!