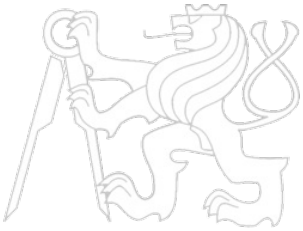
Middleware and Web Services

Lecture 2: Service Architecture and Technologies

doc. Ing. Tomáš Vitvar, Ph.D.
tomas@vitvar.com • @TomasVitvar • http://vitvar.com



Czech Technical University in Prague
Faculty of Information Technologies • Software and Web Engineering • http://vitvar.com/courses/mdw

Modified: Sun Oct 14 2018, 19:49:10
Humla v0.3

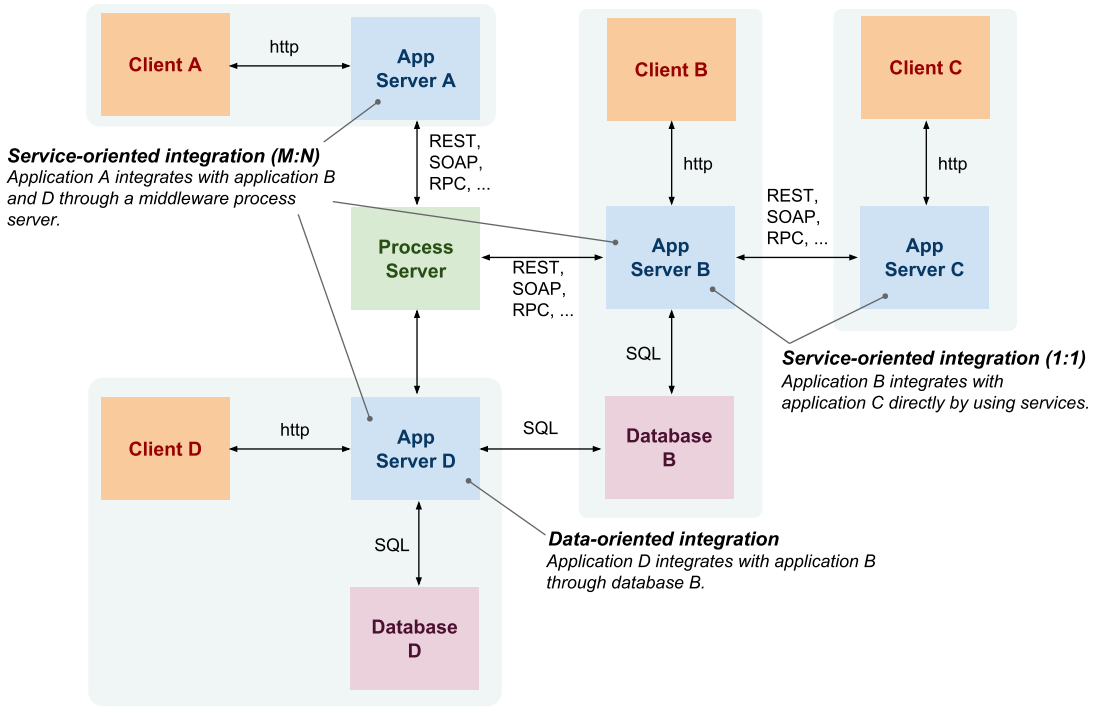
Overview

- Integrating Applications
- Service Definition
- Service Communication
- REST

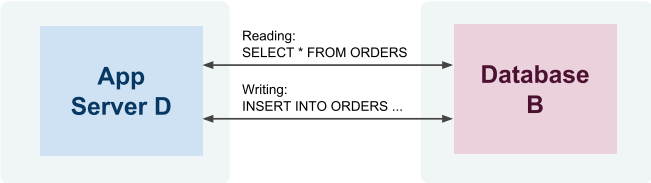
Integration and Interoperability

- Integration
  - *A process of connecting applications so that they can exchange and share capabilities, that is — information and functionalities.*
  - *Includes methodological approaches as well as technologies*
- Interoperability
  - *Ability of two or more applications to understand each other*
  - *Interoperability levels*
    - → *Data – syntax/structure and semantics*
    - → *Functions/Processes – syntax and semantics*
    - → *Technical aspects – protocols, network addresses, etc.*

Integration Approaches Overview

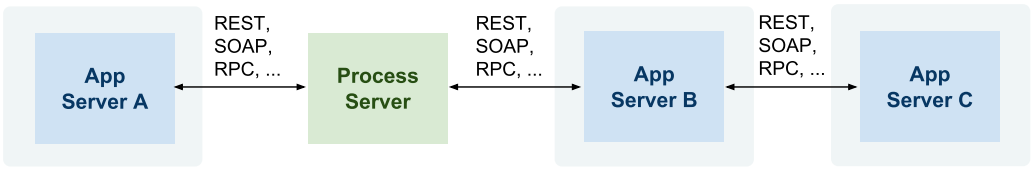


Data-oriented Integration



- Third-party database access
  - *Application D accesses a database of application B directly by using SQL and a knowledge of database B structure and constraints*
  - *In the past: monolithic and two-tier client/server architectures*
  - *Today: ETL (Extract, Transform, Load) technologies*
- Problems
  - *App D must understand complex structures and constraints*
    - → *Data – very complex, includes structure and integrity constraints*
    - → *Functions/processes – hidden in integrity constraints*
    - → *Technical – access mechanisms can vary*

Service-oriented Integration



- Integration at the application layer
  - *Application exposes services that other applications consume*
  - *Services hide implementation details but only define interfaces for integration*
- Problems
  - *Can become unmanageable if not properly designed*
  - *Interoperability*
    - → *Data – limited to input and output messages only*
    - → *Functions/processes – limited to semantics of services*
    - → *Technical – access mechanisms can vary*

Integration and Types of Data

- Transactional data – Web services
  - *Service-oriented integration*
  - *online, realtime communication between a client and a service*
  - *Usually small amount of data and small amount of service invocation in a process*
- Bulk data – ETL
  - *Data-oriented integration*
  - *processing of large amount of data in batches*
- **ESB provides both Web service and ETL capabilities**

Overview

- Integrating Applications
- Service Definition
- Service Communication
- REST

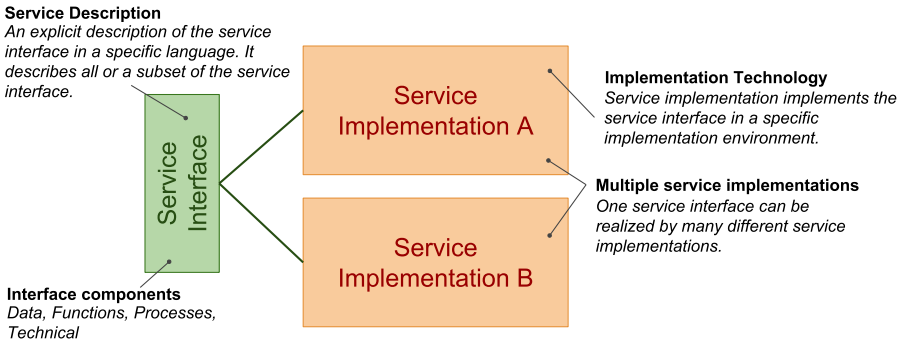
Web Service Architecture

- Web Service Architecture
  - *Defined by W3C in Web Service Architecture Working Group Note*
  - *Defines **views***
    - → *message-oriented view (WSDL and SOAP)*
    - → *resource-oriented view (REST and HTTP)*
  - *Defines **architecture entities** and their **interactions***
    - → *Abstraction over underlying technology*
    - → *Basis for service usage processes and description languages*
- Service Oriented Architecture
  - *Collection of tools, methods and technologies*
  - *There is some implicit understanding of SOA in the community such as*
    - → *SOA provides advances over Enterprise Application Integration*
    - → *SOA is realized by using SOAP, WSDL, (and UDDI) technologies*
    - → *SOA utilizes Enterprise Service Bus (ESB)*
  - ⇒ *~ a realization of Web Service Architecture message-oriented view*

Service

- Difficult to agree on one definition
- Business definition
  - *A service realizes an effect that brings a business value to a service consumer*
    - → *for example, to pay for and deliver a book*
- Conceptual definition
  - *service characteristics*
    - → *encapsulation, reusability, loose coupling, contracting, abstraction, discoverability, composability*
- Logical definition
  - *service interface, description and implementation*
  - *service usage process*
    - → *service use tasks, service types*
- Architectural definition
  - *business service (also application service)*
    - → *external, exposed functionality of an application*
  - *infrastructure service*
    - → *internal/technical, supports processing of requests*

Interface, Description and Implementation



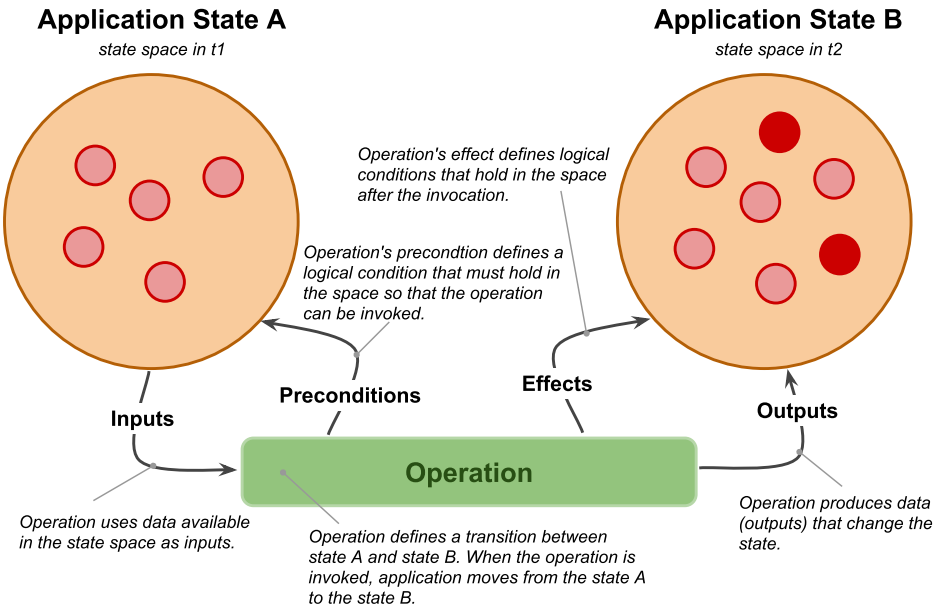
- Terminology clarification
  - *service ~ service interface + service implementation*
  - *WSDL service ~ service description in WSDL language*
  - *SOAP service ~ a service interface is possible to access through SOAP protocol; there is a WSDL description usually available too.*
  - *REST/RESTful service ~ service interface that conforms to REST architectural style and HTTP protocol*

Service Interface

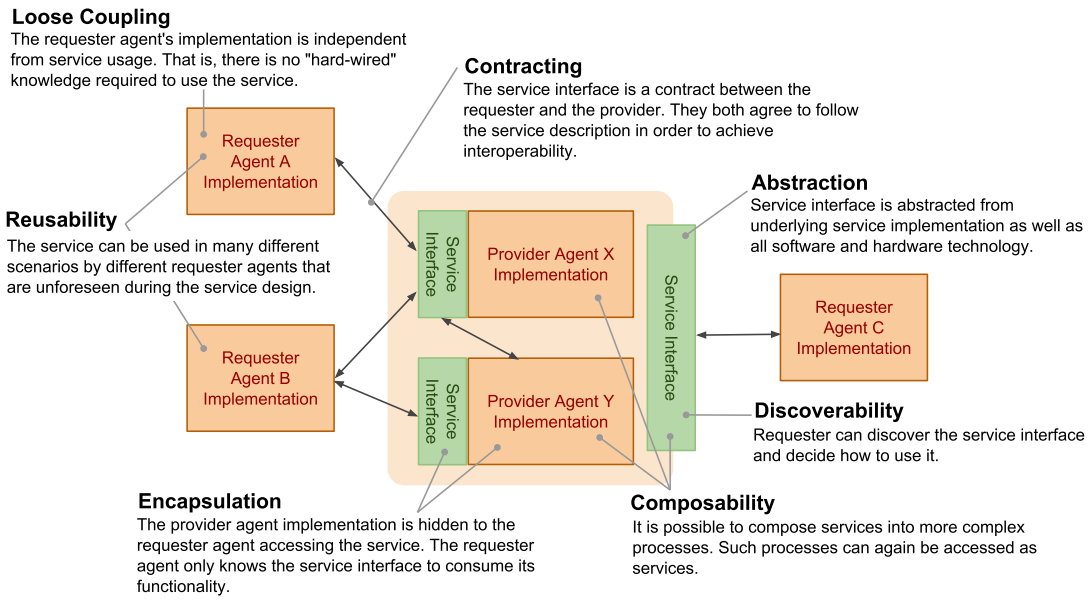
- Service interface components
  - *Data*
    - *→ Data model definition used by the service*
    - *→ for example, input and output messages, representation of resources*
  - *Functions*
    - *→ operations and input and output data used by operations*
  - *Process*
    - *→ public process: how to consume the service's functionality*
    - *→ orchestration: realization of the service's functionality by its implementation*
  - *Technical*
    - *→ security, usage aspects (SLA-Service Level Agreement)*
    - *→ other technical details such as IP addresses, ports, protocols, etc.*

Public Process

- A state diagram
  - *operation of a service defines a **state transition** between two states.*



Service Characteristics

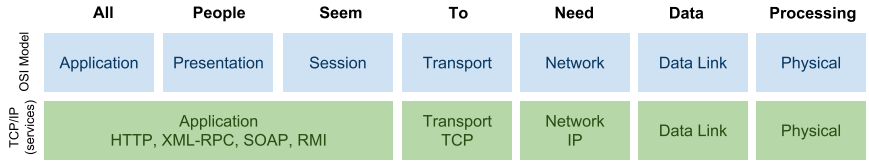


Overview

- Integrating Applications
- Service Definition
- Service Communication
- REST

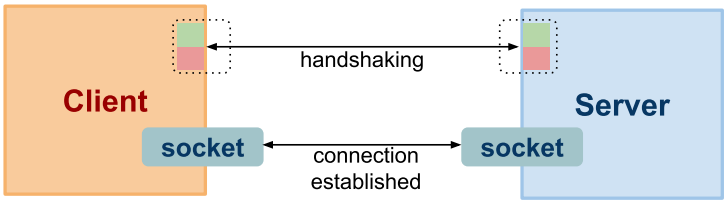
Application Protocols

- Remember this



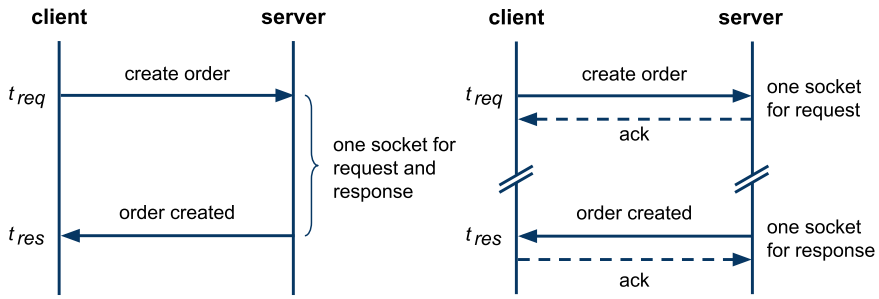
- App protocols mostly on top of the TCP Layer
  - *use TCP socket for communication*
- Major protocols
  - *HTTP – most of the app protocols layered on HTTP*
    → wide spread, but: implementors often break HTTP semantics
  - *RMI – Remote Method Invocation*
    → Java-specific, rather interface
    → may use HTTP underneath (among other things)
  - *XML-RPC – Remote Procedure Call and SOAP*
    → Again, HTTP underneath
  - *WebSocket – new protocol part of HTML5*

Socket



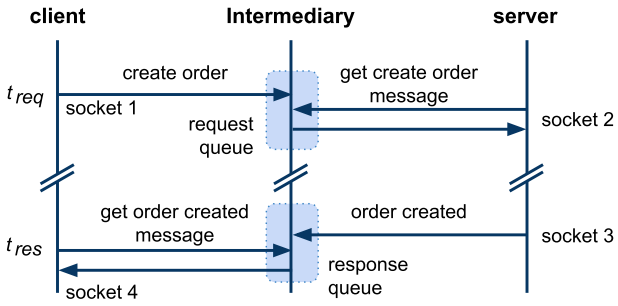
- Handshaking (connection establishment)
  - *The server listens at* `[dst_ip,dsp_port]`
  - *Three-way handshake:*
    - → *the client at* `[src_ip,src_port]` *sends a connection request*
    - → *the server responds*
    - → *the client acknowledges the response, can send data along*
  - *Result is a socket (virtual communication channel) with unique identification:* `socket=[src_ip,src_port;dst_ip,dst_port]`
- Data transfer (resource usage)
  - *Client/server writes/reads data to/from the socket*
  - *TCP features: reliable delivery, correct order of packets, flow control*
- Connection close

Synchronous and Asynchronous Communication



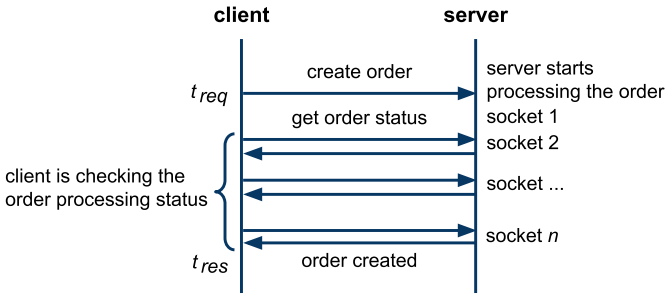
- Synchronous
  - *one socket,* $|t_{req} - t_{res}|$ *is small*
  - *easy to implement and deploy, only standard firewall config*
  - *only the server defines endpoint*
- Asynchronous
  - *request, response each has socket, client and server define endpoints*
  - $|t_{req} - t_{res}|$ *can be large (hours, even days)*
  - *harder to do across network elements (private/public networks issue)*

Asynchronous via Intermediary



- Intermediary
  - *A component that decouples a client-server communication*
  - *It increases reliability and performance*
    - → *The server may not be available when a client sends a request*
    - → *There can be multiple servers that can handle the request*
- Further Concepts
  - *Message Queues (MQ) – queue-based communication*
  - *Publish/Subscribe (P/S) – event-driven communication*

Asynchronous via Polling

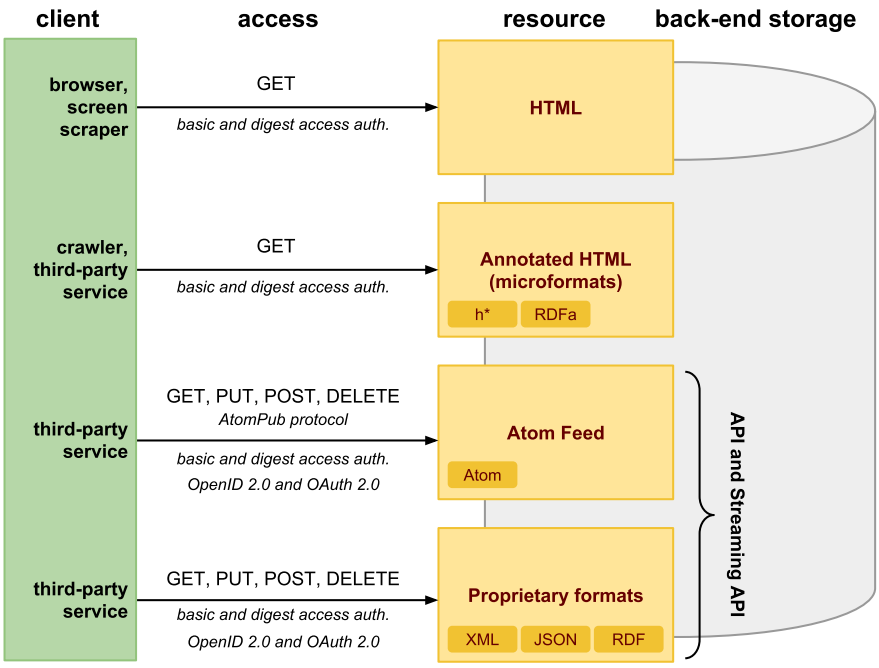


- Polling – only clients open sockets
  - *A client performs multiple request-response interactions*
    - → *The first interaction initiates a process on the server*
    - → *Subsequent interactions check for the processing status*
    - → *The last interaction retrieves the processing result*
- Properties of environments
  - *A server cannot open a socket with the client (network restrictions)*
  - *Typically on the Web (a client runs in a browser)*

Overview

- Integrating Applications
- Service Definition
- Service Communication
- REST
  - *Introduction to REST*
  - *Uniform Resource Identifier*
  - *Resource Representation*
  - *HATEOAS*
  - *Uniform Interface*
  - *Advanced Design Issues*
  - *Caching and Revalidation*
  - *Concurrency Control*

Data on the Web



REST

- REST
  – *Representational State Transfer*
- Architecture Style
  – *Roy Fielding – co-author of HTTP*
  – *He coined REST in his PhD thesis .*
    → *The thesis abstracts from HTTP technical details*
    → *HTTP is one of the REST implementation →* ***RESTful***
    → *REST is a leading programming model for Web APIs*
- REST (RESTful) proper design
  – *people break principles often*
  – *See REST Anti-Patterns for some details.*
- REST and Web Service Architecture
  – *REST is a realization of WSA resource-oriented model*

REST and Web Architecture

- Tim-Berners Lee
  – *"creator", father of the Web*
- Key Principles
  – *Separation of Concerns*
    → *enables independent innovation*
  – *Standards-based*
    → *common agreement, big spread and adoption*
  – *Royalty-free technology*
    → *a lot of open source, no fees*
- Architectural Basis
  – ***Identification:*** *universal linking of resources using URI*
  – ***Interaction:*** *protocols to retrieve resources – HTTP*
  – ***Formats:*** *resource representation (data and metadata)*

HTTP Advantages

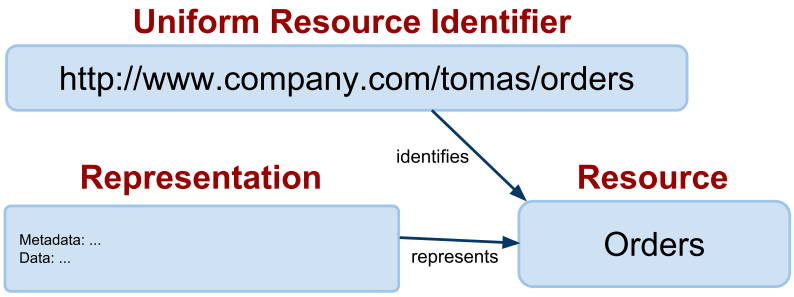
- Familiarity
  - *HTTP protocol is well-known and widely used*
- Interoperability
  - *All environments have HTTP client libraries*
    - *→ technical interoperability is thus no problem*
    - *→ no need to deal with vendor-specific interoperability issues*
  - *You can focus on the core of the integration problem*
    - *→ application (domain, content) interoperability*
- Scalability
  - *you can use highly scalable Web infrastructure*
    - *→ caching servers, proxy servers, etc.*
  - *HTTP features such as HTTP GET idempotence and safe allow you to use caching*

REST Core Principles

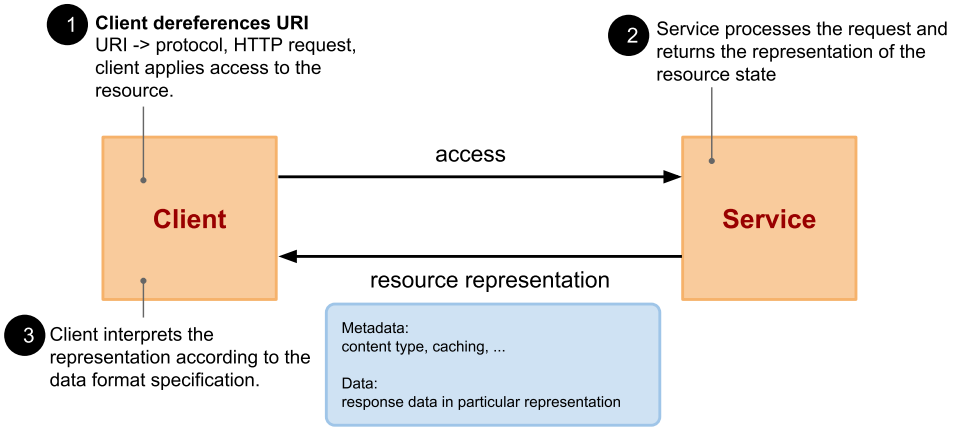
- REST architectural style defines constraints
  - *if you follow them, they help you to achieve a good design, interoperability and scalability.*
- Constraints
  - *Client/Server*
  - *Statelessness*
  - *Cacheability*
  - *Layered system*
  - *Uniform interface*
- Guiding principles
  - *Identification of resources*
  - *Representations of resources and self-descriptive messages*
  - *Hypermedia as the engine of application state (HATEOAS)*

Resource

- A resource can be anything such as
  - *A real object: car, dog, Web page, printed document*
  - *An abstract thing such as address, name, etc. → RDF*
- A resource in REST
  - *A resource corresponds to one or more entities of a data model*
  - *A representation of a resource can be conveyed in a message electronically (information resource)*
  - *A resource has an identifier and a representation and a client can apply an access to it*



Access to a Resource



- Terminology
  - *Client = User Agent*
  - ***Dereferencing URI** – a process of obtaining a protocol from the URI and creating a request.*
  - ***Access** – a process of sending a request and obtaining a response as a result; access usually realized through HTTP.*

Overview

- Integrating Applications
- Service Definition
- Service Communication
- REST
  - *Introduction to REST*
  - *Uniform Resource Identifier*
  - *Resource Representation*
  - *HATEOAS*
  - *Uniform Interface*
  - *Advanced Design Issues*
  - *Caching and Revalidation*
  - *Concurrency Control*

URI, URL, URN

- URI – Uniform Resource Identifier
  - *URI only identifies a resource*
    → it does not imply the resource physically exists
  - *URI could be URL (locator) or URN (name)*
- URL – Uniform Resource Locator
  - *in addition allows to locate the resource*
    → that is — its network location
  - *every URL is URI but an URI does not need to be URL*
- URN – Uniform Resource Name
  - *refers to URI under "urn" scheme (RFC 2141)*
  - *require to be globally unique and persistent*
    → even if the resource cease to exist/becomes unavailable

URI

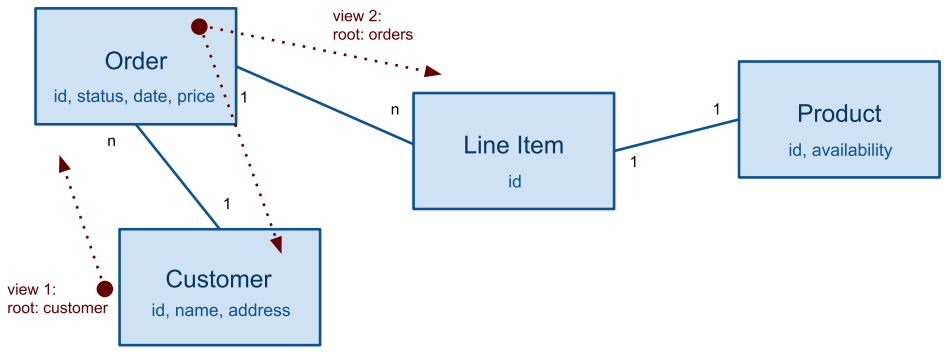
- Definition

```
URI = scheme ":" [ "//" authority ] [ "/" path ] [ "?" query ] [ "#" frag ]
```

- Hierarchal sequence of components
  - `scheme`
    - → *refers to a spec that assigns IDs within that scheme*
    - → *examples:* `http`, `ftp`, `mailto`, `urn`
    - → `scheme != protocol`
  - `authority`
    - → *registered name (domain name) or server address*
    - → *optional port and user*
  - `path and query`
    - → *identify resource within the scheme and authority scope*
    - → *path – hierarchal form*
    - → *query – non-hierarchal form (parameters key=value)*
  - `fragment`
    - → *reference to a secondary resource within the primary resource*

Resources over Entities

- Application's data model
  - *Entities and properties that the app uses for its data*



- URI identifies a resource within the app's data model
  - `path` – *a "view" on the data model*
    - → *data model is a graph*
    - → *URI identifies a resource using a path in a tree with some root*

Examples of Views

- View 1
  - *all customers:* `/customers`
  - *a particular customer:* `/customers/{customer-id}`
  - *All orders of a customer:* `/customers/{customer-id}/orders`
  - *A particular order:* `/customers/{customer-id}/orders/{order-id}`
- View 2
  - *all orders:* `/orders`
  - *All orders of a customer:* `/orders/{customer-id}`
  - *A particular order:* `/orders/{customer-id}/{order-id}`

⇒ Design issues

- Good design practices
  - *No need for 1:1 relationship between resources and data entities*
    - → *A resource may aggregate data from two or more entities*
    - → *Thus only expose resources if it makes sense for the service*
  - *Try to limit URI aliases, make it simple and clear*

Path vs. Query

- Path
  - *Hierarchical component, a view on the data*
  - *The main identification of the resource*
- Query
  - *Can define selection, projection or other processing instructions*
  - *Selection*
    - → *filters entries of a resource by values of properties*

      `/customers/?status=valid`
  - *Projection*
    - → *filters properties of resource entries*

      `/customers/?properties=id,name`
  - *Processing instructions examples*
    - → *data format of the resource → cf. URI opacity*

      `/customers/?format=JSON`
    - → *Access keys such as API keys*

      `/customers/?key=3ae56-56ef76-34540aeb`

Fragment

- Primary resource
  - *Defined by URI path and query*
  - *could be complex, composed resources*
- Sub-resource/secondary resource
  - *Can be defined by a fragment*
  - *No explicit relationship between primary and sub-resource*
    → *For example, we cannot infer that the two resources are in* `part-of`*, or* `sub-class-of` *relationships.*
  - *Fragment semantics defined by a data format*
- Usage of fragment
  - *identification of elements in HTML*
  - *URI references in RDF*
  - *State of an application in a browser*

Fragment Semantics

- Fragment semantics for HTML
  - *assume that* `orders.html` *are in* `HTML` *format.*

```
1 | http://company.com/tomas/orders.html#3456
```

  ⇒ *there is a HTML element with* `id=3456`
- But:
  - *Consider* `orders` *resource in* `application/xml`

```
1 | <orders>
2 |     <order id="3456">...</order>
3 |     ...
4 | </orders>
```

  - *Can't say that* `http://company.com/tomas/orders.xml#3456` *identifies an order element within the* `orders` *resource.*
  - `application/xml` *content type does not define fragment semantics*

Resource ID vs. Resource URI

- Resource ID
  - *Local ID, part of an entity in a data model*
  - *Unique within an application where the resource belongs*
  - *Usually generated on a server (cf. PUT to update and insert)*
  - *Exposed to the resource URI as a path element*
    `/orders/{order-id}`
- Resource URI
  - *Global identifier, valid on the whole Web*
  - *Corresponds to the view on the data model of the app*
  - *Include multiple higher-level resources' IDs*
  - *Example:*
    `/customers/{customer-id}/orders/{order-id}/`
  - *There can be more URIs identifying the same resource*

Major characteristics

- Capability URL
  - *Short lived URL generated for a specific purpose*
  - *For example, an user e-mail verification*
- URI Alias
  - *Two URIs identifying the same resource*
- URI Collision
  - *Two URIs identifying the same resource (misuse of an URI authority)*
- URI Opacity
  - *Content type encoded as part of an URI*
  - `http://www.example.org/customers.xml`
- Resource versions encoded in an URI
  - *Two URIs identifying the same resource of different versions*
  - `http://www.example.org/v1/customers.xml`
- Persistent URL
  - *URL is valid even when the resource is obsolete*
  - *For example, a redirection should be in place*

Overview

- Integrating Applications
- Service Definition
- Service Communication
- REST
  - *Introduction to REST*
  - *Uniform Resource Identifier*
  - *Resource Representation*
  - *HATEOAS*
  - *Uniform Interface*
  - *Advanced Design Issues*
  - *Caching and Revalidation*
  - *Concurrency Control*

Representation and Data Format

- Representation
  - *Various languages, one resource can have multiple representations*
    → *XML, HTML, JSON, YAML, RDF, ...*
    → *should conform to Internet Media Types*
- Data format
  - *Format of resource data*
  - *Binary format*
    → *specific data structures*
    → *pointers, numeric values, compressed, etc.*
  - *Textual format*
    → *in a defined encoding as a sequence of characters*
    → *HTML, XML-based formats are textual*

Metadata

- Metadata ~ self-description
  - *Data about the resource*
  - *e.g., data format, representation, date the resource was created, ...*

  1. *Defined by HTTP response headers*
  2. *Can be part of the data format*
     - → *Atom Syndication Format such as* `author`, `updated`, ...
     - → *HTML* `http-equiv` *meta tags*
- Resource anatomy

Content-Type Metadata

- Access
  - *to be retrieved (GET)*
  - *to be inserted or updated (PUT, POST)*
  - *to be deleted (DELETE)*
- Request
  - *HTTP header* `Accept`*, part of content negotiation protocol*
- Response
  - *HTTP header* `Content-Type: type/subtype; parameters`
  - *Specifies an Internet Media Type of the resource representation.*
    - → *IANA (Internet Assigned Numbers Authority) manages a registry of media types and character encodings*
    - → *subtypes of* `text` *type have an optional charset parameter*

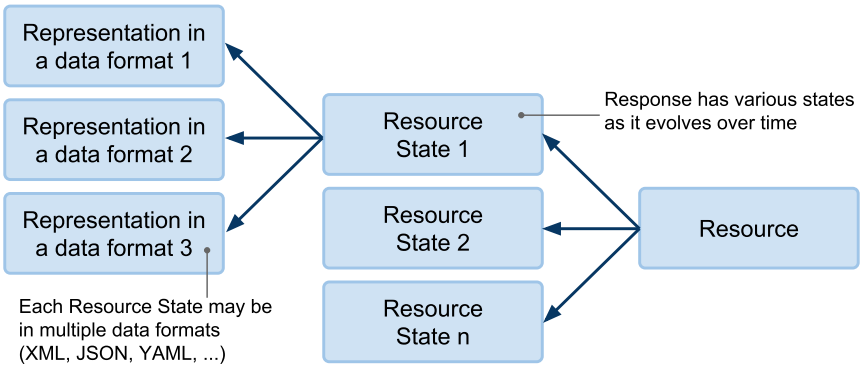
      `text/html; charset=iso-8859-1`
  - *A resource may provide more than one representations*
    - → *promotes services' loose coupling*

Major Media Types

- Common Standard Media Types
  - `text/plain`
    → *natural text in no formal structures*
  - `text/html`
    → *natural text embedded in HTML format*
  - `application/xml`, `application/json`
    → *XML-based/JSON-based, application specific format*
  - `application/wsdl+xml`
    → `+xml` *suffix to indicate a specific format*
- Non-standard media types
  - *Types or subtypes that begin with* `x-` *are not in IANA*
    `application/x-latex`
  - *subtypes that begin with* `vnd.` *are vendor-specific*
    `application/vnd.ms-excel`

Resource State

- State
  - *Resource representation is in fact a* ***representation of a resource state***
  - *Resource may be in different states over time*



- In REST resource states represent application states

Resource State Example

- Time `t1`: client A retrieves a resource `/orders` (GET)

```
<orders>
    <order id="54467"/>
    <order id="65432"/>
</orders>
```

- Time `t2`: client B adds a new order (POST)

```
<order>
    ...
</order>
```

- Time `t3`: client A retrieves a resource `/orders` (GET)

```
<orders>
    <order id="54467"/>
    <order id="65432"/>
    <order id="74567"/>
</orders>
```

- The resource `/orders` has different states in `t1` and `t3`.

Overview

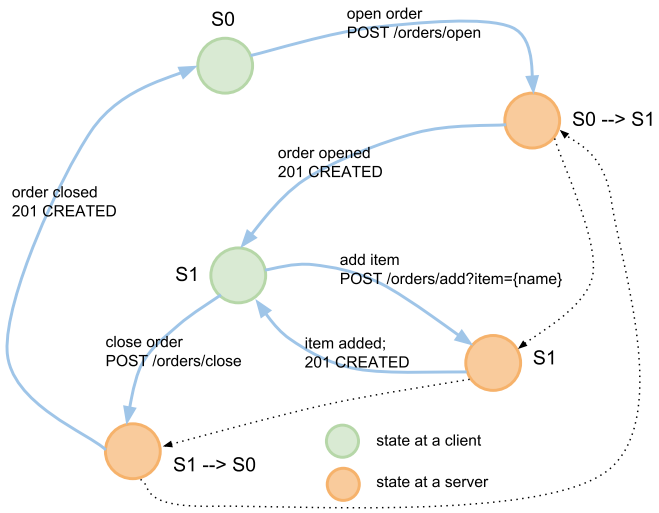
- Integrating Applications
- Service Definition
- Service Communication
- REST
  - *Introduction to REST*
  - *Uniform Resource Identifier*
  - *Resource Representation*
  - *HATEOAS*
  - *Uniform Interface*
  - *Advanced Design Issues*
  - *Caching and Revalidation*
  - *Concurrency Control*

HATEOAS

- HATEOAS = Hypertext as the Engine for Application State
  - *The REST core principle*
  - ***Hypertext***
    - → *Hypertext is a representation of a resource with **links***
    - → *A link is an URI of a resource*
    - → *Applying an access to a resource via its link = state transition*
- Statelessness
  - *A service does not use a memory to remember a state*
  - *HATEOAS enables stateless implementation of services*

Stateful server

- Sessions to store the application state
  - *The app uses a server memory to remember the state*
  - *When the server restarts, the app state is lost*

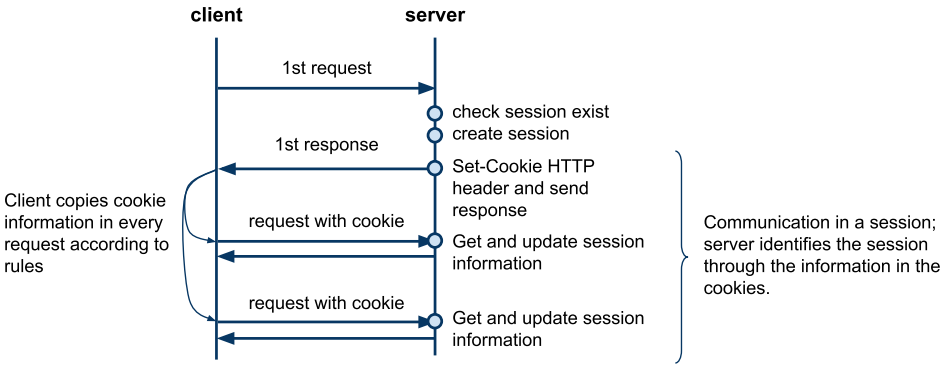


State Management

- HTTP is a stateless protocol – original design
  - *No information to relate multiple interactions at server-side*
    - → *Except* `Authorization` *header is copied in every request*
    - → *IP addresses do not work, one public IP can be shared by multiple clients*
- Solutions to check for a valid state at server-side
  - ***Cookies*** *– obvious and the most common workaround*
    - → *RFC 2109 – HTTP State Management Mechanism*
    - → *Allow clients and servers to talk in a context called* ***sessions***
  - ***Hypertext*** *– original HTTP design principle*
    - → *App states represented by resources (hypermedia), links define transitions between states*
    - → *Adopted by the REST principle* ***statelessness***

Interaction with Cookies

- Request-response interaction with cookies
  - *Session is a logical channel maintained by the server*



- Stateful Server
  - *Server remembers the session information in a server memory*
  - *Server memory is a non-persistent storage, when server restarts the memory content is lost!*

Set-Cookie and Cookie Headers

- `Set-Cookie` response header

```
set-cookie = "Set-Cookie:" cookie ("," cookie)*
  cookie    = NAME "=" VALUE (";" cookie-av)*
  cookie-av = "Comment" "=" value
            | "Domain" "=" value
            | "Max-Age" "=" value
            | "Path" "=" value
```

  - `domain` – *a domain for which the cookie is applied*
  - `Max-Age` – *number of seconds the cookie is valid*
  - `Path` – *URL path for which the cookie is applied*

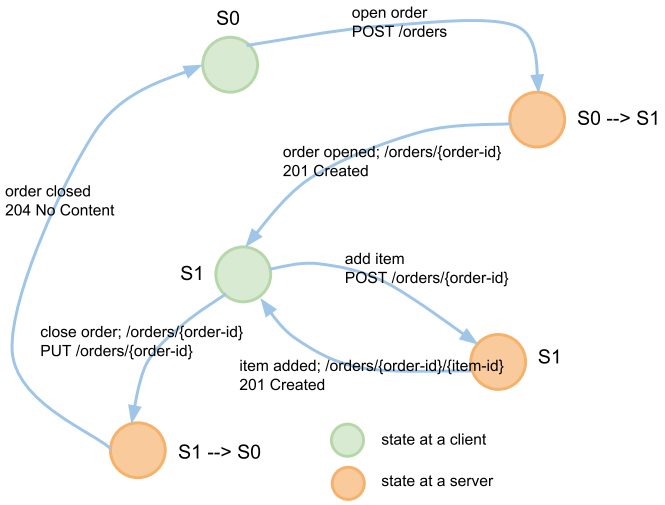
- `Cookie` request header. A client sends the cookie in a request if:
  - `domain` *matches the origin server's fully-qualified host name*
  - `path` *matches a prefix of the request-URI*
  - `Max-Age` *has not expired*

```
cookie  = "Cookie:" cookie-value (";" cookie-value)*
  cookie-value    = NAME "=" VALUE [";" path] [";" domain]
  path            = "$Path" "=" value
  domain          = "$Domain" "=" value
```

  - `domain`, *and* `path` *are values from corresponding attributes of the* `Set-Cookie` *header*

Stateless server

- HTTP and hypermedia to transfer the app state
  - *Does not use a server memory to remember the app state*
  - *State transferred between a client and a service via HTTP metadata and resources' representations*

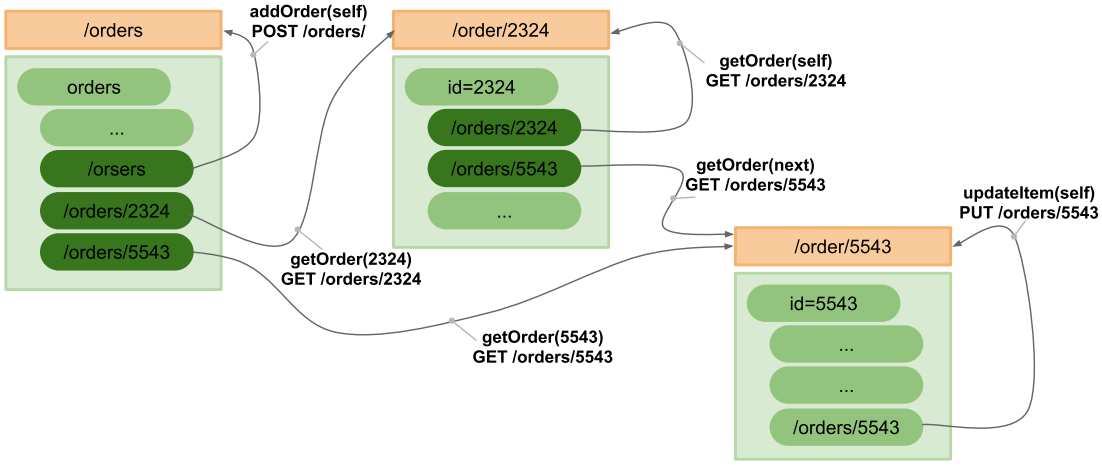


Persistent Storage and Session Memory

- Persistent Storage
  - *Contains the app data*
  - *Data is serialized into resource representation formats*
  - *All sessions may access the data via resource IDs*
- Session Memory
  - *Server memory that contains a state of the app*
  - *A session may only access its session memory*
  - *Access through cookies*
  - *Note*
    → *A session memory may be implemented via a persistent storage (such as in Google AppEngine)*

Link

- Service operation
  - *Applying an access to a link (GET, PUT, POST, DELETE)*
  - *Link: HTTP method + resource URI + optional link semantics*
- Example: `getOrder`, `addOrder`, and `updateItem`



Atom Links

- Atom Syndication Format
  - *XML-based document format; Atom feeds*
  - *Atom links becoming popular for RESTful applications*

```
1  <order a:xmlns="http://www.w3.org/2005/Atom" xmlns="...">
2      <a:link
3          rel="next"
4          href="http://company.com/orders/5543"
5          type="application/xml"/>
6      <customer>Tomas</customer>
7      <items>...</items>
8  </order>
```

  - *Link structure*

    `rel` – *name of the link*

    ~ semantics of an operation behind the link

    `href` – *URI to the resource described by the link*

    `type` – *media type of the resource the link points to*

Link Semantics

- Standard `rel` values
  - *Navigation:* `next`, `previous`, `self`
  - *Does not reflect a HTTP method you can use*
- Extension `rel` values
  - *You can use* `rel` *to indicate a semantics of an operation*
  - *Example: add item, delete order, update order, etc.*
  - *A client associates this semantics with an operation it may apply at a particular state*
  - *The semantics should be defined by using an URI*

```
1  <order a:xmlns="http://www.w3.org/2005/Atom" xmlns="...">
2      <id>2324</id>
3      <a:link rel="http://company.com/op/addItem"
4          href="http://company.com/orders/2324"/>
5      <a:link rel="http://company.com/op/deleteOrder"
6          href="http://company.com/orders/2324"/>
7  </order>
```

Link Headers

- An alternative to Atom links in resource representations
  - *links defined in HTTP Link header, Web Linking IETF spec*
  - *They have the same semantics as Atom Links*
  - *Example:*

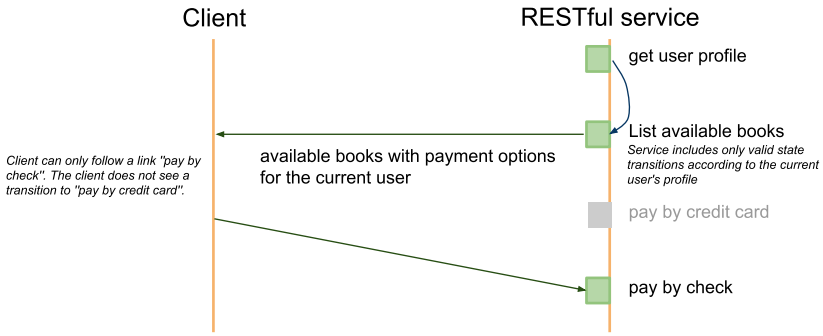
```
> HEAD /orders HTTP/1.1

< Content-Type: application/xml
< Link: <http://company.com/orders/?page=2&size=10>; rel="next"
< Link: <http://company.com/orders/?page=10&size=10>; rel="last"
```

- Advantages
  - *no need to get the entire document*
  - *no need to parse the document to retrieve links*
  - *use HTTP HEAD only*

Preconditions and HATEOAS

- Preconditions in HATEOAS
  - *Service in a current state generates only valid transitions that it includes in the representation of the resource.*
  - *Transition logic is realized at the server-side*



Advantages

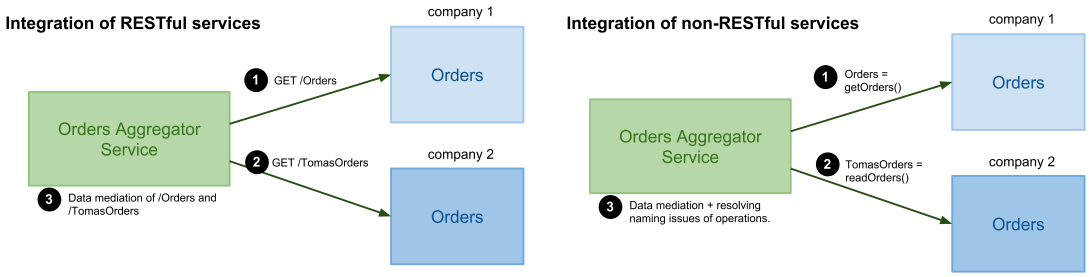
- Location transparency
  - *only "entry-level" links published to the World*
  - *other links within documents can change without changing client's logic*
  - *Hypertext represents the current user's view, i.e. rights or other context*
- Loose coupling
  - *no need for a logic to construct the links*
  - *Clients know to which states they can move via links*
- Statelessness and Cloud
  - *Better implementation of scalability*

Overview

- Integrating Applications
- Service Definition
- Service Communication
- REST
  - *Introduction to REST*
  - *Uniform Resource Identifier*
  - *Resource Representation*
  - *HATEOAS*
  - *Uniform Interface*
  - *Advanced Design Issues*
  - *Caching and Revalidation*
  - *Concurrency Control*

Uniform Interface

- Uniform interface = finite set of operations
  - *Resource manipulation*
    - → *CRUD – Create (POST/PUT), Read (GET), Update (PUT/PATCH), Delete (DELETE)*
  - *operations are not domain-specific*
    - → *For example,* `GET /orders` *and not* `getOrders()`
    - → *This reduces complexity when solving interoperability*
- Integration issues examples



Safe and Unsafe Operations

- Safe operations
  - *Do not change the resource state*
  - *Usually "read-only" or "lookup" operation*
  - *Clients can cache the results and refresh the cache freely*
- Unsafe operations
  - *May change the state of the resource*
  - *Transactions such as buy a ticket, post a message*
  - *Unsafe does not mean dangerous!*
- Unsafe interactions and transaction results
  - `POST` *response may include transaction results*
    - → *you buy a ticket and submit a purchase data*
    - → *you get transaction results*
    - → *and you cannot bookmark this..., why?*
  - *Should be referable with a persistent URI*

Idempotence

- Idempotent operation
  - *Invoking a method on the same resource always has the same effect*
  - *Operations* `GET`, `PUT`, `DELETE`
- Non-idempotent operation
  - *Invoking a method on the same resource may have different effects*
  - *Operation* `POST`
- Effect = a state change
  - *recall the effect definition in MDW*

GET

- Reading
  - `GET` *retrieves a representation of a state of a resource*

```
> GET /orders HTTP/1.1
> Accept: application/xml

< HTTP/1.1 200 OK
< Content-Type: application/xml
<
< ...resource representation in xml...
```

  - *It is read-only operation*
  - *It is* ***safe***
  - *It is* ***idempotent***
  - `GET` *retrieves different states over time but the effect is always the same, cf. resource state hence it is idempotent.*
  - *Invocation of* `GET` *involves content negotiation*

PUT

- Updating or Inserting
  - `PUT` *updates or inserts a representation of a state of a resource*
  - *Updating the resource is a* ***complete replacement of the resource***

```
> PUT /orders/4456 HTTP/1.1
> Content-Type: application/xml
>
> <order>...</order>

< HTTP/1.1 CODE
```

  - *where CODE is:*
    - → `200 OK` *or* `204 No Content` *for updating: A resource with id* `4456` ***exists****, the client sends an updated resource*
    - → `201 Created` *for inserting: A resource* ***does not exist****, the client generates the id* `4456` *and sends a representation of it.*
  - *It is* ***not safe*** *and it is* ***idempotent***

PATCH

- `PATCH` to partial update a resource
  - *IETF specification, see PATCH Method for HTTP*
- Use in GData Protocol
  - *To add, modify or delete selected elements of an Atom feed entry*
  - *Example to delete a description element and add a new title element* `gd:fields` *uses the partial response syntax*

```
1  PATCH /myFeed/1/1/
2  Content-Type: application/xml
3
4  <entry xmlns='http://www.w3.org/2005/Atom'
5      xmlns:gd='http://schemas.google.com/g/2005'
6      gd:fields='description'>
7    <title>New title</title>
8  </entry>
```

  - *Rules*
    - → *Fields not already present are added*
    - → *Non-repeating fields already present are updated*
    - → *Repeating fields already present are appended*

POST

- Inserting
  - `POST` *inserts a new resource*
  - *A server generates a new resource ID, client only supplies a content and a resource URI where the new resource will be inserted.*

```
> POST /orders HTTP/1.1
> Content-Type: application/xml
>
> <order>...</order>

< HTTP/1.1 201 Created
< Location: /orders/4456
```

  - *It is **not safe** an it is **not idempotent***
  - *A client may "suggest" a resource's id using the* `Slug` *header*
    → *Defined in AtomPub protocol*

DELETE

- Deleting
  - `DELETE` *deletes a resource with specified URI*

```
> DELETE /orders/4456 HTTP/1.1

< HTTP/1.1 CODE
```

  - *where CODE is:*
    → `200 OK`*: the response body contains an entity describing a result of the operation.*
    → `204 No Content`*: there is no response body.*
  - *It is **not safe** and it is **idempotent***
    → *Multiple invocation of* `DELETE /orders/4456` *has always the same effect – the resource* `/orders/4456` *does not exist.*

Other

- HEAD
  - *same as* `GET` *but only retrieves HTTP headers*
  - *It is* ***safe*** *and* ***idempotent***
- OPTIONS
  - *queries the resource for resource configuration*
  - *It is* ***safe*** *and* ***idempotent***

Types of Errors

- Client-side – status code `4xx`
  - `400 Bad Request`
    - → *generic client-side error*
    - → *invalid format, such as syntax or validation error*
  - `404 Not Found`
    - → *server can't map URI to a resource*
  - `401 Unauthorized`
    - → *wrong credentials (such as user/pass, or API key)*
    - → *the response contains* `WWW-Authenticate` *indicating what kind of authentication the service accepts*
  - `405 Method Not Allowed`
    - → *the resource does not support the HTTP method the client used*
    - → *the response contains* `Allow` *header to indicate methods it supports*
  - `406 Not Acceptable`
    - → *so many restrictions on acceptable content types (using* `Accept-*`*)*
    - → *server cannot serialize the resource to requested content types*

Types of Errors (Cont.)

- Server-side – status code `5xx`
  - `500 Internal Server Error`
    - → *generic server-side error*
    - → *usually not expressive, logs a message for system admins*
  - `503 Service Not Available`
    - → *server is overloaded or is under maintenance*
    - → *the response contains* `Retry-After` *header*

Use of Status Codes

- Service should respect semantics of status codes!

```
> GET /orders HTTP/1.1
> Accept: application/json

< HTTP/1.1 200 OK
< Content-Type: application/json
<
< { "error" :
<    { "error_text" :
<        "you do not have rights to access this resource " }
< }
```

  - *Client must understand the semantics of the response.*
  - *This breaks loose coupling and reusability service principles*
  - *The response should be:*

```
< HTTP/1.1 401 Unauthorized
< ...

< ...optional text describing the error...
```

Overview

- Integrating Applications
- Service Definition
- Service Communication
- REST
  - *Introduction to REST*
  - *Uniform Resource Identifier*
  - *Resource Representation*
  - *HATEOAS*
  - *Uniform Interface*
  - *Advanced Design Issues*
  - *Caching and Revalidation*
  - *Concurrency Control*

Respect HTTP Semantics

- Do not overload semantics of HTTP methods
  - *For example,* `GET` *is read-only method and idempotent*
  - *REST Anti-pattern:*

    ```
    GET /orders/?add=new_order
    ```

    → *This is not REST!*

    → *This breaks both safe and idempotent principles*
- Consequences
  - *Result of* `GET` *can be cached by proxy servers*
  - *They can revalidate their caches freely*
  - *You can end up with new entries in your storage without you knowing!*
- The same is true for other methods

Change Order Status

- `status` property of `/orders/{order-id}` resource
  - *reflects a state of the process*
  - *No need to use a stateful service, state is communicated through the order representation*
- How do you implement a canceling an order?
  - *You can delete it using* `DELETE`
  - *But you may want to cancel it in order to:*
    - → *maintain a list of canceled orders*
    - → *have a possibility to "roll-back" canceled orders*

DELETE to cancel

- A bad solution to cancel the order
  - *to cancel with DELETE*

    ```
    DELETE /orders/3454/?cancel=true
    ```
  - *you overload the meaning of DELETE*
  - *you violate the uniform interface principle*
- Always ask a question:
  - ***Is the operation a state of the resource?***
  - *if yes, the operation should be:*
    - → *modeled within the data format*
    - → *or as a separated resource (sub-resource)*
- No verbs in `path` and `query` components!
  - `/cancelOrder`, `/orders/{order-id}/?action=delete`, *etc.*
  - *Verbs in URIs indicate that a resource is actually an operation!*

PUT to cancel

- A RESTful solution to cancel an order
  1. *first, have an order's status*
     - *as part of the Order representation format*
     - *we extend "open" and "close" with "cancel"*
  2. *Use PUT to cancel an order*

```
> PUT /orders/{order-id}
> Content-Type: application/json
>
> { "status" : "cancel" }

< HTTP/1.1 204 No Content
```

- Clean-up all cancelled orders
  - *you can have a resource "all valid orders":* `/orders/valid`
    (~ all orders that are not canceled)
    → `GET /orders/valid` *will return all non-canceled orders*
    → `POST /orders/valid` *will purge all cancelled orders*

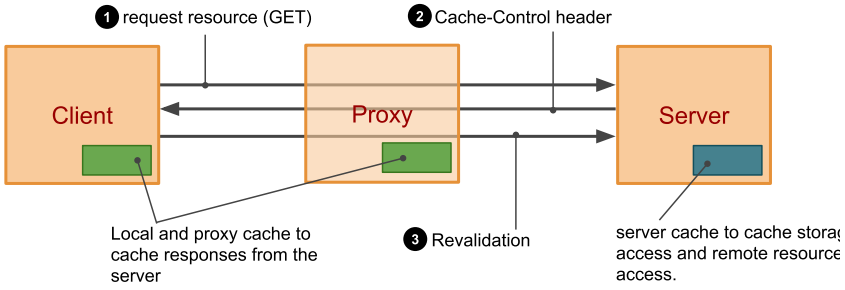
Overview

- Integrating Applications
- Service Definition
- Service Communication
- REST
  - *Introduction to REST*
  - *Uniform Resource Identifier*
  - *Resource Representation*
  - *HATEOAS*
  - *Uniform Interface*
  - *Advanced Design Issues*
  - *Caching and Revalidation*
  - *Concurrency Control*

Scalability

- Need for scalability
  - *Huge amount of requests on the Web every day*
  - *Huge amount of data downloaded*
- Some examples
  - *Google, Facebook: 5 billion API calls/day*
  - *Twitter: 3 billions of API calls/day (75% of all the traffic)*
    → 50 million tweets a day
  - *eBay: 8 billion API calls/month*
  - *Bing: 3 billion API calls/month*
  - *Amazon WS: over 100 billion objects stored in S3*
- Scalability in REST
  - *Caching and revalidation*
  - *Concurrency control*

Caching



- Your service should cache:
  - *anytime there is a static resource*
  - *even there is a dynamic resource*
    → with chances it updates often
    → you can force clients to always revalidate
- three steps:
  - *client GETs the resource representation*
  - *server controls how it should cache through* `Cache-Control` *header*
  - *client revalidates the content via conditional GET*

Cache Headers

- `Cache-Control` response header
  - *controls over local and proxy caches*
  - `private` – *no proxy should cache, only clients can*
  - `public` – *any intermediary can cache (proxies and clients)*
  - `no-cache` – *the response should not be cached. If it is cached, the content should always be revalidated.*
  - `no-store` – *can cache but should not store persistently. When a client restarts, content is lost*
  - `no-transform` – *no transformation of cached data; e.g. compressions*
  - `max-age`, `s-maxage` *a time in seconds how long the cache is valid;* `s-maxage` *for proxies*
- `Last-Modified` and `ETag` response headers
  - *Content last modified date and a content entity tag*
- `If-Modified-Since` and `If-None-Match` request headers
  - *Content revalidation (conditional GET)*

Example Date Revalidation

- Cache control example:

```
> GET /orders HTTP/1.1
> ...

< HTTP/1.1 200 OK
< Content-Type: application/xml
< Cache-Control: private, no-store, max-age=200
< Last-Modified: Sun, 7 Nov 2011, 09:40 CET
<
< ...data...
```

  - *only client can cache, must not be stored on the disk, the cache is valid for 200 seconds.*
- Revalidation (conditional GET) example:
  - *A client revalidates the cache after* `200` *seconds.*

```
> GET /orders HTTP/1.1
> If-Modified-Since: Sun, 7 Nov 2011, 09:40 CET

< HTTP/1.1 304 Not Modified
< Cache-Control: private, no-store, max-age=200
< Last-Modified: Sun, 7 Nov 2011, 09:40 CET
```

Entity Tags

- Signature of the response body
  - *A hash such as MD5*
  - *A sequence number that changes with any modification of the content*
- Types of tag
  - *Strong ETag: reflects the content bit by bit*
  - *Weak ETag: reflects the content "semantically"*
    → *The app defines the meaning of its weak tags*
- Example content revalidation with `ETag`

```
< HTTP/1.1 200 OK
< Cache-Control: private, no-store, max-age=200
< Last-Modified: Sun, 7 Nov 2011, 09:40 CET
< ETag: "4354a5f6423b43a54d"

> GET /orders HTTP/1.1
> If-None-Match: "4354a5f6423b43a54d"

< HTTP/1.1 304 Not Modified
< Cache-Control: private, no-store, max-age=200
< Last-Modified: Sun, 7 Nov 2011, 09:40 CET
< ETag: "4354a5f6423b43a54d"
```


Design Suggestions

- Composed resources use weak ETags
  - *For example* `/orders`
    → *a composed resource that contains a summary information*
    → *changes to an order's items will not change semantics of* `/orders`
  - *It is usually not possible to perform updates on these resources*
- Non-composed resources use strong ETags
  - *For example* `/orders/{order-id}`
  - *They can be updated*
- Further notes
  - *Server should send both* `Last-Modified` *and* `ETag` *headers*
  - *If client sends both* `If-Modified-Since` *and* `If-None-Match`, `ETag` *validation takes preference*

Weak ETag Example

- App specific, `/orders` resource example

```
{
    "orders" :
        [
            { "id"        : 2245,
              "customer"  : "Tomas",
              "descr"     : "Stuff to build a house.",
              "items"     : [...] },
            { "id"        : 5546,
              "customer"  : "Peter",
              "descr"     : "Things to build a pipeline.",
              "items"     : [...] }
        ]
}
```

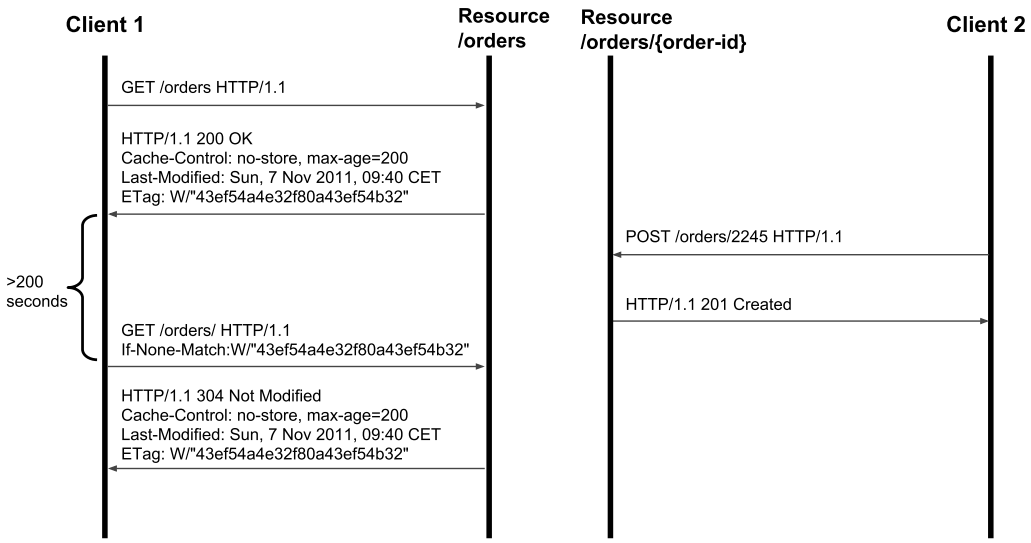
- Weak ETag compute function example
  - *Any modification to an order's items is not significant for* `/orders`*:*

```
var crypto = require("crypto");

function computeWeakETag(orders) {
    var content = "";
    for (var i = 0; i < orders.length; i++)
        content += orders[i].id + orders[i].customer + orders[i].descr;
    return crypto.createHash('md5').update(content).digest("hex");
}
```


Weak ETag Revalidation

- Updating `/orders` resource
  - `POST /orders/{order-id}` *inserts a new item to an order*
  - *Any changes to orders' items will not change the Weak ETag*



Overview

- Integrating Applications
- Service Definition
- Service Communication
- REST
  - *Introduction to REST*
  - *Uniform Resource Identifier*
  - *Resource Representation*
  - *HATEOAS*
  - *Uniform Interface*
  - *Advanced Design Issues*
  - *Caching and Revalidation*
  - *Concurrency Control*

Concurrency

- Two clients may update the same resource

  1) a client GETs a resource `GET /orders/5545`

  2) the client modifies the resource

  3) the client updates the resource via `PUT /orders/5545 HTTP/1.1`

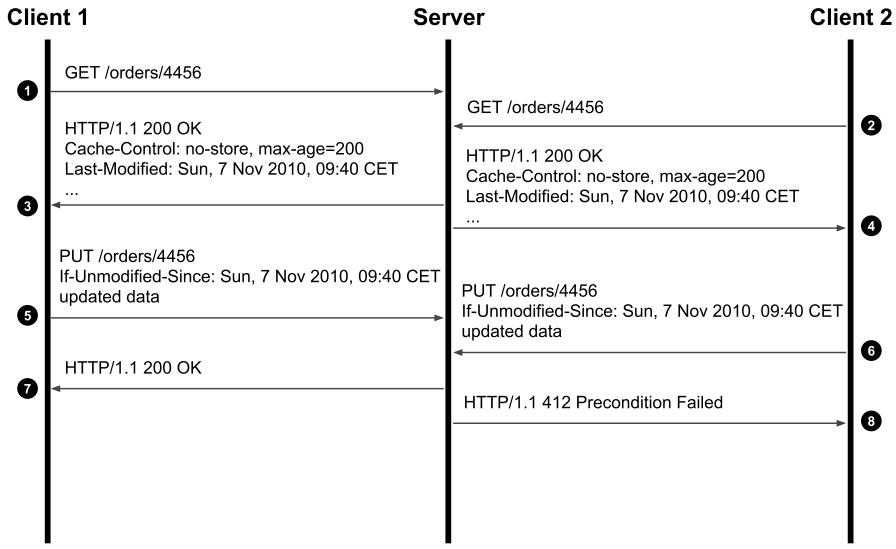
  What happens if another client updates the resource between 1) and 3) ?
- Concurrency control
  - *Conditional* `PUT`

    → *Update the resource only if it has not changed since a specified date or a specified ETag matches the resource content*
  - `If-Unmodified-Since` *and* `If-Match` *headers*
  - *Response to conditional* `PUT`*:*

    → `200 OK` *if the* `PUT` *was successful*

    → `412 Precondition Failed` *if the resource was updated in the meantime.*

Concurrency Control Protocol



- Conditional PUT and ETags
  - *Conditional PUT must always use strong entity tags or date validation*