

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 Sep 30 2018, 21:19:19
Humla v0.3

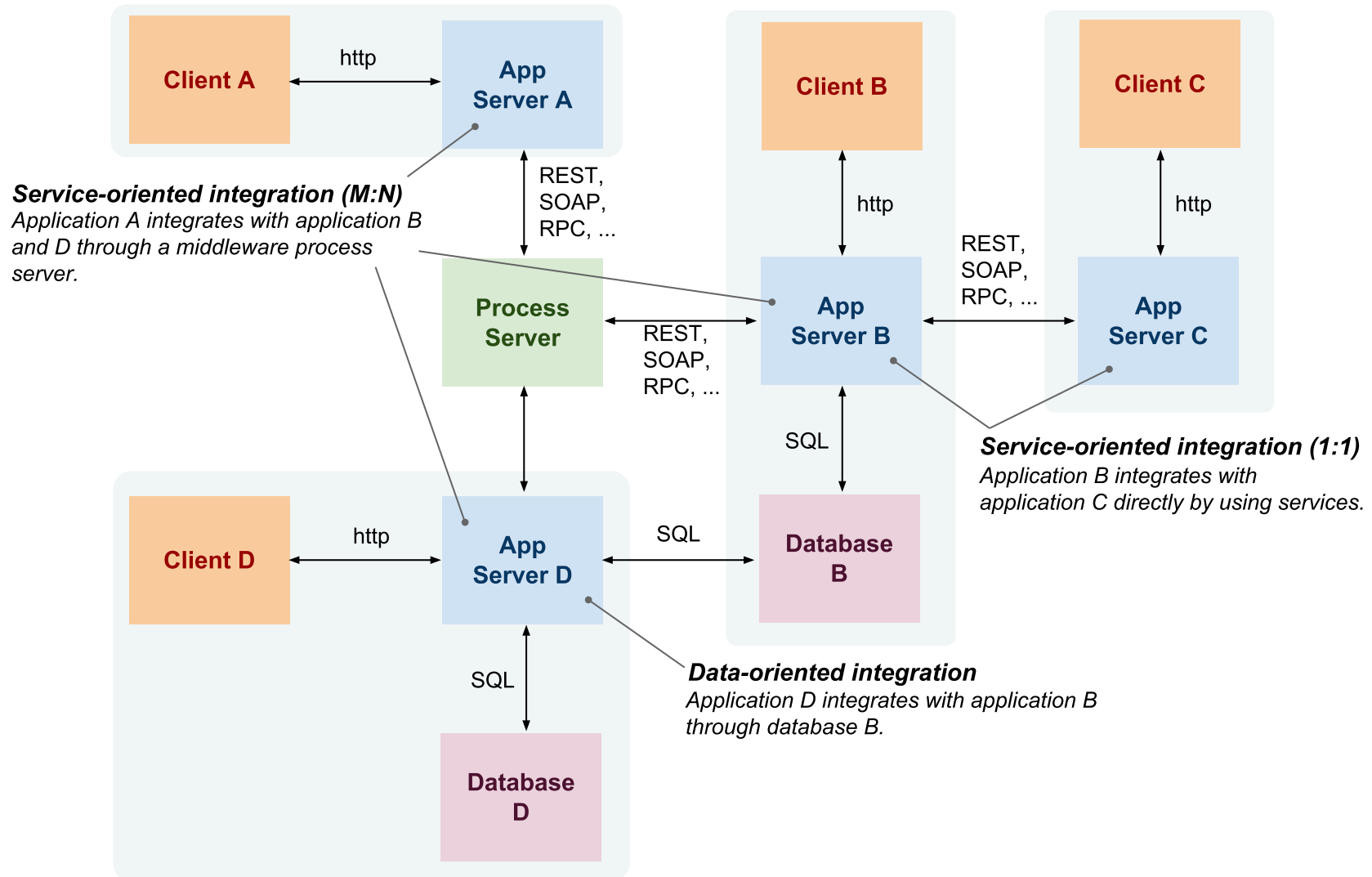
Overview

- Integrating Applications
- Service Definition
- Service Communication
- REST

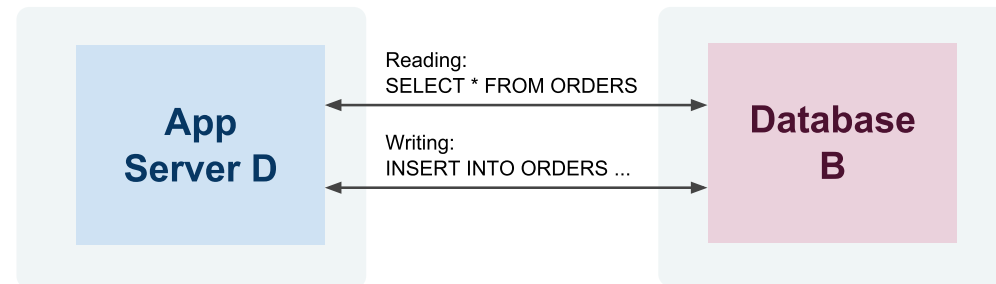
Integration and Interoperability

- 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

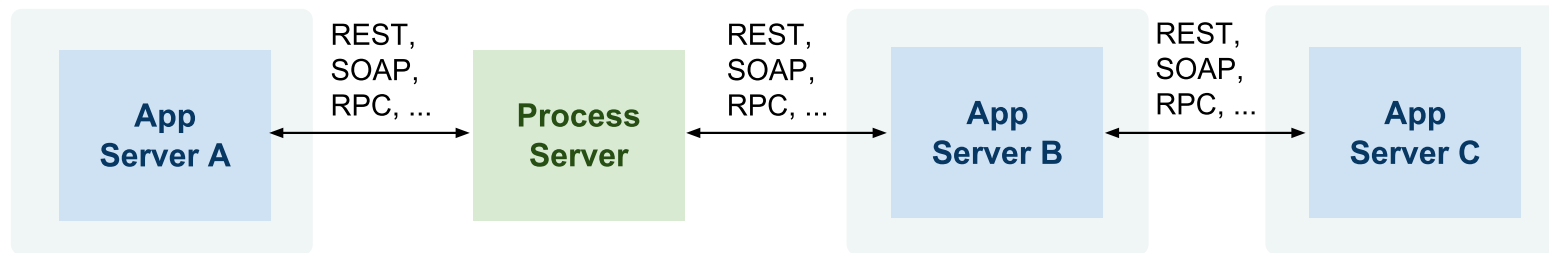


Data-oriented Integration



- 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*

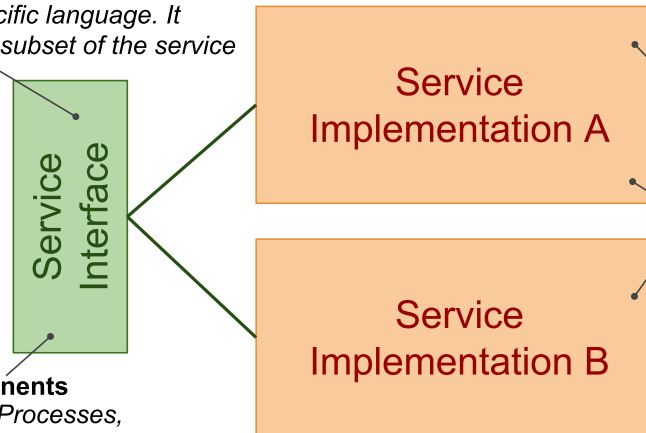
Interface, Description and Implementation

Service Description

An explicit description of the service interface in a specific language. It describes all or a subset of the service interface.

Interface components

Data, Functions, Processes, Technical



Implementation Technology

Service implementation implements the service interface in a specific implementation environment.

Multiple service implementations

One service interface can be realized by many different service implementations.

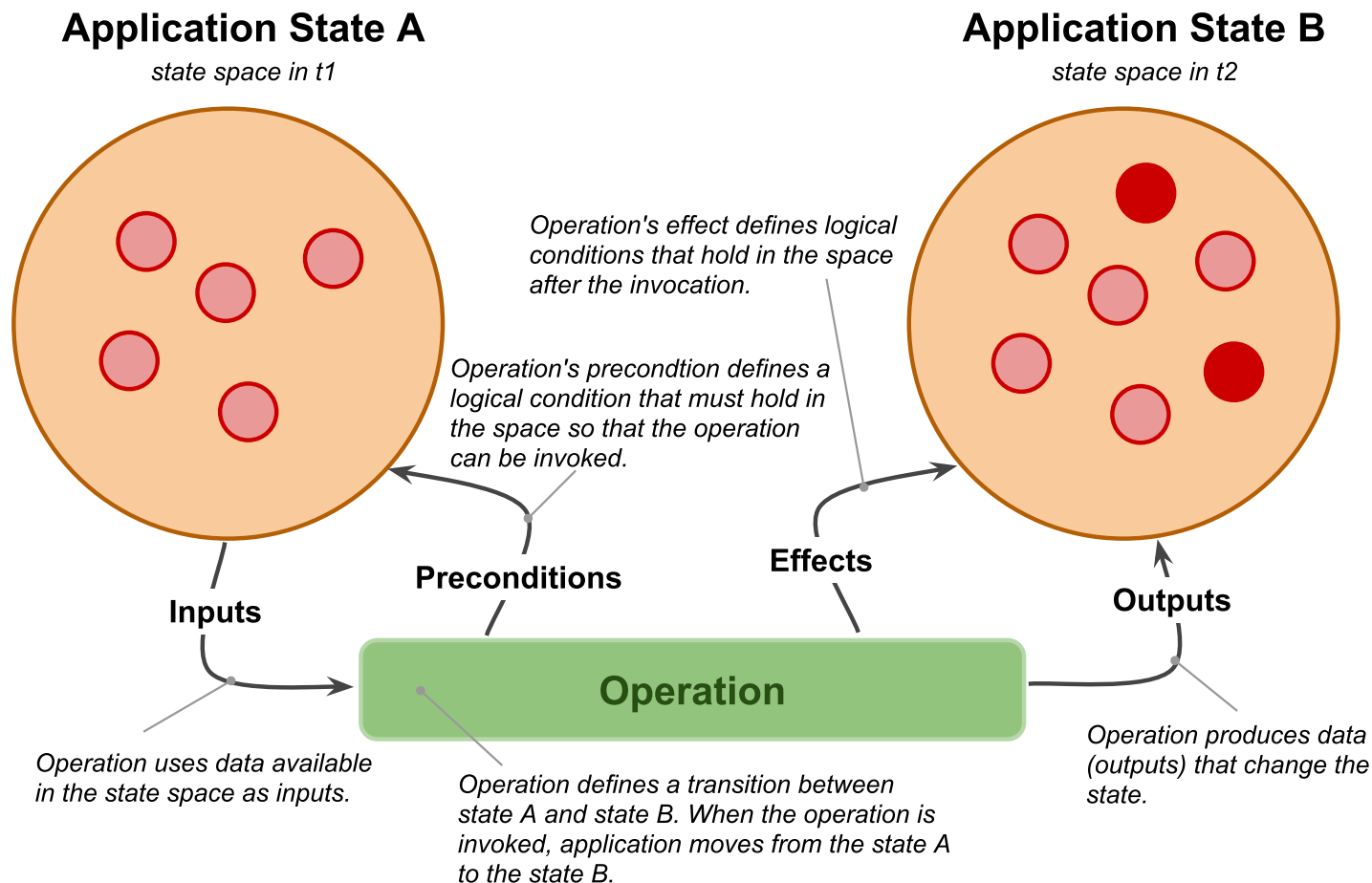
- Terminology clarification
 - *service ~ service interface + service implementation*
 - *WSDL service ~ service description in WSDL language*
 - *SOAP service ~ a service interface is possible to access through SOAP protocol; there is a WSDL description usually available too.*
 - *REST/RESTful service ~ 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, representations of resources
 - *Functions*
 - **Functional definition** of the service
 - operations and input and output data used by operations
 - *Process*
 - **Behavioral definition** of the service
 - public process: how to consume the service's functionality
 - orchestration: realization of the service's functionality by its **implementation**
 - *Technical*
 - **Non-functional definition** of the service
 - 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

Loose Coupling

The requester agent's implementation is independent from service usage. That is, there is no "hard-wired" knowledge required to use the service.

Reusability

The service can be used in many different scenarios by different requester agents that are unforeseen during the service design.

Contracting

The service interface is a contract between the requester and the provider. They both agree to follow the service description in order to achieve interoperability.

Abstraction

Service interface is abstracted from underlying service implementation as well as all software and hardware technology.

Discoverability

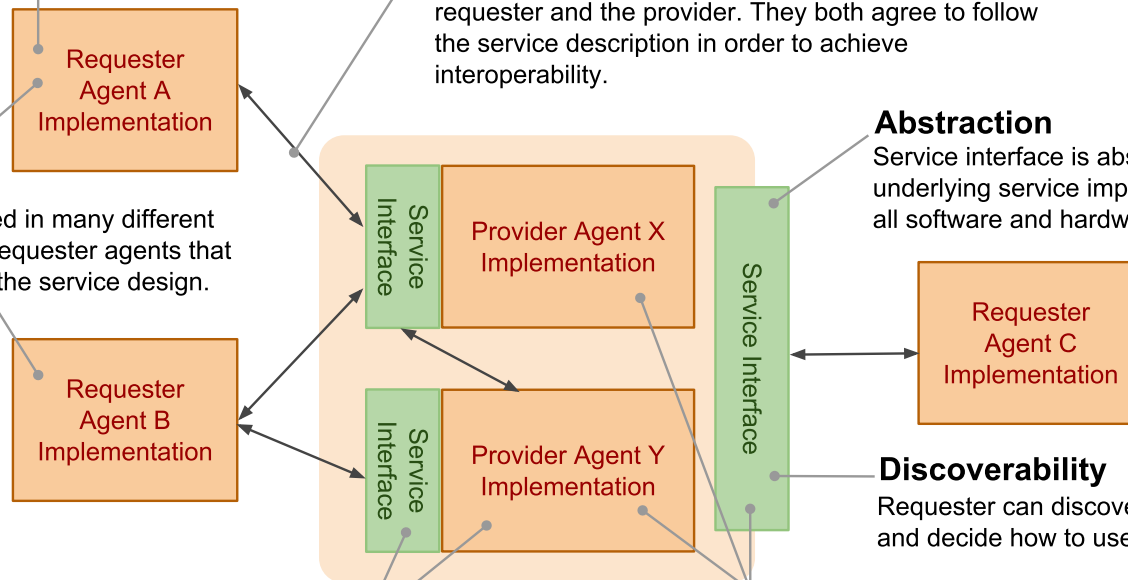
Requester can discover the service interface and decide how to use it.

Encapsulation

The provider agent implementation is hidden to the requester agent accessing the service. The requester agent only knows the service interface to consume its functionality.

Composability

It is possible to compose services into more complex processes. Such processes can again be accessed as services.

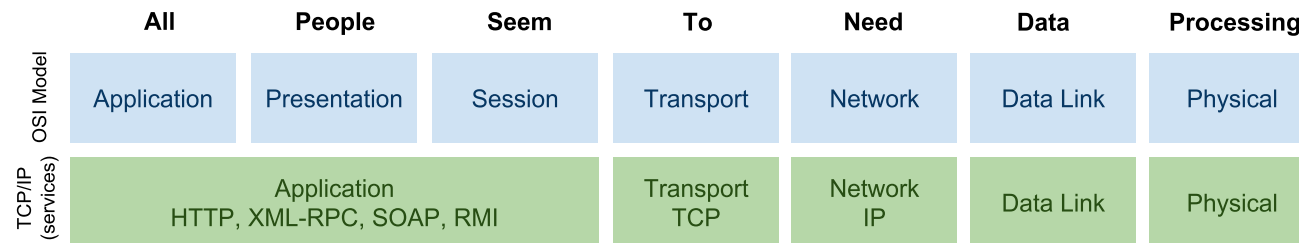


Overview

- Integrating Applications
- Service Definition
- **Service Communication**
- REST

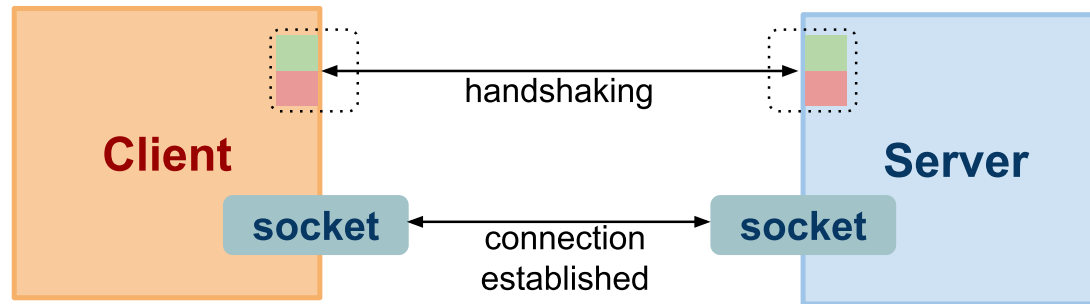
Application Protocols

- Remember this



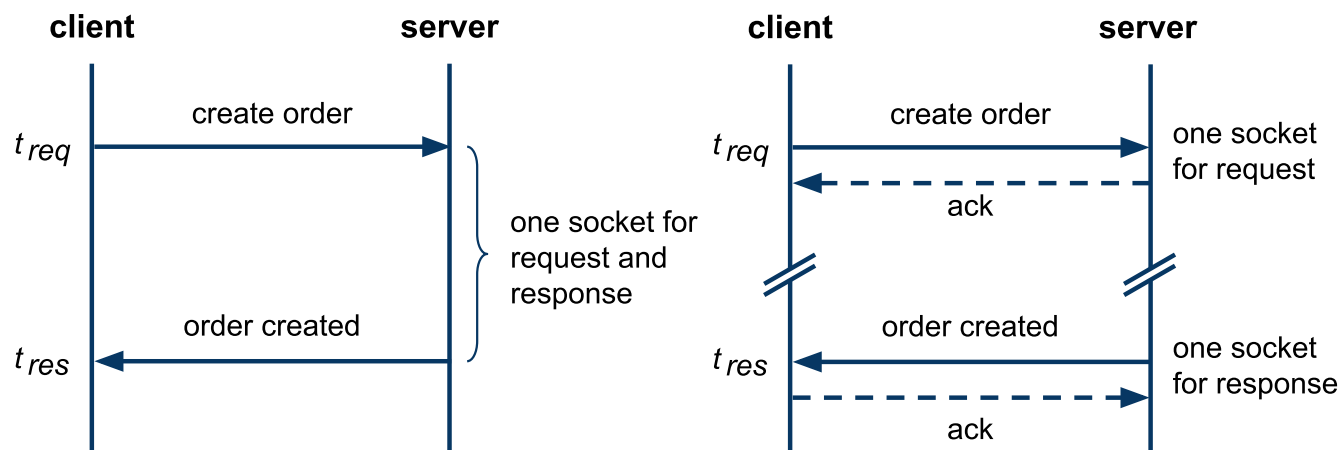
- App protocols mostly on top of the TCP Layer
 - *use TCP socket for communication*
- Major protocols
 - *HTTP – most of the app protocols layered on HTTP*
 - *wide spread, but: implementors often break HTTP semantics*
 - *RMI – Remote Method Invocation*
 - *Java-specific, rather interface*
 - *may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

Socket



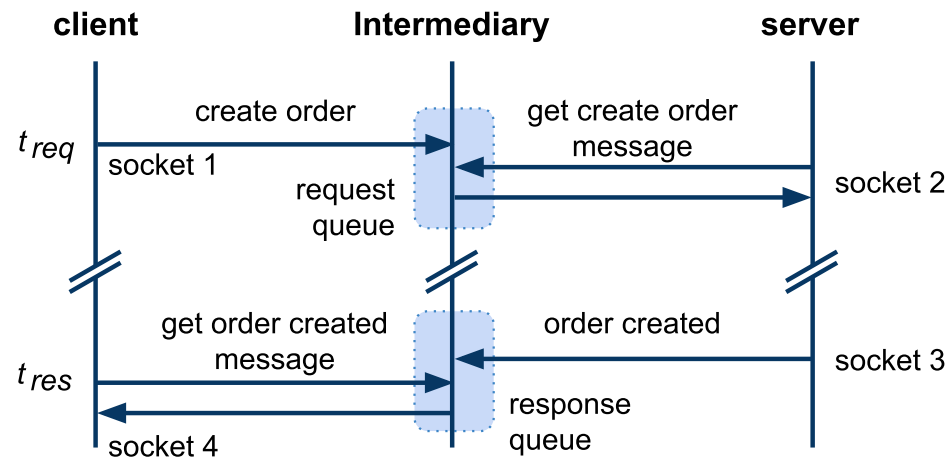
- Handshaking (connection establishment)
 - The server listens at `[dst_ip,dsp_port]`
 - Three-way handshake:
 - the client at `[src_ip,src_port]` sends a connection request
 - the server responds
 - the client acknowledges the response, can send data along
 - Result is a socket (virtual communication 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



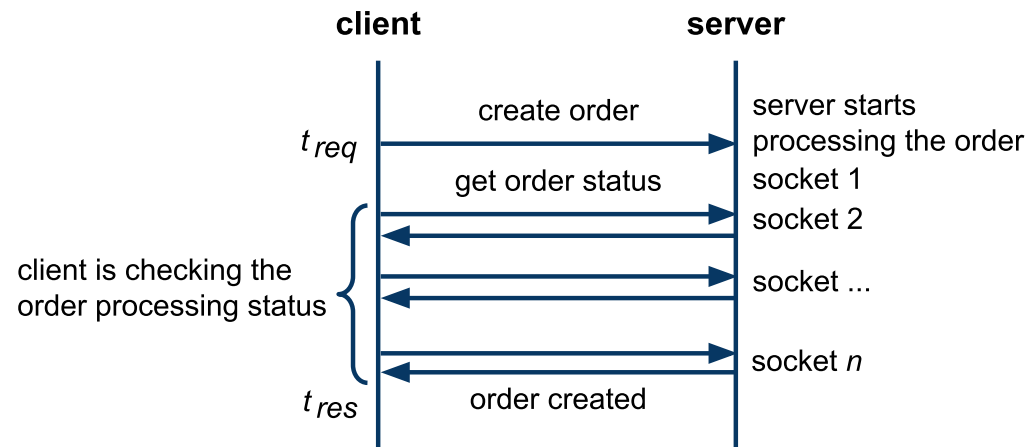
- Synchronous
 - one socket, $|t_{req} - t_{res}|$ is small
 - easy to implement and deploy, only standard firewall config
 - only the server defines endpoint
- Asynchronous
 - request, response each has socket, client and server define endpoints
 - $|t_{req} - t_{res}|$ can be large (hours, even days)
 - 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

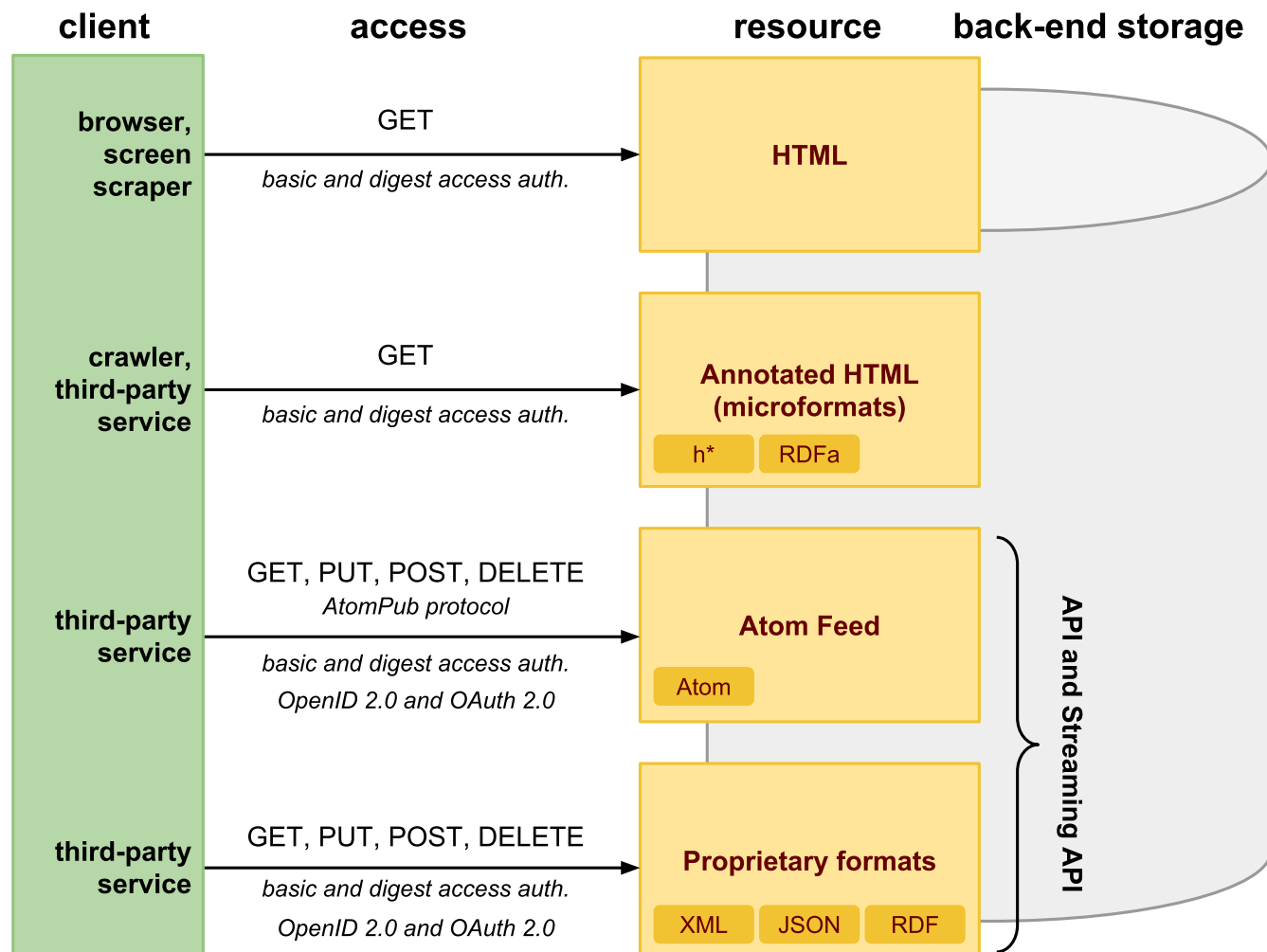


- Polling – only clients open sockets
 - *A client performs multiple request-response interactions*
 - *The first interaction initiates a process on the server*
 - *Subsequent interactions check for the processing status*
 - *The last interaction retrieves the processing result*
- Properties of environments
 - *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*

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

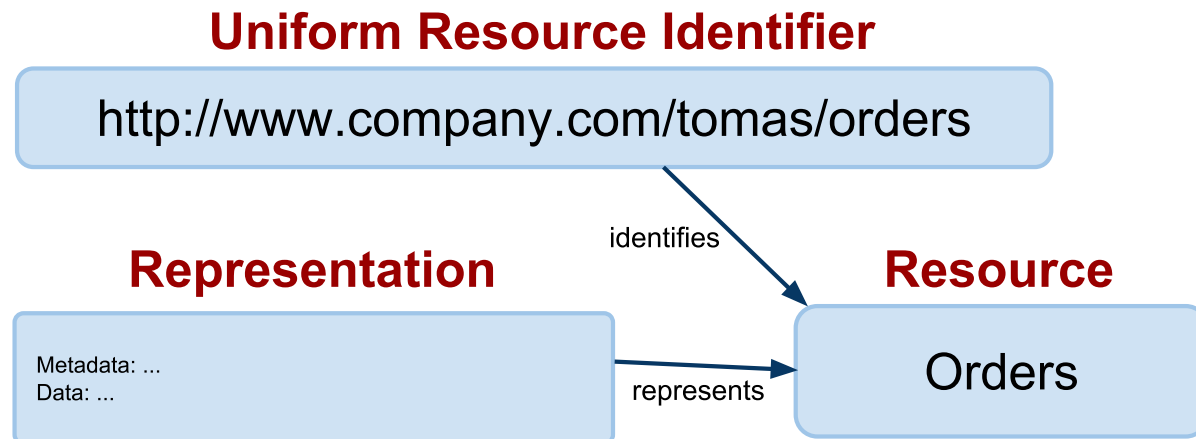
- Familiarity
 - *HTTP protocol is well-known and widely used*
- Interoperability
 - *All environments have HTTP client libraries*
 - *technical interoperability is thus no problem*
 - *no need to deal with vendor-specific interoperability issues*
 - *You can focus on the core of the integration problem*
 - *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

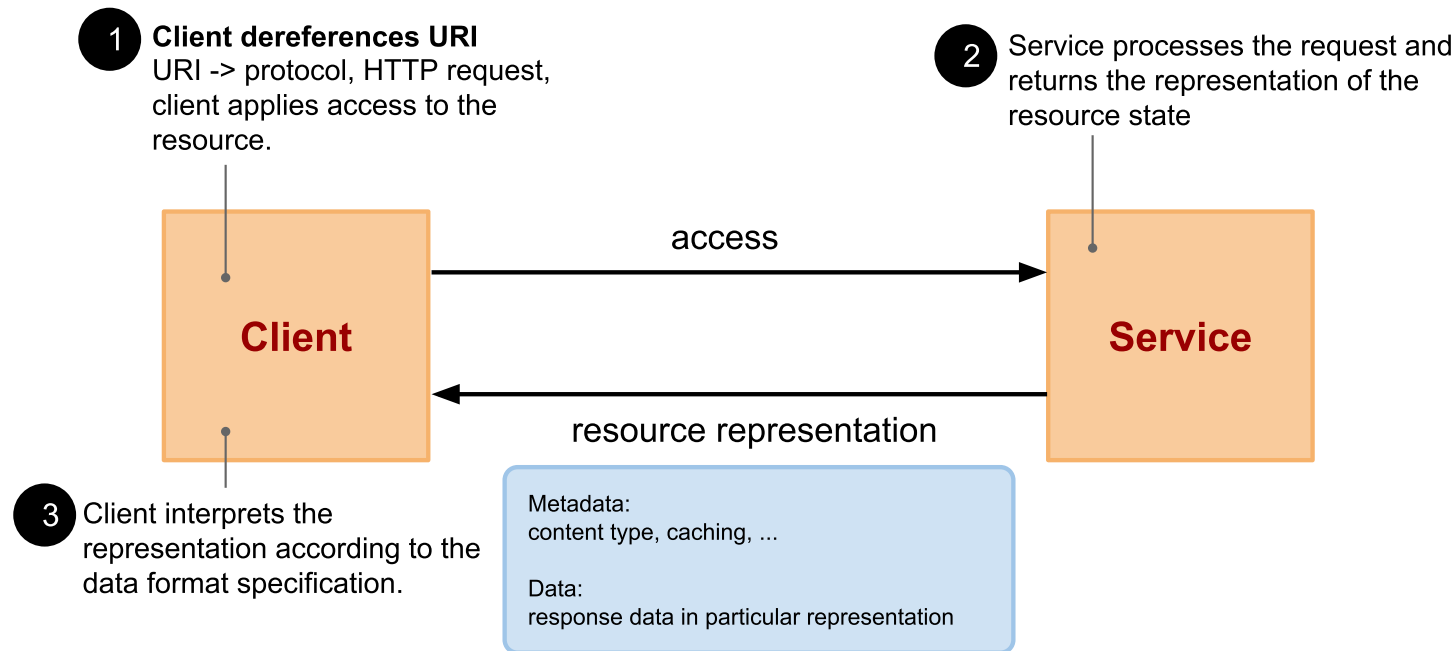
- REST architectural style defines constraints
 - *if you follow them, they help you to achieve a good design, interoperability and scalability.*
- Constraints
 - *Client/Server*
 - *Statelessness*
 - *Cacheability*
 - *Layered system*
 - *Uniform interface*
- Guiding principles
 - *Identification of 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*

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*

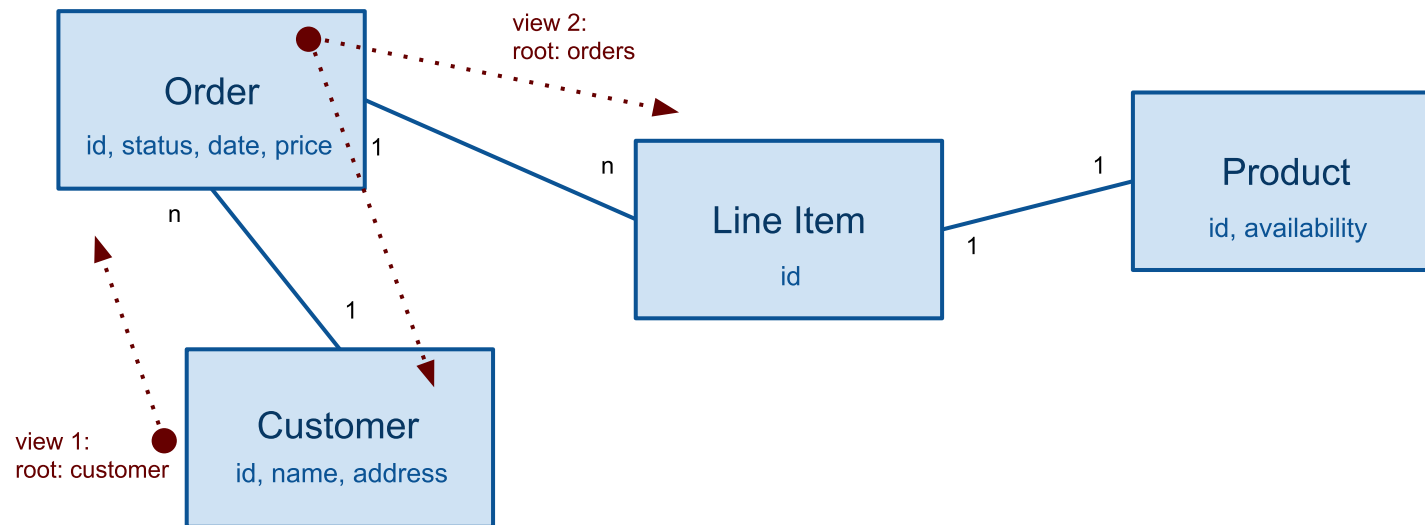
- *query – non-hierarchal form (parameters key=value)*

- 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`

- \Rightarrow *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*

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

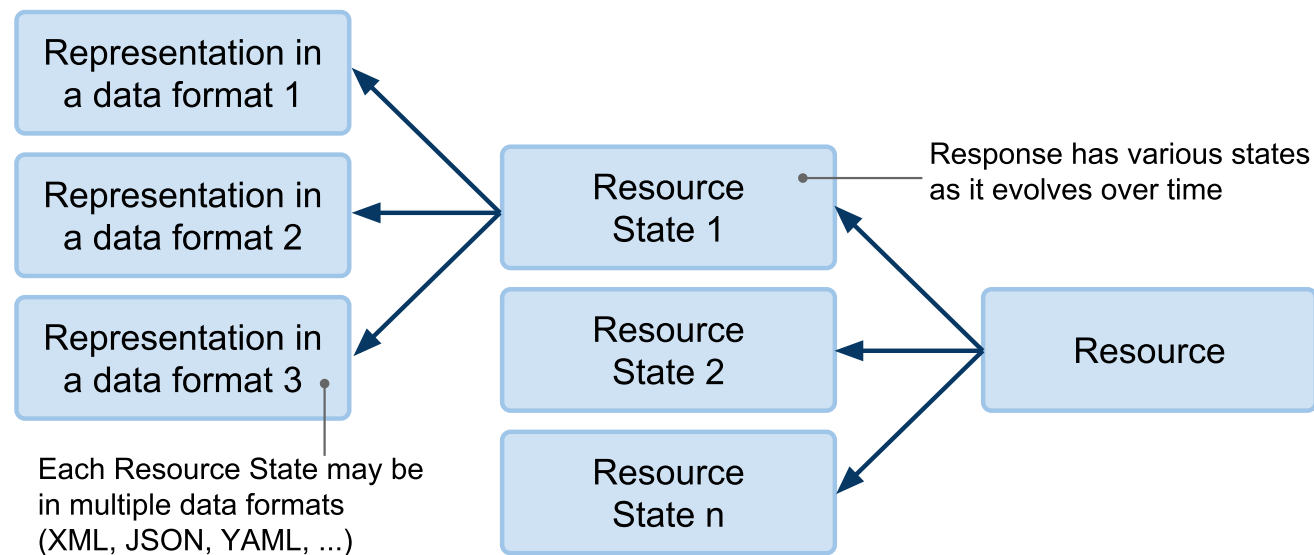
- Access
 - *to be retrieved (GET)*
 - *to be inserted or updated (PUT, POST)*
 - *to be deleted (DELETE)*
- Request
 - HTTP header **Accept**, part of content negotiation protocol
- Response
 - HTTP header **Content-Type: type/subtype; parameters**
 - Specifies an Internet Media Type [☞](#) of the resource 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)

```
1 <orders>
2   <order id="54467"/>
3   <order id="65432"/>
4 </orders>
```

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

```
1 <order>
2   ...
3 </order>
```

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

```
1 <orders>
2   <order id="54467"/>
3   <order id="65432"/>
4   <order id="74567"/>
5 </orders>
```

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

Overview

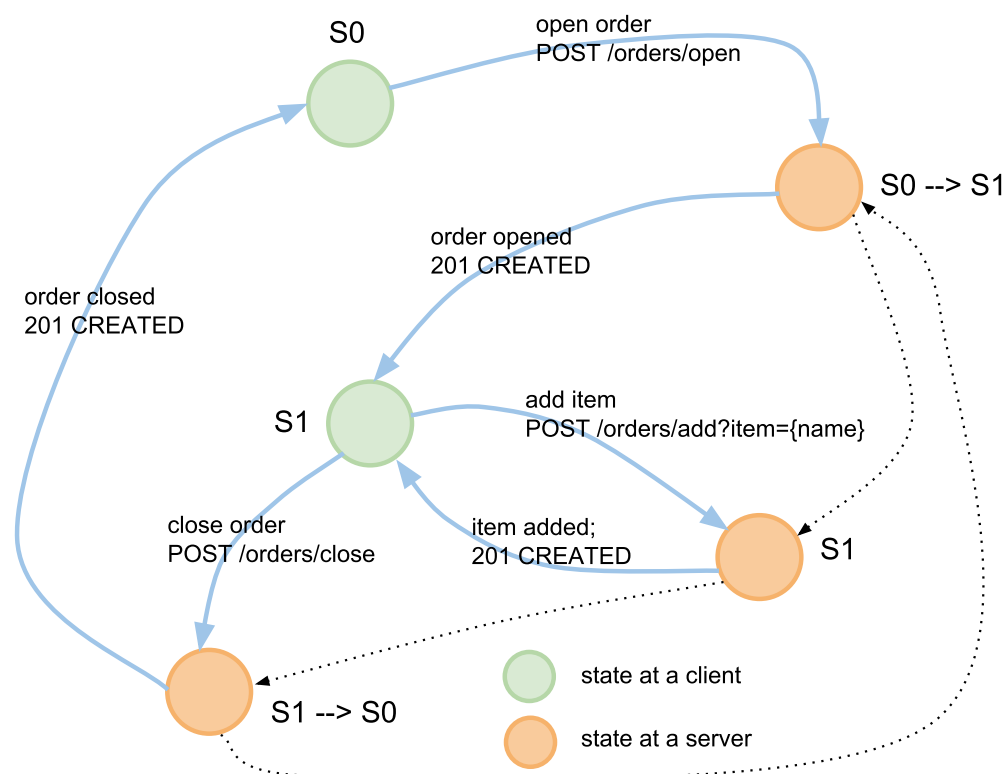
- Integrating Applications
- Service Definition
- Service Communication
- REST
 - *Introduction to REST*
 - *Uniform Resource Identifier*
 - *Resource Representation*
 - *HATEOAS*

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*

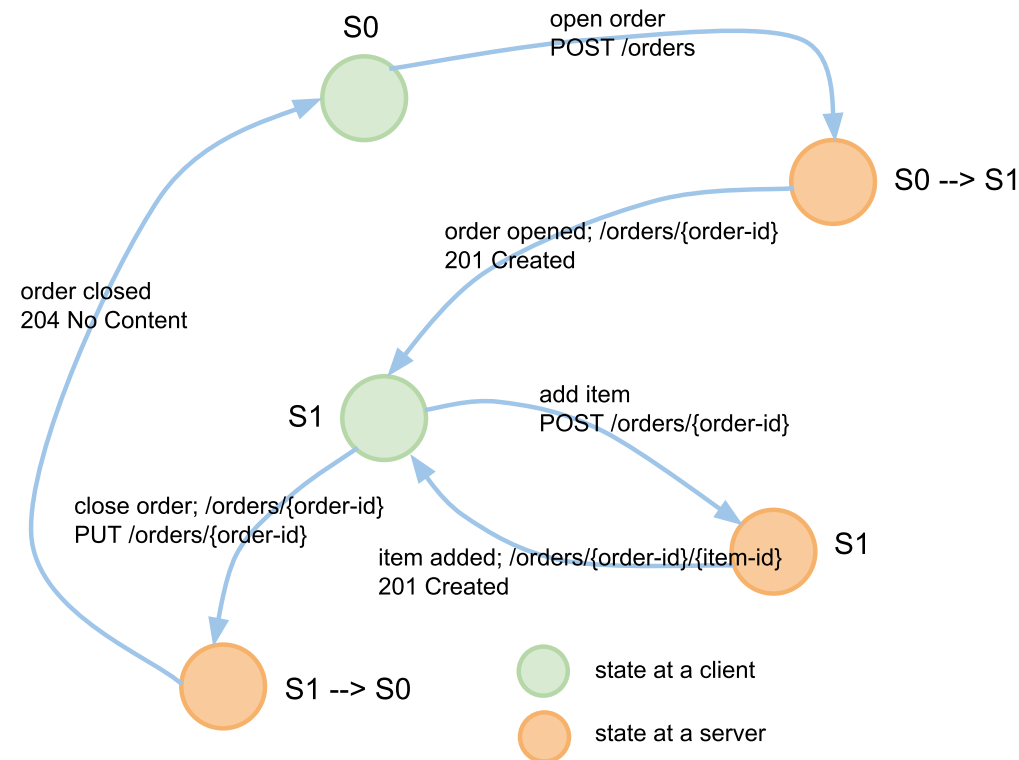
Stateful server

- Sessions to store the application state
 - Recall HTTP state management in MDW
 - The app uses a server memory to remember the state
 - When the server restarts, the app state is lost



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*

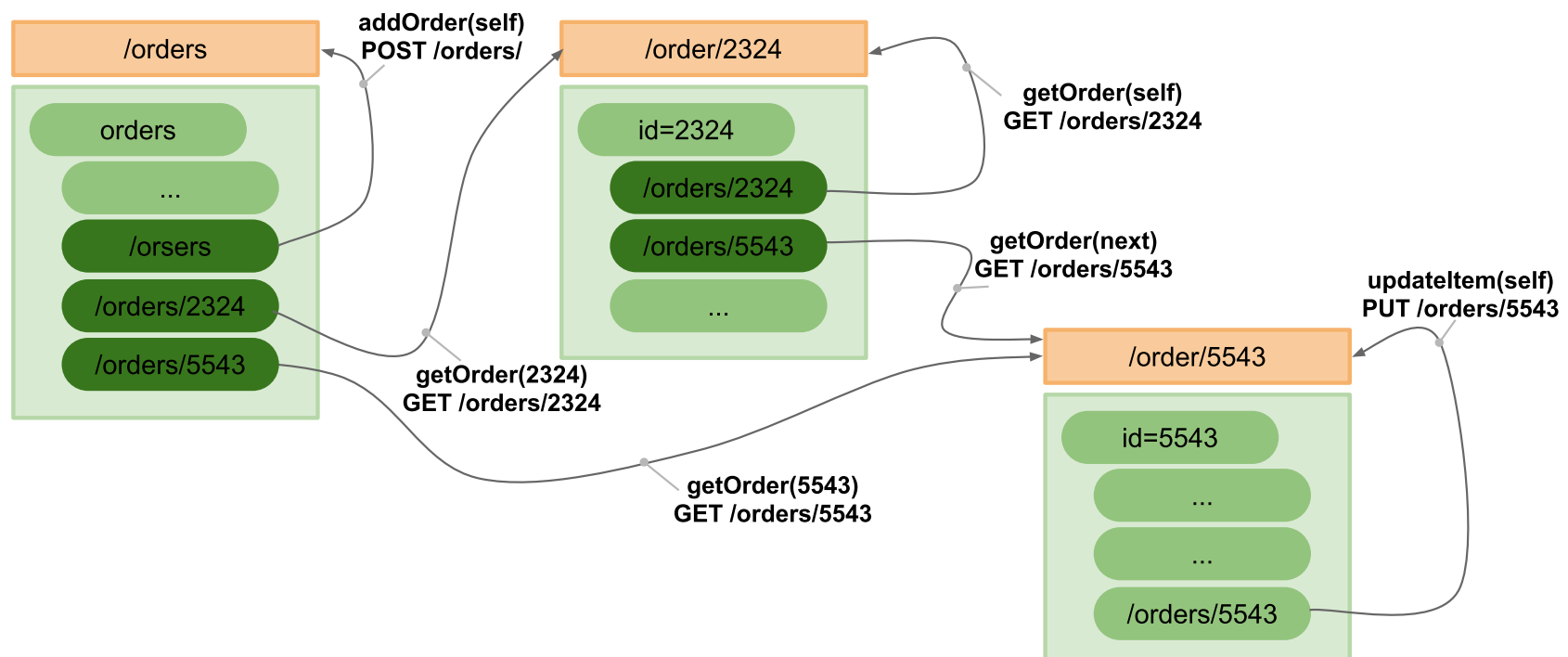


Persistent Storage and Session Memory

- 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
```

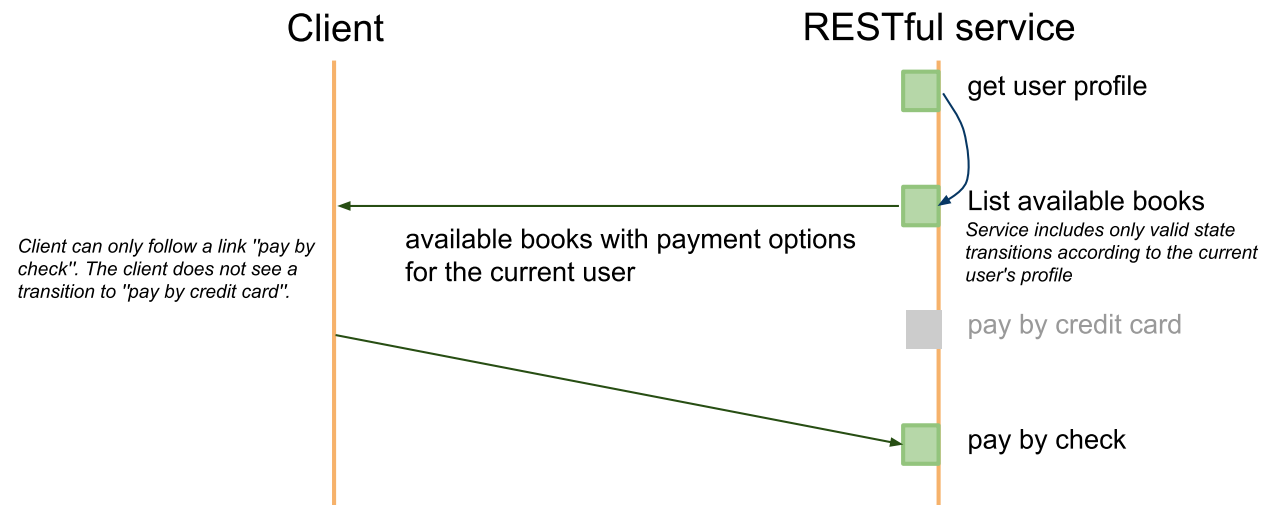
```
< Link: <http://company.com/orders/?page=2&size=10>; rel="next"
```

```
< 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*