Avancier

Avancier Methods (AM)

Software Architecture

Decoupling - part 2

(REST, EDA etc.)

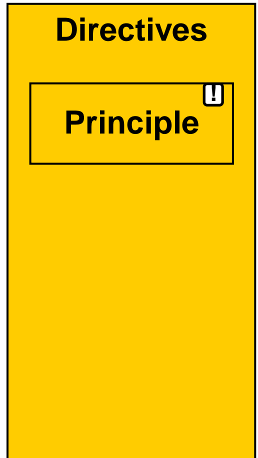
It is illegal to copy, share or show this document
(or other document published at http://avancier.co.uk)
without the written permission of the copyright holder

Copyright Avancier Limited 2007-2016

Principles example – a global organisation

Avancier

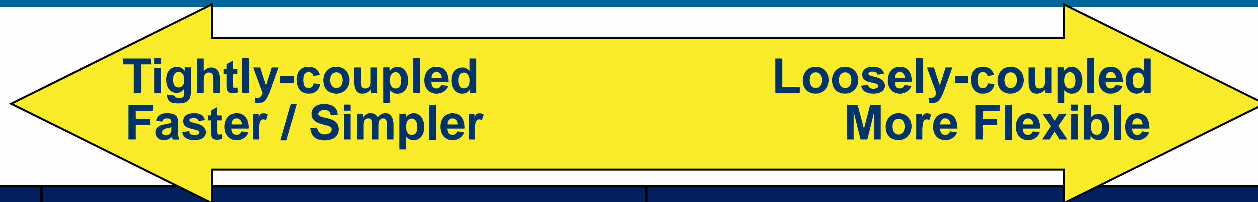
1. Separate concerns (for flexibility and maintainability)
2. Build for competitive advantage / Buy for competitive parity
3. Encapsulate components (for CBD and SOA)
4. Use open APIs for inter-component communication
5. Loosely couple components (for flexibility and availability)
6. Use Event-Driven Architecture for broadcast updates
7. Maintain a single source of truth
8. Design for response time / latency
9. Design for graceful failure – informing users
10. Web first: design for browser and client device independence



Copyright Avancier Limited 2007-2016

Decoupling part 1 - recap

Avancier



Feature	Early OO design presumptions	Recent SOA design presumptions
Naming	Clients use object identifiers One name space	Clients use domain names Multiple name spaces behind interfaces
Paradigm	Stateful objects/modules Reuse by OO inheritance Intelligent domain objects	Stateless objects/modules Reuse by delegation Intelligent process controllers
Time	Request-reply invocations Blocking servers	Asynchronous messaging Non-blocking servers
Location	Remember remote addresses	Use brokers/directories/facades

COBOL modules
Java objects
CORBA

Web Services

Copyright Avancier Limited 2007-2016

Representational State Transfer (REST)

- Roy Fielding defined REST
- We want application components to communicate across a network, and we use internet protocols
- Why not capitalise on what the internet is good at?
  - Naming accessible resources
  - Locating remote resources using a domain name
  - Communication using protocols such as HTTP (and others perhaps)
  - Representation of component state using *hypermedia*
    - a nonlinear medium of information which includes graphics, audio, video, plain text and hyperlinks.

Copyright Avancier 2007-2016

RESTful clients

- Every source/sender and target/receiver component is given a domain name

- **RESTful**
  - **Clients** call server components using network protocol operation names
    - HTTP: get, put, post and delete

- “This enables a distributed system to have desirable properties, such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability.”

Copyright Avancier 2007-2016

Typical RESTful web API HTTP methods (Wikipedia)

Resource	LIST	ITEM
GET **(read)**	**List**	**Retrieve**
PUT **(update)**	**Replace**	**Replace.**
POST **(create)**	**Create**	Not generally used.
DELETE **(delete)**	**Delete.**	**Delete**

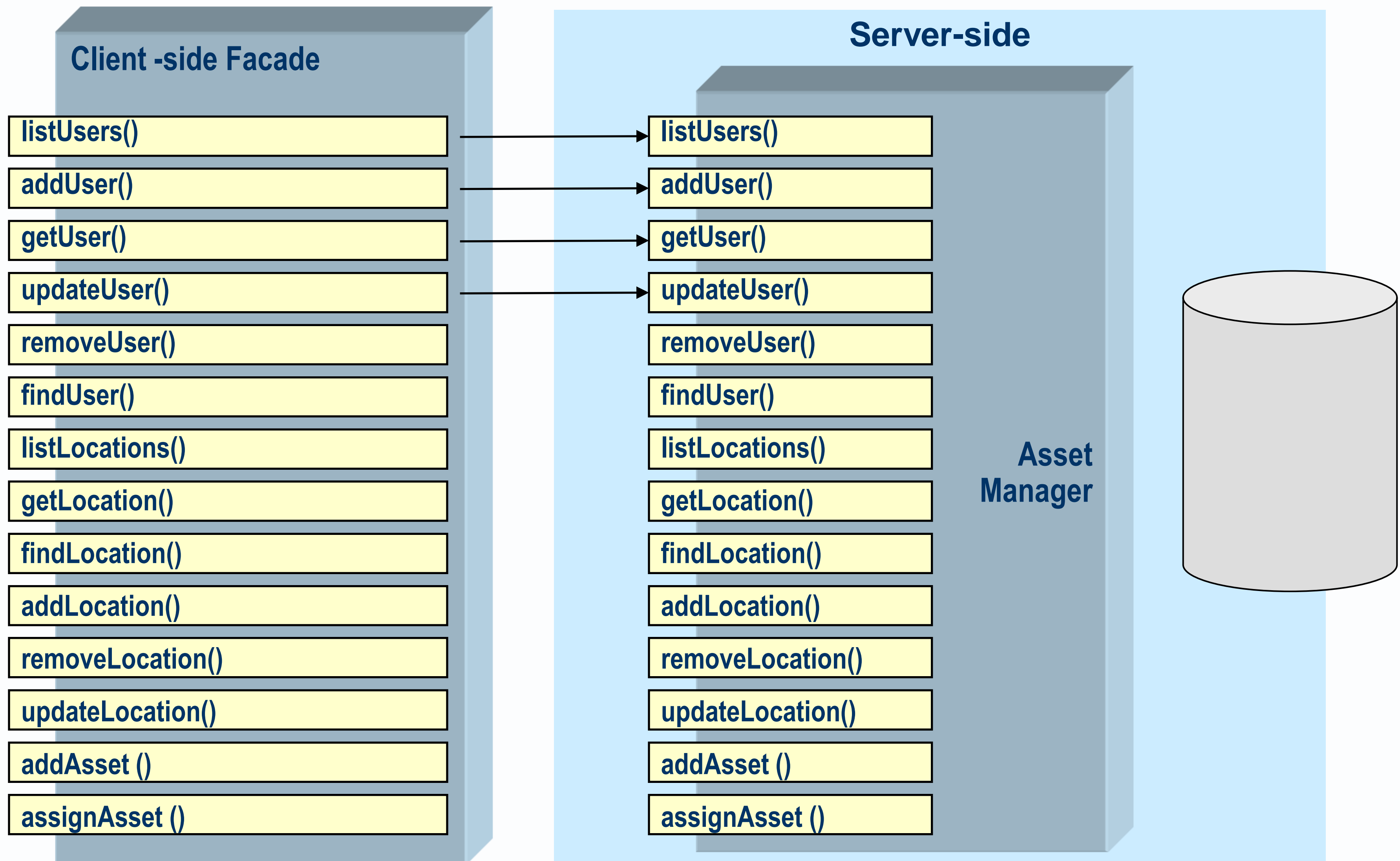
Copyright Avancier 2007-2016

Typical RESTful web API HTTP methods (Wikipedia)

Resource	**Collection URI,** such as http://example.com/resources	**Element URI,** such as http://example.com/resources/item17
GET (read)	**List** the URIs and perhaps other details of the collection's members.	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.
PUT (update)	**Replace** the entire collection with another collection.	**Replace** the addressed member of the collection, or if it doesn't exist, create it.
POST (create)	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	Not generally used. Treat the addressed member as a collection in its own right and create a new entry in it.
DELETE (delete)	**Delete** the entire collection.	**Delete** the addressed member of the collection.

Copyright Avancier 2007-2016

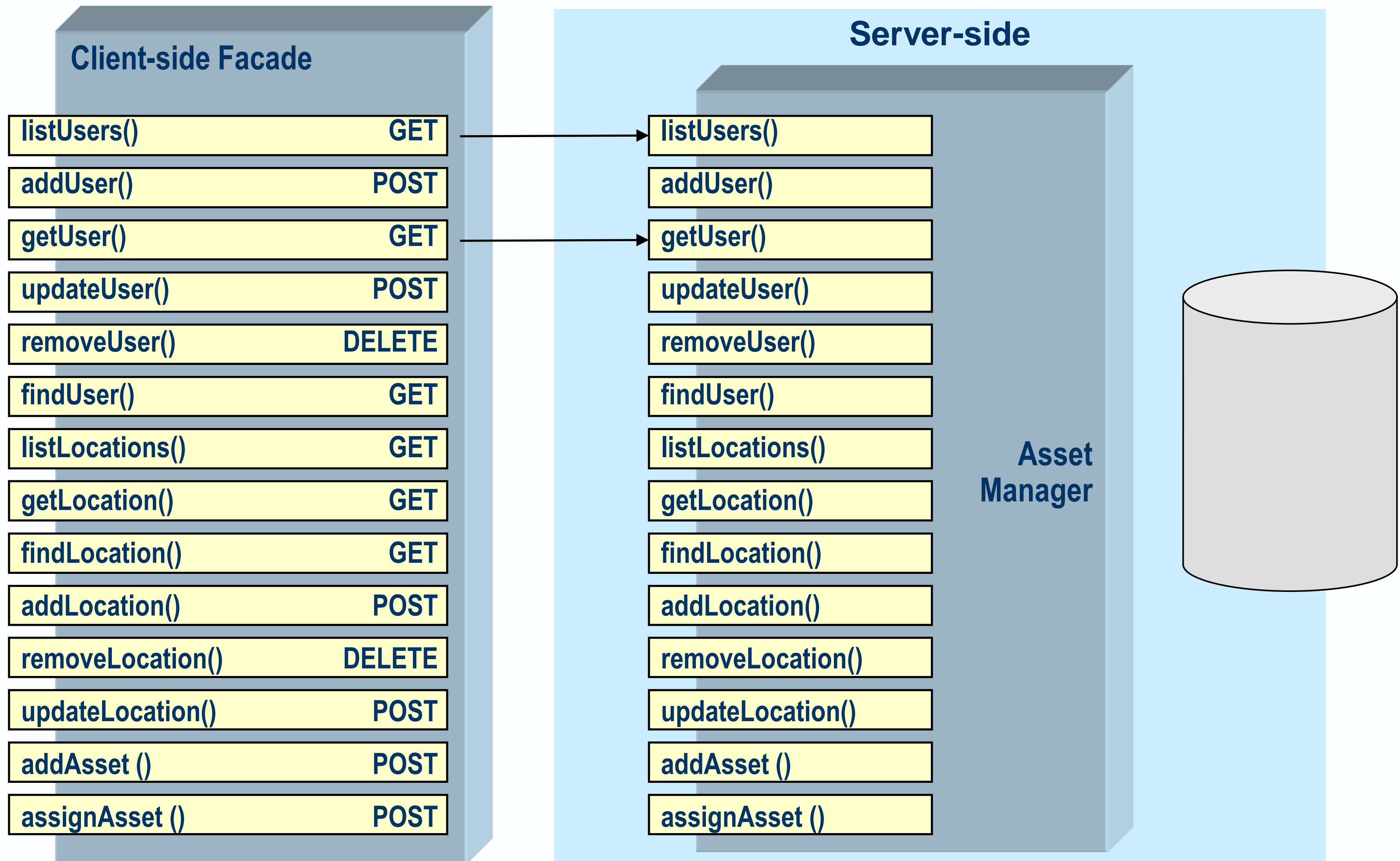
Traditional server-side component: one nouns, many verbs



Copyright Avancier 2007-2016

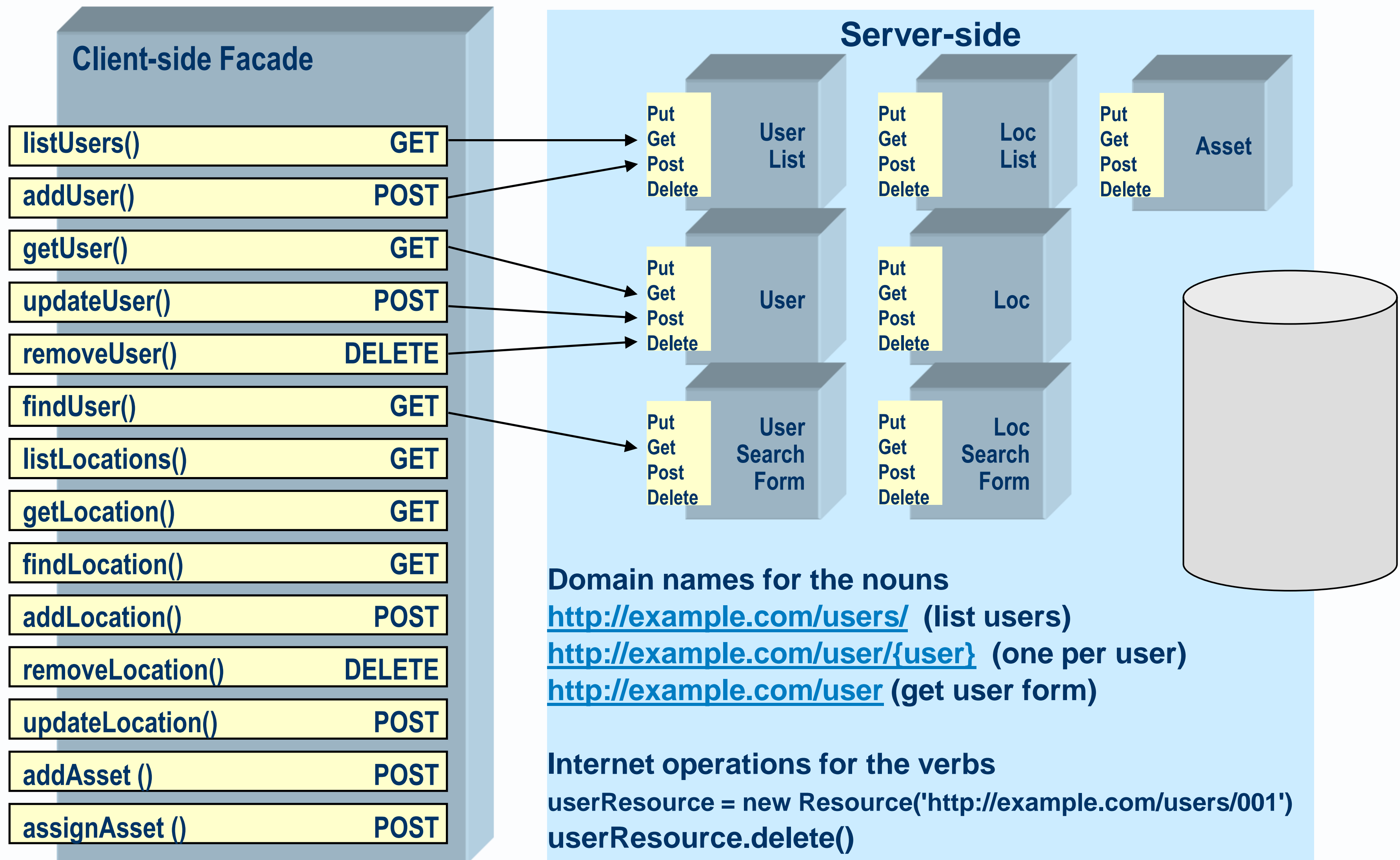
RESTful client: uses HTTP operation names BUT *sends a parameter to qualify it*

Avancier



Copyright Avancier 2007-2016

REST-compliant servers: each has 1 noun and 4 verbs



Copyright Avancier 2007-2016

RESTful and REST-compliant

- Every source/sender and target/receiver component is given a domain name

- **RESTful**
  - **Clients** call server components using network protocol operation names
    - HTTP: get, put, post and delete

- **REST-compliant**
  - **Server** components offer **only** the operations above
  - With no paramaterised variations

Copyright Avancier 2007-2016

RESTful and REST-compliant Web Services

- A Web Service
  - Has a URI
    - such as http://example.com/resources
  - Uses internet-friendly I/O data flow formats
    - Usually XML or JSON

- *An arbitrary* Web Service
  - Offers operations using HTTP methods
    - e.g. GET, PUT, POST, or DELETE
  - But offers any number of operations (parameterised)

- *A REST-compliant* Web service
  - Offers a uniform set of "stateless" operations.

Copyright Avancier 2007-2016

Would you use REST where you need

- Fast response time?
- High availability?
- Guaranteed message delivery?
- Transaction rollback?
- Security?

Copyright Avancier 2007-2016

Would you use REST where you need

- Fast response time?
  - It depends what you compare it with
- High availability?
  - Probably
- Guaranteed message delivery?
  - No (unless you do it by hand)
- Transaction rollback?
  - No (unless you do it by hand)
- Security?
  - HTTP is not secure

Copyright Avancier 2007-2016

More decoupling varieties

Often faster and/or simpler

Often more flexible, but more complex

Avancier

Feature	Tight coupling	Decoupling techniques
Data types	Complex data types	Simple data types
Version	Version dependency	Design to avoid version dependence Apply the open-closed principle
Protocol	Protocol dependency	Design for multiple protocols
Integrity constraints	ACID transactions	BASE: compensating transactions and eventual consistency

Copyright Avancier Limited 2007-2016

Decoupling from complex data type

Feature	Tight coupling	Decoupling techniques
Data types	Complex data types Date: **DD/MM/YY** Address: **5 line address**	Simple data types Date: **days, months and years** Address: **premises, road, town, county/state, country**

Faster and simpler

More flexible and complex

Copyright Avancier Limited 2007-2016

Decoupling by version

Feature	Tight coupling	Decoupling techniques
Version	Version dependency	Design to avoid version dependence Apply the open-closed principle
Protocol	Protocol dependency	Design for multiple protocols

More flexible and complex

Copyright Avancier Limited 2007-2016

Decoupling from integrity/consistency

Avancier

Feature	Tight coupling	Decoupling techniques
Integrity constraints	ACID transactions Impractical when one process spans distributed components Or process volumes are extremely high	BASE: compensating transactions and eventual consistency Decouples transactions within a process and performs them separately within a longer workflow •improves availability of process start •improves scalability •at the cost of consistency •requires **compensating transactions** Most business are not like ebay and Amazon!

Simpler

More flexible and complex

Copyright Avancier Limited 2007-2016

Decoupling part 1 + 2

Often faster and/or simpler

Often more flexible, but more complex

Avancier

Feature	Tight coupling	Decoupling techniques
Naming	Clients use object identifiers One name space	Clients use domain names Multiple name spaces behind interfaces
Paradigm	Stateful objects/modules Reuse by OO inheritance Intelligent domain objects	Stateless objects/modules Reuse by delegation Intelligent process controllers
Time	Request-reply invocations Blocking servers	Asynchronous messaging Non-blocking servers
Location	Remember remote addresses	Use brokers/directories/facades
Data types	Complex data types	Simple data types
Version	Version dependency	Design to avoid version dependence Apply the open-closed principle
Protocol	Protocol dependency	Design for multiple protocols
Integrity constraints	ACID transactions	BASE: compensating transactions and eventual consistency

Copyright Avancier Limited 2007-2016

Finally, Event-Driven Architecture (EDA)

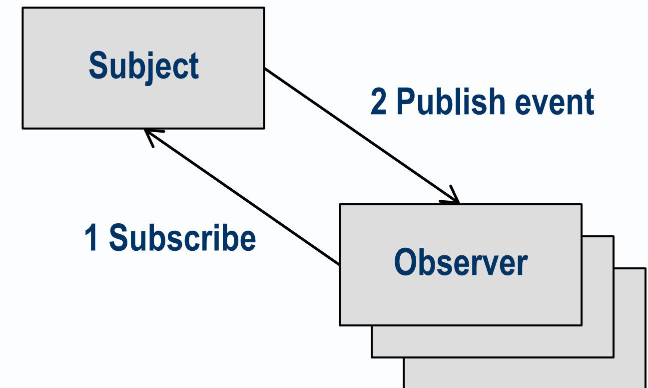
- We've mostly been discussing **request-reply** transactions
- What about **event notification** messages?

Copyright Avancier Limited 2007-2016

Observer (notification, pub/sub) pattern

Avancier

- **Observer**
  - A subject
    - notifies observers of changes to its state
  - Observers
    - register with the subject to be notified of changes.
    - unregister when no longer interested

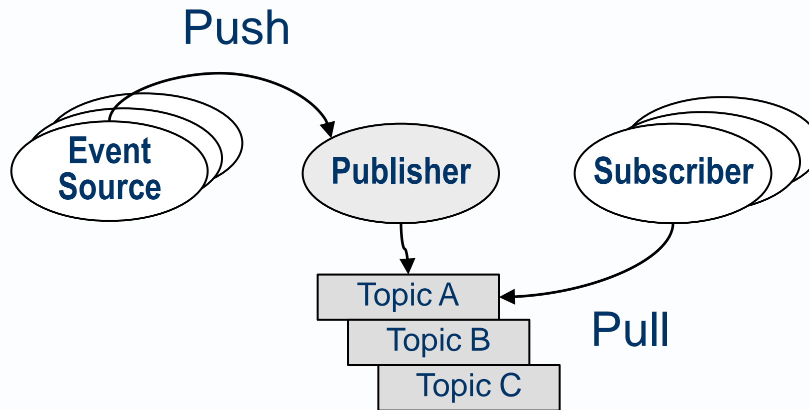


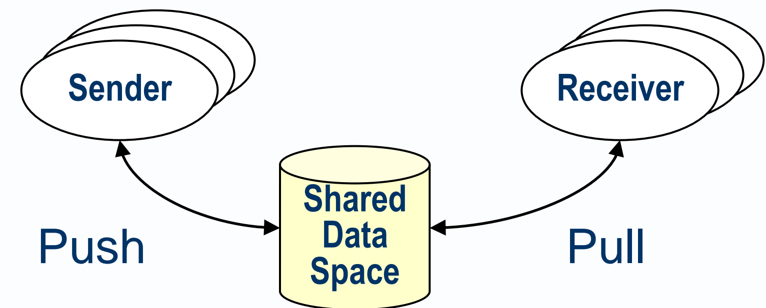
Compare with EDA (hidden)

Copyright Avancier 2013

Event-Driven Architecture (EDA)

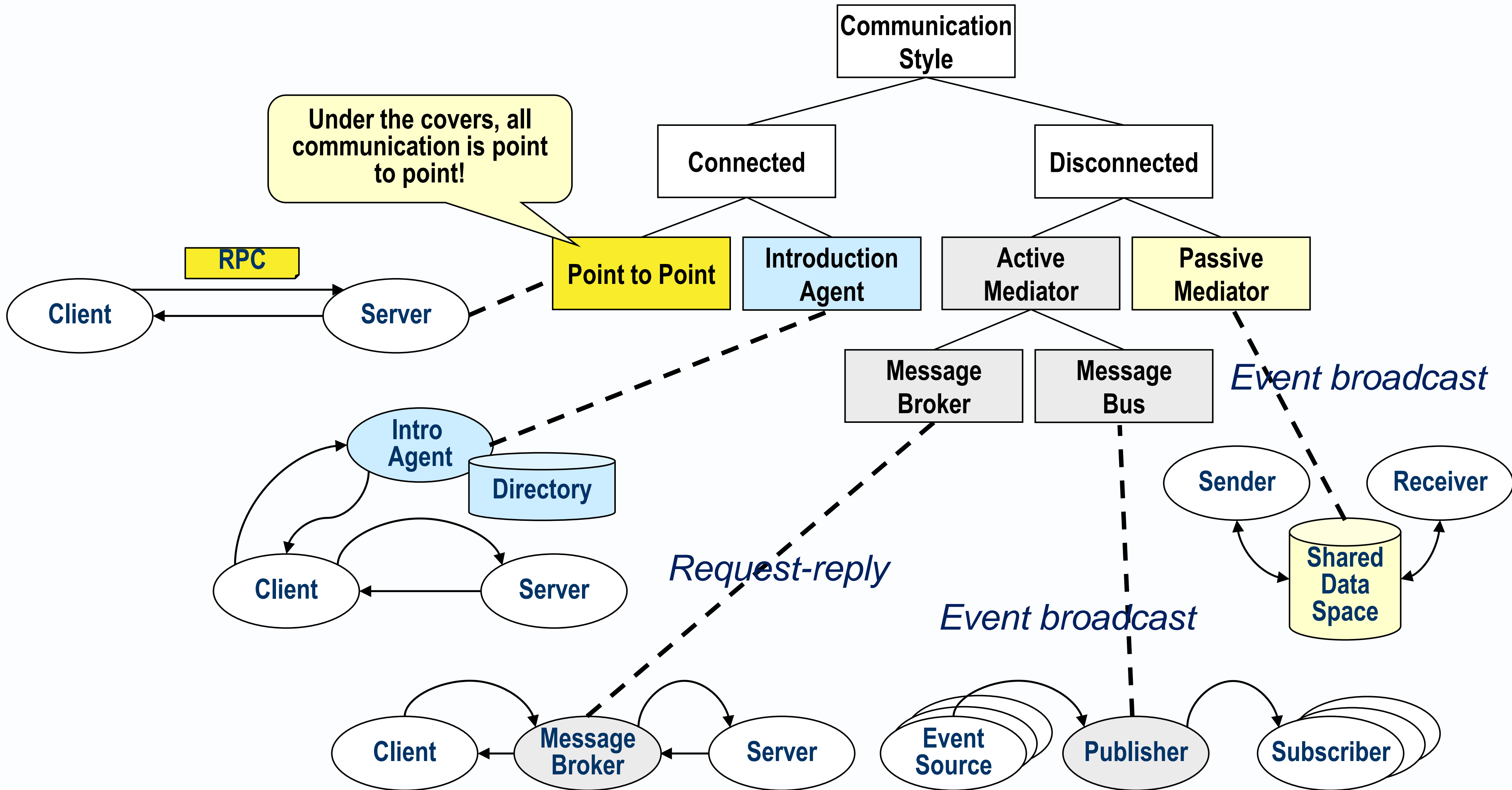
- Two styles





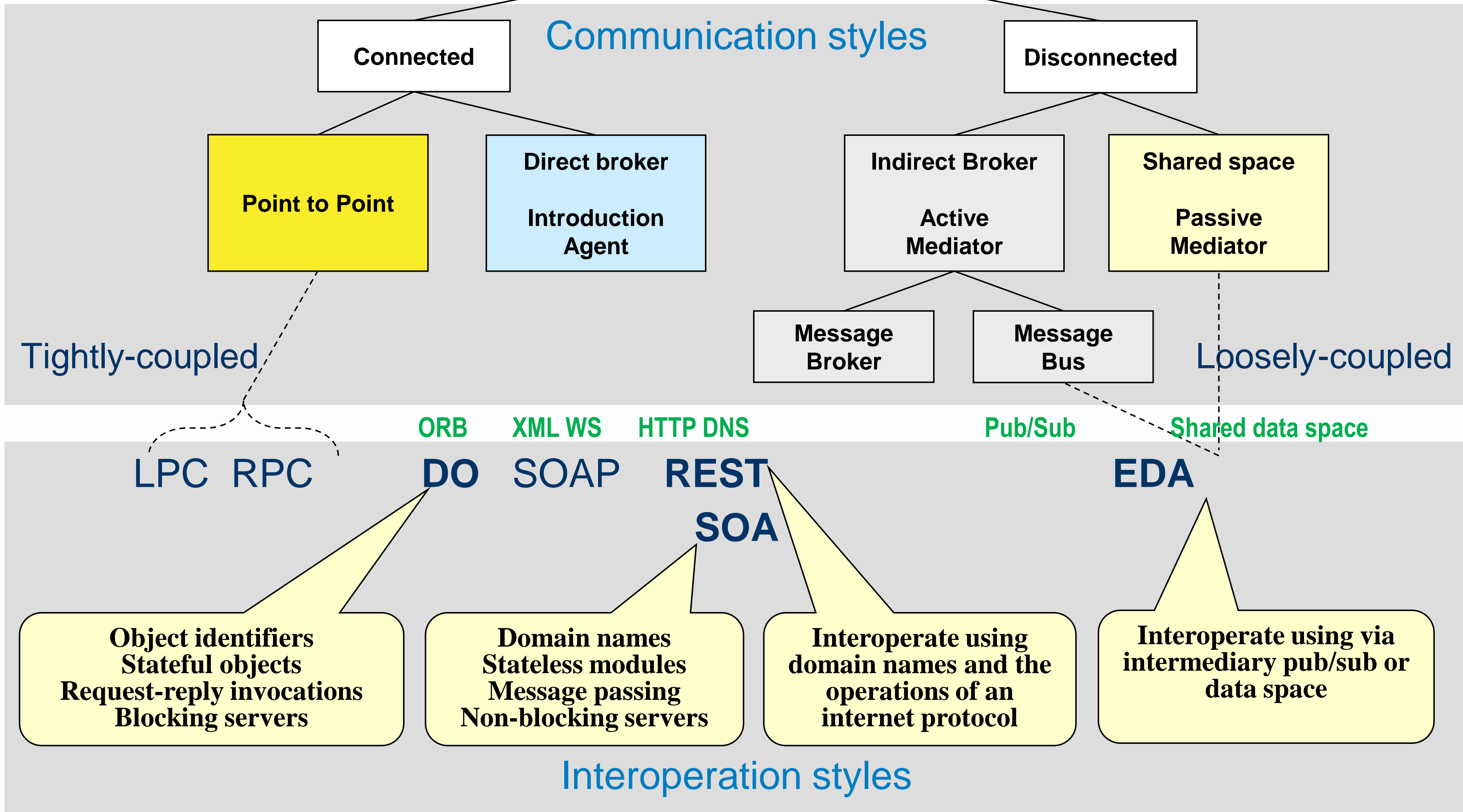
Copyright Avancier Limited 2007-2016

Communication styles – summary overview



Copyright Avancier Limited 2007-2016

Summary



Avancier

Copyright Avancier Limited 2013

Footnotes

- Left overs

Copyright Avancier 2007-2016

A *questionable* family tree of software architecture concepts

Avancier

Modular design

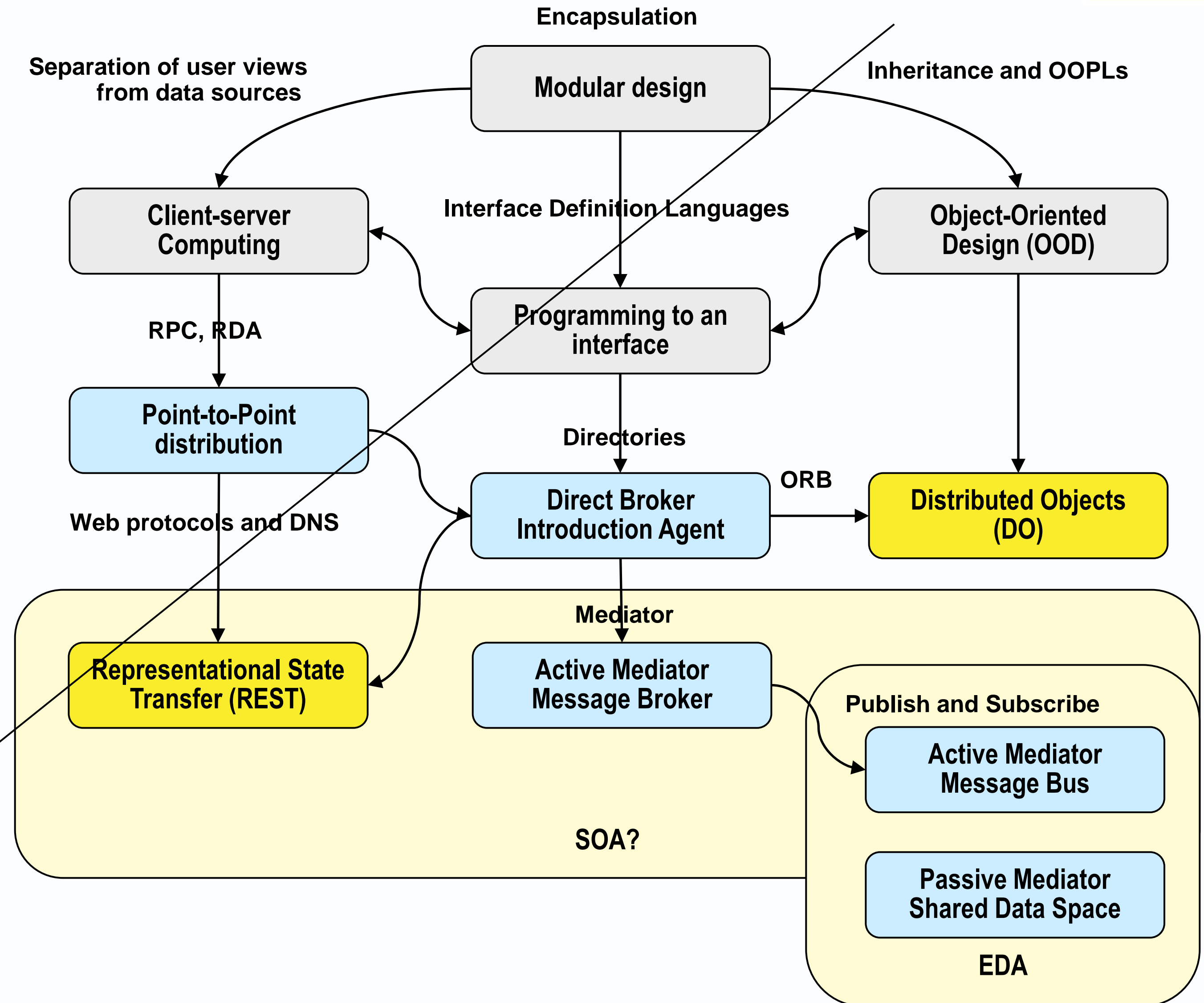
Procedural & OO variations

Decoupling of technology

Decoupling of location

Decoupling of acquaintance

Decoupling of activity



Copyright Avancier 2007-2016

The architectural properties of REST

- **Client–server (separation of concerns)**
  - Servers and clients can be developed and replaced independently, as long as the interface between them is not altered.
- **Stateless**
  - A server holds no client context between requests
  - Session state is held in the client, or in a database.
  - The client sends a request when it is ready to transition to a new state.
  - While requests are outstanding, the client is considered to be *in transition*.
  - The representation returned contains links the client may use initiate a new state-transition.
- **Cacheable**
  - Responses must define themselves as cacheable, or not, to prevent clients from reusing stale or inappropriate data
- **Layered system**
  - A client cannot tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability load balancing, providing shared caches and enforce security policies.
- **Code on demand (optional)**
  - Servers can temporarily extend or customize the functionality of a client by the transfer of executable code (e.g. Java applets and client-side scripts such as JavaScript.)

Copyright Avancier 2007-2016

Also, Uniform interface

- Identification of resources
  - Requests identify resources, usually using URIs
  - A server may **represent its state** in HTML, XML or JSON (none of which are the server's internal representation)
- Manipulation of resources through these representations
  - When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource.
- Self-descriptive messages
  - Each message includes enough information to describe how to process the message.
  - Responses also explicitly indicate their cacheability.
- Hypermedia as the engine of application state
  - Clients make state transitions only through actions that are dynamically identified within hypermedia by the server (e.g., by hyperlinks within hypertext).
  - A client does not assume that any particular action is available for any particular resources beyond those described in representations previously received from the server.

Copyright Avancier 2007-2016

Interoperation styles *versus* technologies

Avancier

- Different interoperation styles have been topical at different times and places.
- There has been drift from more closely coupled to more loosely coupled.

- How to differentiate styles from the technologies?

- **DO, SOA and EDA as styles**
  - We distinguish styles by implications they have for concepts such as object references, statefulness, synchronicity and message passing.
  - This make the concepts examinable without implying any particular standard or technology (XML, SOAP, HTTP, WS or ESB).

- **REST - part style - part technology**
  - REST does come with presumptions about using universal internet protocols and technologies.
  - REST is usually contrasted with SOAP rather than SOA.
  - REST enables a loosely–coupled SOA that comes with various limitations.

Copyright Avancier 2007-2016