UNIVERSIDAD
NACIONAL
DE COLOMBIA

Software Architecture

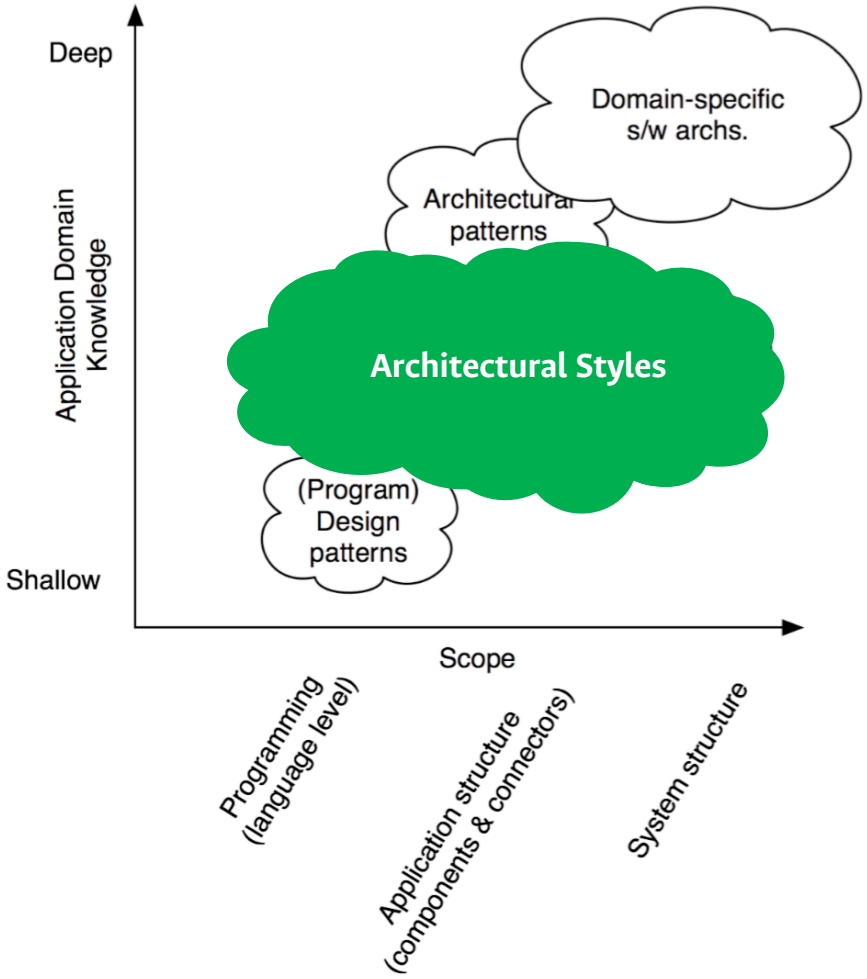
Architectural Styles

Jeisson Andrés Vergara Vargas
Departamento de Ingeniería de Sistemas e Industrial
http://colswe.unal.edu.co/~javergarav/
javergarav@unal.edu.co

2021-I

Architectural Styles

Context



Architectural Styles

Definition

An **architectural style** is a named collection of **architectural design decisions** that:

- Are applicable in a given **development context**.
- Constrain **architectural design decisions** that are specific to a particular system within that context.
- Elicit **beneficial qualities** in each resulting system.

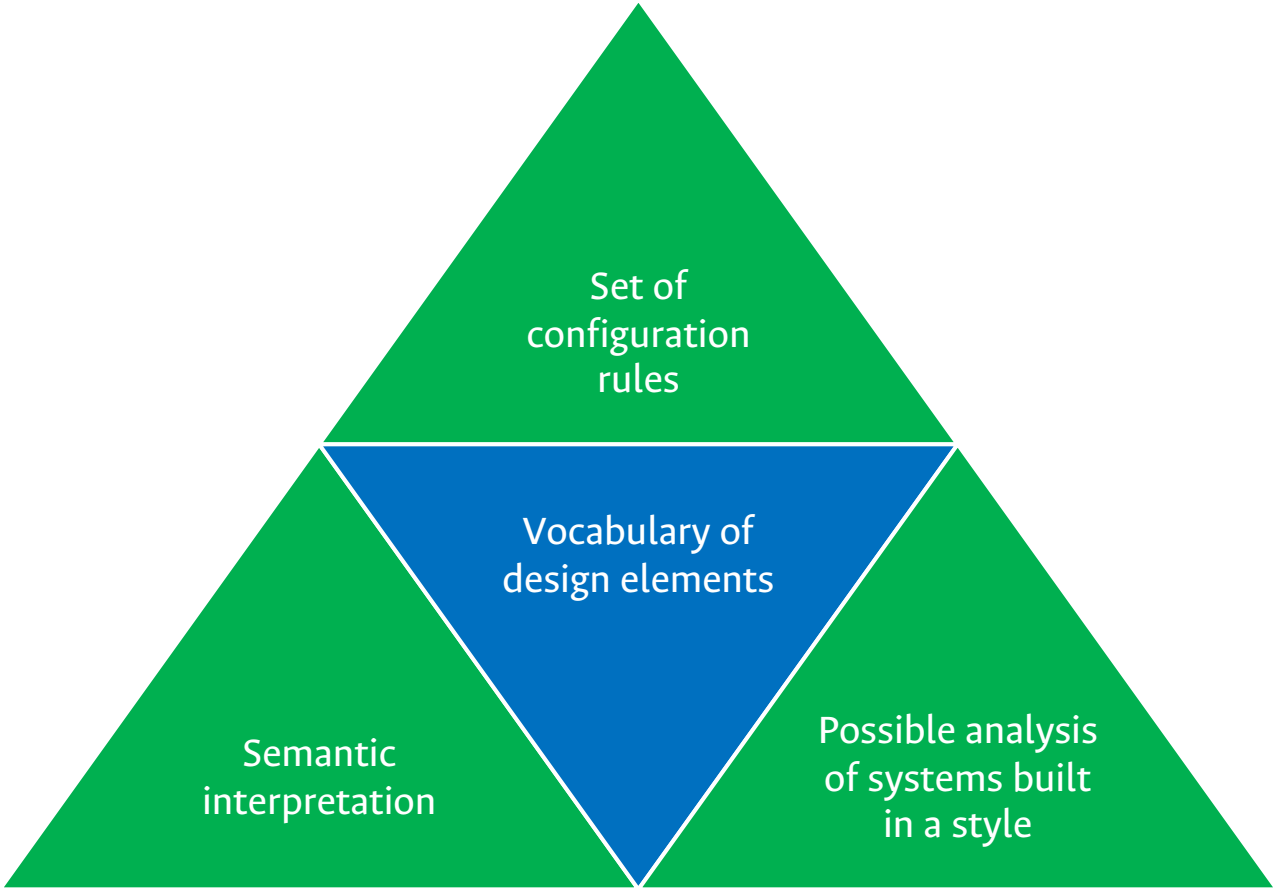
A primary way of **characterizing lessons** from **experience** in software system design.

Reflect **less domain specificity** than **architectural patterns**.

Useful in determining everything from subroutine structure to **top-level** application structure.

Architectural Styles

Discovering Styles



Architectural Styles

Benefits of Using Styles

Design reuse

Well-understood solutions applied to new problems.

Code reuse

Shared implementations of invariant aspects of a style.

Understandability of system organization

A phrase such as “client-server” conveys a lot of information.

Interoperability

Supported by style standardization.

Style-specific analysis

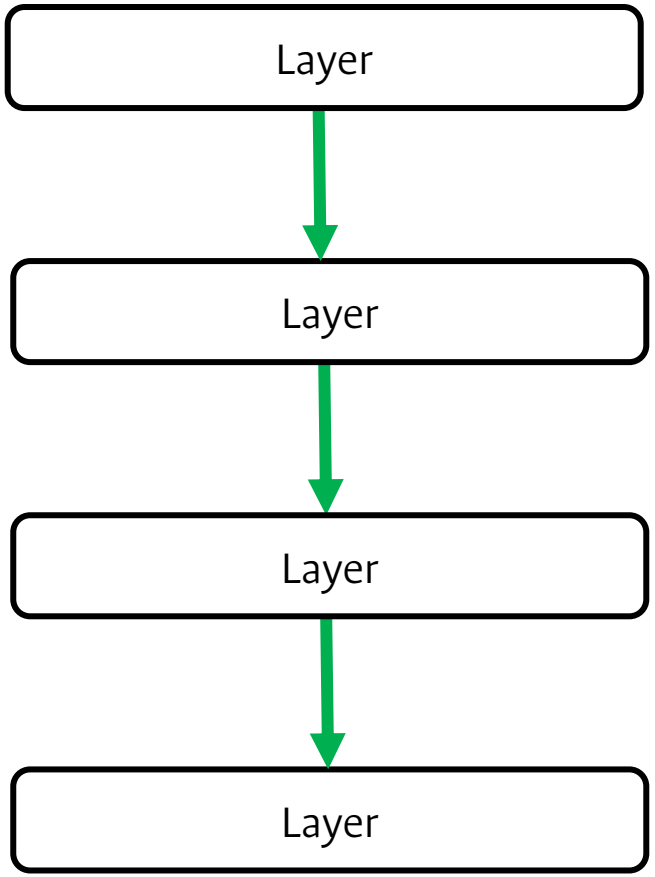
Enabled by the constrained design space.

Visualizations

Style-specific depictions matching engineers’ mental models.

Layered Style
(Layered Architecture)

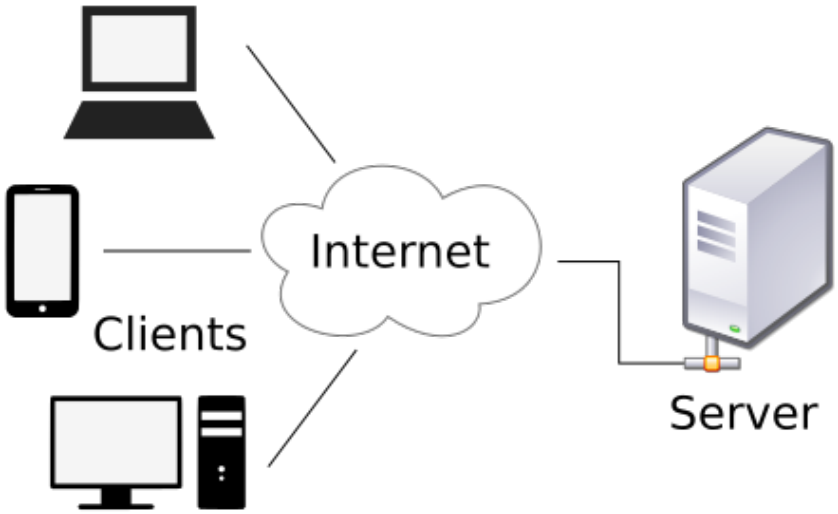
- Hierarchical system organization.
- "Multi-level client-server".
- **Components** are **layers**, each layer exposes an interface (API) to be used by above layers.
- Each layer acts as a:
  - **Server**: service provider to layers "above".
  - **Client**: service consumer of layer(s) "below".
- **Connectors** are **protocols of layer interaction**.



Client-Server Style

(Client-Server Architecture)

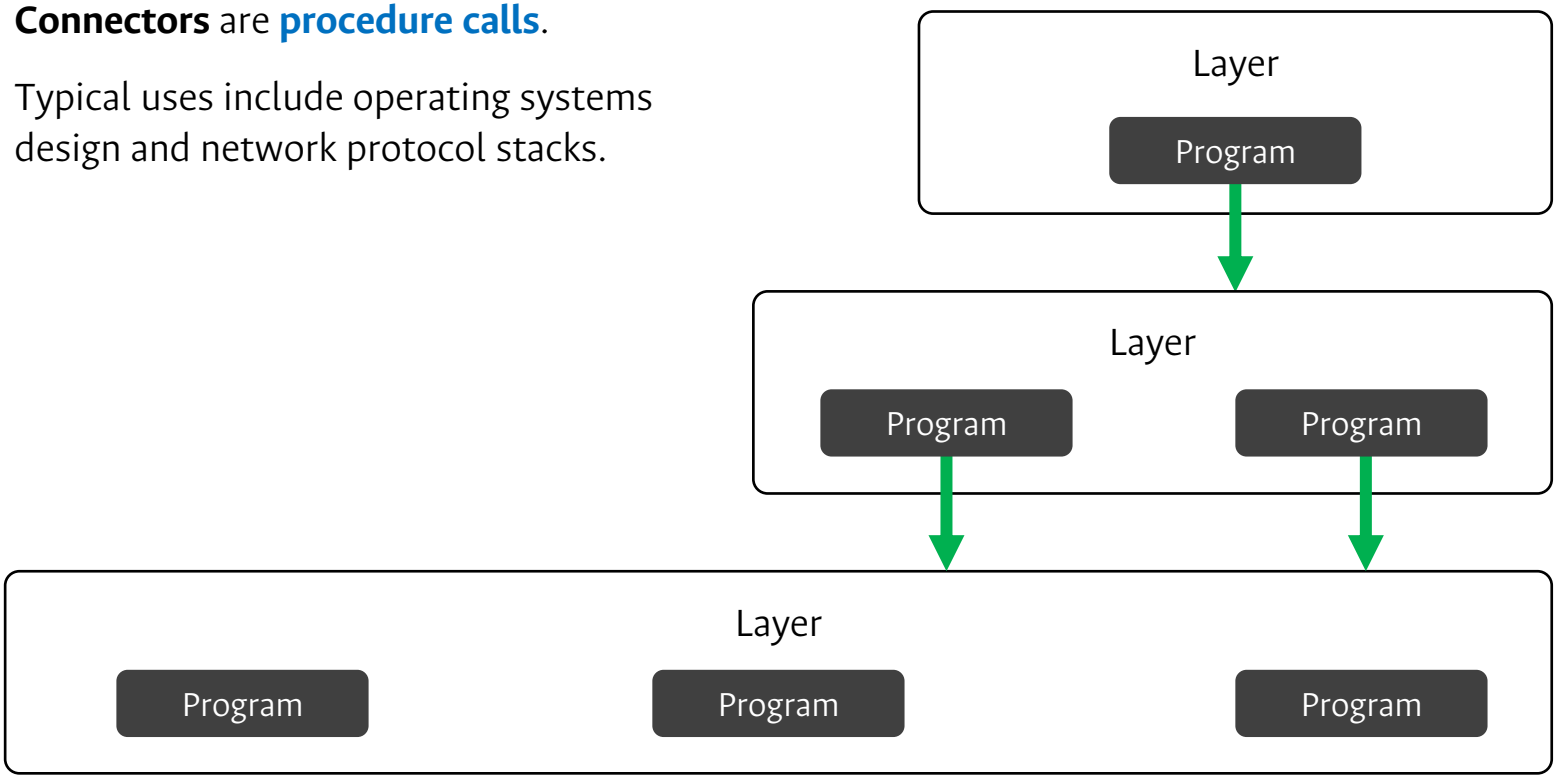
- **Components** are **clients** and **servers**.
- Servers do not know number or identities of clients.
- Clients know server's identity.
- **Connectors** are RPC-based network interaction **protocols**.



* **RPC**: *Remote Procedure Call*

Virtual Machines Style
(Virtual Machines Architecture)

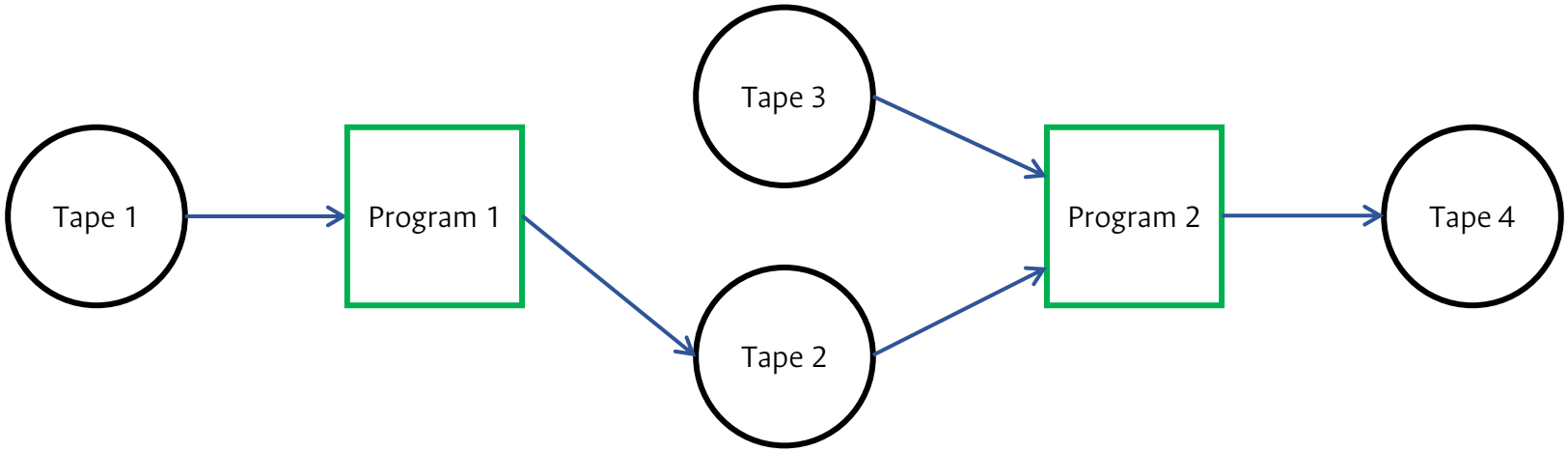
- **Components** are **layers**.
- **Connectors** are **procedure calls**.
- Typical uses include operating systems design and network protocol stacks.



Batch-Sequential Style

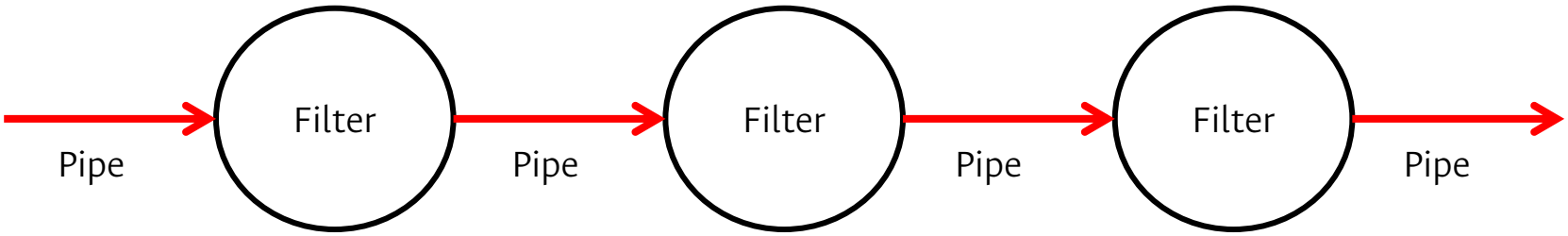
(Batch-Sequential Architecture)

- **Components** are **independent programs**.
- **Connectors** are **tapes**.
- Aggregated data (on magnetic tape) transferred by the user from one program to another.



Pipe-and-Filter Style
(Pipe-and-Filter Architecture)

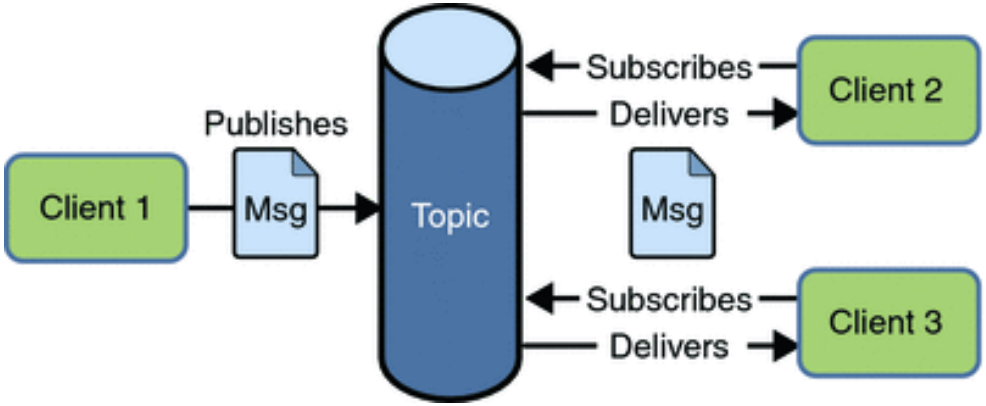
- **Components** are **filters**.
- **Connectors** are **pipes**.
- Style invariants:
  - Filters are independent (no shared state).
  - Filter has no knowledge of up- or down-stream filters.



Publish-Subscribe (Pub/Sub) Style

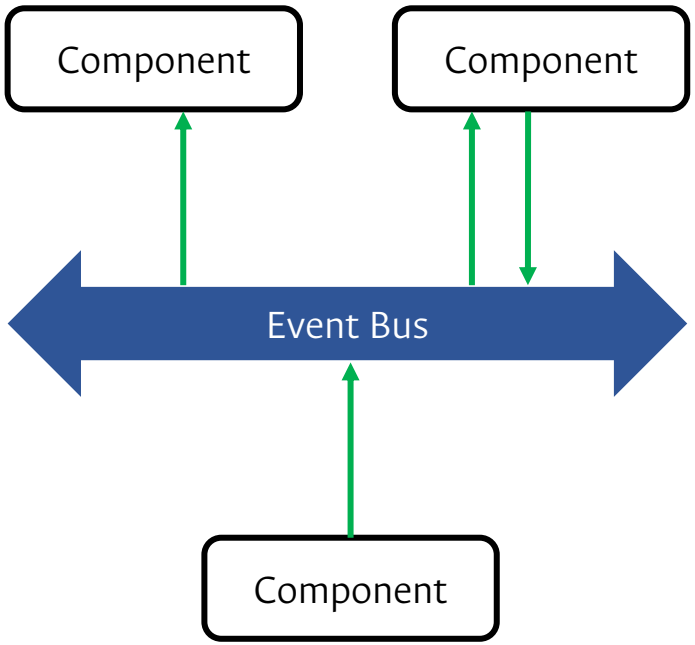
(Publish-Subscribe (Pub/Sub) Architecture)

- **Components**: **publishers**, **subscribers**, proxies for managing distribution.
- **Connectors**: typically a **network protocol** is required. Content-based subscription requires sophisticated connectors.
- Data elements are subscriptions, notifications, published information.



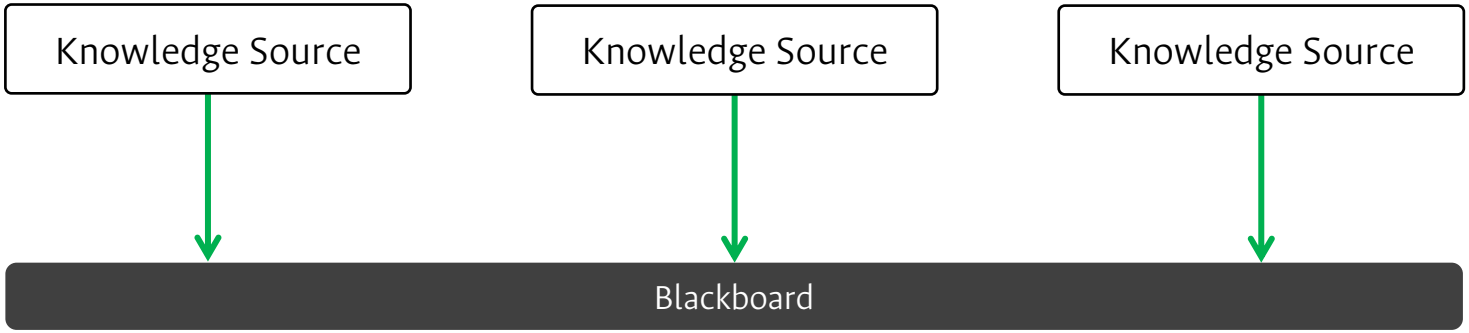
Event-Based Style
(Event-Based Architecture)

- **Components**: independent, concurrent **event generators** and/or **consumers**.
- **Connectors**: **event buses** (at least one).
- Data elements are events–data sent as a first-class entity over the event bus.



Blackboard Style
(Blackboard Architecture)

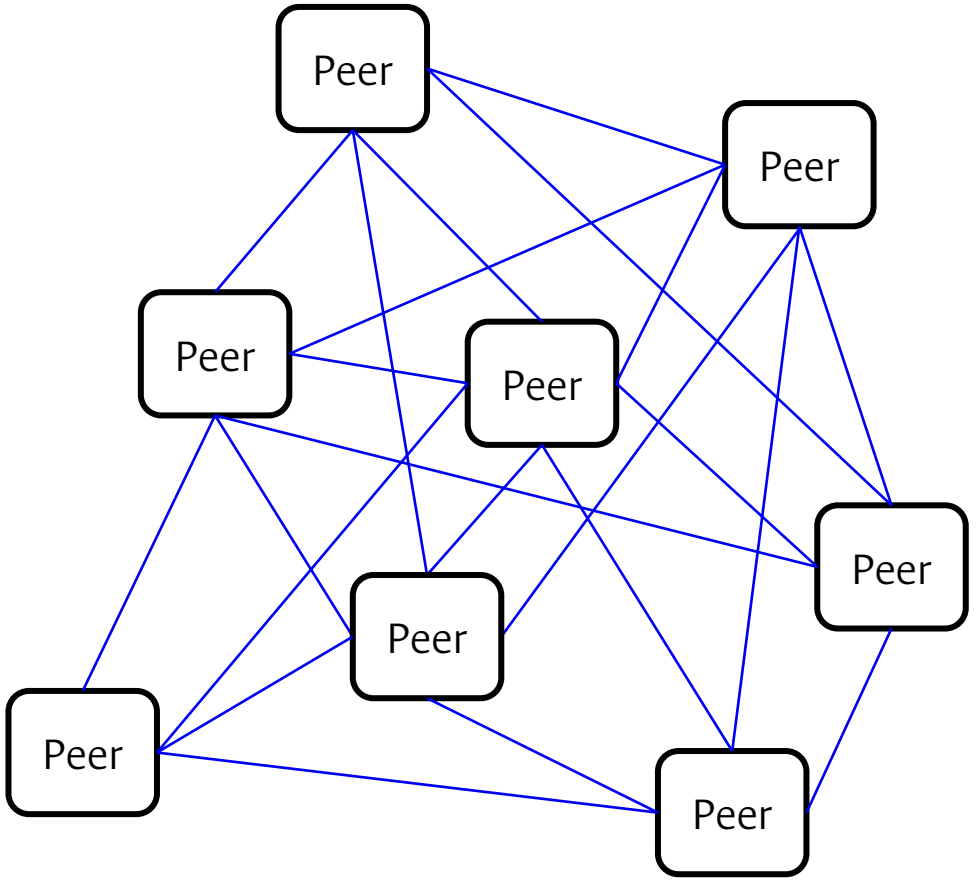
- **Components** are **knowledge sources** are **blackboard**.
- **Connectors** are **memory references**, **procedure calls** and **database queries**.
- Data elements stored in the blackboard.



Peer-to-Peer (P2P) Style

(Peer-to-Peer (P2P) Architecture)

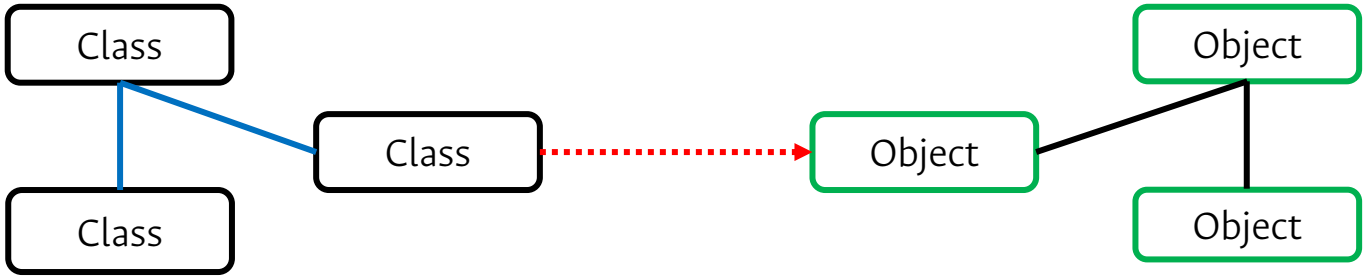
- State and behavior are distributed among **peers** which can act as either **clients** or **servers**.
- **Components** are **peers**, which has their own state and control thread.
- **Connectors** are **network protocols**, often custom.
- Data elements are network messages.



Object-Oriented Style

(Object-Oriented Architecture - OOA)

- **Components** are **classes** and **objects**.
- **Connectors** are **relations**.
- OOA is used to describe a system as a collection of **classes** (entities to be abstracted and the encapsulation of functionalities) that can have **objects** (instances) and communicate between themselves by sending messages.



Component-Based Style

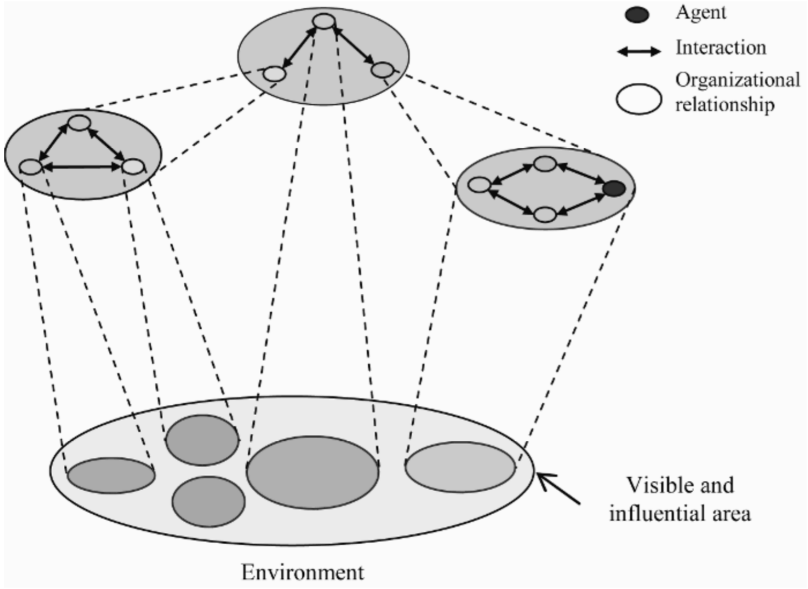
(Component-Based Architecture - CBA)

- **Components** are **components**.
- **Connectors** are **connectors**.
- CBA is used to describe systems as a set of **components** (processing or storage units) that communicate with each other via **connectors** (interaction units).
- A CBA shares common concepts between system users and finally build reusable off-the-shelf component-based heterogeneous systems.

Agent-Oriented Style

(Agent-Oriented Architecture - AOA)

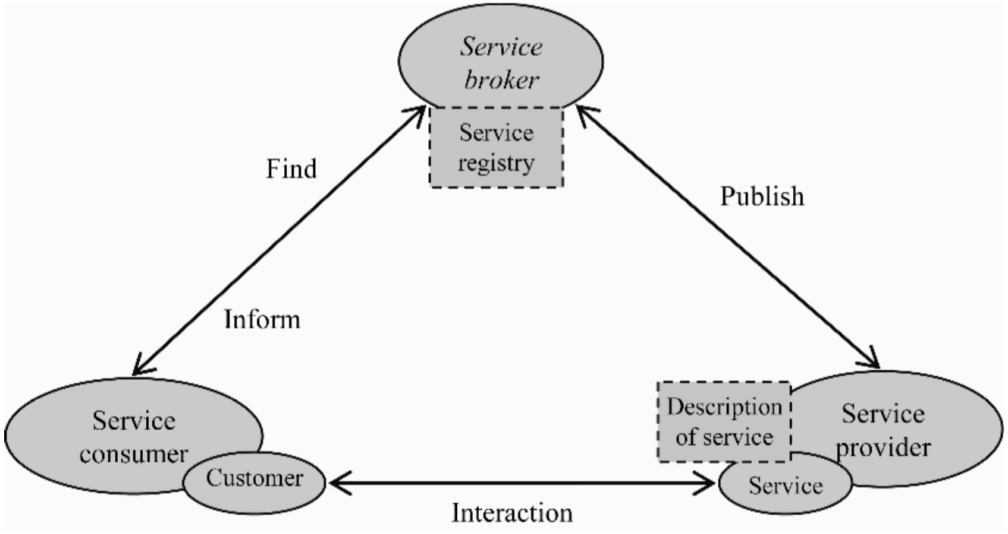
- Multi-Agent System (MAS).
- **Components** are **agents** (autonomous entities able to communicate).
- **Connectors** are **interactions**.
- A MAS is a paradigm for understanding and building **distributed systems**.
- Processing elements (**agents**) have:
  - A partial knowledge of what surrounds.
  - Their own particular **behavior**.
  - Capacity to execute themselves independently.



Service-Oriented Style

(Service-Oriented Architecture - SOA)

- **Components** are **service providers**, **consumers** and **brokers**.
- **Connectors** are **protocols of service interaction**.
- SOA defines a conceptual framework to organize the construction of software systems based on **services**.



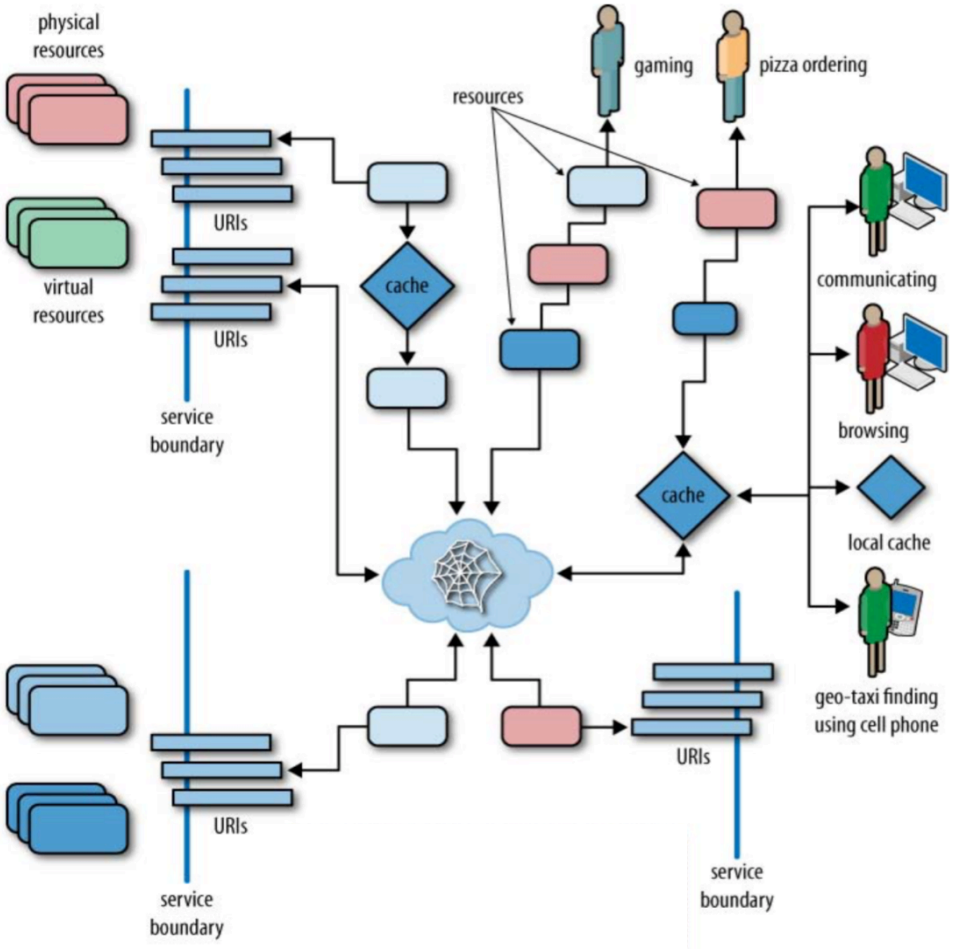
REST Style

(REpresentational State Transfer)

Context

Web Architecture focuses on the foundation technologies and principles which sustain the **Web**, including **URI**s and **HTTP**.

World Wide Web (WWW)



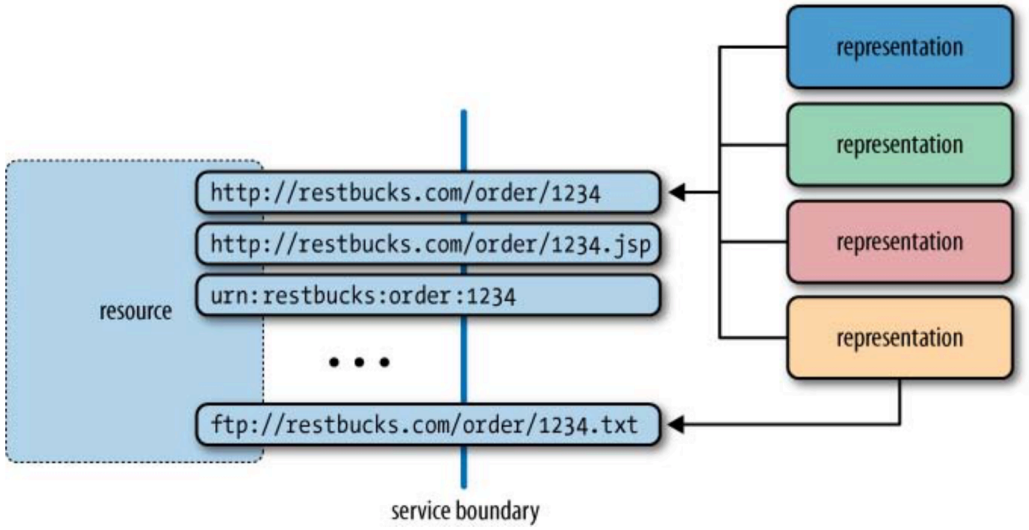
REST Style
(REpresentational State Transfer)

Context

To **use** a web **resource** we need:

- To be able to **identify** it on the network.
- To have some means of **manipulating** it.

Uniform **R**esource **I**dentifier (**URI**)



REST Style
(REpresentational State Transfer)

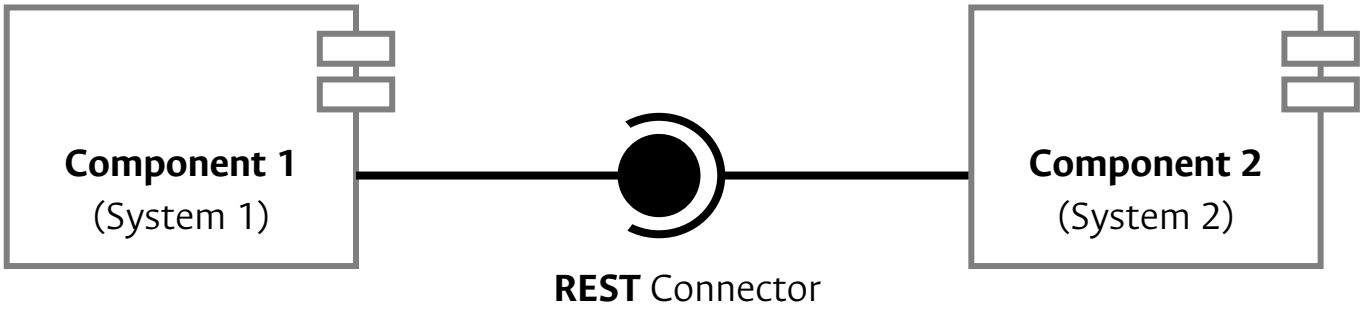
Context

The **HyperText Transfer Protocol (HTTP)** is an application-level **protocol** for distributed, collaborative, hypermedia information systems.

REST Style

(REpresentational State Transfer)

REST is a **client-server-based architectural style** that is structured around a small set of **create**, **read**, **update**, **delete** (CRUD) operations (called **POST**, **GET**, **PUT**, **DELETE** respectively in the REST world) and a single **addressing scheme** (based on a **URI**, or uniform resource identifier).



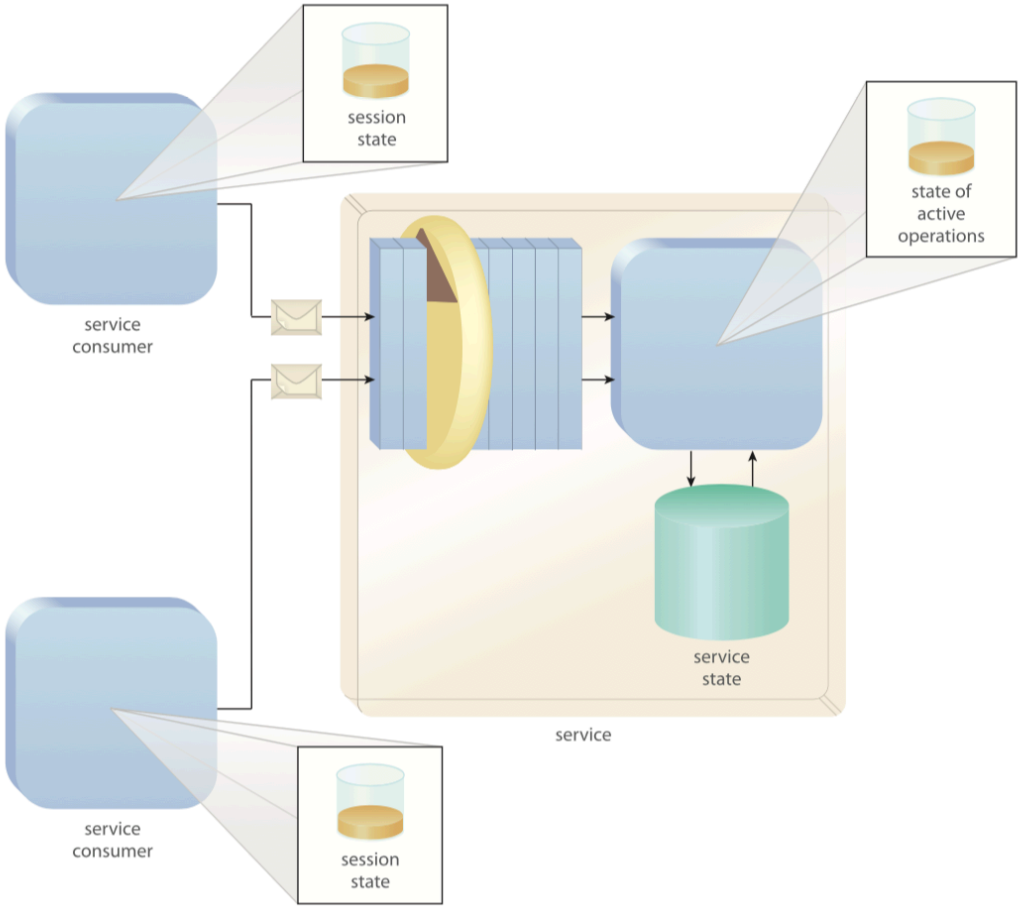
REST Style

(REpresentational State Transfer)

Constraints	Goals
• Client-Server • Stateless • Cache • Interface / Uniform Contract • Layered System • Code-On-Demand	• Performance • Scalability • Simplicity • Modifiability • Visibility • Portability • Reliability

REST Style

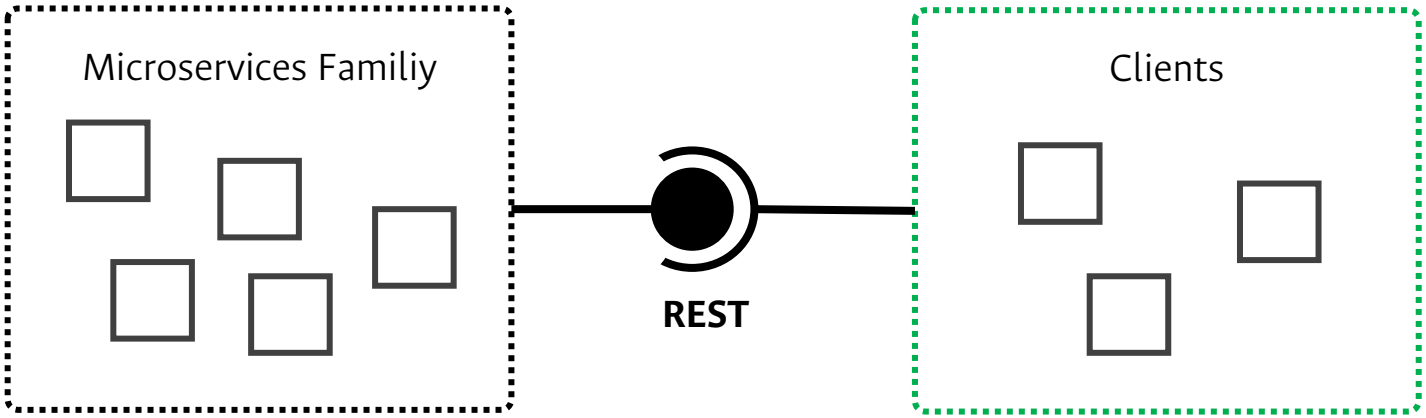
(REpresentational State Transfer)



Microservices Style

(Microservices Architecture - MSA)

- **Components** are **microservices** and **clients**.
- **Connector** is **HTTP protocol (REST connectors)**.
- MSA structures a software system as a collection of loosely coupled **services**, which implement business capabilities.



Object - Component - Agent - Service

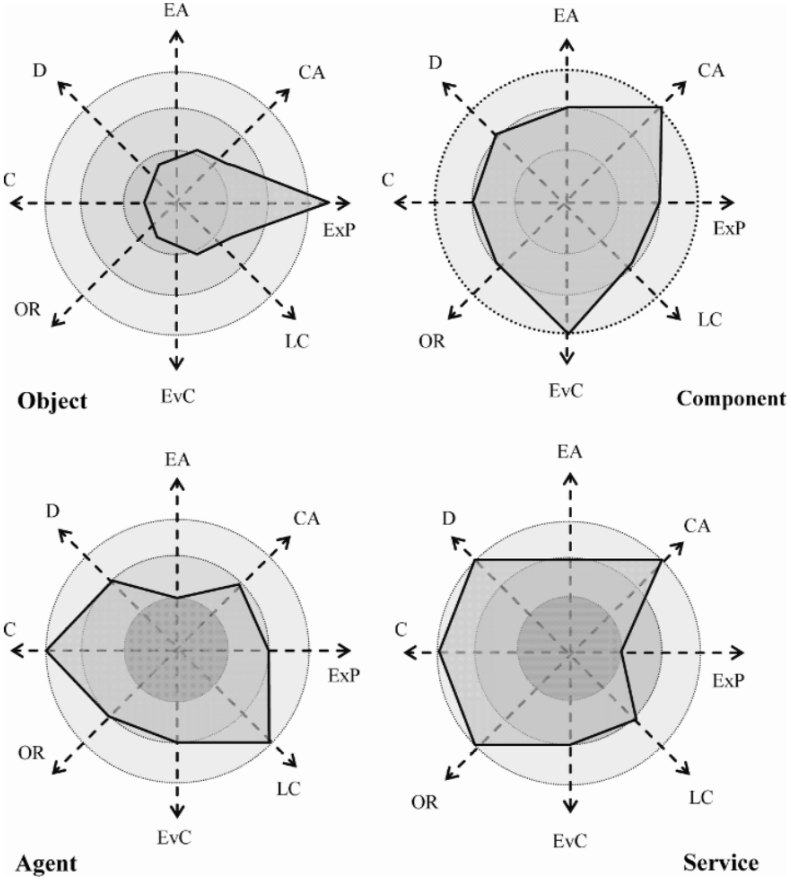


Figure 1.8. *Comparison of criteria with respect to the four paradigms (EA: explicit architecture; CA: communication abstraction; ExP: expressive power; LC: loose coupling; EvC: evolution capacity; OR: owner's responsibility; C: concurrency; D: distribution)*

References

- [**CLEMENTS**] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford, Documenting Software Architectures: Views and Beyond. 2011.
- [**ERL**] T. Erl, SOA with REST, 1st ed. 2013.
- [**GOMAA**] H. Gomaa, Software Modeling & Design: UML, Use Cases, Patterns & Software Architectures. 2011.
- [**OUSSALAH**] M. C. Oussalah, Software Architecture 1. 2014.
- [**QIN**] Z. Qin, J. Xing, and X. Zheng, Software Architecture. 2008.
- [**RICHARDSON-1**] C. Richardson and F. Smith, Microservices: From Design to Deployment. 2016.
- [**RICHARDSON-2**] C. Richardson, Microservices Patterns. 2019.
- [**TAYLOR**] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, Software Architecture - Foundations, Theory, and Practice. 2010.
- [**WEBBER**] J. Webber, S. Parastatidis, and I. Robinson, REST in Practice. 2010.