

Service Oriented Computing: Introduction

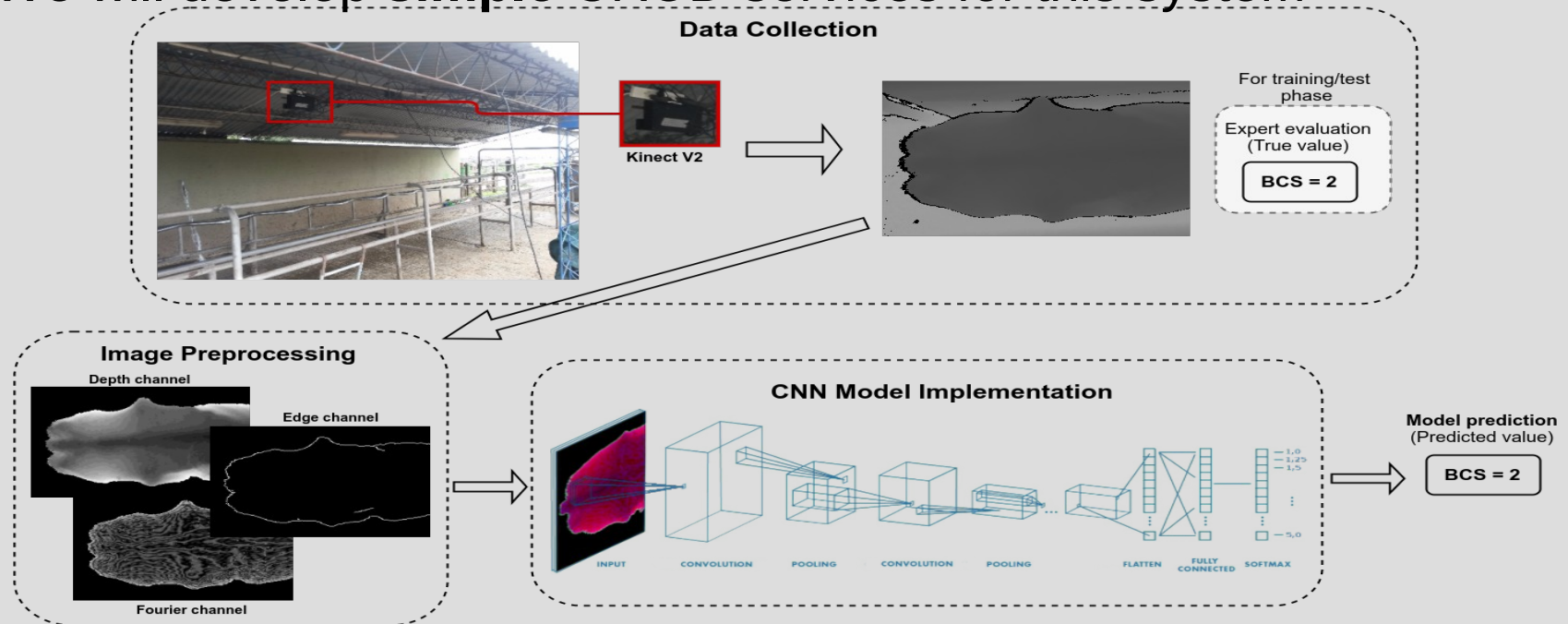
(<http://users.exa.unicen.edu.ar/~cmateos/cos>)
ISISTAN-UNICEN-CONICET

Course overview

- Goals of the course
 - To understand the philosophy behind SOC
 - To discuss the role of XML/JSON in SOC
 - To get a glimpse of the existing SOC technologies
 - Learn to code SOAP and Rest services :)
- Prerequisites
 - Object-oriented and Java programming
 - Web **back-end/front-end** (platforms, frameworks, libraries... all you can remember will help)

This year's application domain: Body Condition Score for cows

- Joint project Uniagro – ISISTAN
- We will develop **simple** CRUD services for this system



Useful resources

- Some interesting links
 - W3C Web site: <http://www.w3.org>
 - *“The World Wide Web Consortium (W3C) is an international community that develops open standards to ensure the long-term growth of the Web”*
 - *Led by Tim-Berners Lee*
 - *Some members: AliBaba, Facebook, MS...*
 - W3C schools: <http://www.w3schools.com>
 - SOC@Wikipedia.org: <http://tinyurl.com/wmzn9>

SOC: Overview

- **Publish/Unpublish**

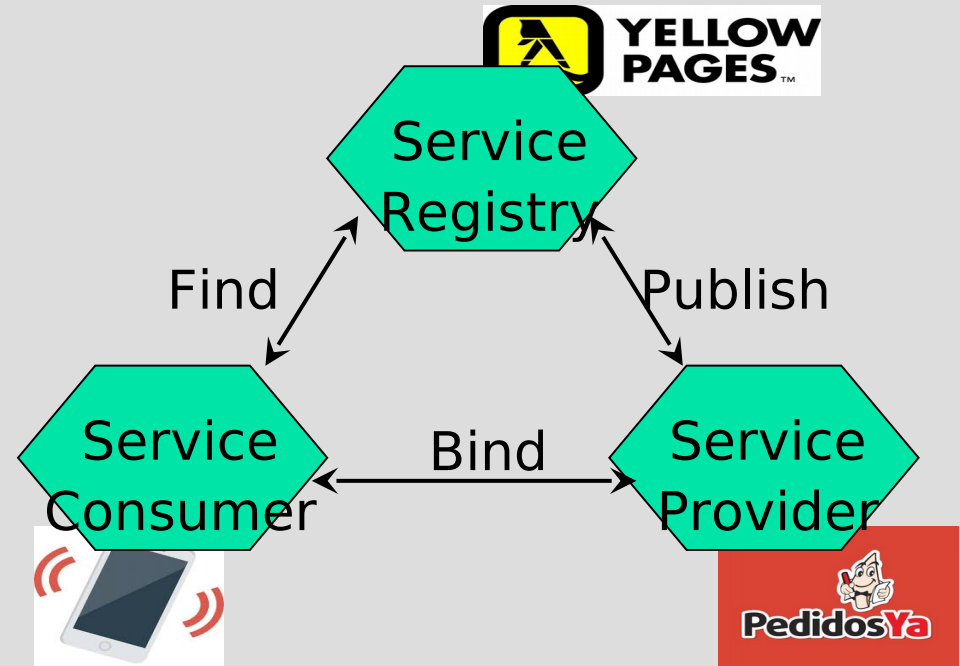
Advertising **services** to a **registry** (publishing) or removing those entries (unpublishing)

- **Find**

The service consumers describe the kinds of services they're looking for, and the service brokers deliver the results that best match the request

- **Bind**

The bind operation takes place between the service consumer and the service provider



Physical services

- Clearly defined, easy-to-use (ideally), somewhat ***standardized interface***
 - *E.g. request protocol for food delivery*
- *Self-contained* with no visible dependencies to computational resources or other services
- ***Always available*** (well, almost); idle until requested
- Services are ***coarse-grained***
- New services can be offered by ***combining existing services*** (e.g., airline + hotel bookings, geolocation + POIs)
- Each service has an associated ***quality of service*** (do not compete on what they do but how they do it...)

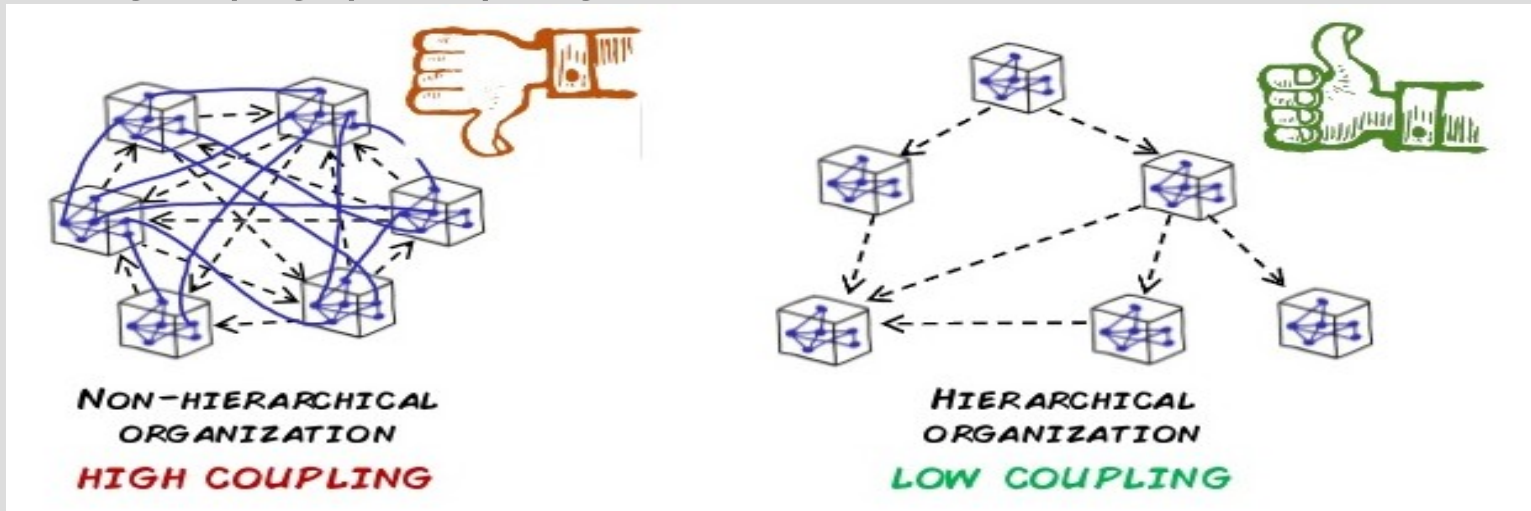
Physical services (cont.)

- Services belonging to the same category provide similar interfaces (*non-proprietary*)
 - Proprietary: A vendor's graphic card API
- Service implementation is decoupled from service interface (*abstraction*)
- Services are mediated through “yellow pages” (*discovery*)...
 - Yellow pages are responsible for finding the best alternative given a certain request (more on this later)
 - These ideas are already present in several distributed technologies: RMI/DCOM, JNDI, CORBA (**broker architectural pattern**)

Physical services (cont.)

Coupling

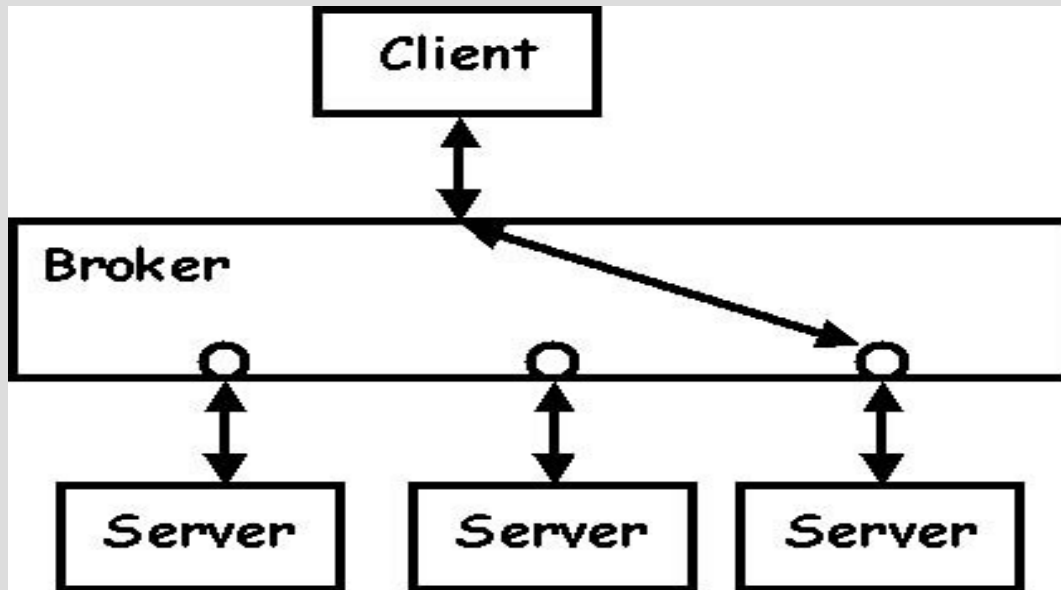
- A **decoupled** system allows changes to be made without having side effects
 - Coupling... “a necessary evil”
 - Loose (low) coupling
 - Tight (high) coupling



Physical services (cont.)

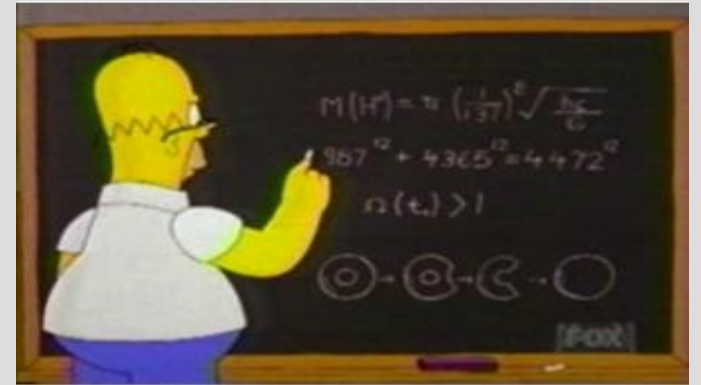
Brokers

- Broker is an **architectural pattern**
 - A pattern is a recurrent good solution to a known problem (e.g. those of Gamma)



Why SOC? A technical perspective

- It helps to fulfill several desirable **quality attributes**
- Development (reusability) and **deployment**
- **Scalability** due to distribution
- **Interoperability**
- Loose coupling
- Composability
- Pervasiveness:
 - Amazon WS: aws.amazon.com
 - Google APIs explorer:
<http://developers.google.com/apis-explorer>
 - Many, many others!



Google APIs explorer: Example

The screenshot displays the Google APIs Explorer interface. At the top, there is a search bar with the text "Search for services, methods, and recent requests..." and a "Loading..." button. Below the search bar, the "APIs Explorer" header is visible, along with a back arrow and a settings gear icon. The main content area shows the breadcrumb "Services > Google+ API v1 > plus.comments.list" and an "Authorize requests using OAuth 2.0" toggle set to "OFF". The interface lists several parameters for the API endpoint:

- activityId**: The ID of the activity to get comments for. (string)
- maxResults**: The maximum number of comments to include in the response, which is used for paging. For any response, the actual number returned might be less than the specified maxResults. (integer, 0-500)
- pageToken**: The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of "nextPageToken" from the previous response. (string)
- sortOrder**: The order in which to sort the list of comments. (string)
- fields**: Selector specifying which fields to include in a partial response. [Use fields editor](#)

At the bottom, there is a legend indicating "bold red = required" and an "Execute" button.

This is just one of the many API sources (more on this later)
<https://nordicapis.com/api-discovery-15-ways-to-find-apis/>

Object “vs” service orientation paradigms

OO World

Classes and objects

Class behavior/interfaces

Attributes

Public and **private** methods

Messages (invocations)

Encapsulation, inheritance, polymorphism, composition

SOC World

Services and service instances

Service contract

No attributes (**stateless** preferred)

Only **public** operations

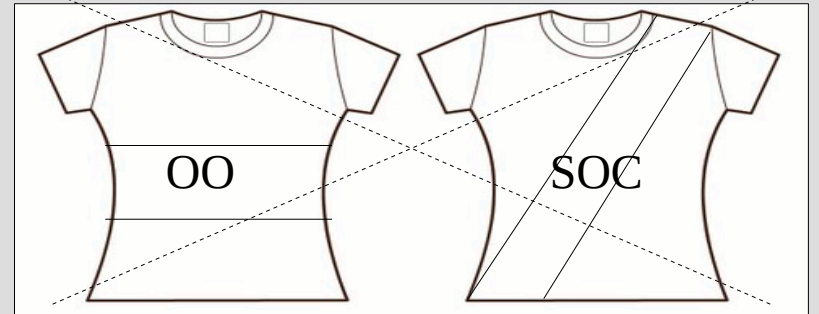
Message passing (XML, JSON)

Encapsulation, composition

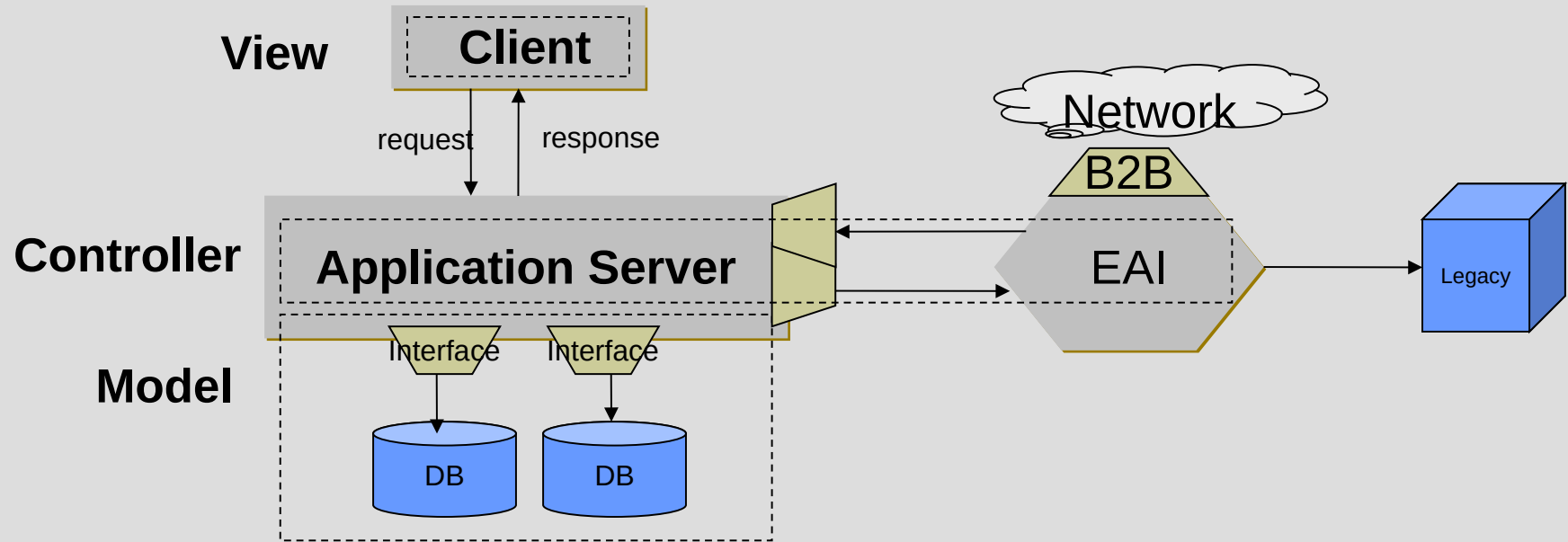
Object “vs” service orientation

SOC shares many similarities with OO, since both:

- Grew out of a need to shape software into units “mirroring” the real world
- Aim to tackle the problem of ever-changing business requirements (and quality attributes)
- Are concerned with minimizing the impact of change upon software programs already deployed and in use (e.g. *microservices*, more on these later)
- Both encompass principles such as Service Loose Coupling and Service Composability
- But... they complement each other!

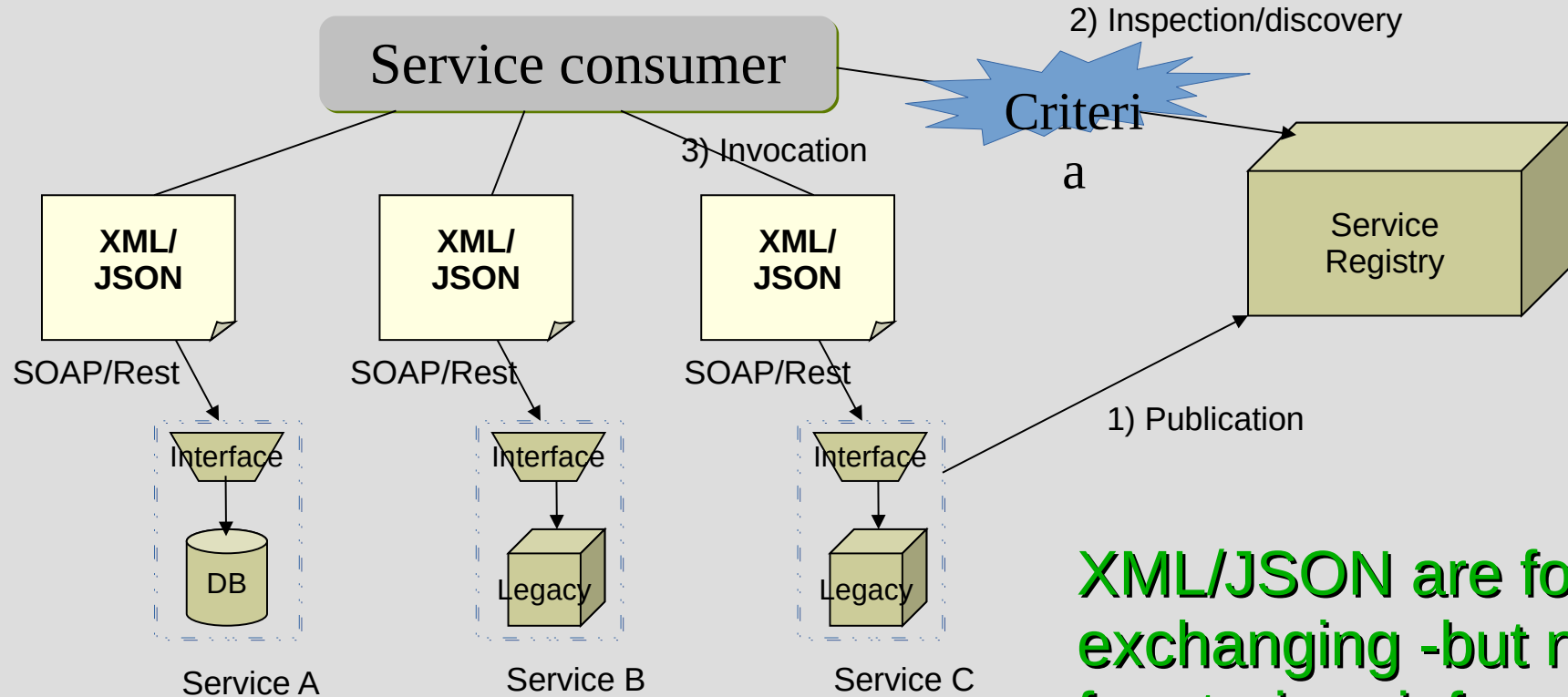


Constructing enterprise software: The way it has been done



- Focus on scalability, 24x7 availability
- Less focus on integration, though some solutions exist (e.g. bus)

Constructing distributed software: The SOC way



XML/JSON are for exchanging -but not likely for storing- information...

SOC: Common Misconceptions

Much of the confusion surrounding the meaning of SOA is caused by how this term has been used in the literature...

- “SOC = SOA”
- “An application that uses Web Services is service-oriented”
 - The sledge hammer anti-pattern...
- “If you understand Web Services you won't have a problem building SOC”
 - SOC requires a change in how business and application logic are viewed, partitioned, automated, deployed...
- “Once you go to SOC, everything becomes interoperable”
 - No! Just like using OO does not mean we ensure reusability...