Information systems modeling

Tomasz Kubik

The Open Group SOA Source Book

http://www.opengroup.org/soa/source-book/intro/index.htm

- Collection of source material for use by Enterprise Architects working with Service-Oriented Architecture.

THE Open GROUP

Subject Areas | Get Involved | Standards | Certifications | Events | Consortia | Software | Publications | About Us

The SOA Source Book

- SOA Source Book
- Service-Oriented Architecture
- The Open Group Service Integration Maturity Model (OSIMM) Version 2
- SOA Reference Architecture
- Service-Oriented Cloud Computing Infrastructure (SOCCI) Framework
- Using TOGAF to Define and Govern Service-Oriented Architectures
- SOA Governance Framework
- Service-Oriented Architecture Ontology Version 2.0

SOA Source Book

The Open Group SOA Source Book is a collection of source material for use by Enterprise Architects working with Service-Oriented Architecture.

It consists of material that has been considered and in part developed by The Open Group SOA Work Group. The SOA Work Group is engaged in a work program to produce definitions, analyses, recommendations, reference models, and standards to assist business and information technology professionals within and outside of The Open Group to understand and adopt SOA.

The Source Book includes the final output of that work program, which is also published separately as a collection of Open Group Standards and Guides, available from The Open Group online bookstore. It also includes interim material, reflecting the current state of work that has not yet resulted in formal Standards and Guides. The content of this material will not necessarily be reflected in the final output.

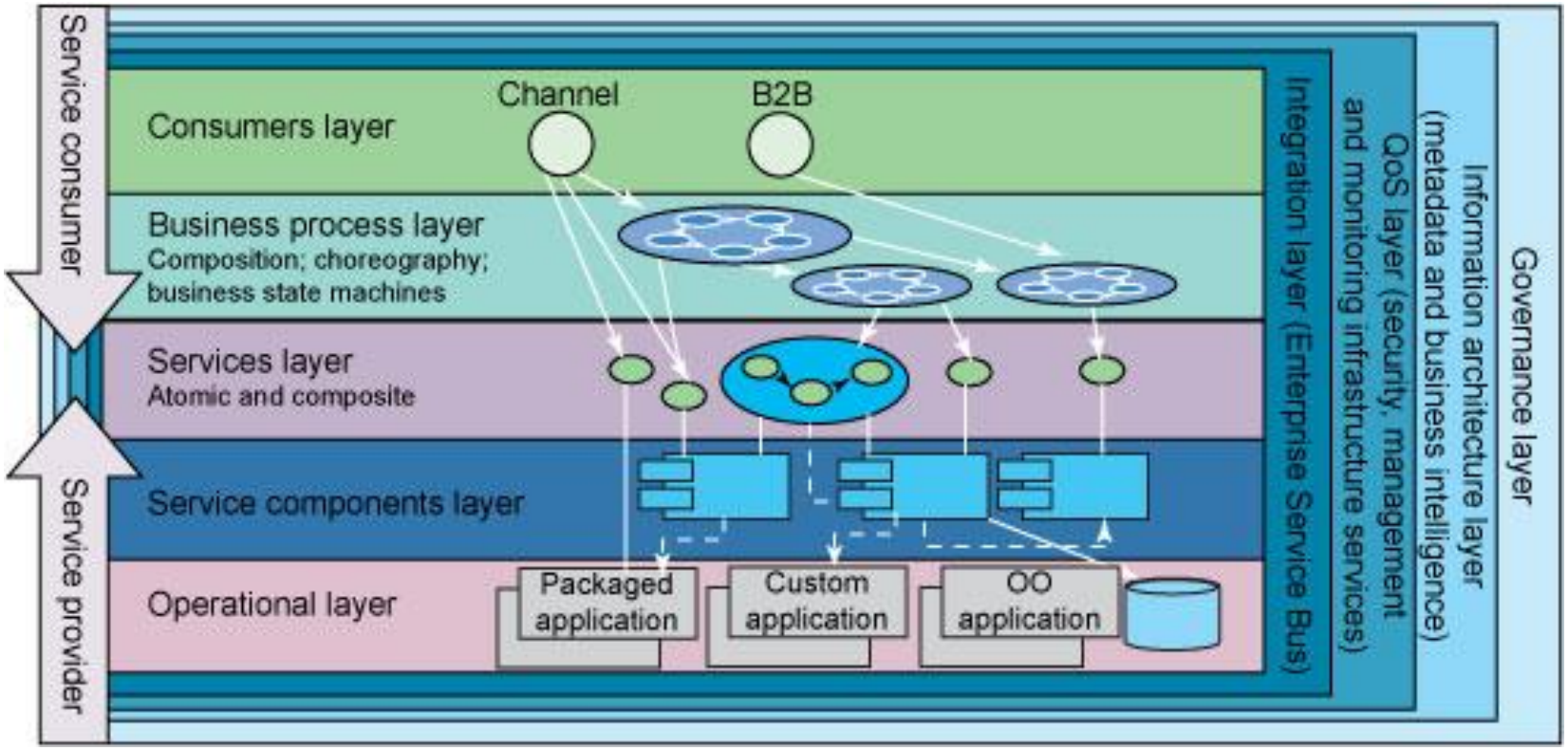
This is Edition 7 of the SOA Source Book. Edition 1, published by Van Haren in April 2009, and also available on the web, contained interim material only. Since it appeared, The Open Group has published five SOA Standards, two SOA Guides, and three SOA White Papers. These have been included in successive editions of the Source Book as they appeared, replacing much of the original interim material.

Two of the Open Group SOA standards – the SOA Reference Architecture and the SOA Ontology – were input to ISO to assist with the development of the International Standard SOA Reference Architecture, ISO/IEC 16384. The Open Group SOA Reference Architecture is now superseded by ISO/IEC 16384:2016, and the reader is encouraged to use the ISO Standard. The Open Group SOA Reference Architecture is included in this edition of the Source Book for reference purposes.

Edition 7 contains:

- A description of Service-Oriented Architecture

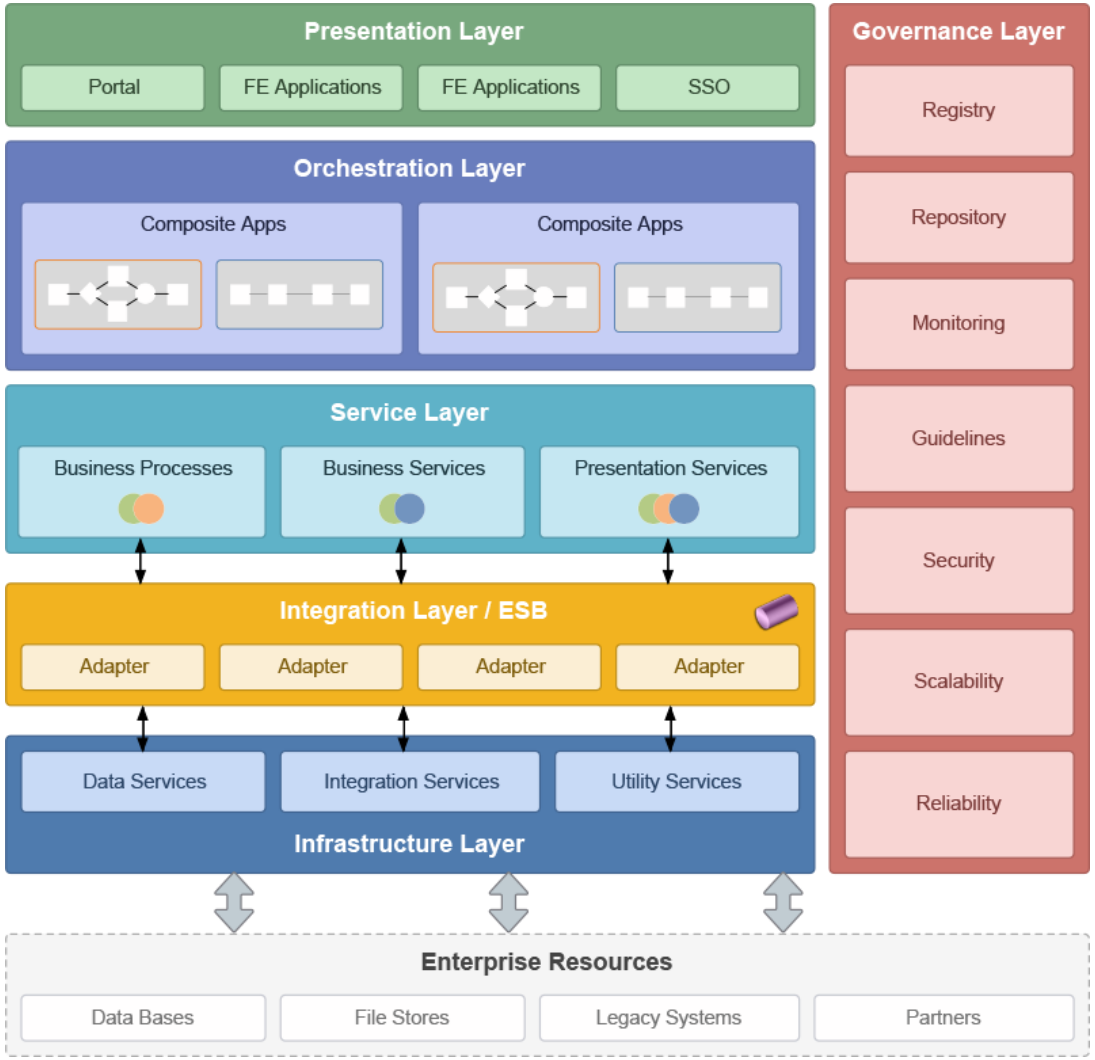
Layers of the SOA reference architecture: Solution stack view



https://www.ibm.com/developerworks/library/ar-archtemp/index.html

Enterprise architecture

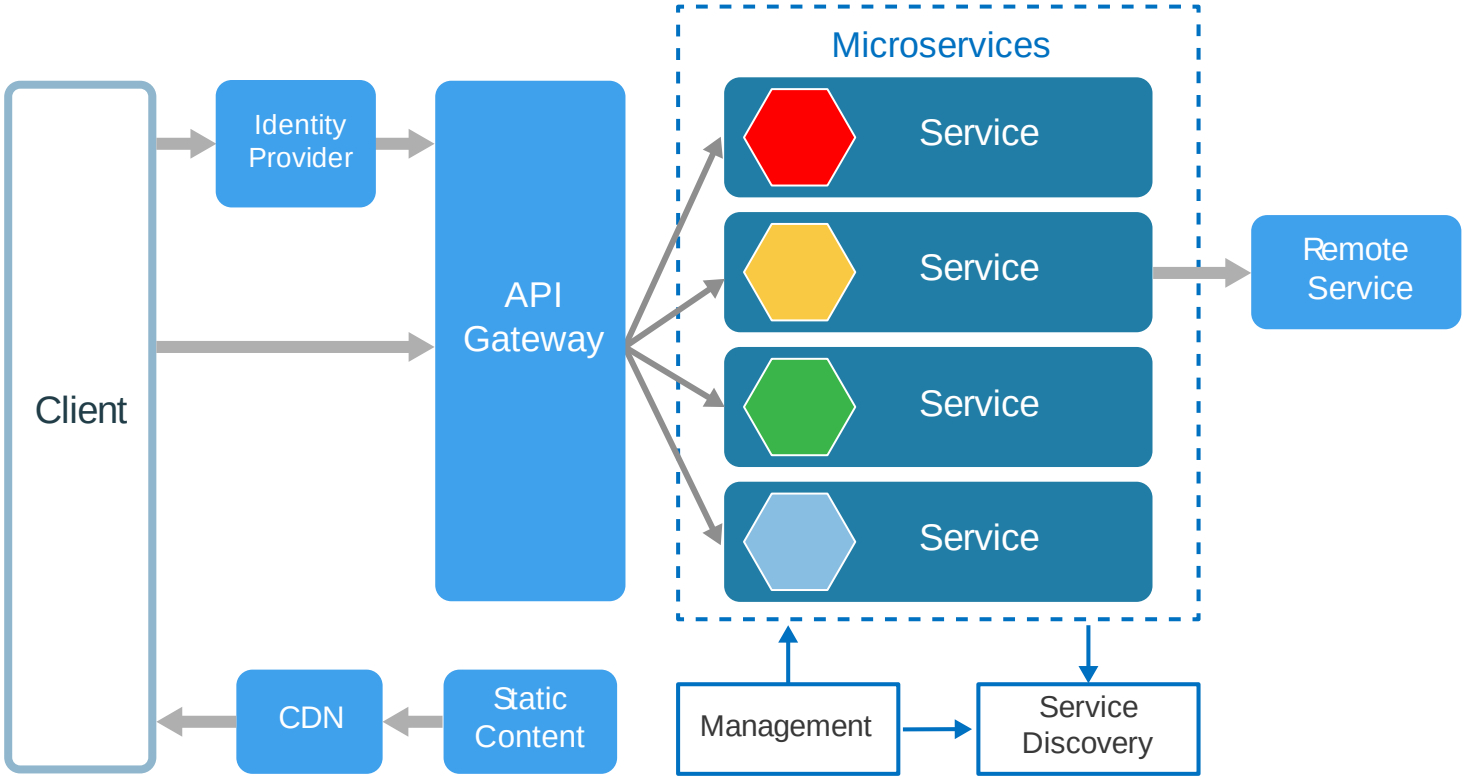
- Service Oriented Architecture encompasses the architectural paradigms for designing applications in a Service centric way with a strong emphasis on services composition (orchestration) and governance.
- Following the SOA paradigm in Enterprise environment is possible only with wide adoption of its principles both at design, development, deployment and integration layers.



https://www.intropro.com/solutions/enterprise-architecture

Microservices architecture

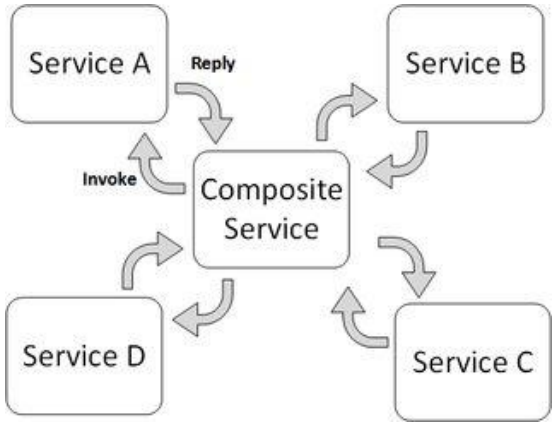
- consists of a collection of small, autonomous services.
- each service is self-contained and should implement a single business capability.



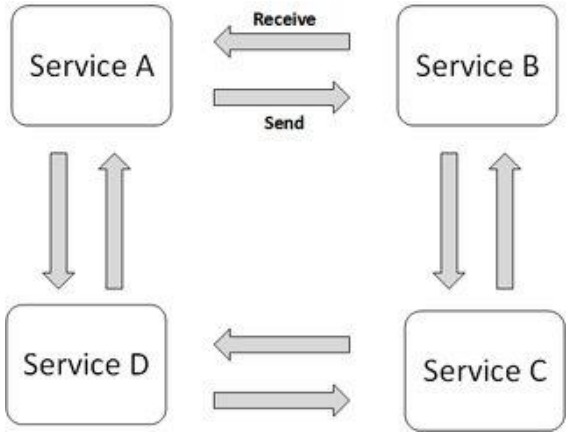
https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

Service compositions

- Orchestration (executable BPEL):
  - there is one particular element used by the composition that oversees and directs the other elements.
  - refers to the coordination of a single participant's process from a local, subjective level
- Choreography
  - the elements used by the composition interact in a non-directed fashion, yet with each autonomous member knowing and following a predefined pattern of behavior for the entire composition.
  - refers to the collaboration of multiple participants from a global view
  - it serves as a contract between parties that clarifies all details of their collaboration
- Collaboration
  - the elements used by the composition interact in a non-directed fashion, each according to their own plans and purposes without a predefined pattern of behavior.



Orchestration = Executable Process



Choreography = Multi-party Collaboration

Orchestration

- All about coordination of web services within a single process from a local, subjective perspective
- Definition of a new service from existing services from controller perspective
- Model including
  - communication actions
    - externally visible message exchanges
    - internally visible message exchanges
  - internal actions
    - processing inside service
    - recording externally and internally visible states
- Can be expressed using an execution language, such as BPEL

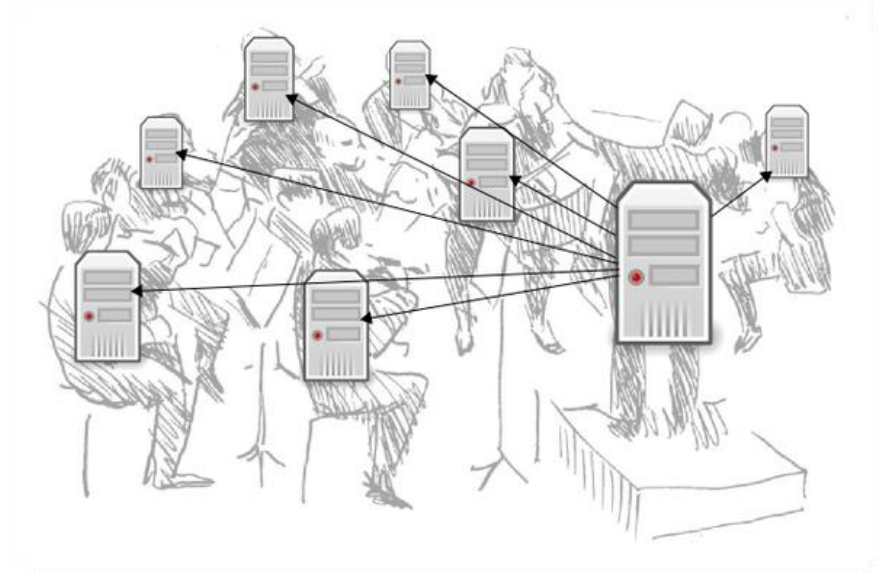
Orchestration = Executable Process

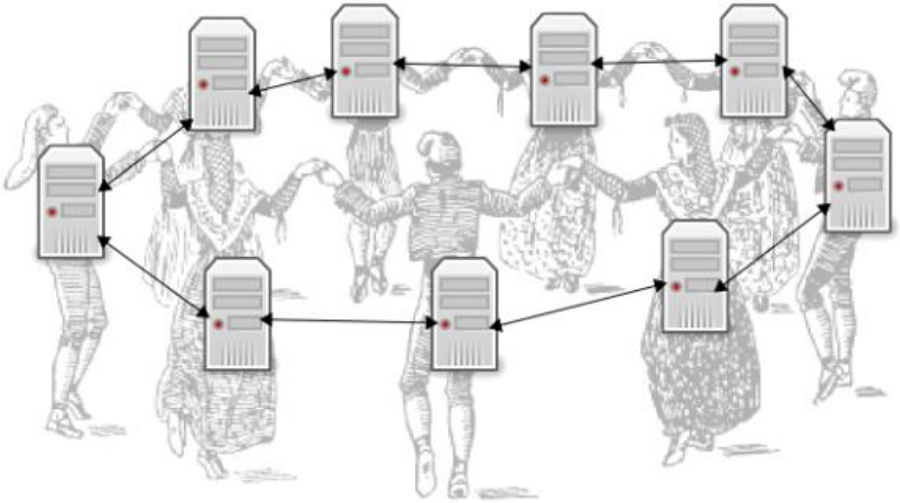
Choreography

- Global coordination of several services
  - it focuses on how to build stateful, conversational, long-running processes out of basic stateless, atomic web services operations
  - it concerns collaboration of two or more participants aimed at achieving a common goal
    - specifies jointly agreed, information driven reactive rules
    - sets control-flow dependencies, data-flow dependencies, transactional dependencies, message correlations, time constraints.
- It does not describe any internal action that occurs within a participants
- It is not an executable process.
  - It serves rather as a contract between parties that clarifies all details of their collaboration

Choreography = Multi-party Collaboration

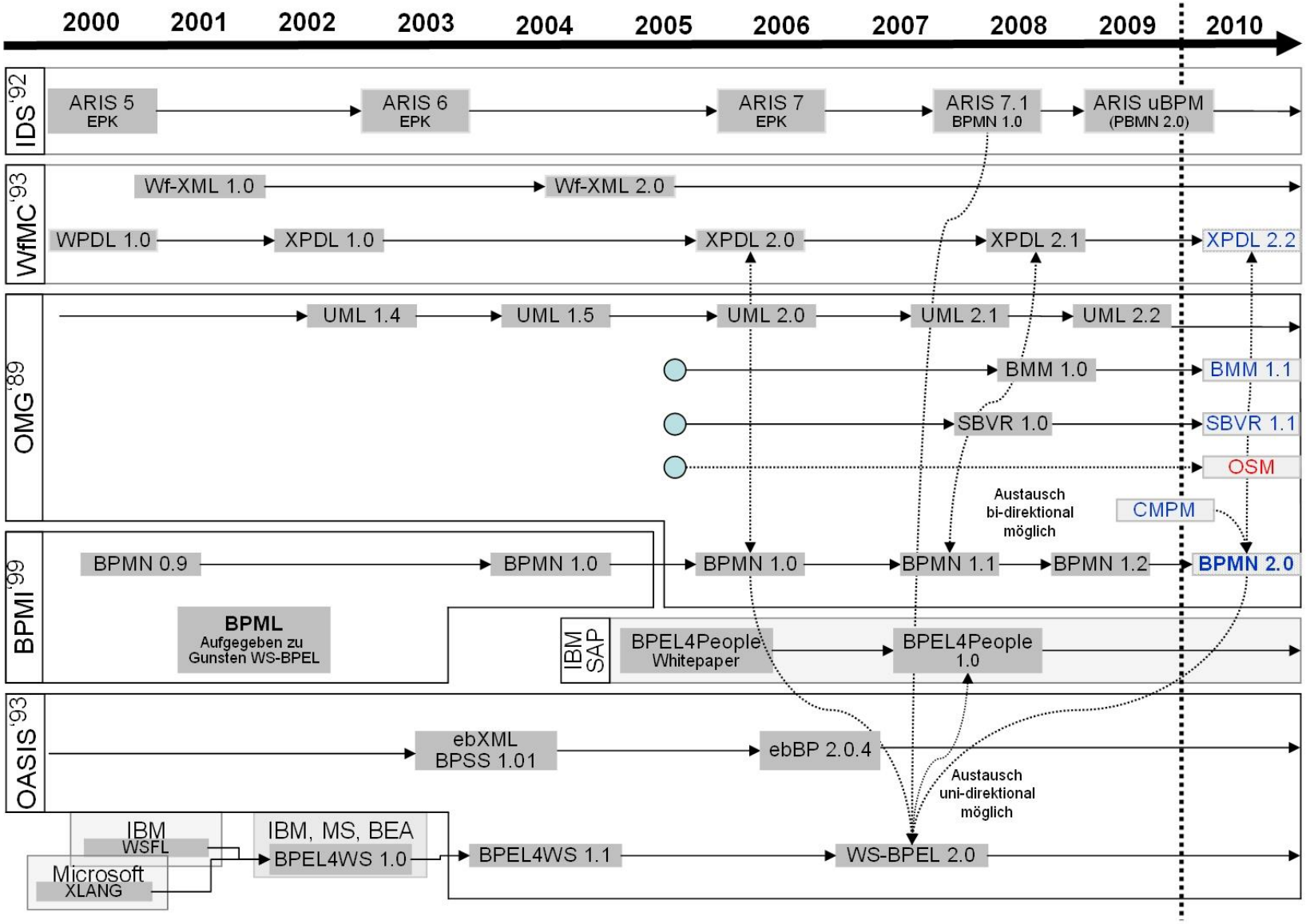
Orchestration vs Choreography



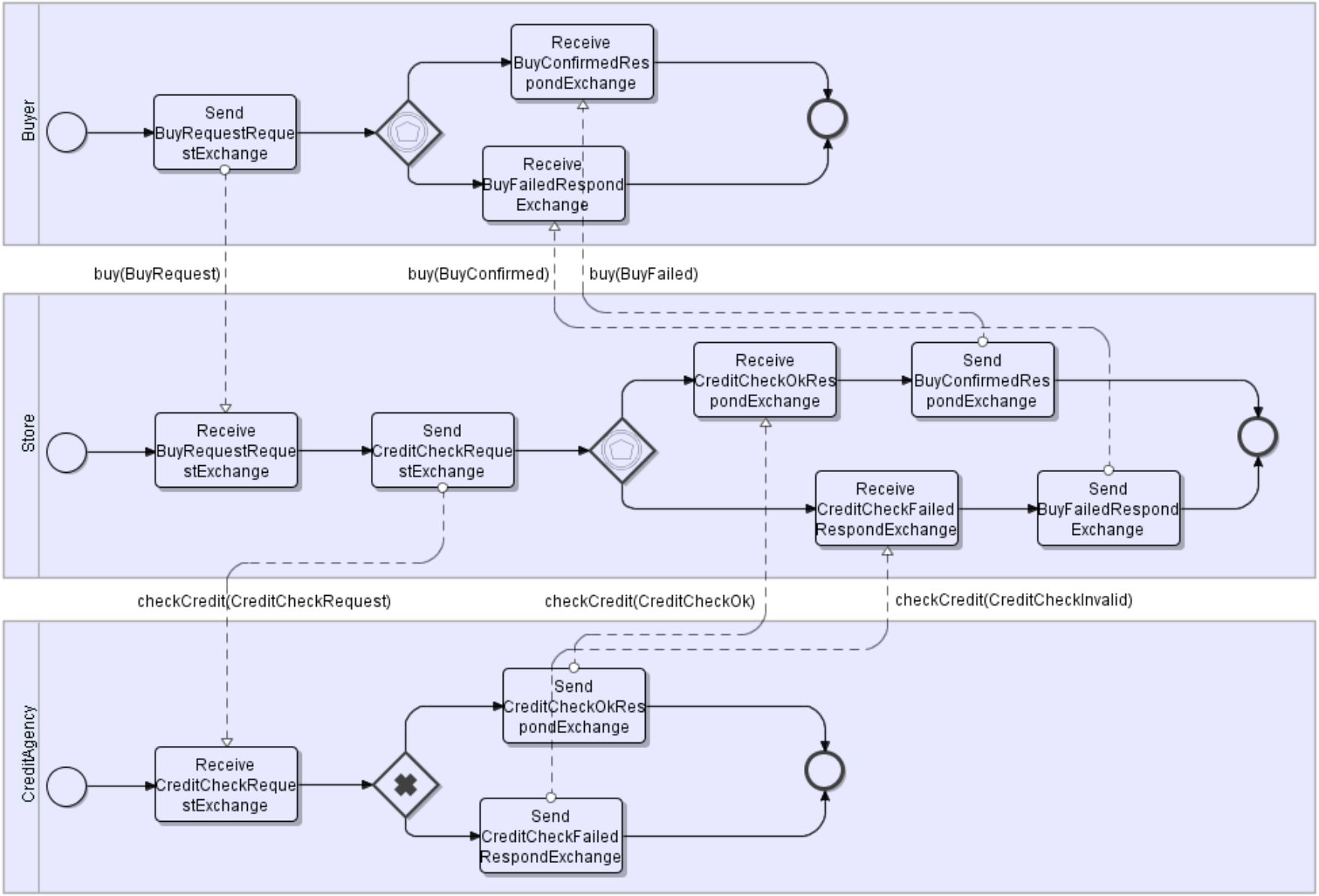


https://jcastellssala.com/tag/choreography/

Timeline of BPM languages



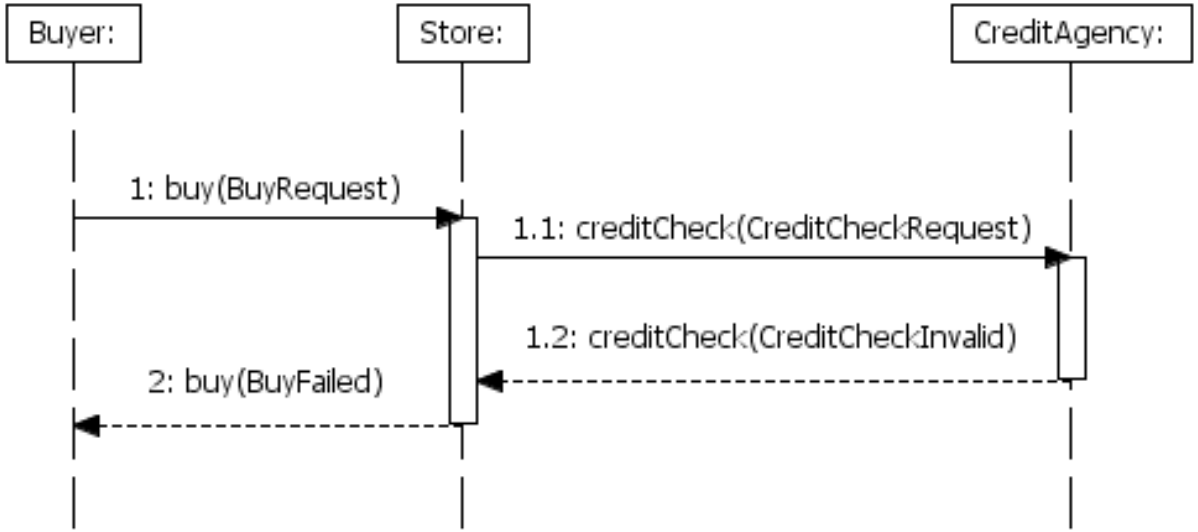
Example of BPM



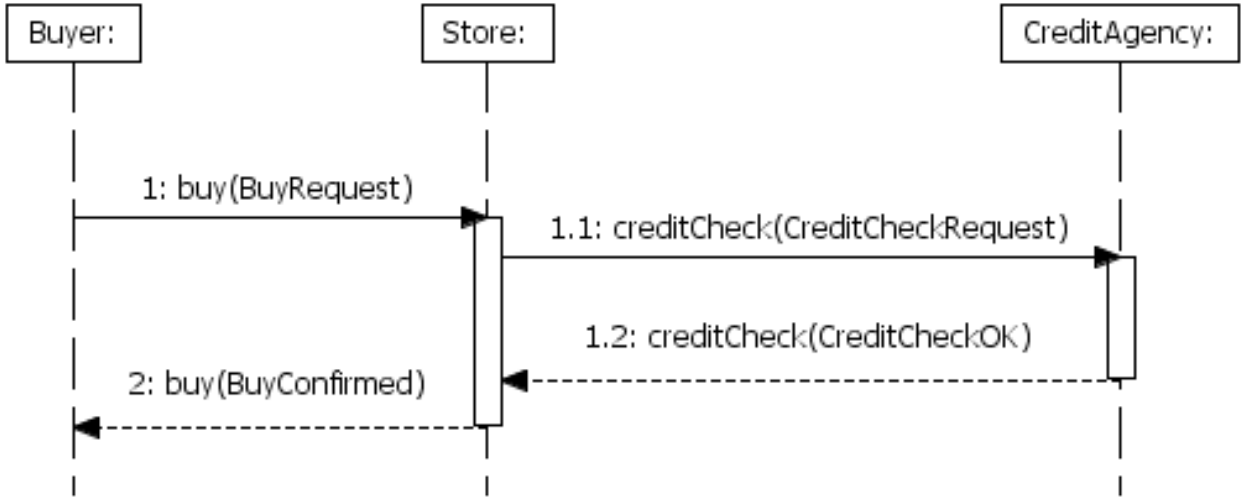
generated based on `http://sourceforge.net/apps/trac/pi4soa/wiki/examples`

UML sequence diagrams

Scenario 1



Scenario 2



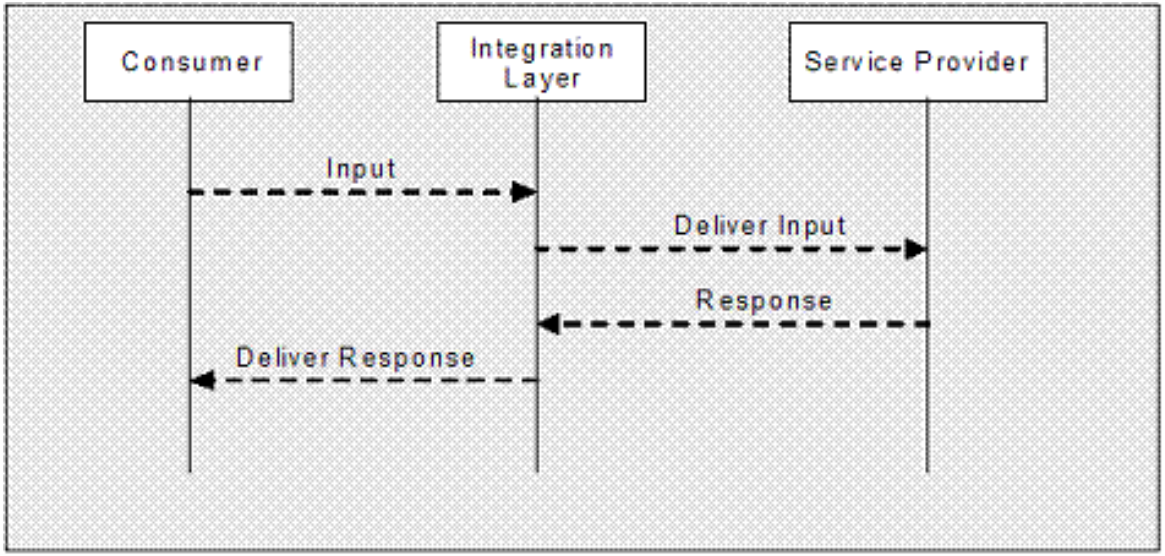
Integration

- enables the loose coupling between the request and the concrete provider by matching the Service Request and Service Implementation.
- this is not only a technical loose coupling addressing protocols, binding, locations or platforms, but can also a business semantic loose coupling performing required adaptations between service requester and provider.

Integration Layer

http://www.opengroup.org/soa/source-book/intro/index.htm

- Provides a level of indirection between the consumer of functionality and its provider
- Consumers and providers are decoupled (it allows integration of disparate systems into new solutions).
- A service consumer interacts with the service provider via the Integration Layer.
- Each service interface is only exposed via the Integration Layer (e.g., ESB), never directly
- point-to-point integration is done at the Integration Layer instead of consumers/requestors doing it themselves.



http://www.opengroup.org/soa/source-book/soa_refarch/p13.htm#figure40

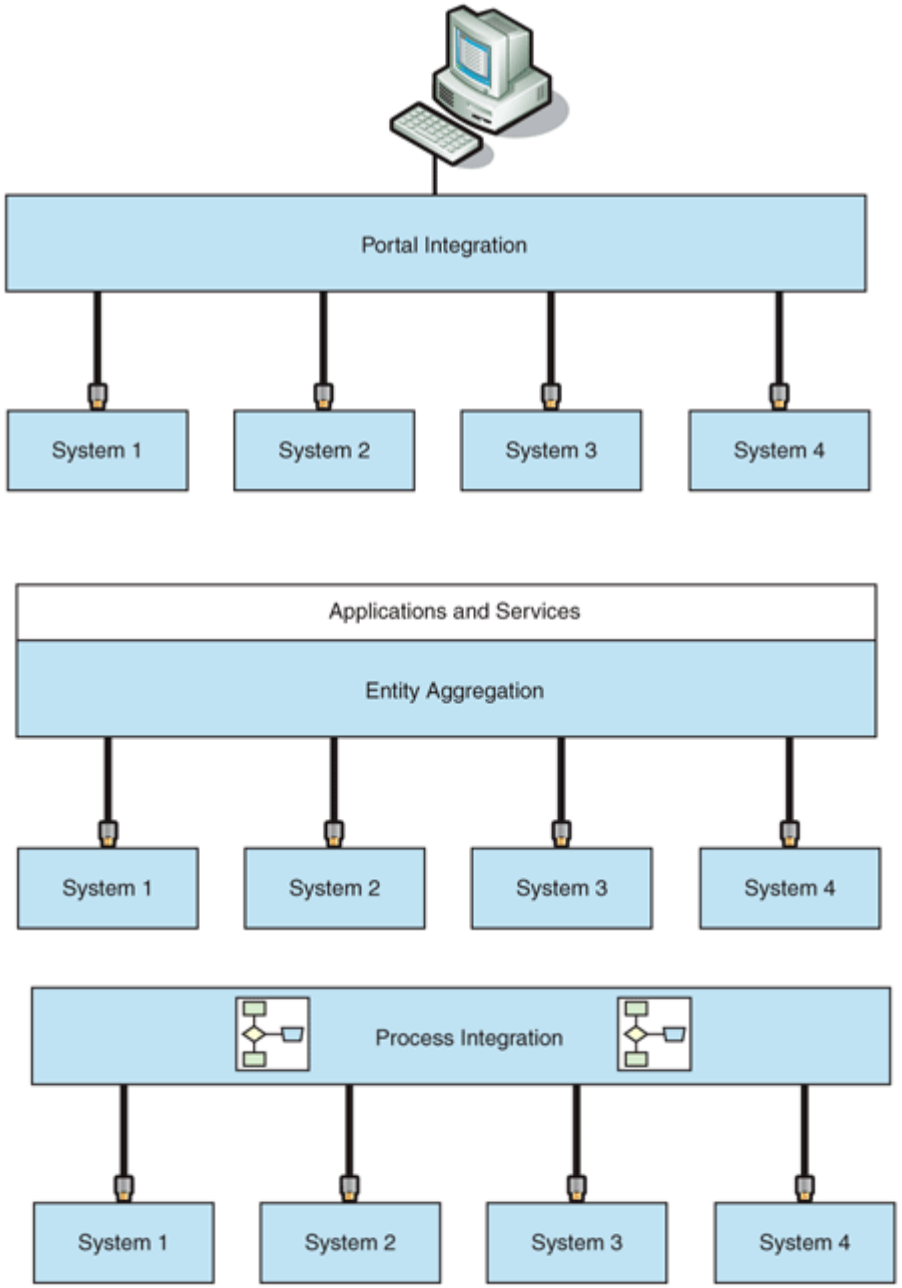
Integration layer capabilities

- **Communication, Service Interaction, and Integration**:
  - ability to route requests to correct the provider after necessary message transformation and protocol conversion
  - ability to connect the service requestor to the service provider and its underlying solutions platforms realizing the requested service.
  - ability to discover services and, at runtime, to support the virtualization of services so that changes to the end-points (or locations from where the services are called and where the services are provided) can occur without impact to service consumers and service providers.
- **Message Processing**:
  - ability to perform the necessary message transformation to connect the service requestor to the service provider and to publish and subscribe messages and events asynchronously.
- **Quality of Service**:
  - handling of transactions and exceptions and other NFRs (non-functional requirements)
- **Security**:
  - helps in enforcement of access privileges and other security policies.
- **Management**:
  - ability to maintain service invocation history and monitor and track the service invocations.

https://msdn.microsoft.com/en-us/library/ff647962.aspx

Integration layer

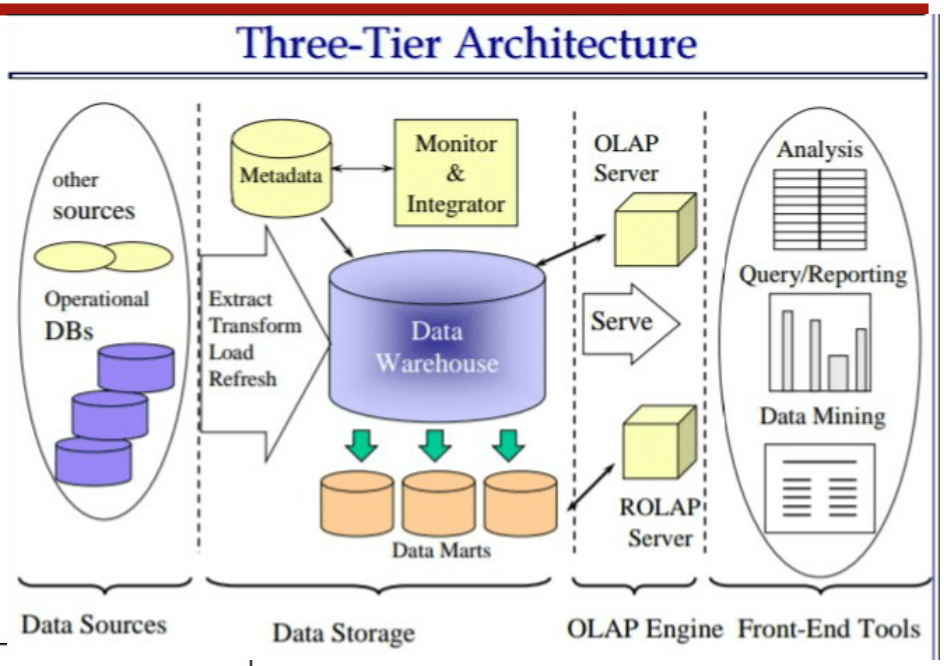
- can be used to orchestrate the activities across multiple applications and to keep track of state.
- is likely to require additional effort, but it manages all interactions from a central point without applications requiring information about each other
- Three approaches towards integrating layers:
  - *Entity Aggregation*
  - *Process Integration*
  - *Portal Integration*



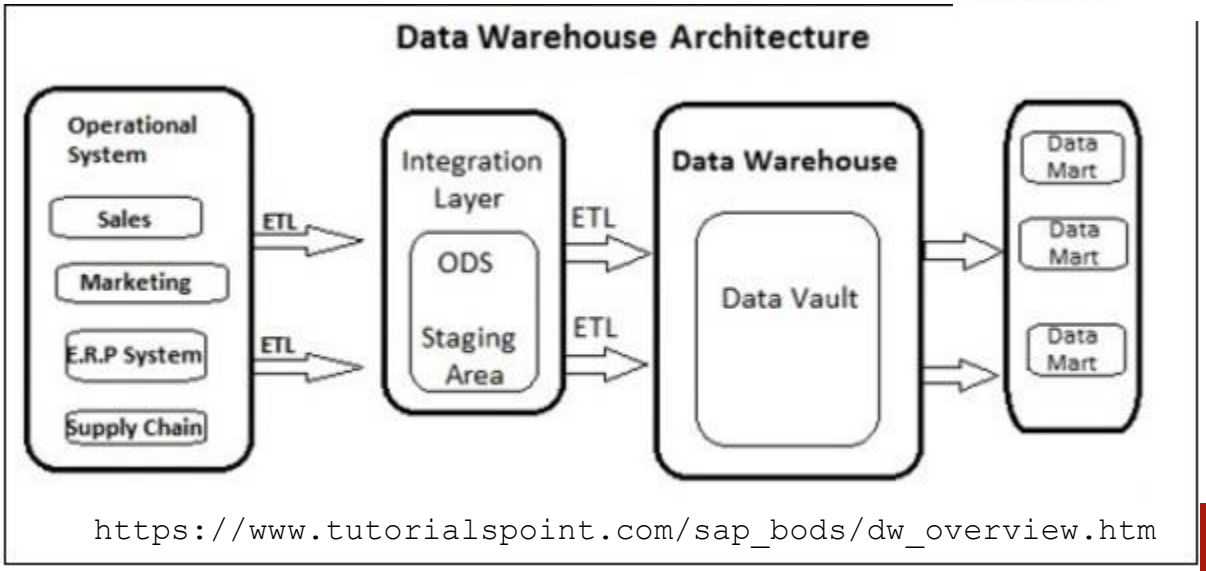
https://msdn.microsoft.com/en-us/library/ff647962.aspx

Data Warehouse architecture

- usually three layers:
  - Data Source Layer,
  - Integration Layer,
  - Presentation Layer



https://www.codeproject.com/Articles/1241849/Designing-and-Implementing-a-Data-Warehouse-in-the



https://www.tutorialspoint.com/sap_bods/dw_overview.htm

Integration Layer
Data integration

- ETL (Extract-Transform-Load)
  - in the Extract step, data is moved from the Source layer and made accessible in the Integration layer for further processing,
  - the Transformation step involves all the operational activities usually associated with the typical statistical production process,
  - as soon as a variable is processed in the Integration layer in a way that makes it useful in the context of data warehouse it has to be Loaded into the Interpretation layer and the Access layer.

http://www.dataintegration.info/etl

20 Best ETL / Data Warehousing Tools in 2019

- QuerySurge
- MarkLogic
- Panoply
- Oracle
- Amazon RedShift
- ….

https://www.guru99.com/top-20-etl-database-warehousing-tools.html

Presentation Layer - patterns

http://www.corej2eepatterns.com/PatternRelationships.htm

- Intercepting Filter
  - Intercepts incoming requests and outgoing responses and applies a filter.
- Context Object
  - Encapsulates state in a protocol-independent way to be shared throughout your application
- Front Controller
  - A container to hold the common processing logic that occurs within the presentation tier
- Application Controller
  - Centralizes control, retrieval, and invocation of view and command processing
- View Helper
  - Encourages the separation of formatting-related code from other business logic
- Composite View
  - suggests composing a View from numerous atomic pieces
- Dispatcher View
  - defers business processing until view processing has been performed.

https://stackabuse.com/java-j2ee-design-patterns

https://www.oracle.com/java/technologies/intercepting-filter.html

Business Layer - patterns

- Patterns `http://www.corej2eepatterns.com/PatternRelationships.htm`
  - Business Delegate
    - reduces coupling between remote tiers and provides an entry point for accessing remote services in the business tier.
  - Service Locator
    - encapsulates the implementation mechanisms for looking up business service components.
  - Session Façade
    - provides coarse-grained services to the clients by hiding the complexities of the business service interactions.
  - Application Service
    - centralizes and aggregates behavior to provide a uniform service layer to the business tier services.
  - Business Object
    - implements your conceptual domain model using an object model.
  - Composite Entity
    - implements a Business Object using local entity beans and POJOs.
  - Transfer Object
    - provides the best techniques and strategies to exchange data across tiers
  - T O Assembler
    - constructs a composite Transfer Object from various sources
  - Value List Handler
    - uses the GoF iterator pattern to provide query execution and processing services

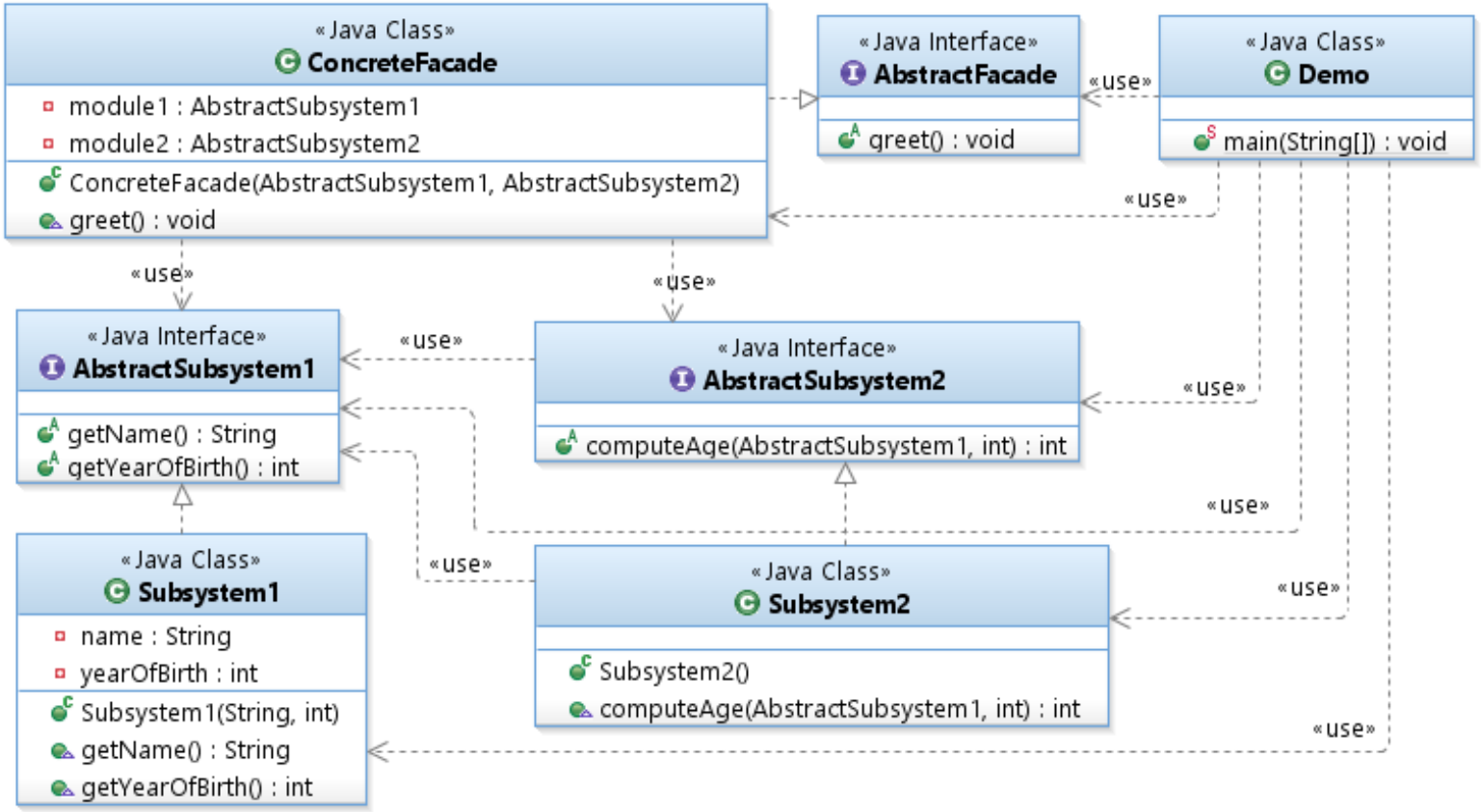
Integration Layer - patterns

- Patterns
  - Data Access Object
    - enables loose coupling between the business and resource tiers
  - Service Activator
    - enables asynchronous processing in your enterprise applications using JMS.
  - Domain Store
    - provides a powerful mechanism to implement transparent persistence for your object model.
  - Web Service Broker
    - exposes and brokers one or more services in your application to external clients as a web service using XML and standard web protocols

http://www.corej2eepatterns.com/DataAccessObject.htm
http://www.corej2eepatterns.com/ServiceActivator.htm
http://www.corej2eepatterns.com/DomainStore.htm
http://www.corej2eepatterns.com/WebServiceBroker.htm

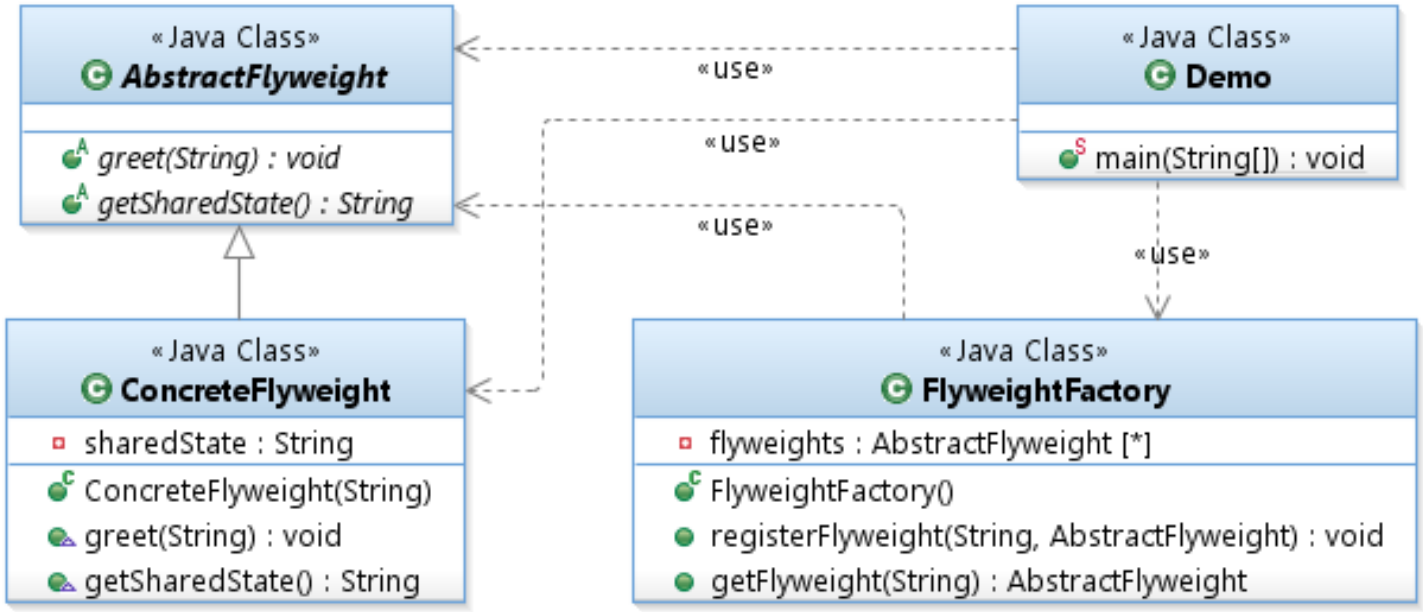
Facade pattern

- adds an interface to existing system to hide its complexities (structural pattern)
- applies to the complex or poorly designed subsystems

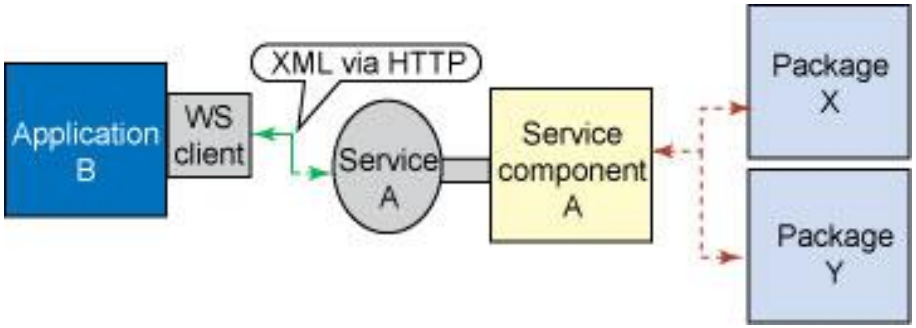


Flyweight pattern

- reduces the number of objects created (structural pattern), decreases memory footprint, increases performance
- allows to reuse already existing objects by storing them and creating new objects only when no matching object is found



Service component as a facade



https://www.ibm.com/developerworks/library/ar-archtemp/index.html