2021/2022

Process and Service Design

POLITECNICO DI MILANO

LUCA GERIN

Summary

1. Service Definition & SOA

What is a **Service**?

*"A **service** is a **change in the condition** of a person, or a good belonging to some economic unit, which is brought about **as the result of the activity** of some other economic unit, **with the prior agreement** of the former person or economic unit"*

P. Hill, On goods and services

*"A **service** is a mechanism to enable access to one or **more capabilities**, where the access is provided using a **prescribed interface** and is exercised consistent with constraints and policies as specified by the **service description**. A service is **provided by an entity** – the service provider – **for use by others**, but the eventual consumers of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by the provider."*

OASIS SOA Reference Model

The meaning of term service is intuitive but a definition of terms service can be different in different contexts like the Economic one and ICT.

In the economic perspective, we can distinguish:

- Goods-dominant logic: focus on goods exchange; goods are tangible, has embedded value (value-in-use) and the exchanges are based transactions.
- Service-dominant logic: focus on service provision; intangible resource which requires a co-creation of value and the provisions are based on relation.

Service-dominant logic (S-D logic) is useful because it implies more interaction with customers and more focus on why a product fits to the customer needs instead of technical specification.

Personal Service
Self-Service
Electronic Service
Pre Industrial Era
Industrial Era
Information Era

Throughout history, at first services were *Personal Services*, as they were man-made and highly customized but with a limited production. Then they became *Industrial Services*in the industrial era, fit for mass production but with practically no customization.

Nowadays, the information era brings *Electronic Services*, which are highly computer based and therefore with high customization offered to customers combined with capability for mass production.

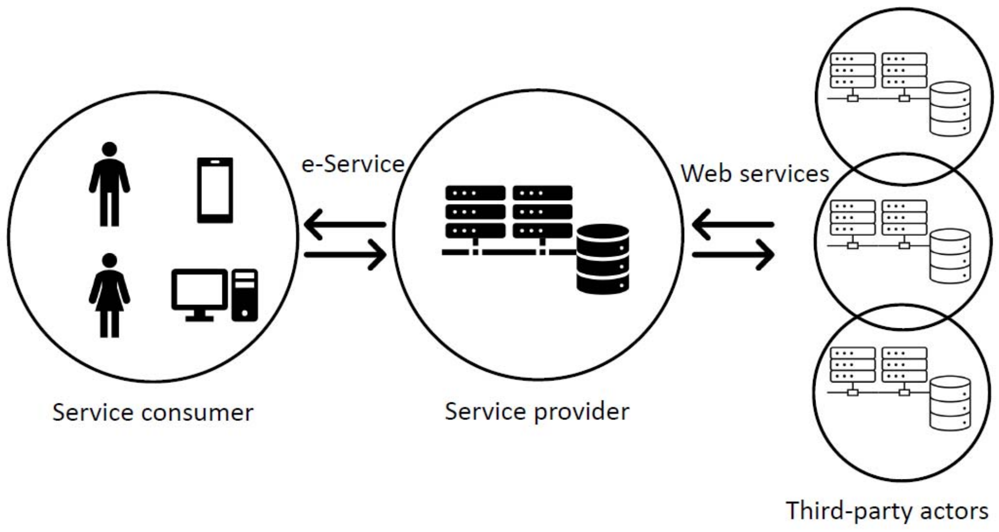
In the IT domain, these services are called Web Services.

When technology comes into play, there are two different perspectives of ICT in S-D logic, which go under the name of **e-Service**:

- As the improvement of a programming paradigm
  - ICT is a goal and it is offered itself as a service
  - Cloud is based on offering resources as services
  - SOA becomes the new way of developing software to be service-ready
- As the automation of economics activities and self-service
  - ICT is a mean
  - The goal is to make the business able to provide services
  - ICT can improve the effectiveness and the efficacy
  - BPM plays an important role

A lot of attention is devoted to aspects like Customization, Scalability, Reliability and Security and Privacy.

The domain of interest is the following:

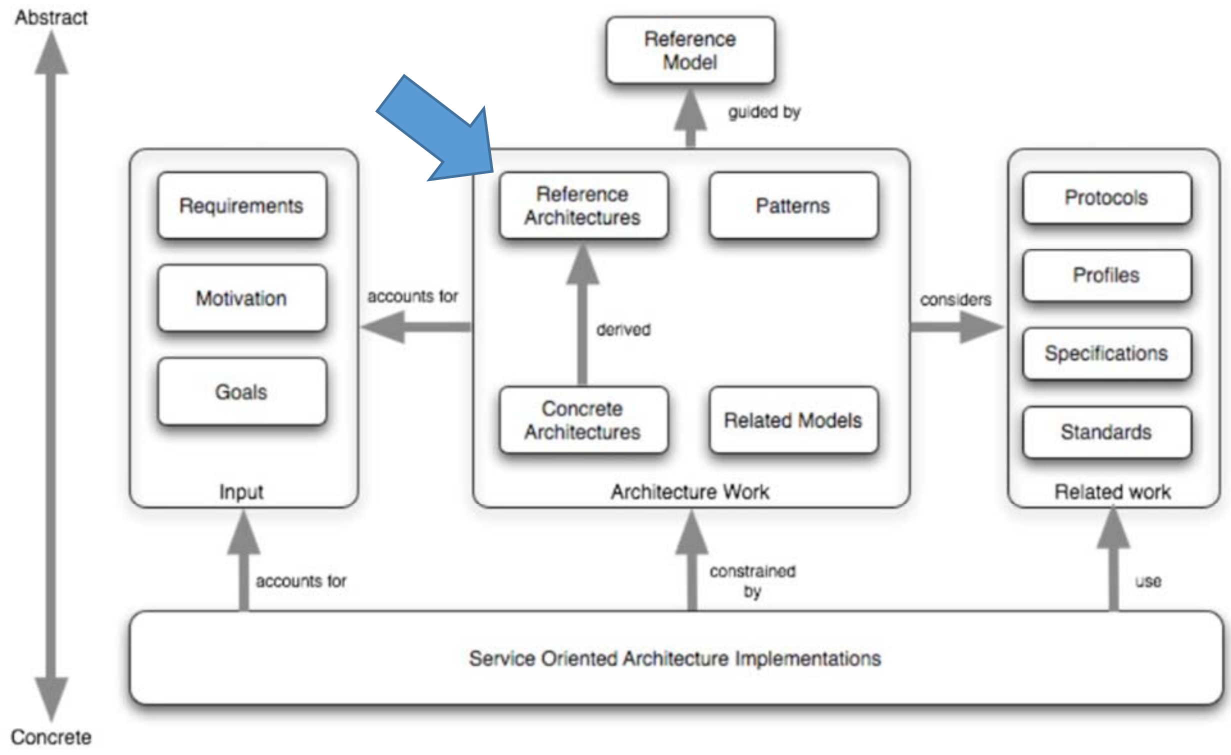


Usually, there are more service consumers who use a GUI (Graphical User Interface) to access the e-Service provided by the service provider. The service provider itself works with third party actors, who are in a way service providers for the service provider, accessing to web services through APIs (Application Programming Interfaces). These services can be used to compose the e-Service provided to the final consumer.

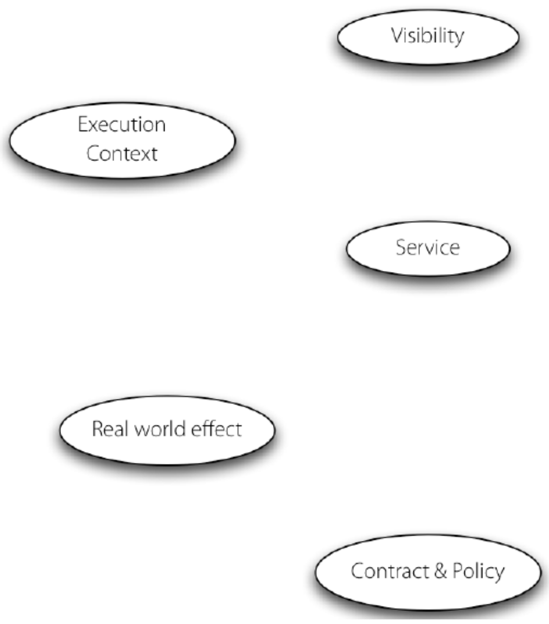
Service-Oriented Architecture (SOA) is a style of software design where services are pro-vided to the other components by application components, through a communication protocol over a network. In service oriented architecture, a number of services communicate with each other.

SOA is a reference model, which is an abstract framework for understanding significant relationships among the entities of some environment. It consists of a minimal set of unifying concepts, axioms and relationships within a particular problem domain. It enables the development of specific reference or concrete architecture and is independent of specific standards, technologies, implementations.

Concrete architectures arise from a combination of reference architectures, architectural patterns and additional requirements, including those imposed by technology environments.



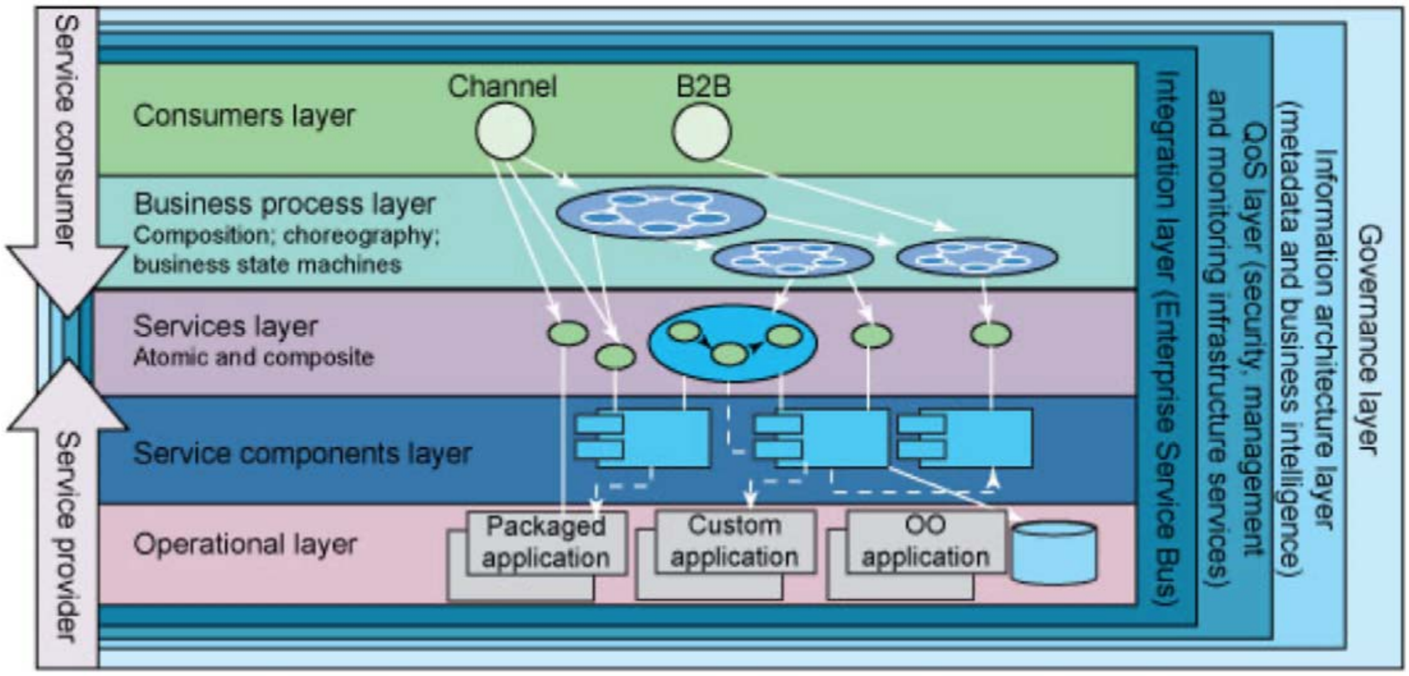
In the end, Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. SOA provides a powerful framework for matching needs and capabilities and for combining capabilities to address those needs.



The main concept inside a reference model such as SOA are the following:

- **Service**: A service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. A service is provided by an entity - the service provider - for use by others, but the eventual consumers of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by the provider. A service is accessed by means of a service interface, where the interface comprises the specifics of how to access the underlying capabilities.
- **Visibility**: it refers to the capacity for those with needs and those with capabilities to be able to see each other. For a service provider and consumer to interact with each other they have to be able to "see" each other. Preconditions to visibility are awareness, willingness and reachability. The initiator must be aware of the other parties, the participants must be predisposed to interaction, and the participants must be able to interact.
- **Interaction**: between the user and the provider, to perform the capabilities of the service. In many cases, the interaction is accomplished by sending and receiving messages.
- **Real World Effect**: both the consumer and the provider are trying to achieve some result by using the service, known as the “real world effect” of using a service.
- **Service Description**: the information needed to use a service. In most cases, there is no one “right" description but rather the elements of description required depend on the context and the needs of the parties using the associated entity.
- **Policies and Contracts**: A policy represents some constraint or condition on the use, deployment or description of an owned entity as defined by any participant. A contract, on the other hand, represents an agreement by two or more parties.
- **Execution Context**: the set of infrastructure elements, process entities, policy assertions and agreements that are identified as part of an instantiated service interaction, and thus forms a path between those with needs and those with capabilities.

The IBM SOA Reference Architecture


Service consumer
Service provider
Consumers layer
Channel
B2B
Business process layer
Composition; choreography; business state machines
Services layer
Atomic and composite
Service components layer
Operational layer
Packaged application
Custom application
OO application
Integration layer (Enterprise Service Bus)
QoS layer (security, management and monitoring infrastructure services)
Information architecture layer (metadata and business intelligence)
Governance layer

The **IBM SOA Reference Architecture** is a layer architecture that expresses the five main aspects, represented as vertical layers, that must be considered when the architecture is built.

There are some layers that are visible only to service providers and some visible only to service consumers. Consumers can see the consumer layer, the Business Process Layer and the Services Layer, while the provider can see the Operational Layer, the Service Components Layer and the Services layer. The Service Layer, which stands in the middle, is the only one visible to both actors.

Going in details about the layers:

- **Operational layer**: description of the infrastructure on top of which to create the service architecture. It contains all the applications and infrastructures already existing in our architecture. This layer has the task to open specific sections and views of the IT architecture to the above layer. It contains the current applications, from legacy applications to up-to-dated applications
- **Service Component Layer**: contains the components to combine to implement the service and expose the functionalities of the operational layer. These components are mainly software components, each of which provide the implementation or "realization" for a service, or operation on a service. Software components hide the complexity of the underlying system and exposes with standard interface what defined in the Service Layer.
- **Service Layer**: the list of service descriptions made visible from the provider to the user. This layer contains the descriptions for services and business capabilities. Here service consumer is able to read the service description, while the service provider can decide to offer service components 'as-ease' or can combine services to create something new to offer.
- **Business Process Layer**: models the interaction between different services (even from different providers) according to the user needs. This layer contains the definition of processes seen as service composition, possibly combining different services from different service providers. Here consumers can compose services using two strategies: orchestration and choreography.
- **Consumer Layer**: this layer is the point where consumers, whether it be a person, program, browser or automation, interact with the SOA. It enables an SOA solution to support a client-independent, channel agnostic set of functionality, which is separately consumed and rendered through one or more channels. This layer provides a solution to offer to the final users an access to the services: through different channels and traversing different layers.

Orthogonally to these main layers, there are then other architectures concerning other non-functional aspects. These are represented as Cross-cutting layers and are horizontal aspects concerning service provisioning or usage, like security, quality, governance, and so on.

- **Integration Layer**: is a cross-cutting concern that enables and provides the capability to mediate, which includes transformation, routing and protocol conversion to transport service requests from the service requester to the correct service provider.
- **Quality of Service Layer**: is a cross-cutting concern that supports non-functional requirement (NFR) related concerns of an SOA and provides a focal point for dealing with them in any given solution.
- **Information Architecture Layer**: is a cross-cutting concern that is responsible for manifesting a unified representation of the information aspect of an organization. This layer includes information architecture, business analytics and intelligence.
- **Governance Layer**: is a cross-cutting concern that ensures that the services and SOA solutions within an organization are adhering to the defined policies, guidelines and standards that are defined as a function of the objectives, strategies and regulations applied in the organization and that an SOA solutions are providing the desired business value.

E-Service

A service is an activity (not necessarily atomic) that can be more or less complex, so it may imply the existence of a process.

A service involves two actors: a Service Provider and a Service Consumer. A service provider may become a service consumer and vice versa.

A service requires a prior agreement. Usually service exchanges last longer than good exchange and the interaction between providers and consumer could be complex.

"*Service systems comprise service providers and service clients working together to co-produce value in complex value chains or networks*"

S. Vargo, R. Lusch, The four service marketing myths: remnants of goods-based, manufacturing model, 2004

Co-creation implies that customers and providers are not fixed roles: provider may become customer of someone else and a customer may become a provider

Service system can be more or less complex involving a different number of actors, so governance of a service network is not an easy task.

There are different scenarios concerning customers and providers' interaction:

a) **Person acting on person**: a person delivers a service to another person and no belongings are involved on both sides
b) **Person with belonging acting on person**: a person delivers a service to another person and provider's belongings are used to deliver the service
c) **Person with belonging acting on belonging**: a person delivers a service to another person changing the conditions of the customer belonging and provider's belongings are used to deliver the service.
d) **Belonging acting on person**: customer interacts with providers belonging to exploit a service, so provider's belongings are used to deliver the service. Customer needs to interact with machines (H2M, Human to Machine)
e) **Belonging acting on belonging**: belongings are directly interacting. Providers belongings are used to deliver the service and customer needs to be equipped with belonging able to communicate with the provider belonging (M2M, Machine to Machine)

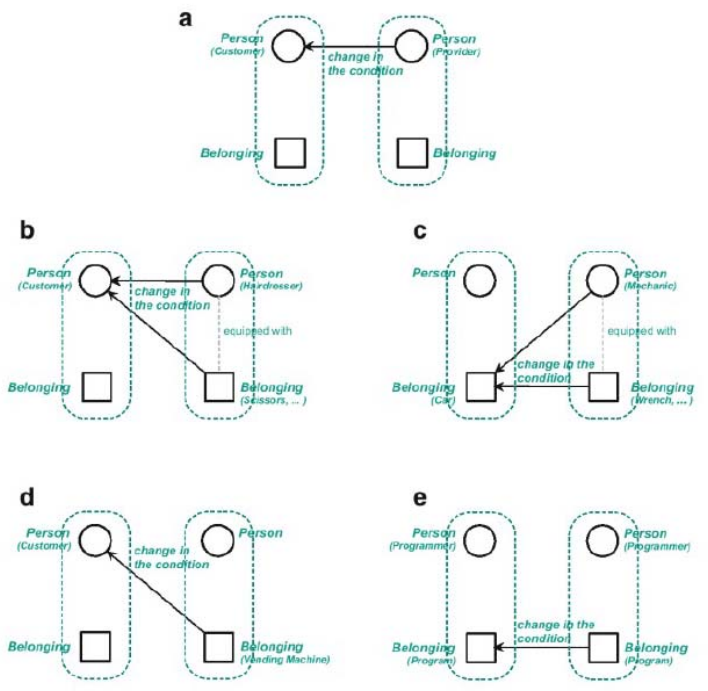


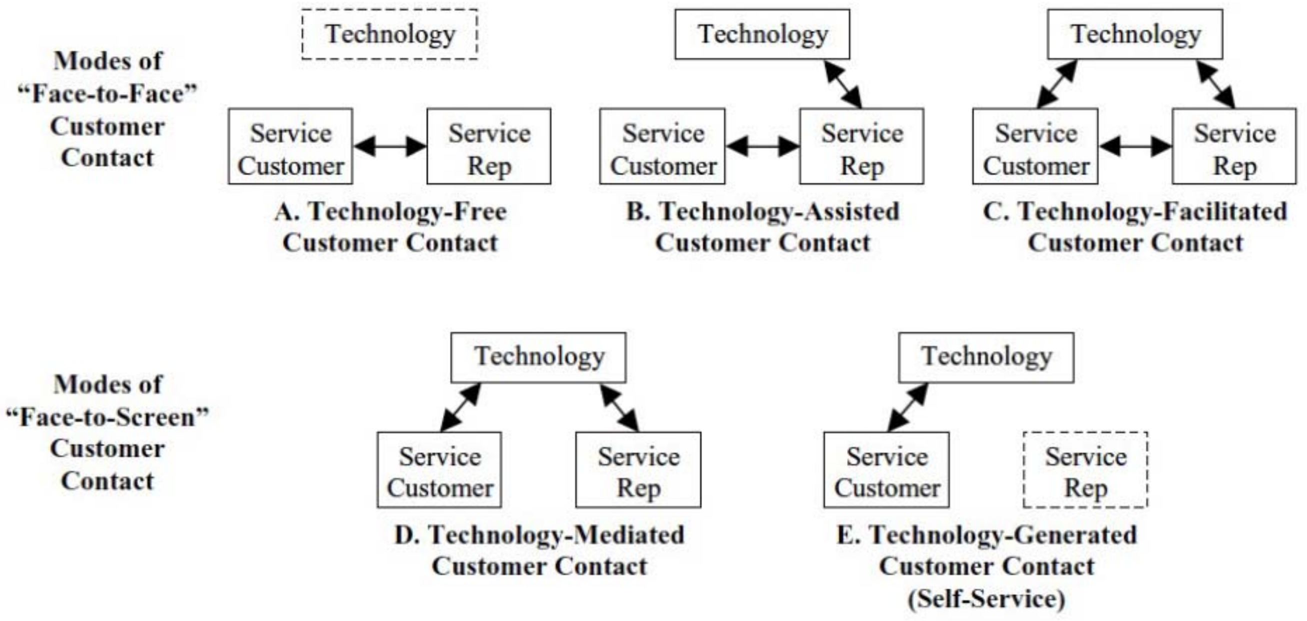
Fig. 1.6 Different service scenarios according to the involvement of persons and their belongings. (**a**) Person acting on person. (**b**) Person with belonging acting on person (Example: Haircut). (**c**) Person with belonging acting on belonging (Example: Car Repair). (**d**) Belonging acting on person (Example: Self Service). (**e**) Belonging acting on belonging (Example: Web Service)

The **resource intensity** of a service is a measure of the resources required for the provision of that service. There are:

- Labor- and capital-intensive services
- Knowledge-intensive services
- Information-intensive services
- Technology-intensive services

Technology can be used at (both) consumer and provider side in different ways and with different pervasiveness. Technology then, can assume different roles.

The possible roles of technology are various.


Modes of "Face-to-Face" Customer Contact
Technology
Service Customer
Service Rep
A. Technology-Free Customer Contact
Technology
Service Customer
Service Rep
B. Technology-Assisted Customer Contact
Technology
Service Customer
Service Rep
C. Technology-Facilitated Customer Contact
Modes of "Face-to-Screen" Customer Contact
Technology
Service Customer
Service Rep
D. Technology-Mediated Customer Contact
Technology
Service Customer
Service Rep
E. Technology-Generated Customer Contact (Self-Service)

In models of **Face-to-Face** costumer contact, the service requires direct interaction between provider and consumer. Service providers and service consumers are human beings. It can be distinguished:

- <u>Technology-assisted services</u>: Service provider uses technology to improve the interaction with the service consumer, while Service consumer does not have access to the technology as the service provider acts as mediator.
- <u>Technology-facilitated services</u>: service provider uses technology to improve the interaction with the service consumer, and also the Service consumer can have access to the technology together with the service provider.

In models of **Face-to-Screen** costumer contact, there is no direct contact between the provider and the consumer. Technology is in the middle between providers and consumers. It can be distinguished:

- Technology-mediate services: service providers and consumers are human beings and they are using the same technology to interact
- Technology-generated services: Service providers are replaced by technology

Both kinds of Face-to-Screen interaction services are by definition e-Services.

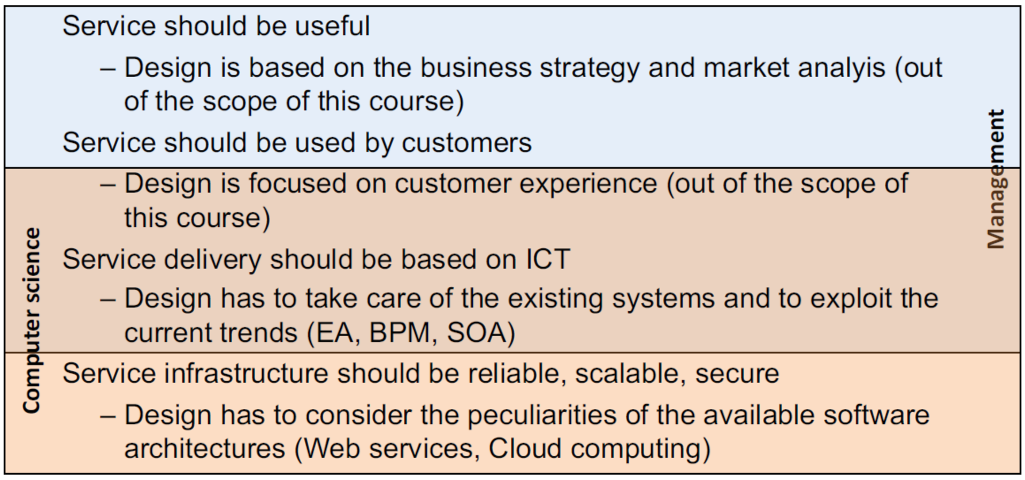
An **e-Service** is: "*An electronic service, a.k.a. e-Service, is a service system (with elements, a structure, a behavior, and a purpose) for which the implementation of many of this elements and behavior is done using automation and programming techniques.*"

J. Cardoso, Fundamentals of Service Systems

The value of e-Services can be nowadays declined in four points:

- **Accessibility**: services can be accessed almost everywhere. Now the communication infrastructure is now available everywhere since the adoption of smartphone is covering the majority of population.
- **Delivery**: delivery is almost instantaneous, but it depends on the nature of the service
- **Human touch**: customer experience is one of the main drawbacks of e-Services adoption, that's why technology is trying to reduce this gap putting focus on customer experience
- **Personalization**: one-to-one relationship is fundamental, private data need to be shared. There is usually just one provider for many consumers, but providers want to make the impression that the service is tailored just for the one consumer.

The e-Service provisioning is usually driven by a business process and the customer of the service is one of the parties involved in the business process. The business process, to be enacted, could be based on other services: some of them human-based, some of them automatic.


Management
Computer science

Service should be useful
– Design is based on the business strategy and market analyis (out of the scope of this course)

Service should be used by customers
– Design is focused on customer experience (out of the scope of this course)

Service delivery should be based on ICT
– Design has to take care of the existing systems and to exploit the current trends (EA, BPM, SOA)

Service infrastructure should be reliable, scalable, secure
– Design has to consider the peculiarities of the available software architectures (Web services, Cloud computing)

2. Modeling Enterprise Architecture

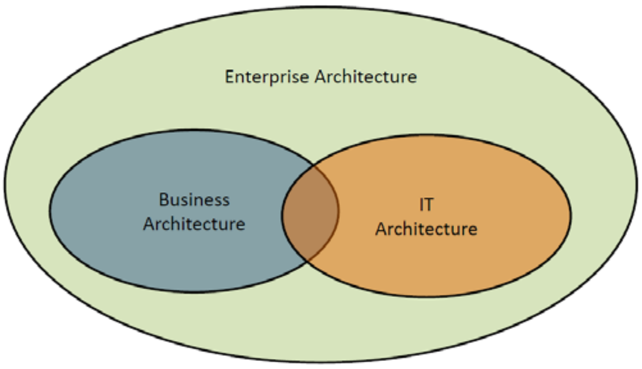
"Architecture is a set of fundamental concepts or properties of a system in its environment, embodied in its elements, relationships, and in the principles of its design and evolution"

ISO 42010:2011

"Enterprise is any collection of organizations that has a common set of goals"

TOGAF

*"**Enterprise architecture** is a coherent whole of principles, methods, and models that are used in the design and realization of an enterprise's organizational structure, business process, information systems, and infrastructure"*



An EA is set of principles to use to create a system good to work in the environment of interest.

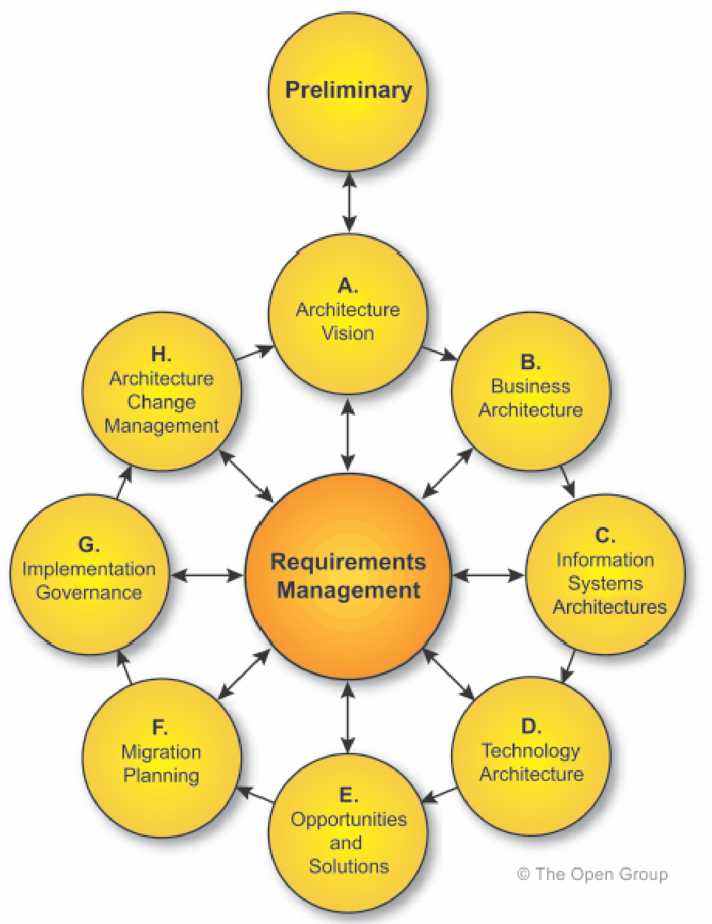
We are mainly focused on IT architecture, but business is built on top of IT architecture. To be even more precise, the two aspects interplay: IT and business are in strict connection. In fact, generally talking, innovation can come from top (from the business) or from the bottom (from the IT).

A company needs to transform accordingly to the environment to remain competitive and an Enterprise Architecture is a way of facing this complexity.

EA is a conceptual framework that describes how the business is constructed, identifies primary components and show the relationship between them. An EA is useful to understand the organization from different points of view and for the organization to evolve.

One of the objective of EA is to reduce complexity and understand how business and Information Technology work together. Business models change and also the technology evolves, so they need to remain aligned. How technology supports the business proves is a key point illustrated in EA.

EA of a company comprehends: business architecture, application architecture, data architecture, technology architecture.


Preliminary
A. Architecture Vision
B. Business Architecture
C. Information Systems Architectures
D. Technology Architecture
E. Opportunities and Solutions
F. Migration Planning
G. Implementation Governance
H. Architecture Change Management
Requirements Management
© The Open Group

An **EA framework** is a guide about managing the life cycle of an EA. It allows to identify the aspects of EA, model it, provide views for different stakeholders and to define transitions between different states of the EA. EA frameworks could be seen as a set of standardized methodologies, methods, tools, and best practices.

The Open Group Architecture Framework (**TOGAF**) is one of the most used frameworks for enterprise architecture and it provides an approach for designing, planning, implementing, and governing an enterprise information technology architecture. TOGAF is a high-level approach to design and is typically modeled at four levels: Business, Application, Data, and Technology.

The **Architecture Development Method** (**ADM**) is applied to develop an enterprise architecture which will meet the business and information technology needs of an organization. It may be tailored to the organization's needs and is then employed to manage the execution of architecture planning activities. The process is iterative and cyclic. Each of the 8 +1 steps check with Requirements.

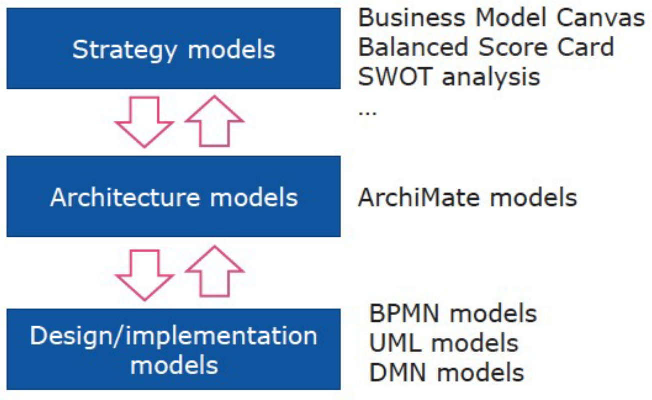
Archimate is an extensible language for enterprise architecture modeling, mainly used to describe building Blocks.

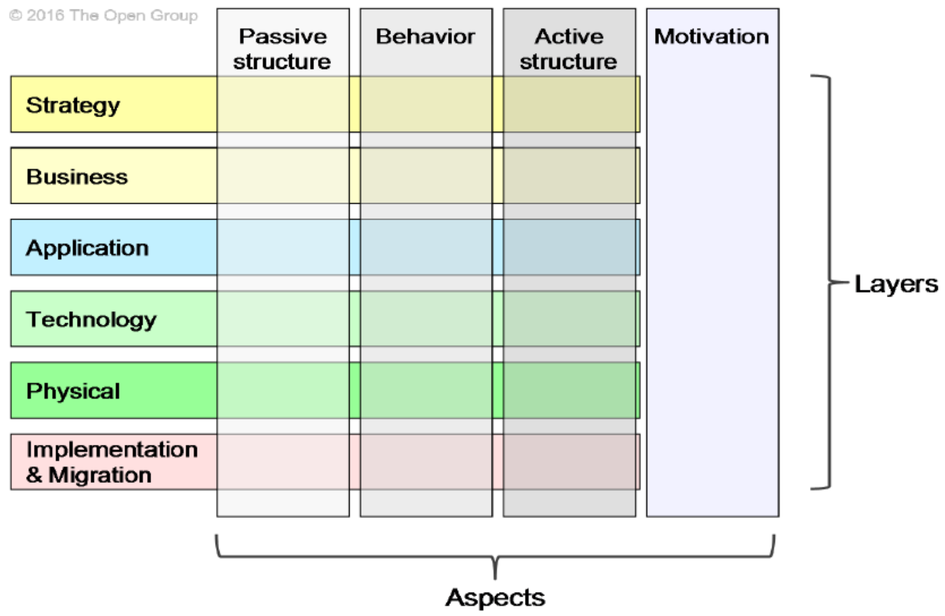
The objective is to support the description, analysis and visualization of architecture within and across business domains in an unambiguous way.

ArchiMate offers a common language for describing the construction and operation of business processes, organizational structures, information flows, IT systems, and technical infrastructure. This insight helps the different stakeholders to design, assess, and communicate the consequences of decisions and changes within and between these business domains.

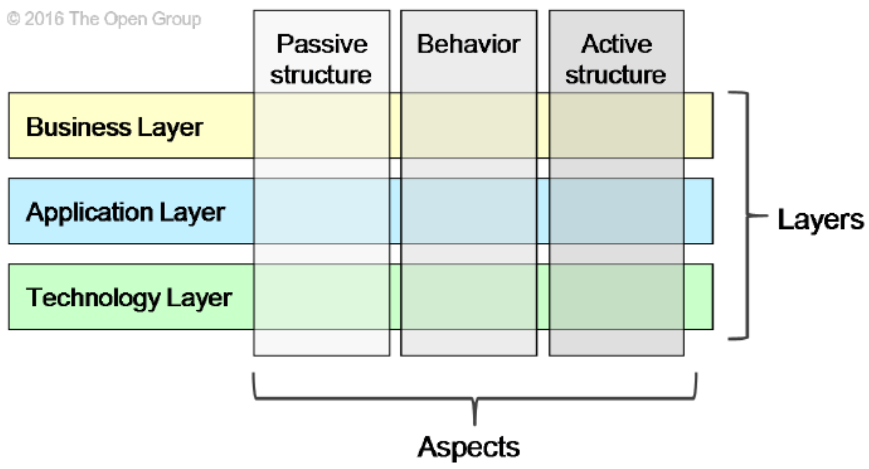
The main concepts and relationships of the ArchiMate language can be seen as a framework, the so-called Archimate Framework. It divides the enterprise architecture into different layers.

ArchiMate®





The core framework is composed of the three most important layers: Business, Application and Technology.



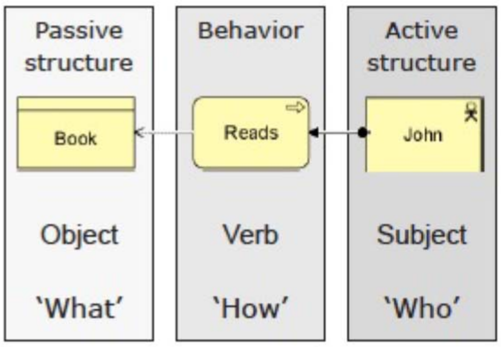
The <u>Business Layer</u> is about how the company is able to offer products and services to external costumers.

The <u>Application Layer</u> supports the business layer with application services, realized by software applications.

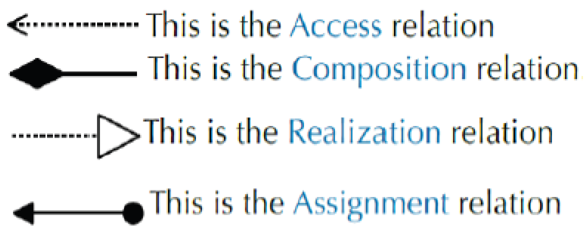
The <u>Technology Layer</u> offers infrastructure services needed to run applications. It is realized by computer and communication.

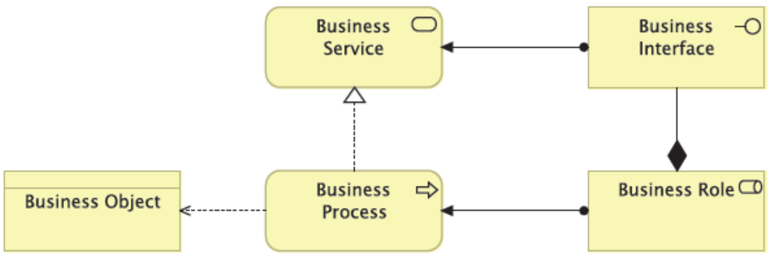
There are three main “shapes” in Archimate:

- Active structure
- Behavior
- Passive structure



Archimate Business Layer


This is the Access relation
This is the Composition relation
This is the Realization relation
This is the Assignment relation



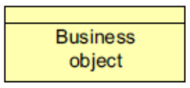
Active elements:

Business actor	A business entity that is capable of performing behavior
Business role	The responsibility for performing specific behavior, to which an action can be assigned
Business collaboration	*An aggregate of two or more business internal active structure elements that work together to perform collective behavior*
Business interface	A point of access where a business service is made available to the environment

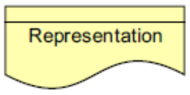
Behavior elements:

Business process	A sequence of business behaviors that achieves a specific outcome
Business function	A collection of business behaviors based on a chosen set of criteria, aligned to an organization
Business interaction	*A unit of collective business behavior performed by two or more business roles*
Business service	An explicitly defined exposed business behavior
Business event	A business behavior element that denotes an organizational state change.

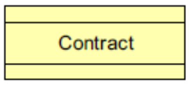
Passive elements:



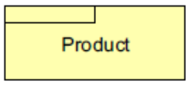
A concept used within a particular business domain



The perceptible form of information carried out by a business object



A formal or informal specification of an agreement between a provider and a consumer



A collection of services and passive structure elements, which is offered as a whole to customers

Structural Relationships		**Notation**
Composition	Indicates that an element consists of one or more other concepts.	◆——
Aggregation	indicates that an element groups a number of other concepts.	◇——
Assignment	Expresses the allocation of responsibility, performance of behavior, or execution.	●——▶
Realization	Indicates that an entity plays a critical role in the creation, achievement, sustenance, or operation of a more abstract entity.	┈┈▷
Dependency Relationships		**Notation**
Serving	Models that an element provides its functionality to another element.	——>
Access	Models the ability of behavior and active structure elements to observe or act upon passive structure elements.	┈┈ ┈┈> <┈┈>
Influence	Models that an element affects the implementation or achievement of some motivation element.	__+/-__>
Dynamic Relationships		**Notation**
Triggering	Describes a temporal or causal relationship between elements.	——▶
Flow	Transfer from one element to another.	------▶
Other Relationships		**Notation**
Specialization	Indicates that an element is a particular kind of another element.	——▷
Association	Models an unspecified relationship, or one that is not represented by another ArchiMate relationship.	——
Junction	Used to connect relationships of the same type.	● (And) Junction ○ Or Junction

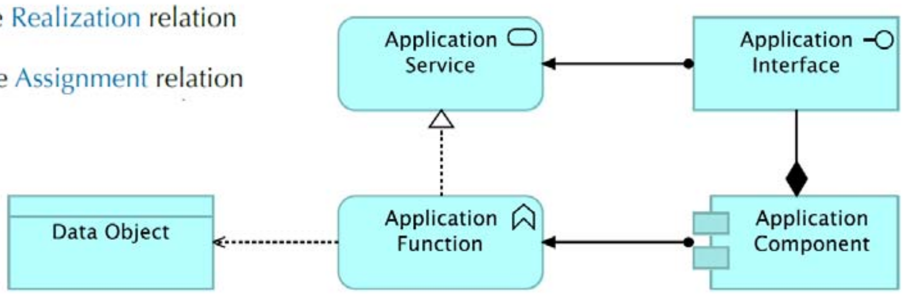
Archimate Application Layer

<┈┈┈ This is the Access relation

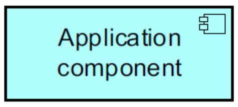
◆—— This is the Composition relation

┈┈▷ This is the Realization relation

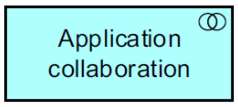
◀——● This is the Assignment relation



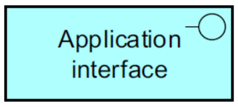
Active elements:



An encapsulation of application functionality aligned to implementation structure, which is modular and replaceable

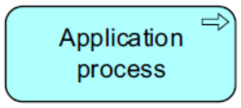


An aggregate of two or more application components that work together to perform collective behavior

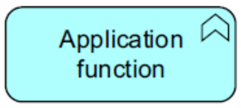


A point of access where an application service is made available to a user, another application component, or a node

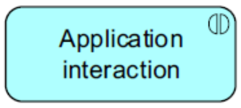
Behavior elements:



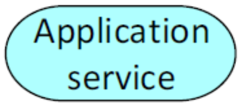
A sequence of application behaviors that achieves a specific outcome



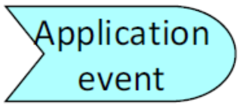
Automated behavior that can be performed by an application component



A unit of collective application behavior performed by two or more application components

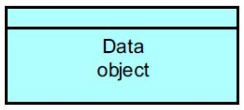


An explicitly defined exposed application behavior



An application behavior element that denotes a state change.

Passive elements:



Data structured for information processing

3. Business Process and BPM

To define a **Business Process** from a managerial point of view:

"A Business process is a collection of inter-related events, activities and decision points that involve a number of actors and objects, and that collectively lead to an outcome that is of value to at least one customer"

Dumas, La Rosa, Mendling, Rejiers, Fundamentals of Business Process Management, Springer, 2013

While its definition from and IT point of view is the following:

"A Business Process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations"

Weske, Business Process Management: Concepts, Languages, and Architectures (2nd ed.), Springer, 2012

While at an organizational level, business processes are essential to understanding how companies operate, business processes also play an important role in the design and realization of flexible information systems.

The main elements of a business process are:

- Organizations
- Activities/Tasks (units of work)
- Coordination
- Events (many events define a BP)
- Outcome

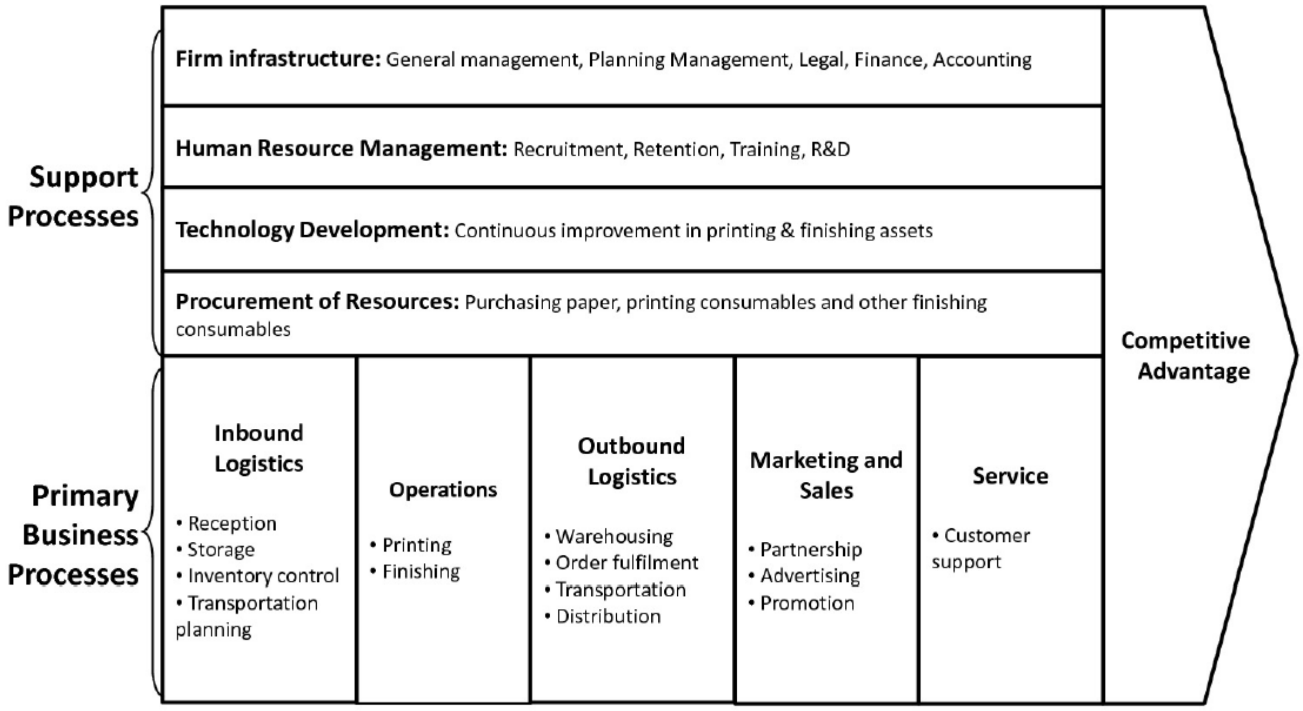
Note: activity and process are two different and distinct concepts.

"***Organizations** are social entities that are goal directed and designed as deliberately structured and coordinated activity systems, and are linked to the external environment*"

R. Daft, "Organization Theory and Design", Cengage Learning, 2006

Organizations are often defined in terms of processes that are performed inside the organization, in fact it's possible to say that an organization is nothing more than a set of processes.

Porter Value Chain breaks down activities to analyze their contribution to the commercial success of an organization. These activities need to be organized in a way to be process oriented.


Support Processes
Firm infrastructure: General management, Planning Management, Legal, Finance, Accounting
Human Resource Management: Recruitment, Retention, Training, R&D
Technology Development: Continuous improvement in printing & finishing assets
Procurement of Resources: Purchasing paper, printing consumables and other finishing consumables
Primary Business Processes
Inbound Logistics
• Reception
• Storage
• Inventory control
• Transportation planning
Operations
• Printing
• Finishing
Outbound Logistics
• Warehousing
• Order fulfilment
• Transportation
• Distribution
Marketing and Sales
• Partnership
• Advertising
• Promotion
Service
• Customer support
Competitive Advantage

There are many models to describe business processes, the more useful and unambiguous they are, the better.

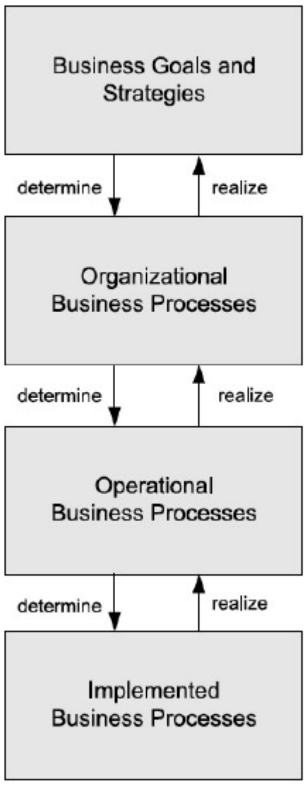
Different levels can be identified in business process management, ranging from high-level business strategies to implemented business processes.

At high level, **Business Goals and Strategies** are defined. They consist in long-term concepts with a high level description and without formal modeling.

These strategies determine the **Organizational Business Processes**, which are abstract business processes which realize the goals of the organization. Organizational business processes are high-level processes that are typically specified in textual form by their inputs, their outputs, their expected results, and their dependencies on other organizational business processes.

Organizational business processes determine the **Operational Business Processes**, that are executable models written in details. While organizational business processes characterize coarse-grained business functionality, typically there are multiple operational business processes required that contribute to one organizational business process. In operational business processes, the activities and their relationships are specified, but implementation aspects of the business process are disregarded.

From these models, **Implemented Business Processes** are created. They contain short terms concepts to actuate, provided with a low level description created with formal modeling. Implemented business processes contain information on the execution of the process activities and the technical and organizational environment in which they will be executed.


Business Goals and Strategies
determine
realize
Organizational Business Processes
determine
realize
Operational Business Processes
determine
realize
Implemented Business Processes

The **Business Process Management System** (BPMS) acts as the orchestrator of the executable model it is provided: it executes and controls the model. Otherwise, in alternative to the BPMS, there can be more and different systems and modules cooperating with each other's.

In any case, the IT is in charge of the control of the execution, not of the execution itself.

There are two main kinds of business processes:

- **Orchestration**: represents a single centralized executable business process (the orchestrator) that coordinates the interaction among different services. The orchestrator is responsible for invoking and combining the services. The orchestrator is an actor that is not able to do anything but coordinating the others.
- **Choreography**: is a global description of the participating services, which is defined by exchange of messages, rules of interaction and agreements between two or more endpoints. Choreography employs a decentralized approach for service composition. In choreography there is not an orchestrator, but processes are coordinated by a previous agreement between all the parts. In a sense, it's possible to define the choreography as a public protocol for the parties participating to follow in order to complete the process.

The choreography describes the interactions between multiple services, whereas orchestration represents control from one party's perspective.

Another way to classify business process is through their **degree of automation**. There are business processes that are fully automated, meaning that no human is involved in the enactment. On the other side, a lot of processes involve human activities in the loop, so less automation. Humans use interfaces that are offered to them to interact in the process.

Business processes can be distinguished also based on their **degree of repetition**. Usually, more repetitive processes do not involve human interaction, as they are the most automated and present the largest number of process instances (or cases). On the other hand, processes that have an occasional execution often involve more human interaction and less instances.

A process can be **well structured**, if it is completely defined before its execution starts. These are activity centric business processes. The different options for decisions that will be made during the enactment of the process have been defined at design time. On the contrary, **ad-hoc activities** are knowledge intensive business processes, which depend a lot on the knowledge and can vary from time to time. The order of execution can vary from case to case as it is defined by the knowledge applied to the circumstances. So, human component is fundamental because we need it to define the real structure of the process at run time. We are classifying the business processes according to their degree of structuring.

Business Process Management

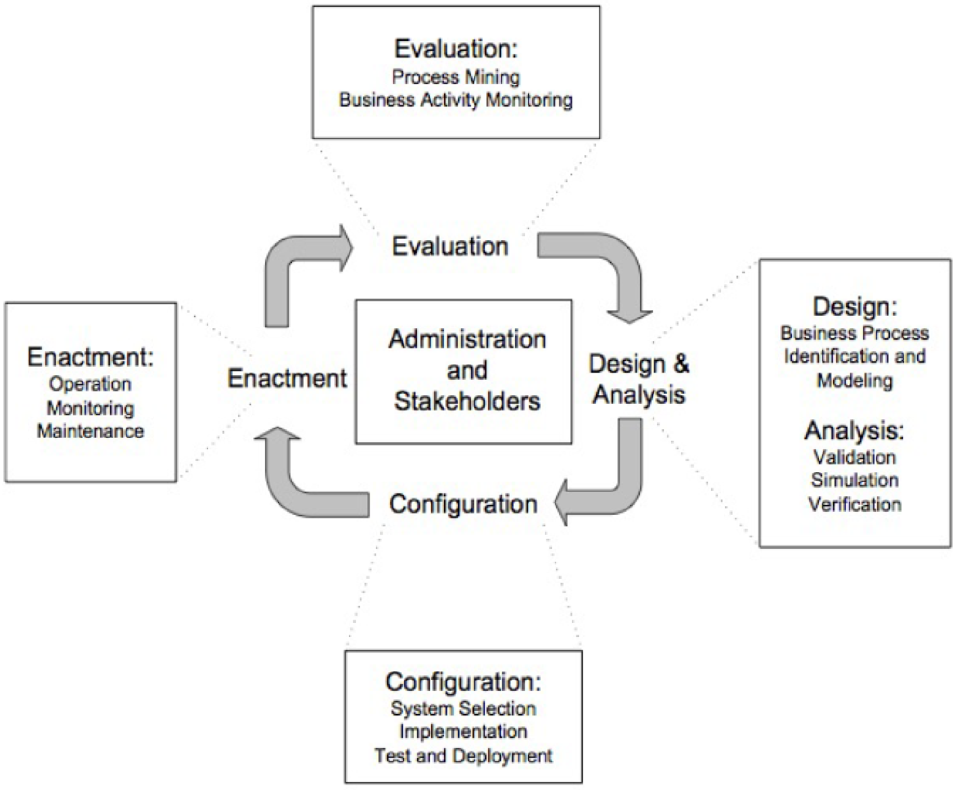
"***Business Process Management*** *(BPM) includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes*"

Weske, Business Process Management: Concepts, Languages, and Architectures (2nd ed.), Springer, 2012

Business Process Management is useful to:

- To understand what is happening in an organization
- To increase productivity and remove waste (Lean)
- To do the right things at the right time (Six Sigma)
- To increase the quality of an organization (TQM)
- To support not only operations but also managers
- To support "Industry 4.0"
- To exploit the Cloud (and Fog) Computing opportunities

One key aspect is managing the Business Process Lifecycle, shown in the image:



The business process lifecycle consists of phases that are related to each other and are organized in a cyclical structure, showing their logical dependencies. These dependencies do not imply a strict temporal ordering in which the phases need to be executed. Many design and development activities are conducted during each of these phases, and incremental and evolutionary approaches involving concurrent activities in multiple phases are not uncommon.

<u>Design and analysis</u>:

Surveys on the business processes and their organizational and technical environment are conducted. Based on these surveys, business processes are identified, reviewed, validated, and represented by business process models. Business process modelling techniques as well as validation, simulation, and verification techniques are used during this phase.

Design:

- Identification: Which are the processes now running in my organizations?
- Modeling: How are structured those processes?

Analysis:

- Validation/Simulation: Are they reaching to desired goals?
- Verification: Are they behave correctly?
- Qualitative: Principles to be followed
- Quantitative: Techniques that states how much good is the process

Configuration:

Once the business process model is designed and verified, the business process needs to be implemented, from the process design to the system design.

Business Process Management Systems, as generic software system driven by explicit process representation to coordinate the enactment of a business process, could be utilized. The business process model is enhanced with technical information that facilitates the enactment of the process by the business process management system.

There are two main strategies to adopt a BPMS:

- Make
- Buy

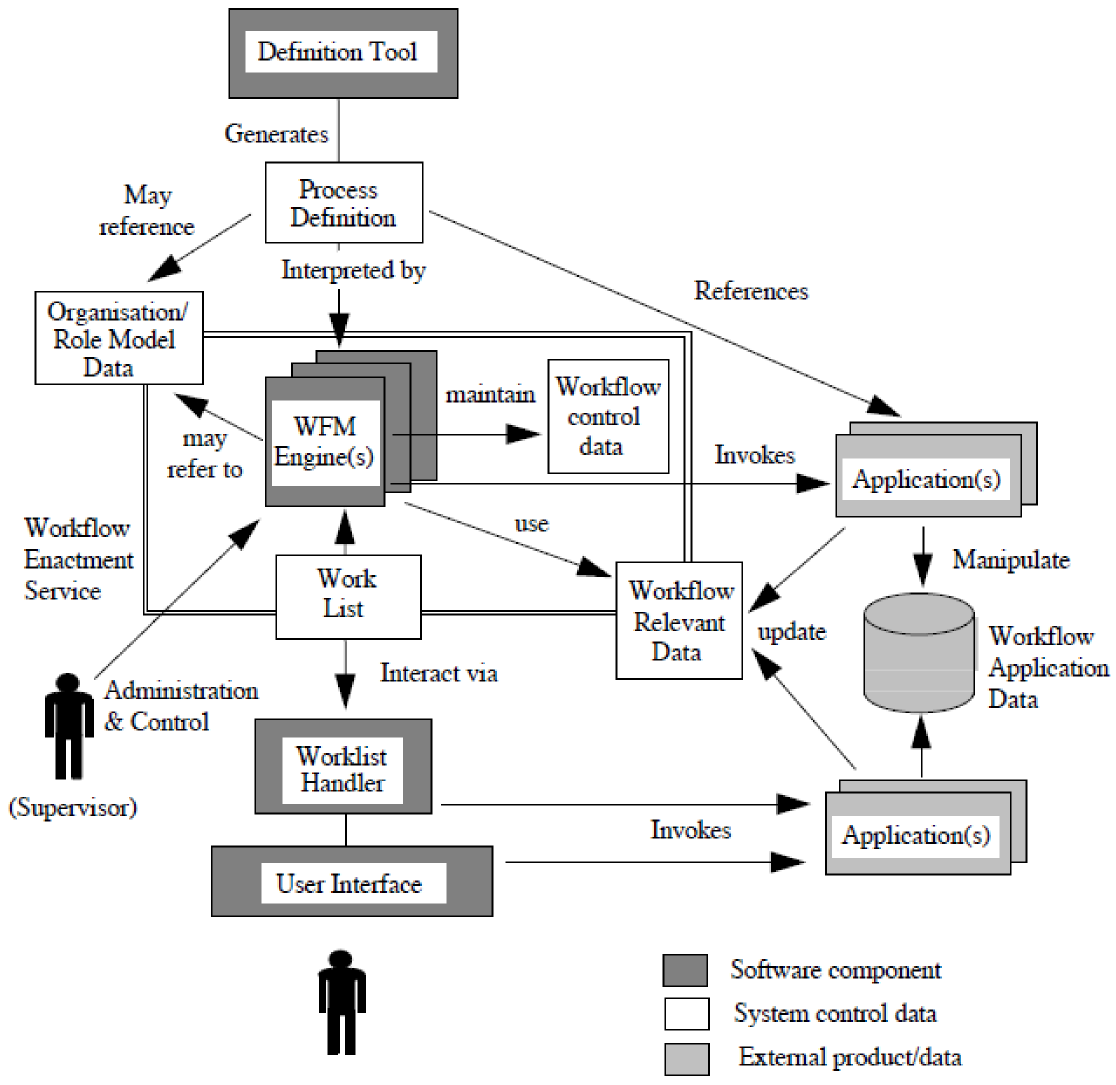
The buy option is more often chosen, but it requires integration.

Some elements need to be considered: the selection of Business Process Management Systems (BPMS), the adoption of Service Oriented Architecture, moving to the Cloud (BPaaS).

Enactment:

In this phase, the Business Process runs on the BPMS. Business process instances are initiated and the business process management system actively controls the execution of business process instances as defined in the business process model. Process enactment needs to cater to a correct process orchestration, guaranteeing that the process activities are performed according to the execution constraints specified in the process model. Process monitoring is an important mechanism for providing accurate information on the status of business process instances.

Logs are also collected to make possible the further step with process mining.



Evaluation:

The goal of this phase is to analyze the running and/or terminated business process to check:

- Compliance: ability to ensure that policies and procedures are designed to comply with internal and external policies.
- Conformance: ability to stick the execution of the process to the designed process model. BPMS should impose the model, so that there are no possibilities of not following the indicated path.

Once the analysis of the terminated execution is completed, what is learned is used to guide the improvement of the processes.

4. Business Process Modeling

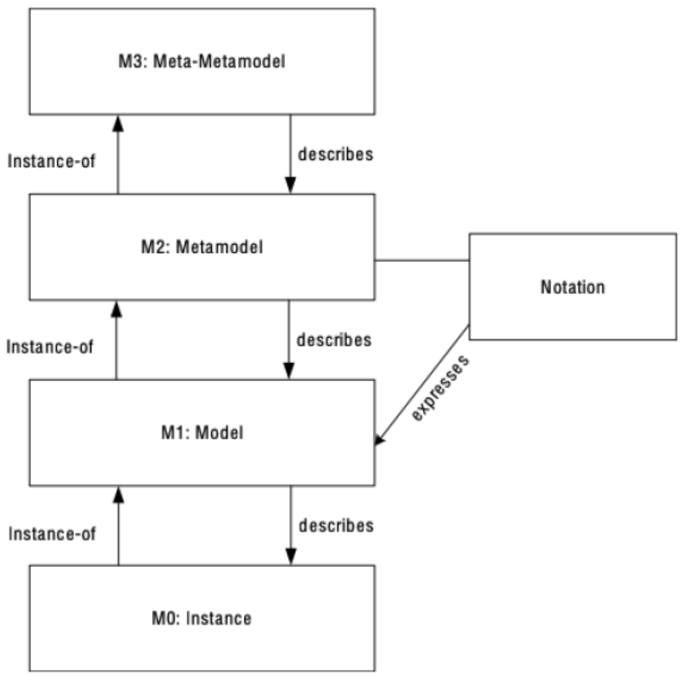
Business processes consist of activities whose coordinated execution realizes some business goal. These activities can be system activities, user interaction activities, or manual activities.

- **Manual activities** are not supported by information systems
- **User interaction activities** are activities that knowledge workers perform, using information systems and in which there is no physical activity involved.
- **System activities** do not involve a human user; they are executed by information systems.

Certain parts of a business process can be enacted by workflow technology. A workflow management system can make sure that the activities of a business process are performed in the order specified, and that the information systems are invoked to realize the business functionality.

Horizontal and Vertical Abstraction

To capture the complexity in business process management, different abstraction concepts are introduced. A traditional abstraction concept in computer science is the separation of modelling levels, from instance level to model level to meta-model level, denoted by **horizontal abstraction**.

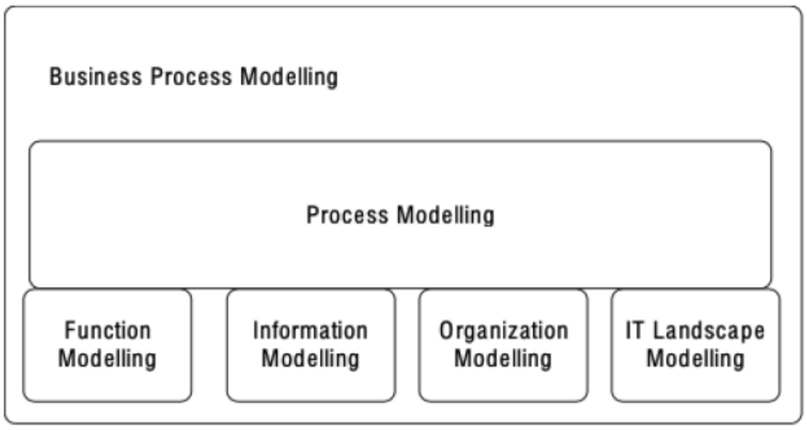


The first level from below is the **instance** level, where there are the concrete entities that are involved in business processes. Executed activities, concrete data values, and resources and persons are represented at the instance level.

To organize the complexity of business process scenarios, a set of similar entities at the instance level are identified and classified at the **model level**. For example, in object modeling, a set of similar entities is represented by a class, and in data modelling using the Entity Relationship approach, a set of similar entities is represented by an entity type, and similar relationships between entity types are represented by a relationship type.

Models are expressed in **metamodels** that are associated with notations, often of a graphical nature. In data modelling, the Entity Relationship metamodel defines entity types, relationship types, and connections between them.

The complete set of concepts and associations between concepts is called metamodel. A metamodel becomes useful if there is a notation for this metamodel that allows expressing models in a convenient way that allows communication between stakeholders in the modelled real-world situation.



A **vertical abstraction**, on the other hand, represents the modeling domains. These modeling domains are all integrated by the domain of **process modeling**. Process modeling is so at the center of the modelling effort, as it integrates the modelling efforts that are conducted in other subdomains.

Function modelling, data modelling, organization modelling, and modelling of the operational information technology landscape are required to provide a complete picture of a business process.

The functional model investigates the units of work that are being enacted in the context of business processes. The specification of these functions can be done at different aggregation levels and could be informal, using English text, or formal, using syntactic or semantic specifications of functions.

The investigation and representation of data in business processes is of key importance, as decisions made during processes depend on them. Also data dependencies between activities need to be taken into account in process design.

The proper representation of the organizational structure of a company is an important requirement. Activities in the business process can then be associated with particular roles or departments in the organization. Many activities in a business process are performed by or with the assistance of information systems.

The operational information technology landscape, i.e., the information systems, their relationships, and their programming interfaces, needs to be represented to use the functionality provided by the information systems.

As already said, process modelling defines the glue between the subdomains. A process model relates functions of a business process with execution constraints, so that, for instance, the ordering and conditional execution of functions can be specified.

When a business process is started, the business functions that it contains need to be executed. Therefore, each activity in a business process requires an implementation.

A functional decomposition of coarse-grained business functions to fine-granular activities defines the functional perspective of a business process.

Process Models and Process Instances

Business processes consist of a set of related activities whose coordinated execution contributes to the realization of a business function.

Business processes are represented by business process models.

Process Instances reflect the actual occurrences of a business process. Each process instance is an instance of a process model, which is itself also described by process metamodels.

In order to express process models, there needs to be a notation that provides notational elements for the conceptual elements of process metamodels.

Any modelling effort starts with identifying the main concepts that need to be represented. In metamodelling, the concepts to be represented are models.

- Process Model: A process model represents a blue print for a set of process instances with a similar structure. Process models have a two-level hierarchy, so that each process model consists of a set of activity models. Each process model consists of nodes and directed edges.
- Edge: Directed edges are used to express the relationships between nodes in a process model.
- Node: A node in a process model can represent an activity model, an event model, or a gateway model.
    - Activity Models: describe units of work conducted in a business process. Each activity model can appear at most once in a process model. No activity model can appear in multiple process models.
    - Event Models: capture the occurrence of states relevant for a business process. Process instances start and end with events.
    - Gateway Models: Gateways are used to express control flow constructs

Each process model consists of nodes and edges. The nodes represent activity models, event models and gateway models, while the edges represent control flow between nodes. Each edge is associated with exactly two nodes, relating them in a particular order. Each node is associated with at least one edge.

Activity models reflect the work units to be performed, event models represent the occurrence of states relevant for the business, and gateway models represent execution constraints of activities.

Each process starts with exactly one event, the initial event, and ends with exactly one event, the final event.

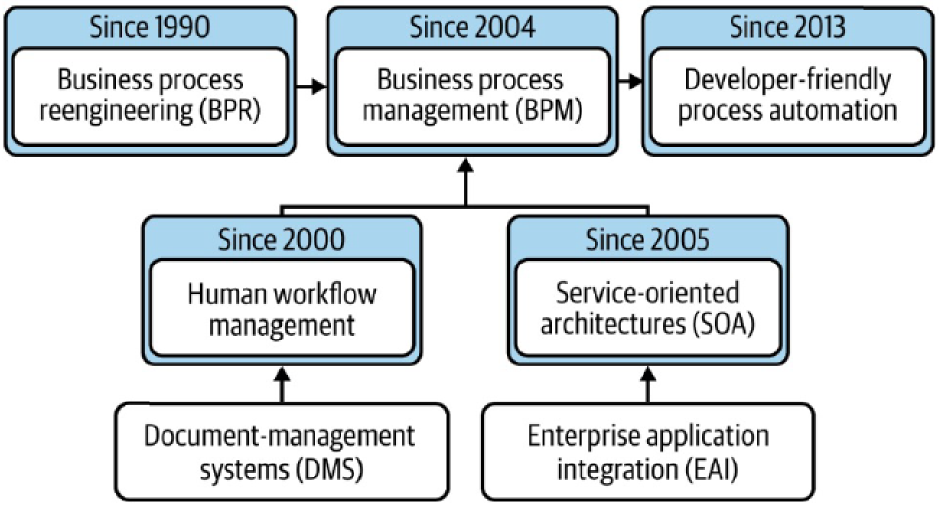
Process models define restrictions on process instances that belong to the process model. In fact, a process instance consists of a number of activity instances as well as event and gateway instances. There are one-to-many relationships between the processes at the model level and at the instance level. Each process instance is associated with exactly one process model, and each process model is associated with more instances.

The language used to model Business processes is **BPMN**.

5. Process Execution

The main goal of organizations has always been the process automation.

Business Process Management has evolved along two main branches: the Human workflow management, focusing on human activities to create solutions to reduce the paper, and the Service-oriented architecture (SOA), with focus on the control flow in order to coordinate heterogeneous systems belonging to the same or different companies.


Since 1990
Business process reengineering (BPR)
Since 2004
Business process management (BPM)
Since 2013
Developer-friendly process automation
Since 2000
Human workflow management
Since 2005
Service-oriented architectures (SOA)
Document-management systems (DMS)
Enterprise application integration (EAI)

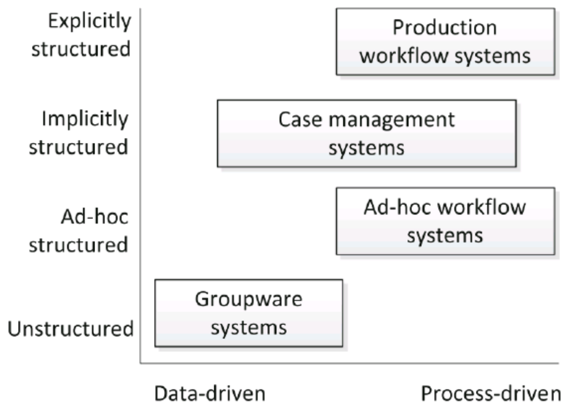
In order to automate processes, a fundamental component of information systems is the **Process-Aware Information System** (PAIS). It relies on the knowledge about the processes running in the organization to automate them, with concern on the automatic execution of tasks, in which users are not involved, and on the possibility for the user to provide inputs or consume outputs managed by it.

There are two categories of Process-Aware Information Systems:

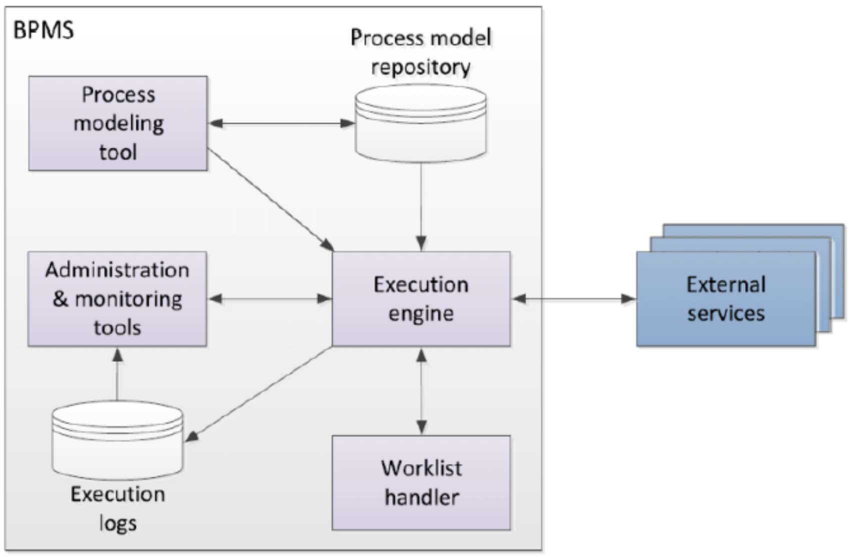
- <u>Domain-specific PAIS</u>: inside the system there is a process logic specific for a domain used to communicate with other modules.
- <u>Domain-agnostic PAIS</u>: tools that can be placed in different companies and fed by models, without caring about the domain, and able to orchestrate everything. They are more flexible, but have less efficiency.

The proper IT support for the business process enactment is always chosen in the configuration phase.

A BPMS is a system that supports the design, analysis, execution, and monitoring of business process on the basis of explicit models. It is a standard type driven by the language specification: for BPMS the language is BPMN. BPMSs can be classified according to the degree of support and the orientation.



The following image shows the architecture of a BPMS, with its main components:



The **execution engine** (or workflow engine or process engine) is the core of the systems, tasked to create process instances (or cases) and to distribute work among the participants. It also manages data and updates logs.

The abstract and executable models that are utilized and analyzed are created using a **process modeling tool**, which allows to store and retrieve models and to deploy the executable ones to the execution engine.

The **worklist handler** connects the process participants (the users) and offers them work items. It control the life cycle of the work and the statuses of execution.

Administration and modeling tools are used to define users and roles along with their permissions and provide the possibility of log analysis.

An important contribution is given by **external services**, which perform automated activities exposing web services or service components to the system that might utilize them.

Here are the main advantages and challenges of a BPMS:

Advantages	Challenges
• **Workload reduction** o **Transporting work** o **Coordination** o **Gathering information** • **Flexible integration** o **Allows to "extract" business logic and reuse it** o **Process models can be modified with limited impact on the called systems** • **Execution transparency** o **Collects operational (running cases) and historical information** o **Provides insights on what happened or is happening** • **Rule enforcement** o **Rules are first-class citizens**	• Technical challenges o Need to talk with closed systems o Systems are rarely process-aware • Organizational challenges o Risk for a Fordism approach o Impressions to be constantly monitored

Executable business process model

To produce an executable business process, a good starting point could be an abstract BPMN.

An **abstract/conceptual model** is made by domain experts and intended for their use. Its goal is to be human readable and it often messes technical details or contains ambiguities, making it unsuitable for execution.

That's why the need for an **executable model**, made by IT experts to be machine readable and focused on technical details, containing no ambiguities.

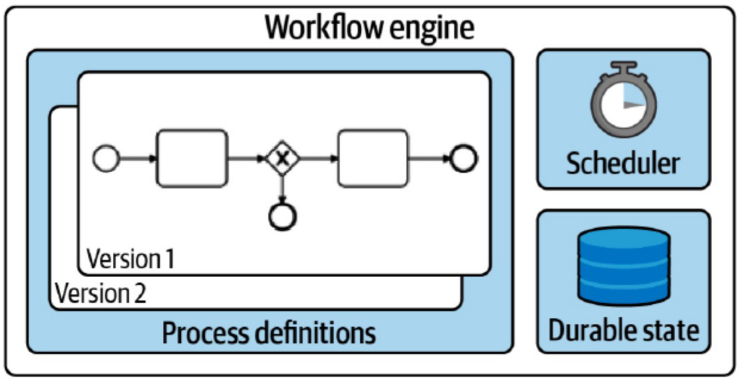
There is a method to create and executable model starting from an abstract one, composed of steps:

1. Identify the automation boundaries: there are some portion of the process that cannot be managed by the BPMS, identified by looking at the types of tasks that can be automated, a user task or a manual task.
2. Review manual tasks: manual tasks cannot be coordinated by a BPMS, so in order to make them visible to the BPMS a new user task or automated task representing them needs to be in the process.
3. Complete the process model: be sure that the modeler specifies all the relevant elements. Typical elements that are not included in the model as exceptions, as modelers usually focus on the "happy" path.
4. Bring the process model to an adequate granularity level: there is not necessarily a one to one mapping between the tasks in a business oriented model and those in the corresponding executable model. So it is appropriate to aggregate two or more consecutive tasks assigned to the same resource and split tasks if they require different resources.
5. Specify execution properties: the implementation details of a business process are called execution properties and refer to process variables, messages, errors, code snippets, rules assignment, sequence flow expressions, forms and connectors, etc. These properties do not have a graphical representation in BPMN, but are stored in the BPMN 2 0 interchange format, which is a textual representation of a BPMN model in XML format, it supports the interchange of BPMN models between tools and also to serve as input to a BPMN execution engine.

The execution (workflow) engine

The execution engine is a state machine with managing and scheduling tasks. It has three main elements:

- Durable state: states of the machine keeping track of all running process instances.
- Scheduling: keeps track of timing
- Versioning: with the pool of processes definition, containing the processes, manages the update of processes' versions and the instances running when there is an update.

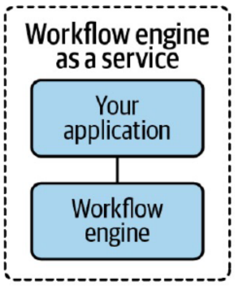


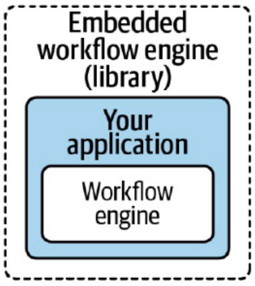
BPMS and the application and the subsystem

The **application** is the software which offers to the user an interface to perform the different stages. A BPMS coordinates activities performed with the usage of different applications.

There are two main approaches to this:

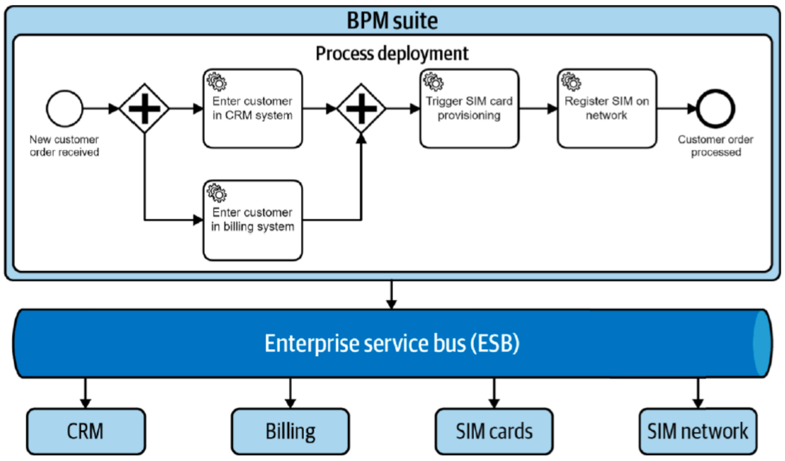
- Workflow engine as a service: provides flexibility and modules are interchangeable, but there are issues about communication between modules and performance.
- Embedded workflow engine: often associated to a monolithic solution, there is less flexibility but the performance is generally higher. One constraint is that the whole application must be in the same programming language of the workflow engine.

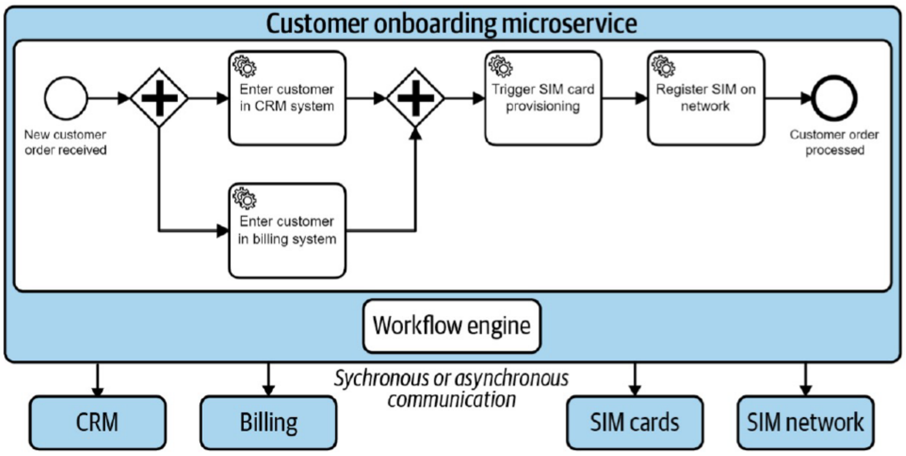




There are different solutions regarding the BPMS and the subsystems.

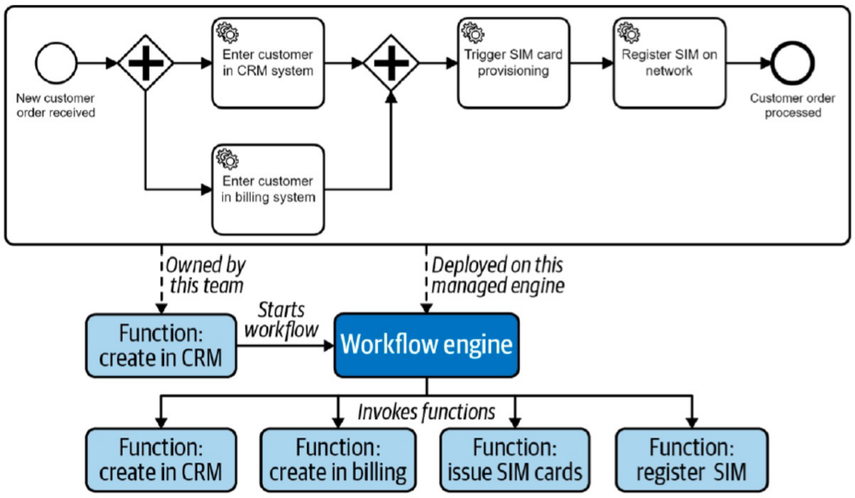
The historical solution is the **Software oriented architecture** (SOA). It is based on a centralized element in which a common interface is offered to the services, the Enterprise service bus. This component requires high maintenance but is the one allowing to make calls to the right module when needed.

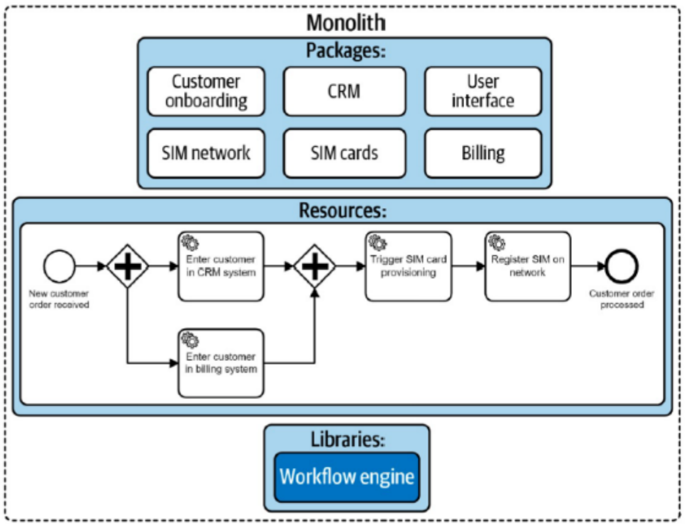

BPM suite
Process deployment
New customer order received
Enter customer in CRM system
Enter customer in billing system
Trigger SIM card provisioning
Register SIM on network
Customer order processed
Enterprise service bus (ESB)
CRM
Billing
SIM cards
SIM network


Customer onboarding microservice
New customer order received
Enter customer in CRM system
Enter customer in billing system
Trigger SIM card provisioning
Register SIM on network
Customer order processed
Workflow engine
Sychronous or asynchronous communication
CRM
Billing
SIM cards
SIM network

The **micro-services architecture** is composed of a micro-service for the workflow engine and many functions (CRM, billing, etc.) at its disposal. Usually, there is a server specialized for each function.

In a **server-less architecture**, an extremization of the concept happens: single small functions are offered to the engine to call them. This way, the application is a lot distributed. This architecture is an immediately deployable solution, which comes in handy to face downtime problems.


New customer order received
Enter customer in CRM system
Enter customer in billing system
Trigger SIM card provisioning
Register SIM on network
Customer order processed
Owned by this team
Deployed on this managed engine
Starts workflow
Function: create in CRM
Workflow engine
Invokes functions
Function: create in CRM
Function: create in billing
Function: issue SIM cards
Function: register SIM



The **monolith architecture** is about efficiency, with its constraints and limits in flexibility.

6. Choreography

Business processes reside in a single organization. Since enterprises cooperate with each other, it is essential to consider the interaction between them. This interaction can be described by the interaction of business processes of these enterprises.

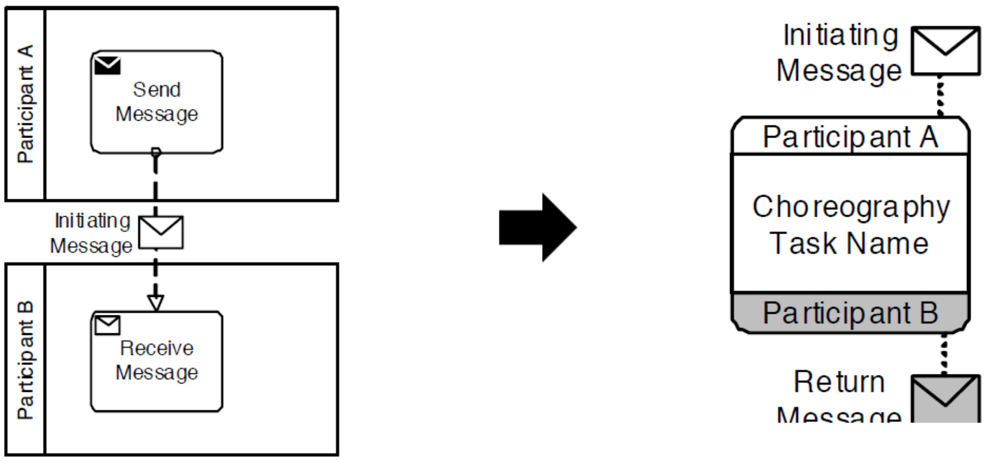
These interactions typically occur in a peer-to-peer style, following an agreed-upon process choreography.

Interacting process instances can be visualized adequately by collaboration diagrams, in which participants communicate only by sending and receiving messages. Process choreography provides a high level perspective focused only on the relationships among the organizations.

In details, a **Choreography diagram** defines the sequence of interaction among the participants. In it, all the organizations have the same importance, as the choreography does not take the standpoint of anyone.

In a way, the choreography defines the public protocol for the organizations to follow in order to complete their business processes.

Also for choreographies, BPMN supports the definition of process choreographies with a specific set of elements.



7. Process Verification

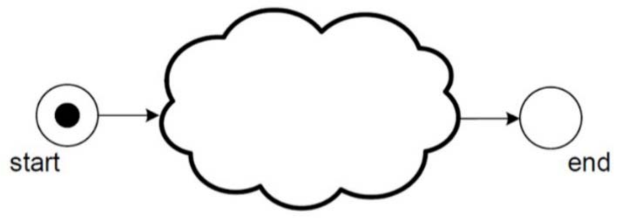
During the Analysis of a business process, the verification of it happens. To be a god process, the process needs to eventually terminate, and when it does, no activities should be still running. The capability of a process to come to a conclusion in an autonomous way is verified by verifying the so-called *soundness* of the process.

The process analysis should focus on the semantic validation of the process (also at design time), to ensure the process behaves as expected. To be semantically correct, the process model needs to be checked for the absence of behavioral anomalies.

To analyze processes, they can be translated and graphically represented by petri nets.

Workflow nets are a class of Petri Nets where there is:

- Only one initial place
- Only one final place
- None of the nodes are not in the path from the initial place to the final place.



AWF-net is a Petri net with a dedicated source place where the process starts and a dedicated sink place where the process ends. Moreover, all nodes are on a path from source to sink.

A business process that is represented by a Petri Net with the Workflow Net property, is also structurally sound.

In fact, a Business Process is **structurally sound** if:

- There is exactly one initial node (the only node without any incoming edges)
- There is exactly one final node (the only node without any outgoing edges)
- Each node in the process model is on a path from the initial node to the final node

Structural soundness is one of the many properties that are verifiable for a process to ensure its quality. In fact, structural soundness is usually not enough: not every WF-Net represents a correct business process. In a Workflow Net, there can be deadlocks, never activated activities, livelocks, etc.

Other properties that can be verified are, apart from structural soundness: Soundness, Weak soundness, lazy soundness. The strictest of these is soundness, but if it is not possible to asses that a process is sound then weak soundness or lazy soundness can be checked. Another property is the *relaxed soundness*, which informally means that at least one path exists in the petri net so that it satisfies the soundness.

Structural soundness is a necessary condition to verify soundness.

A WF-Net, corresponding to a business process, is **sound** if and only if:

"for any case, the procedure will terminate eventually, and at the moment the procedure terminates there is a token in place 'o' and all the other places are empty"

from W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, Application and Theory of Petri Nets 1997, LNCS1248, pages 407–426. Springer-Verlag, 1997.

A little bit ore formally a WF net is sound iif:

- For every state M reachable from state i there exists a firing sequence σ so that $M\ [\sigma\rangle\ o$
- State o is the only state reachable from state i with at least one token in place o
- There are no dead transitions in the workflow net in state i

In order to verify the soundness of a Workflow net, reachability graphs can be used.

Van der Aalst proves that a WF-net is **sound** iif for the extended WF-Net the following properties hold:

- Liveness
- Safeness (1-boundness)

The extended WF-Net is obtained adding to the WF-net an additional transition which has o (final place) as its input place and i (initial place) as its output place.

Some tools exist to automatically perform a semantical analysis of petri nets and check whether they are structurally sound or sound. One example of such tools is *WoPed*.

8. Service Oriented Architecture (SOA)

*"**Service Oriented Architecture** is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains"*

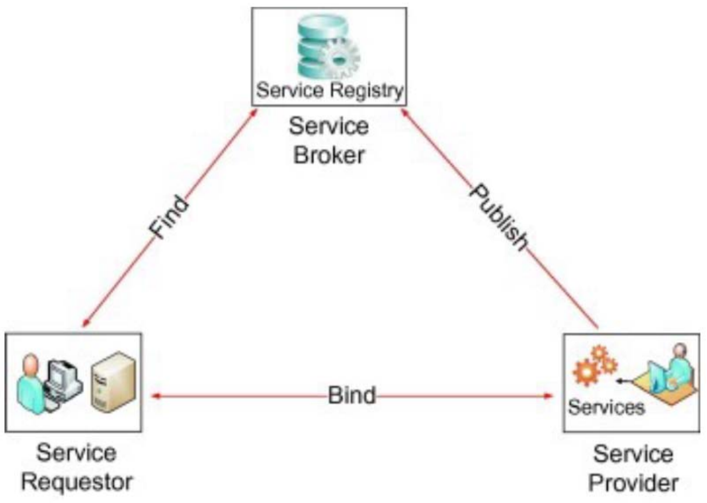
C. M. MacKenzie et al, 2006

SOA is a paradigm, intended to be a distributed paradigm, thus the connection among systems is mandatory. SOA allows to model services provided even by different owners.

There are different actors playing in the SOA context:

- Service Provider: publishes services' interfaces and descriptions and provides the service through them.
- Service Broker: collects the published services and makes them visible to the users searching for services.
- Service Requestor: searches for services and finds them thanks to service brokers, then Binds with the provider to utilize the needed service.

In this way, the consumer lifecycle is completely separated from the provider lifecycle.



Some useful definitions:

The **Common Object Request Broker Architecture** (CORBA) is a standard defined by the Object Management Group (OMG) designed to facilitate the communication of systems that are deployed on diverse platforms. CORBA enables collaboration between systems on different operating systems, programming languages, and computing hardware. CORBA is now made obsolete by SOA paradigms.

An interface description language or **interface definition language** (IDL), is a generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language. IDLs describe an interface in a language-independent way, enabling communication between software components that do not share one language.

The **Web Services Description Language** (WSDL) is an XML-based interface description language that is used for describing the functionality offered by a web service.

Universal Description, Discovery, and Integration (UDDI) is an XML-based registry for business internet services. A provider can explicitly register a service with a Web Services Registry such as UDDI or publish additional documents intended to facilitate discovery such as Web Services Inspection Language (WSIL) documents.

Service-oriented architecture is a way of designing, developing, deploying, and managing systems, in which:

- Services provide reusable business functionality via well-defined interfaces
- Service consumers are built using functionality from available services, and possibly by a composition of them
- There is clear separation between service interface and service implementation. In fact, consumers can see only the interface, while the implementation is visible only to providers
- A SOA infrastructure enables discovery, composition, and invocation of services

Traditional approaches to problems in enterprises usually result in the development of multiple siloed applications, that at the end of the day implement the same functionalities multiple times wherever they are needed along with the others. The governance of the application portfolio could be difficult to manage because of the different IT requirements and because of the challenge of integrating different systems.

With SOA, functionalities and capabilities of the system are encapsulated and reused for different purposes and contexts. So, services must be designed to support not only a specific need, but also to be reusable in different context. Moreover, services are considered as autonomous entity, that possibly can be composed with other services to provide a new service.

When switching from a traditional to a SOA design approach, enterprises can follow two methods:

- Top-down: services can be pulled from the IT infrastructure as a way to re-organize the existing applications.
- Bottom-up: services can be created from scratch as new business requirements arise.

Services could be designed and developed for **internal users**, which means that they are meant for other components of the same enterprise as internal end users. So, the audience of the services is in this case well known.

Otherwise, services could be designed and implemented for **external users**. In this case, it is possible to talk about *services as a product*, where services are usually exposed as API. In this case, only the audience model is known.

Service-orientation design principles

There are 8 design principles to respect in service orientation.

First, a **Standardized Service Contract**. It is a contract that includes both functional and non-functional aspects of the service. Functional aspects are usually described via API: WSDL for Web Services (SOAP), and OpenAPI (Swagger) for REST Services. Non-functional aspects are not so standardized so far: WS-Agreement, although limited, is one of the most adopted standards. Penalties and responsibilities are also included in the contract.

From a technical perspective, contracts are exposed as service descriptions with machine readable and understandable languages.

A service contract (comprised of a technical interface or one or more service description documents) is provided with the service and is standardized through the application of design standards.

The second principle is **Service Loose Coupling**. Lifecycle of a service should be as much as possible independent from its consumer, with the only constraint that the service provider must not violate the contract. Implementation and adopted technology (how a service fulfills the contract) must be independent.
So, the contract itself is ideally decoupled from technology and implementation details. There might be minimal consumer coupling requirements.

Third principle is the **Service Abstraction**, meaning that the service contract should contain only the minimal set of information to allow the consumer to invoke the services and all the information about how the service is implemented must be hidden to the customer.

Service Reusability is the principle ensuring that a service should not be designed only to solve a specific need. As the need for a service may be driven by contingency, the design of a service should be as generic as possible in order to facilitate its reuse in different contexts. Having a reusable service may result in a tangible value for the business of the enterprise (API economy).

The logic encapsulated by the service should be agnostic of the usage scenario, and sufficiently generic to facilitate numerous different usages by different types of service consumers. Services are designed to facilitate simultaneous access by multiple consumer programs operating in different contexts.

Another property is the **Service Autonomy**. Service should support the self-* properties: self healing, self configuring, self optimizing, self protecting. A proper execution environment (SOA infrastructure) is required to enable these properties. In fact, services have a contract that expresses a well-defined functional boundary that should not overlap with other services, and service instances are hosted by an environment that accommodates high concurrency for scalability purposes.

Service Statelessness states that a service should not retain any state, to increase the autonomy and the loose coupling. State must be stored by the consumer and not by the provider. This also increase the scalability and flexibility of the resulting system. A service needs to have a highly business process-agnostic logic so that the service is not designed to retain state information for any specific parent business process.

In addition to the contract, services should be accompanied with metadata to increase their **Service Discoverability** and further describe their purposes and capabilities to humans. The visibility of a service could be inside and/or outside an enterprise.

A service can be designed as a composition of other services, and this property is the **Service Composability**. Composition could be driven by a business process.

SOA benefits

The benefits of the SOA paradigm can be listed:

- Increase consistency in how functionalities and data is represented
- Reduced dependencies between units of solution logic
- Reduce awareness of underlying solution logic design and implementation details
- Increased opportunities to use a piece of solution logic for multiple purposes
- Increased opportunities to combine units of solution logic into different configurations
- Increased behavioral predictability
- Increased availability and scalability
- Increased awareness of available solution logic

These benefits are given by standardization, loose coupling, reusability and availability and scalability inside SOA.

Inside enterprises, applications are built as a composition of services, so the same service can be used in different applications and applications can be process-driven (I can reuse the applications according to a process logic).

The advantage about integration is that service-oriented applications are intrinsically interoperable.

We can say that Service Oriented Architecture (SOA) is a technology architecture enabling the service-orientation principles based on four main characteristics:

- Business driven: continuous alignment between the technology and the business needs
- Vendor neutral: vendors do not influence the architecture
- Enterprise centric: service is an enterprise resource that can be reused
- Composition centric: service can live alone or inside one or more (business process driven) compositions

There are different types of SOA:

- Service Architecture: a single service architecture
- Service Composition Architecture: set of composed services
- Service Inventory Architecture: support for managing the collection of available services
- Service-oriented Enterprise Architecture: services at enterprise level

A SOA infrastructure could support one or more of these architectures.

9. SOAP and REST services

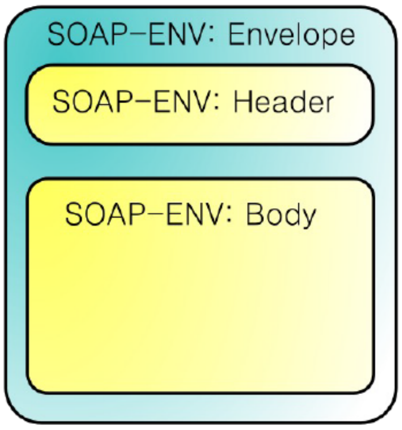
SOAP Services

SOAP is a protocol, based on RPC, that defines the structure of message to be exchanged over the web. It enables for interoperability among a wide range of programs, on a wide range of platforms. The main idea behind designing SOAP was to ensure that programs built on different platforms and programming languages could exchange data in an easy manner.

SOAP makes use of existing technologies wherever possible:

- HTTP and SMTP as transport protocols
- XML as interaction encoding scheme

SOAP services are described using Web Services Description Language (WSDL). WSDL is used to describe what a service provides (the methods) and is independent of any programming language.



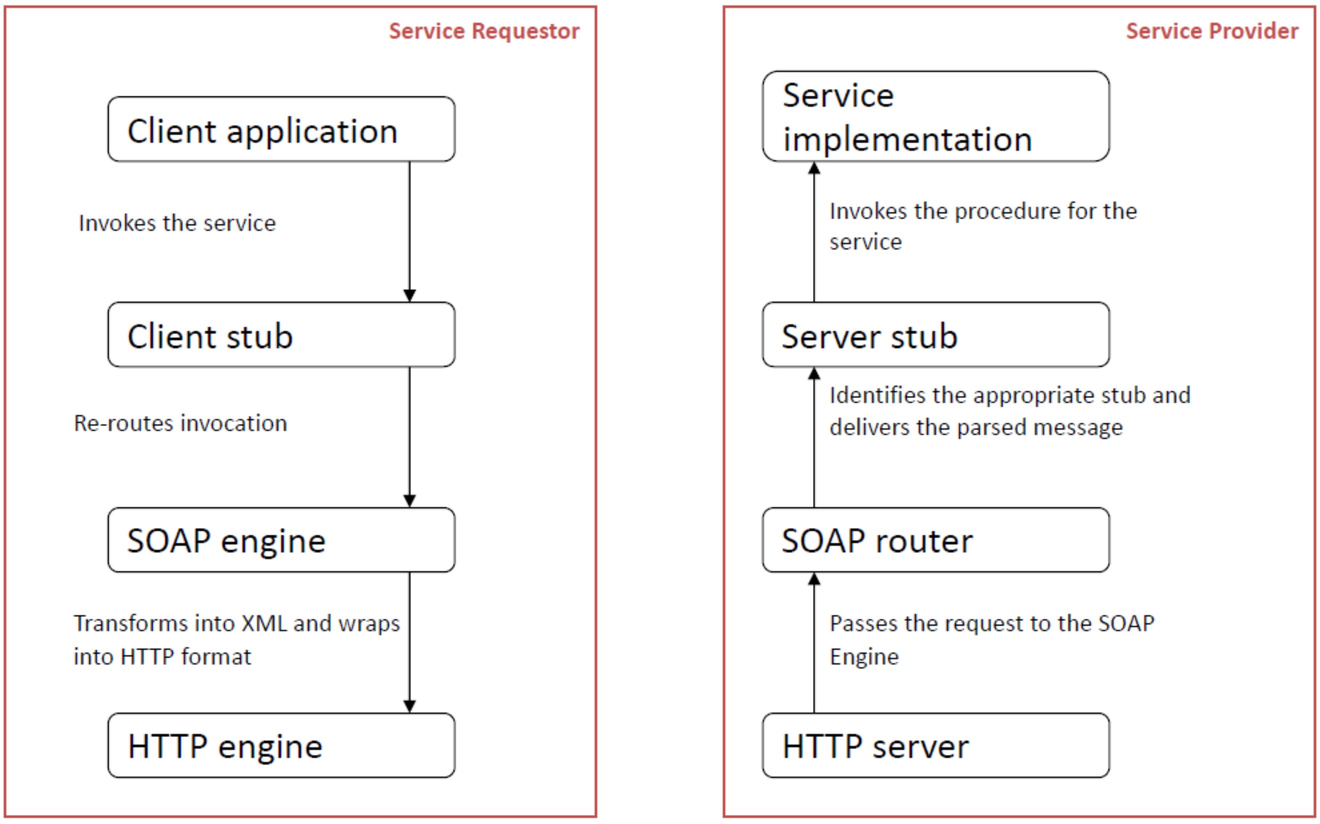
The structure of SOAPO messages is composed by the Envelope, the Header, and the body.

Header is optional and contains a description of the content, for example payload length, encryption, constraints, etc. So, it contains meta-data about the message and also allows to extend the SOAP messages. It usually concerns security mechanisms, transactional properties or routing.

The body contains the payload, which can be transmitted on different protocols: HTTP, SMTP, UDP, JMS. The content depends on the nature of the service, only exception being that SOAP dictates a specific format only in case of faults.

```
<soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <getProductDetails xmlns="http://magazzino.example.com/ws">
         <productId>827635</productId>
      </getProductDetails>
   </soap:Body>
</soap:Envelope>
```

The general implementation of SOAP is shown in the next image:


Service Requestor
Client application
Invokes the service
Client stub
Re-routes invocation
SOAP engine
Transforms into XML and wraps into HTTP format
HTTP engine
Service Provider
Service implementation
Invokes the procedure for the service
Server stub
Identifies the appropriate stub and delivers the parsed message
SOAP router
Passes the request to the SOAP Engine
HTTP server

Web Service Description

A **web service description** is a document by which the service provider communicates the specifications for starting the web service to the service requester.

Web service descriptions are expressed in the XML application known as **Web Service Description Language** (WSDL). In addition to WSDL, some extensions about semantic, quality of service, and other stuff, might be used, such as OWL-S or SA-WSD.

The service description contains information about the service interface, reachability, information model, functionalities, contract and policy, service behavior, and many other specifications.

WSDL is independent of the programming language utilized and the implementation of the service.

In any case, the description and all the specifications are XML-based. The aim is also to automate the discovery, engagement, negotiation, and execution of a service.

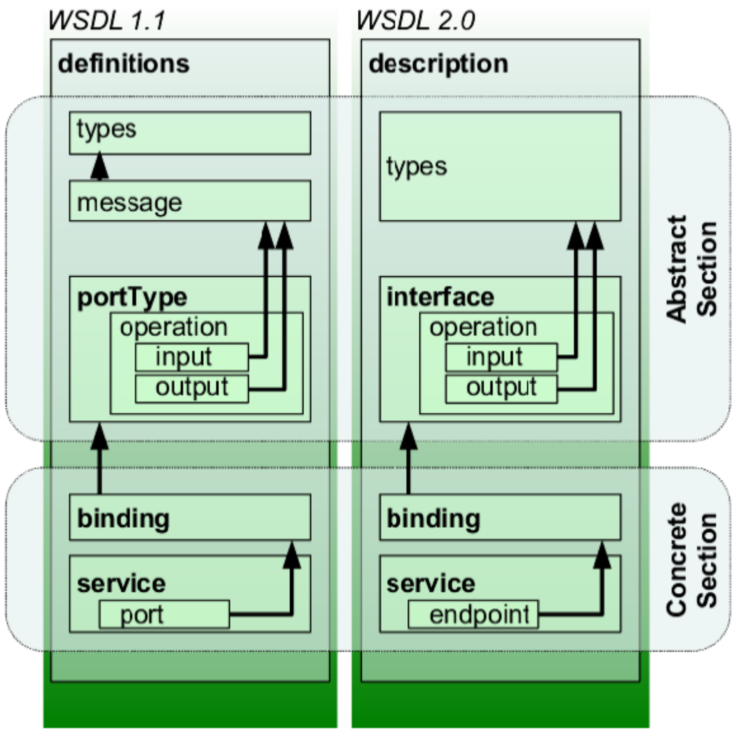
WSDL 2.0 is one of the versions available and utilized. It's always a XML-based specification used to describe a service programmable interface, describing what a service provides (WSDL includes the specification of the public state) while leaving out how a service works (WSDL does not include the private state).

The main elements of WSDL 2.0 are 6 and provide the description of the service:

- Type: data types relevant for the service (information model), it defines data types used in the rest of the document.
- Operation: basic functionalities
- Interface: group of operations (functionality) along with the messages. It is an abstract definition of the service, it logically groups the <operation> and defines the functionalities that a service offers. The same service can have more than one <interfaces>, although it's usually better to have just one.
- Binding: association between portType and port, binding of the functionality offered by the interface. It Offers the <operation> included in an interface through a specific protocol.
- Port: operations offered through a specific protocol (reachable functionality)
- Service: identify a service (service interface), it represent the service at a whole, including all the <endpoint> available.

An interface contains also the message exchange pattern, which Defines how the service interacts with the client when offering a functionality. There are four possible patterns:

- Request-Response (input-output): client sends a request to server and server responds
- Solicit-Response(output-input): client receives a solicitation from server and responds
- One-way (input)
- Notification (output)



REST Services

Representational state transfer, or **REST**, is an architectural style, which provides an approach to implement web service more scalable than RPC-SOAP, being REST resource oriented instead of method oriented. It takes inspiration from the Web (HTTP) which is flexible and

scalable by definition, but also defines some constraints over HTTP.

REST has some constraints:

- Client-Server: as the basic architectural style enabling the decoupling between components
- Uniform interface: common contract between client and server
- Stateless: the request contains all the data needed for the processing at server side
- Cacheable: response could be stored and used for equal upcoming requests to improve the efficiency
- Layered System: client is agnostic of the complexity at server side
- Code on Demand (optional): server can transmit code to the client to be executed at client-side

REST architectural style can be applied on HTTP. In this case these are the main concepts:

- Resource: abstraction of an entity, what is actually exchanged between the client and the server
- Representation: concrete entity encoded via specific grammars (e.g. JSON, XML)
- Methods: how to work with resources (GET, PUT, POST, DELETE, PATCH, HEAD)
- URI: Uniform Resource Identifier, to uniquely identify a resource

What is peculiar about REST is the distinction between the entity and the resource. In general, the client requests a resource using a URI and the server responds with a representation of the resource.

To obtain a uniform interface and better describe our API and related resources, the approach is the following:

- Distinguish between resource from representation
- Few and well-defined operations allowed on the resources
- Messages in addition to the resource info also includes all the information to proper manage the resource (HATEOAS) to reduce the a-priori knowledge

A resource represents a data structure, is uniquely identified by a URI and can be related to other resources. Every resource is only accessible through methods. There are different archetypes of resources:

- Document (singular): represents an object instance or a database record, it is composed by fields and/or link to other resources.
- Collection: a server-managed directory of resources. Usually, a client may propose to add new resources to a collection and the server mediates and decides if it can be stored.
- Store: a client-managed resource repository. The client completely manages the resource, without any middleware to mediate between the client and the storage.
- Procedure (Controller or Action): required when a resource can reach the same state in different ways and it is needed to keep track of this. In this case, CRUD operations (create, read, update, and delete) are not enough.

```
{
  "_type": "vm",
  "name": "A virtual machine",
  "memory": 1024,
  "cpu": {
    "cores": 4,
    "speed": 3600
  },
  "boot": {
    "devices":["cdrom","harddisk"]
  }
}
```

The **REST API** is nothing more than a set of URIs. It can be considered as a root resource, which does not contain any application-specific data model, but contains all the other resources which map the application-specific data model. In addition, it also contains metadata such as, for example, information on the API version.

The methods used with a REST API are:

Method	Description
GET (idempotent)	HTTP request method used to retrieve a representation of a resource's state.
PUT (idempotent)	HTTP request method used to insert a new resource into a store or update a mutable resource.
PATCH (idempotent)	HTTP request method used to partially update a mutable resource.
POST (not idempotent)	HTTP request method used to create a new resource within a collection or execute a controller.
DELETE (idempotent)	HTTP request method used to remove its parent.
HEAD (idempotent)	HTTP request method used to retrieve the metadata associated with the resource's state.
OPTIONS (idempotent)	HTTP request method used to retrieve metadata that describes a resource's available interactions.

The Response Status Code Categories are the following:

Category	Description
1xx: Informational	Communicates transfer protocol-level information.
2xx: Success	Indicates that the client's request was accepted successfully.
3xx: Redirection	Indicates that the client must take some additional action in order to complete their request.
4xx: Client Error	This category of error status codes points the finger at clients.
5xx: Server Error	The server takes responsibility for these error status codes.

HATEOAS (Hypermedia as the Engine of Application State) is a constraint of the REST application architecture that keeps the RESTful style architecture unique from most other network application architectures. The term "hypermedia" refers to any content that contains links to other forms of media such as images, movies, and text.
REST architectural style lets us use the hypermedia links in the response contents. It allows the client can dynamically navigate to the appropriate resources by traversing the hypermedia links. Furthermore, API responses can be customized to return different links based on the user who is calling that method.

```
GET /account/12345 HTTP/1.1
HTTP/1.1 200 OK
<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">100.00</balance>
    <link rel="deposit" href="/account/12345/deposit" />
    <link rel="withdraw" href="/account/12345/withdraw" />
    <link rel="transfer" href="/account/12345/transfer" />
    <link rel="close" href="/account/12345/close" />
</account>
```

```
GET /account/12345 HTTP/1.1
HTTP/1.1 200 OK
<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">-25.00</balance>
    <link rel="deposit" href="/account/12345/deposit" />
</account>
```

REST vs SOAP

SOAP	REST
SOAP is a protocol. SOAP was designed with a specification. It includes a WSDL file which has the required information on what the web service does in addition to the location of the web service.	REST is an Architectural style in which a web service can only be treated as a RESTful service if it follows the constraints of being, Client Server, Stateless, Cacheable, Layered System, Uniform Interface
SOAP uses service interfaces to expose its functionality to client applications. In SOAP, the WSDL file provides the client with the necessary information which can be used to understand what services the web service can offer.	REST exposes a uniform interface in the form of URIs, which uniquely identify entities (resources). Then the only methods available are based on HTTP.
SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot (it includes not only the body of the request).	REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON (or XML) messages.
SOAP can only work with XML format. As seen from SOAP messages, all data passed is in XML format.	REST permits different data format such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON.

	SOAP	REST
Design	Standardized protocol with pre-defined rules to follow.	Architectural style with loose guidelines and recommendations.
Approach	Function-driven (data available as services, e.g.: “getUser”)	Data-driven (data available as resources, e.g. “user”).
Statefulness	Stateless by default, but it’s possible to make a SOAP API stateful.	Stateless (no server-side sessions).
Caching	API calls cannot be cached.	API calls can be cached.
Performance	Requires more bandwidth and computing power.	Requires fewer resources.
Message format	Only XML.	Plain text, HTML, XML, JSON, YAML, and others.
Advantages	High security, standardized, extensibility.	Scalability, better performance, browser-friendliness, flexibility.
Disadvantages	Poorer performance, more complexity, less flexibility.	Less security, not suitable for distributed environments.