# Process and Service Design

Professor: Pierluigi Plebani

Author: Simone Staffa

Politecnico di Milano
Computer Science & Engineering — Fall 2019-2020

# Contents

# 1 Introduction

The meaning of term **service** is intuitive but is definition can be different in different contexts like:

- Economic
- ICT

We firstly need to understand similarities and differences among these different terms:

- Service
- e-Service
- Web-Service

Definition of service (economic perspective):
*"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)*


Figure 1: Looking for a dress, in different eras

The above picture represents the evolution of the services along the years. Specially, it refers to the dress service. In the Pre Industrial era was more like a personal service, where a tailors offered totally customized dresses to people directly in their houses. Then with the Industrial Era, dresses were sold by retailer, so the final product was not really customized by the consumers, who just had to select the already made dress they liked. Finally, nowadays in the Information Era, service started to be offered through IT Platforms, with full customization without even

the need to go to the shop, dresses can be just bought and received at home.
This can be also applied to the IT domain, for what we call nowadays Web Services.
The Pre Industrial Era corresponds to when people used to build their own computers, buying different parts and installing them by their selves. The Industrial Era corresponds to when companies started building personal computers to be sold to consumers, who just had to pay and bring them home. Finally, the Information Era corresponds to nowadays Cloud Platform and Web Services that allow people to rent virtual computers online, to perform their own computation.

Just to recap:

- **Personal service (Pre Industrial era)**: highly and man-made customization but limited production
- **Industrial service (Industrial era)**: mass production but no customization
- **Electronic service (Information era)**: highly and computer-based customization combined with mass production

This to say that we are **evolving to a service-oriented society**.
*"We don't need a drill, we need a hole in the wall."*

From an economic perspective, we are switching from a Good-dominant logic to a Service-dominant logic.

- **Good-dominant logic** focuses on goods exchange, that's why goods are considered to be tangible and each of them has an embedded value that is taken into account to operate exchange transactions.
- **Service-dominant logic** focuses on service provision, where resources are intangible and require a co-creation of value.

S-D logic is useful because it opens to more interactions with customers and allows to focus on why a product fits to the customer needs instead of technical specification.

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
  - SOA (Service Oriented Architecture) becomes the new way of developing software to be service-ready
- As the automation of economics activities and self-service
  - **ICT is a mean** and can improve the effectiveness and the efficacy
  - The goal is to make the business able to provide services

As a mean, ICT has a fundamental role in the evolution towards the S-D logic. Who deliver products must pay a lot of attention on Customization, Scalability, Reliability and Security/Privacy. As a goal, ICT is also a service to be delivered and a possible way to deliver these service could be using a Web Service.

# 2 Service Definition

Definition of service (IT perspective):
*"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." (P. Hill, On goods and services)*

## 2.1 Software Oriented Architecture

**Service-Oriented Architecture (SOA)** is a style of software design where services are provided 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, in one of two ways: through passing data or through two or more services coordinating an activity.

### 2.1.1 Reference Model

SOA in the end is a reference model. A reference model is an abstract **framework** for understanding the main elements characterizing an architecture. It enables the development of specific reference or concrete architectures using consistent standards or specifications supporting that environment.

A reference model consists of a minimal set of unifying concepts, axioms, policies and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or the concrete details.
In the SOA context, the first two elements that we need to define are: service providers (who create capabilities to support a solution) and service consumers (who is facing a problem in the course of their business).

Following discussion on SOA RM is taken from [1]

### 2.1.2 SOA Reference Model

The goal of this reference model is to define the essence of service oriented architecture, and emerge with a vocabulary and a common understanding of SOA.

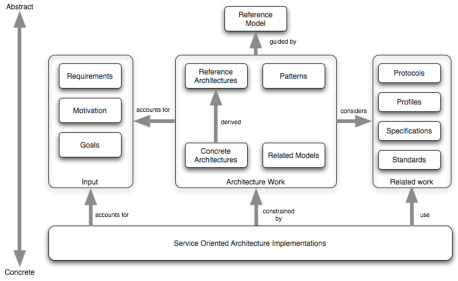



Figure 2: How the Referenced Model relates to other work

Figure 2 shows how a reference model for SOA relates to other distributed systems architectural inputs. The concepts and relationships defined by the reference model are intended to be the basis for describing references architectures and patterns that will define more specific categories of SOA designs. Concrete architectures arise from a combination of reference architectures, architectural patterns and additional requirements, including those imposed by technology environments.

### 2.1.3 What is Service Oriented Architecture

Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.
In general, entities (people and organizations) create capabilities to solve or support a solution for the problems they face in the course of their business. It is natural to think of one person's needs being met by capabilities offered by someone else; or, in the world of distributed computing, one computer agent's requirements being met by a computer agent belonging to a different owner.
The perceived value of SOA is that it provides a powerful framework for matching needs and capabilities and for combining capabilities to address those needs.

Visibility, interaction, and effect are key concepts for describing the SOA paradigm. **Visibility** refers to the capacity for those with needs and those with capabilities to be able to see each other. This is typically done by providing descriptions.
**Interaction** is the activity of using a capability. Typically mediated by the exchange of messages, an interaction proceeds through a series of information exchanges and invoked actions. This permits service providers and consumers to interact and provides a decision point for any policies and contracts that may be in force.

The purpose of using a capability is to realize one or more **real world effects**. At its core, an interaction is "an act" as opposed to "an object" and the result of an interaction is an effect. This effect may be the return of information or the change in the state of entities (known or unknown) that are involved in the interaction.

We are careful to distinguish between *public* actions and *private* actions; private actions are inherently unknowable by other parties. On the other hand, public actions result in changes to the *state* that is shared between at least those involved in the current execution context and possibly shared by others. Real world effects are, then, couched in terms of changes to this **shared state**.

This description of SOA has yet to mention what is usually considered the central concept: the **service**. The noun "service" is defined in dictionaries as "The performance of work (a function) by one for another".
While both needs and capabilities exist independently of SOA, **in SOA services are the mechanism by which needs and capabilities are brought together.**

SOA is a means of organizing solutions that promotes reuse, growth and interoperability. It is an organizing and delivery paradigm that enables one to get more value from use both of capabilities which are locally "owned" and those under the control of others.

The concepts of visibility, interaction, and effect apply directly to services in the same manner as these were described for the general SOA paradigm. Visibility is promoted through the

service description which contains the information necessary to interact with the service and describes this in such terms as the service inputs, outputs, and associated semantics. The service description also conveys what is accomplished when the service is invoked and the conditions for using the service.

In general, entities (people and organizations) offer capabilities and act as service providers. Those with needs who make use of services are referred to as service consumers.

### 2.1.4 Benefits of SOA

The value of SOA is that it provides a simple scalable paradigm for organizing large networks of systems that require interoperability to realize the value inherent in the individual components. Indeed, SOA is scalable because it makes the fewest possible assumptions about the network and also minimizes any trust assumptions that are often implicitly made in smaller scale systems.

Through this inherent ability to scale and evolve, SOA enables an IT portfolio which is also adaptable to the varied needs of a specific problem domain or process architecture. The infrastructure SOA encourages is also more agile and responsive than one built on an exponential number of pair-wise interfaces. Therefore, SOA can also provide a solid foundation for business agility and adaptability.

## 2.2 The Reference Model

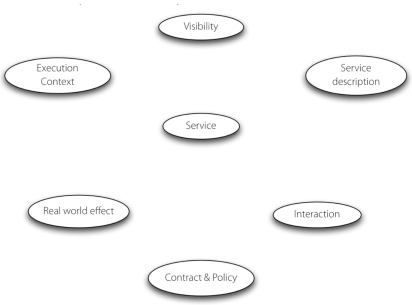



Figure 3: Principal concepts in the Reference Model

### 2.2.1 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.

A service is opaque in that its implementation is typically hidden from the service consumer except for (1) the information and behavior models exposed through the service interface and

(2) the information required by service consumers to determine whether a given service is appropriate for their needs.

The consequence of invoking a service is a realization of one or more real world effects. These effects may include:

1. Information returned in response to a request for that information
2. A change to the shared state of defined entities
3. A combination of (1) and (2)

### 2.2.2 Dynamics of Services

From a dynamic perspective, there are three fundamental concepts that are important in understanding what is involved in interacting with services: the visibility between service providers and consumers, the interaction between them, and the real world effect of interacting with a service.

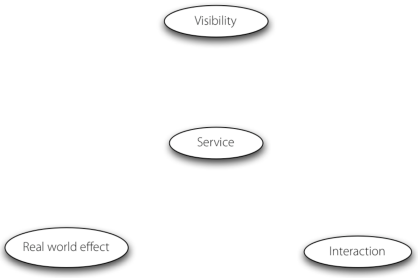



Figure 4: Concepts around the dynamics of a service

**Visibility**

For a service provider and consumer to interact with each other they have to be able to "see" each other. This is true for any consumer/provider relationship. In the case of SOA, visibility needs to be emphasized because it is not necessarily obvious how service participants can 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.

- **Awareness**: both the service provider and the service consumer MUST have information that would lead them to know of the other's existence. For a service consumer to discover a a service, the service provider must be capable of making details (service description and policies) of the service available to potential consumers.
- **Willingness**: it is an intentional act to initiate and to participate in a service interaction. For example, the willingness for a service provider to interact with other consumers can be expressed in the action of publishing itself on a service registry/repository, where consumers can easily find them.

- **Reachability**: the relationship between service participants where they are able to interact, possibly by exchanging information. Participants MUST be able to communicate with each other. A service consumer may have the intention of interacting with a service, however, if the service is not reachable, the service would not be visible by the consumer.

Service description is fundamental for the visibility concept. It allows service to expose its details to consumers, such as:

- Functional aspects: what the service is providing
- Technical requirements: how the service is provided
- Constraints: e.g., only authenticated people can connect
- Policies: e.g., 3 times/day usage cap
- Quality of Service: e.q., promise a level of availability

**Interaction**

Interacting with a service involves performing actions against the service. In many cases, this is accomplished by sending and receiving messages, but there are other modes possible that do not involve explicit message transmission. For example, a service interaction may be effected by modifying the state of a shared resource. However, for simplicity, we often refer to message exchange as the primary mode of interaction with a service.

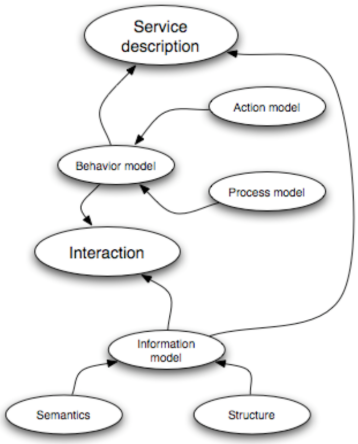



Figure 5: Service Interaction concepts

Figure 5 illustrates the key concepts that are important in understanding what it is involved in interacting with services; these revolve around the service description – which references a information model and a behavior model.

- **Information model**: a characterization of the information that may be exchanged with the service. Only information and data that are potentially exchanged with a service are generally included within that service's information model. The scope of the information model includes the format of information that is exchanged, the structural relationships within the exchanged information and also the definition of terms used. Knowing the

representation, structure, and form of information required is a key initial step in ensuring effective interactions with a service. There are several levels of such structural information; including the encoding of character data, the format of the data and the structural data types associated with elements of the information.

- **Behavior model**: knowledge of the actions invoked against the service and the process or temporal aspects of interacting with the service. This is characterized as knowledge of the actions on, responses to, and temporal dependencies between actions on the service. For example, in a security-controlled access to a database, the actions available to a service consumer include presenting credentials, requesting database updates and reading results of queries. The action model of a service is the characterization of the actions that may be invoked against the service. The process model characterizes the temporal relationships and temporal properties of actions and events associated with interacting with the service.

**Real World Effect**

There is always a particular purpose associated with interacting with a service. Conversely, a service provider (and consumer) often has a priori conditions that apply to its interactions. The service consumer is trying to achieve some result by using the service, as is the service provider. At first sight, such a goal can often be expressed as "trying to get the service to do something". This is sometimes known as the "real world effect" of using a service. For example, an airline reservation service can be used to learn about available flights, seating and ultimately to book travel – the desired real world effect being information and a seat on the right flight.
As previously stated, a real world effect can be the response to a request for information or the change in the state of some defined entities shared by the service participants. In this context, the shared state does not necessarily refer to specific state variables being saved in physical storage but rather represents shared information about the affected entities.

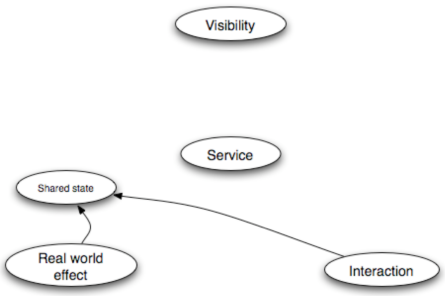



Figure 6: Real World Effect and shared state

In addition, the internal actions that service providers and consumers perform as a result of participation in service interactions are, by definition, private and fundamentally unknowable. By unknowable we mean both that external parties cannot see others' private actions and, furthermore, SHOULD NOT have explicit knowledge of them. Instead we focus on the set of facts shared by the parties – the shared state. Actions by service providers and consumers lead to modifications of this shared state; and the real world effect of a service interaction is the accumulation of the changes in the shared state.

For example, when an airline has confirmed a seat for a passenger on a flight this represents

a fact that both the airline and the passenger share – it is part of their shared state. Thus the real world effect of booking the flight is the modification of this shared state – the creation of the fact of the booking. Flowing from the shared facts, the passenger, the airline, and interested third parties may make inferences – for example, when the passenger arrives at the airport the airline confirms the booking and permits the passenger onto the airplane (subject of course to the passenger meeting the other requirements for traveling).
For the airline to know that the seat is confirmed it will likely require some private action to record the reservation. However, a passenger should not have to know the details of the airline internal procedures. Likewise, the airline does not know if the reservation was made by the passenger or someone acting on the passenger's behalf. The passenger's and the airline's understanding of the reservation is independent of how the airline maintains its records or who initiated the action.

### 2.2.3 About services

In support of the dynamics of interacting with services are a set of concepts that are about services themselves. These are the service description, the execution context of the service and the contracts and policies that relate to services and service participants.


Figure 7: About services

**Service description**

The service description represents the information needed in order 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. While there are certain elements that are likely to be part of any service description, most notably the information model, many elements such as function and policy may vary. The purpose of description is to facilitate interaction and visibility, particularly when the participants are in different ownership domains, between participants in service interactions. By providing descriptions, it makes it possible for potential participants to construct systems that use services and even offer compatible services. Best practice suggests that the service description SHOULD be represented using a standard, referenceable format.

The service description makes available critical information that a consumer needs in order to decide whether or not to use a service. In particular, a service consumer needs to possess the following items of information:

1. That the service exists and is **reachable**
2. That the service performs a certain function or set of functions

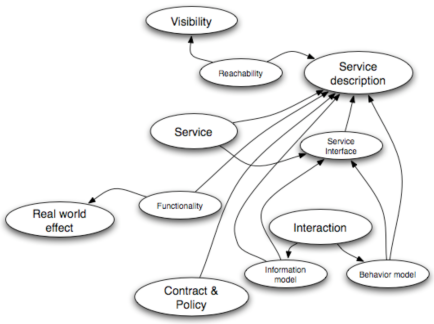



Figure 8: Service description

3. That the service operates under a specified set of constraints and policies
4. That the service will (to some implicit or explicit extent) comply with policies as prescribed by the service consumer
5. How to interact with the service in order to achieve the required objectives, including the format and content of information exchanged between the service and the consumer and the sequences of information exchange that may be expected

**Service Reachability** Reachability is an inherently pairwise relationship between service providers and service consumers. However, a service description SHOULD include sufficient data to enable a service consumer and service provider to interact with each other. This MAY include metadata such as the location of the service and what information protocols it supports and requires. It MAY also include dynamic information about the service, such as whether it is currently available.

**Service Functionality** A service description SHOULD unambiguously express the function(s) of the service and the real world effects that result from it being invoked. The description of functionality may include, among other possibilities, a textual description intended for human consumption or identifiers or keywords referenced to specific machine-processable definitions. Part of the description of functionality may include underlying technical assumptions that determine the limits of functionality exposed by the service or of the underlying capability.

**Policies Related to a Service** A service description MAY include support for associating policies with a service and providing necessary information for prospective consumers to evaluate if a service will act in a manner consistent with the consumer's constraints.

**Service Interface** The service interface is the means for interacting with a service. It includes the specific protocols, commands, and information exchange by which actions are initiated that result in the real world effects as specified through the service functionality portion of the service description. The specifics of the interface SHOULD be syntactically represented in a standard referenceable format. These prescribe what information needs to be provided to the service in order to access its capabilities and interpret responses. This is often referred to as the service's information model. it is assumed that for a service to be usable, its interface MUST be represented in a format that allows interpretation of the interface information by its consumers.

**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. Like policies, agreements are also about the conditions of use of a service.

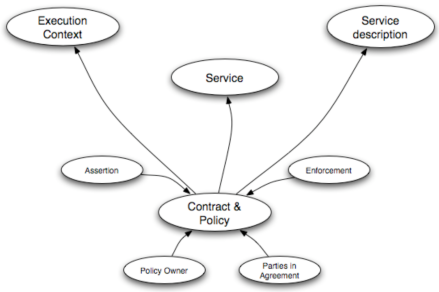



Figure 9: Policies and Contracts

**Service Policy** Conceptually, there are three aspects of policies: the policy assertion, the policy owner (sometimes referred to as the policy subject) and policy enforcement.

- **Policy Assertion**: for example, the assertion: "All messages are encrypted" is an assertion regarding the forms of messages. As an assertion, it is measurable: it may be true or false depending on whether the traffic is encrypted or not. Policy assertions are often about the way the service is realized; i.e., they are about the relationship between the service and its execution context.
- **Policy Enforcement**: techniques for the enforcement of policies depend on the nature of the policy. Conceptually, service policy enforcement amounts to ensuring that the policy assertion is consistent with the real world. This might mean preventing unauthorized actions to be performed or states to be entered into; it can also mean initiating compensatory actions when a policy violation has been detected.

Policies potentially apply to many aspects of SOA: security, privacy, manageability, Quality of Service and so on. Beyond such infrastructure-oriented policies, participants MAY also express business-oriented policies – such as hours of business, return policies and so on.

**Service Contract** Whereas a policy is associated with the point of view of individual participants, a contract represents an agreement between two or more participants. Like policies, contracts can cover a wide range of aspects of services: quality of service agreements, interface and choreography agreements and commercial agreements. Note that we are not necessarily referring to legal contracts here.

Thus, following the discussion above, a service contract is a measurable assertion that governs the requirements and expectations of two or more parties. Unlike policy enforcement, which is usually the responsibility of the policy owner, contract enforcement may involve resolving disputes between the parties to the contract. The resolution of such disputes may involve appeals to higher authorities.

**Execution Context**

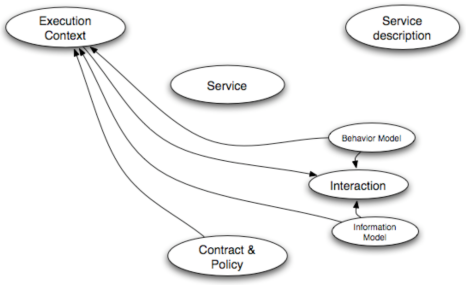



Figure 10: Execution Context

The execution context of a service interaction is 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.

As discussed in previous sections of this document, the service description (and a corresponding description associated with the service consumer and its needs) contains information that can include preferred protocols, semantics, policies and other conditions and assumptions that describe how a service can and may be used. The participants (providers, consumers, and any third parties as noted below) must agree and acknowledge a consistent set of agreements in order to have a successful service interaction, i.e. realizing the described real world effects. The execution context is the collection of this consistent set of agreements.

The consumer and provider can be envisioned as separate places on a map and, for a service to actually be invoked, a path must be established between those two places. This path is the execution context.

The execution context also allows us to distinguish services from one another. Different instances of the same service – denoting interactions between a given service provider and different service consumers for example – are distinguished by virtue of the fact that their execution contexts are different.

Finally, the execution context is also the context in which the interpretation of data that is exchanged takes place. A particular string has a particular meaning in a service interaction in a particular context – the execution context.

### 2.2.4 IBM SOA Reference Architecture

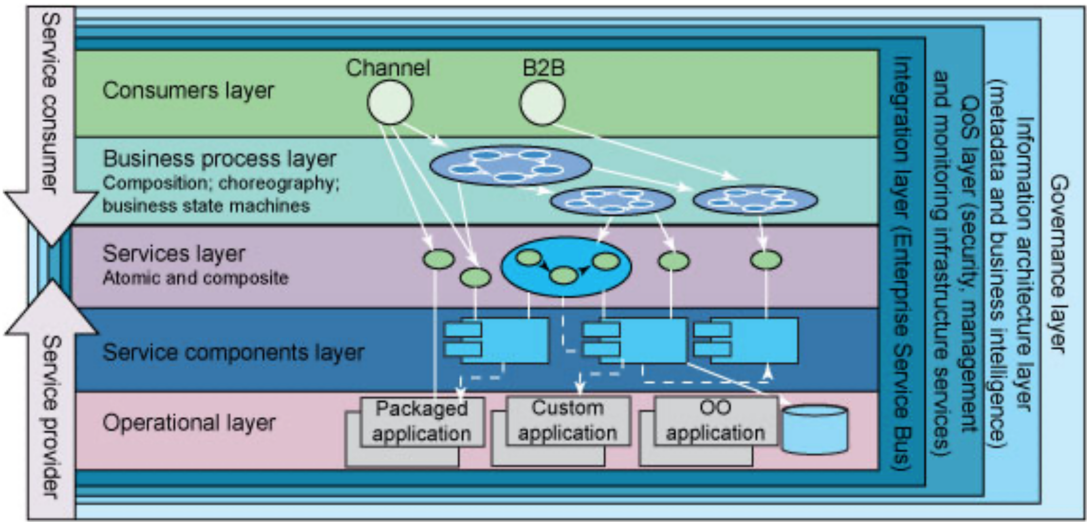



Figure 11: IBM SOA Reference Architecture

The IBM SOA Reference Architecture is used as reference for our course. It consists of different layers, with different levels of visibility, that can be used to describe a Service Oriented Architecture.

**Summary of the layers**

First, as we can see from Figure 11, there are some layers that are visible only to service providers and some visible only to service consumers which are Operational Layer + Service Components Layer and Consumers Layer + Business Process Layer respectively. Then in the middle we have the Service Layer which is the only one visible to both actors.
The layers that are defined in an SOA RA are as follows:

- **Operational Layer**: it contains all the applications and infrastructures already existing in our architecture. In this layer, those applications are exposed and running possibly through services. Here we are opening specific sections and views of our IT architecture to the above layer. (e.g., applications, databases, ERP, CRM)
- **Service Component Layer**: contains 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. Software components can be of different types. (e.g., microservices or monolith applications).
- **Service Layer**: 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**: 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. Data flow and control flow are used to enable interactions between services and business processes.
- **Consumer Layer**: this layer is the point where consumers, whether if 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 (client platforms and devices). This layer provides the capability to quickly create the front end of the business processes and composite applications in order to respond to changes in the marketplace.
- **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. It provides the means of ensuring that an SOA meets its requirements with respect to: monitoring, reliability, availability, manageability, transactionality, maintainability, scalability, security, safety, life cycle, etc.
- **Information Architecture Layer**: is a cross-cutting concern that is responsible for manifesting a unified representation of the information aspect of an organization as provided by its IT services, applications, and systems enabling business needs and processes and aligned with the business vocabulary. 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.

## 2.3 e-Service

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

- **Service provider**
- **Service consumer**

A service provider may become a service consumer and the 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)

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.

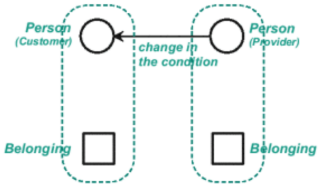
Service system can be more o r less complex involving a different number of actors, that's why governance of a service network is not an easy task.

### 2.3.1 Customers and providers interaction

There are different scenarios according to the involvement of persons and their belongings.

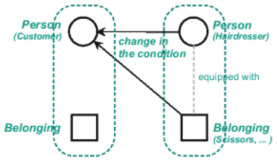
**Person acting on person**

A person delivers a service to another person and no belongings are involved on both sides. Note that this scenario is not interesting to us.



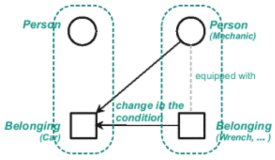

**Person with belonging on person**

A person delivers a service to another person (e.g., haircut, medical service, teaching). Providers belongings (tools needed to provide the service) are used to deliver the service.



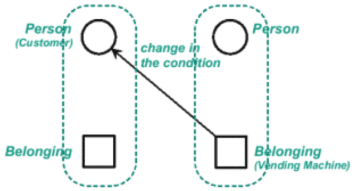

**Person with belonging acting on belonging**

A person delivers a service to another person changing the conditions of the customer belonging (e.g., car repair, transportation). Providers belongings (tools) are used to deliver the service. Customer belongings are managed by the provider.



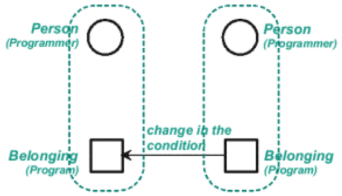
**Belonging acting on person**

Customer interacts with providers belonging to exploit a service (e.g., self-service). Providers belongings (tools) are used to deliver the service. Human intervention at provider side is limited. Customer needs to interact with machines (H2M).
Customer does not communicate with the service provider but he/she uses their belongings (e.g, Consumer uses the ATM without "seeing" the bank).




**Belonging acting on belonging**

Belongings are directly interacting (e.g., web services). Providers belongings (tools) are used to deliver the service. Human intervention at provider side is limited. Customer needs to be equipped with belonging able to communicate with the provider belonging (M2M). This is the case of a computer communicating with a web service.




### 2.3.2 Resource intensity of services

Resource intensity is a measure of the resources required for the provision of a unit of a good or service.

- **Laborand capital-intensive services**: labor costs outweigh the costs for equipment and materials (e.g., hospital)
- **Knowledge-intensive services**: heavily rely on professional knowledge (e.g., medical doctors)
- **Information-intensive services**: activities are mainly focused on data management (e.g., data analytics)
- **Technology-intensive services**: labor costs becomes negligible (e.g., web service, cloud services)

### 2.3.3 Technology in e-Services

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

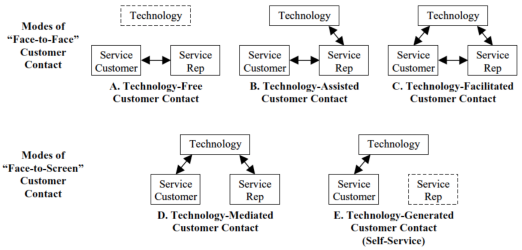



Figure 12: Possible roles of technology

**Face-to-face interaction**

Here services providers and service consumers are human beings. We distinguish:

- **Technology-assisted services (e.g., bank teller)**:
  - technology is used only by the service provider
  - service consumer does not have access to the technology as the service provider acts as a mediator
  - Example: a person (consumer) ask for an address to another person (provider) and he checks for the address on his phone (here the consumer is not allowed to see the mobile screen)
- **Technology-facilitated services (e.g., beamer)**:
  - service provider uses technology to improve the interaction with the service consumer
  - service consumer can have access to the technology together with the service provider
  - Example: students (consumers) in class can follow the lesson watching the beamer installed by the professor (provider). They can even talk to each other.

**Face-to-screen interaction**

Here technology is in the middle between providers and consumers (e.g., telephone,web). These below by definitions are e-Services.

- **Technology-mediate services**:
  - service providers and consumers are human beings and they are using the same technology to interact
  - e.g., Microsoft Teams, Amazon Mechanical Turk
- **Technology-generated services**:
  - service providers are replaced by technology
  - e.g., ATM, home banking

### 2.3.4 e-Services

*"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**

- **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 (ATM vs. booking)
- **Human touch**: customer experience is one of the main drawbacks of e-Services adoption, that's why technology is trying to reduce this gap through UX studies
- **Personalization**: one-to-one relationship is fundamental, private data need to be share

The e-Service provisioning is usually driven by a business process. 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 (more interesting for us).

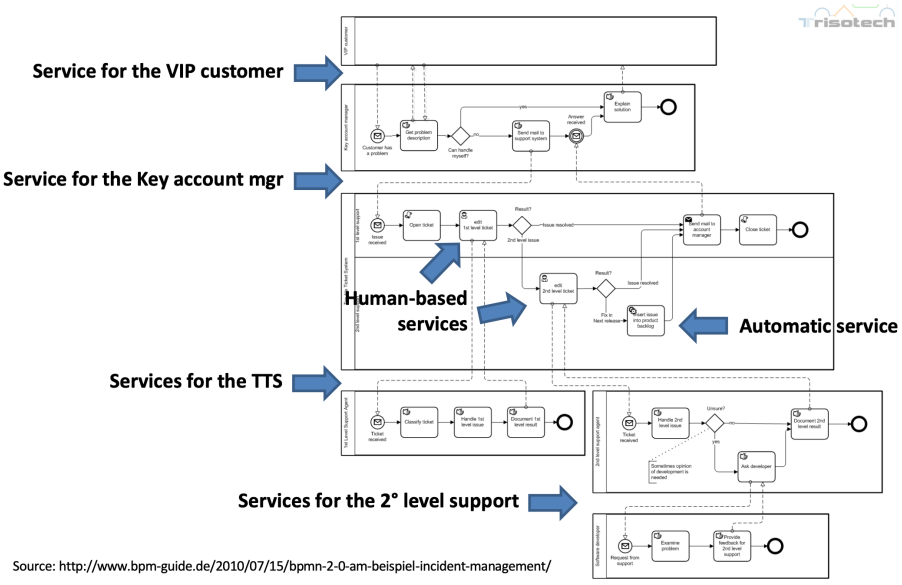



Figure 13: BPM Collaboration diagram

Figure 12 shows an example of BPM collaboration diagram in which different services and processes collaborate at different levels in order to provide a complex service.

# 3 Service Blueprint

## 3.1 Service design

Service design is design for experiences that reach people through many different touch points, and that happen over time. It is composed by a set of methods and principles coming from different disciplines which are systematically applied to the creation of new services (Interaction designers, Information science, Management science, ...)

### 3.1.1 Service design principles

- **Human-centered**: everything starts from a deep understanding of the needs of the users. Designing our services based upon how they are experienced from the 'customer's perspective.' Placing the user or consumer in the centre of the service allows us to discover how the patient experiences the service. We do this by implementing a range of techniques including interviews, observations and field research to gather insights into the consumer's needs, experiences and behaviours.

- **Co-designed**: service innovation is not possible without the acceptance of all stakeholders. All stakeholder groups should be involved in the service design process. Co-creation is the process of involving users not only in the design of the solution but also the production and development of it. Multidisciplinary teams from all levels within our organisation will allow for a range of expertise, knowledge and skills and will generate great ideas. Work out who is within the service 'ecosystem' and make sure every party is represented.

- **Sequenced**: services must be designed as dynamic processes that take place over time.

- **Holistic**: all the aspects of the service, both digital and physical should be considered. The entire environment in which the service exists and is delivered should be considered. Holistic services look at the whole user journey and consider each touch point of that service. Touchpoints refer to any interactions that occur between users and a service. A holistic approach can be achieved by using service blue prints, personas or fictional characters to highlight different user experiences and user journeys. A well-rounded and complete service also involves the functionality, safety and reliability of the service.

- **Evidenced**: visualize and prototype our service concept early and often. In a collaborative team environment, it is more expressive to use visual aids than to rely purely on words. Get the team to use sketches, pictures, graphs, maps and prototypes where applicable. Be creative with coloured pens and glue. Visual tools can be less complicated to digest and more tangible. They make things easier to remember and can help to properly explain what we're trying to achieve.

- **Agile**: Our process is incremental and iterative: start small, test early and learn quickly from failure and success. One of the main features of services design is not avoiding making mistakes but learning from them. This is achieved by prototyping and testing. You can save a large amount of time and money if we test the experience before spending lengthy periods of time developing it. It's imperative to prototype solutions before we launch them but it is also a good idea to release and test our iterations many times through the process. Don't just wait until the end of the project to do so.

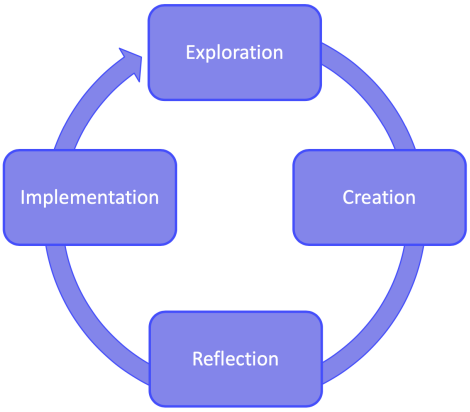



Figure 14: Service design process

Each of the phases in Figure 13 consists in different activities.

| Phase | Step | Objective | Typical activities | Methods (examples) |
|---|---|---|---|---|
| Exploration | Understand challenge | Understand and (re-)frame the challenge | Frame the actual challenge or problem<br>Identify impact factors and stakeholders | Design space map<br>Stakeholder map<br>Janus Cones, Context Map<br>Customer System |
| | Investigate humans | Understand the customers, further stakeholders and the service environment | Identify customers' needs, requirements, and demands<br>Identify touchpoints between provider and customer<br>Assess the overall experiences made by customers as well as their development over time<br>Assess the service from the perspective of the provider and other stakeholders<br>Analyse the environment<br>Identify initial options for improvement | Persona<br>A day in a life<br>Shadowing<br>Observe, interview, engage with customer<br>Empathy map<br>Customer journey maps |
| | Set focus | Synthesize the findings from the discovery phase in order to facilitate the subsequent creation phase | Aggregate and abstract the data points found Identify core findings and room of further action | |
| | Setup Project | Setup the project for success | Form the right teams<br>Create or tailor methods/tools | |

Figure 15: Exploration phase

| Phase | Step | Objective | Typical activities | Methods (examples) |
|---|---|---|---|---|
| Creation | Ideate | Enable radical and outside the box thinking & ideas | Conduct divergent activities oriented towards the far future<br>Open approaches: Involve employees and customer | Creativity methods<br>Value curve methods<br>Co-opetition model<br>Service theater (enacting) |
| | Conceptualize | Detail out a service concept from an experience perspective | Create low-resolution prototypes (often made of paper) to discuss functionality with customers / stakeholders rather than graphics | Mock-ups<br>Prototypes<br>Storyboards |
| | Assess & Select | Reduce the amount of ideas to follow up with | Compile alternative service episodes based on ideas and test them with customers and stakeholders | Story-telling<br>Request customer feedback |
| Reflection | | Explore feasibility and fit of the service | Compare to existing portfolios and strategies | SWOT analyses<br>Business cases<br>Customer acceptance tests |
| Implementation | | Detail out a service concept from an operational perspective | Organization-, process-, and (IT-)technology-specific competences | Service Blueprint<br>High-resolution prototypes<br>Graphical design<br>Business Model Canvas |

Figure 16: Creation, Reflection and Implementation

## 3.2 Service blueprinting

A **service blueprint** describes how a company delivers service to customers in order to improve customers' and employees' experience. It is also used to plan and organize resources: people, tools and processes.

**Service blueprinting** is Technique to help organizations understanding their service offering (or designing new services) and delivery processes by enabling to examine service processes from a customers point of view. The result is a graphical representation, i.e., map, of:

- The user journey (phase by phase, step by step)
- The touchpoints (channel by channel, touchpoint by touchpoint)
- The backstage processes (stakeholder by stakeholder, action by action)

The definition of a blueprint usually requires a team work.

Why do we need a service blueprint?
It helps changing perspective. The main question is not how a service is delivered (still relevant). **The main questions are what is delivered**. It is about framing problem before solving it. For example, Mc Donald, is an highly designed service:

- What you obtain
- The time in which you obtain goods
- The experience (what the employees say)

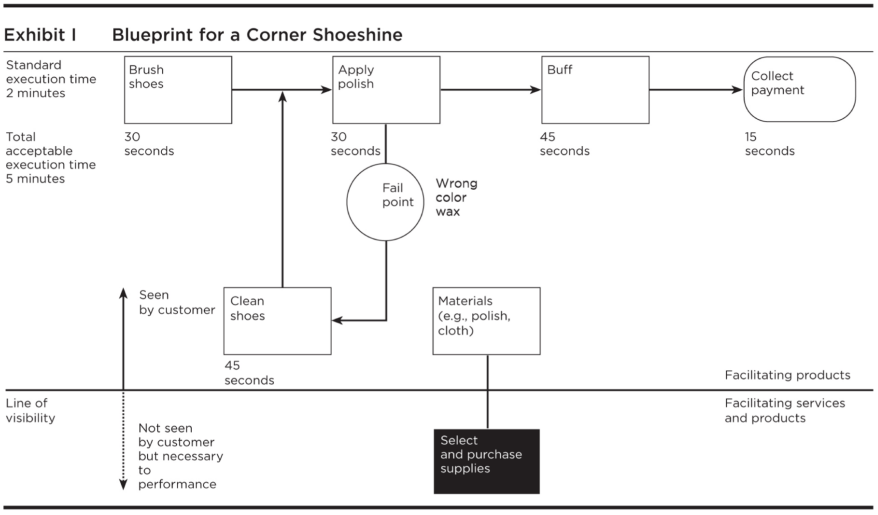



Figure 17: First blueprint example

*!Note that there exists no standard blueprint, every domain has its own.*

Figure 17 shows an example blueprint for a Corner Shoeshine. The line of visibility separates what is visible to the consumer (upper part) from what is visible only to the service provider.

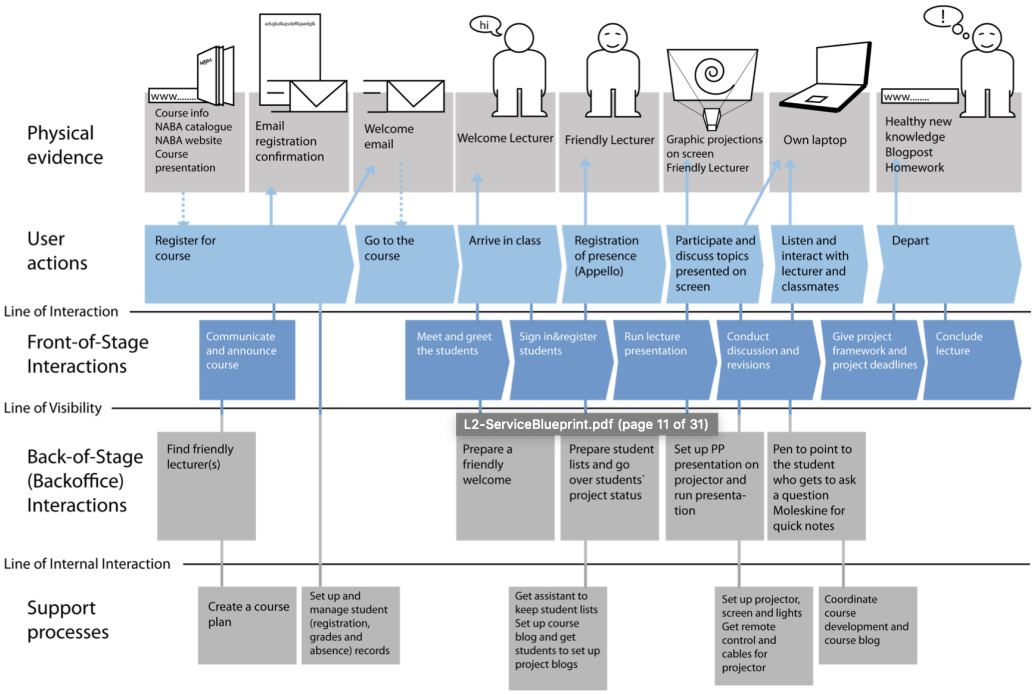



Figure 18: Second blueprint example

**Main concepts:**

- **Customer actions and Physical evidence**: include all of the steps that customers take as part of the service delivery process. Customer actions are depicted chronologically across the top of the blueprint. For each customer action, and every moment of truth, the physical evidence that customers come in contact with is described at the very top of the blueprint. These are all the tangibles that customers are exposed to that can influence their quality perceptions.
- **Frontstage**: both visible and invisible actions of frontline contact employees that occur as part of a face-to-face (social-media, e-mail, phone) or face-to-computer (website, chatbot) encounter. This stage is composed by touchpoints or channels that the company uses to interact with the users.
- **Backstage**: everything that appears above the line of visibility is seen by the customer, while everything below it is invisible. Below the line of visibility, all of the other contact employee actions are described, both those that involve nonvisible interaction with customers (e.g., telephone calls) as well as any other activities that contact employees do in order to prepare to serve customers or that are part of their role responsibilities.
- **Support process**: these are all of the activities carried out by individuals and units within the company who are not contact employees but that need to happen in order for the service to be delivered.

**Blueprint lines:**

- **Line of interaction**: is between customer actions and touch points and it is used to divide the user from service.
- **Line of visibility**: is between touch points and backstage and it defines what can be seen by the user.
- **Line of internal interaction**: defines the boundary between the company and third parties.

**Blueprint basic template**

Figure 19 includes the first and simplest blueprint template from where to start designing

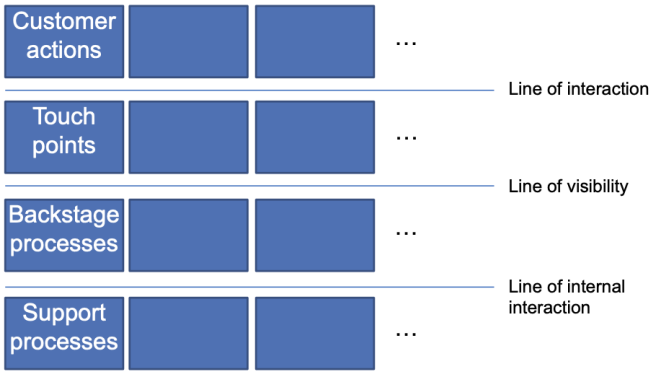



Figure 19: First blueprint template

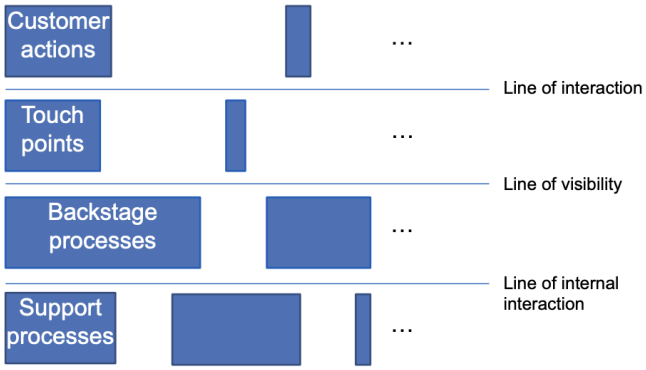



Figure 20: A longer square means a longer activity

a service. As we can see we have different squares indicating the activities that needs to be performed in each stage.
Figure 20 includes the same exact template, but with squares of different dimension, this to show that **time is central in a blueprint**, and it is represented with space. A longer square means a longer activity. More space between activity means longer waiting time.

### 3.2.1 Service blueprint creation process

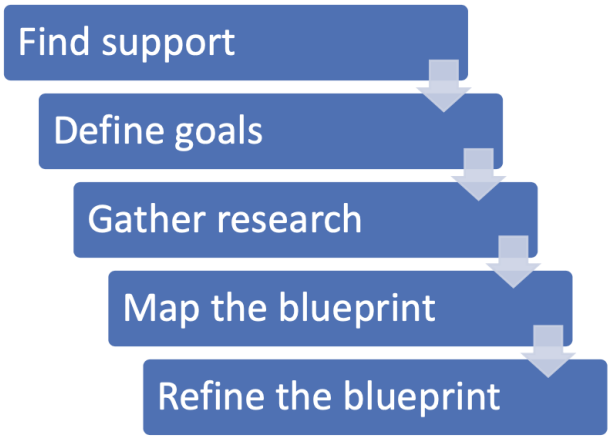



Figure 21: Service blueprint creation process.

1. **Find support:** blueprints require multidisciplinary skills/expertise. In this step we should identify required skills and build a team.
2. **Define goals**: defining goals is central to define the boundary of the analysis. In this step we should define the objective (service), the scope and the users (one service blueprint per user type!)
3. **Gather research**: gather knowledge that will be used to define the service blueprint. In this step we need to conduct customer research and interview employees running marketing analysis and questionnaires.
4. **Map the blueprint**: create a first draft of the blueprint and discuss it with the team. In this step we need to do some team work in order to define customer actions, touchpoints and the backstage.
5. **Refine the blueprint**: insert further information in the blueprint, and then add time metrics to it. Finally, refine the backstage and start again from step 3.

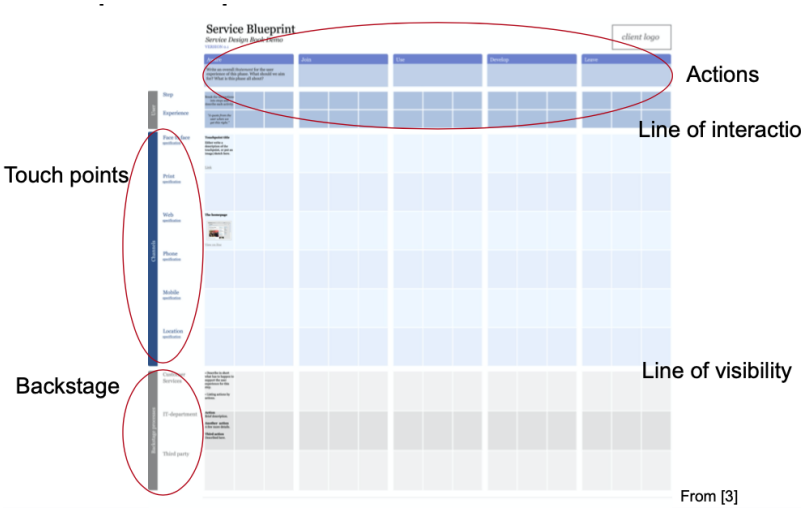



Figure 22: Example of blueprint.

In Figure 21 we have an example of Blueprint that is characterized by multiple **Phases (Actions)**:

- **Aware**: how the user first learn about the service
- **Join**: the sign-up/registration phase
- **Use**: a session of usage
- **Leave**: when user leaves the service forever

Then the **Touchpoints** (or channels) are identified on the left. As touchpoint we indicate how our service can be consumed. Notice, for a given service, a touchpoint can be used in different phases. A touchpoint can be a media channel (e.g., Website, e-mail) or a face-to-face interaction.

Finally, in the **Backstage** we identify which are the people (if any) involved in providing the service and connect them to the activities and touchpoints. We also need to identify possible external parties required for executing the process.

**Benefits**

- **Innovation**: allow to develop innovative services or improving existing ones
- **Quality**: graphical representation allows the designer to easily identify possible failure points
- **Training**: the blueprint can be used to train newcomers
- **Efficiency**: blueprint analysis allows the designer how to improve the process
- **Standardization**: it provides a common background in terms of semantic and also an information asset

# 4 Business Process Management

## 4.1 Introduction

Business process management is based on the observation that each product that a company provides to the market is the outcome of a number of activities performed. Business processes are the key instrument to organizing these activities and to improving the understanding of their interrelationships. Business processes are an important concept to facilitating this effective collaboration.

In many companies there is a gap between organizational business aspects and the information technology that is in place. Narrowing this gap between organization and technology is important, because in today's dynamic markets, companies are constantly forced to provide better and more specific products to their customers.

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.

### 4.1.1 Definitions

1. *"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)
2. *"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)

### 4.1.2 Main Elements

- **Organizations**: enterprise, non profit organizations...
- **Activities/Tasks**: single unit of work (as seen as by the process owner)
- **Coordination**: decision points, messages (inside/outside the organization)
- **Events**: things that happen atomically (no duration)
- **Outcome**: deliver values to the actors involved in the process

### 4.1.3 Classification of Business Process

**Organizational vs. Operational**

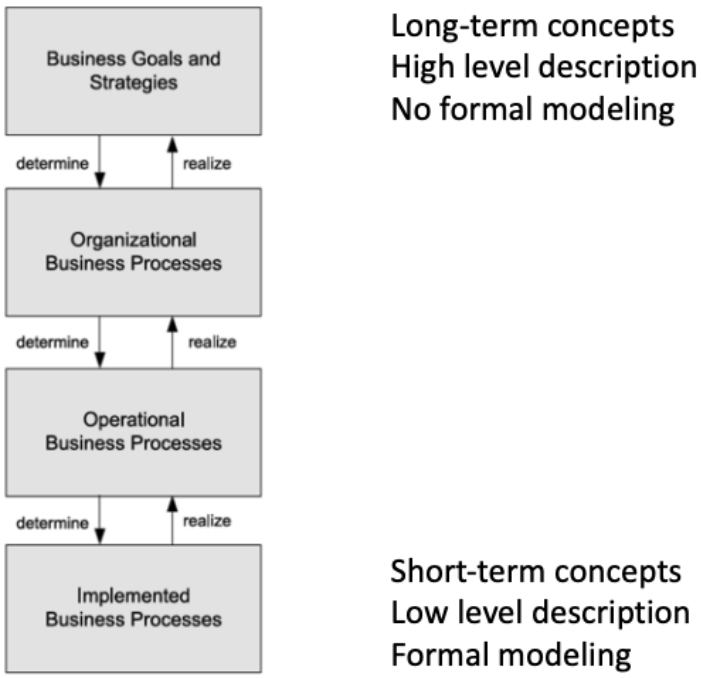



Figure 23: Organizational vs Operational business processes

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

1. At the highest level, the strategy of the company is specified, which describes its long-term concepts to develop a sustainable competitive advantage in the market.
2. At the second level, the business strategy is broken down to operational goals. These goals can be organized, so that each goal can be divided into a set of subgoals.
3. At the third level, organizational business processes can be found. 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. These business processes act as supplier or consumer processes.

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. Operational business processes are specified by business process models.

Operational business processes are the basis for developing implemented business processes. Implemented business processes contain information on the execution of the process activities and the technical and organizational environment in which they will be executed.

**Orchestration vs. Choreography**

**Service 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.

**Service 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.

The choreography describes the interactions between multiple services, where as orchestration represents control from one party's perspective. This means that a choreography differs from an orchestration with respect to where the logic that controls the interactions between the services involved should reside. In choreography there is not an orchestrator, but processes are coordinated by a previous agreement between all the parts.

**Degree of Automation**

Business processes can diverge in the level of automation. There are business processes that are fully automated, meaning that no human is involved in the enactment of such a business process. An example is ordering an airline ticket using Web interfaces. While the process is fully automated on the side of the airline, the customer is involved with manual activities, such as providing address information via Web browser interfaces.

**Degree of Repetition**

Business processes can be classified according to their degree of repetition. Examples of highly repetitive business processes include business processes without human involvement, such as online airline ticketing.

At the other end of the repetition continuum, there are business processes that occur a few times only. Examples include large engineering efforts, such as designing a vessel.

**Degree of structuring**

If the business process model prescribes the activities and their execution constraints in a complete fashion, then the process is structured. The different options for decisions that will be made during the enactment of the process have been defined at design time.

If process participants who have the experience and competence to decide on their working procedures perform business process activities, structured processes are more of an obstacle than an asset. Skipping certain process activities the knowledge worker does not require or executing steps concurrently that are ordered sequentially in the process model is not possible in structured business processes.
To better support knowledge workers, business process models can define processes in a less rigid manner, so that activities can be executed in any order or even multiple times until the

knowledge worker decides that the goals of these activities have been reached. So called ad *hoc activities* are an important concept for supporting unstructured parts of processes.

## 4.2 BPM

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

### 4.2.1 Why BPM?

Today:

- To understand what is happening in an organization
- To increase productivity and remove wase (Lean)
- To do the right things at the right time (Six Sigma)
- To increase the quality of an organization (TQM)

Tomorrow:

- To support not only operations but also managers
- To support "Industry 4.0"
- To exploit the Cloud (and Fog) Computing opportunities

### 4.2.2 Business Process Lifecycle

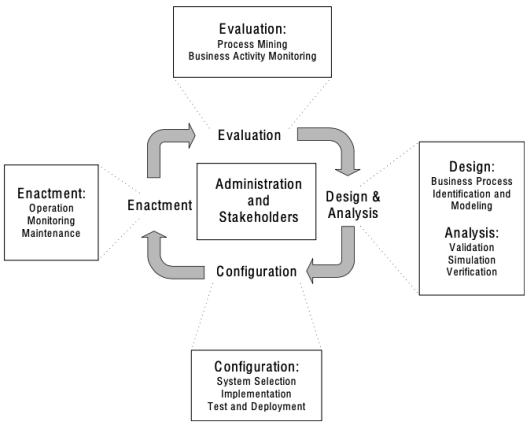



Figure 24: Business process lifecycle

The business process lifecycle is shown in Figure 24; it consists of phases that are related to each other. The phases 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.

**Design and Analysis**

The business process lifecycle is entered in the Design and Analysis phase, in which 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. Business process modelling is the core technical subphase during process design. Based on the survey and the findings of the business process improvement activities, the informal business process description is formalized using a particular business process modelling notation.

Simulation techniques can be used to support validation, because certain undesired execution sequences might be simulated that show deficits in the process model. Simulation of business processes also allows stakeholders to walk through the process in a step-by-step manner and to check whether the process actually exposes the desired behaviour.

The Design phase can execute following two approaches:

- Bottom-Up: interview people in the organization and abstract a business process.
- Top-Down: understand the business process starting from the process reference architecture.

The Design phase can be further divided in:

- Identification: understanding the process orientation
- Modeling: formalize what have been identified using some languages (BPMN)

The analysis phase can be further divided in:

- Validation: syntactical and soundness check
- Simulation: use a simulator to try different parameters and understand if the process is efficient for my resources
- Verification: verify if everything is behaving correctly

In the end of the Design & Analysis step we should not expect to have an error free process, but our aim is to eliminate a as much error as possible

**Configuration**

Once the business process model is designed and verified, the business process needs to be implemented. There are different ways to do so. It can be implemented by a set of policies and procedures that the employees of the enterprise need to comply with. In this case, a business process can be realized without any support by a dedicated business process management system.

In case a dedicated software system is used to realize the business process, an implementation platform is chosen during the configuration phase. 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 (very rarely): write legacy software
- Buy (more often): integrate existing software systems and transform the BPM into an executable model that can be understood by a BPMS (e.g., Camunda)

**Enactement**

Once the system configuration phase is completed, **business process instances can be enacted**. The process enactment phase encompasses the actual run time of the business process. Business process instances are initiated to fulfil the business goals of a company. Initiation of a process instance typically follows a defined event, for instance, the receipt of an order sent by a customer.

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.

A monitoring component of a business process management system visualizes the status of business process instances. Process monitoring is an important mechanism for providing accurate information on the status of business process instances.

**Evaluation**

The evaluation phase uses information available to evaluate and improve business process models and their implementations. Execution logs are evaluated using business activity monitoring and process mining techniques. These techniques aim at identifying the quality of business process models and the adequacy of the execution environment.

For instance, business activity monitoring might identify that a certain activity takes too long due to shortage of resources required to conduct it. Since this information is useful also for business process simulation, these phases are strongly related.

The goal of this step is to **analyse 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.

# 5 BPModeling

## 5.1 Conceptual Model and Terminology

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. An example of a manual activity is sending a parcel to a business partner.
- **User interaction activities** go a step further: these are activities that knowledge workers perform, using information systems. There is no physical activity involved. An example of a human interaction activity is entering data on an insurance claim in a call centre environment.
- **System activities** do not involve a human user; they are executed by information systems. An example of a system activity is retrieving stock information from a stock broker application or checking the balance of a bank account.

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. We argue that workflow is not a subclass of business process, since a workflow realizes a part of a business process, so a workflow is not in an "is-a" relationship with a business process, but is an association.

With regard to the types of activities mentioned, system activities are associated with workflows, since system activities can participate in any kind of workflow, system workflow or human interaction workflow. User interaction activities and manual activities, however, can only participate in human interaction workflows.

## 5.2 Abstraction Concepts

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 metamodel level, denoted by **horizontal abstraction**.

Aggregation can also be used to cope with complexity, motivating another type of abstraction. At a higher level of abstraction, multiple elements of a lower level of abstraction can be grouped and represented by a single artefact. This type of abstraction is called **aggregation abstraction**, because the coarse-grained business function aggregates activities of smaller granularity.

Aggregation abstraction is different from horizontal abstraction, because all activities (the small-grained and the coarse-grained) are at one horizontal level of abstraction, e.g., the instance level. Aggregation abstraction is primarily used in the functional subdomain, where functions of smaller granularity are combined to create functions of larger granularity.

### 5.2.1 Horizontal Abstraction

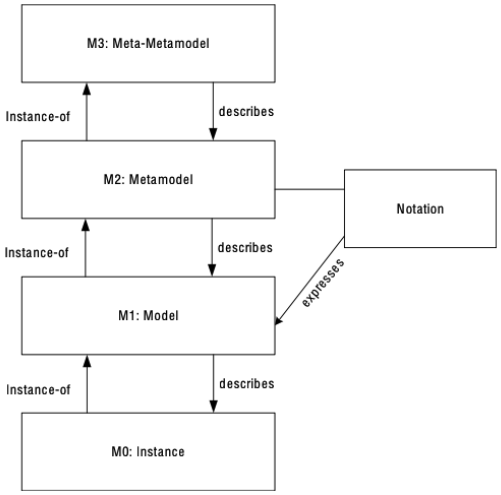



Figure 25: Levels of abstraction

The **instance level** reflects the concrete entities that are involved in business processes. Executed activities, concrete data values, and resources and persons are represented at the instance level (e.g., PetriNet).

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** (e.g., BPM).

In object modelling, 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** (e.g., BML) 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. Typical graphical notations of the Entity Relationship metamodel are rectangles for entity types and diamonds for relationship types, connected by lines.

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.

### 5.2.2 Vertical Abstraction

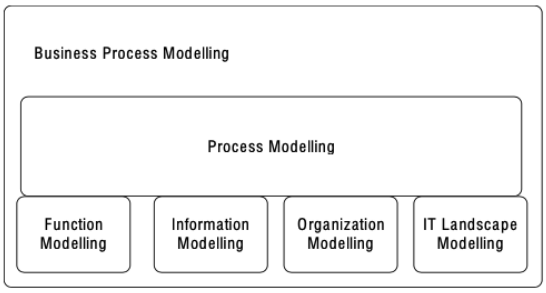



Figure 26: Business process modelling includes multiple modelling domains, integrated by process modelling

in Figure 26, distinct modelling subdomains are identified. 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 **informal**, using English text or formal, using syntactic or semantic specifications of functions. While informal descriptions are mostly done at the coarse business level, more precise specifications are required in the software layer when it comes to implementation of certain functions using information systems.
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.

**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. Data aspects are covered because particular process instances may depend on the structure and value of data involved in a particular business process. For example, in a credit approval business process, the type of approval depends on the credit amount requested. In addition, data dependencies between activities need to be taken into account in process model design.

## 5.3 Process Models and Process Instances

### 5.3.1 Business Process meta-model

In this section, a simple process metamodel is introduced. 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. Activity model nodes do not act as split or join nodes, i.e., each activity model has exactly one incoming edge and exactly one outgoing edge.
  - Event Models: capture the occurrence of states relevant for a business process. Process instances start and end with events, so process models start and end with event models.
  - Gateway Models: Gateways are used to express control flow constructs, including sequences, as well as split and join nodes.

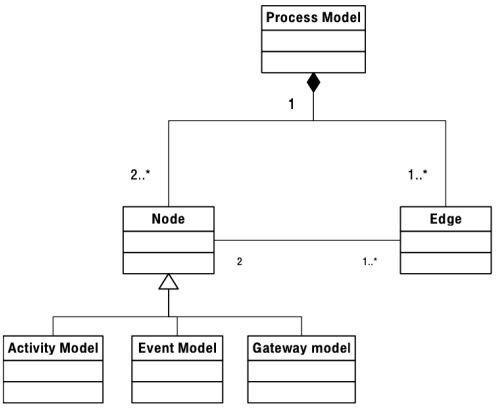



Figure 27: BModel for process models: process metamodel

## 5.4 Activity Models and Activity Instances

An activity model describes a set of similar activity instances, analogously to a process model describing a set of process instances with the same structure. While process models are typically expressed in graph-like notations (to be investigated in detail in the next chapter), activity models can be expressed in different forms, for instance, by plain text or by some formal specification or references to software components that implement them.

Activity instances represent the actual work conducted during business processes, the actual units of work. Each activity instance during its lifetime is in different states. These states and the respective state transitions can be represented by a **state transition diagram**.

The states that an activity instance adopts during its lifetime are described as follows: when it is created it enters the **init state**. By the enabled state transition the activity instance can enter the **ready state**. If a particular activity instance is not required, then the activity instance can be skipped, represented by a **skip state** transition from the not started state to the **skipped state**. From the ready state, the activity instance can use the begin state transition to enter the running state. When the activity instance has completed its work, the terminate state transition transfers it to the **terminated state**. When an activity instance is in the terminated or the skipped state, then it is **closed**. However, the substates of the termination state are not rep-

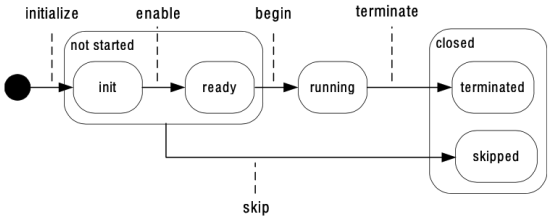



Figure 28: State transition diagram for activity instances

resented. The simple diagram does not provide different states for successful and unsuccessful termination. When an activity instance can be started, it enters the ready state. If during the execution of a process instance certain activity instances are currently not available for execution, they can be disabled. Activity instances that are initialized, disabled, or enabled are in the not started state. Once an activity instance is ready, it can be started, entering the running state. Running activities can be temporarily suspended, to be resumed later. An activity instance can terminate either successfully or in failure. Terminated activity instances can be undone, using compensation or transactional recovery techniques.

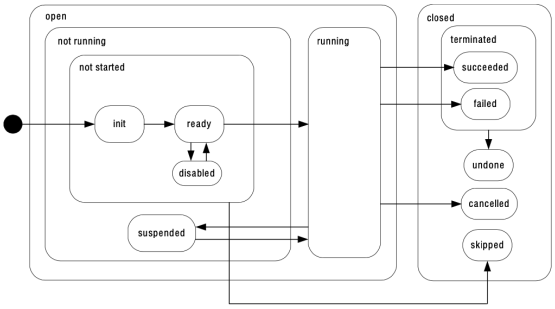



Figure 29: State transition diagram for activity instances, detailed version

These events have a temporal ordering. Based on the state transition diagram for activity instances, each activity instance can be represented by a totally ordered set of events. In event

diagrams, time proceeds from left to right, and events are shown as bullets. The causal relationships of events are represented by directed arcs. Due to the nature of event diagrams, they form directed acyclic graphs, where the nodes are events and the edges reflect causal ordering between events.

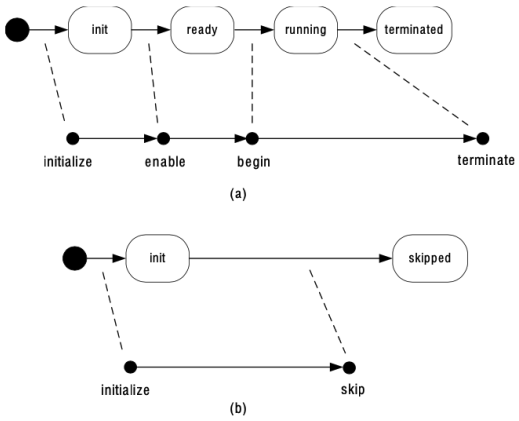



Figure 30: Event diagram for (a) executed activity instance and (b) skipped activity instance

## 5.5 BPMN Model

Business Process Model and Notation (used in this course): https://www.omg.org/spec/BPMN/2.0/PDF

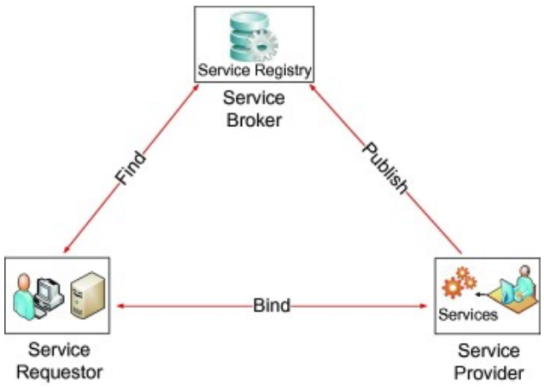
# 6 Service Oriented Architecture

## 6.1 Main concepts

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

Key concepts:

- SOA is a paradigm, not a technology!
- SOA is intended to be distributed, thus the connection among systems is mandatory
- SOA allows to model services provided even by different owners




Different actors are playing in the SOA context. There is a **Service Provider** who publishes services' interfaces and description of his software components to a **Service Broker**. Then a **Service Consumer**, who is looking for a specific function/service to achieve its goals can search for services on the broker.

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

**Service Oriented Architecture is a way of designing, developing, deplying, and managing systems**, in which:

- Service Providers 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 a clear separation between the service interface and service implementation (consumers can see only the interface, while the implementation is visible only to providers)
- A SOA infrastructures enables discovery, composition, and invocation of services

The main benefit that comes with SOA, specially for enterprises, consist in the fact that, while traditional approaches result in developing different applications with similar functionalities, **with SOA functionalities and capabilities of the system can be encapsulated and**

**reused for different purposes and contexts**. Moreover, services can (and MUST!) be considered as autonomous entity, that possibly can be composed together 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 end users**, this means that they can be used by other components of the same enterprices for which the audience is well-known.
They could also be designed and implemented for **external users**. In this case, we are talking about *services as a product*, where services are usually exposed as API.

## 6.2 Service-orientation Design principles

The service-orientation design principles may be broadly categorized as follows, following Thomas Erl's, SOA Principles of Service Design:

- **Standardized Service Contract**: contract includes both functional and non-functional aspects of the service
  - Functional aspects (e.g, input/output, protocols, uri) are usually described via API (WSDL for SOAP and OpenAPI for REST)
  - Non-functional aspects are not so standardized so far (WS-agreement is one of the most adopted standard)

  From a technical perspective, contracts are exposed as service descriptions with machine readable and understandable languages.
- **Service Loose Coupling:** lifecycle of a service should be as much as possible indepedent from its consumer. Implemementation and adopted technology, i.e., how a service fulfills the contract, must be independent.
- **Service Abstraction:** the service contract should contain only the minimal set of information to allow the consumer to invoke the services. All the information about how the service is implemented must be hidden.
- **Service Reusability**: a service should not be designed only to solve a specific need
  - The design of the service should be more generic in order to allow the reuse in other contexts. Having a reusable service may result in a tangible value for the business of the enterprise (API economy)
- **Service Autonomy:** service should support the self-* properties
  - self-healing
  - self-configuring
  - self-optimizing
  - self-protecting

- **Service Statelessness:** to increase the autonomy and the loose coupling a service should not retain any state. State must be stored by the consumer and not by the provider to increase scalability and flexibility of the resulting system.
  - Stateless service: only client maintains the state
  - Stateful service: server maintains information (the state) about each interaction for all the clients
- **Service Discoverability:** in addition to the contract, services should be accompained with metadata to increase their discoverability and further describe their purposes and capabilities to humans.
- **Service Composability:** a service can be designed as a composition of other services. The composition could be driven by a business process.

## 6.3 Service-orientation benefits

- **Standardization**: increase consistency in how functionalities and data is represented
- **Loose Coupling**:
  - Reduced dependencies between units of solution logic
  - Reduce awareness of underlying solution logic design and implementation details
- **Reusability**:
  - increased opportunities to use a piece of solution logic for multiple purpose
  - increased opportunities to combine units of solution logic into different configurations
- **Availability and Scalability**

# 7 SOAP and REST

## 7.1 SOAP

**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. Indeed it 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.

## 7.2 REST

**REST is an architectural style** (NOT A PROTOCOL!). It provides a more scalable approach to implement web services w.r.t. RPC-SOAP. While RPC-SOAP are method oriented, **REST is resource oriented**. It takes inspiration from the Web (HTTP) which is flexible and scalable by definition.

REST tries to overcome some SOAP limits:

- SOAP is based on RPC (another architectural style)
- SOAP as an high overhead for transformation and conversion of messages, which applies both on sending and receiveing
- SOAP uses only XML as encoding scheme, which is computation intensive when parsing
- SOAP is stateful, and this has an impact on scalability

### 7.2.1 REST constraints

:

- **Client-Server**: the basic architectural style enabling 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 (client must maintain the state)
- **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

### 7.2.2 REST on HTTP

Main Concepts:

- **Resource**: abstraction of an entity
- **Representation**: concrete entity encoded via specific grammars (e.g. JSON, XML)
- **Methods**: how to work with resources (GET, PUT, POST, DELETE, PATCH, HEAD). These are the only methods permitted and are known a priori, while in SOAP we had WSDL specifying the methods exposed by a service.
- **URI**: how to uniquely identify a resource

To obtain a uniform interface and better describe our API and related resources, we can follow this approach:

- Distinguish between resource (real data) and representation
- Few and well-defined operations allowed on the resources, possibly only the CRUD (Create, Read, Update, Delete) operations
- Messages in addition to the resource info also includes all the information to proper manage the resource (HATEOAS) to reduce the a-priori knowledge

### 7.2.3 Resources

A resource represents a data structure

- Identified by a URI (unique name of a resource)
- Accessible through methods

We can distinguish different archetypes of resources:

- **Document (singular)**:
  - represents an object instance or a database record
  - its URI has a singular noun
  - Examples (single documents inside a collection):
    *http://api.soccer.restapi.org/leagues/seattle*
    *http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet*
    *http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players/mike*
- **Collection**: server-managed directory of resources
  - Client may propose to add new resources to a collection
  - Server mediates and decide if it can be stored (e.g., status and validity checks)
  - Examples:
    *http://api.soccer.restapi.org/leagues*
    *http://api.soccer.restapi.org/leagues/seattle/teams*
    *http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players*
- **Store**: client-managed resource repository

– Client completely manages the resource
– No middleware mediates between the client and the storage (client can add whatever)
– Examples:
*http://api.music.restapi.org/artists/mikemassedotcom/playlists*

```xml
<vm xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <name type="xs:string">My VM</name>
   <memory type="xs:int">1024</memory>
   <cpu>
      <cores type="xs:int">4</cores>
      <speed type="xs:int">3600</speed>
   </cpu>
   <boot>
      <devices type="xs:list">
         <device type="xs:string">cdrom</device>
         <device type="xs:string">harddisk</device>
      </devices>
   </boot>
</vm>
```

Figure 31: XML representation of a resource

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

Figure 32: JSON representation of a resource

| URL | Description |
|---|---|
| /api | The API entry point |
| /api/:coll | A top-level collection named “coll” |
| /api/:coll/:id | The resource “id” inside collection “coll” |
| /api/:coll/:id/:subcoll | Sub-collection “subcoll” under resource “id” |
| /api/:coll/:id/:subcoll/:subid | The resource “subid” inside “subcoll” |

Figure 33: Example of URL structure

| 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. |

Figure 34: REST request methods explained

| 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. |

Figure 35: REST response status code categories

### 7.2.4 HATEOAS

```
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>
```

Figure 36: HATEOAS examples: different links can be returned according to the operations permitted to the specific user

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 customised to return different links based on the user who is calling that method.

## 7.3 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. |

# 8 Service Portfolio

Service Portfolio design concerns the need to design in a cohernt way the set of services of an organization.

After having identified and modeled our business process, we should follow a number of steps to design our portfolio.
Thomas Erl defined 4 types of services inside our process (we only analyze consider 3 of them):

- **Task service**:
  - non-agnostic and focused on a specific purpose inspired by a business process
  - perform stateful activities which are well-defined in a specific context and invoked in a specific step in our business process
- **Entity service**:
  - agnostic with respect to the functional context
  - models the connection between our business process and the informational resources (databases)
- **Utility Service**:
  - agnostic and focused on a specific purpose
  - similar to task services, but related to common activities (e.g., logging, notification, message exchange)

Erl proposed also a process to design our service portfolio (in classes we analyzed a simplified version of the original one):

1. **Revise business process**:
   - Ensure that the model has no different level of aggregation: functional sub-processing is ok, while hierarchical sub-processing should be solved (expanded)
   - Clarify who/what system is in charge of executing the activities (specify icons for all the activities in the business process)
2. **Filter-out unsuitable actions**: remove the activities marked as "manual task" because we are not able to manage them within the service context
3. **Define Entity Service Candidates**: highlight entity services i.e., the one referring to business object, which usually are REST services whose activities are marked as "service task" or "user task"
4. **Define Utility Service Candidates**: highlight utility services that usually refers to common functions (e.g., message events)
5. **Define Task Service Candidates**: highlight task services, the one implementing a specific activity which is fundamental in the context of our business process and that could be a composition of a process at a lower level

6. **Identify Process-Specific Logic**:
    - specify the interface that customers can use to interact with our business process
    - specify who/what is in charge to evaluate branch conditions and how
7. **Design Entity Services**: define collections managed by entities along with the permitted operations. Usually each collection is managed by a different service (3 collections = 3 services
8. **Design Utility Services**: usually embedded in BPMS. We can use and additional entity service for notifications/messages, thus another collection and another REST service.
9. **Design Task Services**:
    - with BPMS: having a BPMS means to rely on the API exposed by it, thus the process model should include service calls to the identified services based on the agreed interface
    - without BPMS: develop a service and expose it through a well defined interface. The service will then be called by the client.

# 9 References

[1] C. M. MacKenzie et al.,Reference Model for Service Oriented Architecture 1.0., OASIS Open, 2006