cedric

le cnam

Quality-driven reference architecture incremental design: an industrial experience

N. LEVY

BASED ON A WORK DONE TOGETHER WITH

MUSTAPHA DERRAS, LAURENT DERUELLE, VALÉRIE REINER FROM BERGER-LEVRAULT

JEAN MICHEL DOUIN, YANN POLLET, GERARDO RODRIGUEZ FROM CEDRIC - CNAM

FRANCISCA LOSAVIO FROM UNIVERSIDAD CENTRAL DE VENEZUELA

Context

Berger-Levrault designs solutions for local authorities and public administrations as well as public and private healthcare facilities, educational institutions, universities and private companies.

These solutions MUST be compliant with all kind of legislation

Problems faced :

- Legislation changes very often (and it will not stop)
- Cities are different, based on their size, location, etc but have similar needs
- Berger-Levrault has acquired different companies proposing similar products

Problem

How to define a single system, common to all and adapted for everyone?

First idea of solution

Product Line

Have a **core system** common to all and **variation points** where specific behaviors will be defined for each one.
These specific behaviors can be chosen and refined by a **configuration**

New problem

How to define such a **Product Line** ?

Problem

How to define a **Product Line** = a core system with variation points to be configured ?

Second idea of solution

- Describe a **Reference Architecture** for a given domain (**the core**)
- Introduce in the Reference Architecture **variation points**,
  that are hooks or place holders
  where different solutions will be attached

How will variations points be chosen ?

Quality requirements are a major cause of variability

Problem

How to define a **Product Line** = a configurable core system with variation points ?

Second idea of solution

- Describe a **Reference Architecture** for a given domain (**the core**)
- Take into consideration **products quality aspects :**
  introduce **non functional components** in the reference architecture
- Declare as **variation points** these non functional components …
  and maybe others …
- Enable the **configuration** of the system to select a solution

Problem

Is this a reusable approach ?

Third idea of solution

Describe a **Methodology** to define a **Product Line** of a given domain with

- Functional and
- Non-functional

} components

where some will be denoted as **Variation points**

together with a mechanism of **Configuration**

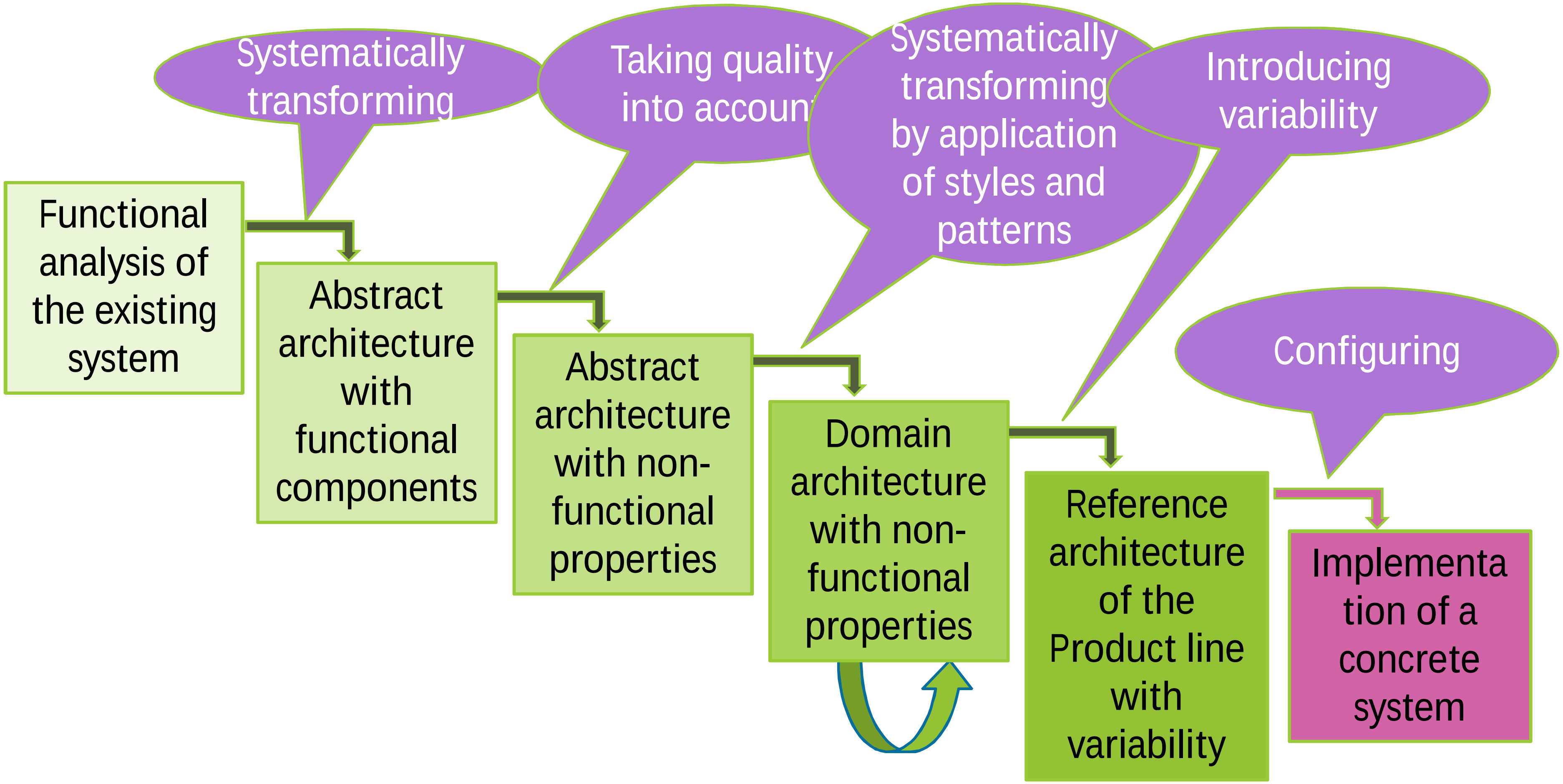
Top-down ⬄ Bottom-up approaches

A **bottom-up strategy** will be followed starting from an existing product

Case study with **Berger-Levrault** :

An industrial experience in the Human Resources domain : a Vacation Request System that takes into account different regulations

Big picture of the methodology


Systematically transforming
Taking quality into accoun
Systematically transforming by application of styles and patterns
Introducing variability
Configuring
Functional analysis of the existing system
Abstract architecture with functional components
Abstract architecture with non-functional properties
Domain architecture with non-functional properties
Reference architecture of the Product line with variability
Implementation of a concrete system

Case study : vacation request

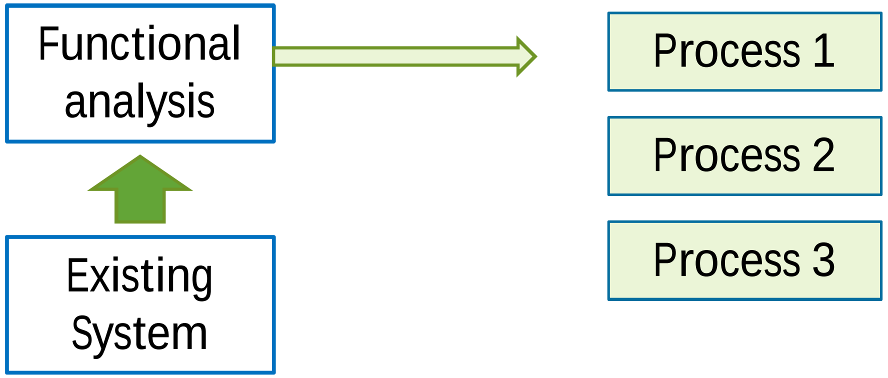
Bottom-up approach

Based on interviews with stakeholders of our industrial partner **Berger-Levrault**

→ Definition of some data flow diagrams for the **vacation request** business process followed by French municipal communities

Functional analysis of the existing system

Description of functional processes



Functional analysis

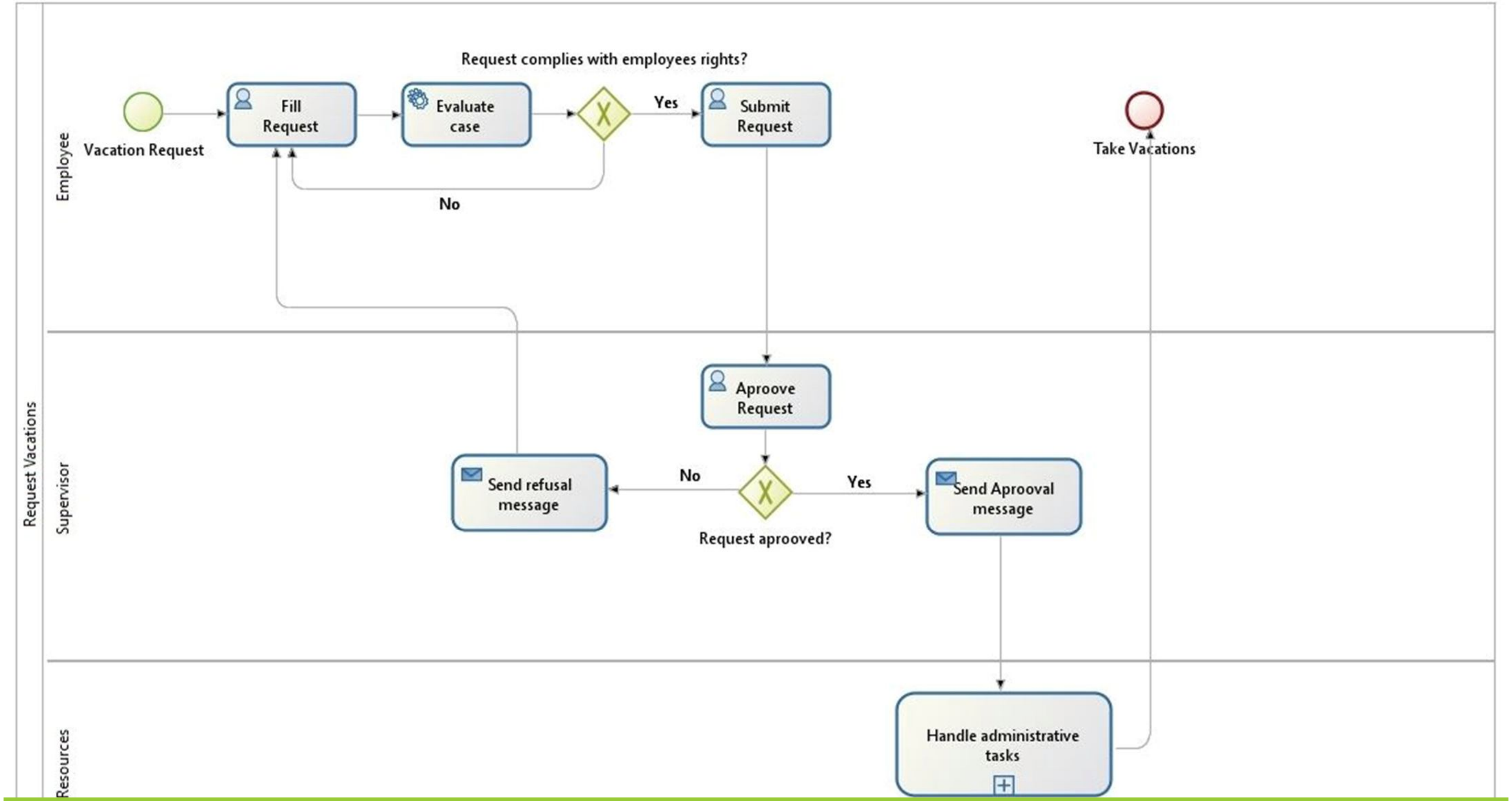
Existing System

Process 1

Process 2

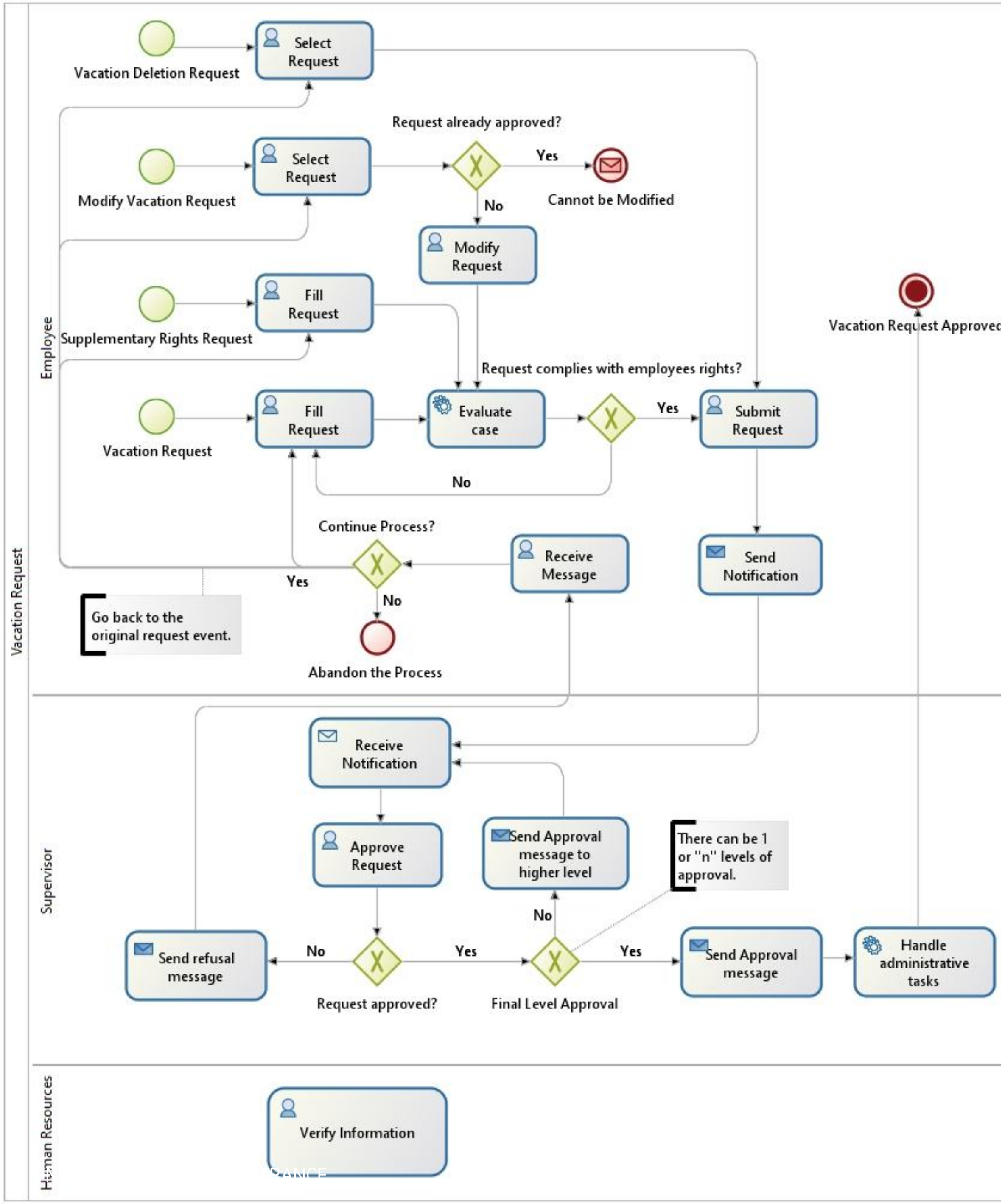
Process 3

BPMN



Vacation request business process

Complete vacation request business process

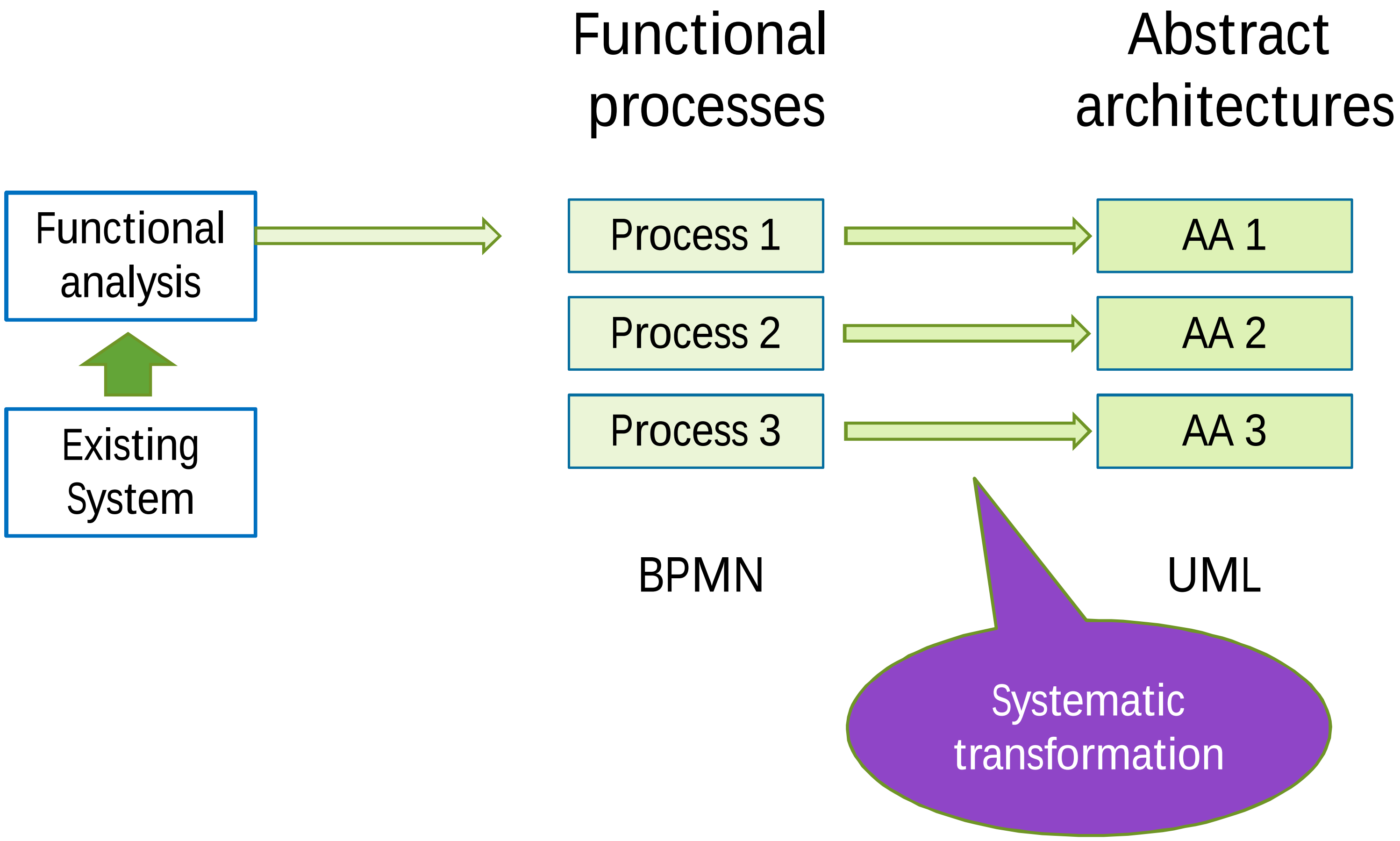


From Functional analysis to Abstract architecture

- Introduce a component for each stakeholder’s lane in a pool that accomplishes some business goal
- Introduce a sub-component for each task
- Only functionalities and their cooperation are concerned

→ the result is an abstract architecture with only functional components

From functional analysis to abstract architecture



Functional processes

Abstract architectures

Functional analysis

Existing System

Process 1

Process 2

Process 3

AA 1

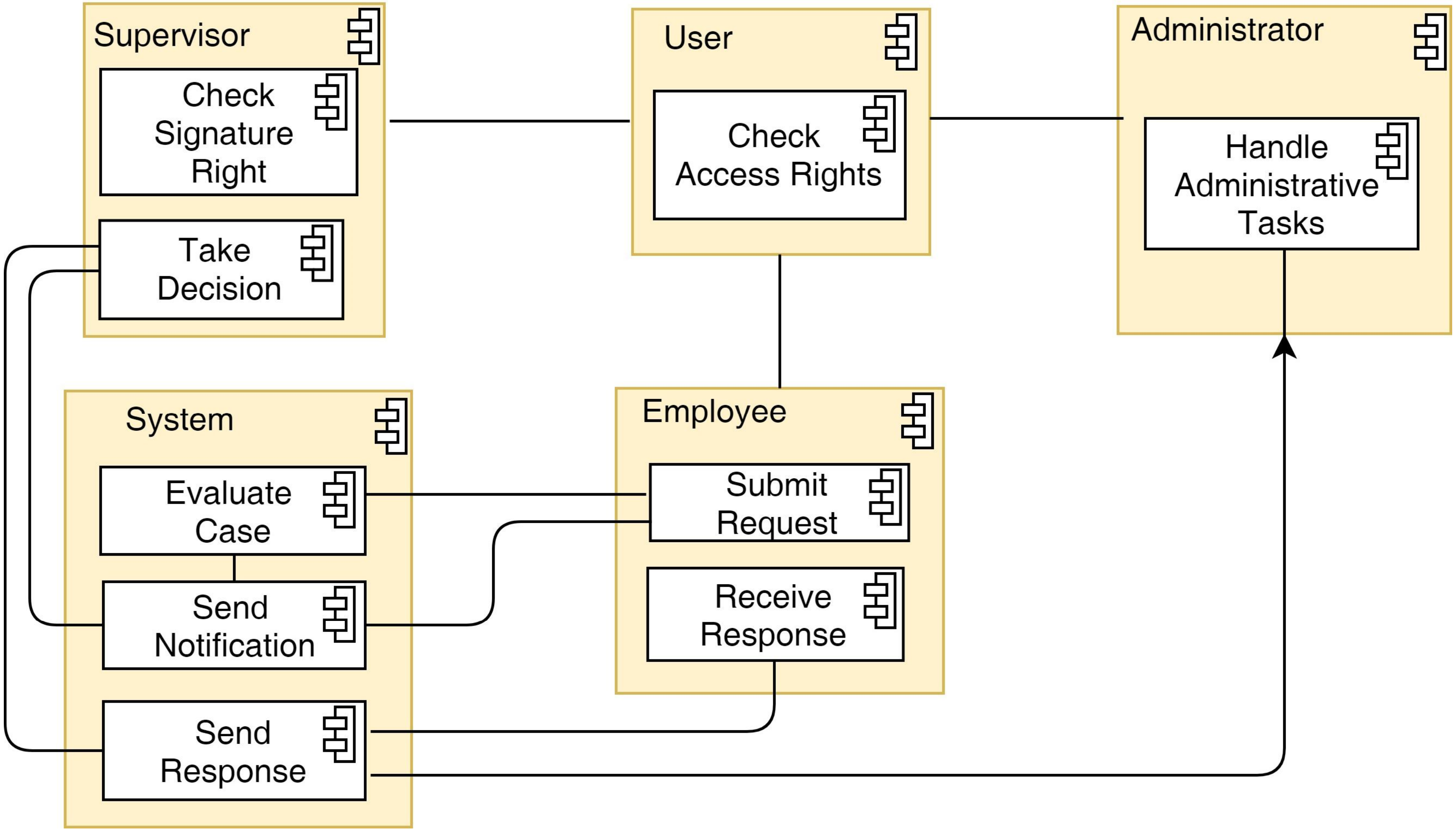
AA 2

AA 3

BPMN

UML

Systematic transformation



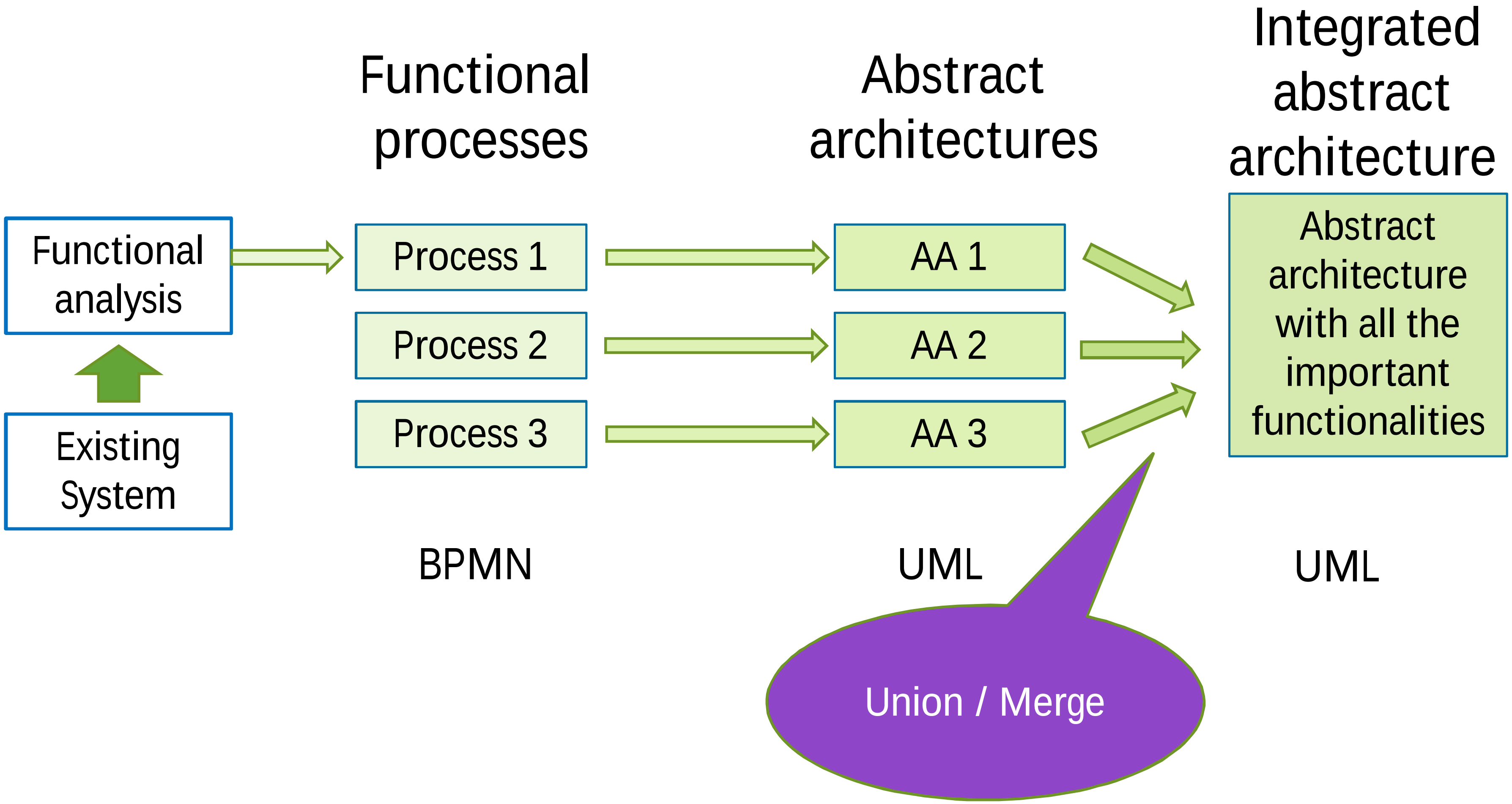
Abstract architecture with functional components

From various abstract architectures to a single one

This is done by

- Unifying all the components
- Merging the components that correspond to the same functionality

From various abstract architectures to a single one



Functional processes

Abstract architectures

Integrated abstract architecture

Functional analysis

Existing System

Process 1

Process 2

Process 3

AA 1

AA 2

AA 3

Abstract architecture with all the important functionalities

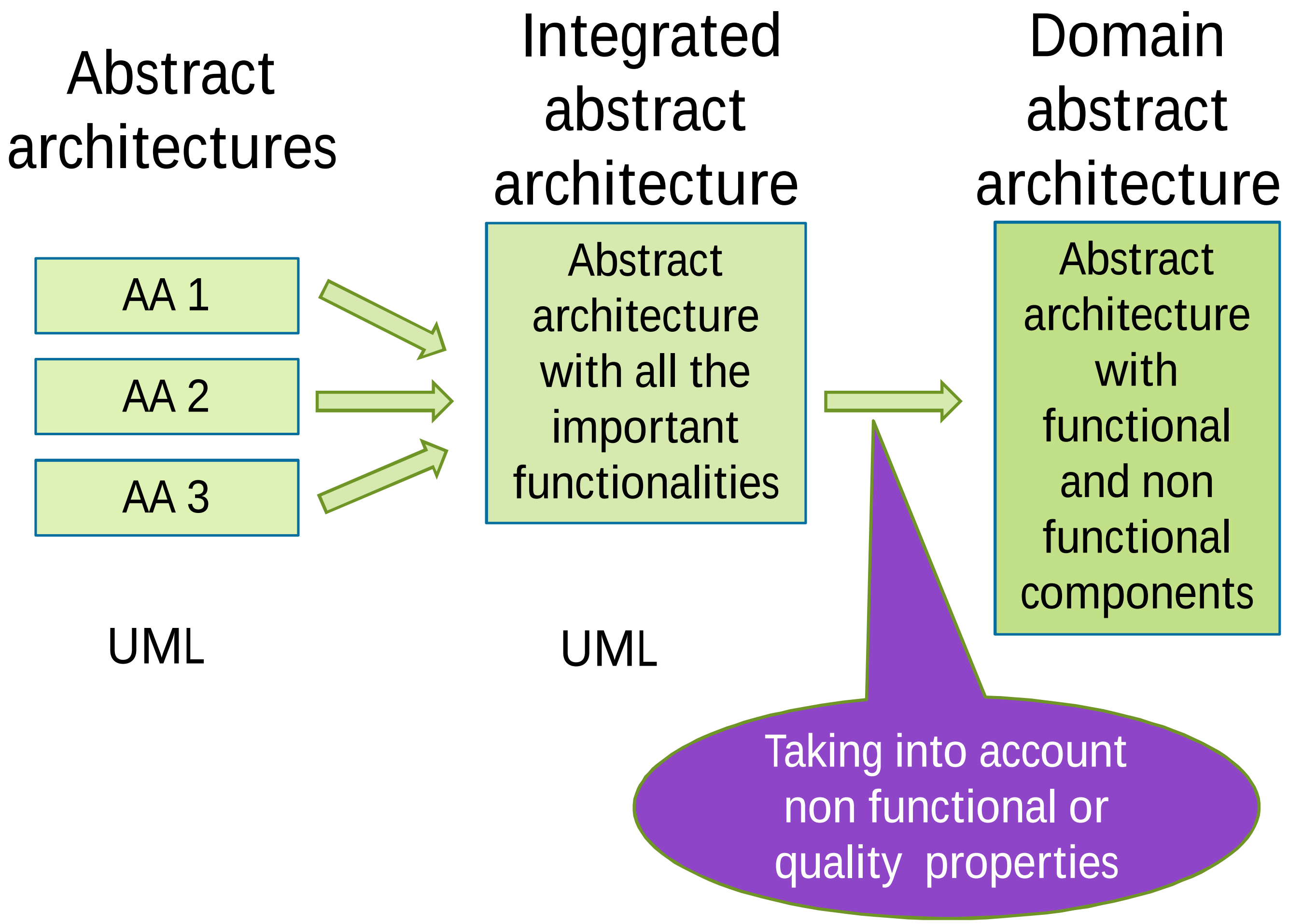
BPMN

UML

UML

Union / Merge

Introducing non functional properties



Abstract architectures

AA 1

AA 2

AA 3

UML

Integrated abstract architecture

Abstract architecture with all the important functionalities

UML

Domain abstract architecture

Abstract architecture with functional and non functional components

Taking into account non functional or quality properties

Taking into account non functional or quality properties

Non functional or **quality properties** are not directly perceived by the user, but they are required by the functional components to satisfy completely their business goals

1. Choose the quality properties expected from the system as the Product Quality Model. We use the ISO 25010
2. Assign priorities, with respect to the business goals, to each quality properties
3. For each quality property :

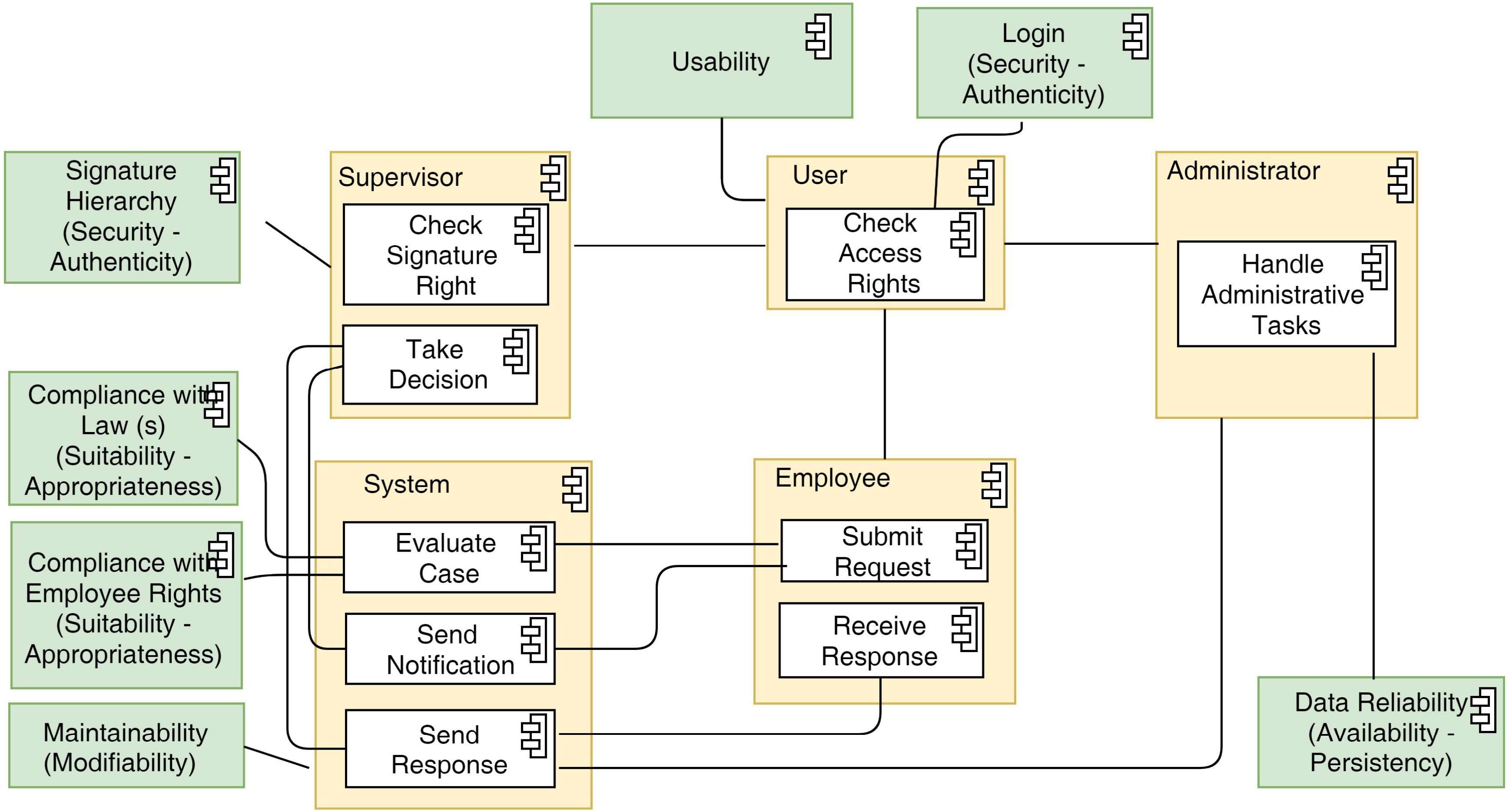
   integrate it as a non-functional component,

   relate it with the functional component requiring it

Quality properties for the vacation request case study

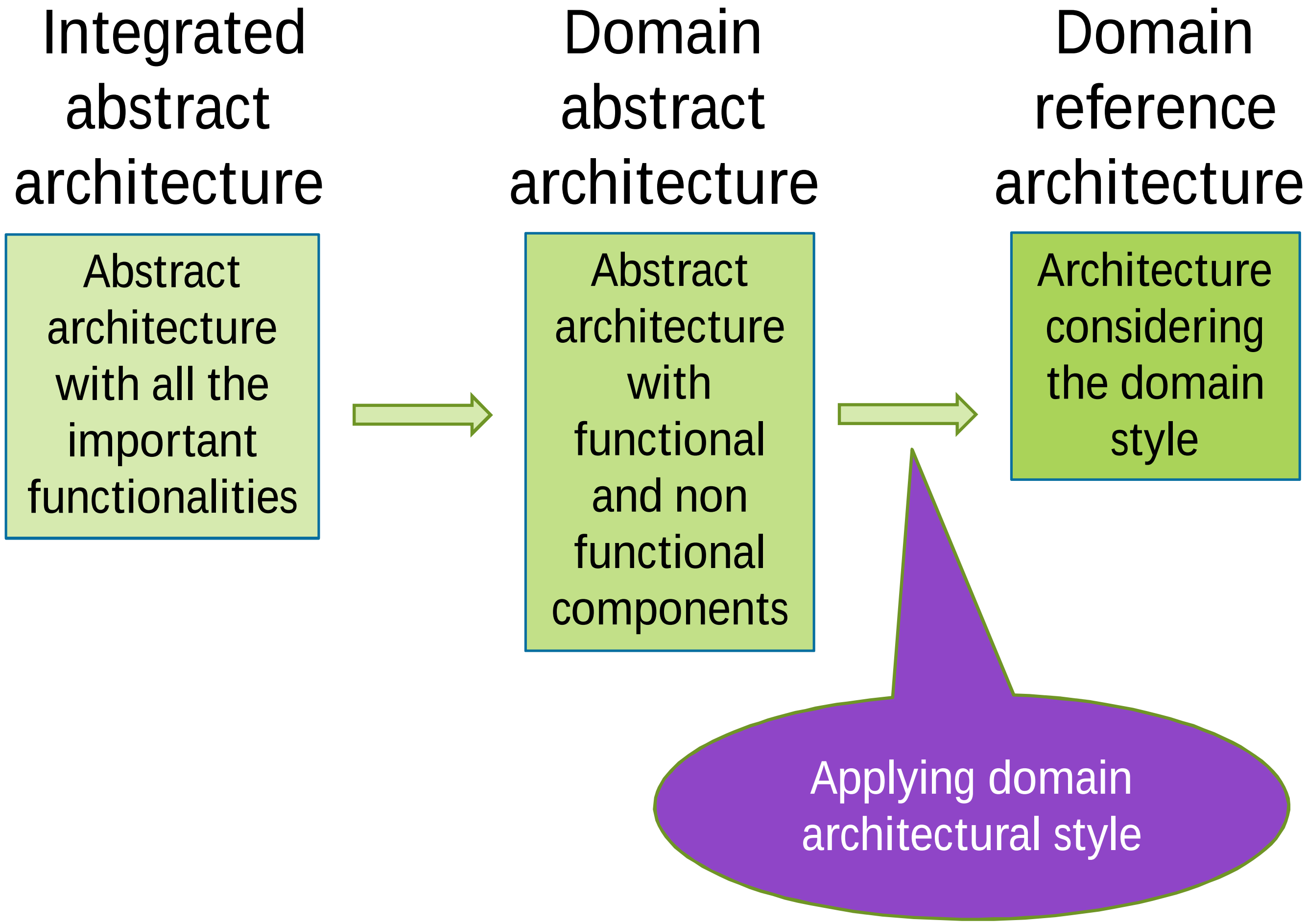
In our case, the main quality properties are :

1. *Security (authenticity)* for all users and concerning the hierarchy to be considered for signatures
2. *Functional suitability (appropriateness, correctness) :* compliance with law and with employee's right
3. *Usability* for all users
4. *Maintainability (modifiability)* concerning the system
5. *Reliability (availability, persistency)* concerning the administrative data


Usability
Login (Security - Authenticity)
Signature Hierarchy (Security - Authenticity)
Supervisor
Check Signature Right
Take Decision
User
Check Access Rights
Administrator
Handle Administrative Tasks
Compliance with Law (s) (Suitability - Appropriateness)
Compliance with Employee Rights (Suitability - Appropriateness)
Maintainability (Modifiability)
System
Evaluate Case
Send Notification
Send Response
Employee
Submit Request
Receive Response
Data Reliability (Availability - Persistency)

Abstract architecture with functional and non functional components

Refining the architecture



Integrated abstract architecture

Abstract architecture with all the important functionalities

Domain abstract architecture

Abstract architecture with functional and non functional components

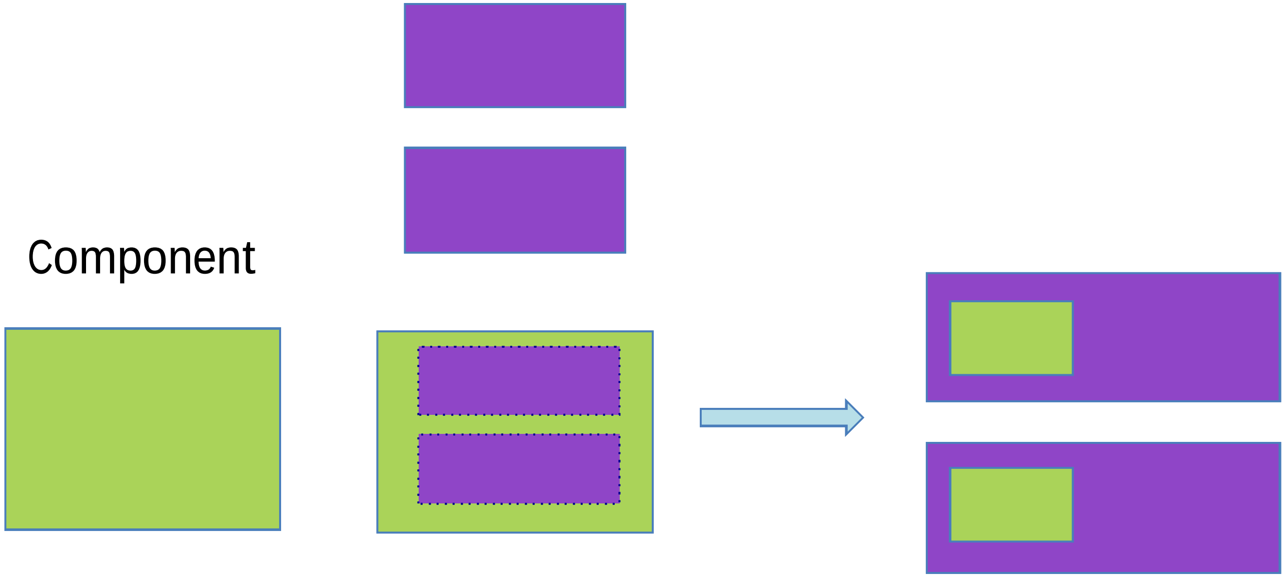
Domain reference architecture

Architecture considering the domain style

Applying domain architectural style

Applying an architectural pattern to a component

Architectural pattern = <architectural configuration , properties>



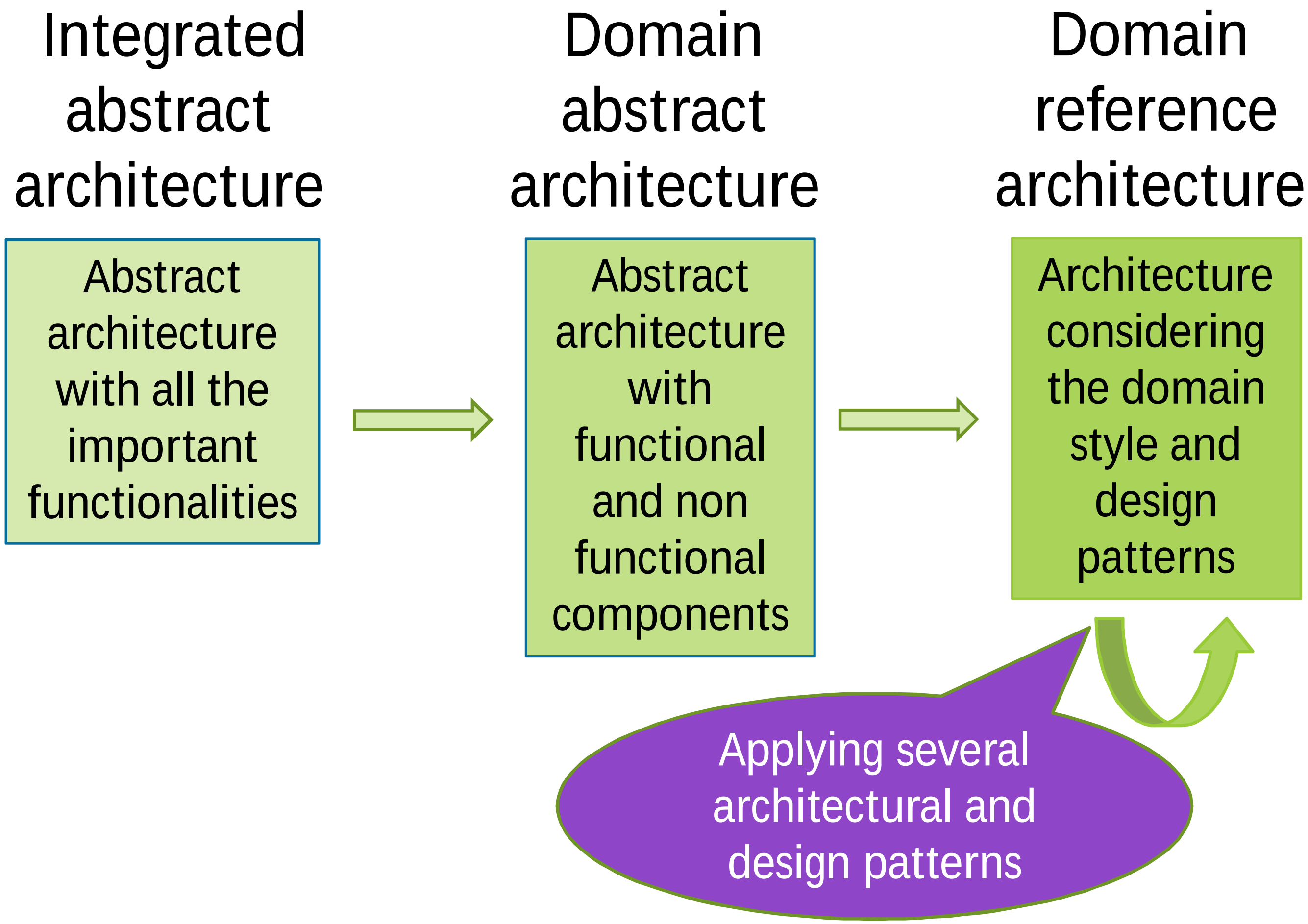
Pattern

Component

Taking into account the style in the component

Architecture taking into account domain patterns

Refining the architecture



Integrated abstract architecture

Abstract architecture with all the important functionalities

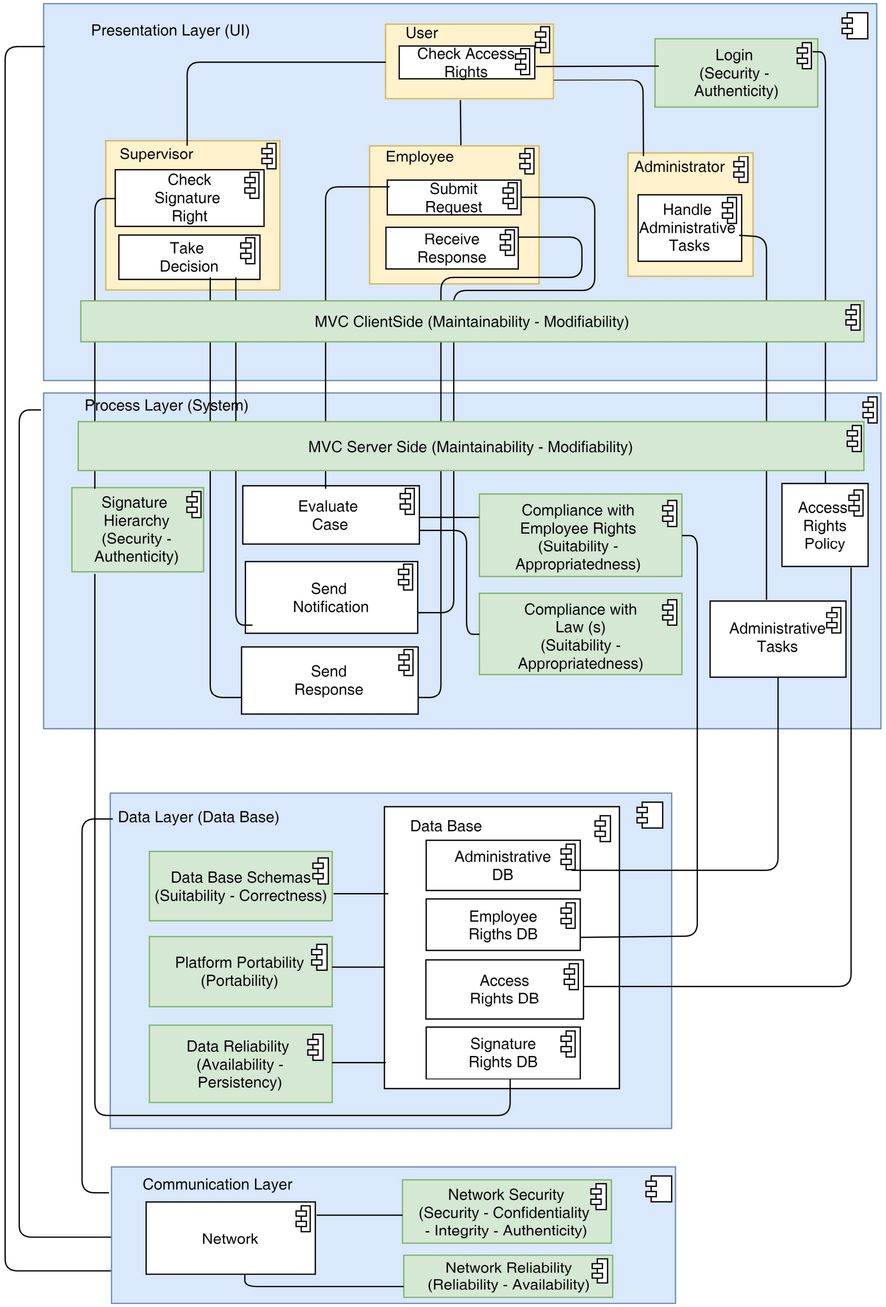
Domain abstract architecture

Abstract architecture with functional and non functional components

Domain reference architecture

Architecture considering the domain style and design patterns

Applying several architectural and design patterns

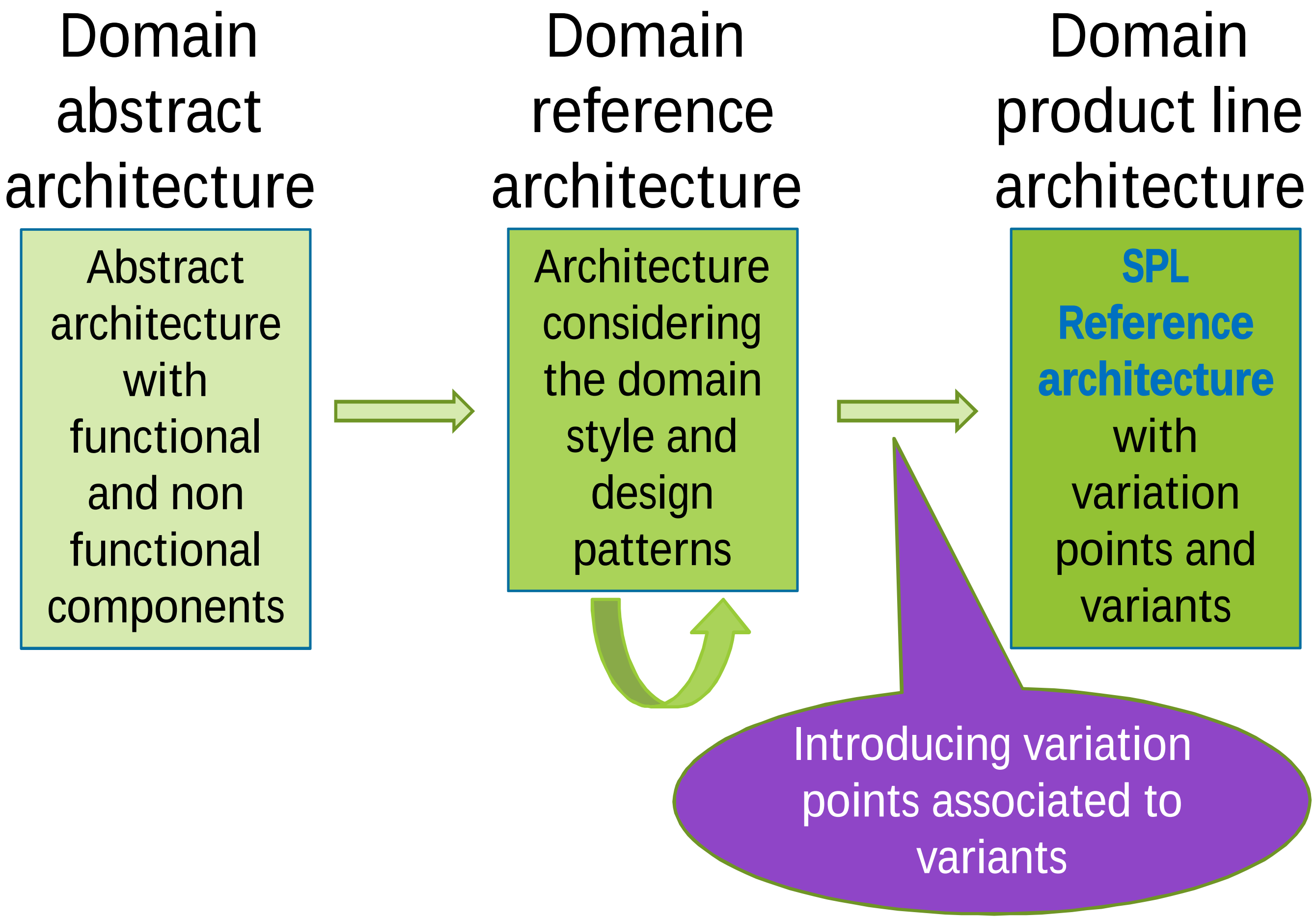

Presentation Layer (UI)
User
Check Access Rights
Login (Security - Authenticity)
Supervisor
Check Signature Right
Take Decision
Employee
Submit Request
Receive Response
Administrator
Handle Administrative Tasks
MVC ClientSide (Maintainability - Modifiability)
Process Layer (System)
MVC Server Side (Maintainability - Modifiability)
Signature Hierarchy (Security - Authenticity)
Evaluate Case
Compliance with Employee Rights (Suitability - Appropriatedness)
Access Rights Policy
Send Notification
Compliance with Law (s) (Suitability - Appropriatedness)
Administrative Tasks
Send Response
Data Layer (Data Base)
Data Base
Data Base Schemas (Suitability - Correctness)
Administrative DB
Employee Rigths DB
Platform Portability (Portability)
Access Rights DB
Data Reliability (Availability - Persistency)
Signature Rights DB
Communication Layer
Network
Network Security (Security - Confidentiality - Integrity - Authenticity)
Network Reliability (Reliability - Availability)

Architecture considering the domain style

The architectural layer pattern is composed of different layers :

- Presentation layer
- Process layer
- Data layer
- Communication layers in between

Introducing variability



Domain abstract architecture

Abstract architecture with functional and non functional components

Domain reference architecture

Architecture considering the domain style and design patterns

Domain product line architecture

SPL Reference architecture with variation points and variants

Introducing variation points associated to variants

Introducing variability

“Software variability is the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context “

Jan Bosch

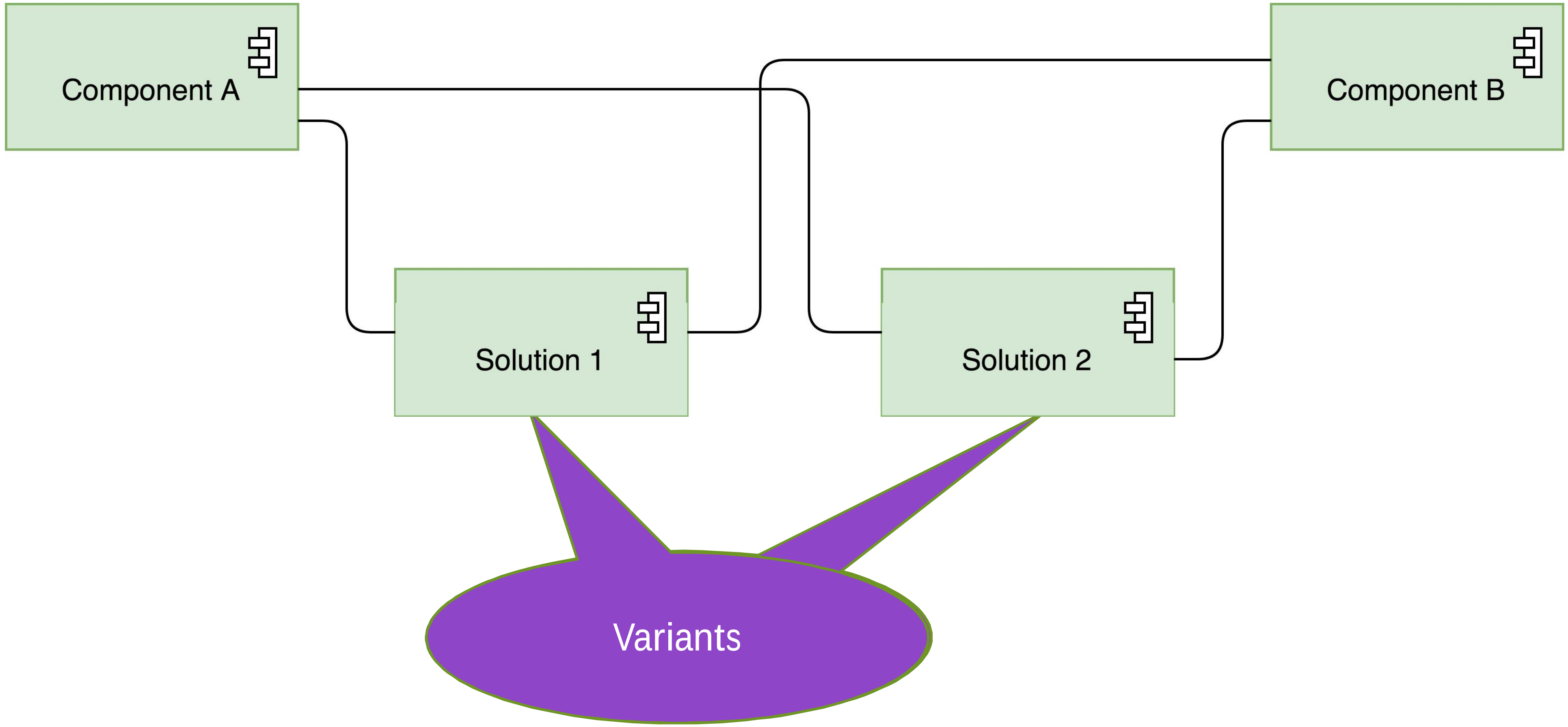
To organize variant elements so they can be reused when deriving a concrete products we introduce

Variation points and their **variants** attached to them

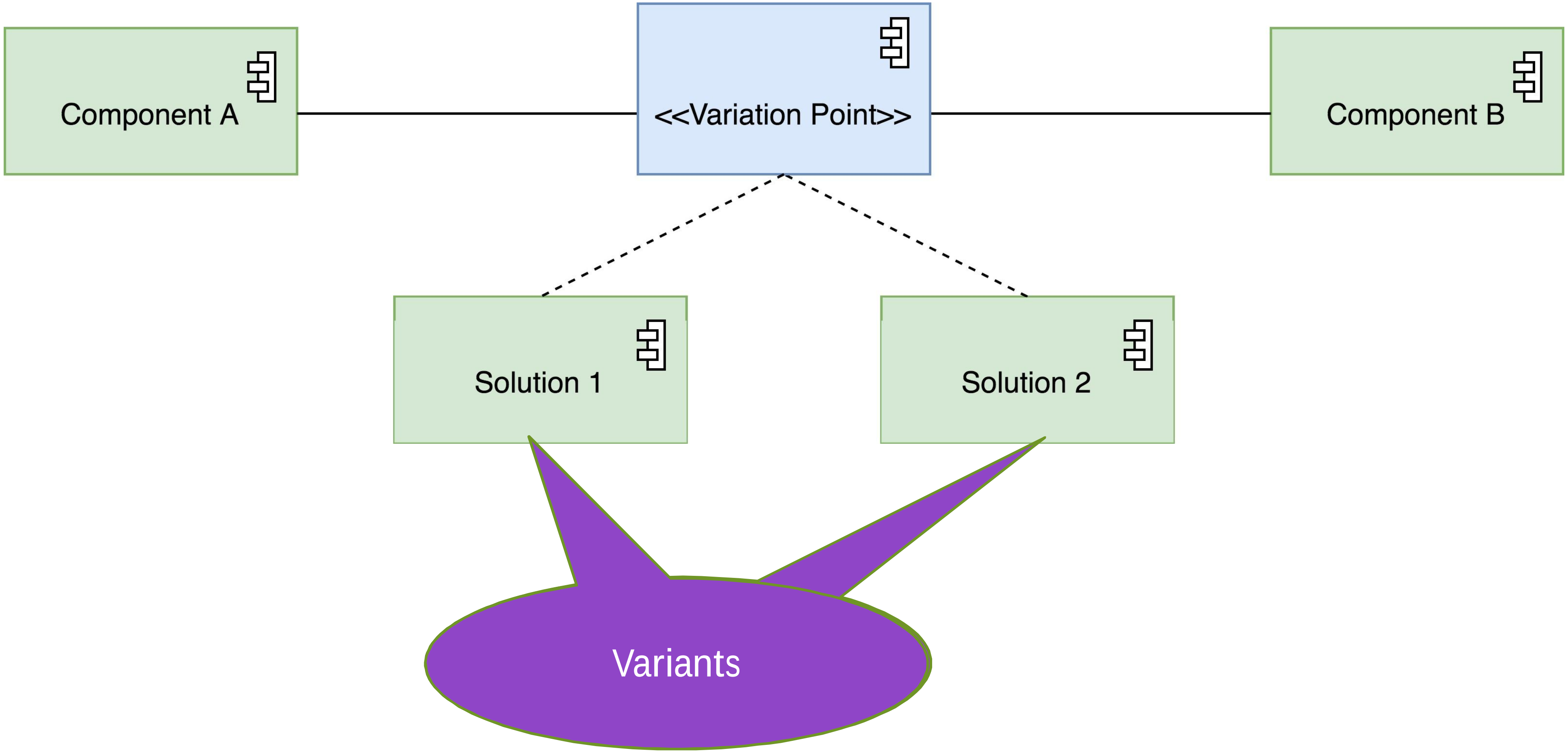
Variation points are denoted << *name*>> as UML stereotypes.

They are sets of components, whose elements are called *variants*

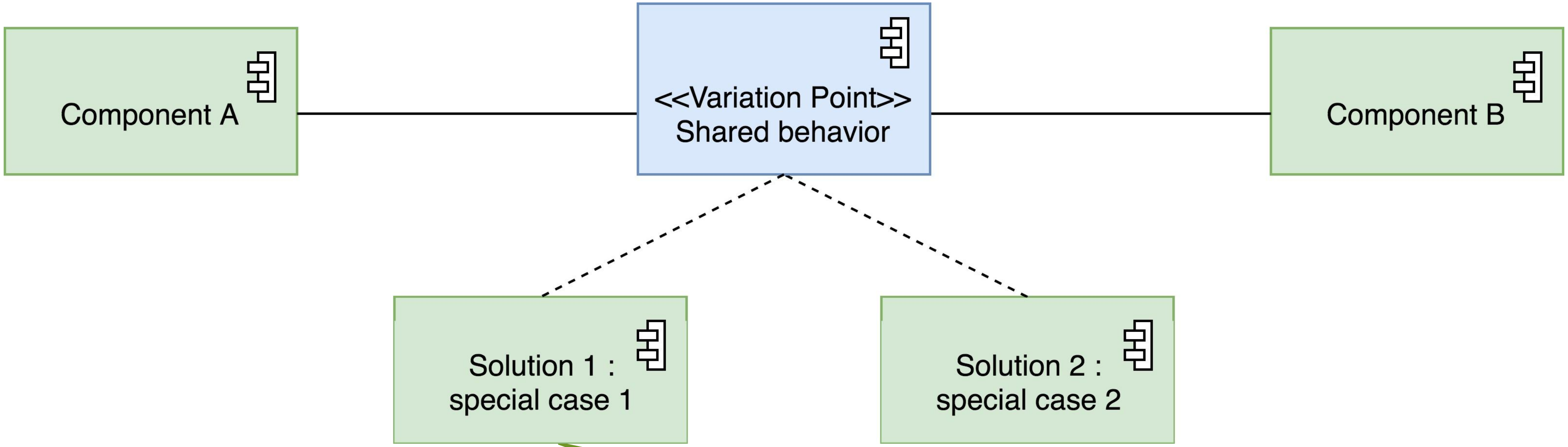
Defining variability



Defining variability



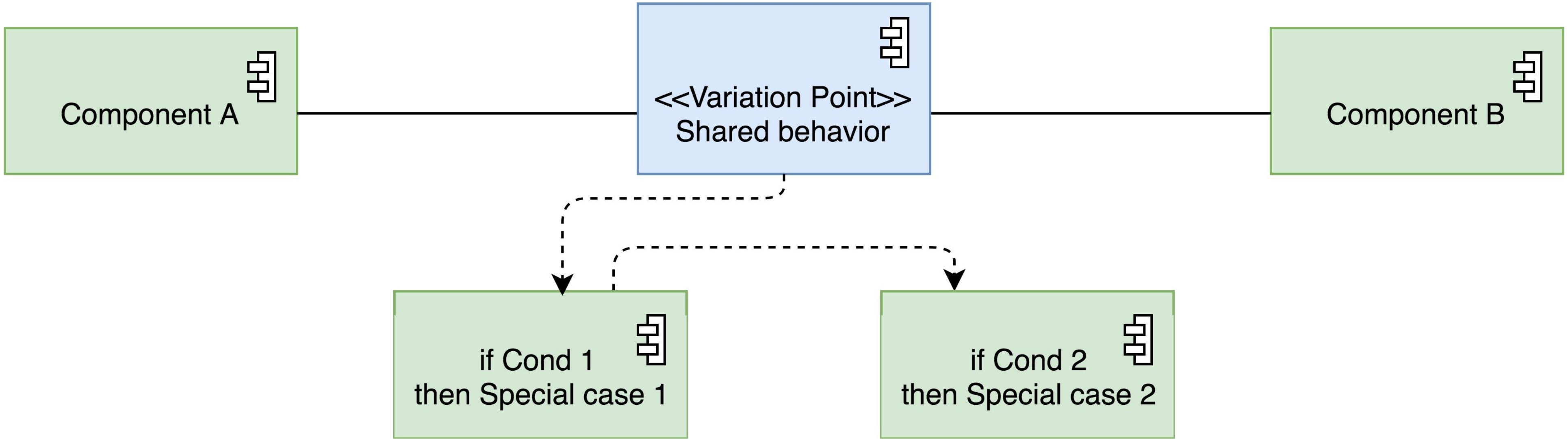
Defining variability



Variants :
1 – Shared behavior + special case 1
2 – Shared Behavior + special case 2

Variants having partly the same behavior and having also a specific one

Defining variability

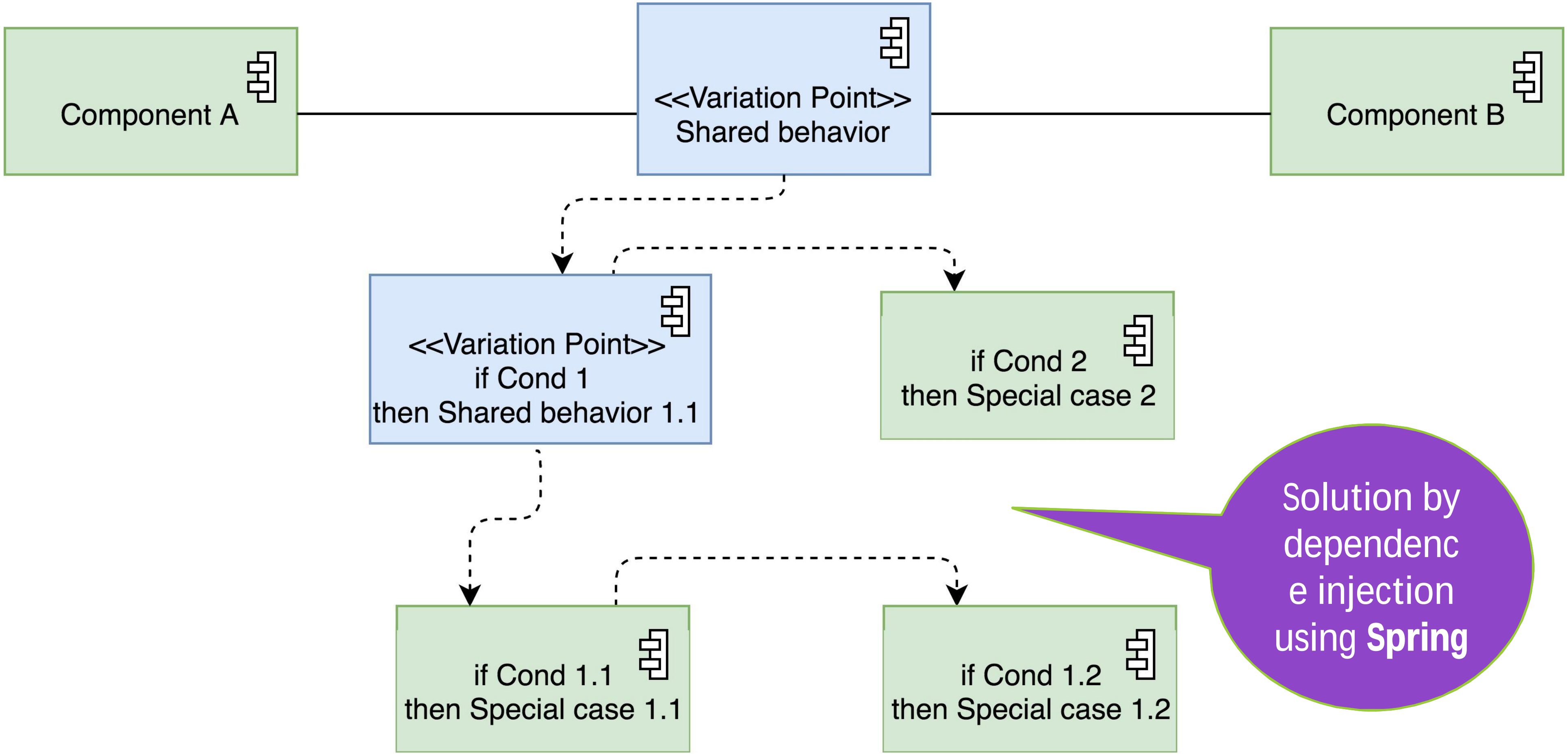


Variants :

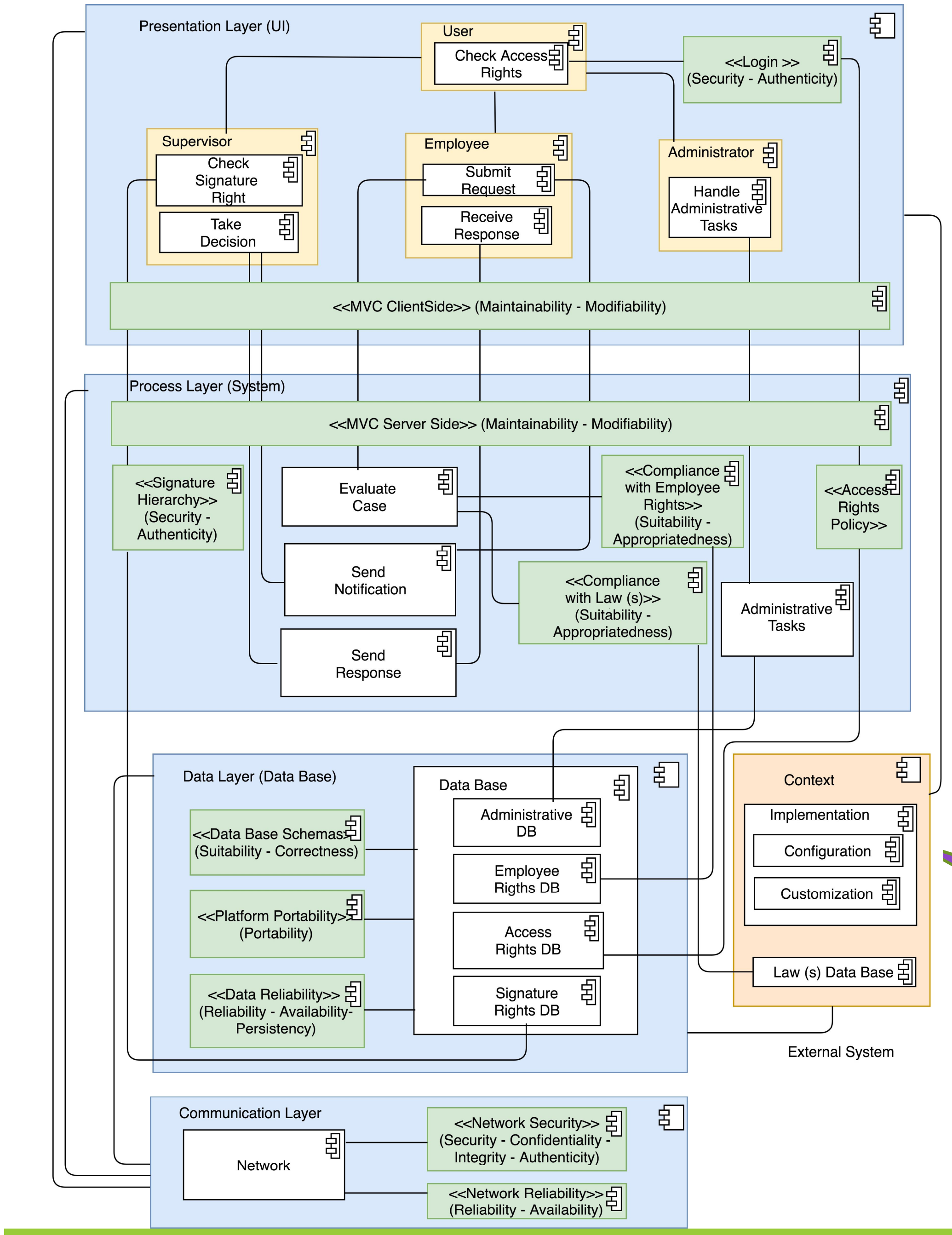
1 – Shared behavior
2 – Shared behavior + special case 1
3 – Shared Behavior + special case 2
4 – Shared Behavior + special case 1
+ special case 2

Variants sharing a behavior and having a conditioned special behavior

Defining variability



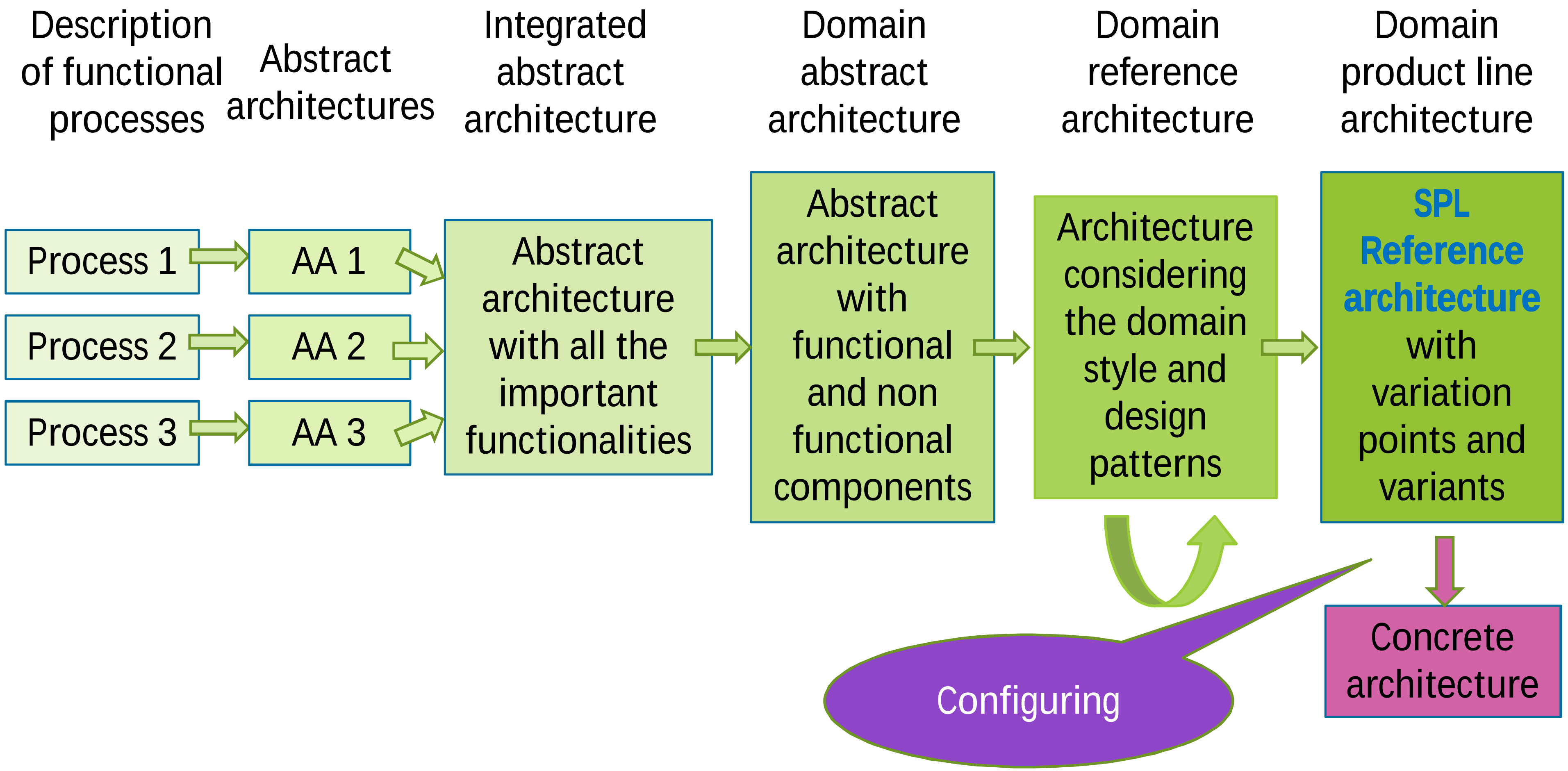
Solution by dependence injection using **Spring**


Presentation Layer (UI)
User
Check Access Rights
<<Login >> (Security - Authenticity)
Supervisor
Check Signature Right
Take Decision
Employee
Submit Request
Receive Response
Administrator
Handle Administrative Tasks
<<MVC ClientSide>> (Maintainability - Modifiability)
Process Layer (System)
<<MVC Server Side>> (Maintainability - Modifiability)
<<Signature Hierarchy>> (Security - Authenticity)
Evaluate Case
<<Compliance with Employee Rights>> (Suitability - Appropriatedness)
<<Access Rights Policy>>
Send Notification
<<Compliance with Law (s)>> (Suitability - Appropriatedness)
Administrative Tasks
Send Response
Data Layer (Data Base)
Data Base
Administrative DB
<<Data Base Schemas>> (Suitability - Correctness)
Employee Rigths DB
<<Platform Portability>> (Portability)
Access Rights DB
<<Data Reliability>> (Reliability - Availability- Persistency)
Signature Rights DB
Context
Implementation
Configuration
Customization
Law (s) Data Base
External System
Communication Layer
Network
<<Network Security>> (Security - Confidentiality - Integrity - Authenticity)
<<Network Reliability>> (Reliability - Availability)

Reference architecture of the product line with variability

Need for a configuration mechanism to select the variants

The proposed methodology



Description of functional processes

Abstract architectures

Integrated abstract architecture

Domain abstract architecture

Domain reference architecture

Domain product line architecture

Process 1

Process 2

Process 3

AA 1

AA 2

AA 3

Abstract architecture with all the important functionalities

Abstract architecture with functional and non functional components

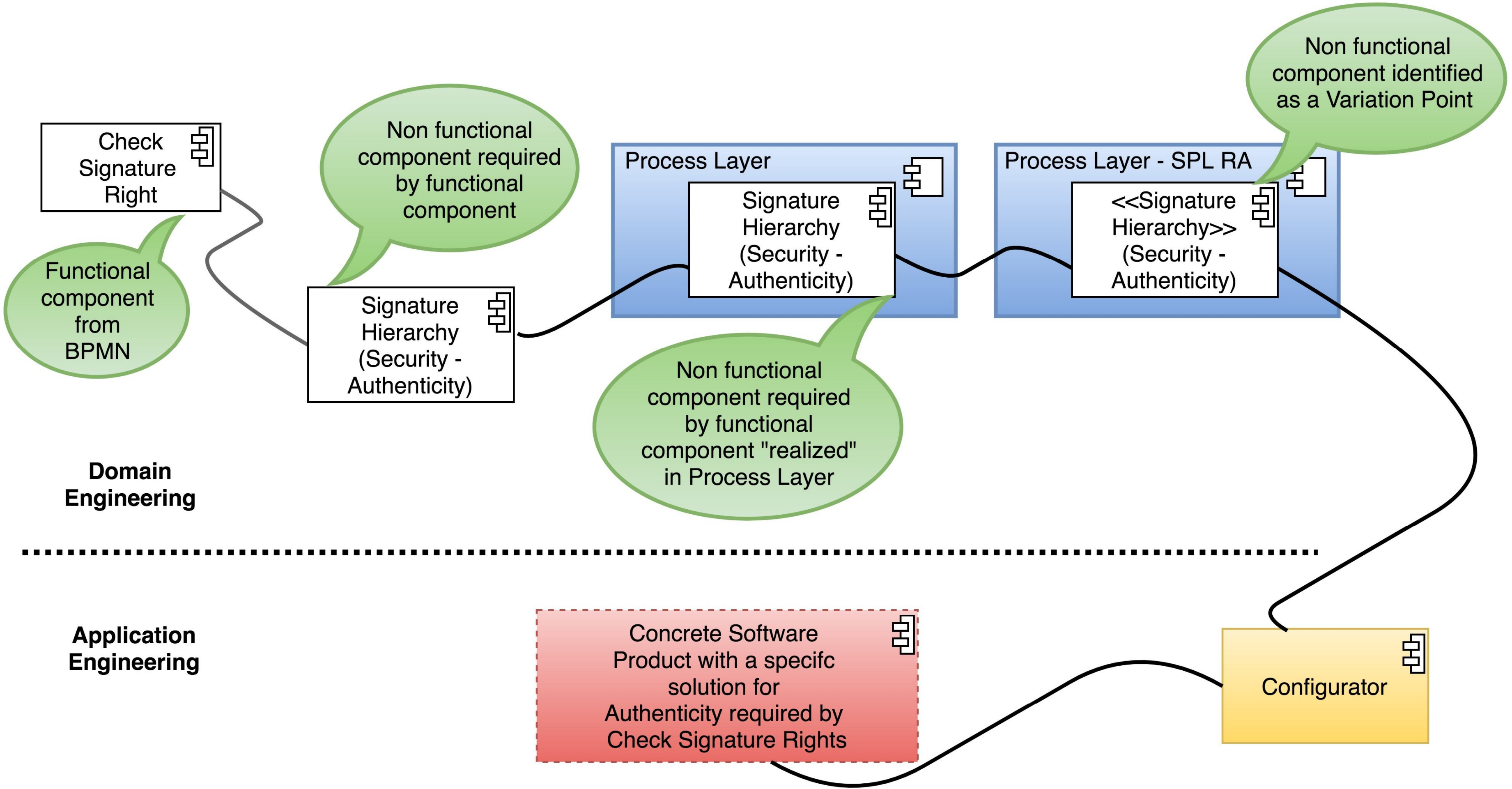
Architecture considering the domain style and design patterns

SPL Reference architecture with variation points and variants

Concrete architecture

Configuring

Traceability


Check Signature Right
Functional component from BPMN
Non functional component required by functional component
Signature Hierarchy (Security - Authenticity)
Process Layer
Signature Hierarchy (Security - Authenticity)
Non functional component required by functional component "realized" in Process Layer
Process Layer - SPL RA
<<Signature Hierarchy>> (Security - Authenticity)
Non functional component identified as a Variation Point
Domain Engineering
Application Engineering
Concrete Software Product with a specifc solution for Authenticity required by Check Signature Rights
Configurator

Conclusion

Starting from an existing system, we have reengineered the system architecture from interviewing domain experts

We identified functional components and their non functional requirements

The non functional requirements have been expressed as components

We identified the common core and the variants introducing variability

→ we obtained the

Software Product Line Reference Architecture

from an existing system with its variations points and variants

We have considered the suitability to legal requirements (laws and regulations) as a priority quality requirement, since they change often overtime. Our approach eases the modifiability thanks to traceability

Perspectives

We have studied only one of Berger-Levrault’s system

→ we will enhance our methodology studying various

Our objective is to built support tools

To facilitate the configuration, we have represented the reference architecture as an ontology in order to ease the transformations

Thank you !

Questions ?