Context Variability

Network on New Trends in Software Architecture and Variability

Dr. Rafael Capilla

rafael.capilla@urjc.es

GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES
AGENCIA ESTATAL DE INVESTIGACIÓN

- The challege for open variability models
- Adaptation and Context
- Programming techniques
- Case study

Part I

Open Variability

Universidad Rey Juan Carlos

What products you want to develop and how?

Universidad Rey Juan Carlos

Android’s features

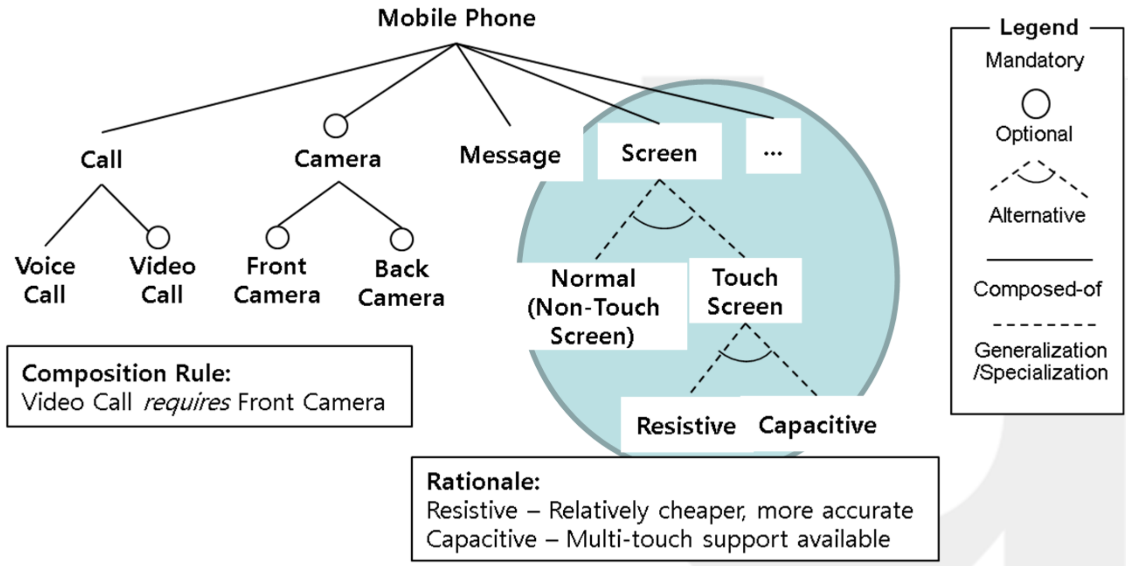
- Messaging
- Auto Correction and Dictionary
- Voice-based features
- Multi-touch
- Screen capture
- Video calling
- Multiple language support
- NFC
- Media support
- Etc…

Open Variability

- Different kind of features (e.g., mandatory, alternative, optional, external, etc.) define the visible properties of systems, from the “required variability” to the “provided variability”

“Required variability” ⟺ “Provided variability”

Notation makes provided variability explicit



Composition Rule:
Video Call *requires* Front Camera

Rationale:
Resistive – Relatively cheaper, more accurate
Capacitive – Multi-touch support available

Open Variability

- In most the cases the provided variability cannot be changed once the variability model is designed and the variability is implemented
- New customer’s needs often demand new features (differentiating features)
- New features often imply a redesign of the variability model (human and manual activity)
- Feature models are often closed to evolution

Open Variability

- Support for new features requires "**extensibility**" of existing feature models

- Open variability models are those prepared for the evolution for handling new customer features and preferably with minimal human intervention in the redesign task

- Open variability models are closer to variability managed at post-deployment time

Open Variability

- In most the cases the provided variability cannot be changed once the variability model is designed and the variability is implemented

  Payment Options *(Credit Card, PayPal)*

  Stripe payment platform (add new feature)

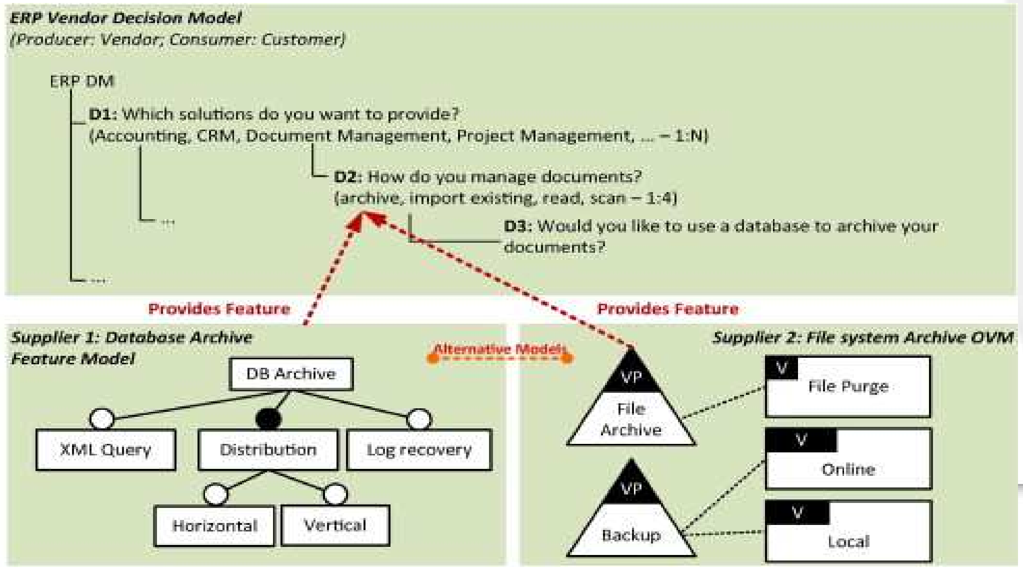
Option 1: Add the option manually and redesign the VP

Option 2: Add the feature automatically as a new value

Option 3: Implement a mechanism to add a feature dynamically

Open Variability

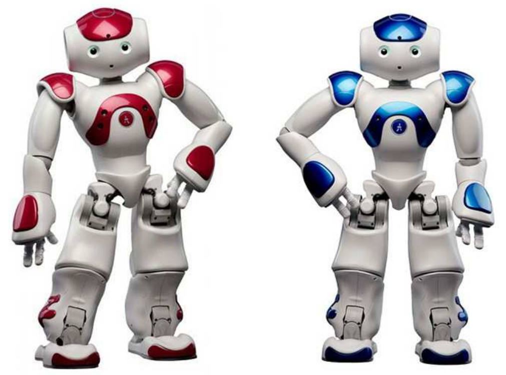
- **Differentiating functionality (challenges)**
  - Provide open variability models
  - The Product Line Architecture should be extensible
  - Support to unticipated requirements
  - Runtime facilities to manage feature updates
  - Innovative functionality to highlight our products from competitors


ERP Vendor Decision Model
(Producer: Vendor; Consumer: Customer)
ERP DM
D1: Which solutions do you want to provide?
(Accounting, CRM, Document Management, Project Management, ... – 1:N)
D2: How do you manage documents?
(archive, import existing, read, scan – 1:4)
D3: Would you like to use a database to archive your documents?
Provides Feature
Provides Feature
Supplier 1: Database Archive Feature Model
Alternative Models
Supplier 2: File system Archive OVM
DB Archive
XML Query
Distribution
Log recovery
Horizontal
Vertical
VP
File Archive
VP
Backup
V
File Purge
V
Online
V
Local

Part II

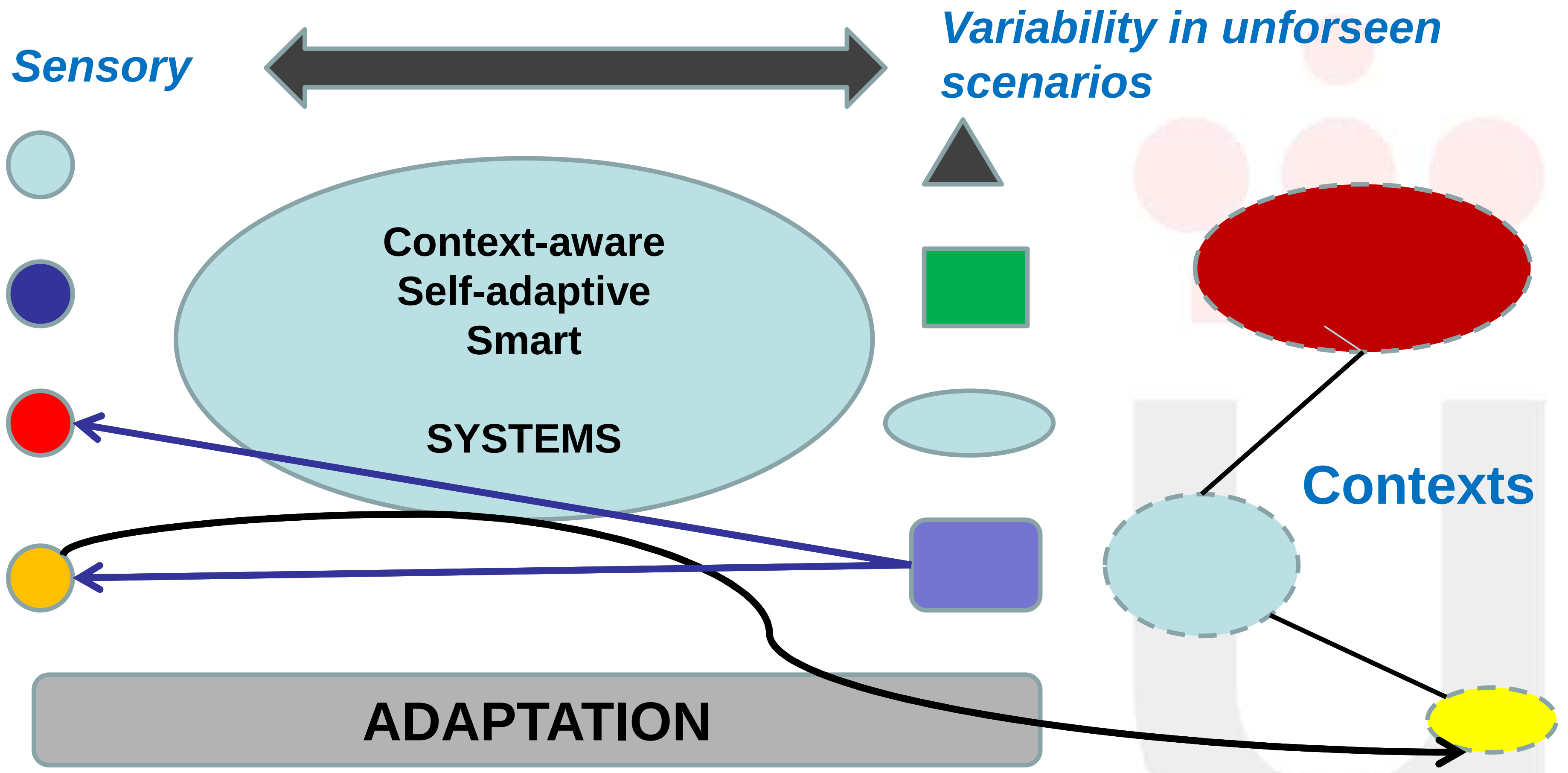
Adaptation and Context

Context-aware systems we want/need to build



Energy Management
Applications for Patients to monitor activity
POWER
HOSPITAL
BANK
Pollution Monitoring
Video Surveillance
Home Area Network
HOTEL
Transportation Monitoring
CITY HALL
City Services: Safety & Security / Emergency Responce
Traffic Monitoring
Parking Monitoring
SCHOOL
Communications

A combination of variability and awareness



Adaptation in Modern Systems Development

- Adaptation is a property where systems interacting with the environment >>>>>>>>> have the *ability to react (smartly)*

- Self-adaptation enhances the capability of adaptive systems to perform **autonomously** a change in their behavior

Mobile software offers highly configurable devices and system's options in the customer side

The **robotics area** needs of strong self-adaptive capabilities limited by tiny memory models that may hamper software updates during robot's execution

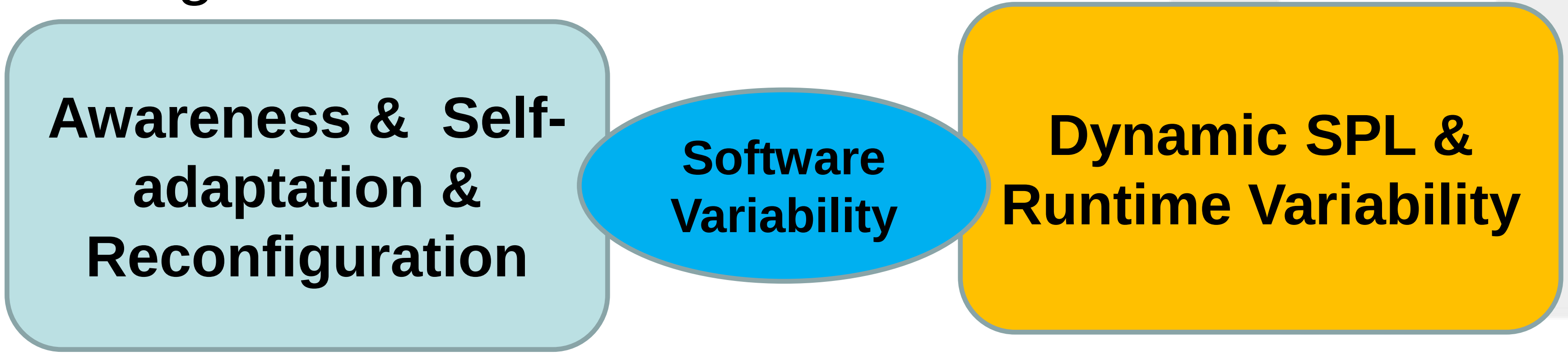
Autonomous Drones require unattended landing and take off

Challenges & Goals

Challenge 1: *Support awareness and context knowledge*

Challenge 2: *Changes in the structural variability supporting context properties (Runtime Evolution)*

Challenge 3: *Handle unmanaged (but supervised) changes*


Awareness & Self-adaptation & Reconfiguration
Software Variability
Dynamic SPL & Runtime Variability

Context-aware systems & Context variability modeling

- ***Awareness*** as a combination of **knowledge + monitoring**
- Advanced capabilities for monitoring, managing, and reacting to new context conditions, sometimes in unttended mode, are necessary
- Adaptation of systems should become…
  - Smart Reaction, Autonomous, Reconfigurable,
  - ***All in one***….*Better self-adaptable*

Context-aware systems & Context variability modeling

A variety of Systems-of-Systems and mission critical systems (e.g., *industrial automation, airport mgmt, systems, robots, smart cities*) use **context information** gathered from physical sensors

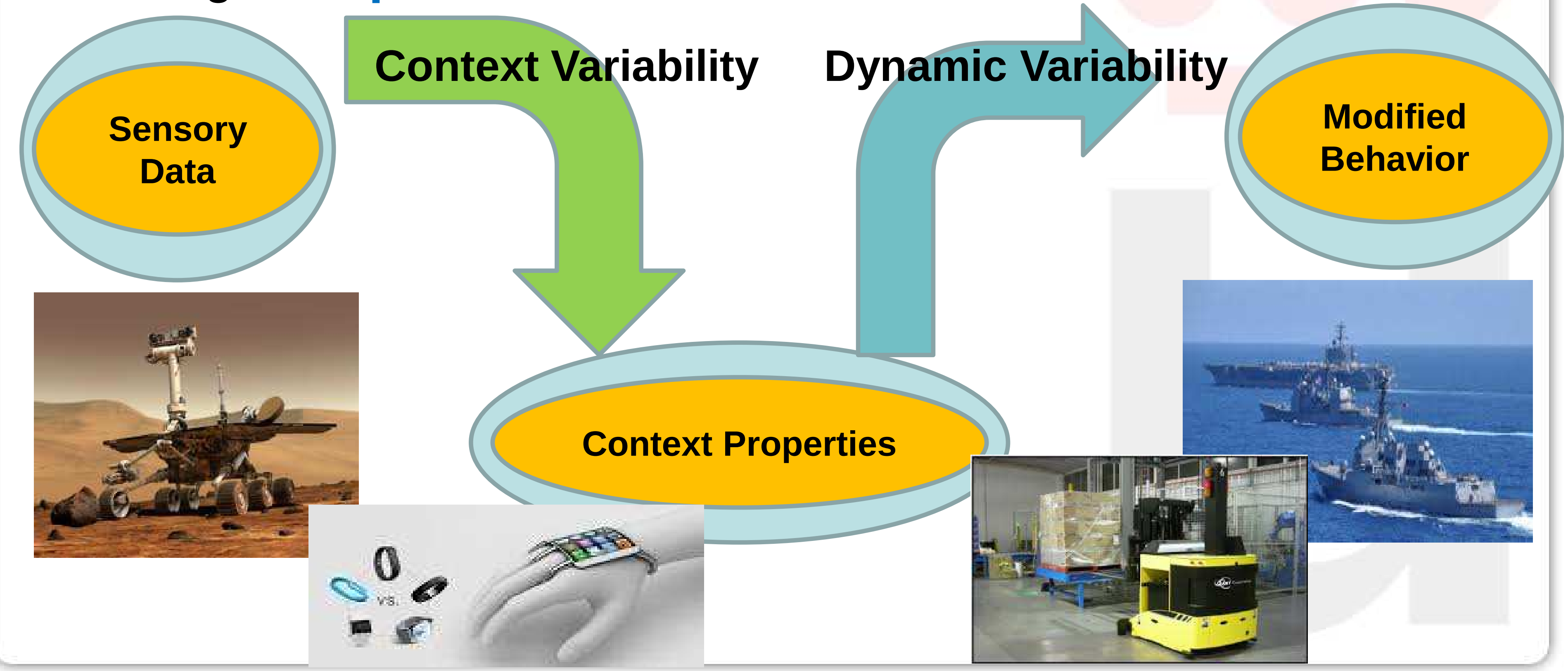
Advanced capabilities should include (among others) "adding/removal/replacement" of CONTEXT FEATURES

E.g., A Smart City Mgmt. System could add dynamically and (semi-)automatically a new functionality (…but supervised)

Universidad Rey Juan Carlos

Context Variability Modeling

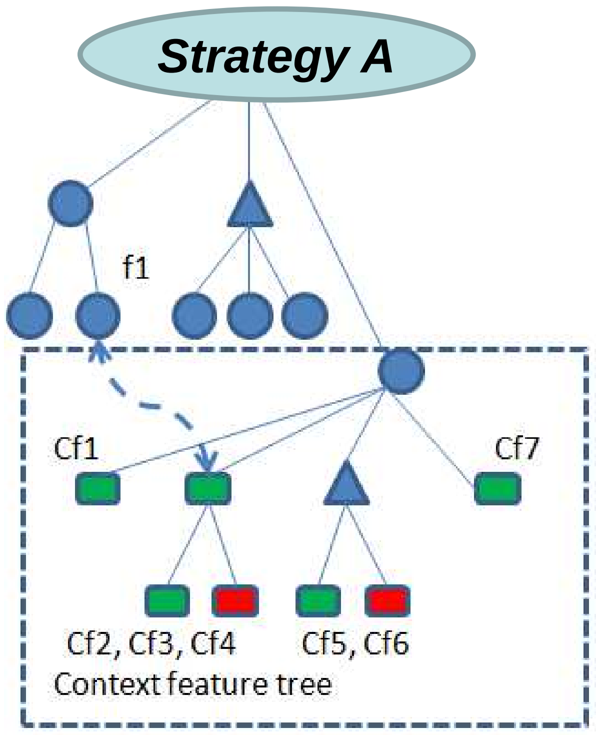
- Context properties for **autonomous decision-making**
- Variable options for **multiple and optimal decisions**
- Manage **“unpredictable” scenarios**

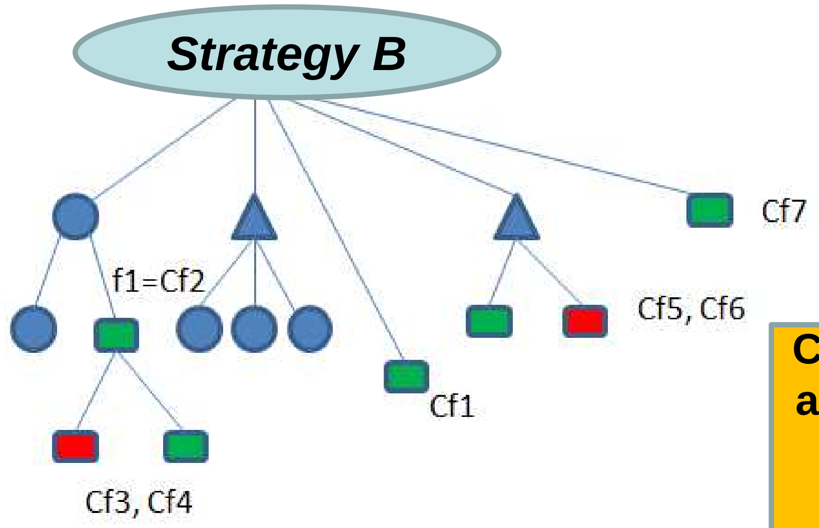


DSPL: Context Analysis

- Identify "context features" + "non-context features"
- Context features are used to model and manage the "awareness" property of context-aware systems
- **Context Variability** is understood as a **branch** of Feature Modeling where "**context properties**" are modeled jointly or separately with non-context features

Context feature modeling strategies





Context features entrangled in the FODA tree with non-context features

Context features are annotated in the FM and Database

Features that change dynamically can be annotated too

Legend

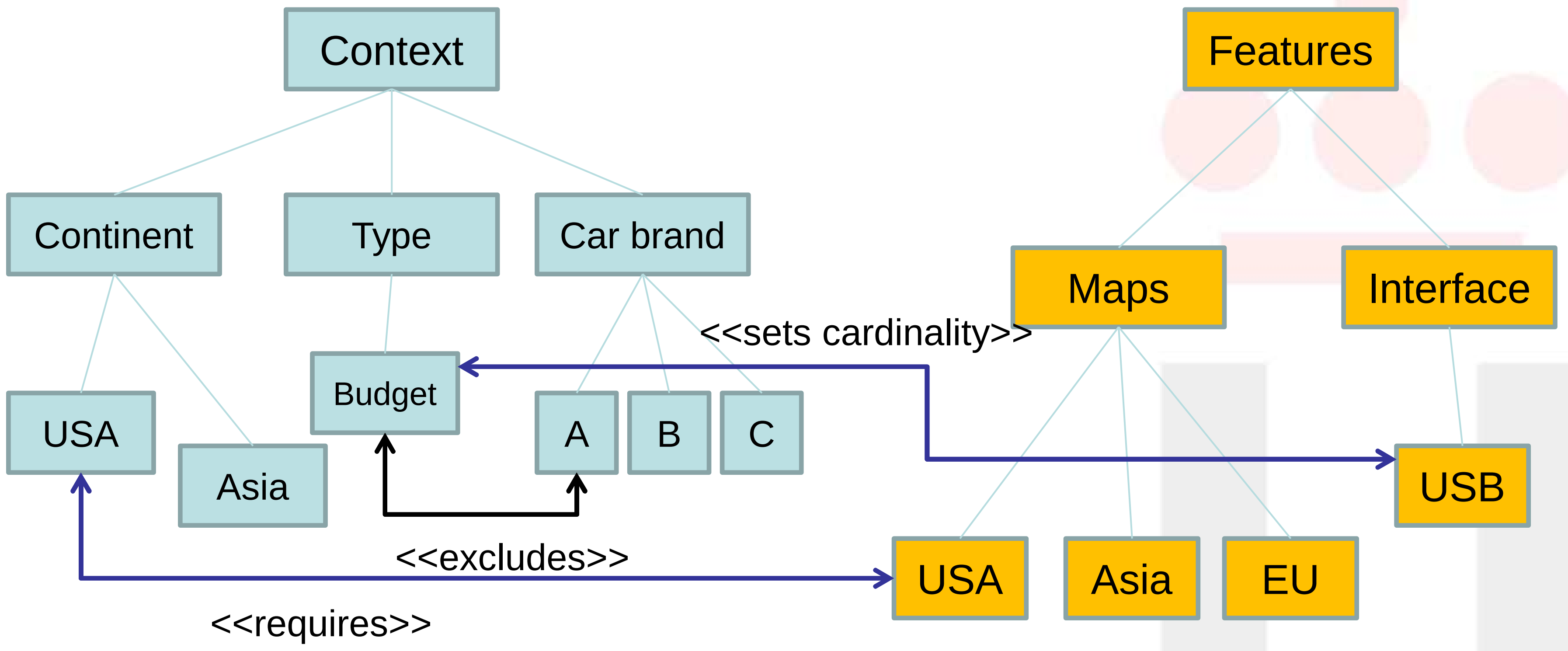
●	Feature o variant
▲	Variation point
▬	Context feature
🟩	Context feature activated
🟥	Context feature deactivated

Taxonomy of context features

Type 1: Cf1, Cf2
Type 2: Cf3, Cf4, Cf5
Type 3: Cf6, Cf7
Compatiple types: Type 1, Type 3

Universidad Rey Juan Carlos

Context features and feature modeling

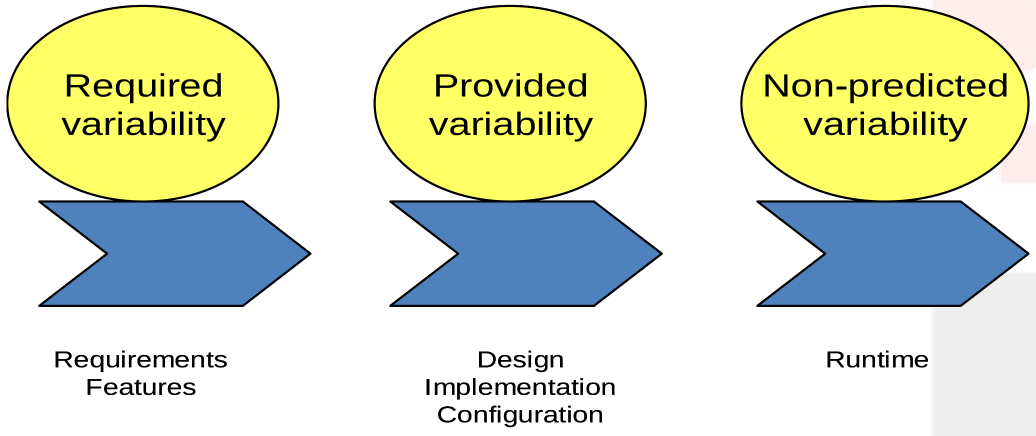


Example: the DVD systems for European cars Region 1 [Hartmann et al, SPLC'08]

```
Continent.Europe <<requires>>
European Maps {Rationale:obviuos}
Continent.Europe <<requires>> DVD Region 1 {Rationale:standard}
```

Dynamic (Runtime) Variability

Runtime or dynamic variability *offers a good choice for systems that experience dynamic changes in their quality and context*

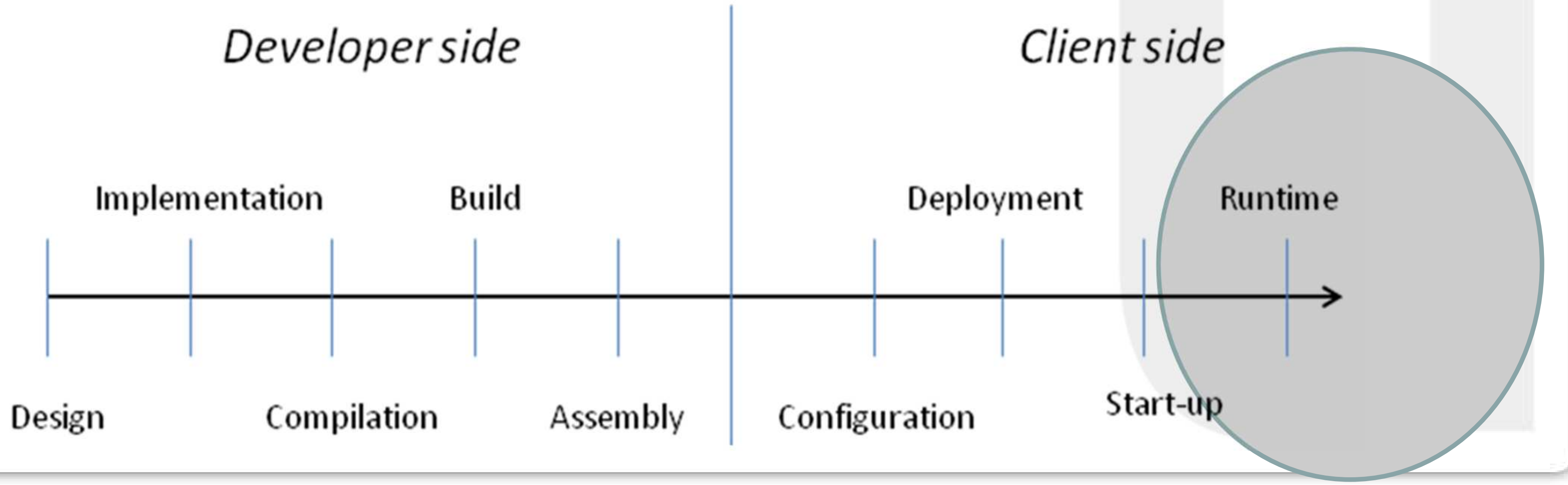


- **Design time variability** is (often) hidded to the user and managed by the product's developer,
- **Dynamic variability** is an open model managed in the customer side

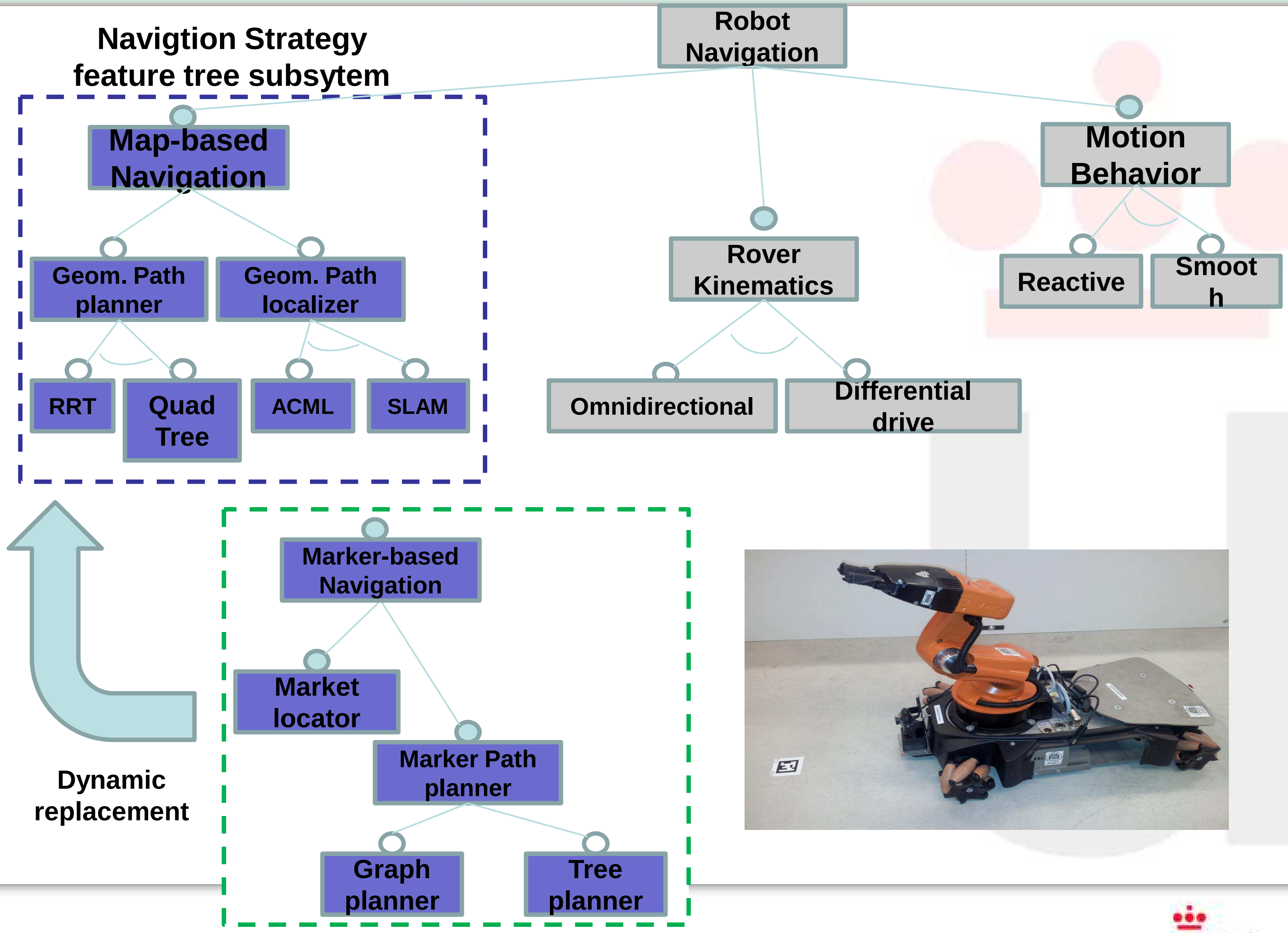
Dynamic Variability: Runtime Challenges

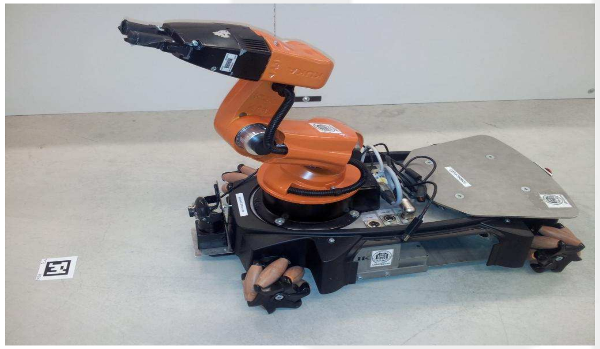
Activating and deactivating system's options is not enough

- Modify the structural variability at runtime
- Automate runtime validation and checking (new constraints!!!)
- Automatic (re)deployment and (re)binding of system configuration with minimal interruption



Dynamic Variability in Robotic Systems (replacing a full branch)





Part III

Programming Techniques

Universidad Rey Juan Carlos

Implementation techniques

- **Delegation**: Support optional features
- **Design patterns**: Decouple variability
- **Context-oriented programming (COP):** functionality is activated using layers
- **Dynamic link libraries/classes**: in support of configurable options at runtime and actívate options dynamically

Delegation

- A functionality is delegated (performed) in another component or function
- Delegation is suitable to implement optional features but has problems with alternative features

Example: Object A implements the functions A1, A2 and A3. **A1 is mandatory** for all product line members while **A2 is optional** and supported only by extended products. **A3 is alternative** meaning that an extended product provides an extended implementation of A3

Delegation requires that all methods be defined.A1 is implemented in all products. A2 is defined in all products but is really implemented only in the extended versions. A3 is defined in all products and implemented differently in different versions.

When A2 is invoked in the standard version an exception could be raised. The developer of the extended version could then implement A2 in a delegation class.

Universidad Rey Juan Carlos

Delegation

```
public void A2() {
            delegationObject.A2();
}
```

Delegation with alternative features

```
public void A3() {
  /* standard code */
  foo1();
  foo2();
/* here comes an optional part that gets delegated */
   delegationObject1.foo3();
}
```

(Implementing product lines variabilities, Anastopoulos and Gacek)

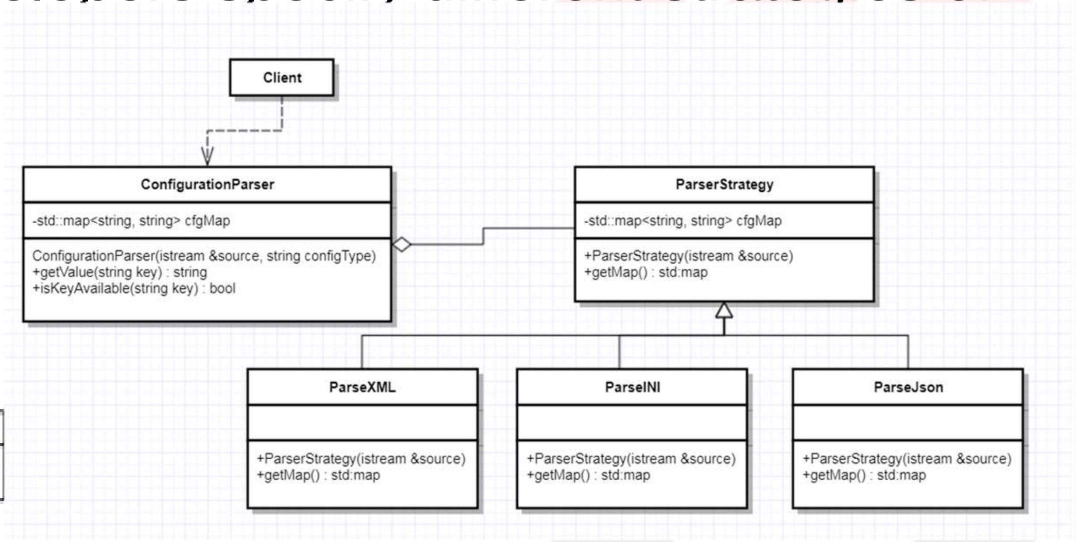
Universidad Rey Juan Carlos

Design Patterns

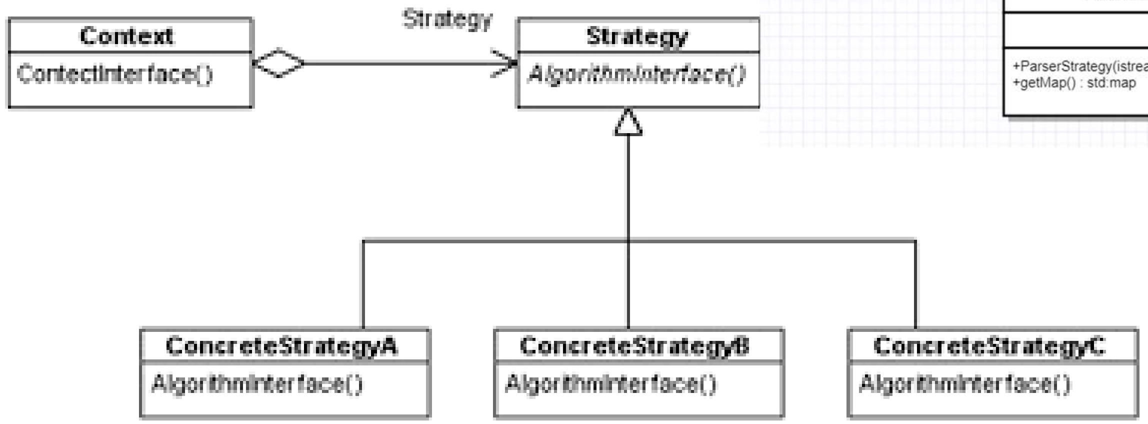
- Variability is often scattered using parameters and code is undisciplined
- DPs have evolved to decouple variability and where classes collaborate together
- Examples of patterns: Strategy, Decorator, Observer, Template-method, etc.
- E.g. The Strategy pattern is suitable to implement alternative features with different implementations

Design Patterns

- Strategy pattern: typically used to implement alternative features (e.g. variability in algorithms). It’s a behavioral pattern that uses delegation and where developers specify different strategies of use.



Client
ConfigurationParser
-std::map<string, string> cfgMap
ConfigurationParser(istream &source, string configType)
+getValue(string key) : string
+isKeyAvailable(string key) : bool
ParserStrategy
-std::map<string, string> cfgMap
+ParserStrategy(istream &source)
+getMap() : std:map
ParseXML
+ParserStrategy(istream &source)
+getMap() : std:map
ParseINI
+ParserStrategy(istream &source)
+getMap() : std:map
ParseJson
+ParserStrategy(istream &source)
+getMap() : std:map



Design Patterns

- Implementation of the Strategy pattern

```
interface BillingStrategy {
// Use a price in cents to avoid floating point round-off error int
getActPrice(int rawPrice);

// Normal billing strategy (unchanged price)
static BillingStrategy normalStrategy() {
   return rawPrice -> rawPrice; }

// Strategy for Happy hour (50% discount)
static BillingStrategy happyHourStrategy() {
 return rawPrice -> rawPrice / 2; } }
```

Context Oriented Programming Languages (COP)

- COP languages are an alternative to model “contexts” that can be activated dynamically
- A variety of COP languages exist (JCOP, ContextLua, EventCJ, Subjective-C, ContextJS)
- Context dependent behavior and variations
- Layered and non-layered COP languages (layers vs contexts)
  - **Layer-based**: contexts are identified as layers that are activated with respect to context changes
  - **Non-layer**: contexts are reified as an object used in the method invocation. Context behavior is not modularized using layers

Universidad Rey Juan Carlos

COP

- COP are research languages that use the notion of layers to activate a functionality dynamically
- It’s a programming technique that supports context-dependent behavioral variations
- Layers are first-class entities that can be invoked at runtime
- Suitable for context-awareness systems
- Examples of COP: ContextL, ContextJ, ContextS, Subjective-C, JCOP, PyContext, ContextJS, etc.

Example with COP

- Context-traits.js

```
var cop = require('context-traits');
var Untrusted = new cop.Context();
var LowBattery = new cop.Context({
              name: 'low battery',
              description: 'The remaining battery charge is low.'});

window.addEventListener('batterystatus',
function (info) {
 if (info.level <= 30) LowBattery.activate();
   else LowBattery.deactivate(); })
```

Context creation

Context detection

Example with COP

Subjective-C

UILabel class

drawTextInRect:

Draws the receiver’s text in the specified rectangle.

```
- (void)drawTextInRect:(CGRect)rect
```

Parameters

rect

The rectangle in which to draw the text.

Discussion

You should not call this method directly. This method should only be overridden by subclasses that want to modify the default drawing behavior for the label’s text.

Availability

Available in iOS 2.0 and later.

Declared In

`UILabel.h`

Universidad Rey Juan Carlos

Example with COP

Subjective-C

Application Behaviour

Open classes
Objective-C

COP
Subjective-C

```
@implementation UILabel (color)
@context(Landscape)
- (void)drawTextInRect:(CGRect)rect {
  self.textColor = [UIColor greenColor];
  [superContext drawTextInRect:rect];
}
@end
```

- ✓ Adaptation of any existing component
- ✓ No access to original source code needed
- ✓ Adaptations can be cleanly modularized

Example with COP

Subjective-C

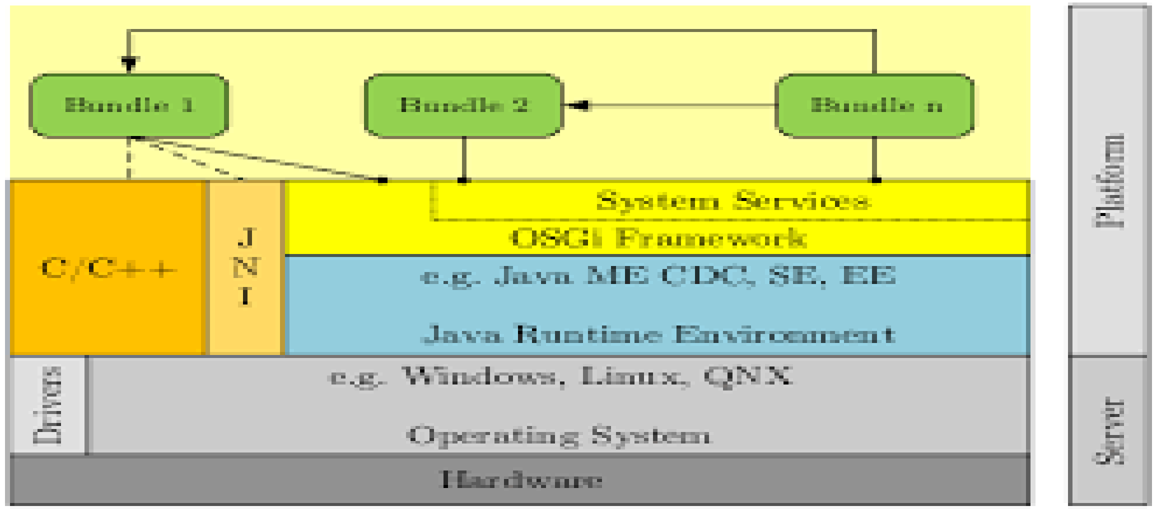
```
if motion.isGyroAvailable {
   self.motion.gyroUpdateInterval = 1.0 / 60.0
   self.motion.startGyroUpdates()

   self.timer = Timer(fire: Date(), interval: (1.0/60.0),
          repeats: true, block: { (timer) in
     // Get the gyro data.
     if let data = self.motion.gyroData {
       let x = data.rotationRate.x
       let y = data.rotationRate.y
       let z = data.rotationRate.z
       if(getDegrees(y) > 270 && getDegrees(y) < 360 ) {
                                                        @activate(Landscape)
                                                       } else { @deactivate(Landscape) }
      }
    })
    // Add the timer to the current run loop.
    RunLoop.current.add(self.timer!, forMode: .defaultRunLoopMode)
  }
@end
```

Dynamic Link Libraries/Classes

- At runtime a DLL containing libraries or classes can be invoked to include new variants
- Dynamic files can be uploaded in support of a new system's configuration
- Features and configurations can be replaced dynamically

Example: the OSGi platform uses bundles (a group of Java classes) can be remotely installed, started, stopped, updated, and uninstalled without stopping the system.



Dynamic Link Libraries/Classes

- Dynamic loading in Unix (Dlsym)

```
void* sdl_library = dlopen("libSDL.so", RTLD_LAZY);
if (sdl_library == NULL)
{ // report error ... }
else
{ // use the result in a call to dlsym }
```

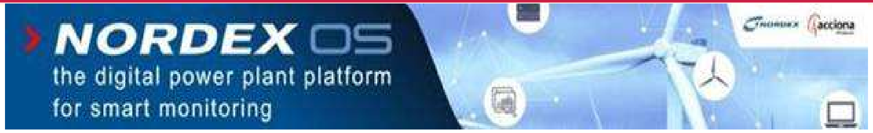
- JavaScript

```
//dynamically load and remove"javascript.php" as a JavaScript file
loadjscssfile("javascript.php", "js")
removejscssfile("somescript.js", "js")
```

Part IV

Wind farm case study

Wind farm case study



- NORDEX is a German company and partner of Acciona (Spain)
- NORDEX develops and deploy Wind power using Wind Turbines in different countries
- NORDEX is one of the largest manufacturers of wind turbines

- It counts with at least 3 series of wind turbines encopassing different models and configurations
- Turbines are customized to each customer’s need (from Africa to Finland)

Wind farm case study

- We have an ongoing collaboration between NORDEX-URJC-University of Hamburg
- Several changes and configurations on an already existing system
- Specific conditions depending on the countries being deployed with a lot of specifications and fluctuations of the wind turbines
- Around 130 base configurations and the number of grid codes are at least 100
- One variant for every country

Universidad Rey Juan Carlos

Wind farm case study

- Variability at NORDEX
  - 3 different tower variants (not software)
  - Advanced anti-icing system
  - Climate operating range
  - Power consumption modes
  - Condition monitoring system
  - Sound reduction from the rotors
  - Grid codes
  - Country specifications
  - 130 base configurations

Wind farm case study

- Challenges:
  - Software adaptations for every project (lot of effort to be coded and tested)
  - Lot of variants for grid codes that must be satisfied and configure the windfarm controllers more automatically
  - The code is not standardized for different countries and effort for different controllers and signals to the wind turbines
  - Specific conditions depending on the countries being deployed with a lot of specifications and fluctuations of the wind turbines

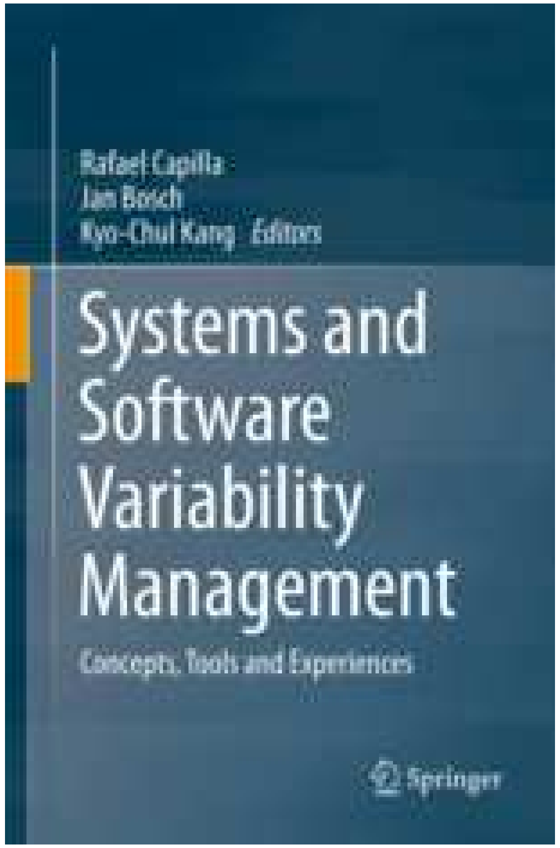
Wind farm case study

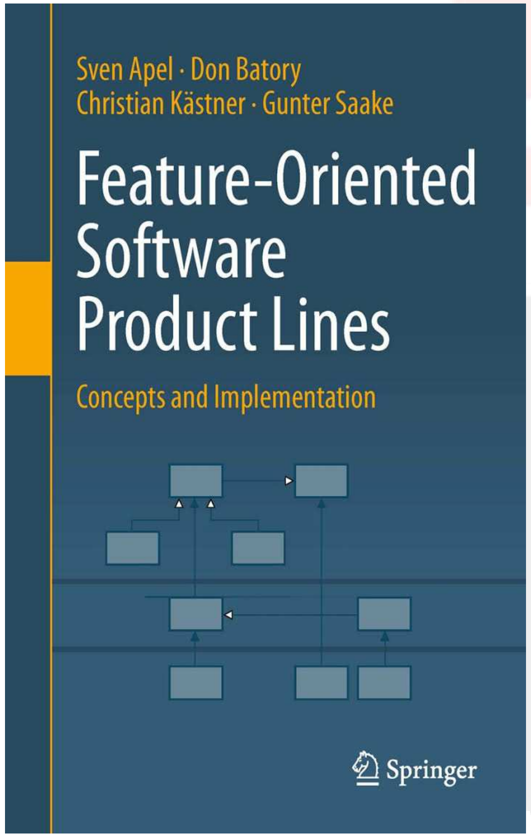
- Goals:
  - Adapt the software by reconfiguration and not writing new code and reduce the engineering effort
  - The most important features is to control the algorithms
- Solutions (R+D)
  - Develop a new method to identify and model context features
  - Improve the reconfiguration processes to provide automatic support (dynamic variability)
  - Reduce the engineering and configuration effort via the two previous solutions

Universidad Rey Juan Carlos

Further reading

- Two contemporary books (2013)


Rafael Capilla
Jan Bosch
Kyo-Chul Kang Editors
Systems and Software Variability Management
Concepts, Tools and Experiences
Springer


Sven Apel · Don Batory
Christian Kästner · Gunter Saake
Feature-Oriented Software Product Lines
Concepts and Implementation
Springer