Supporting Reuse With Aspects

CDI, Software Product Lines and Apect-Oriented Change Realization

Software Component

Software component **is unit of composition** with explicitly determined (approval) and required interfaces and dependencies. Similarly, it can be independently deployed and is **subject of composition** performed by third parties. Component has no externally observable state.

Source: SZYPERSKI, Clemens, 2002. Component Software : Beyond Object-Oriented Programming. ISBN 0-201-745572-0.

GOOD TO ACHIEVE REUSE?

Composition Of Components

SOA

Services are composed based on
on Loose coupling

Architectural level

Aspects

Aspects are composed with the rest of the code

Implementational level

Microservices

Micro-frontend

Using orchestrator like Docker Swarm or Kubernetes

Architectural level

Domain Knowledge

Timeless compression of mental models of end users and other stakeholders

Mental models whose patterns are tacidly driven by commonality and variation

Source: James O. Coplien and Gertrud Bjrnvig. 2010. Lean Architecture: for Agile Software Development. Wiley Publishing.

Introducing Software Product Lines

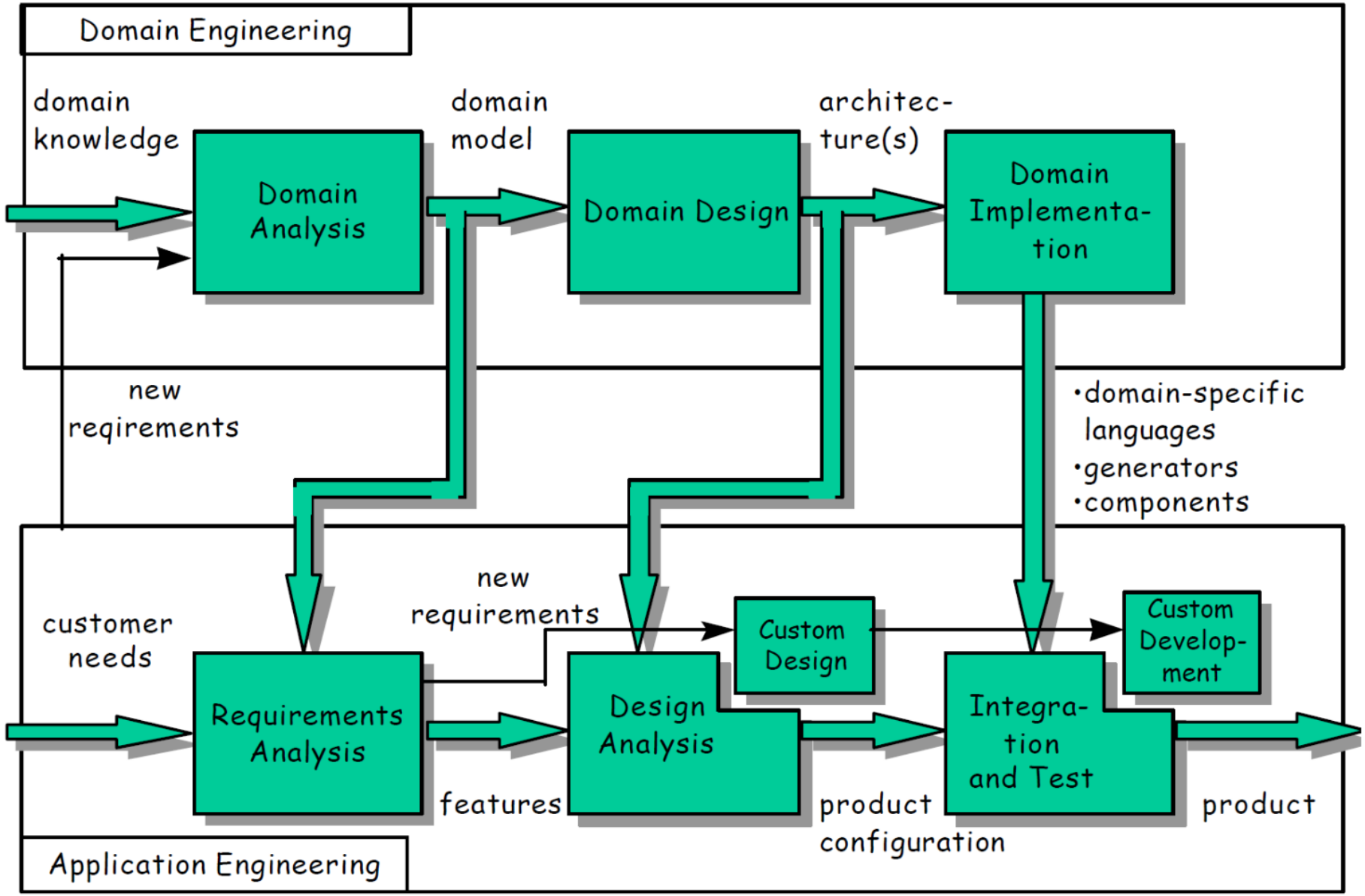
Evolutionary ⟺ Revolutionary

Existing Set of Products ⟺ New Set of Products

	Revolutionary	Evolutionary
No product base	Development of a new product line before delivering the first product	Gradual development of a product line during which products are being delivered
Existing product base	Development of a new product line out of the existing set of products before delivering the first product	Gradual development of a product line out of the existing set of products

Source: https://poetisania.com/val/aosd/index.html

J. Bosch. Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. Addison-Wesley, 2000.

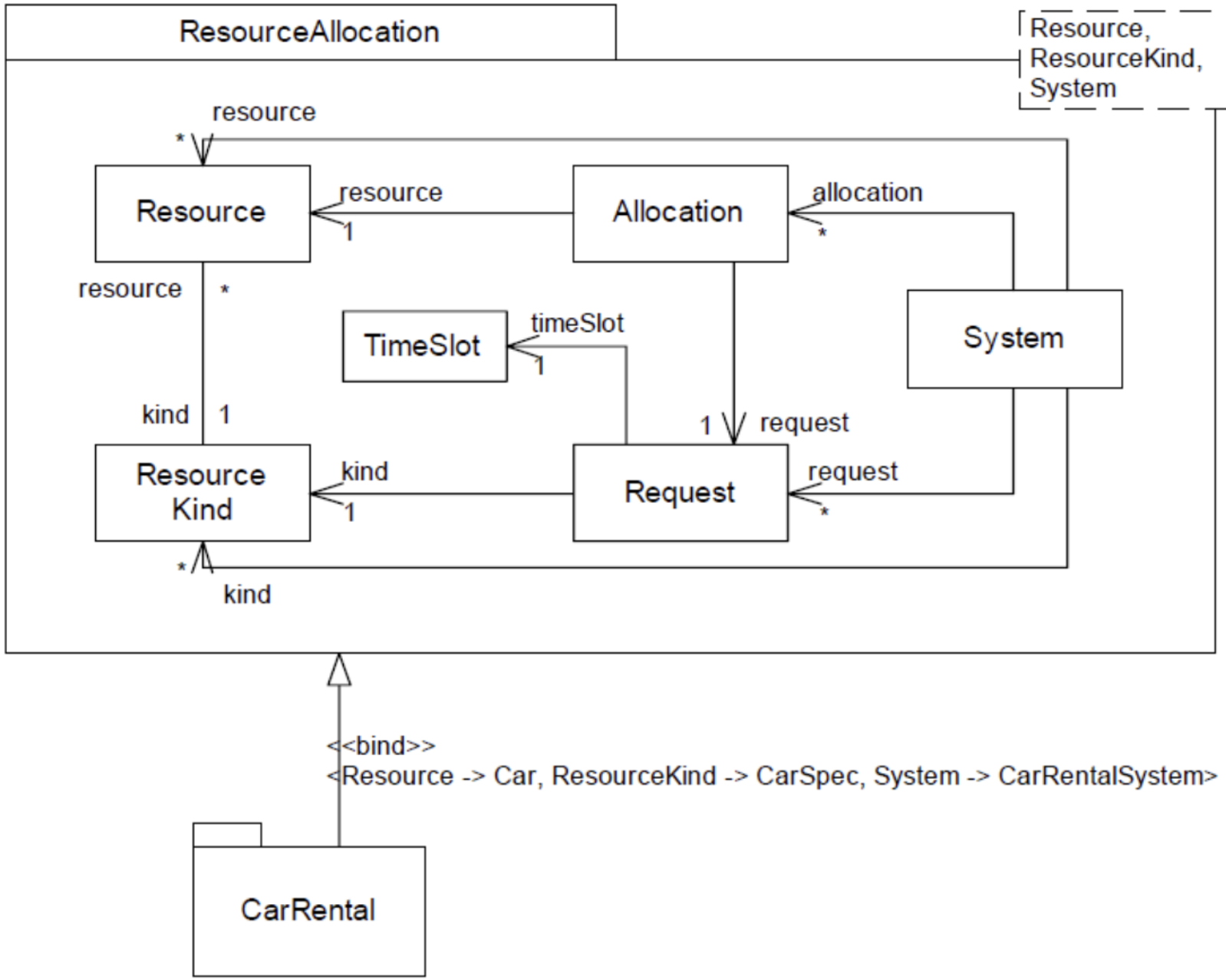


Krzysztof Czarnecki. Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models. PhD thesis, TU Ilmenau, Germany, 1999. (Attributed to SEI.)

Table 1. Product line activities and associated requirements on implementation technologies

Activity		Effort	Factor
Framework Engineering	Implementing reusable code	Effort for making code reusable across the product line (development for reuse)	Reuse techniques
			Variation types
			Granularity levels
		Effort for testing reusable code	Testability
	Reacting to evolutionary change	Effort for integrating system-specific code into the product line	Component inte-gration impact
		Effort for adding and removing variations (variability management)	Automation
		Maintenance effort	Reuse techniques
Application Engineering	Reusing code	Effort for reusing code to derive a concrete product (development with reuse)	Reuse techniques
	Resolving variations	Effort for creating a concrete product line member	Binding time
			Automation

Source: Anastasopoulos, M. and Muthig, D. (2004) 'An Evaluation of Aspect-Oriented Programming as a Product Line Implementation Technology', in J. Bosch and C. Krueger (eds) *Software Reuse: Methods, Techniques, and Tools*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 141–156.



OMG. OMG Unified Modeling LanguageTM (OMG UML), Superstructure. Version 2.4.1, 2011.

Use of ECaesarJ

Why?

Insufficient mechanisms in OOP for modularizing the features

Only **for individual objects/classes**

Multiple affected objects/classes by Features

Large-scale extension mechanism

PROVIDES LARGE-SCALE SELECTION AND COMPOSITION MECHANISMS

VIRTUAL CLASSES

-inner classes
-late-bound instantiation
-can be refined in subclasses of enclosing class
-as family members
- members of instances of the enclosing class [family objects/classes]

PROPAGATING THE MIXIN COMPOSITION

Composition propagates into virtual classes

ALL INHERITED DECLARATIONS OF VIRTUAL CLASSES WITH THE SAME NAME ARE MERGED AUTOMATICALLY

Implementation in ECaesarJ

Feature
In family class

feature-oriented modularization of software product lines

Domain objects
In virtual classes

To not confuse this with Java **class**
-to use it as extension ofJava

cclass

New CaesarJ dat structure

Taken from: *Rashid, A., Royer, J.C., Rummler, A. (eds.): Aspect-Oriented, Model-Driven Software Product Lines: The AMPLE Way (09 2011)*

```
cclass HouseStructure {
    abstract cclass Location { }

    abstract cclass CompositeLocation extends Location {
        abstract List<? extends Location> locations();
    }

    cclass Room extends Location { }

    cclass Floor extends CompositeLocation {
        List<Room> rooms;
        List<Room> rooms() { return rooms; }
        void addRoom(Room r) { rooms.add(r); }
        List<? extends Location> locations() { return rooms(); }
    }

    cclass House extends CompositeLocation {
        List<Floor> floors;
        List<Floor> floors() { return floors; }
        void addFloor(Floor r) { floors.add(r); }
        List<? extends Location> locations() { return floors(); }
    }

    House house = new House();
    House house() { return house; }
}
```

Figure 6.3 Implementation of a house structure as a family class.

The DCI Architecture: A New Vision of Object-Oriented Programming

- ***Vision to capture the end user cognitive model***

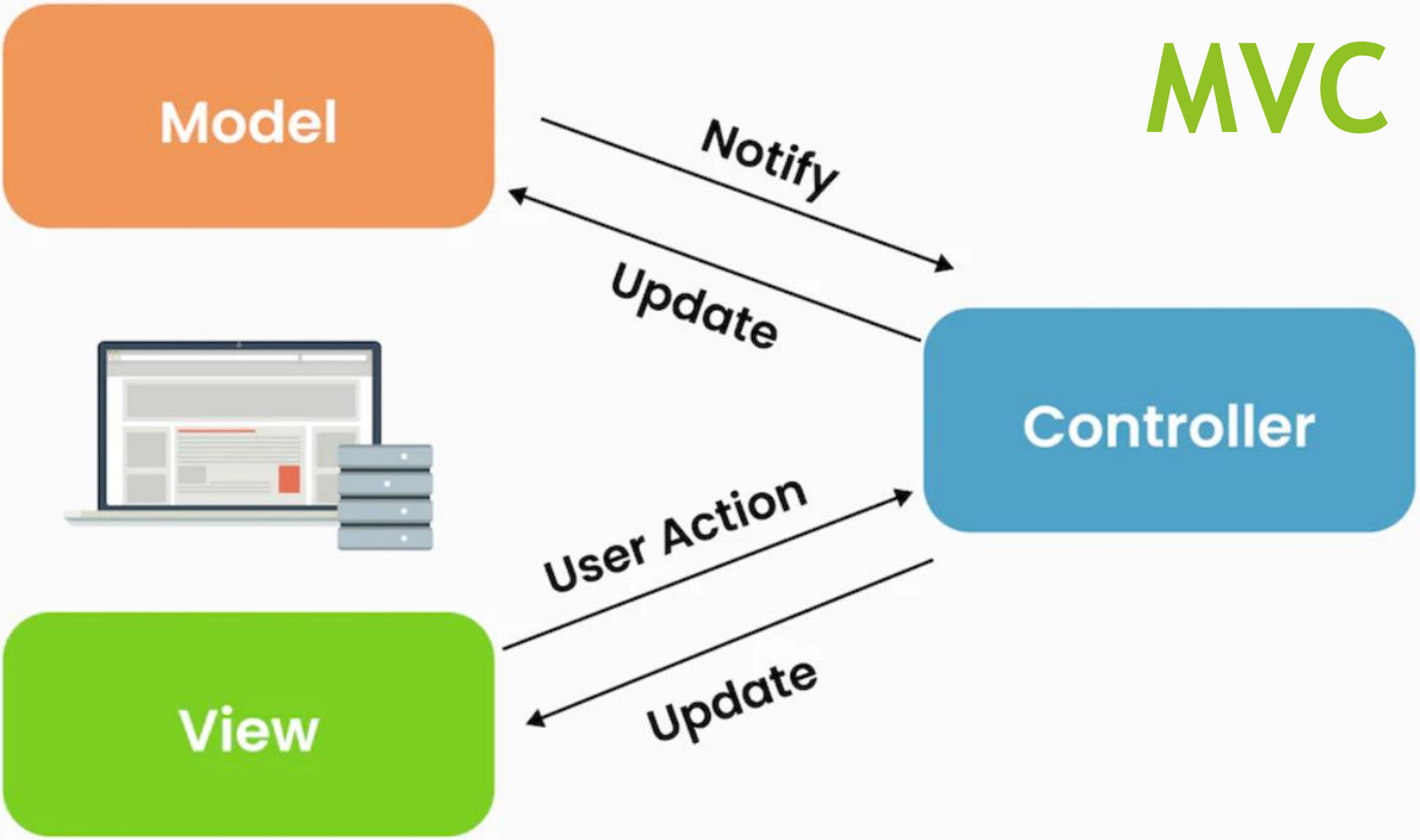
  Model of roles and interaction between these roles

- **A way how to combine roles, algorithms, objects, and associations between them to provide a stronger mapping between the code and the end-user mental model**

Objects capture structure, but fail to capture system action

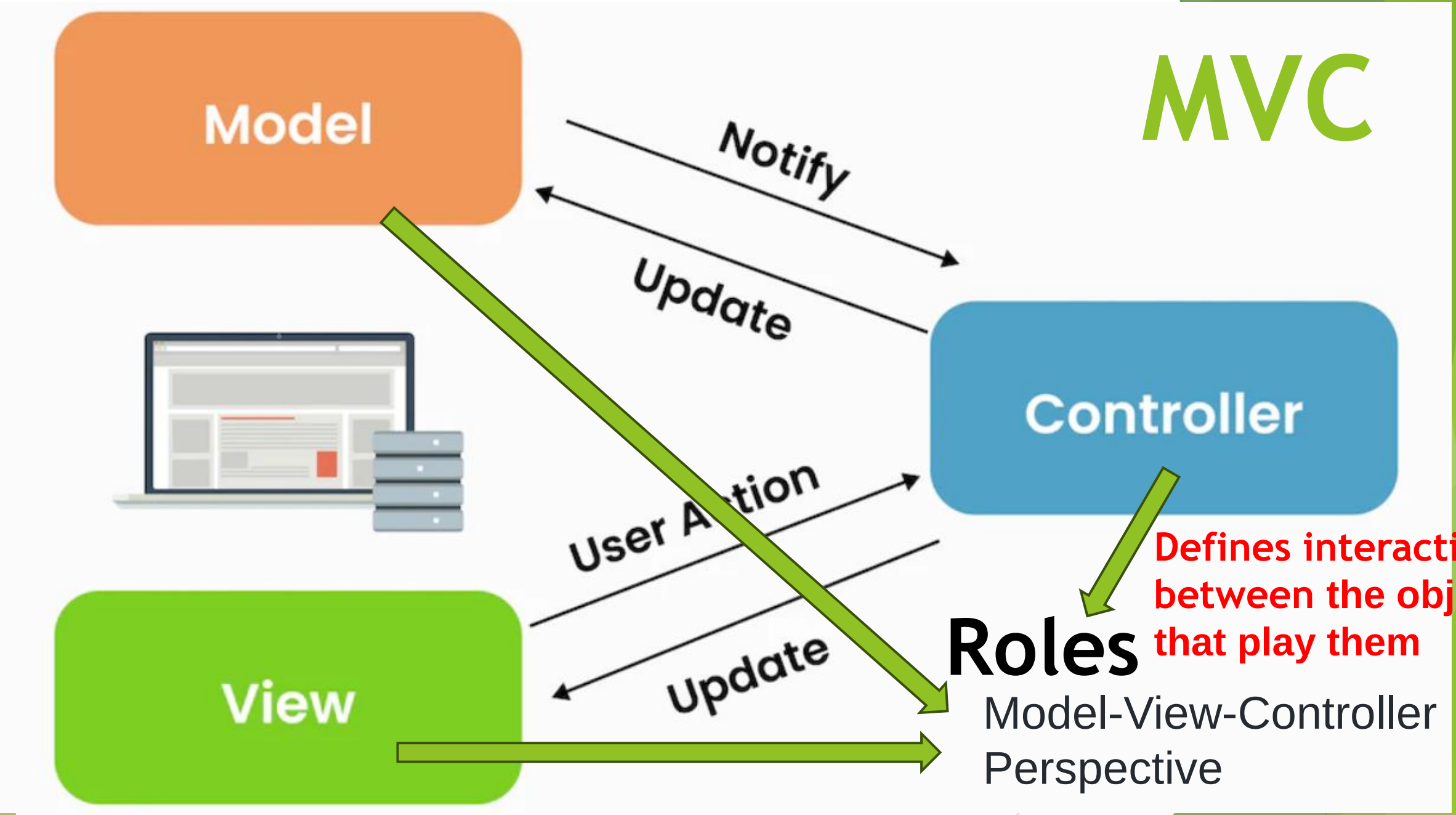
Source: https://www.artima.com/articles/the-dci-architecture-a-new-vision-of-object-oriented-programming

MVC



Source: https://www.linkedin.com/pulse/model-view-controller-architecture-programming-aditira-jamhuri

MVC



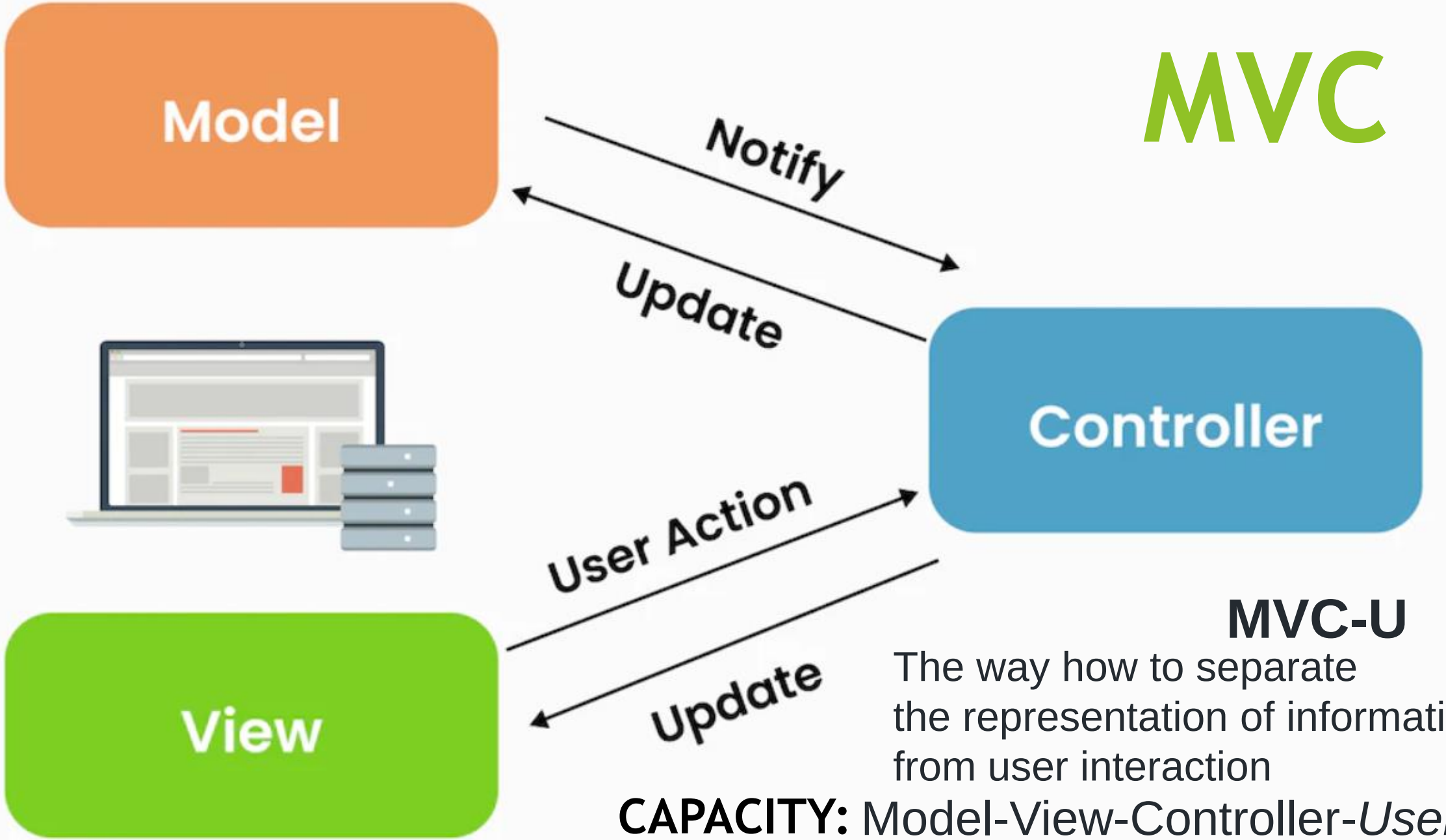
Defines interaction between the objects that play them

Roles

Model-View-Controller Perspective

Source: https://www.linkedin.com/pulse/model-view-controller-architecture-programming-aditira-jamhuri

MVC



MVC-U

The way how to separate the representation of information from user interaction

CAPACITY: Model-View-Controller-*User*

Source: https://www.linkedin.com/pulse/model-view-controller-architecture-programming-aditira-jamhuri

MVC-U

Interpretation of data:

Information

Key element in end user mental model

The aim of good program: capturing the information model in the data model

Data ⟷ Staff in User Head

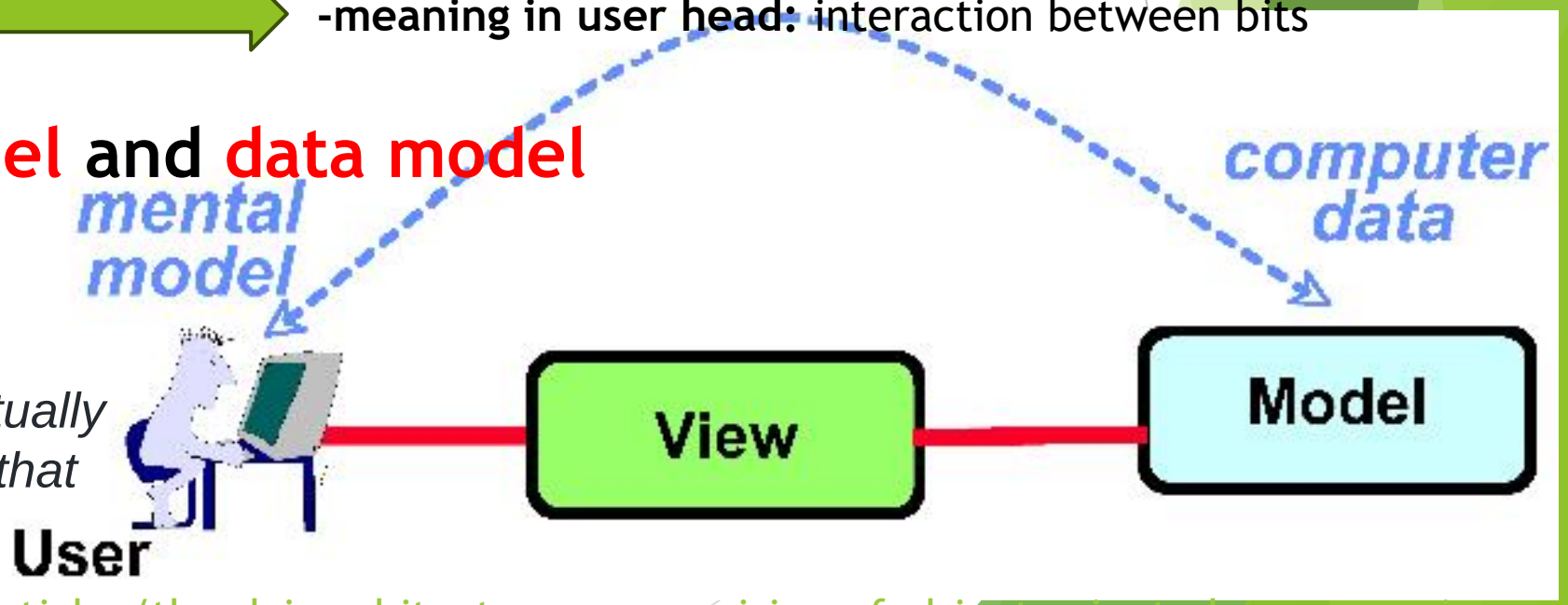
-representation of information

-in computers: bits ⟷ **-meaning in user head:** interaction between bits

Union of cognitive model and data model

<u>Direct manipulation metaphor:</u>

the sense that end users are actually manipulating objects in memory that reflect the images in their head



Source: https://www.artima.com/articles/the-dci-architecture-a-new-vision-of-object-oriented-programming

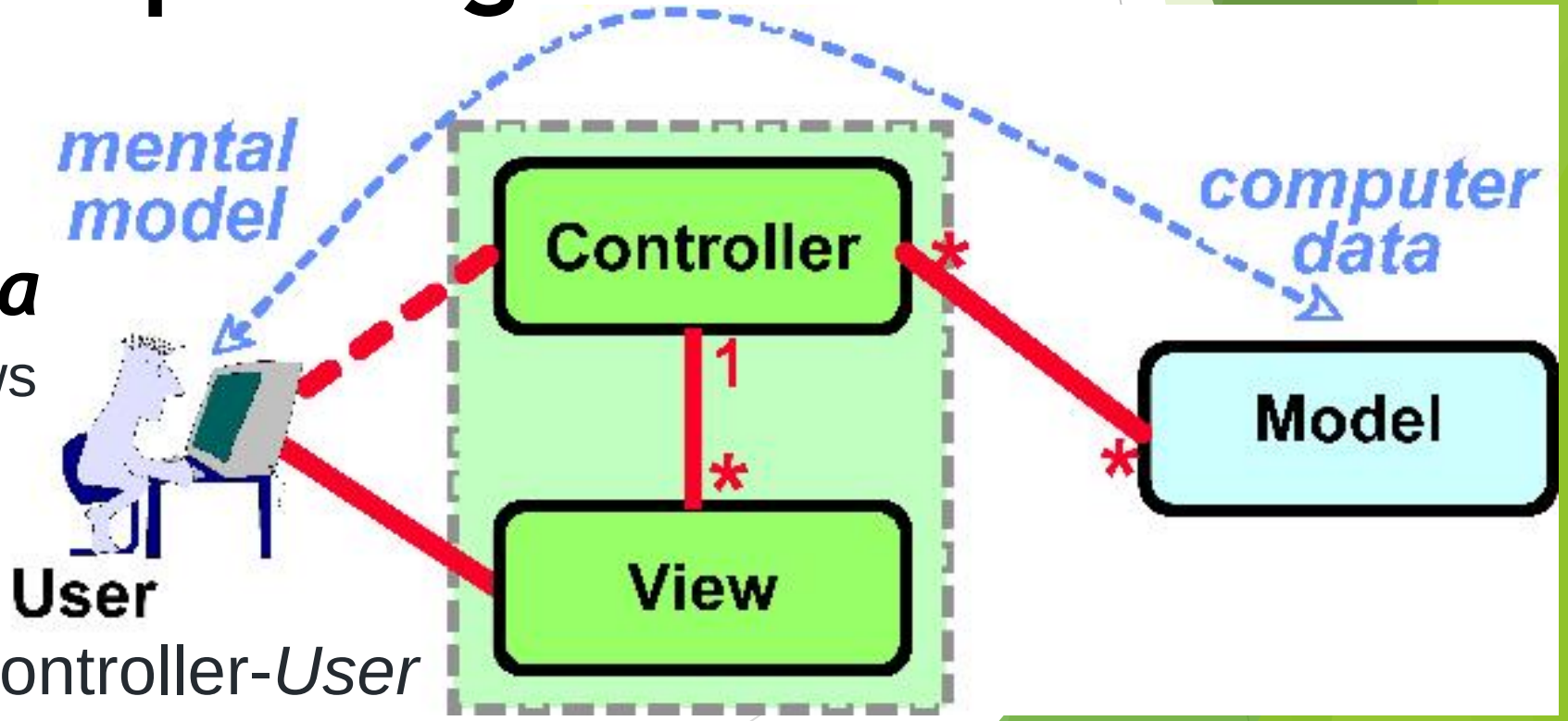
Direct Manipulation Metaphor

Raw Data ⟷ **Simple Cognitive Models**

Model

Filters the data

The Controller creates Views and coordinates Views and Models.



CAPACITY: Model-View-Controller-*User*

Source: https://www.artima.com/articles/the-dci-architecture-a-new-vision-of-object-oriented-programming

Raw Data ⟷ Simple Cognitive Models

Basic building blocks **Model** ***Filters the data***

Example: half-call in telephony domain

Telephone operator: duration, may shrink number of parties → Supported user illusion with model

…

Another actor: sees data differently/ different perspective → Supported user illusion with model

↓

User has a feel that he is manipulating with real object

Particularly simplistic rule of thumb

nouns ⟷ **verbs**

objects — methods

Solved by:

Extension by Derivation

Smalltalk

class as implementation tool
for object as analysis concept

Static type system

Objects are stable

All code in objects thus cannot
represent a change

Agile vision of
evolution and
maintainability

separed

Solved by:

inheritance graphs as design abstraction
-fully represented by base class interface

Inheritance

Problem of inheritance hierarchies 25 deep

Solved by:

stable

changeable

Supports open-closed principle

programming by extension

programming by difference

newly added methods either
didn't appear in the base class

Data *domain entity class*

- domain classes should be dumb

stable data models

Roles

-behaviors that are about what objects *do*

New concept of an action from users heads

dynamic behavioural models

roles embody generic, abstract algorithms

weave the algorithms through the roles

Interaction as algorithms

- mirrors from the user's mind into the code

- data with their own vocabulary and rules

OBJECT

Unifies two models in users head

WHAT SYSTEM IS DATA MODEL

Static behaviour – for thinking

WHAT SYSTEM DOES ALGORITHM MODEL

Dynamic behaviour

Role Model Synthesis

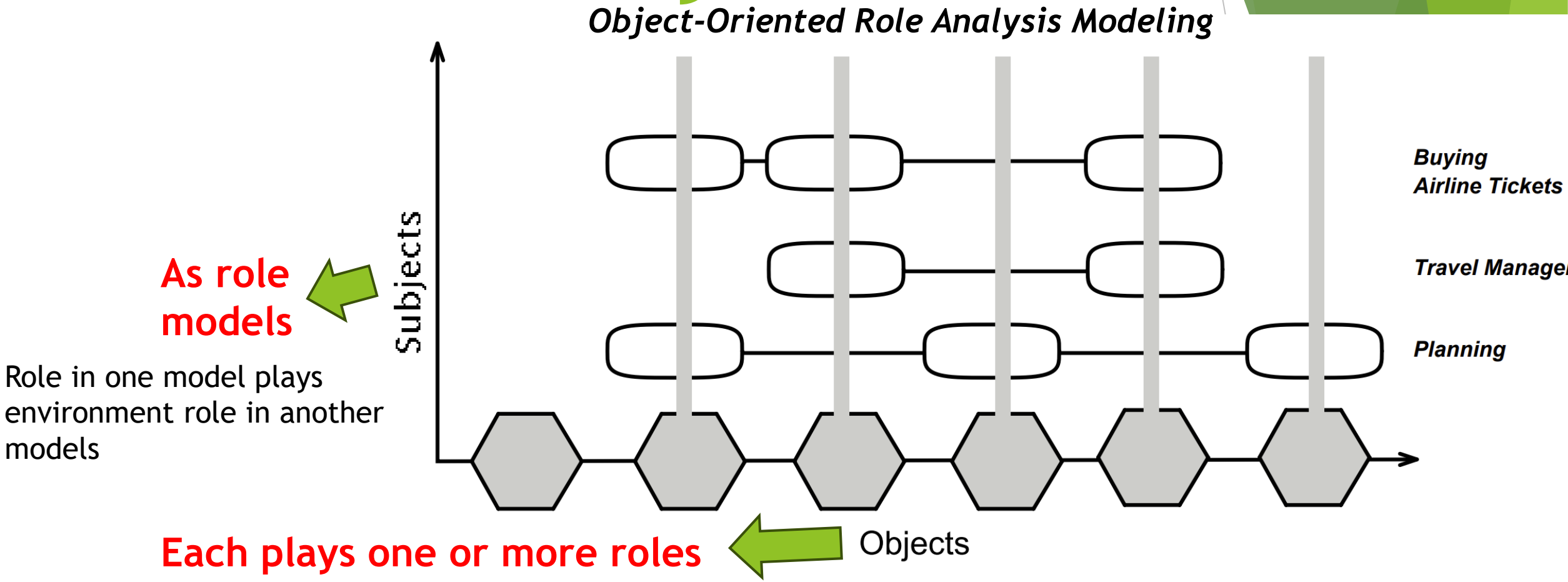


Figure 10. This 'hat stand' synthesis illustration is due to Philip Dellaferra.

Source: *Reenskaug, Trygve; P. Wold; O. A. Lehne (1996). Working with Objects: The OOram Software Engineering Method. Manning/Prentice Hall.*

What System is?

What System Does?

All users perspective

Single user perspective

Domain Knowledge ⟷ The Mental Model of End User

Objects ⟷ Roles

Interaction of Roles

Domain Analysis ⟷ Use Case Modeling

Source: https://poetisania.com/val/as/index.html

Domain Knowledge

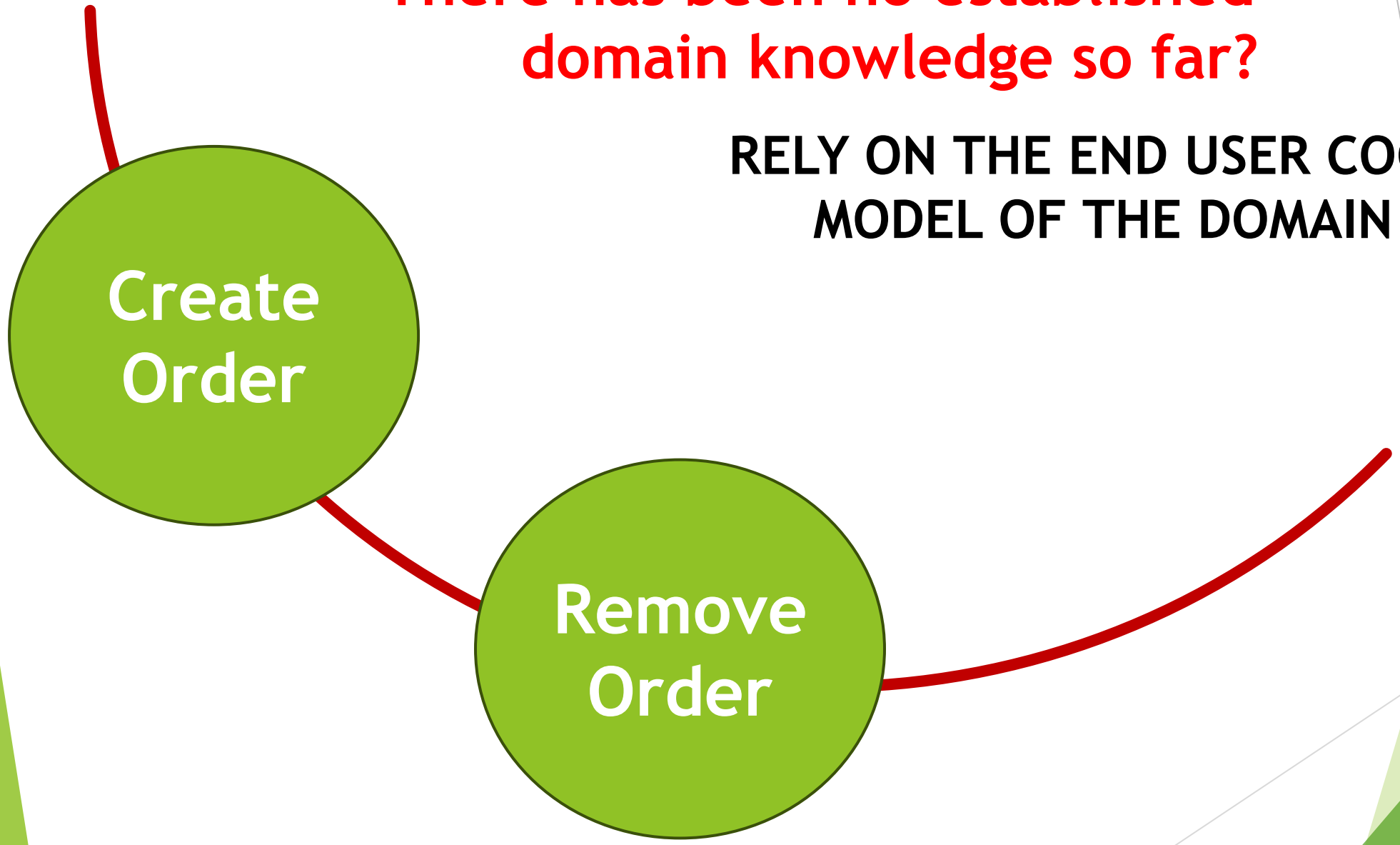
Timeless compression of mental models of end users and other stakeholders

Mental models whose patterns are tacidly driven by commonality and variation

Source: James O. Coplien and Gertrud Bjrnvig. 2010. Lean Architecture: for Agile Software Development. Wiley Publishing.

There has been no established domain knowledge so far?

RELY ON THE END USER COGNITIVE MODEL OF THE DOMAIN



Source: https://poetisania.com/val/as/index.html

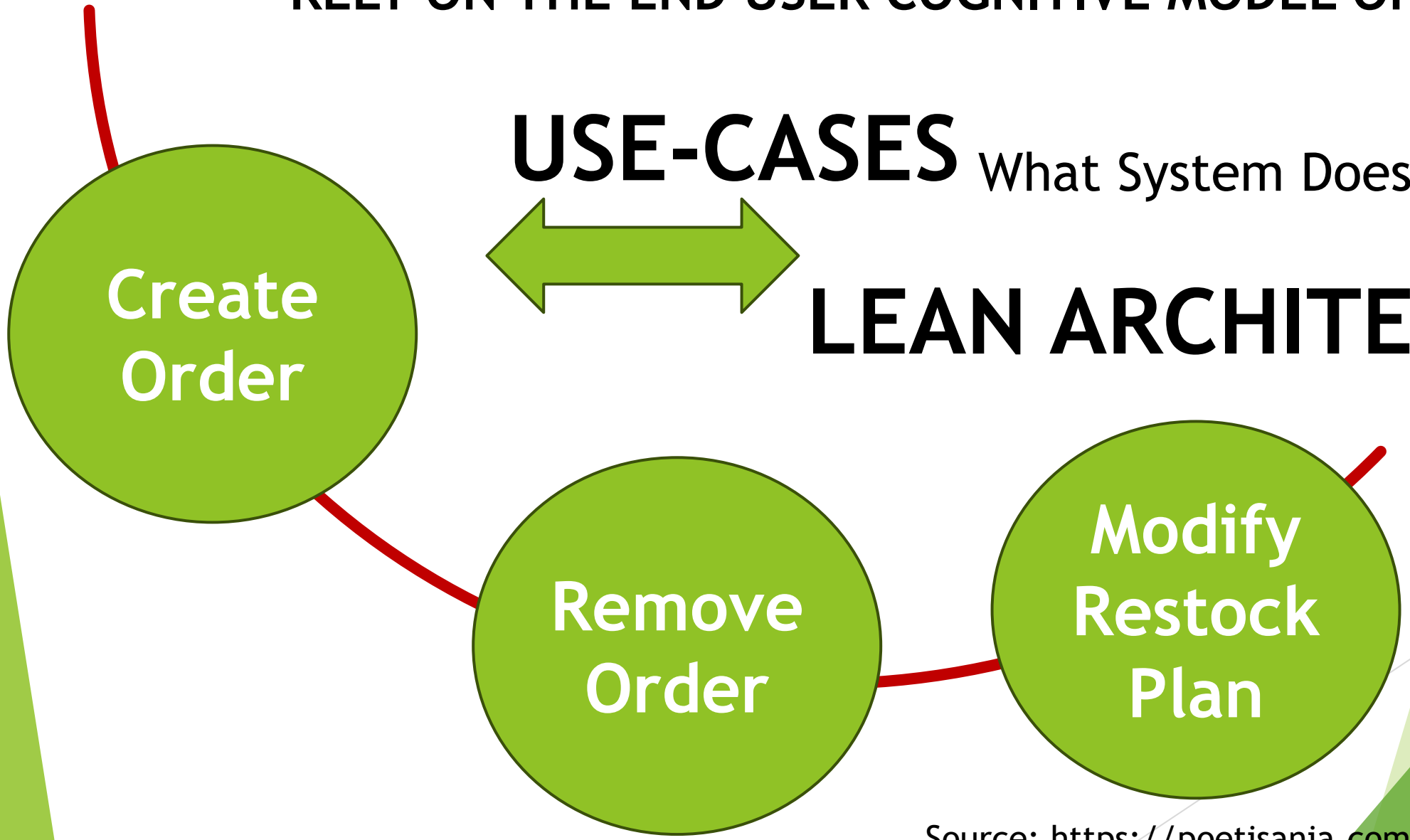
There has been no established domain knowledge so far?

RELY ON THE END USER COGNITIVE MODEL OF THE DOMAIN

USE-CASES What System Does

LEAN ARCHITECTURE

What System is



Source: https://poetisania.com/val/as/index.html

USE-CASES

preserved in code

Producing only what is neccessary without modyfying an existing functionality

... like stringing corals on a thread

Lean Architecture

Only as much architecture as is neccessary

Agile Software Development

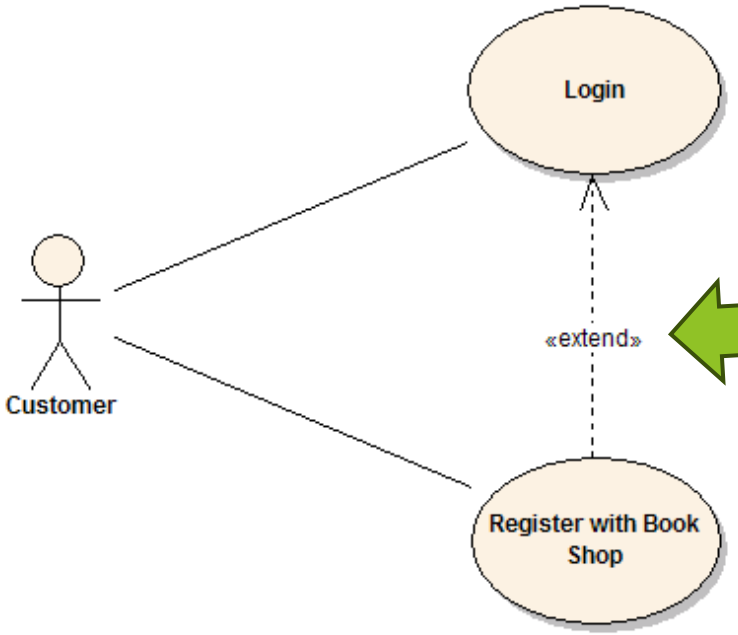
Avoid producing waste...

Pull, Do not push *end users*

Making decisions at responsible moment

System architecture has to reflect the end user cognitive model being able to accommodate emerging use-cases

Preserving Aspects in Code



Use-case modularity problem

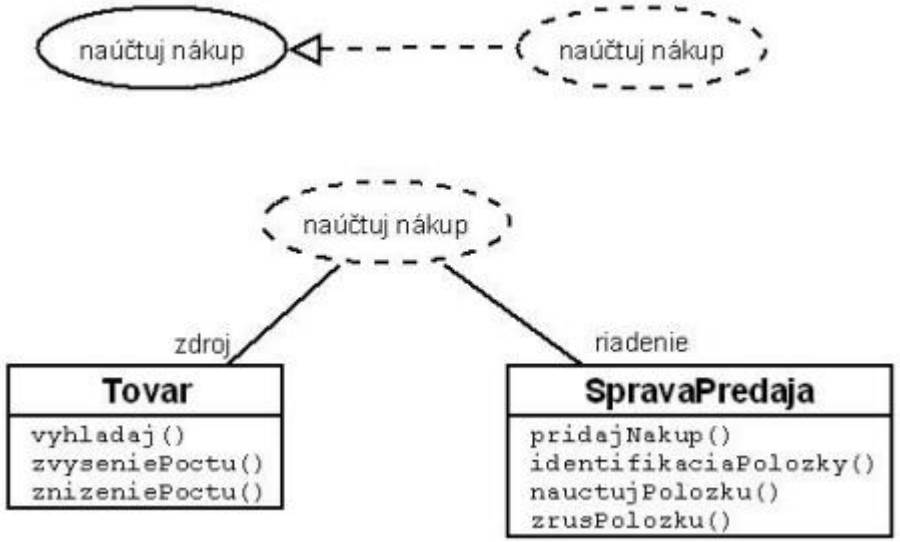
Previously unsupported in analytical models and in implementational environments

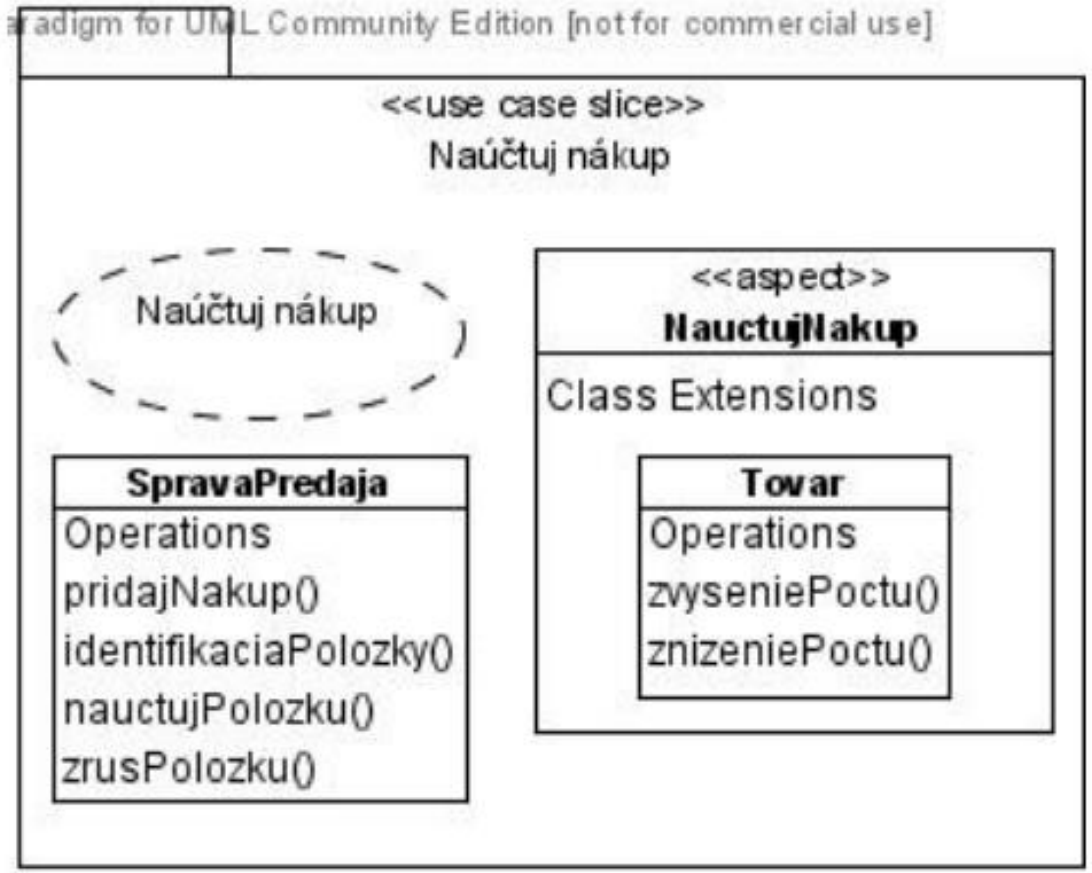
Jacobson, I., Ng, P.: Aspect-Oriented Software Development with Use Cases. Addison Wesley Professional (2004), ISBN 0-321-26888-1.

Solution to Peer Use Cases: *Intertype Declaration*

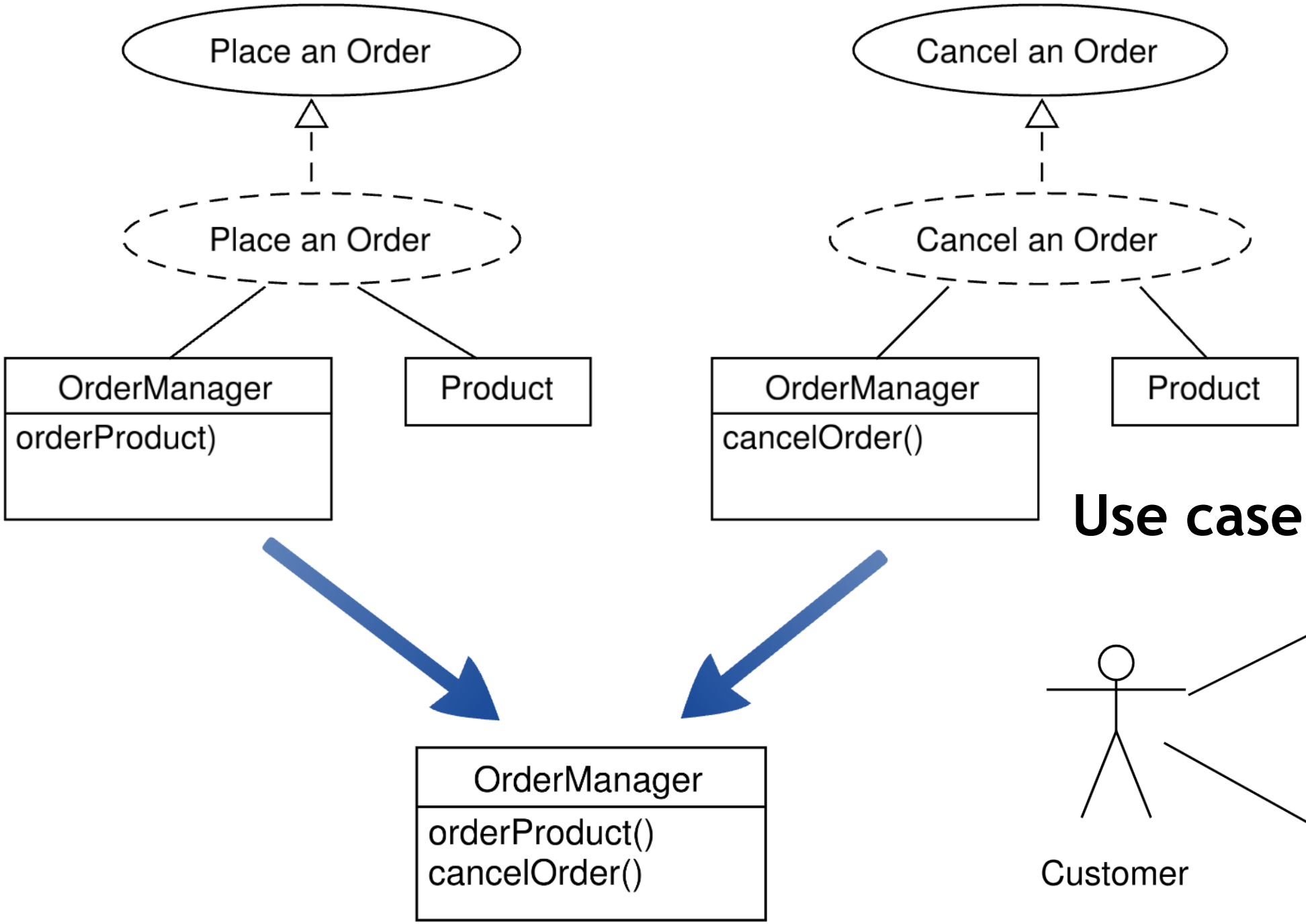
-we create use case slice…

…containing only specifics for this use case (Accounting the purchase in Figure)

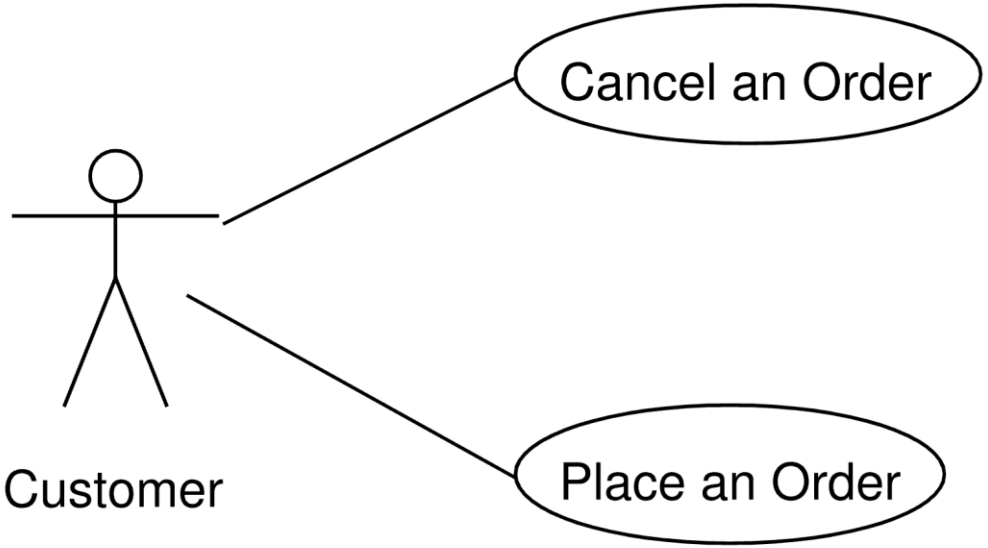




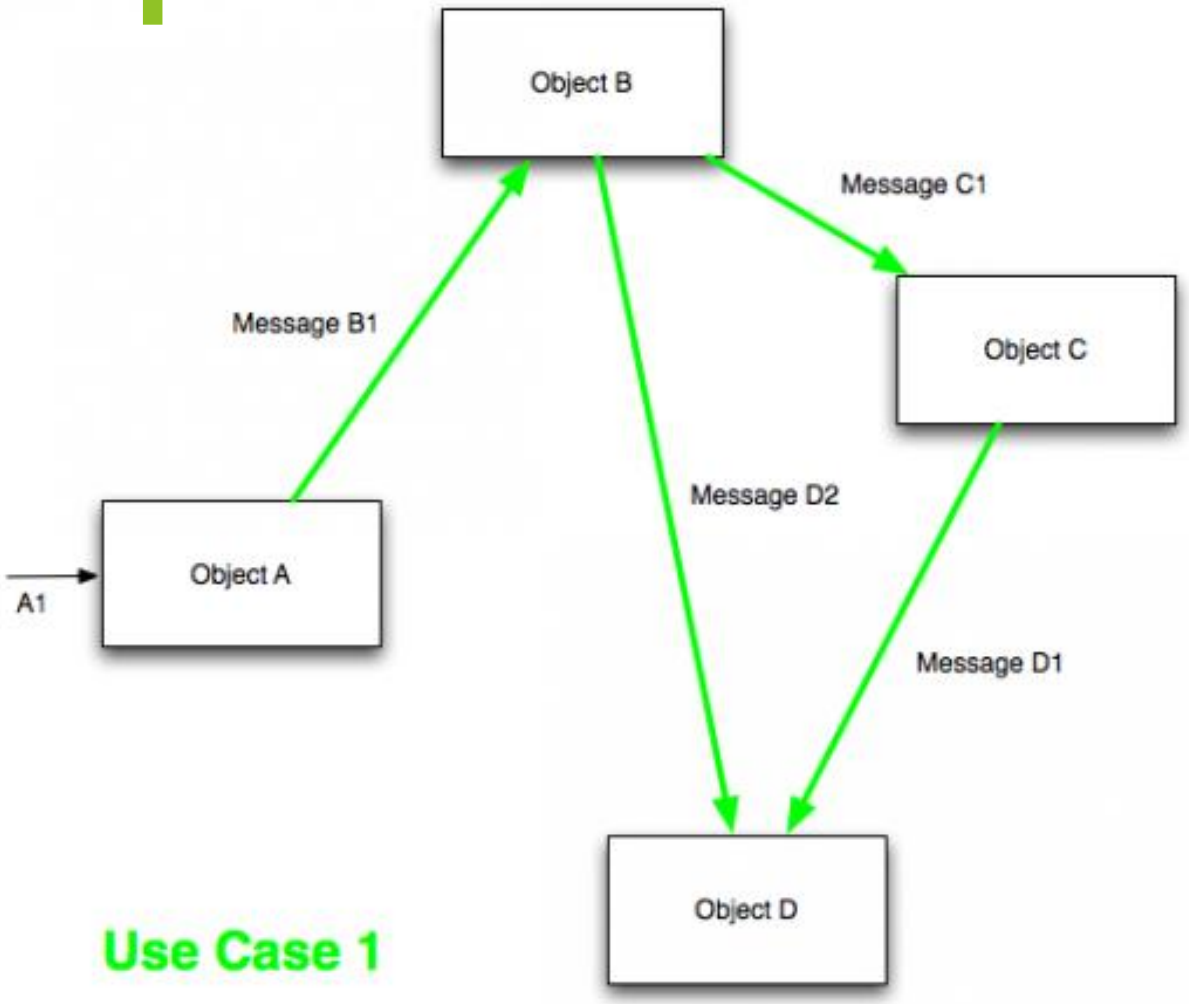
Zdroje z Bc. Pavol Michalco: PRÍPADY POUŽITIA A TÉMY V PRÍSTUPE THEME/DOC

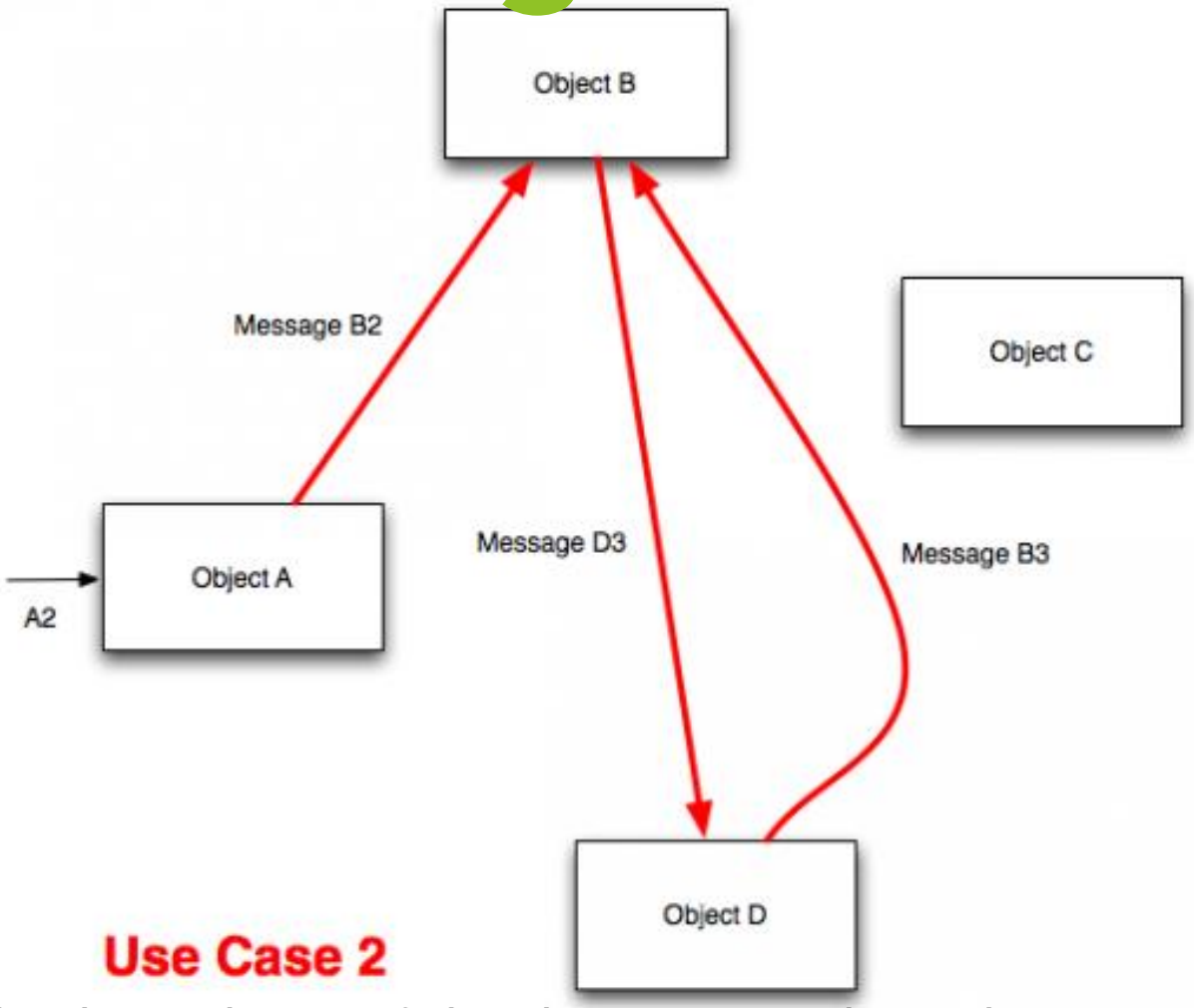

Place an Order
Place an Order
OrderManager
orderProduct)
Product
Cancel an Order
Cancel an Order
OrderManager
cancelOrder()
Product
OrderManager
orderProduct()
cancelOrder()

Use case:

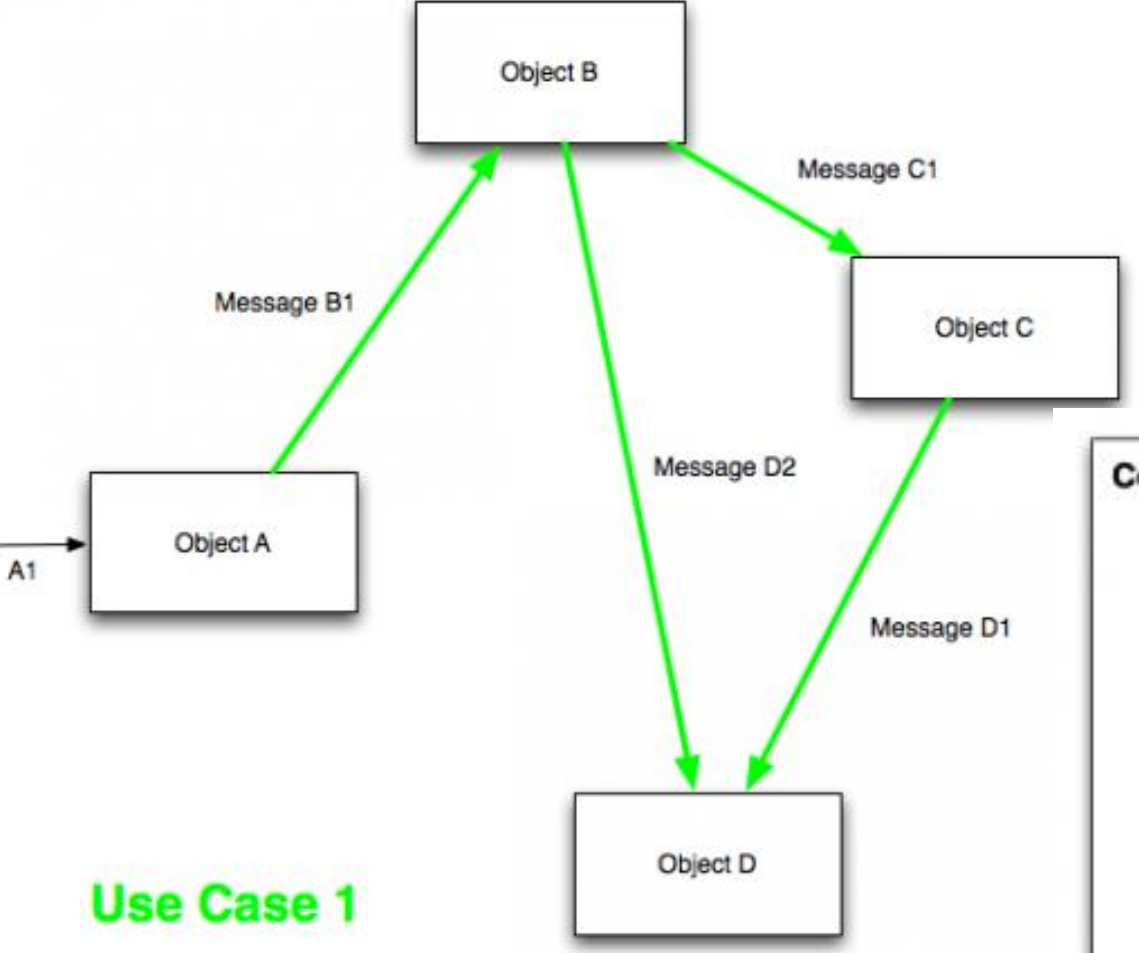

Cancel an Order
Customer
Place an Order

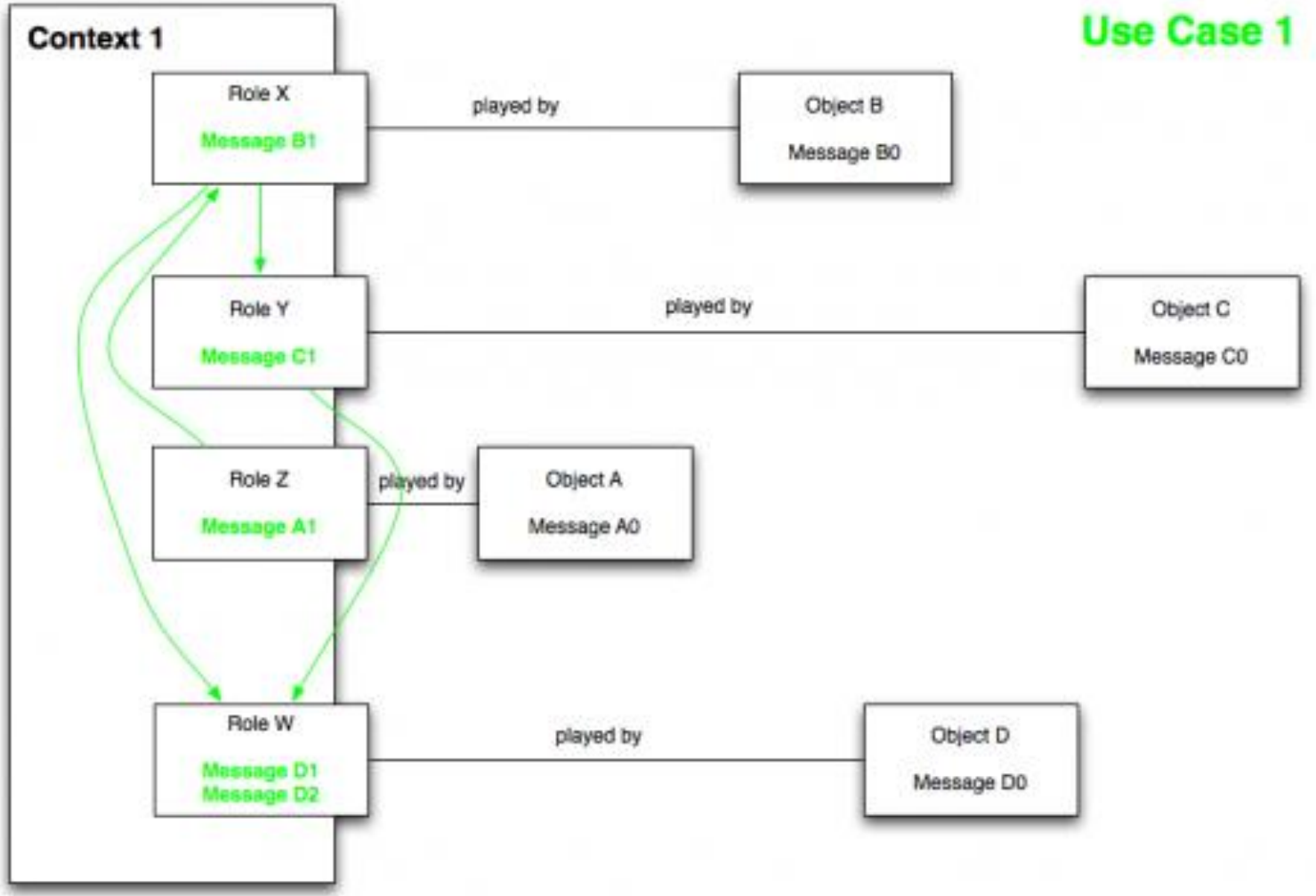
No Representation of System Operations In Code Using OOP



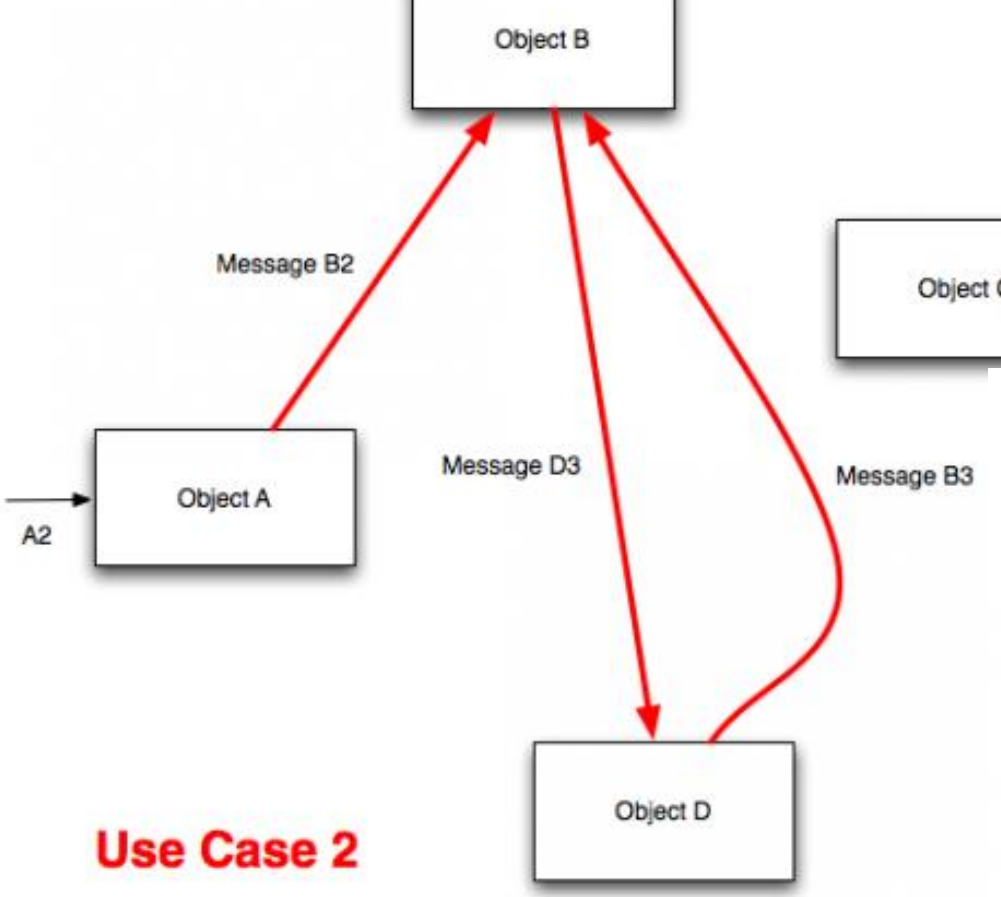


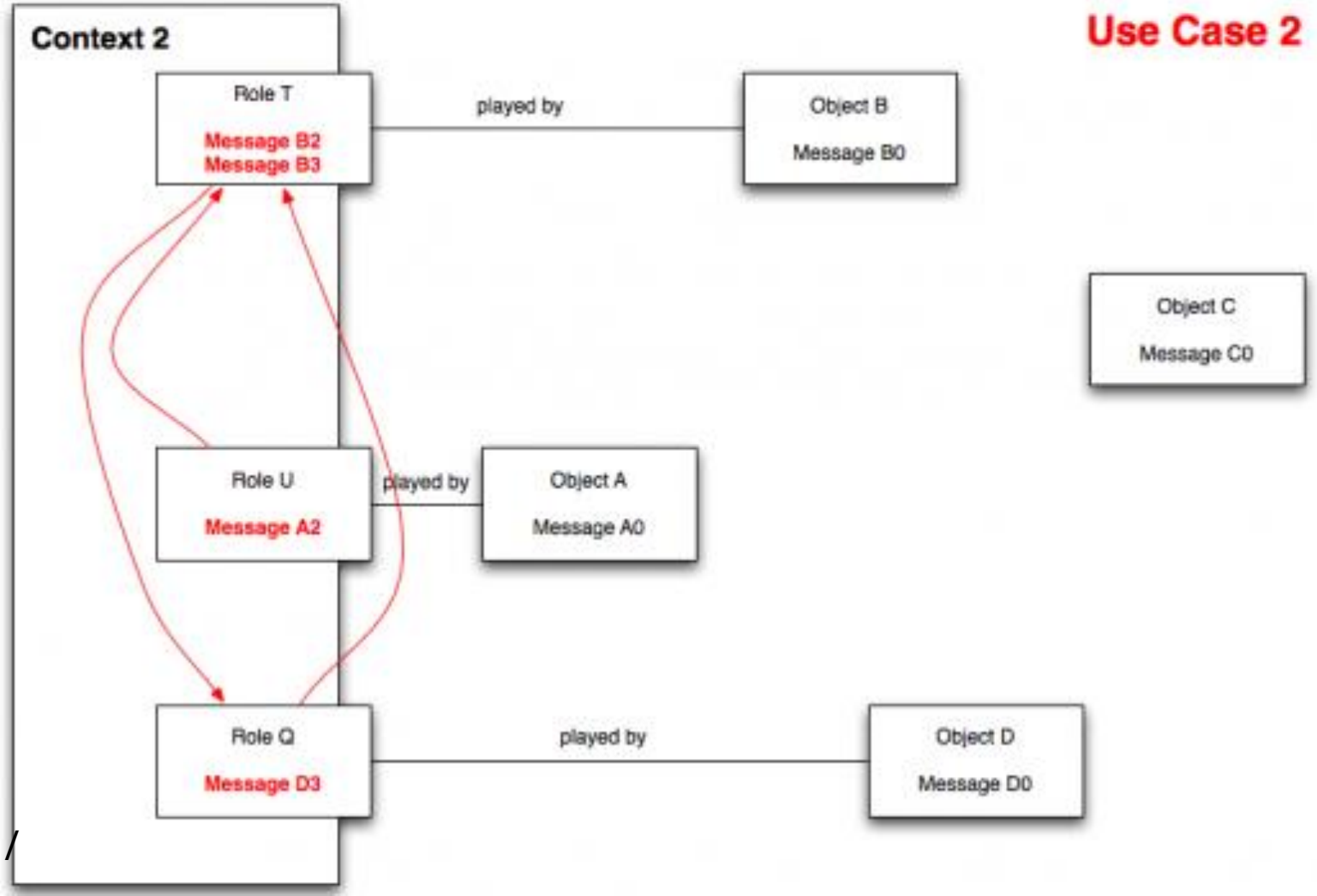
Source: https://www.sitepoint.com/dci-the-evolution-of-the-object-oriented-paradigm/





Source: https://www.sitepoint.com/dci-the-evolution-of-the-object-oriented-paradigm/

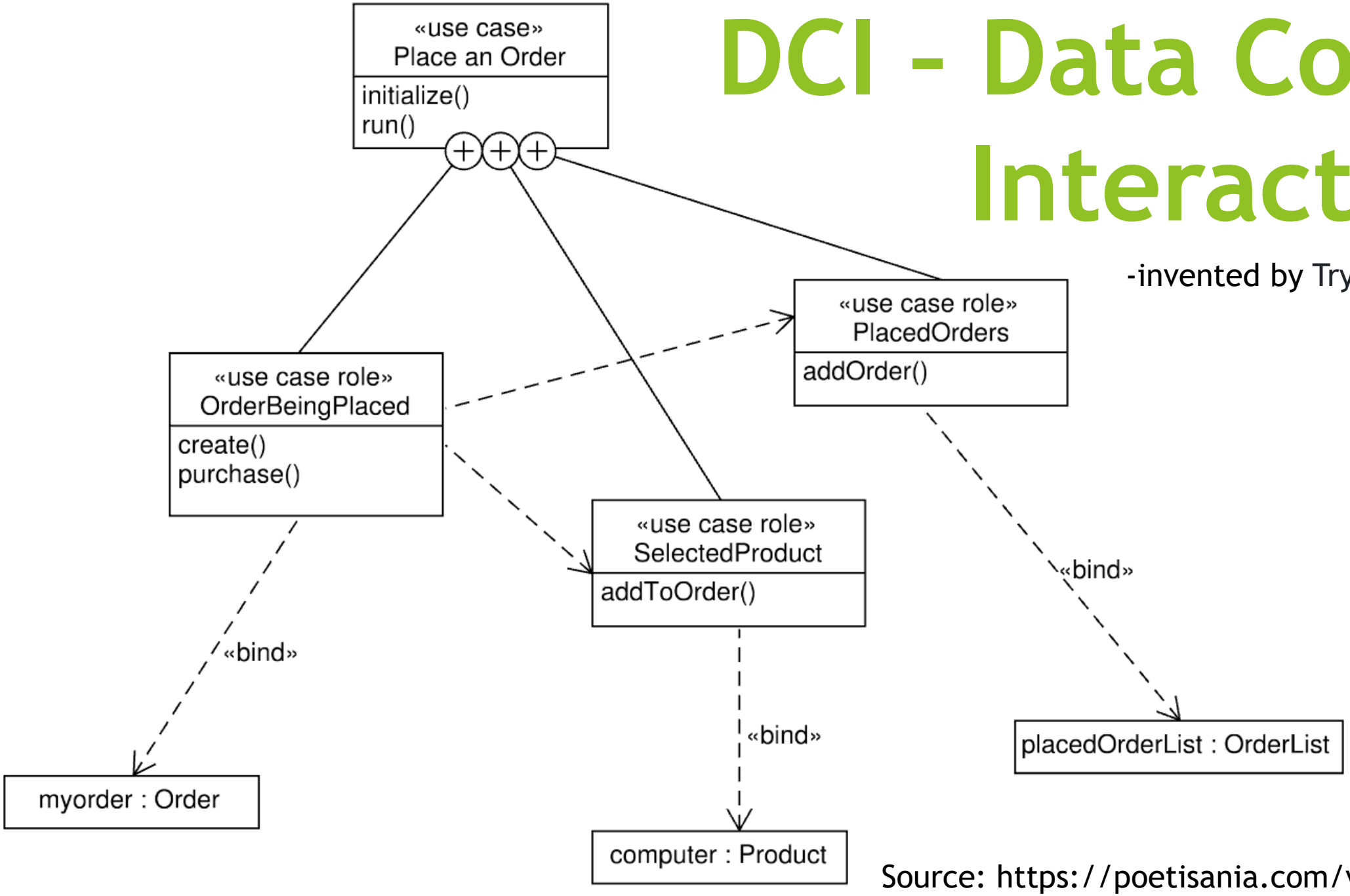




Source: https://www.sitepoint.com/dci-the-evolution-of-the-object-oriented-paradigm/

DCI – Data Context Interaction

-invented by Trygve Reenskaug

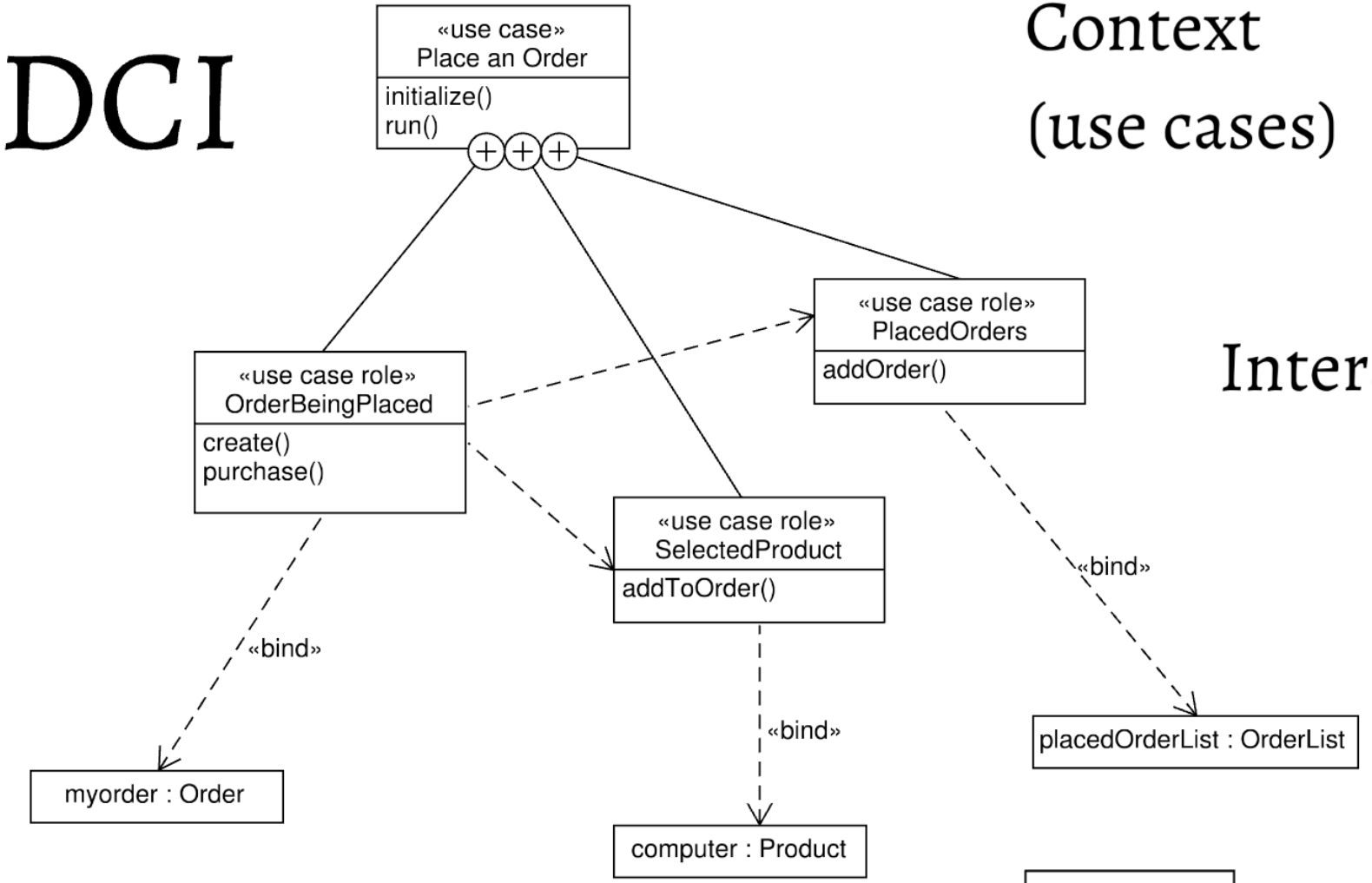


Source: https://poetisania.com/val/aosd/index.html

DCI

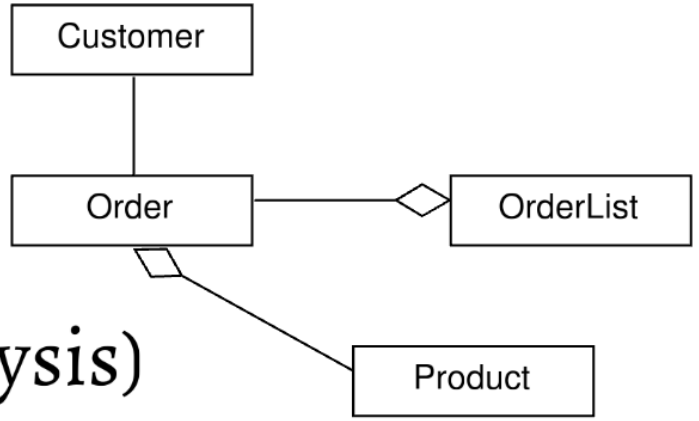
Context

(use cases)



Interaction

Source: https://poetisania.com/val/aosd/index.html



Data

(from domain analysis)

DCI Class

Data

-expresses exclusively
object inner functionality
(not concerning neighbor objects)

Interaction

- ”What the system is”

-a data and associated local methods

LOCALITY

-each use case in
separate file

**SEPARATION TOWARDS
STABLE SOFTWARE**

Source code matches the runtime

-which values are assigned to particular entities

=> ***Observable from the context***

Context

- ”What the system does”

-contextual behavior - **only methods which occur in use case**

DCI Context

-expresses only communication between objects

IS CHANGING RAPIDLY

FOCUS

DCI: Data, Context and Interaction

Decouples

the stable part of the architecture

(domain objects – data)

from

its variabile part

(use cases)

with

their flexible binding

(roles)

Source: https://poetisania.com/val/aosd/index.html

Variability at the model level?

Theme/UML

> The themes to be composed can be selected

> The composed model can be generated

Source: https://poetisania.com/val/aosd/index.html

Aspect-Oriented Model-Driven Software Product Line Engineering (AO-MD-PLE)

Aspect-Oriented Change Realization

CHANGE AS CROSSCUTTING REQUIREMENS

Change

-initiated by a change request made by stakeholder (user,…)

Change request

-usually focused on changes to be realized
-containing even interrelated requirements
- has to be split into individual changes, their generalization and aggregation according to particular domain

Domain Specific Changes

-returning another SMTP Server instead of original one using Cuckoo's egg pattern

```
public class AnotherClass extends MyClass {
 ...
}
 ...
public aspect MyClassSwapper {
  public pointcut myConstructors():
    call(MyClass.new ());
  Object around(): myConstructors()
  {
    return new AnotherClass();
  }
}
```

Generalization to Class exchange change type

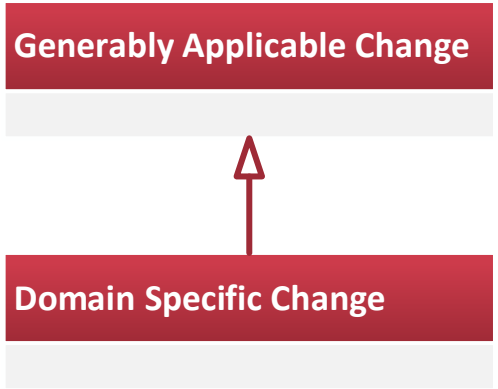
```
public class SMTPServerM extends SMTPServer {
 ...
}
 ...
public aspect SMTPServerBackupA {
  public pointcut SMTPServerConstructor(URL url,
                                        String user,
                                        String password):
    call(SMTPServer.new (..)) && args(url, user,
                                        password);
  SMTPServer around(URL url, String user,
                                        String password):
    SMTPServerConstructor(url, user, password)
  {
    return getSMTPServerBackup(proceed(url, user,
                                        password));
  }
  private SMTPServer
  getSMTPServerBackup(SMTPServer obj)
  {
    if (obj.isConnected()) {
      return obj;
    } else {
      return new SMTPServerM(obj.getUrl(),
                                        obj.getUser(),
                                        obj.getPassword());
    }
  }
}
```

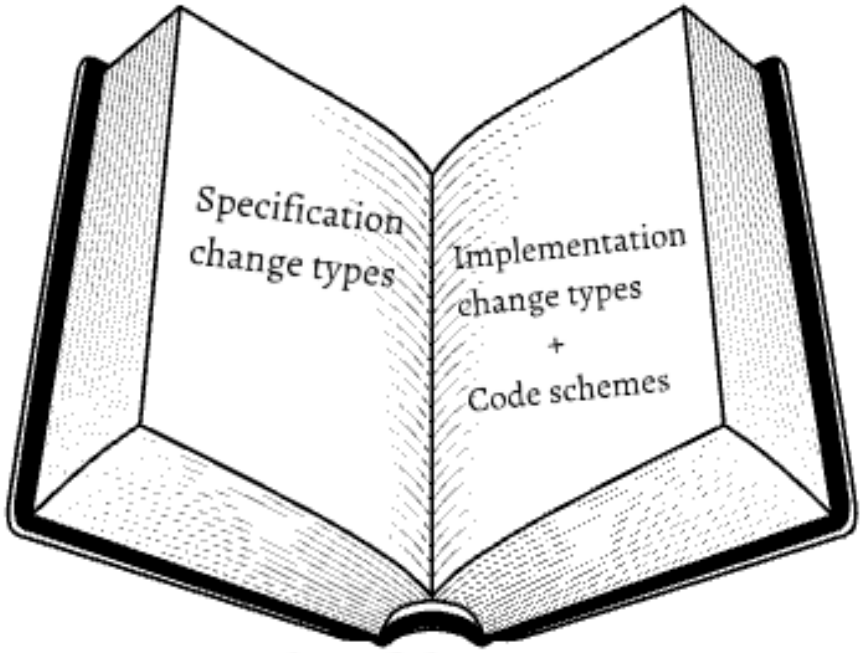
Source: **Aspect-Oriented Change Realizations and Their Interaction** Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

Catalog of Change Types

-to provide developer hints about incorporated changes

MAINTAINING CATALOG OF CHANGES





Catalog of change types

1. Generalize the change (description)
2. Find the corresponding specification change type in the catalog
3. Apply the matching implementation type with its code scheme

Source: **Aspect-Oriented Change Realizations and Their Interaction** Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

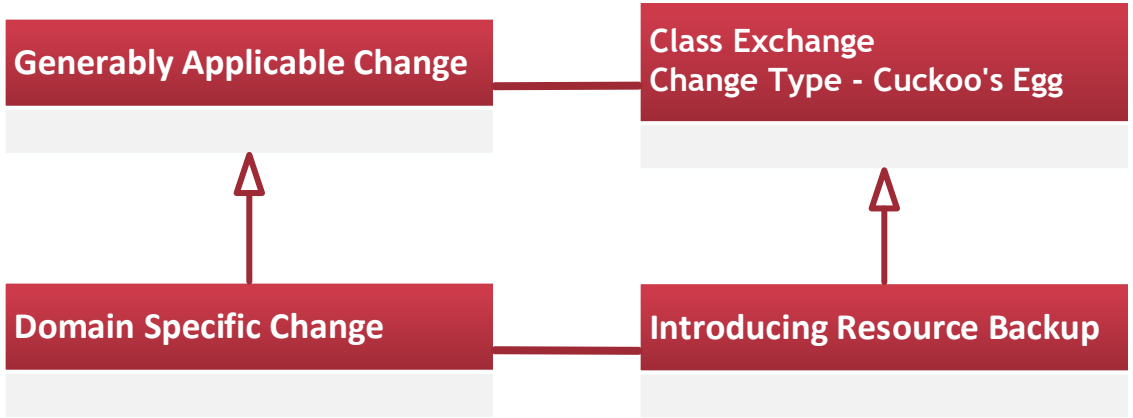
Domain Specific Changes

-returning another SMTP Server instead of original one using Cuckoo's egg pattern

```
public class AnotherClass extends MyClass {
 ...
}
 ...
public aspect MyClassSwapper {
  public pointcut myConstructors():
    call(MyClass.new ());
  Object around(): myConstructors()
  {
    return new AnotherClass();
  }
}
```

Generalization to Class exchange change type

```
public class SMTPServerM extends SMTPServer {
 ...
}
 ...
public aspect SMTPServerBackupA {
  public pointcut SMTPServerConstructor(URL url,
                                        String user,
                                        String password):
    call(SMTPServer.new (..)) && args(url, user,
                                        password);
  SMTPServer around(URL url, String user,
                                        String password):
    SMTPServerConstructor(url, user, password)
  {
    return getSMTPServerBackup(proceed(url, user,
                                        password));
  }
  private SMTPServer
  getSMTPServerBackup(SMTPServer obj)
  {
    if (obj.isConnected()) {
      return obj;
    } else {
      return new SMTPServerM(obj.getUrl(),
                                        obj.getUser(),
                                        obj.getPassword());
    }
  }
```

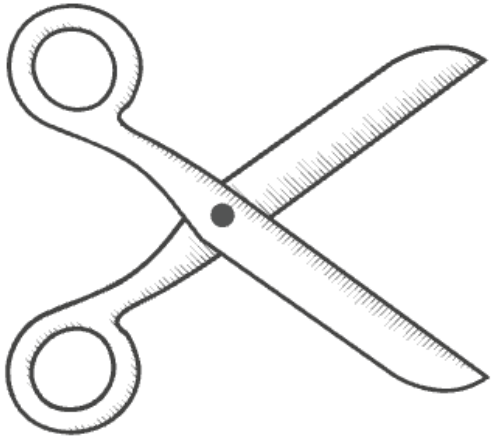


Source: **Aspect-Oriented Change Realizations and Their Interaction** Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

Applying Changes: Example

CHR03:

The administrator should be able to block and unblock an account from the accounts view.



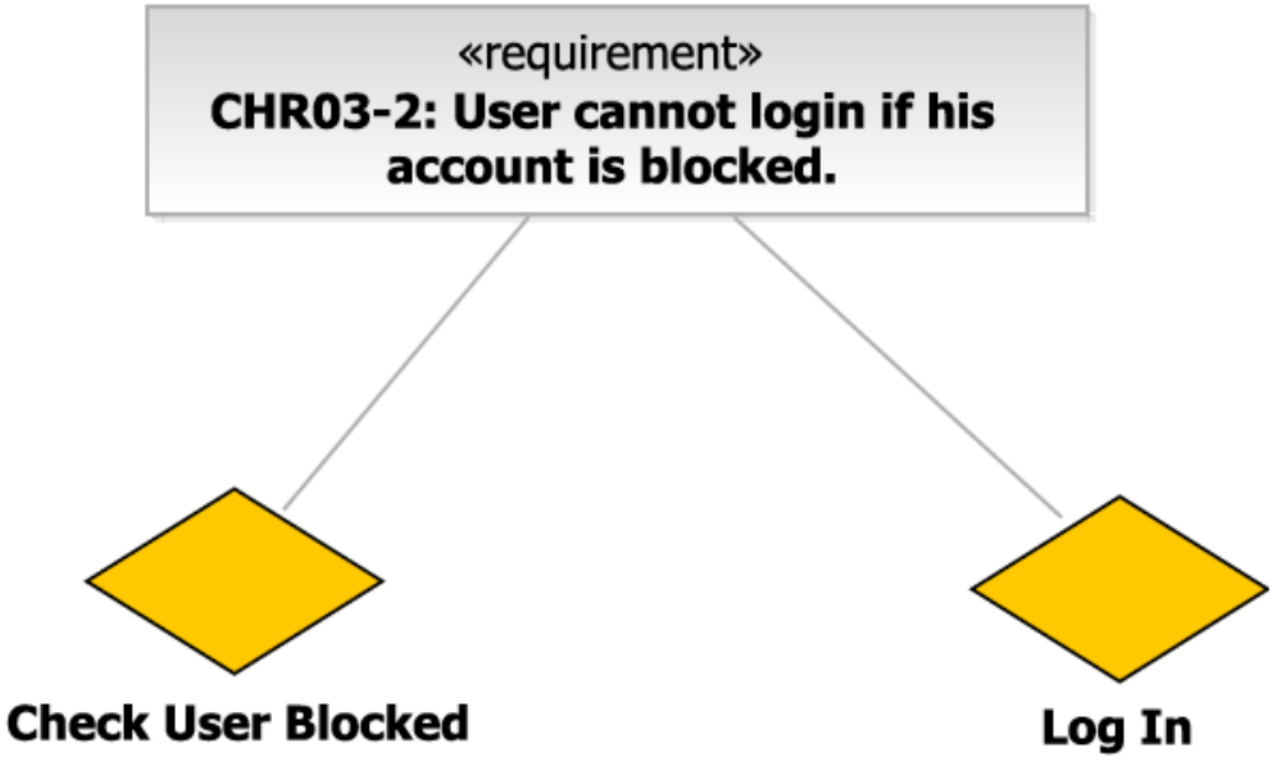
Source: https://poetisania.com/val/

CHANGE REQUIREMENTS:

Change CHR03-1: The administrator can block and unblock an account from the accounts view

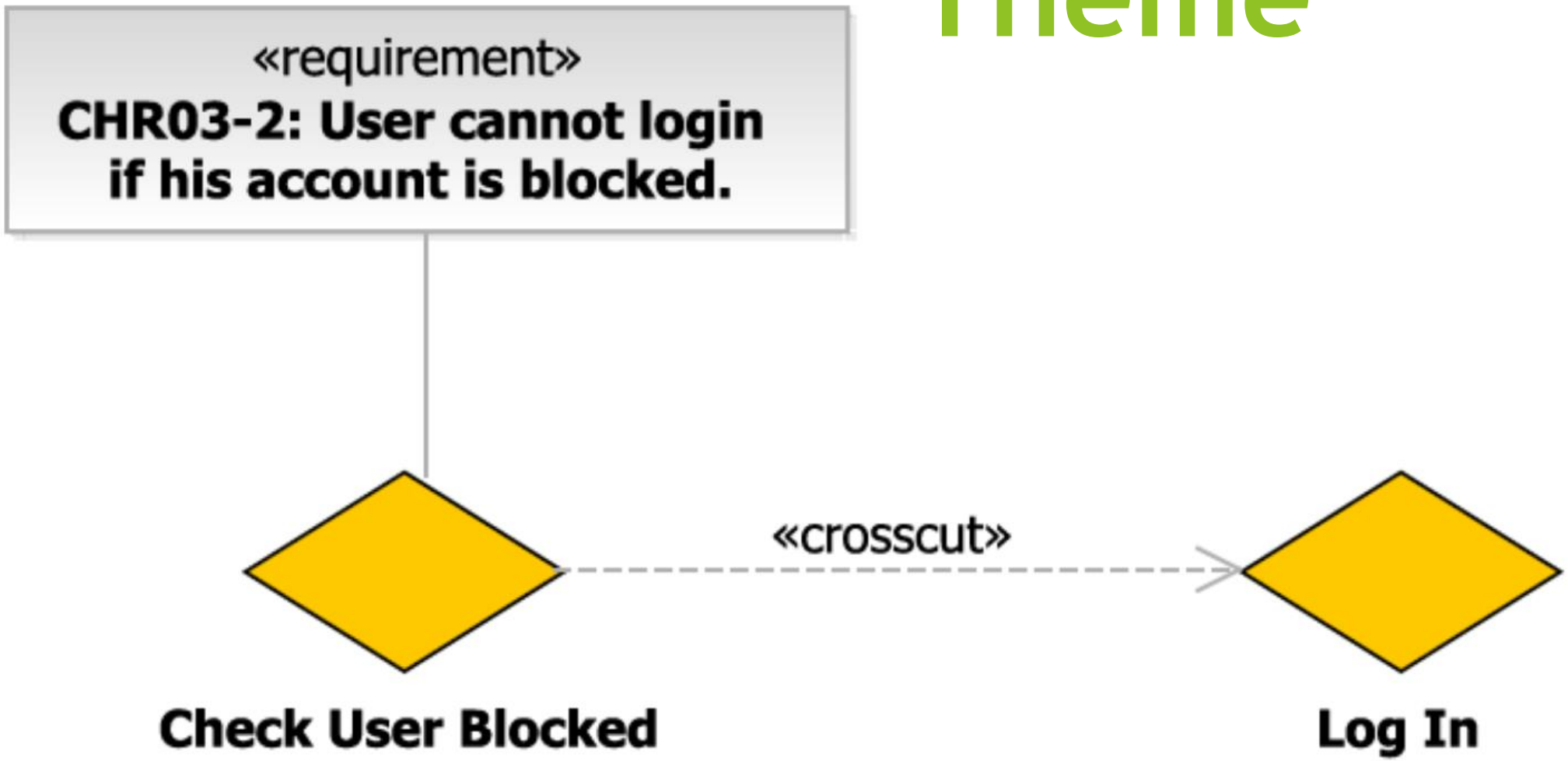
Change CHR03-2: A user cannot log in if his/her account is blocked

1. Identification of Themes in Change Request



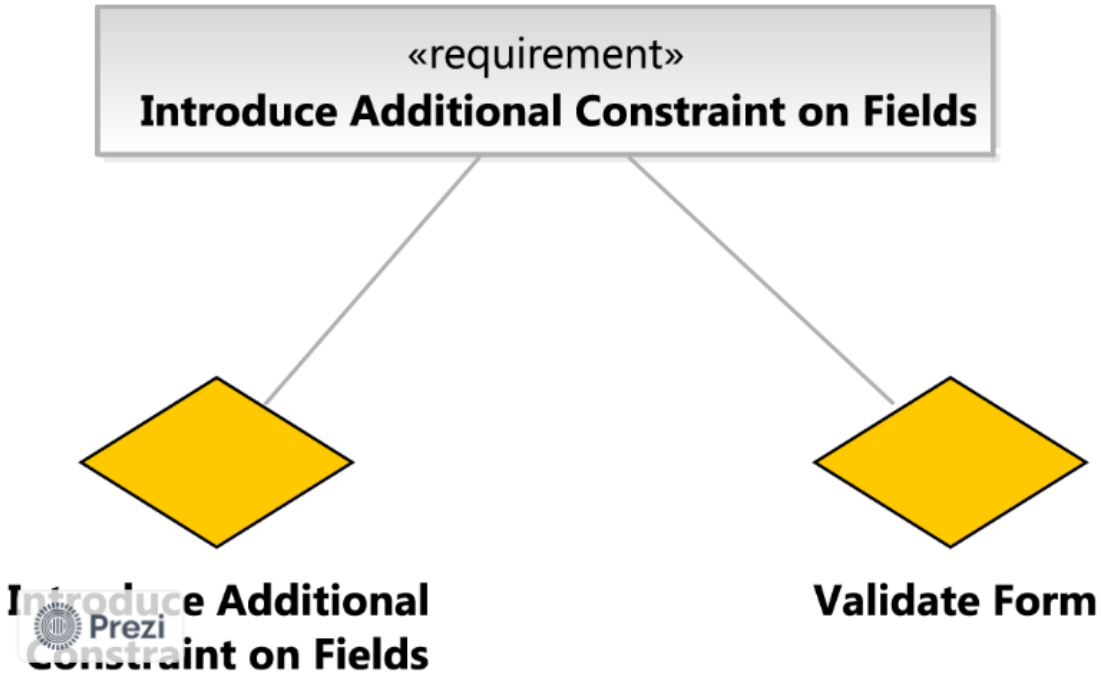
Source: https://poetisania.com/val/

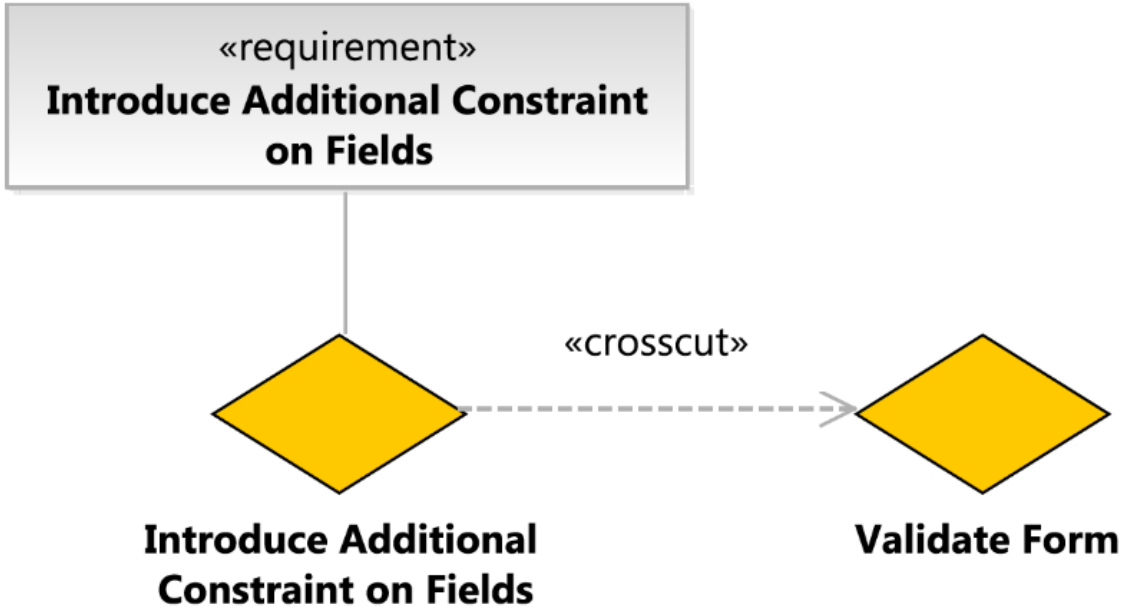
2. Determining Crosscutting Theme



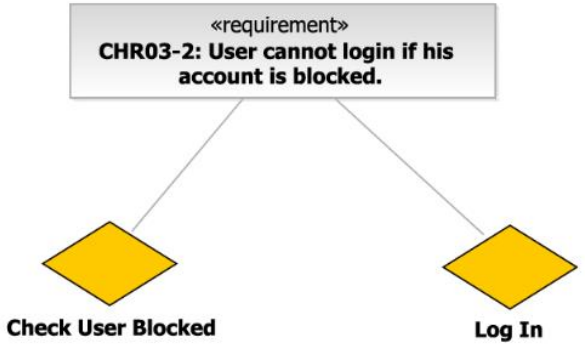
Source: https://poetisania.com/val/

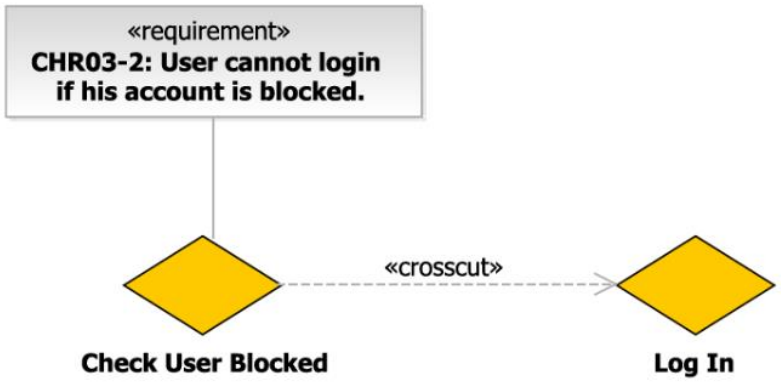
3. Observing Corresponding Specification Change Type in Catalog



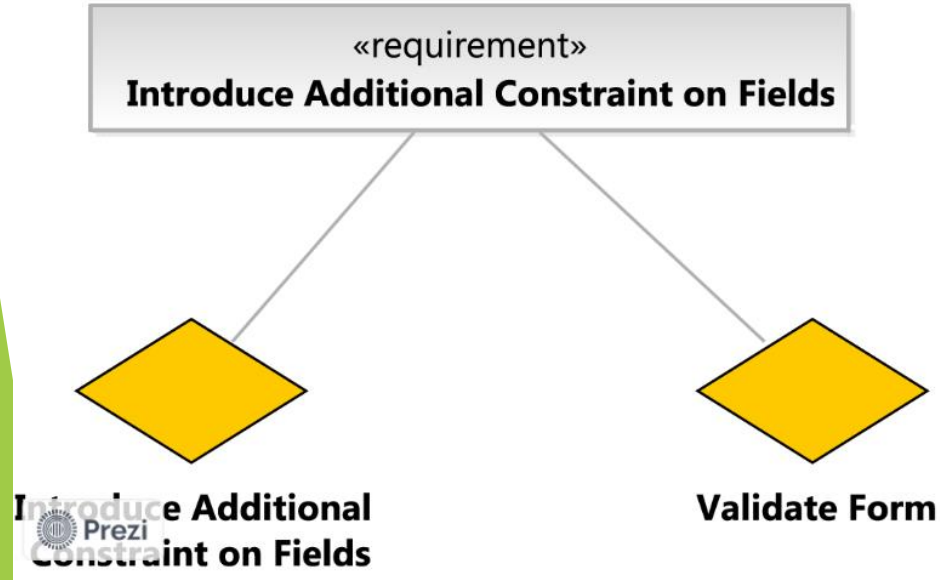


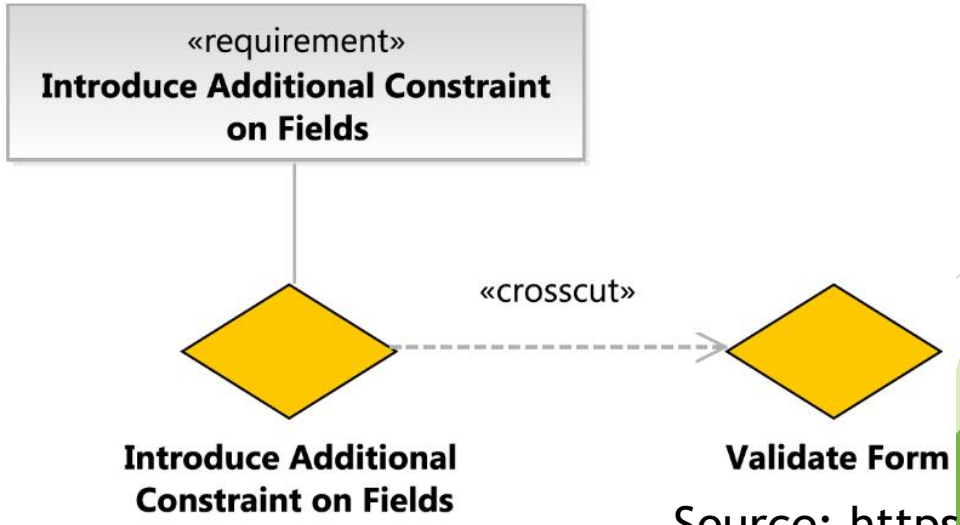
Source: https://poetisania.com/val/





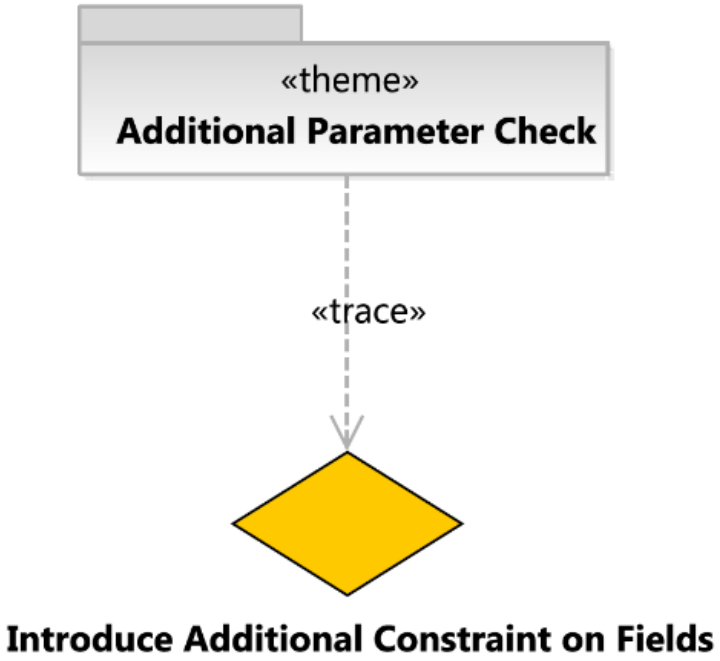
Find the corresponding specification change type in the catalog

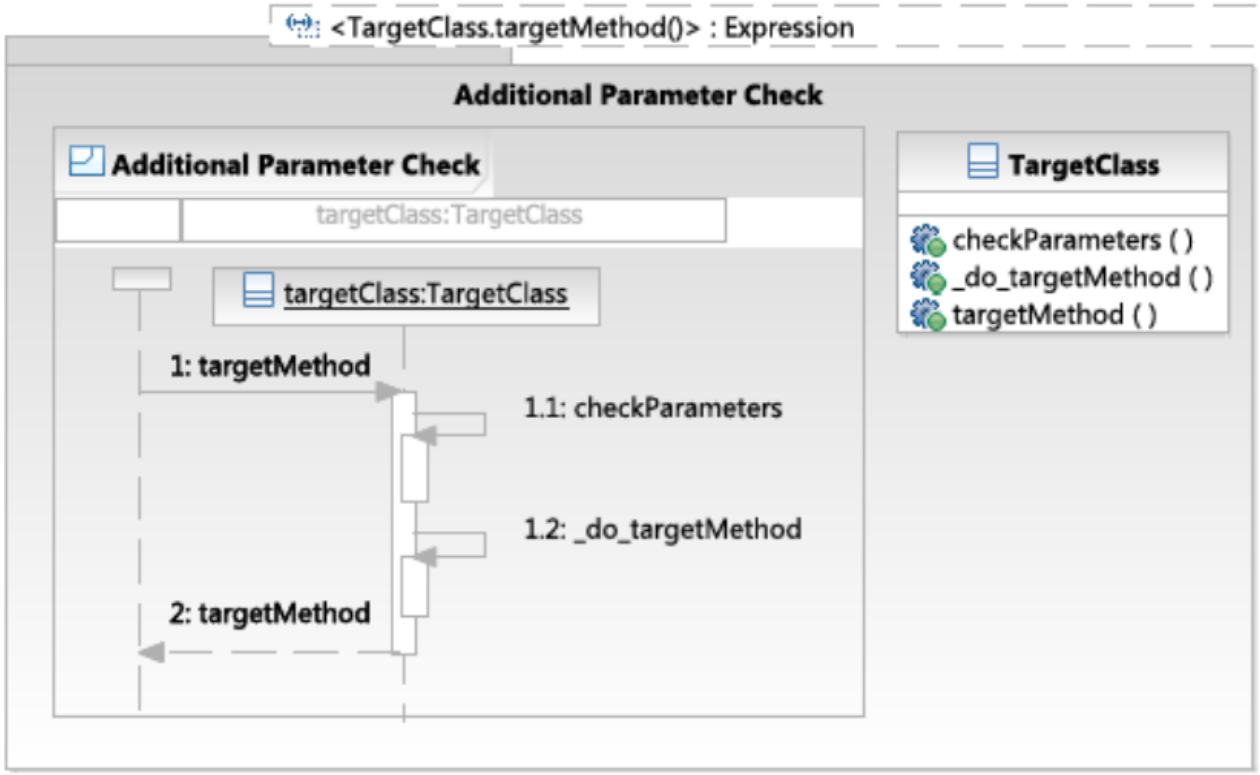




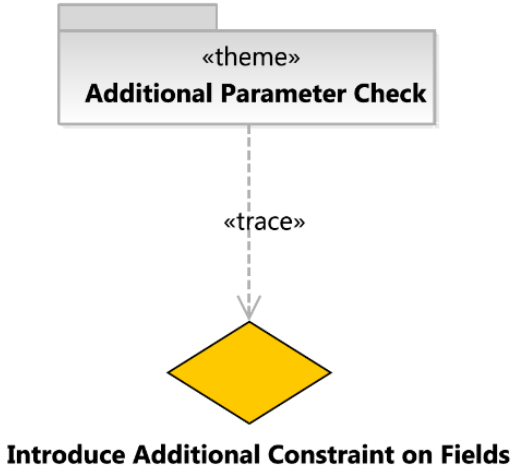
Source: https://poetisania.com/val/

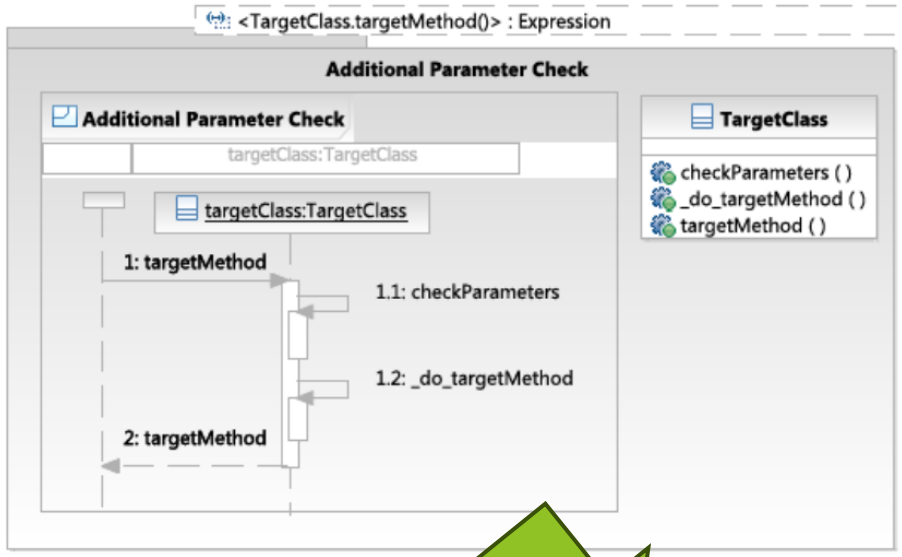
Finding Matching Realization Change in Catalog



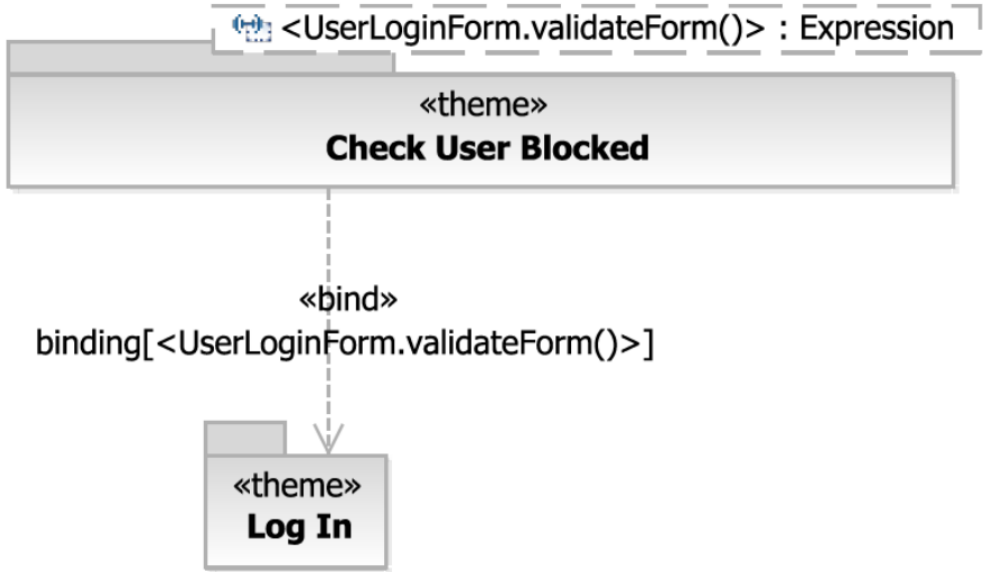


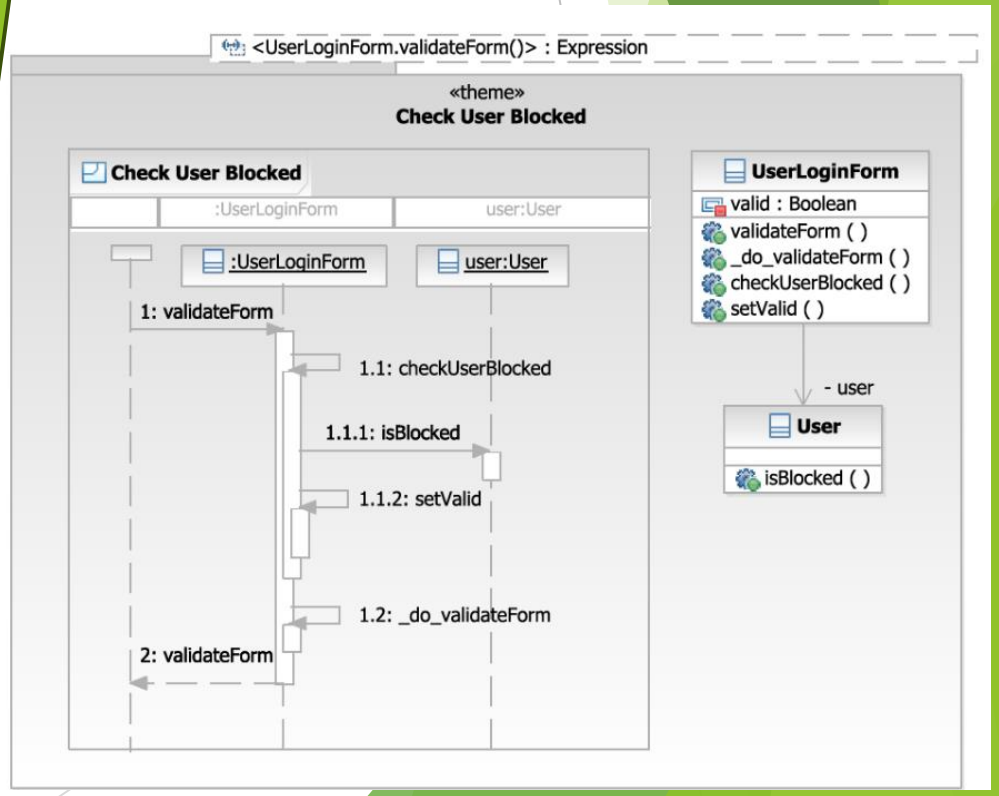
Source: https://poetisania.com/val/



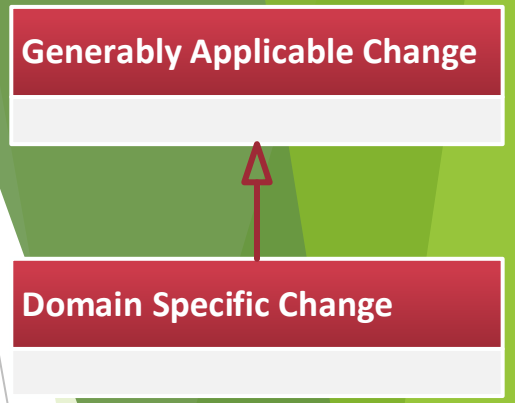


Apply the change to the original model element





Applying Change Type



3. Searching and getting Generally applicable change types according to Domain specific types to complete it (G# - G1, G2)

6. Introducing Domain independent code scheme for each Generally applicable change

4. Introducing aspects as Generally applicable change type

2. Indentifying domain specific changes (D# - D1, D2)

5. Introducing aspects to complete Domain specific change

7. Incorporating adaptation in the context of Domain specific change usually as kind of refinement

1. Developer chooses change request

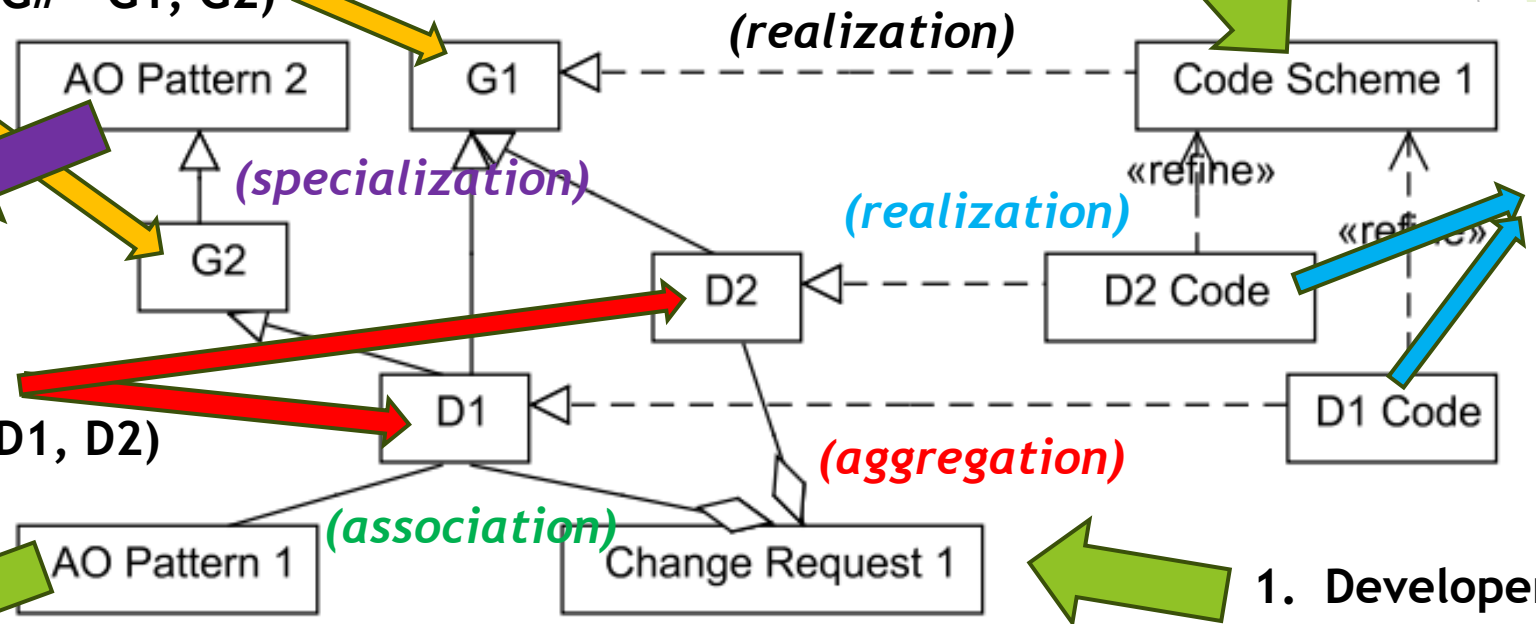


Figure 1. Generally applicable and domain specific changes

Source: **Aspect-Oriented Change Realizations and Their Interaction** Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

Integration Changes:

One Way Integration: Performing Action After Event

```
public aspect PerformActionAfterEvent {
  pointcut methodCalls(TargetClass t, int a):...;
  after( /* captured arguments */):
          methodCalls( /* captured arguments */)
  {
          performAction( /* captured arguments */);
  }
  private void performAction( /* arguments */)
  {
    /* action logic */
  }
}
```

such as Notification of incoming events

the integrating application notifies the integrated application of relevant events

Capturing certain events

Performed action after event

Such as a post to the newsletter sign-up/sign-out script and pass it the e-mail address and name of the newly signed-up or deleted affiliate

Applied Patterns

Method Substitution

```
public aspect MethodSubstition {
   pointcut methodCalls(TargetClass t, int a): . . .;
   ReturnType around(TargetClass t, int a):
                                    methodCalls(t, a) {
      if (. . .) {
         . . . } // the new method logic
      else
         proceed(t, a);
   }
}
```

Souirce: Aspect-Oriented Change Realizations and Their Interaction Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

Boudary Control

```
pointcut prohibitedRegion():
   (within(application.Proxy)
   && call(void *. * (..)))
   ||  (within(application.campaigns. +)
   && call(void *. * (..)))
   ||  within(application.banners. +)
   ||  call(void Affiliate .decline (..))
   ||  call(void Affiliate .delete (..));
```

Enumeration Modification Change

Introducing new enumeration value:

```
public aspect NewEnumType {
    public static EnumValueType
            EnumType.NEWVALUE = new EnumValueType(10, "");
}
```

Source: Aspect-Oriented Change Realizations and Their Interaction Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

Changing a Change Using Aspects

-separation of crosscutting concerns in the application
=> IMPROVING MODULARITY
=> MAKES FURTHER CHANGES EASIER

Aspect-oriented refactoring

1. Use the primitive pointcut to capture execution of all advices:

adviceexecution()

2. Annotating and accessing the advices:

within()/withincode()

Or handling multiple advices by annotating each with the pointcut:

@annotation()

Capturing Change Interactions By Feature Models

Aspect-Oriented Change Realizations and Their Interaction Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

- Mutual change dependencies of some change realizations
- Dependencies on underlying system affected by other change realizations

Possible escalation into a serious problems

FEATURES

-virtually pluggable
as variable features

FEATURE MODELING

Only on will be in resulting system

-variable features are used as the systems extensions

-including variability among changes

Parent feature must be included

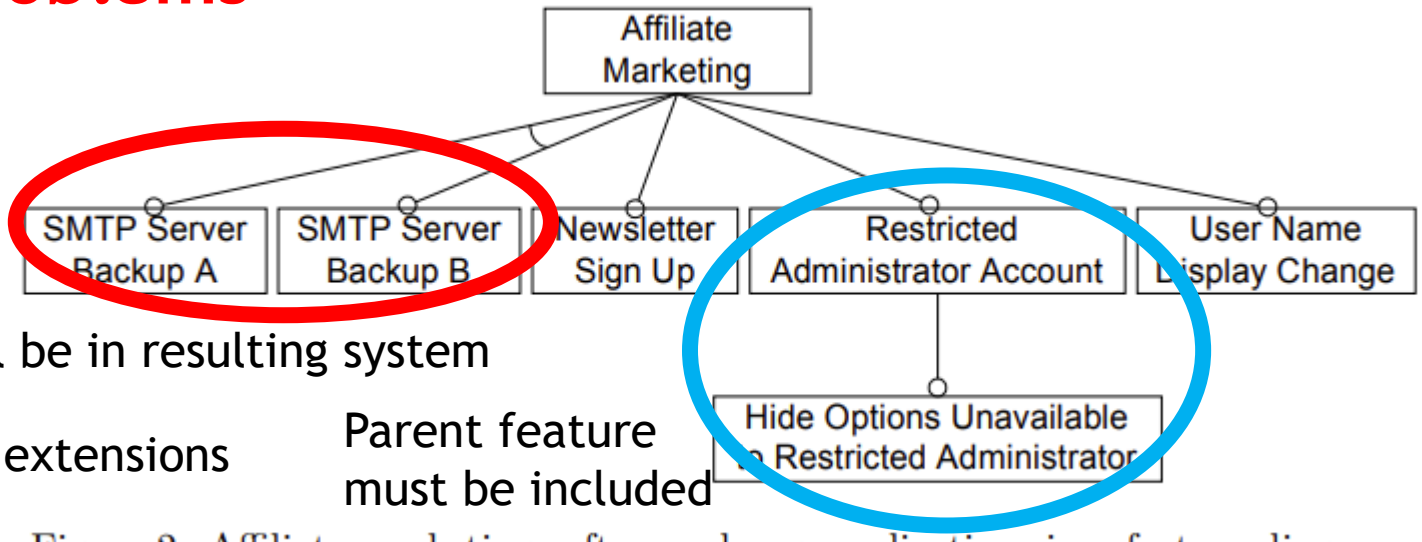


Figure 2. Affiliate marketing software change realizations in a feature diagram

Capturing change interactions with a feature diagram…

… Modeling change realizations

change realizations as features

affected software concept as feature model

change interaction as each dependency in feature model

-indirect change dependencies may represent indirect change interactions

Determining if features interact REQUIRES FURTHER ANALYSIS OF SEMENTICS

Beyond capabilities of feature modeling!

Direct Change Interactions

-affecting common join points

Occurs among alternative features

Feature and its subfeatures

- Parent feature must be included

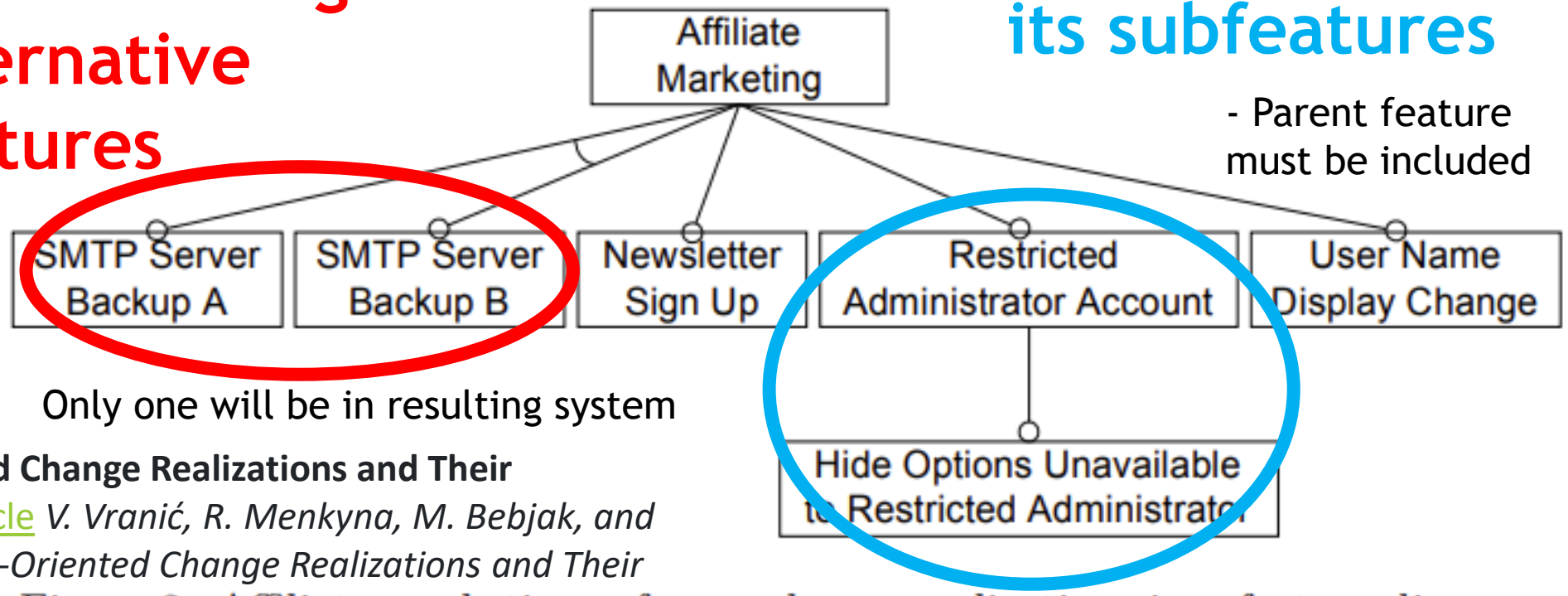


Figure 2. Affiliate marketing software change realizations in a feature diagram

Only one will be in resulting system

Aspect-Oriented Change Realizations and Their Interaction Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

Feature model in Czarnecki-Eisenecker basic notation

Partial Feature Model Construction

From Bottom Up

Grouping reached changes in a common subtree
-identifying parent features

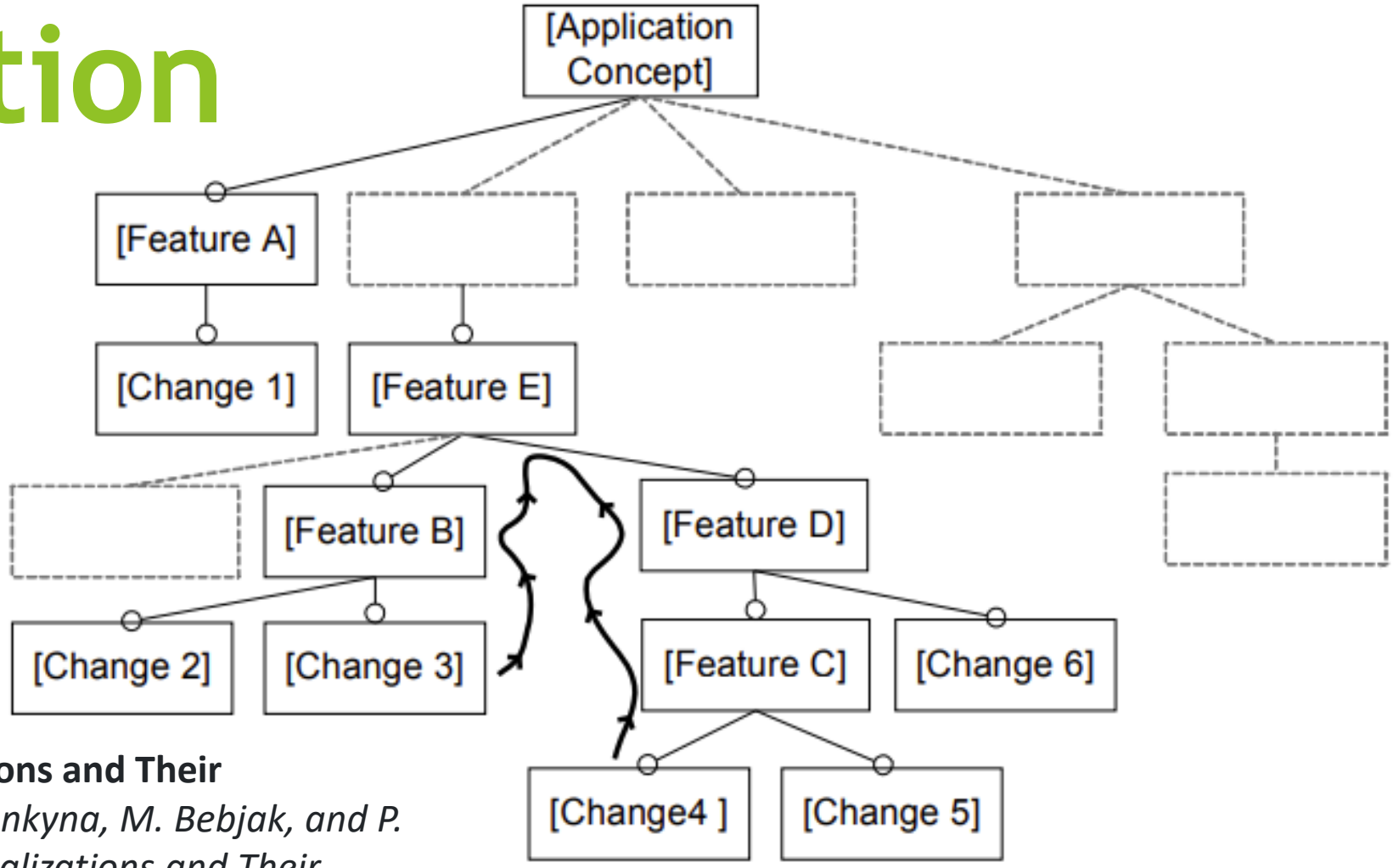


Figure 3. Constructing a partial feature model

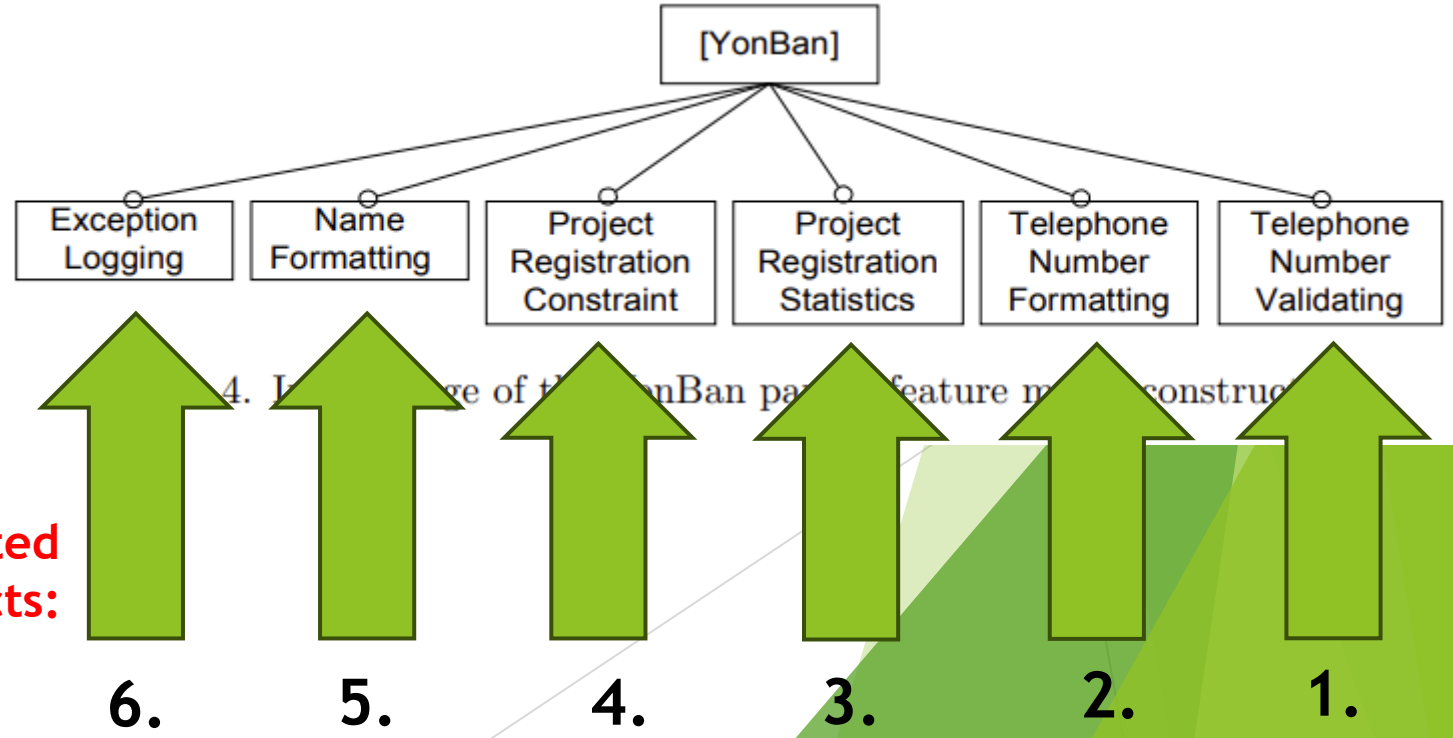
Aspect-Oriented Change Realizations and Their Interaction Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

Demonstration: YonBan

- **1.** Telephone Number Validating (realized as Performing Action After Event): to validate a telephone number the user has entered
- **2.** Telephone Number Formatting (realized as Additional Return Value Checking/Modification): to format a telephone number by adding country prefix
- **3.** Project Registration Statistics (realized as One Way Integration): to gain statistic information about the project registrations
- **4.** Project Registration Constraint (realized as Additional Parameter Checking/Modification) to check whether the student who wants to register a project has a valid e-mail address i his profile
- **5.** Exception Logging (realized as Performing Action After Event): to log the exceptions thrown during the program execution
- **6.** Name Formatting (realized as Method Substitution): to change the way how student names are formatted.

Implemented with aspects:

Aspect-Oriented Change Realizations and Their Interaction Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*



Identifying Parents and Final Refinement

DEPENDENCIES

DEPENDENCY (OF ENTERING THE NAME WHILE REGISTERING THE USER)

Open concept of a system (using [])
due to no other specification

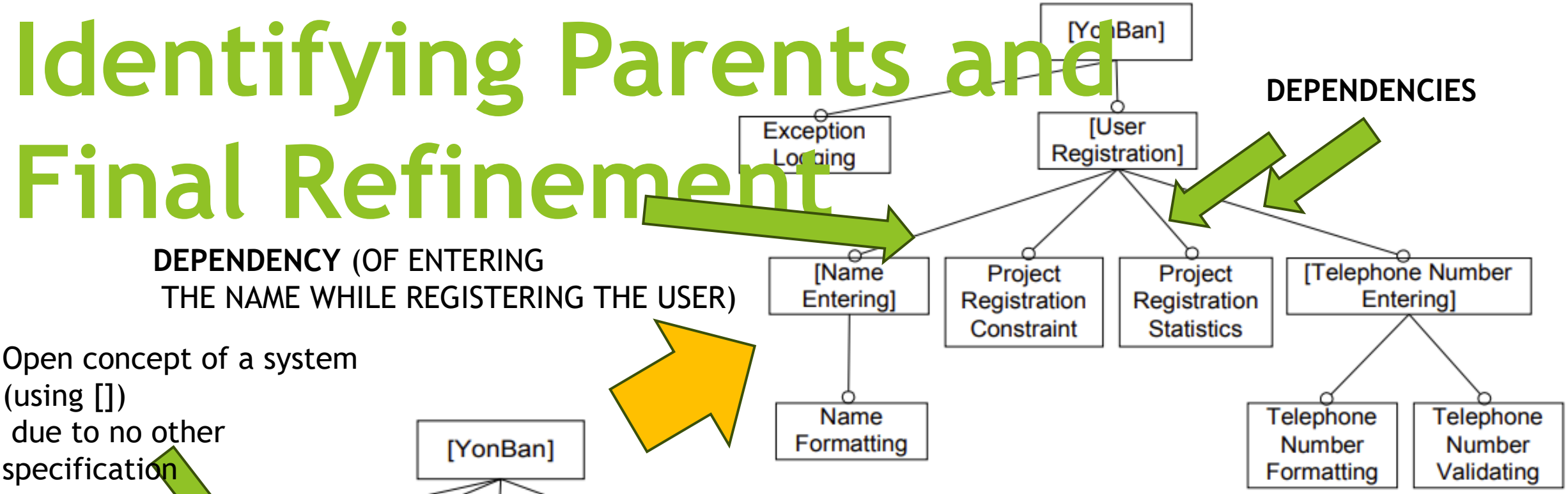


Figure 6. The final YonBan partial feature model

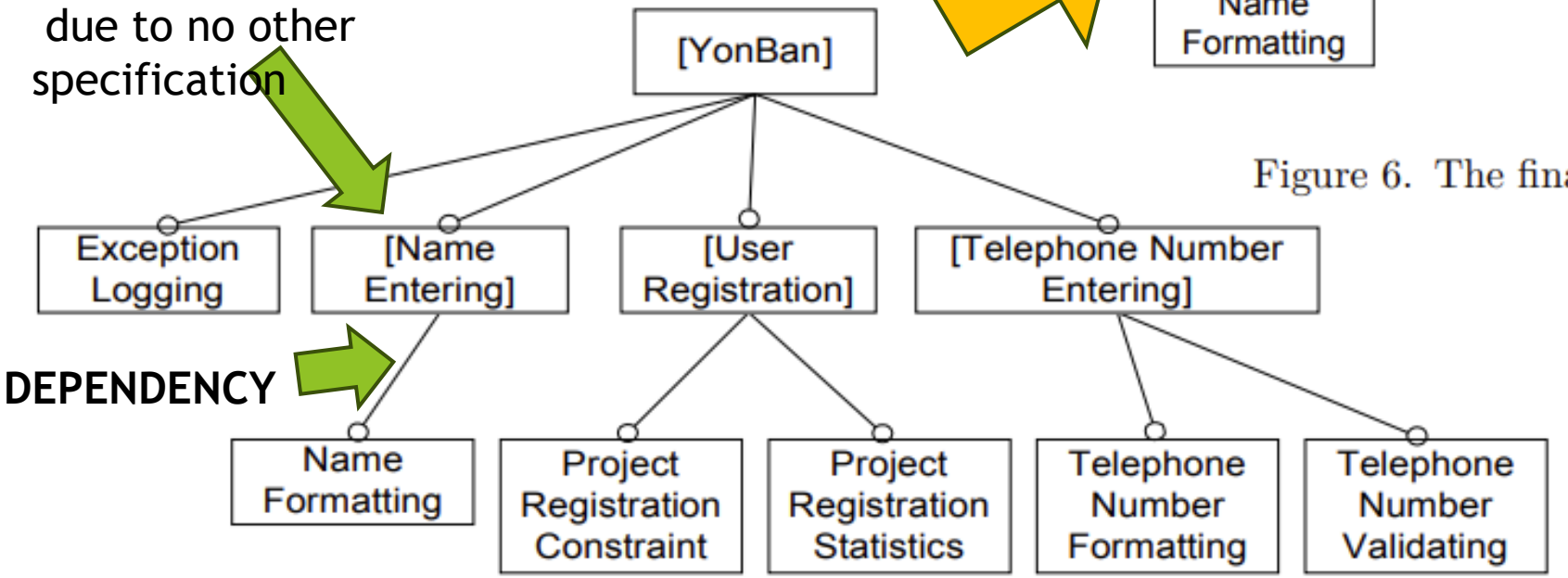


Figure 5. Identifying parent features in YonBan partial feature model construction

Aspect-Oriented Change Realizations and Their Interaction Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

Resolving Conflicts

The code that implements the parent feature altered by one of the sibling change features can be dependent on the code altered by another sibling change feature or vice versa.

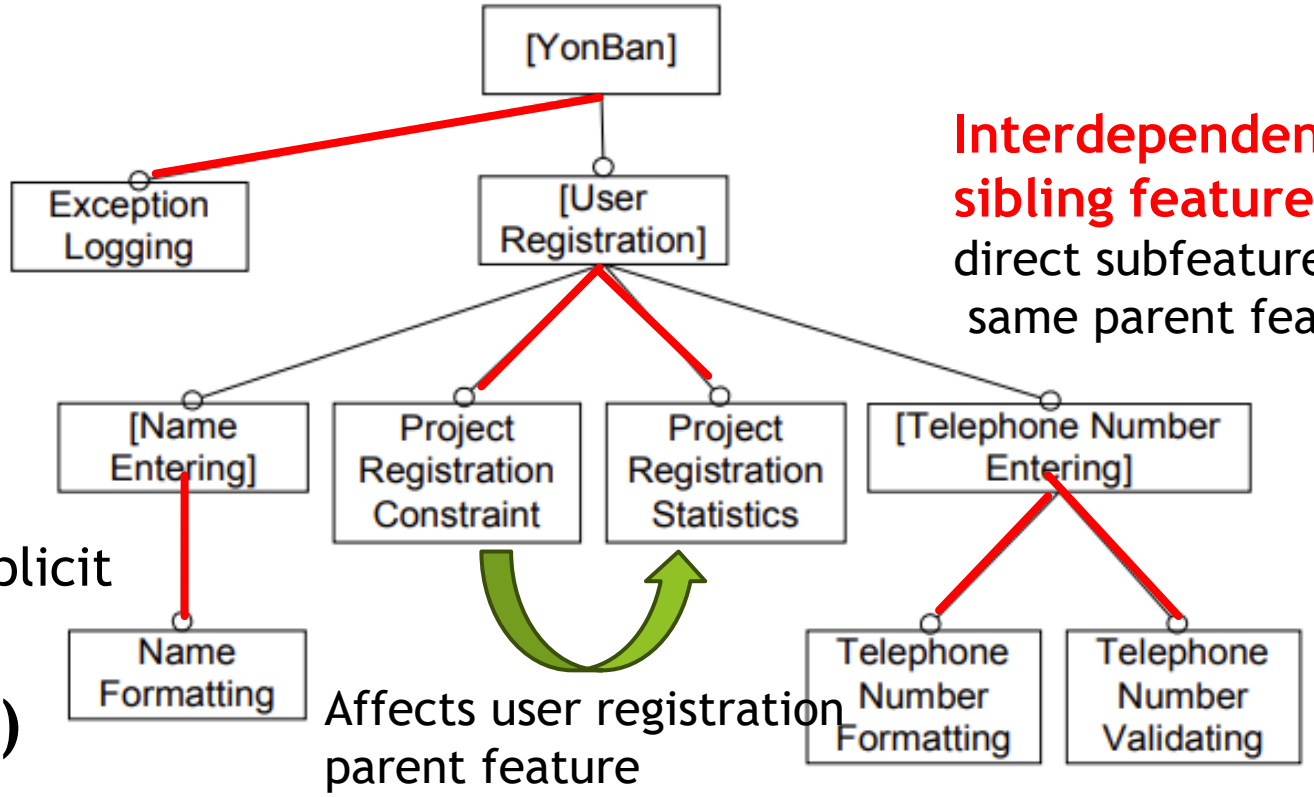


Figure 6. The final YonBan partial feature model

Interdependence of sibling features - direct subfeatures the same parent feature

Aspect Priority implicit setting:

call()
vs
execution()
poincuts

Affects user registration parent feature

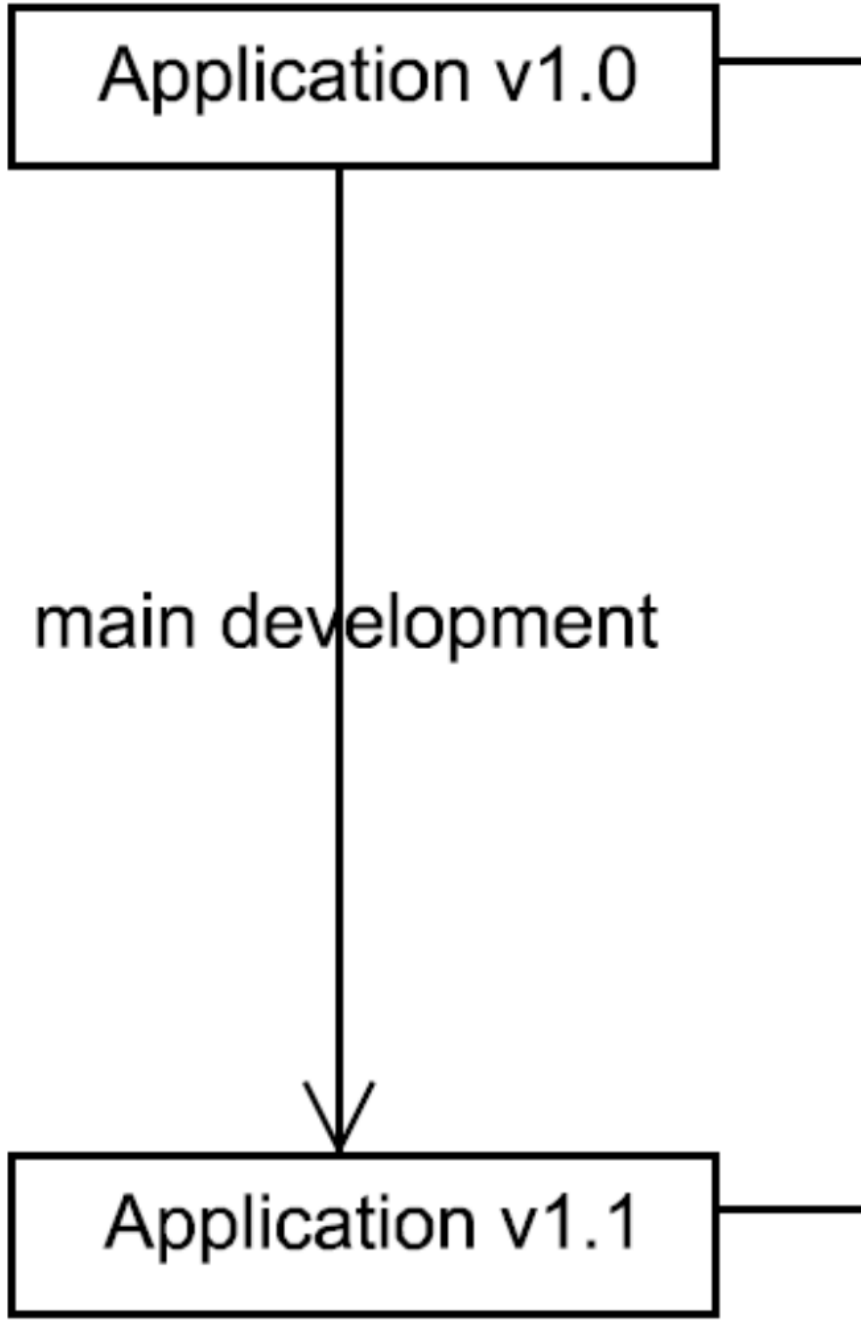
Resolving dependencies between aspects by setting priorities to aspects

Aspect-Oriented Change Realizations and Their Interaction Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

Change Realization

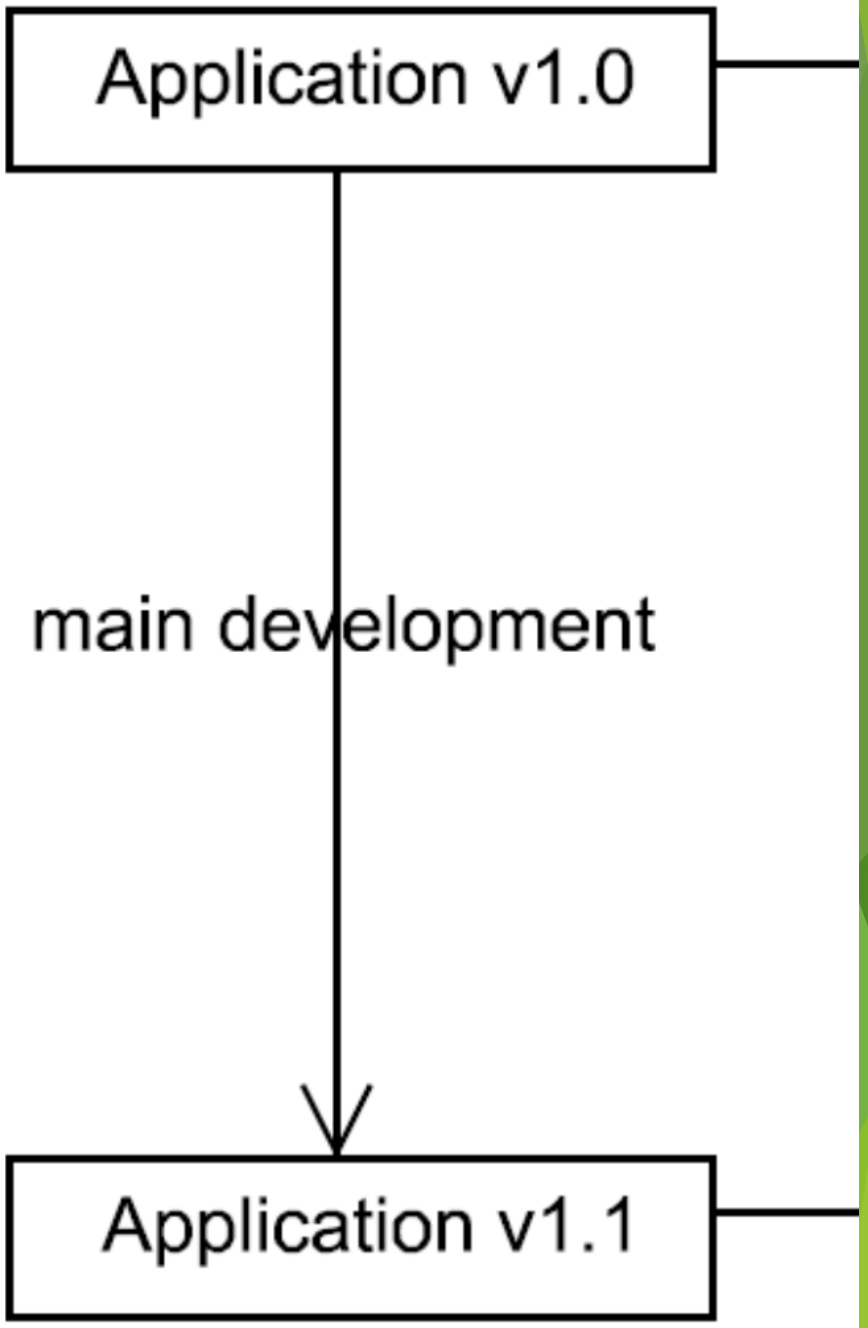
Implementing a change separately ⟺ Directly incorporating a change to source code

Aspect-Oriented change realization

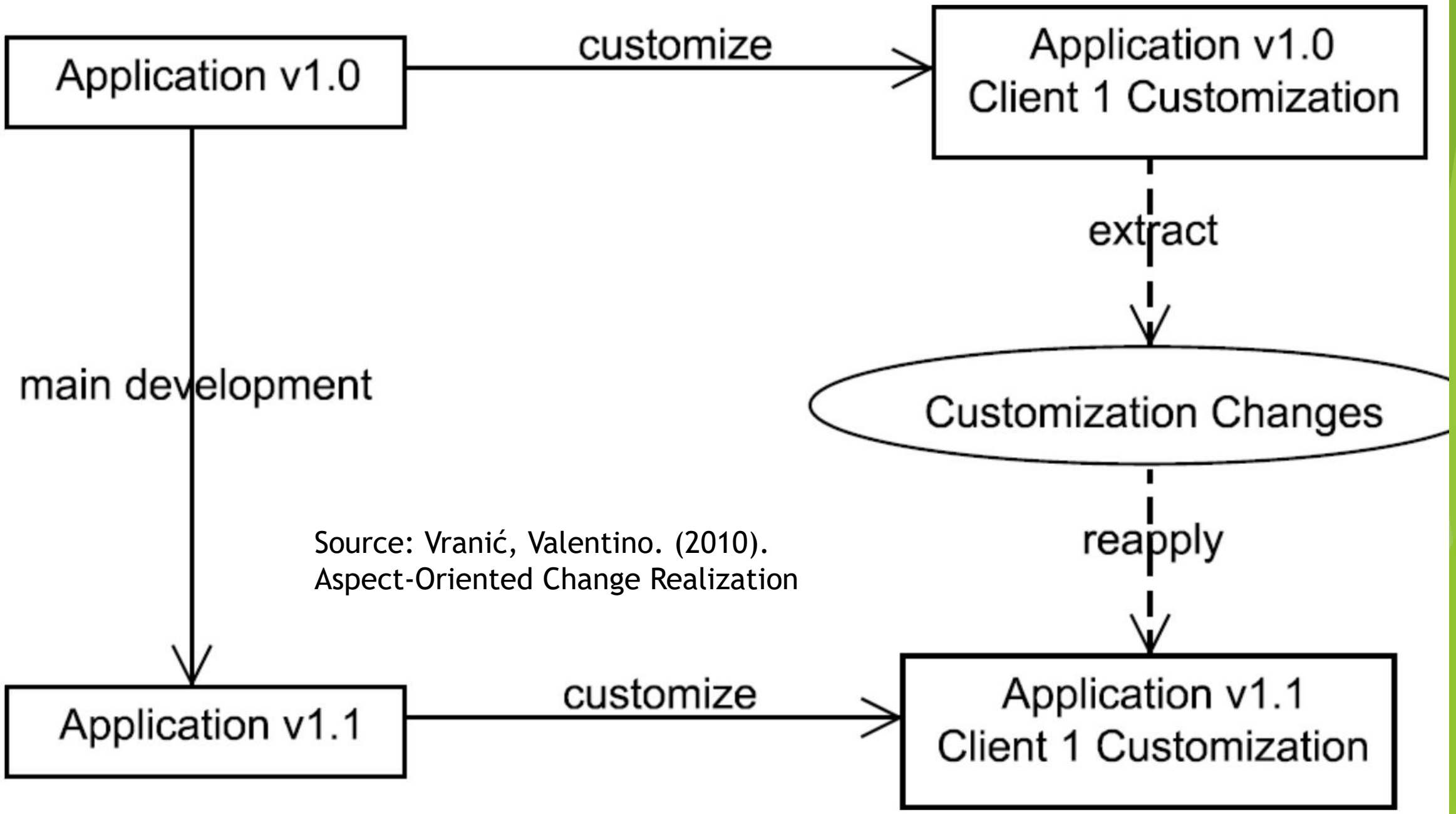


Aspect-Oriented Change Realizations and Their Interaction Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

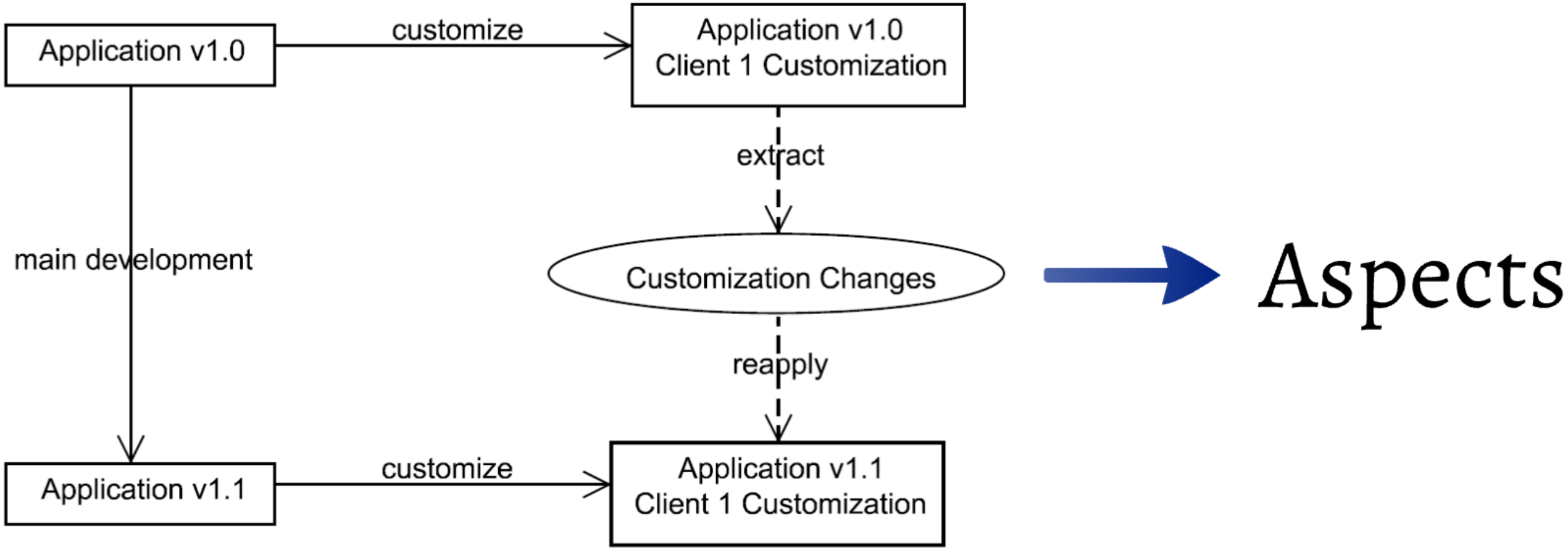
Aspect-Oriented Change Realization



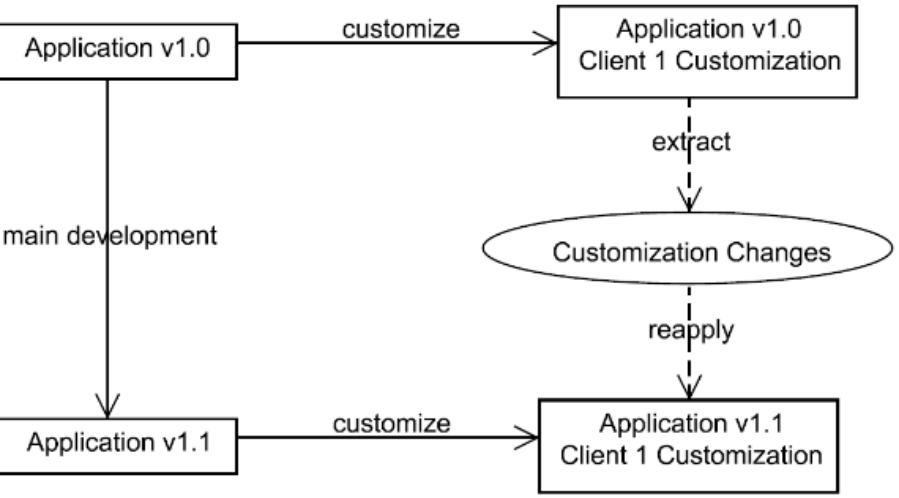
Aspect-Oriented Change Realizations and Their Interaction Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*



Source: Vranić, Valentino. (2010).
Aspect-Oriented Change Realization



Source: Aspect-Oriented Change Realizations and Their Interaction Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*



Aspects

Aspect-oriented change realization

Aspect-Oriented Change Realizations and Their Interaction Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*

References

- **Trygve Reenskaug and James O. Coplien:** The DCI Architecture: A New Vision of Object-Oriented Programming March 20, 2009
- **Savkin:** Data Context Interaction: The Evolution of the Object Oriented Paradigm
- **Perdek, Jakub, and Valentino, Vranić. "Lightweight Aspect-Oriented Software Product Lines with Automated Product Derivation."** In New Trends in Database and Information Systems (pp. 499–510). Springer Nature Switzerland, 2023.
- **FIGUEIREDO, Eduardo, Nelio CACHO, Claudio SANT'ANNA, Mario MONTEIRO, Uira KULESZA, Alessandro GARCIA, Sergio SOARES, Fabiano FERRARI, Safoora KHAN, Fernando FILHO a Francisco DANTAS, Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability.** 2008, s. 10.
- **JACOBSON, Ivar, Martin GRISS a Patrik JONSSON, 1997. Software Reuse: Architecture, Process and Organization for Business Success.** USA: ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-92476-5.
- **KOHUT, Jan a Valentino VRANIC,** Guidelines for using aspects in product lines: 2010 IEEE 8th International Symposium on Applied Machine Intelligence and Informatics (SAMI) [online]. Herlany: IEEE, s. 183–188 [cit. 30.9.2021]. ISBN 978-1-4244-6422-7. Dostupné na: doi:10.1109/SAMI.2010.5423741
- **Developing Applications with Aspect-Oriented Change Realization:** Article *Valentino Vranić, Michal Bebjak, Radoslav Menkyna, and Peter Dolog. Developing Applications with Aspect-Oriented Change Realization. In Proceedings of 3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2008, Revised Selected Papers, LNCS 4980, October 2008, Brno, Czech Republic, Springer, 2011*
- **Aspect-Oriented Change Realization Based on Multi-Paradigm Design with Feature Modeling** Article *Radoslav Menkyna and Valentino Vranić. Aspect-Oriented Change Realization Based on Multi-Paradigm Design with Feature Modeling. In Proceedings of 4th IFIP TC2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2009, Revised Selected Papers, LNCS 7054, October 2009, Krakow, Poland, Springer, 2012.*
- **Aspect-Oriented Change Realizations and Their Interaction** Article *V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog. Aspect-Oriented Change Realizations and Their Interaction. e-Informatica Software Engineering Journal, 3(1):43-58, 2009*