Aspects in Analysis And Design

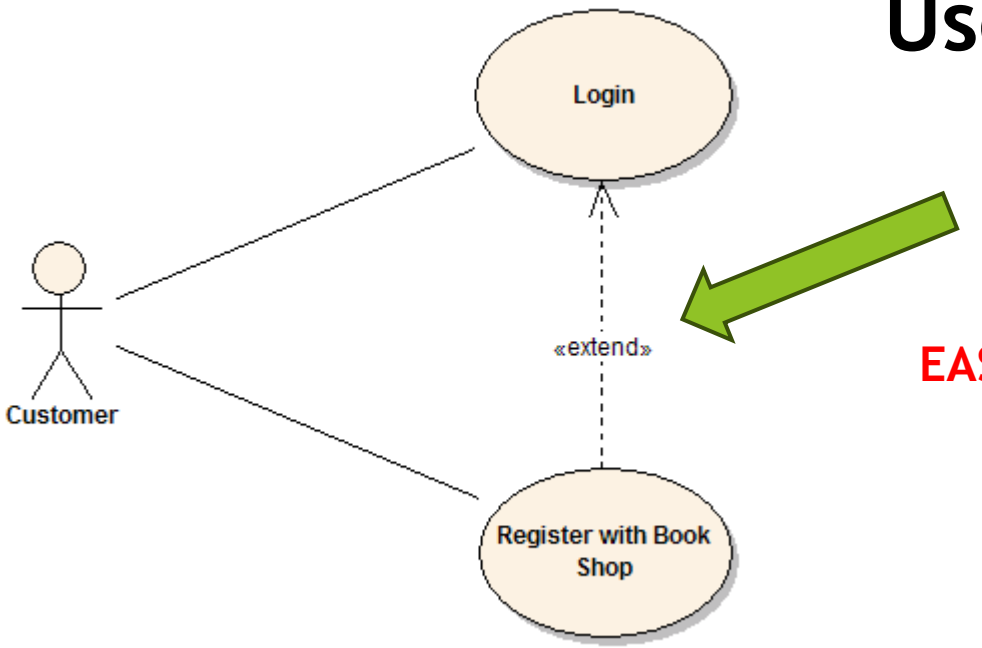
Themes and JPDD

Analysis

AOSD via Use Cases

- **Ivar Jacobson** - the one who first came with this idea in 2003

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



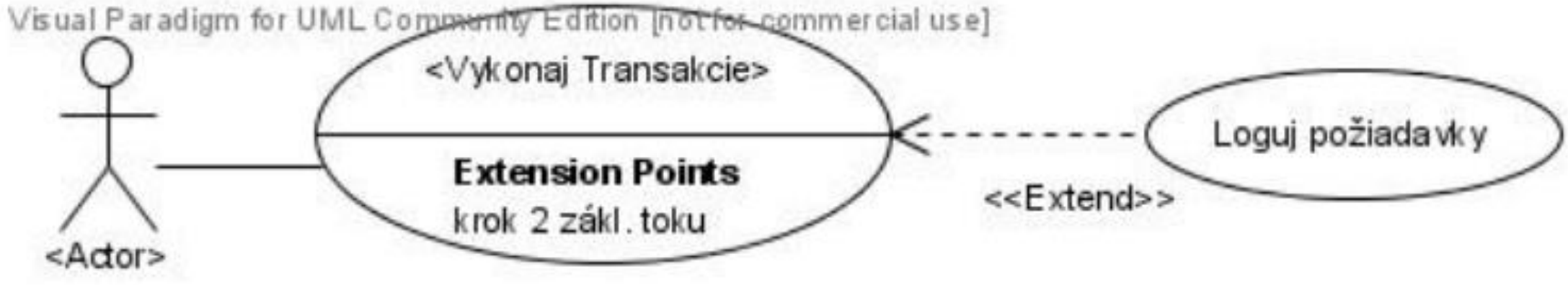
Use-case modularity problem

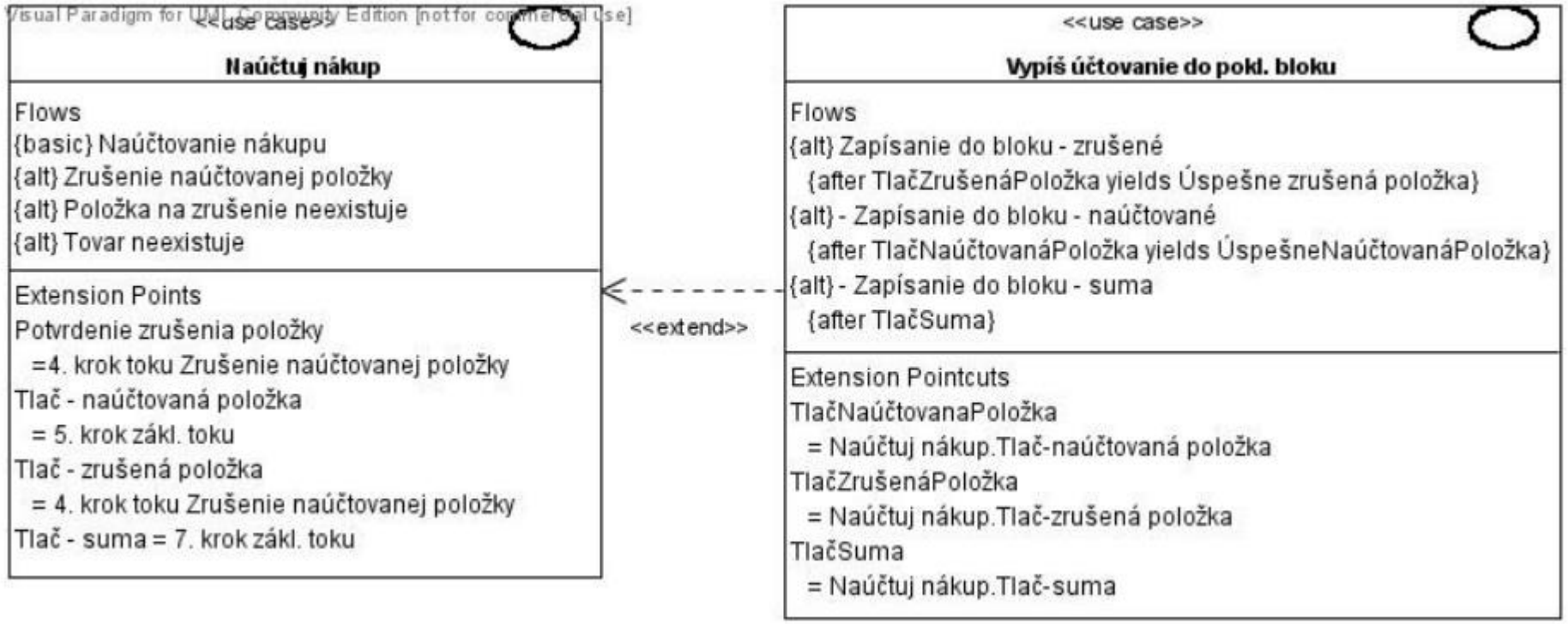
Previously unsupported in analytical models and in implementational environments

EASILY representable with aspect-oriented programming - solved by AOP programming language

-moved to implementation level

Prebrané z https://sparxsystems.com/images/screenshots/UCLogin.gif





Prebrané z [Bc. Pavol Michalco: PRÍPADY POUŽITIA A TÉMY V PRÍSTUPE THEME/DOC]

Peer Use Cases

PEER USE CASES: *Use cases without binding between each other*

-> independent of each other

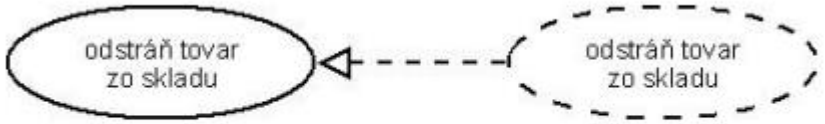
-> can be processed in paralell

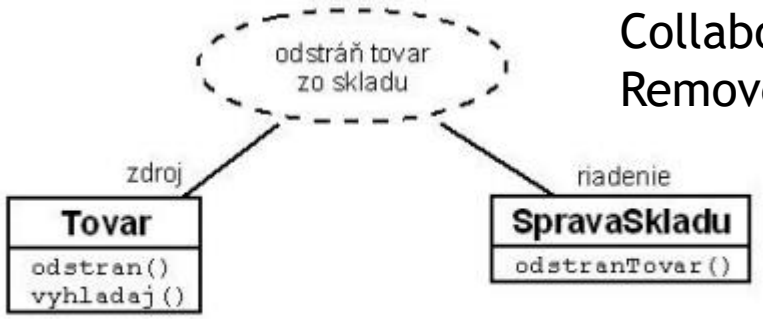
-> have an affect on shared/common entity

Separation of concerns can be problematic in peer/extension use cases

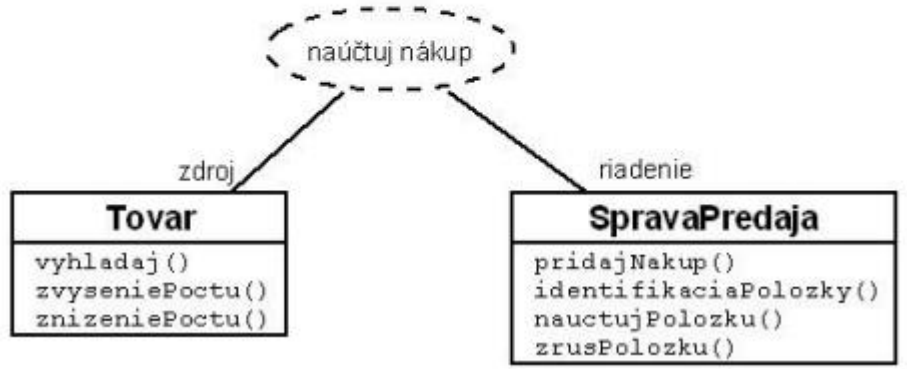
Collaboration diagrams:

Collaboration diagram: Accounting the purchase

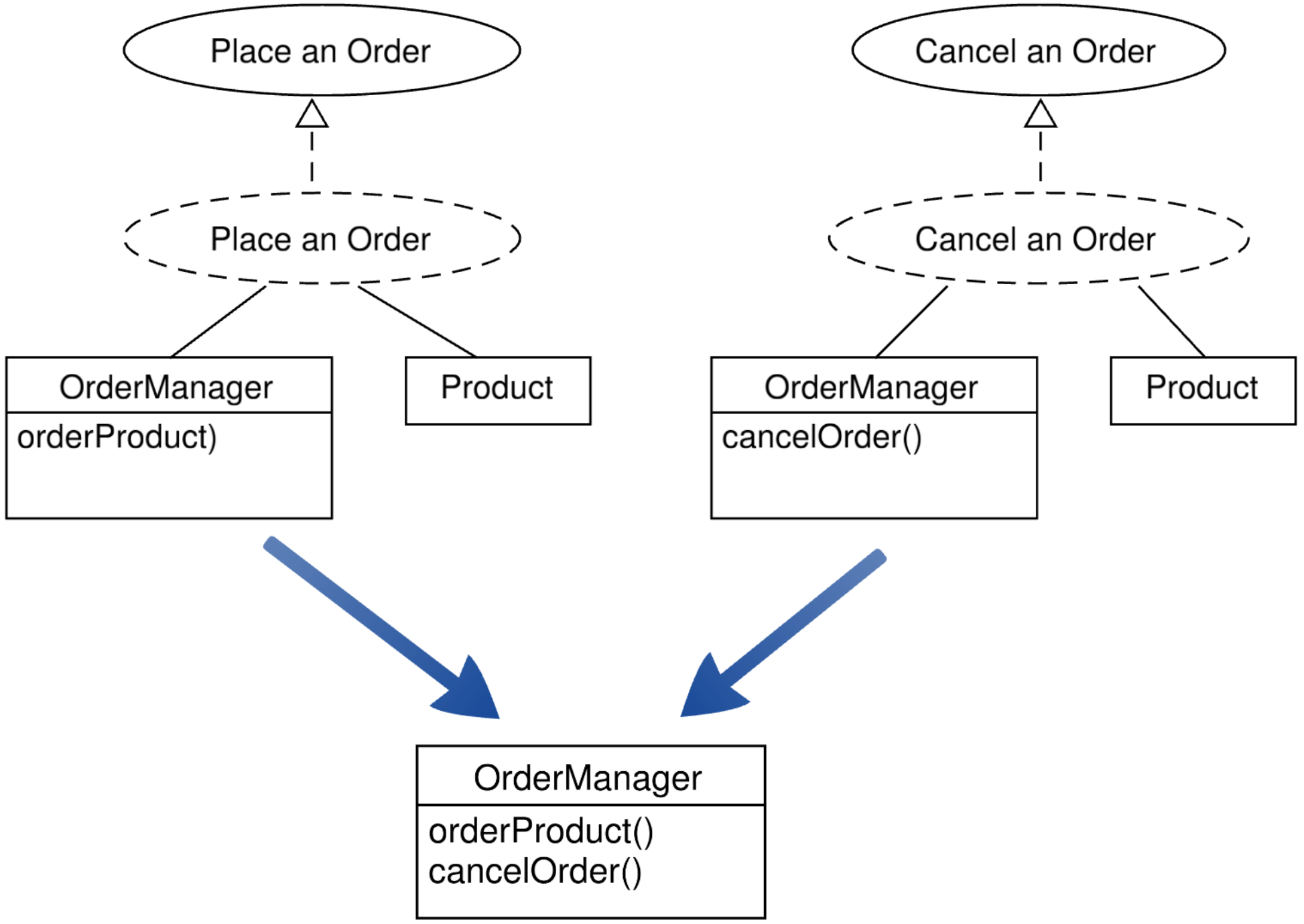




Collaboration diagram:
Remove goods from warehouse



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


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

Dynamically extending class

Prototype-based programming

1. Declaring class

```
class VerticePair {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
    getX() { return this.x; }
    getY() { return this.y; }
}
```

2. Instantiating class

```
var verticePair = new VerticePair(5, 6);
var newX = verticePair.x;
console.log(newX); //newX //(or) //to prints newX
```

3. Extending class dynamically

-possibly with new features… that can be then evolved independently

```
VerticePair.prototype.checkPoint = function() { if (this.x > 2) { throw new Error('Coordinate X is greater!');} }
```

4. Using the extension

```
verticePair.checkPoint();
```

Test

```
> class VerticePair {

      constructor(x, y) {
          this.x = x;
          this.y = y;
      }

      getX() { return this.x; }
      getY() { return this.y; }
  }

  var verticePair = new VerticePair(5, 6);
  var newX = verticePair.x;
  console.log(newX); //newX //to prints newX


  VerticePair.prototype.checkPoint = function() { if (this.x > 2) { throw new Error('Coordinate X is greater!');} }
  verticePair.checkPoint();
```

```
▶ Uncaught Error: Coordinate X is greater!                    VM266:17
      at VerticePair.checkPoint (<anonymous>:17:73)
      at <anonymous>:18:13
```

Dynamically extending object

Dynamic Object Modification

1. Declaring class

```
class VerticePair {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
    getX() { return this.x; }
    getY() { return this.y; }
}
```

2. Instantiating class

```
var verticePair = new VerticePair(5, 6);
var newX = verticePair.x;
console.log(newX); //newX //(or) //to print newX

var verticePair0 = new VerticePair(1, 6);
```

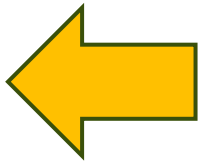
3. Extending class dynamically

-possibly with new features… that can be then evolved independently

```
verticePair.checkPoint = function() { if (this.x > 2) { throw new Error('Coordinate X is greater!');} }
```

4. Using the extension

```
verticePair0.checkPoint();
verticePair.checkPoint();
```



Is it successfull? Is extension bound to particular object?

Test

```
> class VerticePair {

      constructor(x, y) {
          this.x = x;
          this.y = y;
      }

      getX() { return this.x; }
      getY() { return this.y; }
  }

  var verticePair = new VerticePair(5, 6);
  var newX = verticePair.x;
  console.log(newX); //newX //to prints newX

  var verticePair0 = new VerticePair(1, 6);
  verticePair.checkPoint = function() { if (this.x > 2) { throw new Error('Coordinate X is greater!');} }

  verticePair.checkPoint();
```

```
▶ Uncaught Error: Coordinate X is greater!                    VM311:17
    at verticePair.checkPoint (<anonymous>:17:63)
    at <anonymous>:19:13
```

```
> class VerticePair {

      constructor(x, y) {
          this.x = x;
          this.y = y;
      }

      getX() { return this.x; }
      getY() { return this.y; }
  }

  var verticePair = new VerticePair(5, 6);
  var newX = verticePair.x;
  console.log(newX); //newX //to prints newX

  var verticePair0 = new VerticePair(1, 6);
  verticePair.checkPoint = function() { if (this.x > 2) { throw new Error('Coordinate X is greater!');} }
  verticePair0.checkPoint();
  verticePair.checkPoint();
```

```
▶ Uncaught TypeError: verticePair0.checkPoint is not a function    VM300:18
    at <anonymous>:18:14
```

Peer Use Cases

PEER USE CASES: *Use cases without binding between each other*

-> independent of each other

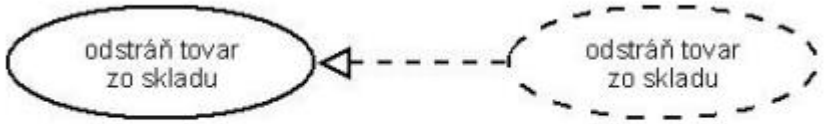
-> can be processed in paralell

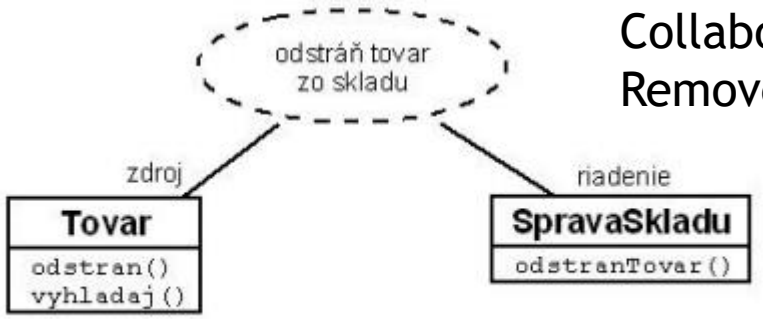
-> have an affect on shared/common entity

Separation of concerns can be problematic in peer/extension use cases

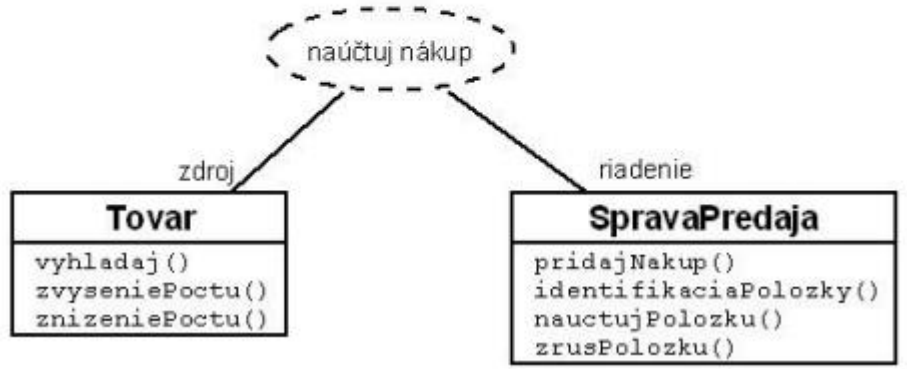
Collaboration diagrams:

Collaboration diagram: Accounting the purchase





Collaboration diagram:
Remove goods from warehouse

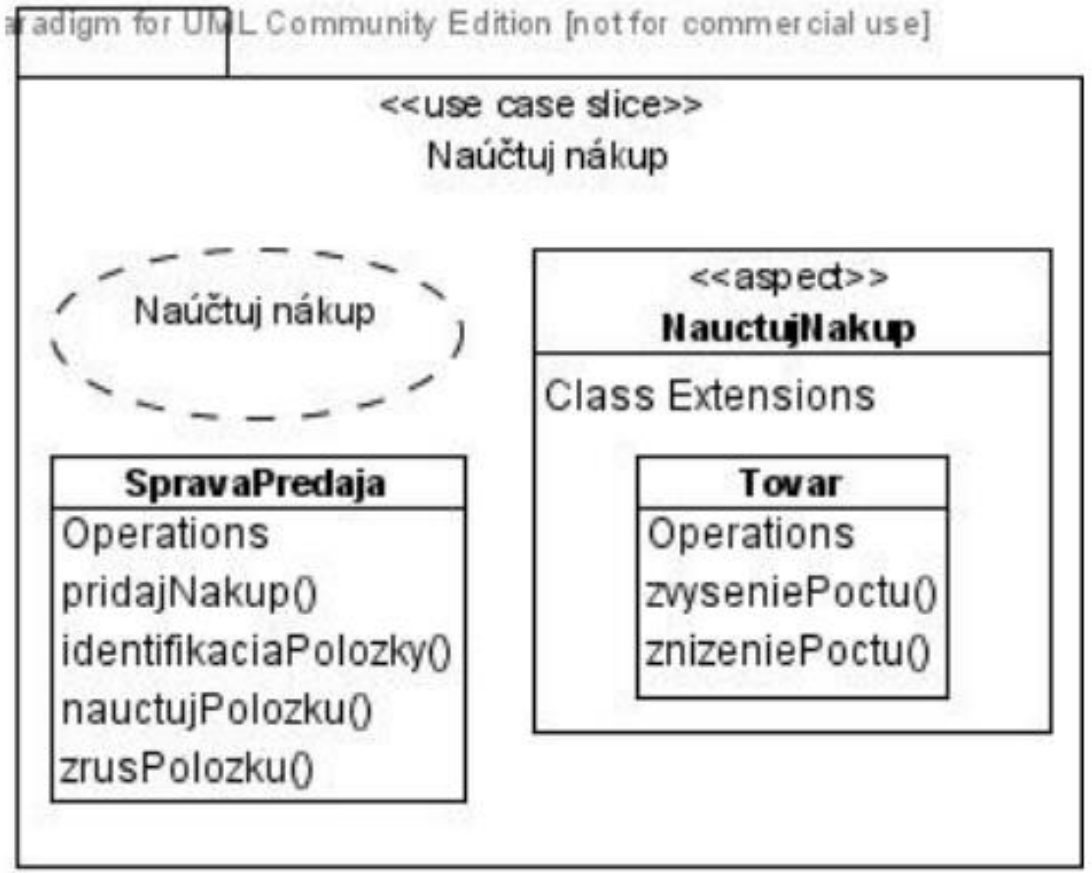


Zdroje z Bc. Pavol Michalco: PRÍPADY POUŽITIA A TÉMY V PRÍSTUPE THEME/DOC

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

Overlapping Problem in Peer Use Cases

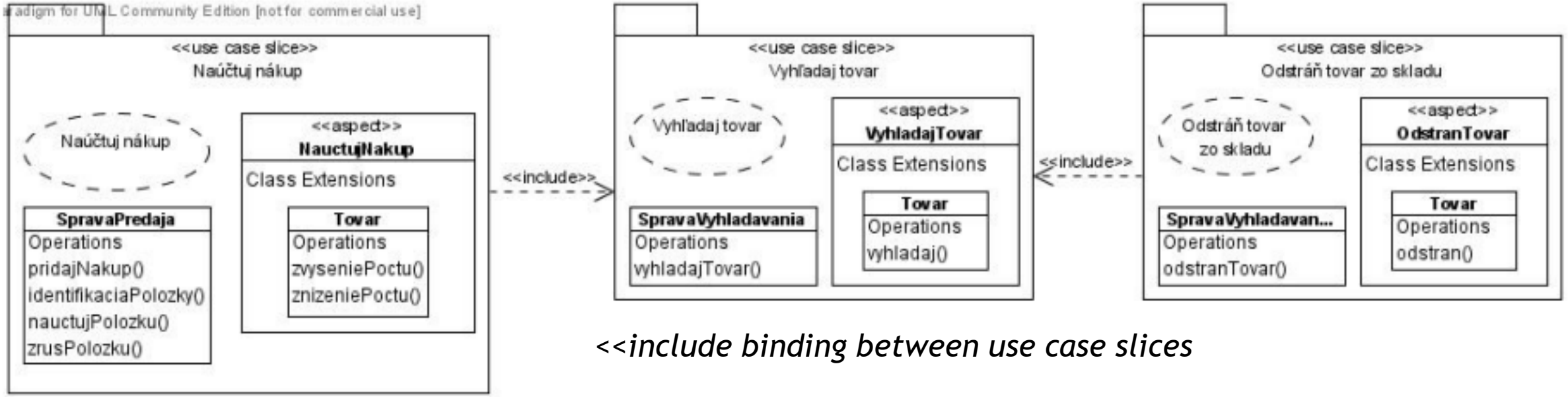
The functionality in form of vyhladaj() method is overlapping:

Sources from: [Bc. Pavol Michalco: PRÍPADY POUŽITIA A TÉMY V PRÍSTUPE THEME/DOC]

Solving Overlapping Problem in Peer Use Cases

1. **New use-case slice with handling of overlapping behaviour:**

-this use case slice is connected with <<include>>



<<include binding between use case slices

Source from: Bc. Pavol Michalco: PRÍPADY POUŽITIA A TÉMY V PRÍSTUPE THEME/DOC

2. Handling/defining in superior use case:

3. New non use-case slice with handling of overlapping behaviour:

NON USE CASE: domain representative

-provides access in one place to classes that are shared with many use case slices

-not containing aspects at all

Identifying the execution place of the extension of operation

Necessary information

STRUCTURAL CONTEXT

-position of the extension of operation in designed structure (package, class,…)

BEHAVIORAL CONTEXT

-point in execution flow where extension operation will extend the existing operation

->called **extension points** in **USE CASES** ⟺ ->called **join points** in **AOP**

call Pokladna.zrusPol()

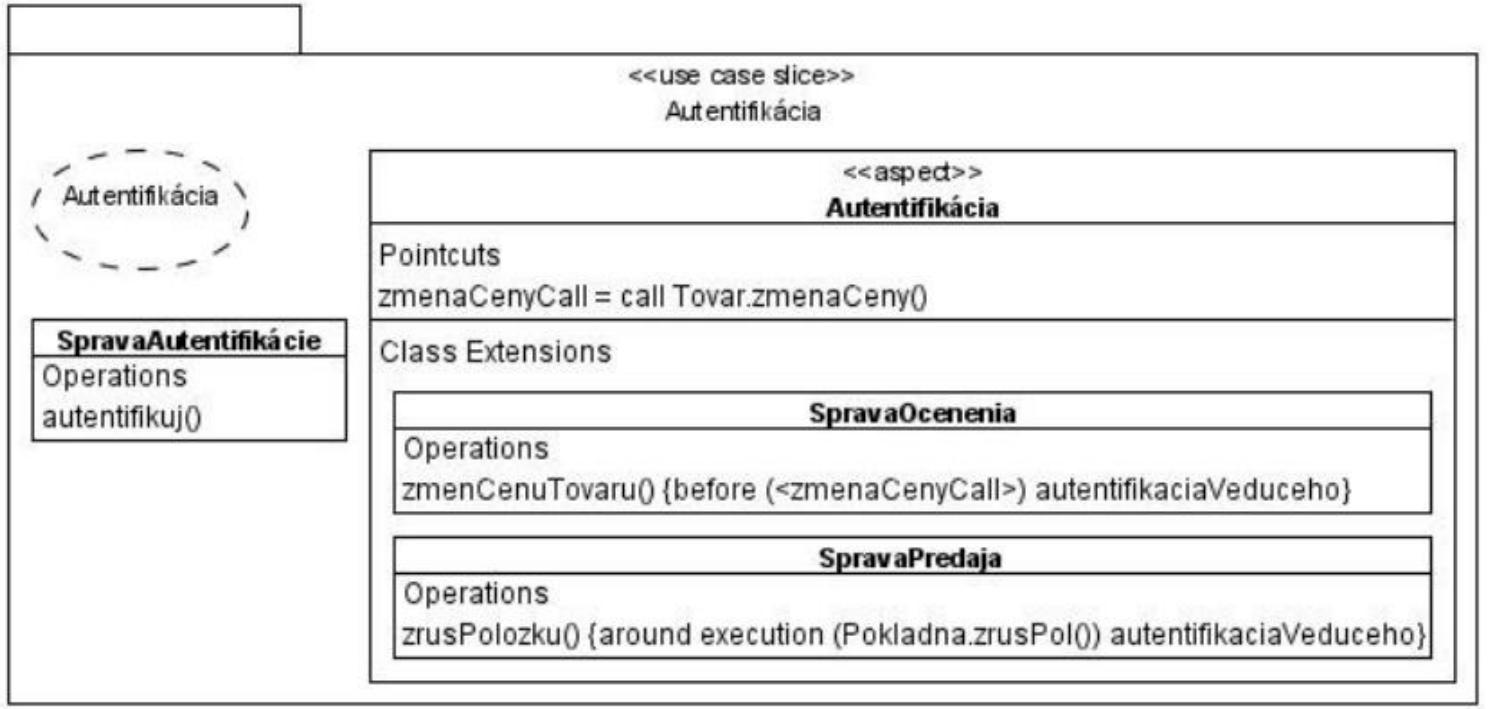
execution Pokladna.zrusPol()

Pri vykonaní

zrusPol() metódy triedy Pokladna

Aspects And Use Cases

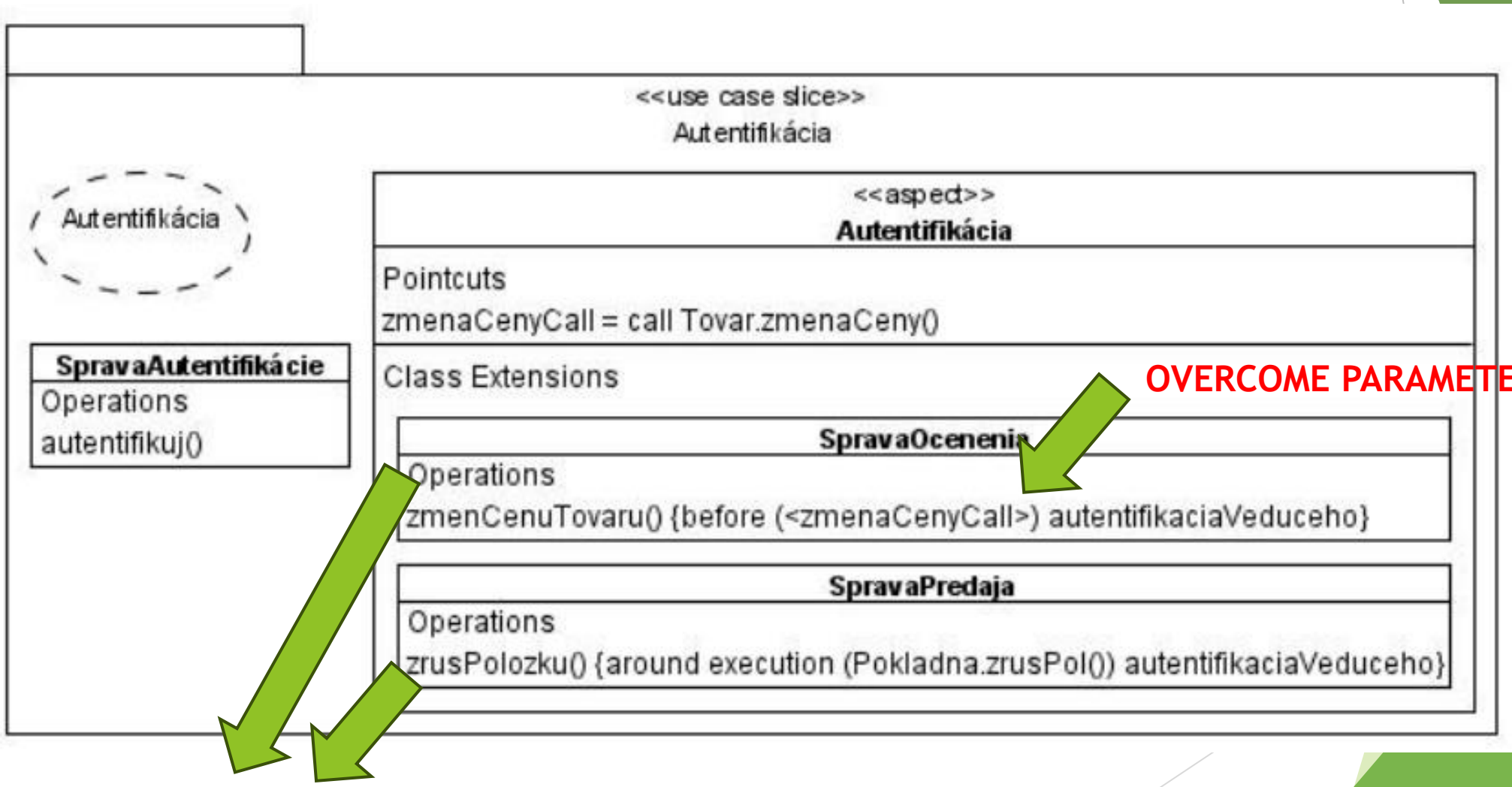
... Example ...




```
1.  package foo.Autentifikacia;
2.  import app.veduci.SpravaOcenenia ;
3.  import app.pokladnik.SpravaPredaja ;
4.  import domain.tovar.Pokladna ;
5.  import domain.tovar.Tovar ;
6.
7.  public aspect Autentifikacia {
8.  pointcut zmenaCenyCall() :
9.     withincode(void SpravaOcenenia.zmenCenuTovaru())
10.             &&  call(void Tovar.zmenaCeny()) ;
11.
12. before () : zmenaCenyCall() {
13. // kod autentifikacie veduceho
14. }
15.
16.       void       around()       :       withincode(void
SpravaPredaja.zrusPolozku())
17.    && execution(void Pokladna.zrusPol()) {
18.  // kod autentifikacie veduceho
19. }
20. }
```

Source from: Bc. Pavol Michalco: PRÍPADY POUŽITIA A TÉMY V PRÍSTUPE THEME/DOC

Details


<<use case slice>>
Autentifikácia
Autentifikácia
<<aspect>>
Autentifikácia
Pointcuts
zmenaCenyCall = call Tovar.zmenaCeny()
SpravaAutentifikácie
Operations
autentifikuj()
Class Extensions
OVERCOME PARAMETER
SpravaOcenenia
Operations
zmenCenuTovaru() {before (<zmenaCenyCall>) autentifikaciaVeduceho}
SpravaPredaja
Operations
zrusPolozku() {around execution (Pokladna.zrusPol()) autentifikaciaVeduceho}
CAN BE ALSO DEFINED IN INHERITED ASPECTS SEPARATELY

Themes in Analysis

Theme/Doc

- Themes in documentation (Theme/Doc), aspect-oriented analysis
- Do not identify themes for us
- Serves to identify and clarification in requirements of:
  - basic and crosscutting (aspect) themes **[IDENTIFICATION OF ASPECTS IN REQUIREMENTS]**
  - Relations between basic and crosscutting themes
- To visualize relations between themes and requirements in following views:

1. VIEW OF THEMES AND RELATIONS

2. CROSSCUTTING THEMES VIEW

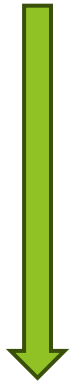
3. INDIVIDUAL VIEW

Operations With Themes According to Requirements

- Adding themes
- Removing themes
- Splitting themes
- Aggregating themes
- Postponing the requirements
- Associating with the requirements
- Associating/Connecting with the theme

Processes Ongoing in Themes

1. Deciding about themes
2. Observing responsibility of themes
3. Planning of a design

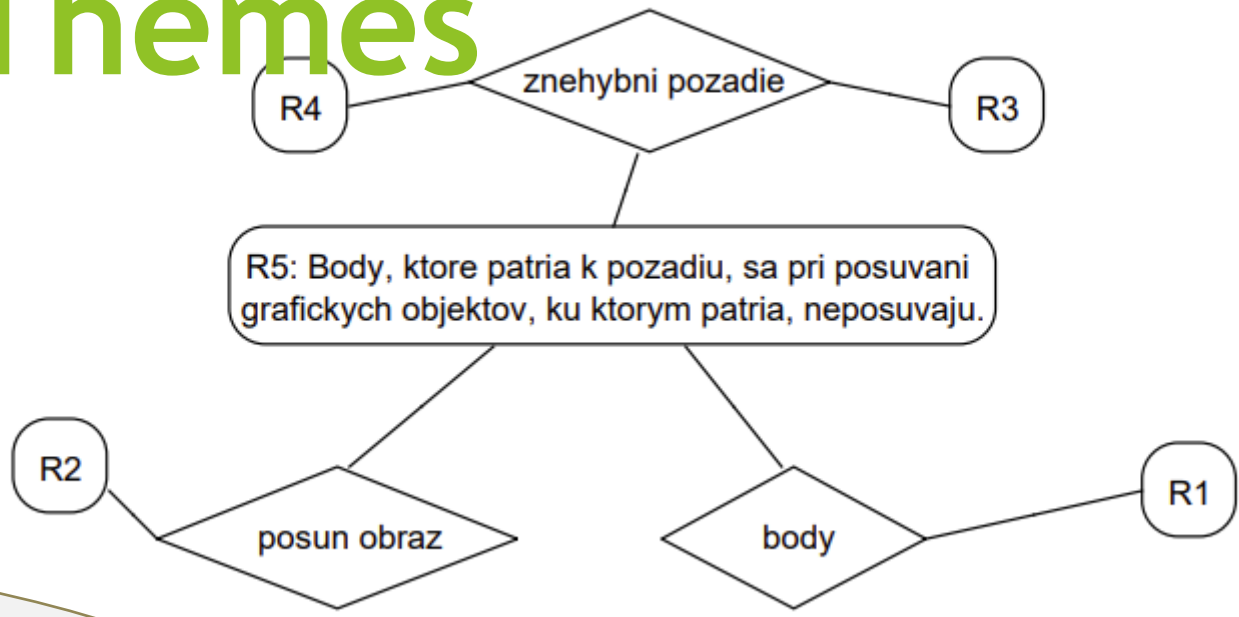


Processes run in this order

Not strictly separated

1. Deciding About Themes

- Identification of themes in requirements
  - How themes are connected (traced) with
  - related (for particular theme) requirements
- Operating on themes and relations between them



Theme ⟷ Requirement

- Displayed as rhombus

GRAPH

EDGES

- Connects only related themes with related requirements

- in basic view are only associations

- Displayed as rectangle with rounded edges

SHARED REQUIREMENTS

- If connected with multiple themes

ORPHANED REQUIREMENTS

- If connected with no theme

1. Deciding About Themes - Steps

Iterative approach

1. Determining initialization themes:

a) usually from use cases – each use case is usually one initialization theme

b) from requirements

Siobhàn Clarke and Elisa Baniassad. 2005. Aspect-Oriented Analysis and Design. Addison-Wesley Professional.

1. Identification of key functionalities in requirements and treating them as themes

-if themes are not coherent then will be further decomposed (split)

2. Identification of all verbs in requirements and treating them as themes

-themes have to be aggregated / generalized
-possibility to solve automatically with program

3. Combination of previous two cases

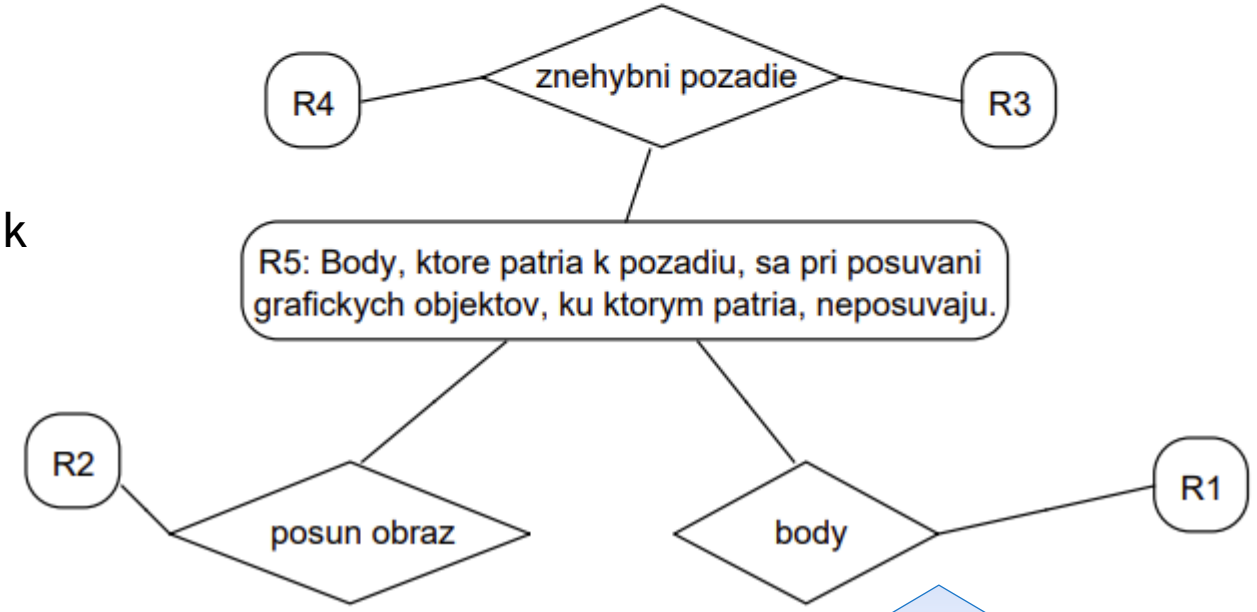
-requirements are searched for signs of features, behaviour or concerns

Examples of Basic Diagrams of Themes

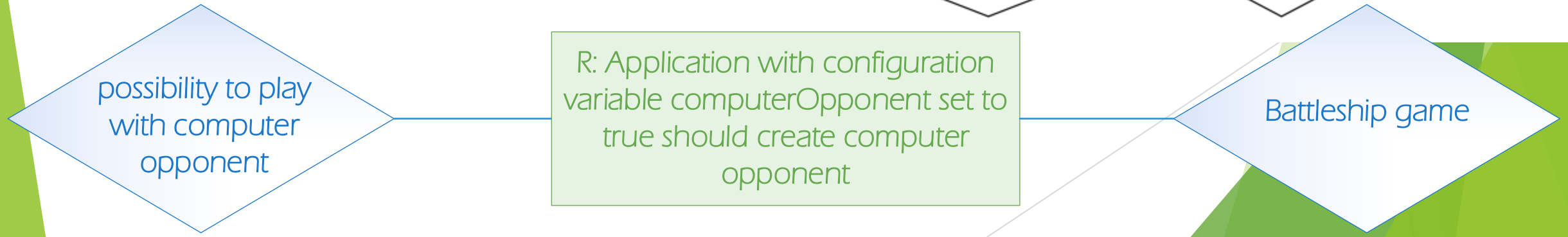
- R1: Pre body sa evidujú ich súradnice a to, či patria k pozadiu.
- R2: Obraz musí byť možné posúvať, pričom sa posunú všetky objekty, z ktorých pozostáva.

...

- R5: Body, ktoré patria k pozadiu, sa pri posúvaní grafických objektov, ku ktorým patria, neposúvajú.



Source: Aspekty v analýze a návrhu – prístupy Theme a JPDD Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoja softvéru, Valentino Vranić, 2017



2. Decomposing/splitting themes that are too general

-especially in cases where the same theme is used for different things such as in case of ***object removal*** that is split into:
erasing the drawn image or *moving puzzle away to the drawer*

3. Aggregating themes to simplify the graph (improve comprehension)

-from selected two or more themes we create new one in overview of themes with relations between them and in crosscutting view

-in **individual view** aggregated/grouped themes are displayed separately

4. Merging similar themes (themes which have the same purpose)

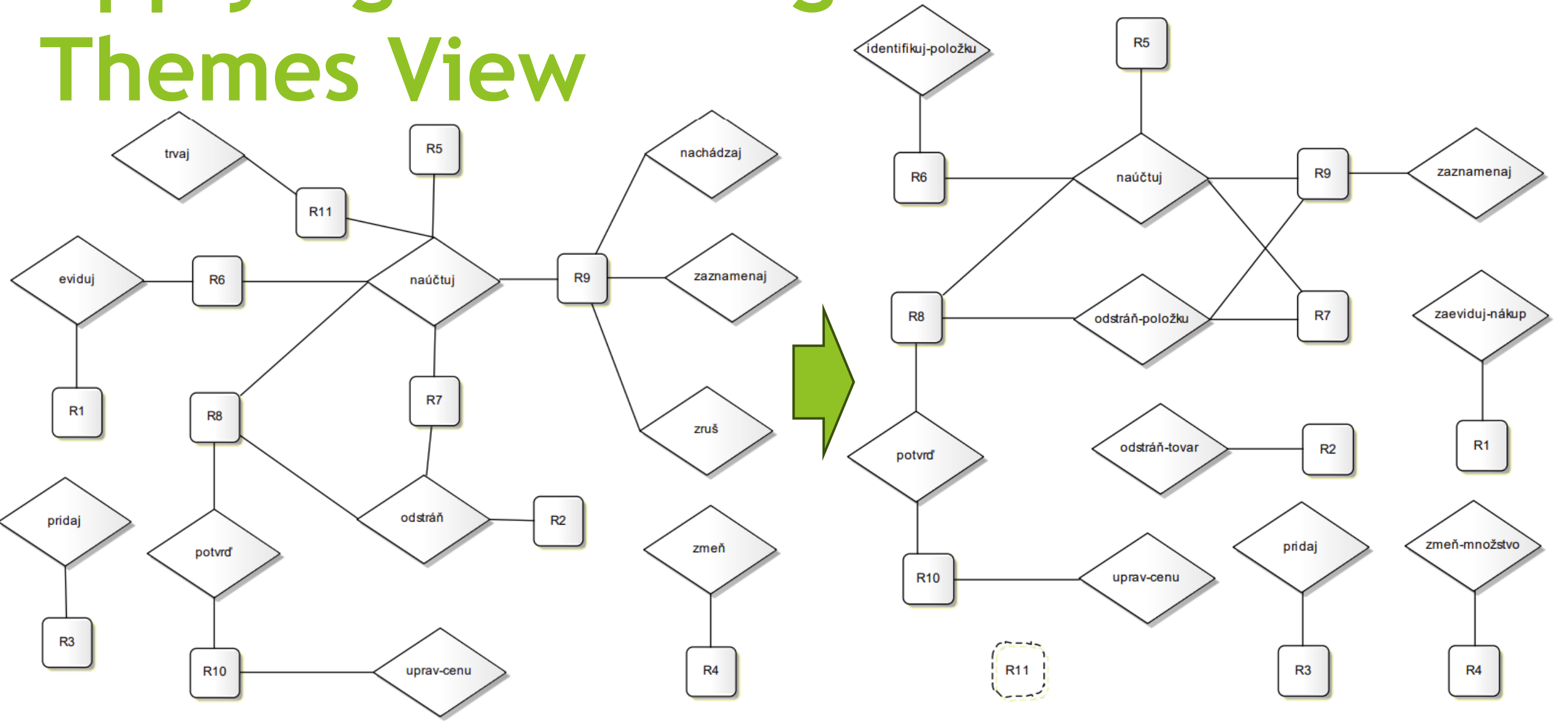
-mostly its caused due to synonyms that are extracted from requirements

Etc. same purpose: *Register item to canvas, bind item to canvas*

Etc. synonyms: *change, modify*

5. Deleting irrelevant themes (not characteristic for system)

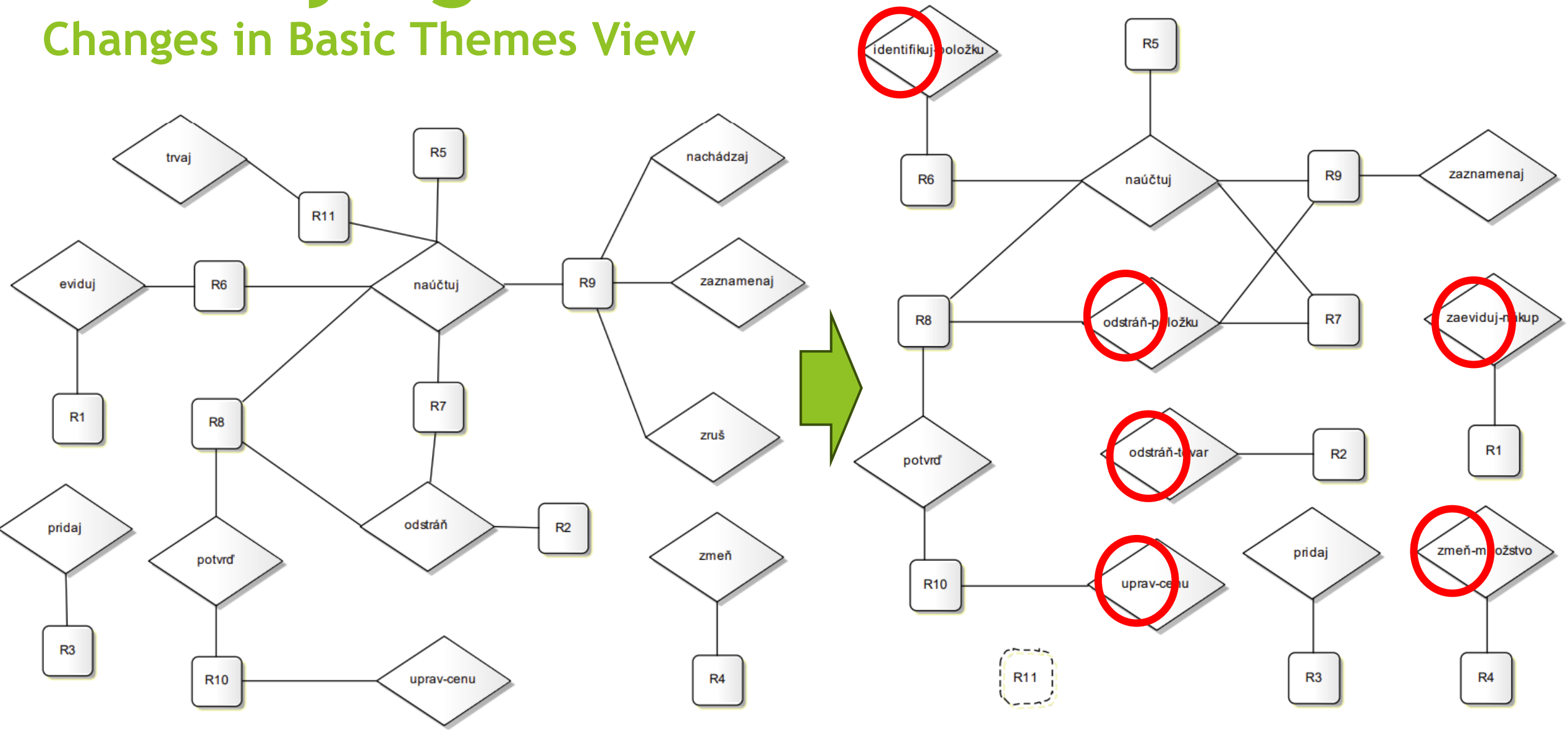
Applying the Changes in Basic Themes View



Source from: Bc. Pavol Michalco: PRÍPADY POUŽITIA A _TÉMY V PRÍSTUPE THEME/DOC

*we do not want to treat R11 requirement, left for next development phases

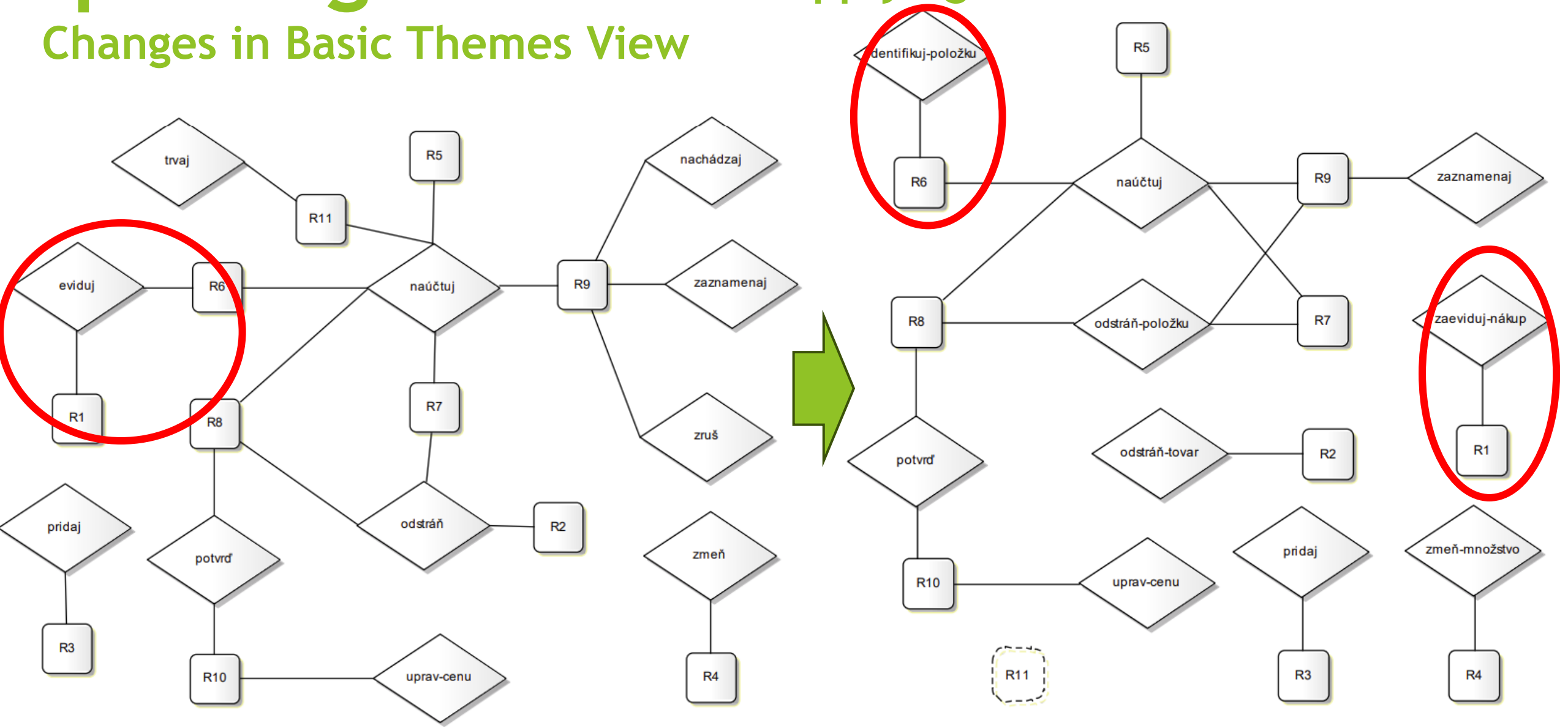
Identifying Themes - Applying the Changes in Basic Themes View



Source from: Bc. Pavol Michalco: PRÍPADY POUŽITIA A _TÉMY V PRÍSTUPE THEME/DOC

*we do not want to treat R11 requirement, left for next development phases

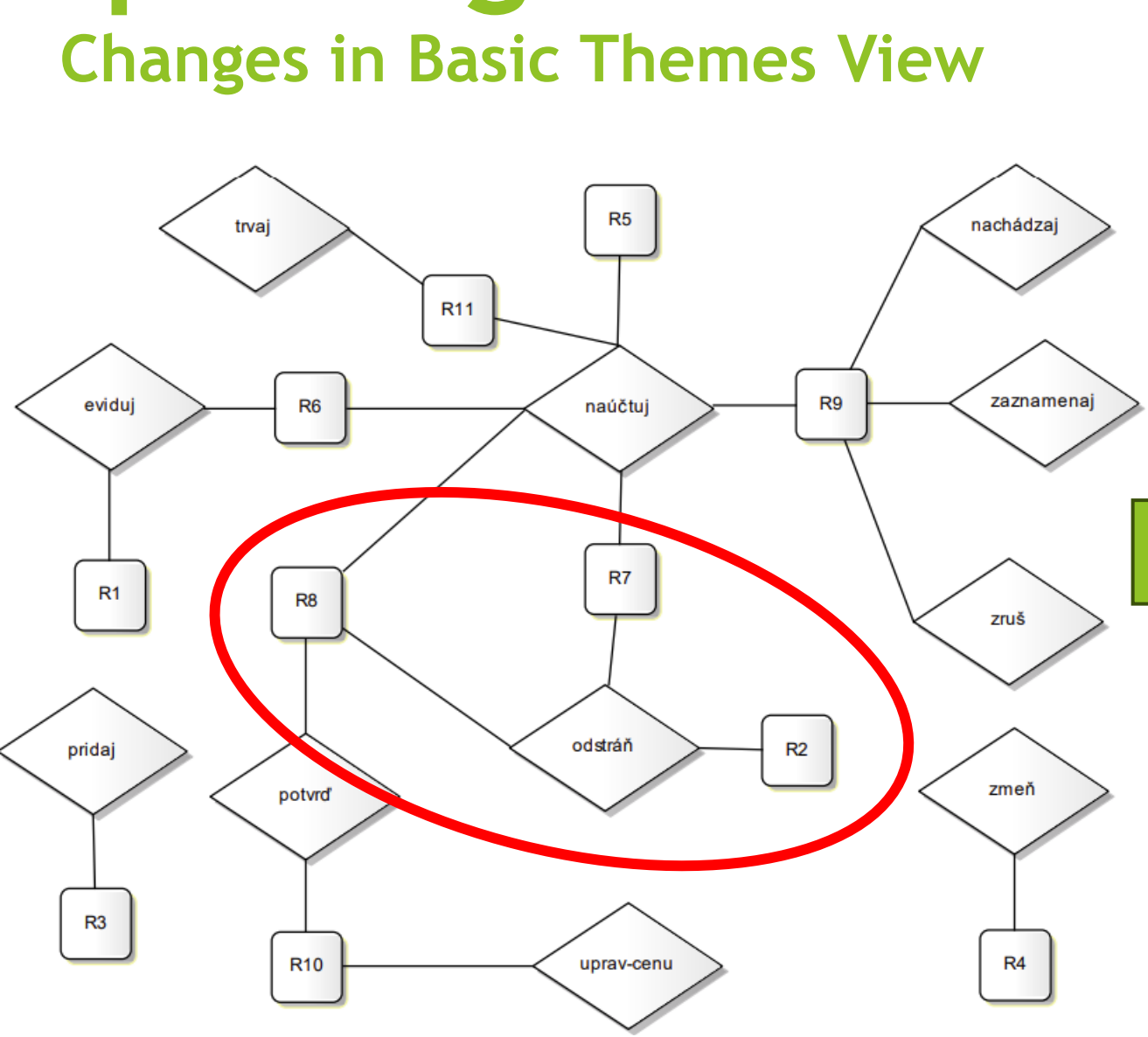
Splitting themes - Applying the Changes in Basic Themes View

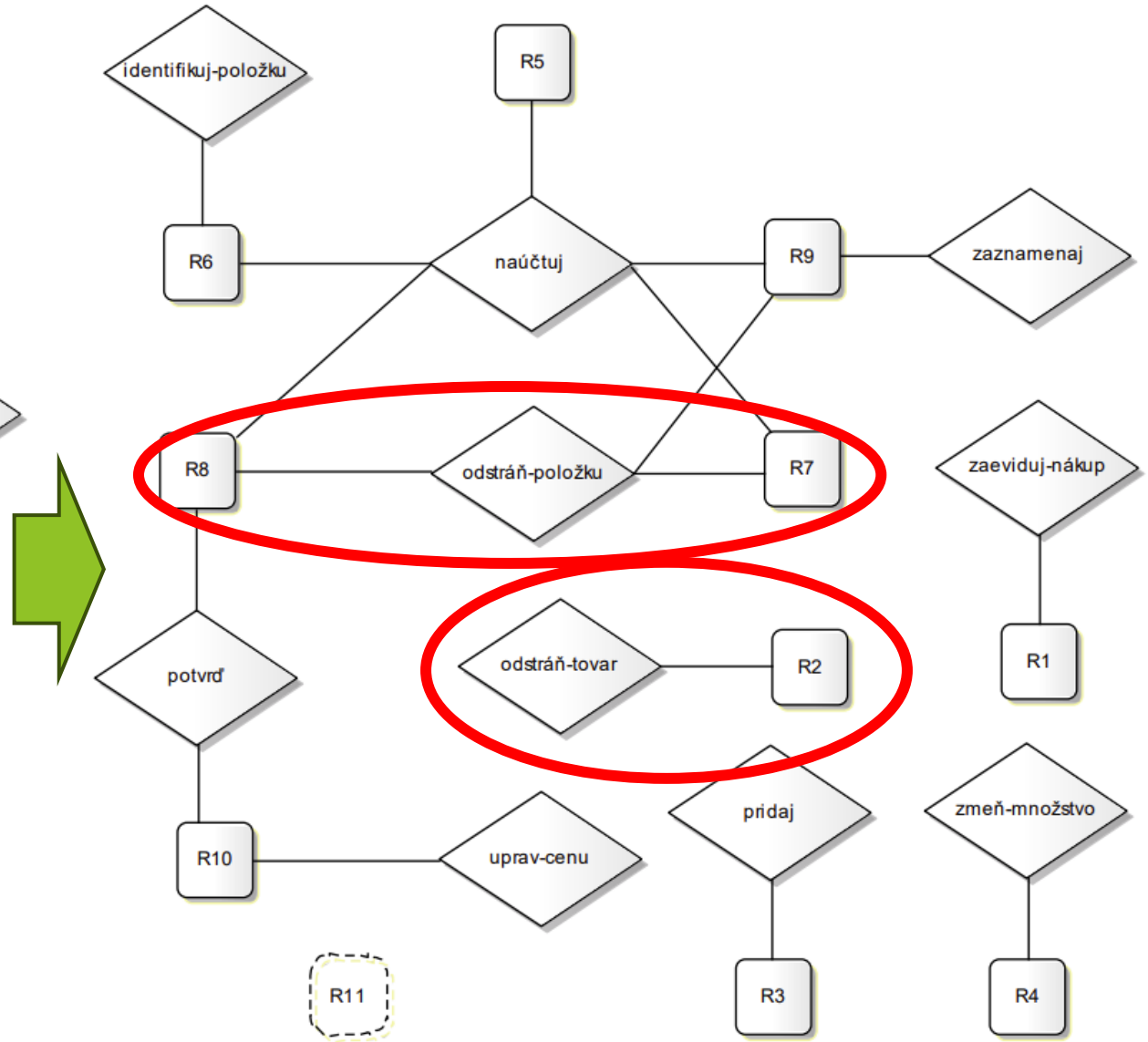


Source from: Bc. Pavol Michalco: PRÍPADY POUŽITIA A _TÉMY V PRÍSTUPE THEME/DOC

*we do not want to treat R11 requirement, left for next development phases

Splitting Themes - Applying the Changes in Basic Themes View

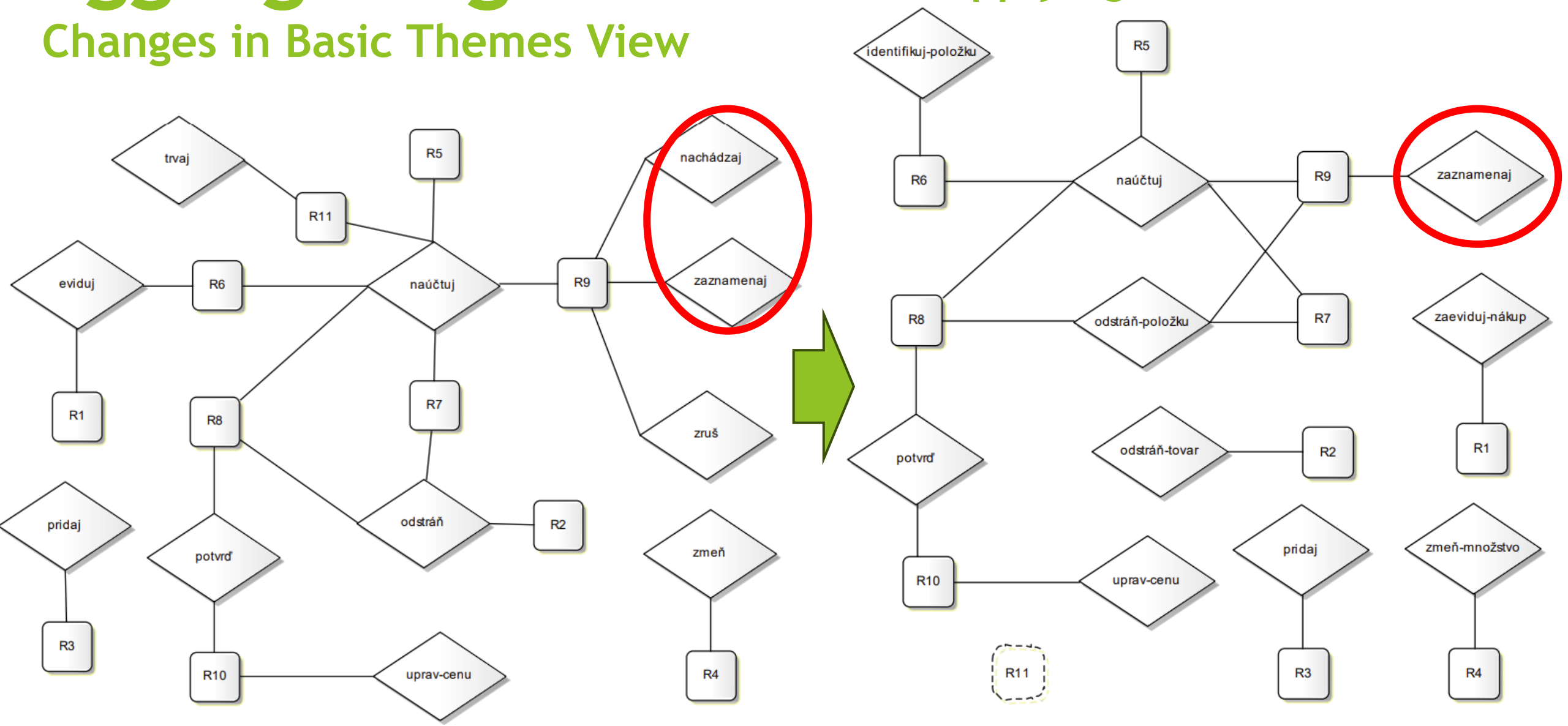




Source from: Bc. Pavol Michalco: PRÍPADY POUŽITIA A _TÉMY V PRÍSTUPE THEME/DOC

*we do not want to treat R11 requirement, left for next development phases

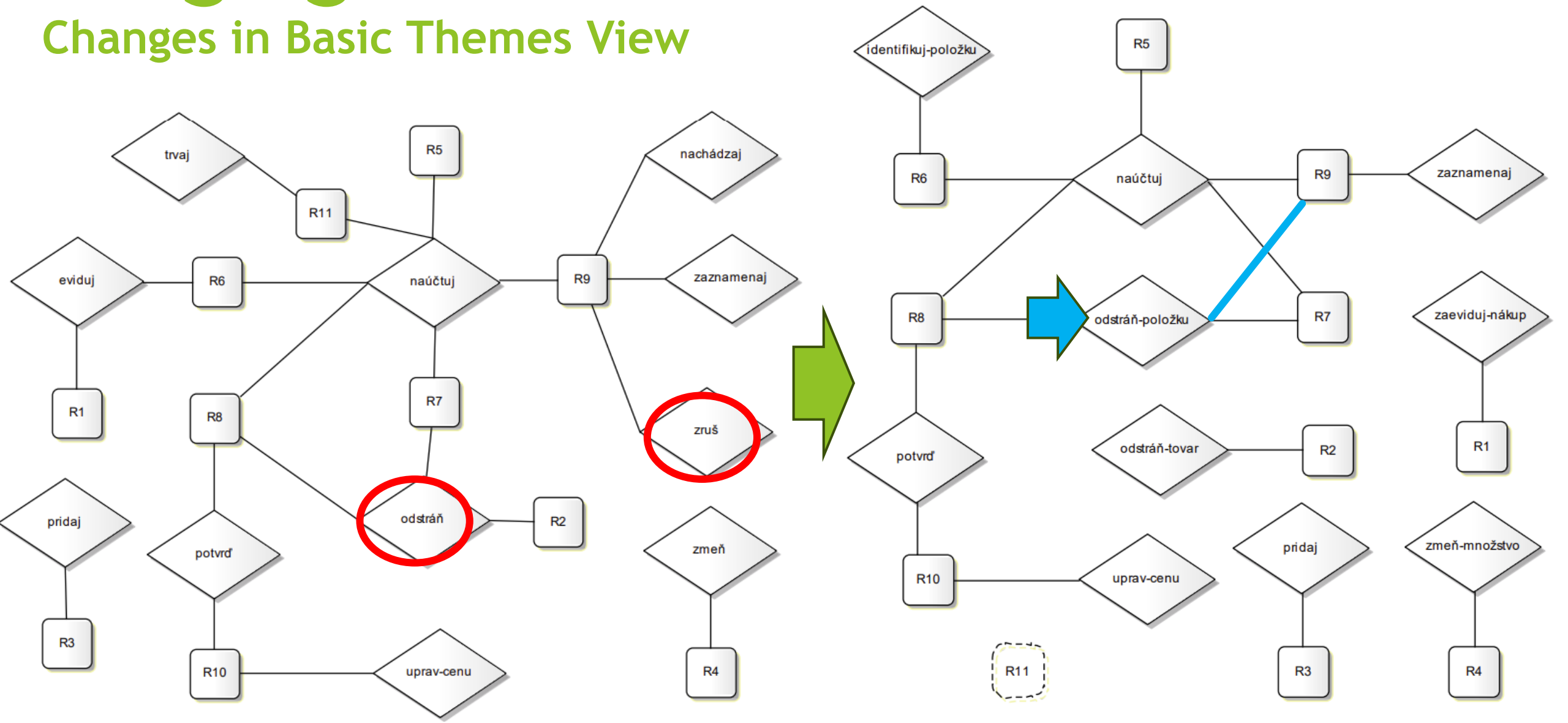
Aggregating Themes - Applying the Changes in Basic Themes View



Source from: Bc. Pavol Michalco: PRÍPADY POUŽITIA A _TÉMY V PRÍSTUPE THEME/DOC

*we do not want to treat R11 requirement, left for next development phases

Merging Similar Themes - Applying the Changes in Basic Themes View

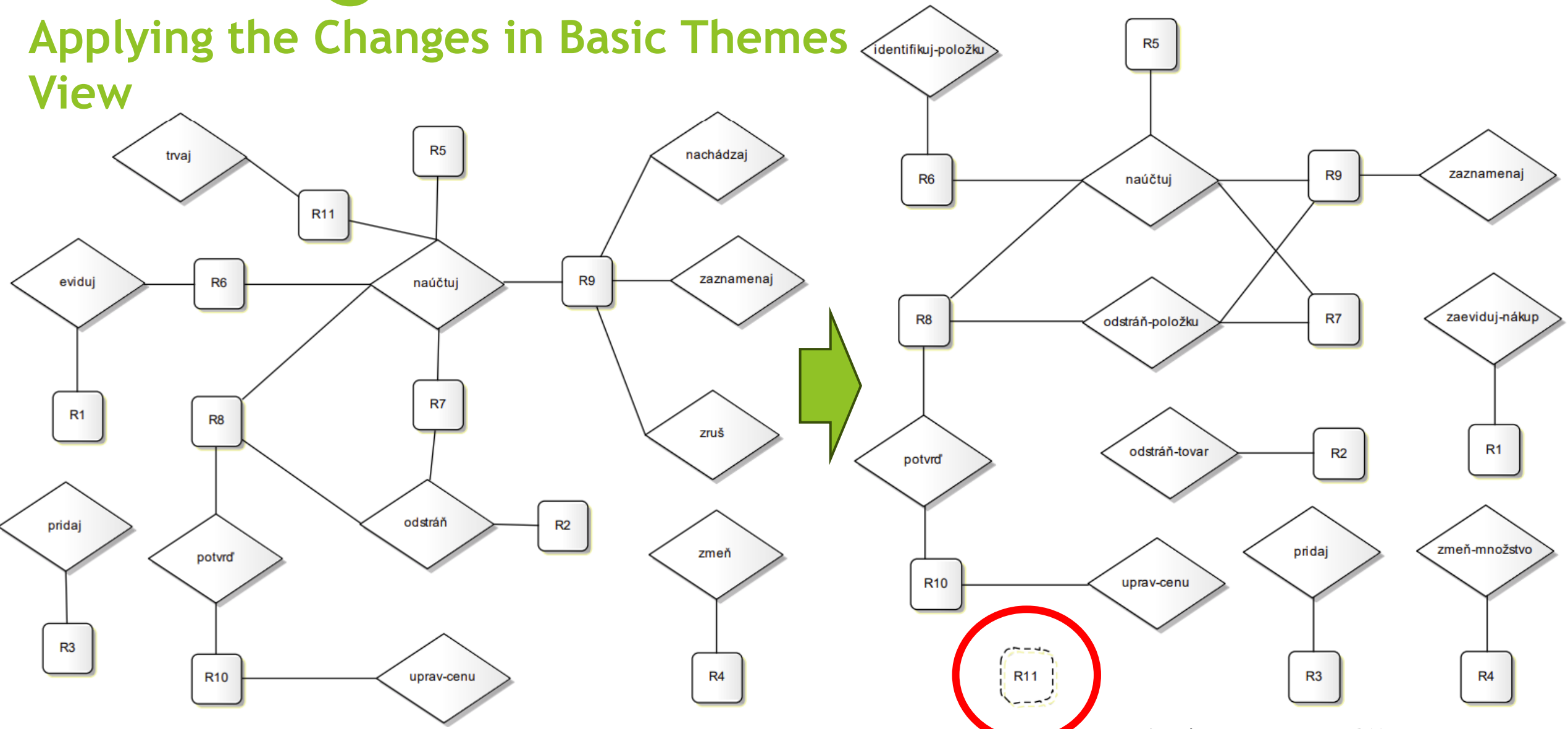


Source from: Bc. Pavol Michalco: PRÍPADY POUŽITIA A _TÉMY V PRÍSTUPE THEME/DOC

*we do not want to treat R11 requirement, left for next development phases

Deleting Irrelevant Themes -

Applying the Changes in Basic Themes View



Source from: Bc. Pavol Michalco: PRÍPADY POUŽITIA A _TÉMY V PRÍSTUPE THEME/DOC

*we do not want to treat R11 requirement, left for next development phases

Observing Responsibility of Themes

- Identification of crosscutting themes
- The content of one functionality which description crosscuts the description of another functionality in particular requirement

Aspect is a theme which description in requirement specification crosscuts another theme

Crosscutting place: requirements connected with more than one theme.

The need to find dominant theme for particular requirement

-this theme is **aspect theme** ***in coverage of the requirement amongst all themes***

Highlighting of Croscutting behaviour: with oriented connection.

Identification of Crosscutting Themes

Siobhàn Clarke and Elisa Baniassad. 2005. Aspect-Oriented Analysis and Design. Addison-Wesley Professional.

All of following rules must hold:

Cannot decompose requirement to isolate its relation with other themes

- **1. Cannot decompose particular requirement** to isolate themes and prevent sharing
- **2. Candidate theme is selected according to the dominance in coverage of the requirement amongst all themes (because it crosscuts them)**
  - *Checking of requirements if particular theme has information about its behaviour*
  - *(for example authorization: particular business functionality should not have information if rights are verified)*

  From DOMINANT theme we create directed connections to other
- **3. Base theme activates aspect**
  - the behaviour of aspect is activated with behaviour in base theme
- **4. Activated and dominant theme is externally activated in multiple situations**
  - The need to have multiple situations to beneficially apply aspects

Examples of Basic Diagrams of Themes

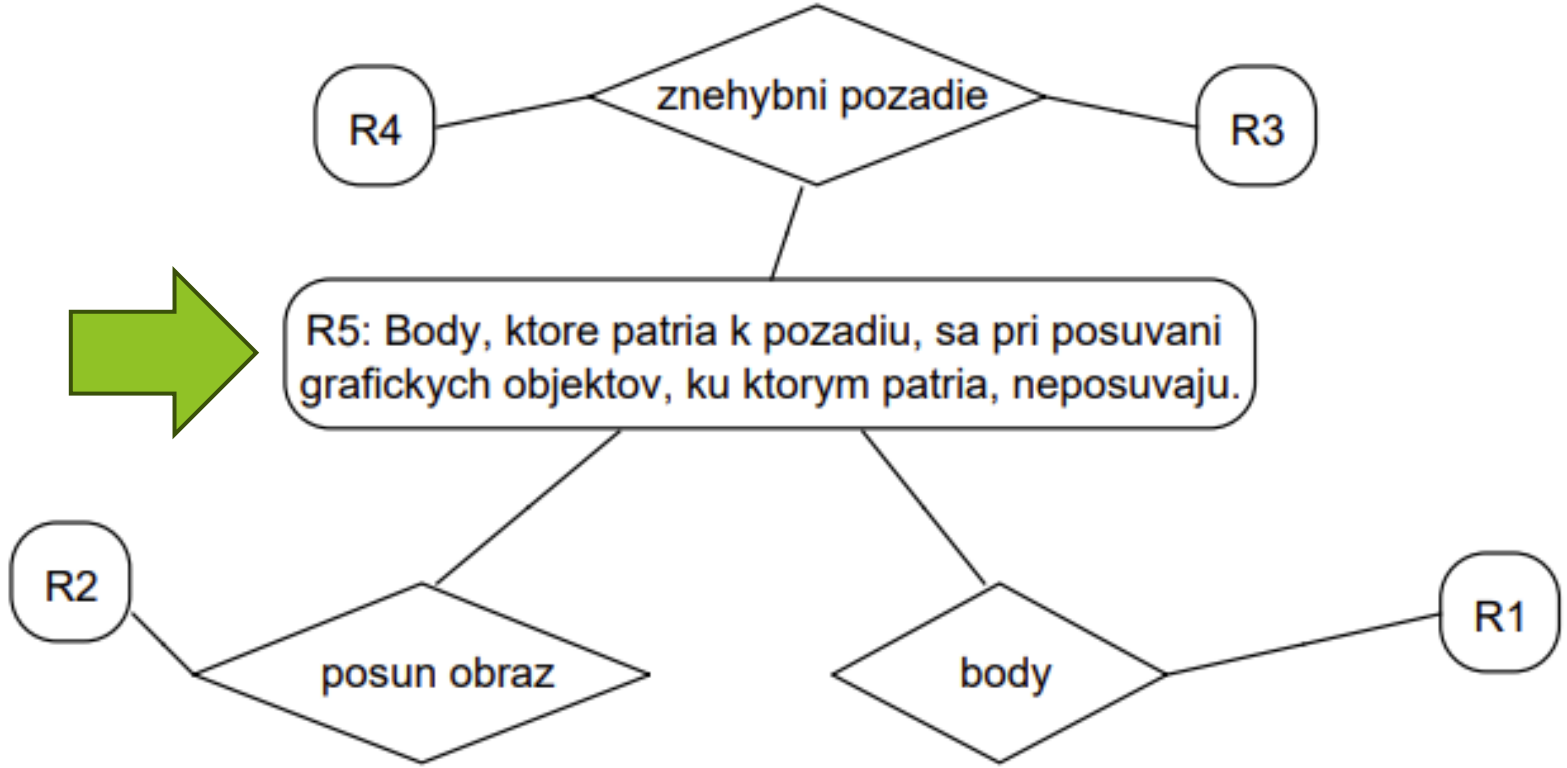
- R1: Pre body sa evidujú ich súradnice a to, či patria k pozadiu.
- R2: Obraz musí byť možné posúvať, pričom sa posunú všetky objekty, z ktorých pozostáva.

...

- R5: Body, ktoré patria k pozadiu, sa pri posúvaní grafických objektov, ku ktorým patria, neposúvajú.

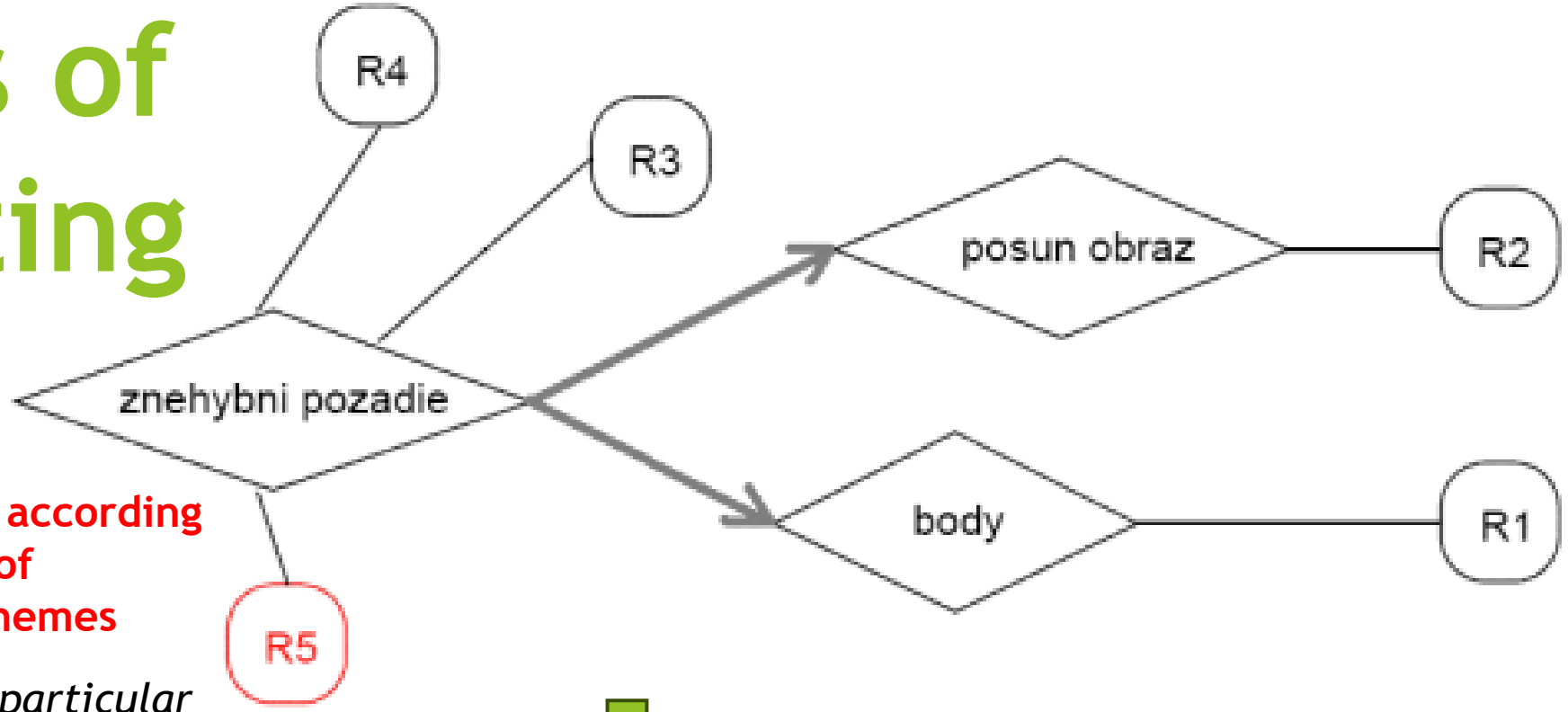
1. Cannot decompose particular requirement to isolate themes and prevent sharing

R5 Cannot be decomposed



Source: Aspekty v analýze a návrhu – prístupy Theme a JPDD Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoja softvéru, Valentino Vranić, 2017

Examples of Crosscutting View



2. Candidate theme is selected according to the dominance in coverage of the requirement amongst all themes

Checking of requirements if particular theme has information about its behaviour

- R1: Pr[illegible]body sa evidujú ich súradnice a to, či patria k pozadiu.
- R2: Obraz musí byť možné posúvať, pričom sa posunú všetky objekty, z ktorých pozostáva.

-------- Information about behaviour ------

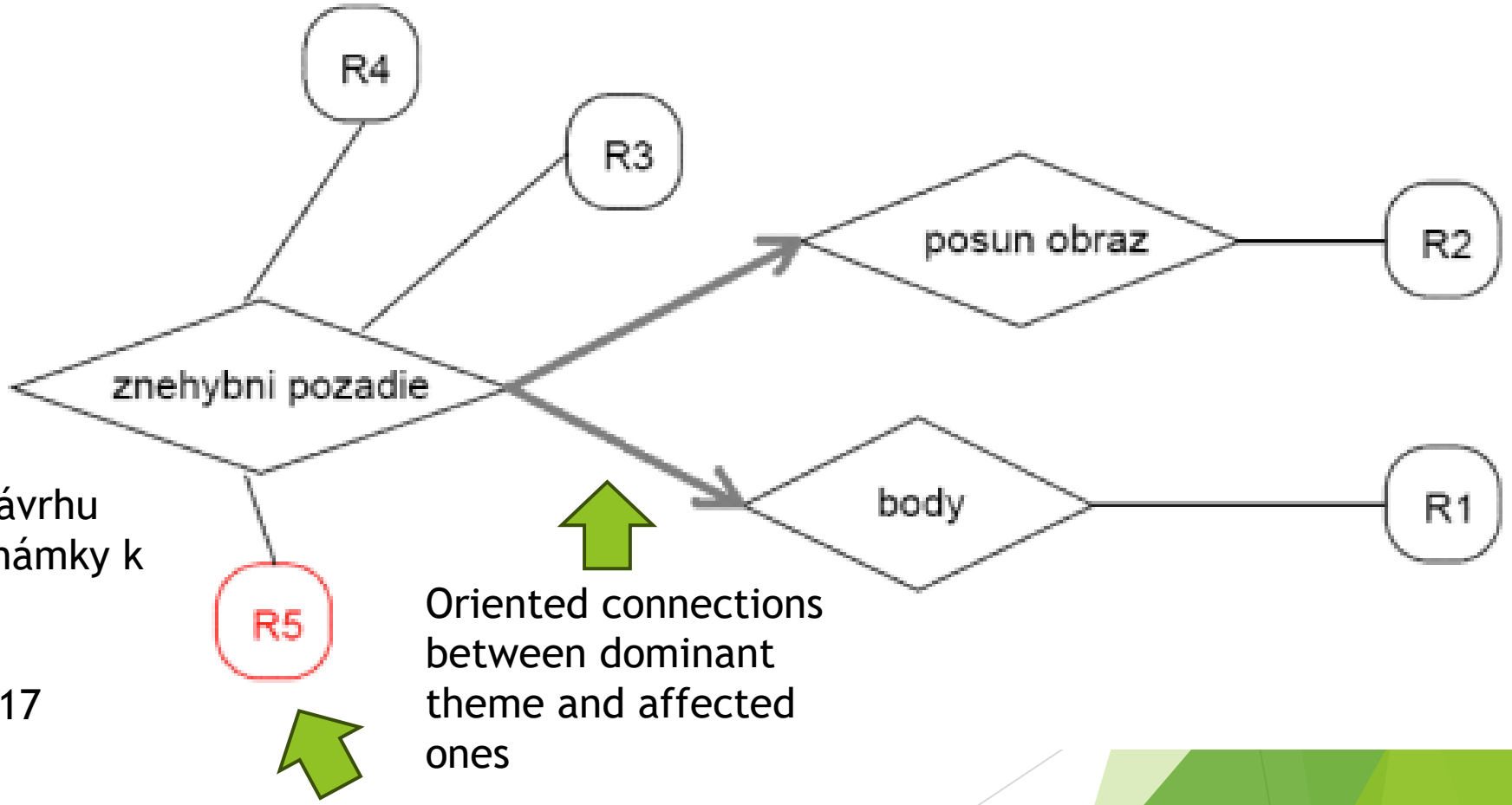
- R5: Body, ktoré patria k pozadiu, sa pri posúvaní grafických objektov, ku ktorým patria, neposúvajú.

4. Activated and dominant theme is externally activated in multiple situations

Source: Aspekty v analýze a návrhu – prístupy Theme a JPDD Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoja softvéru, Valentino Vranić, 2017

3. Base theme activates aspect

Examples of Crosscutting View



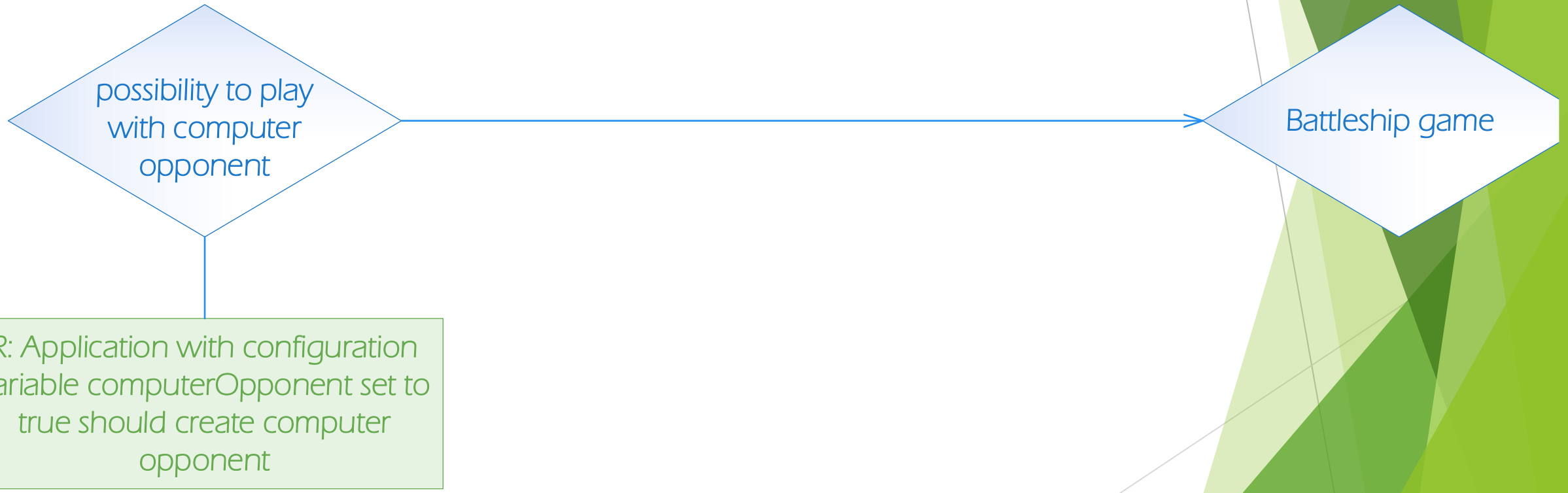
Source: Aspekty v analýze a návrhu – prístupy Theme a JPDD Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoja softvéru, Valentino Vranić, 2017

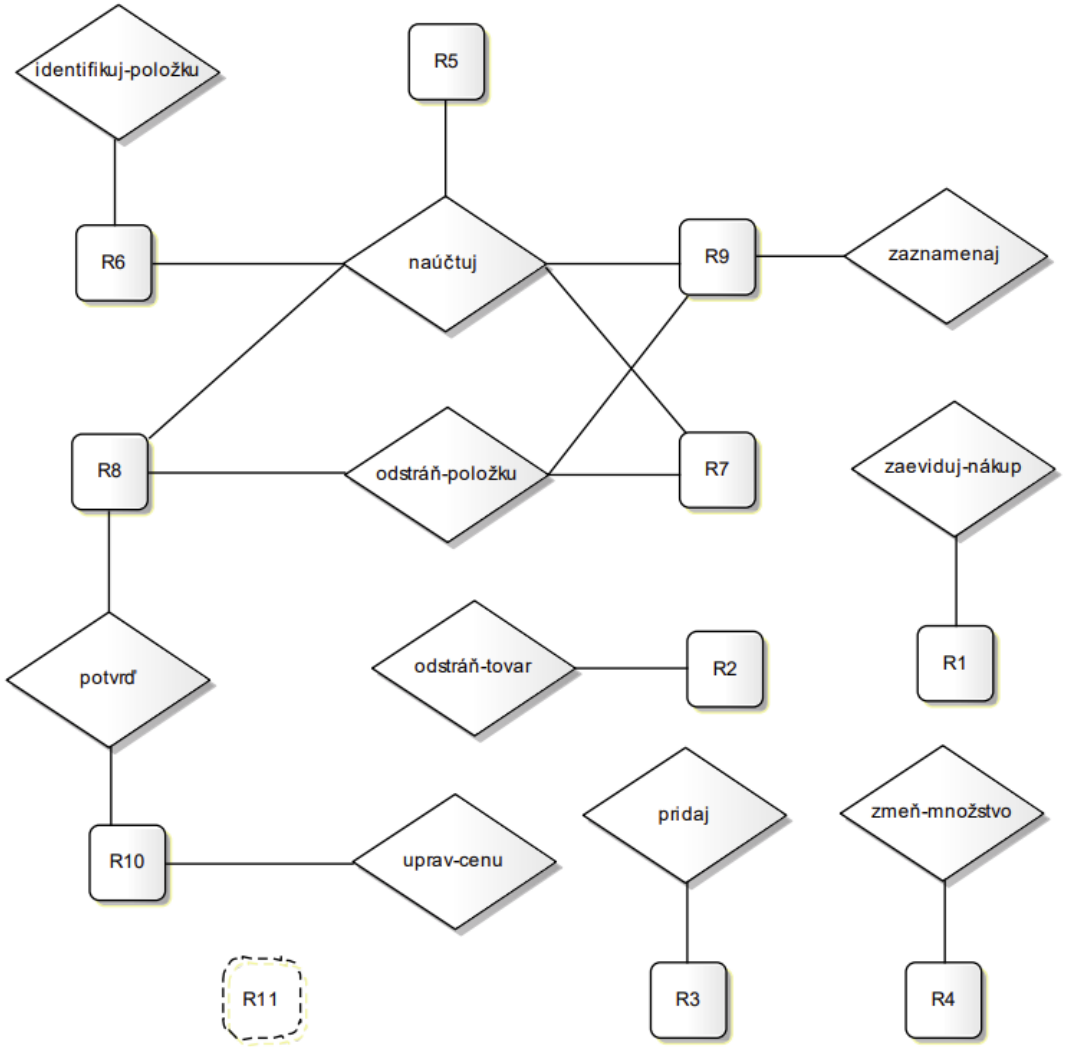
Oriented connections between dominant theme and affected ones

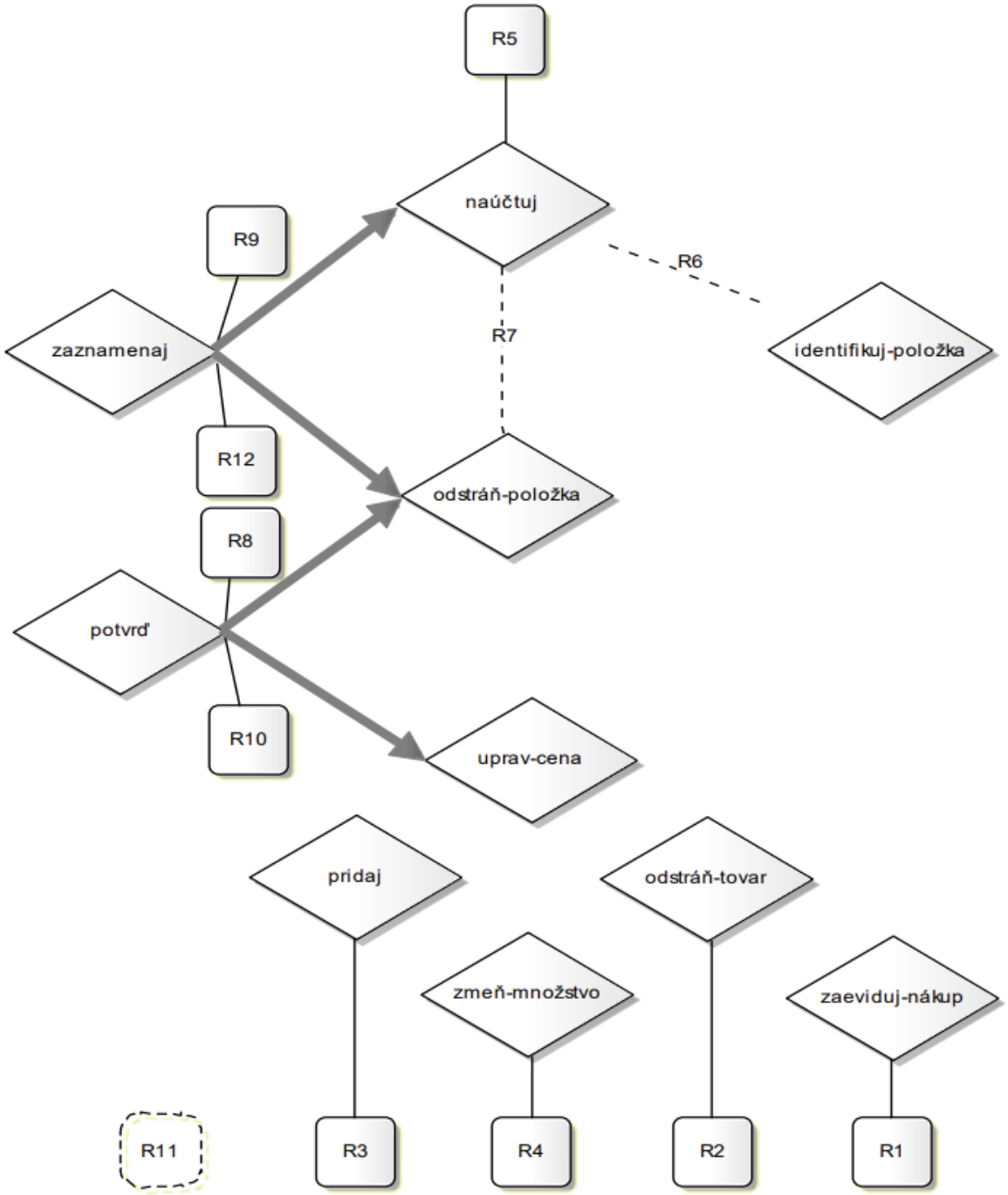
The requirement that cannot be split/ remained shared

[All such requirement associated with more then one theme are identified]

Examples of crosscutting view



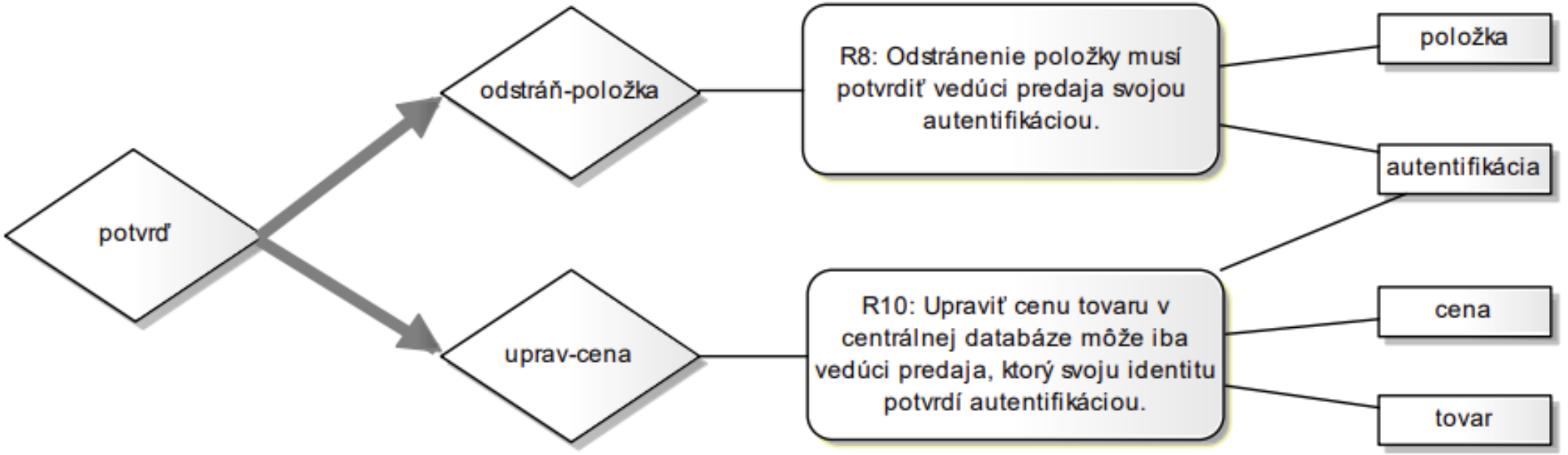




Source from: [Bc. Pavol Michalco: PRÍPADY POUŽITIA A TÉMY V PRÍSTUPE THEME/DOC]

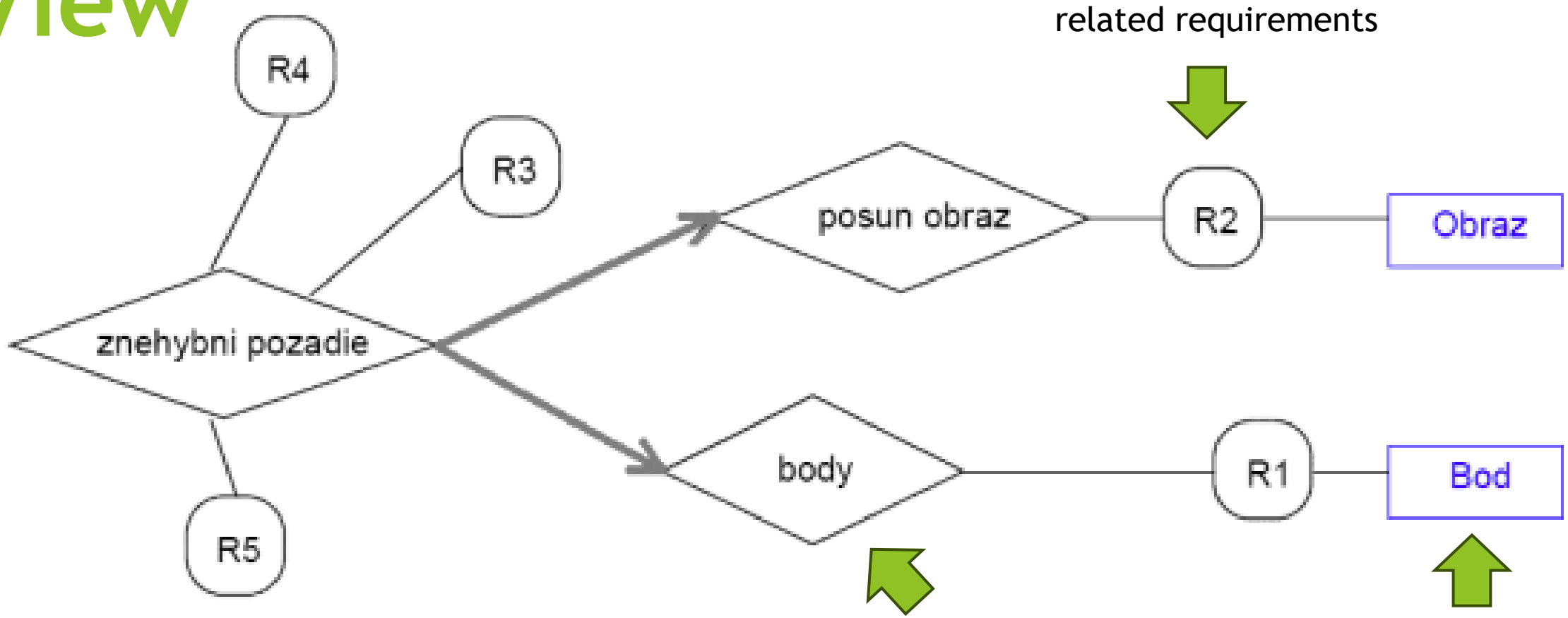
Individual View

- Created in the end of analysis as preparation for design
- Overview of crosscutting themes for exactly one particular theme
- Contains:
  - related requirements
  - identified entities
  - Individual views on related themes (in case of crosscutting view)



Source from: Bc. Pavol Michalco: PRÍPADY POUŽITIA A TÉMY V PRÍSTUPE THEME/DOC

Example of Individual View



Source: Aspekty v analýze a návrhu – prístupy Theme a JPDD Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoja softvéru, Valentino Vranić, 2017

Design

Themes in Design

Theme/Uml

- As extension to Uml, similarly used primarily in design
- For symmetric/asymmetric orientation
- Programming language independent
- Crosscutting views are captured via parametrized packages (temple packages)

Principles

Theme

- Each theme models one **view on system**
- Expressed with packages

Packages

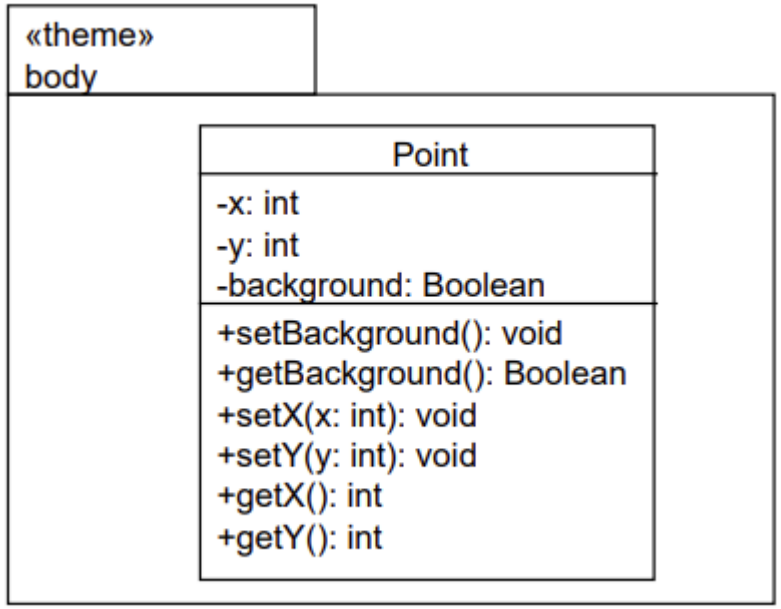
- Specifies:
  - **The behaviour (UML sequence diagram)**
  - **The structure (UML class diagram)**
- Contains multiple views

Kind of themes

Basic ***Crosscutting***

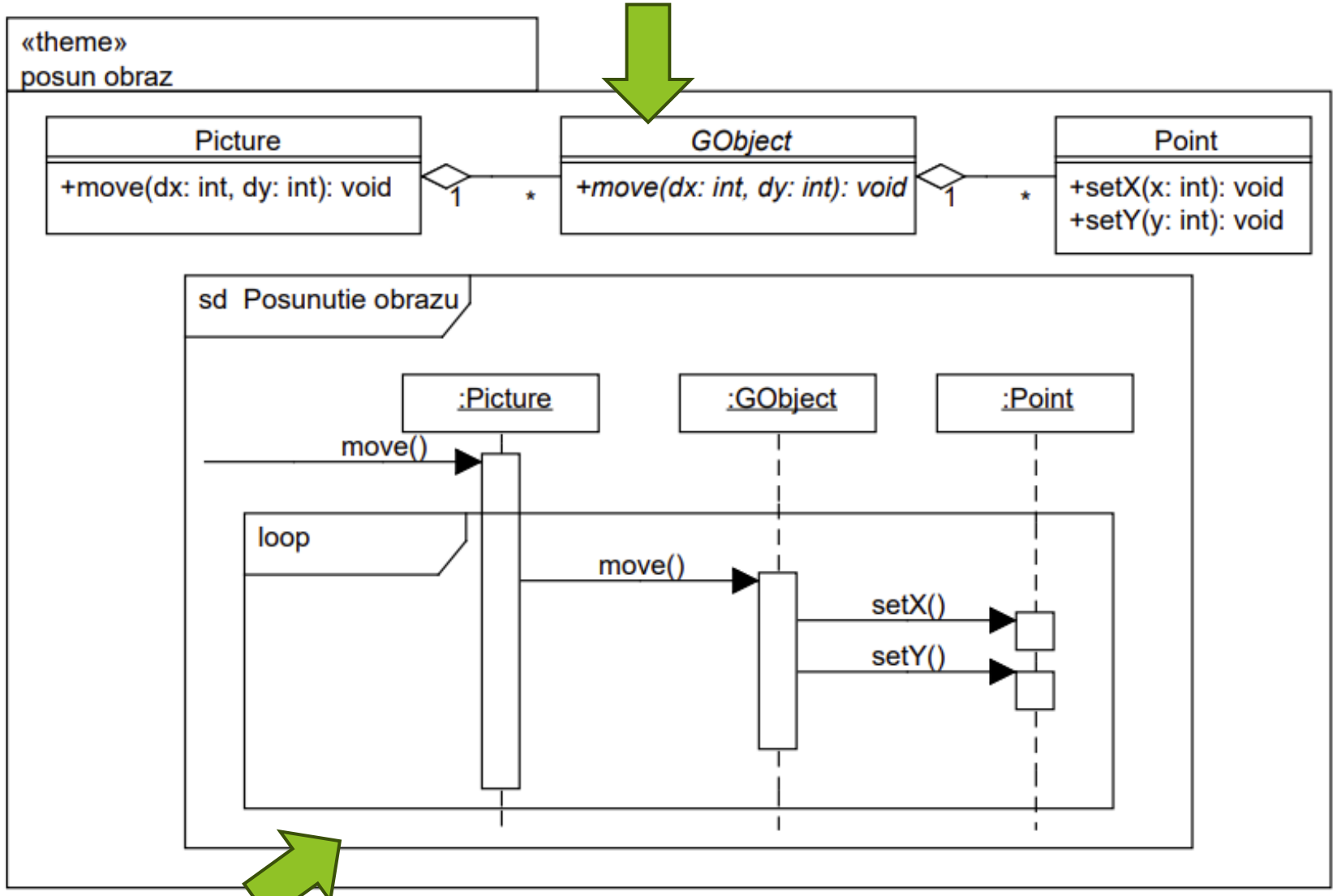
Composition of themes

Examples of basic view



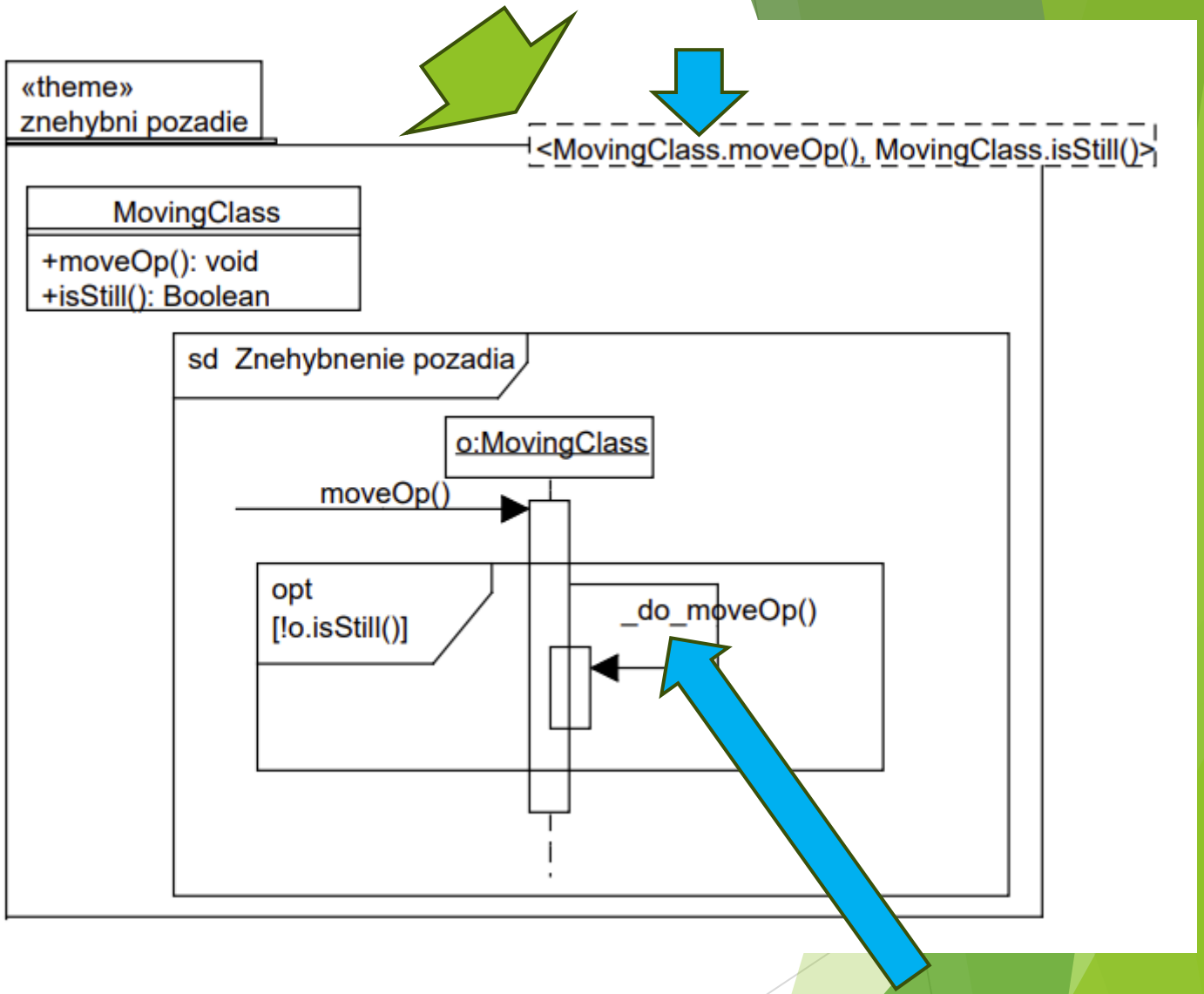
Source: Aspekty v analýze a návrhu – prístupy Theme a JPDD Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoja softvéru, Valentino Vranić, 2017

Static part - structure



Dynamic part - behaviour

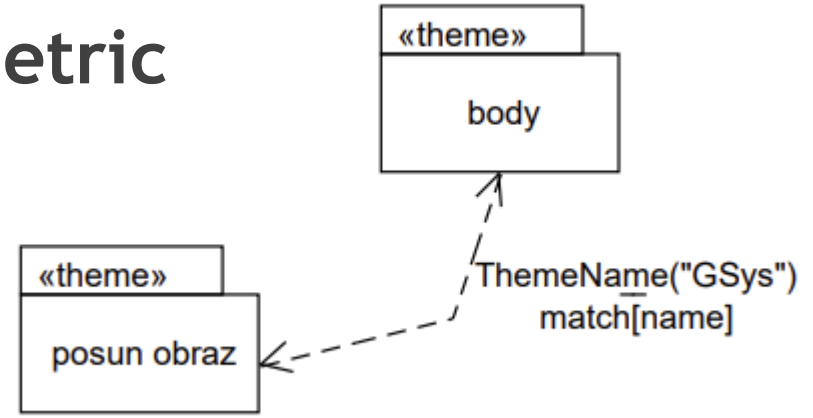
Parametrized Packages



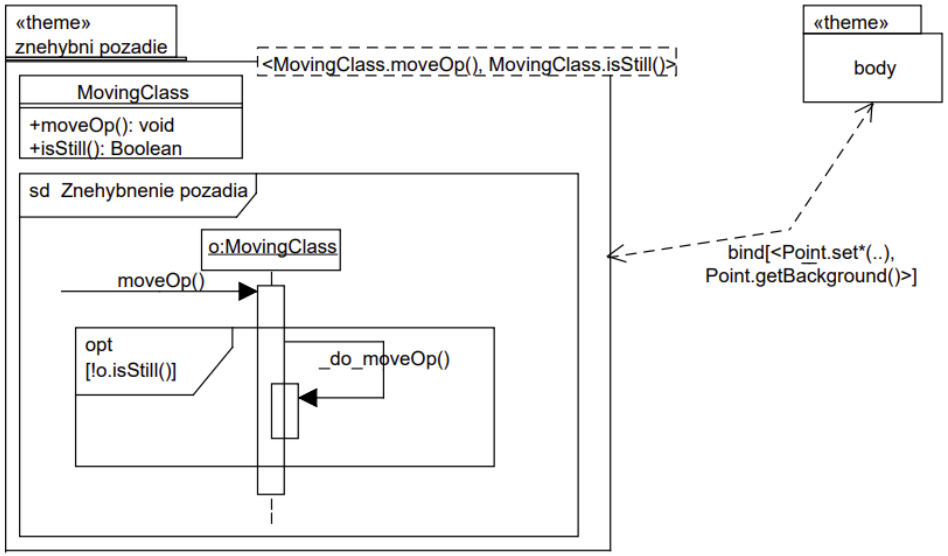
Source: Aspekty v analýze a návrhu – prístupy Theme a JPDD Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoja softvéru, Valentino Vranić, 2017

Examples of Composition of Themes

Symmetric

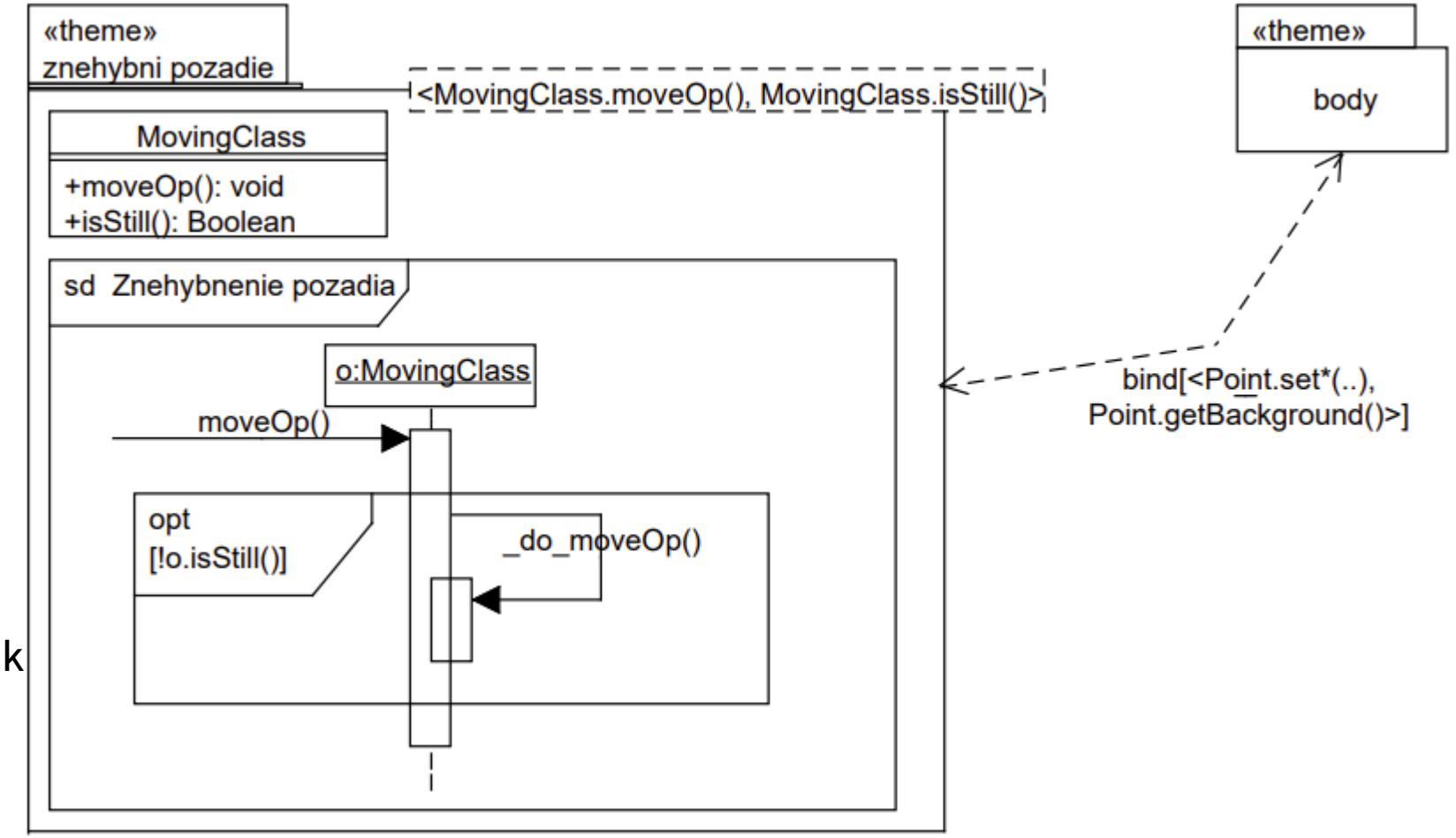


Asymmetric



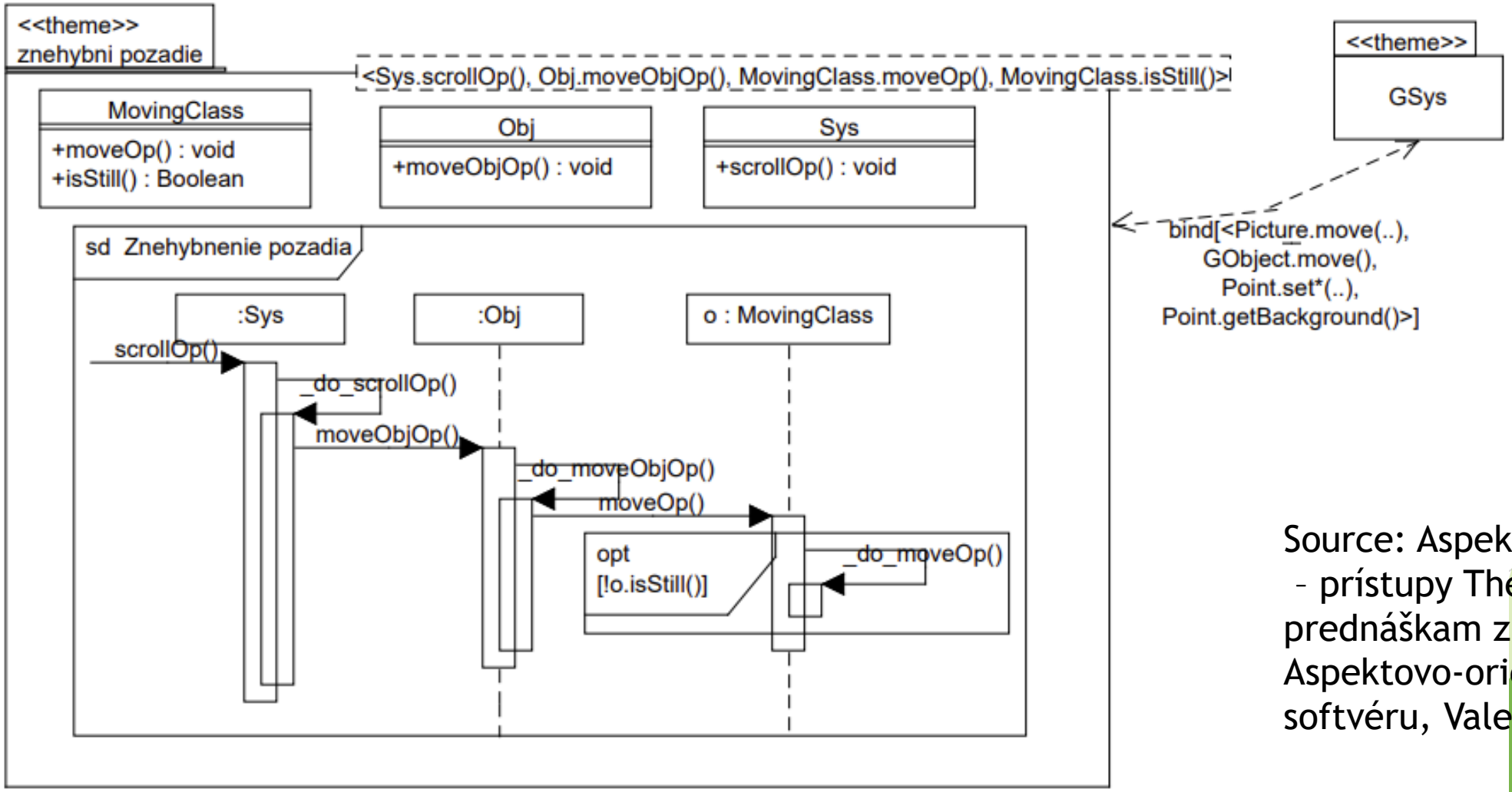
Source: Aspekty v analýze a návrhu – prístupy Theme a JPDD Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoja softvéru, Valentino Vranić, 2017

Examples of Composition of Crosscutting Themes



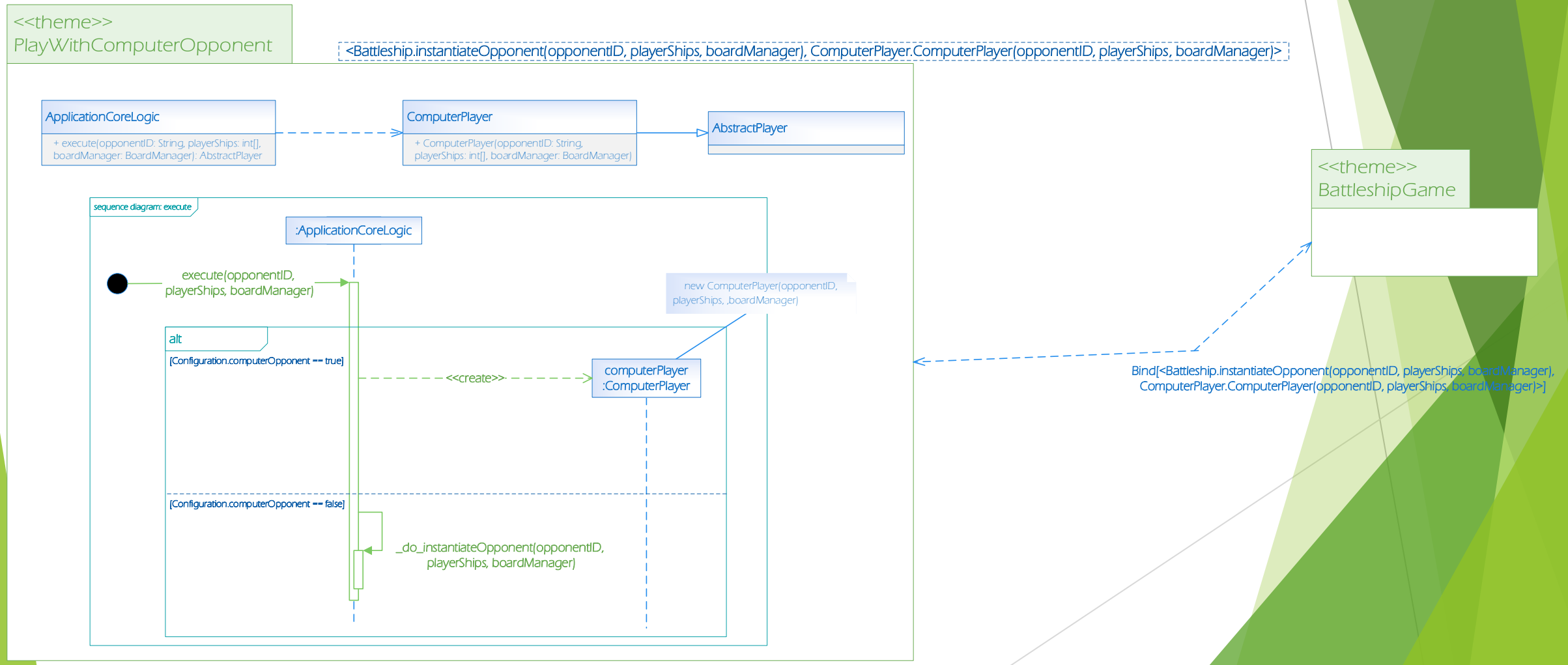
Source: Aspekty v analýze a návrhu – prístupy Theme a JPDD Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoja softvéru, Valentino Vranić, 2017

Examples of Composition of Crosscutting Themes

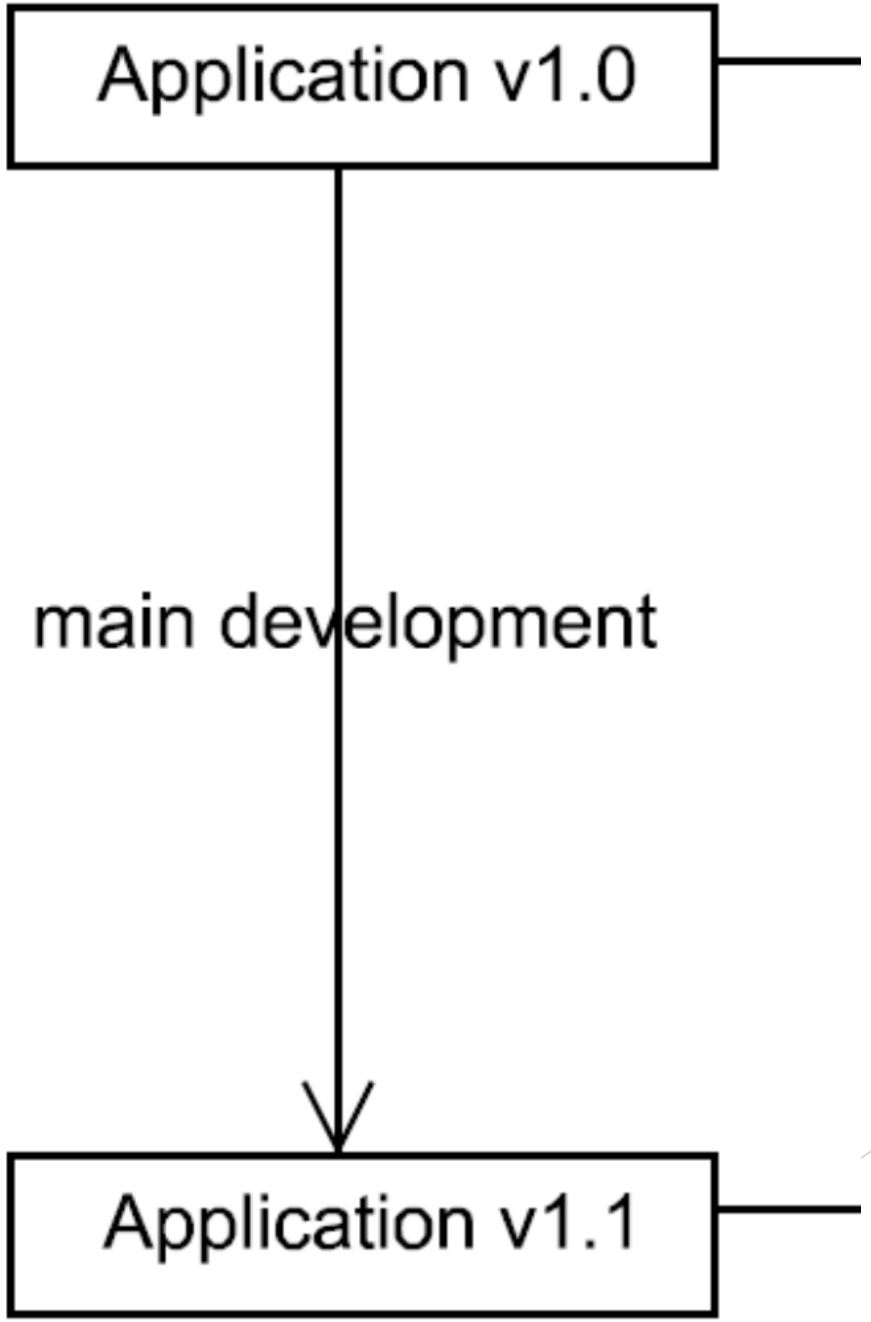


Source: Aspekty v analýze a návrhu – prístupy Theme a JPDD Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoja softvéru, Valentino Vranić, 2017

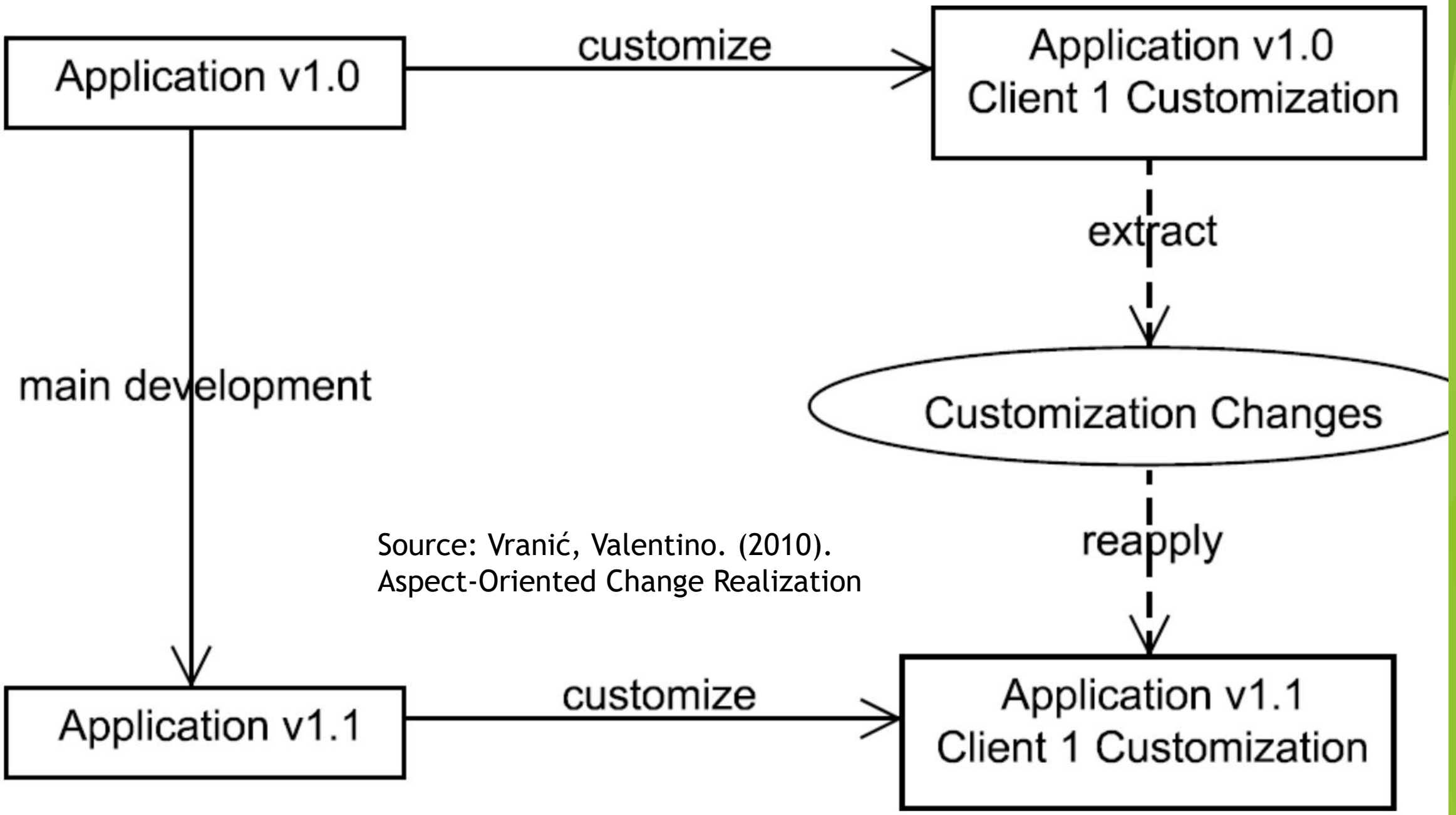
Examples of Composition of Crosscutting Themes



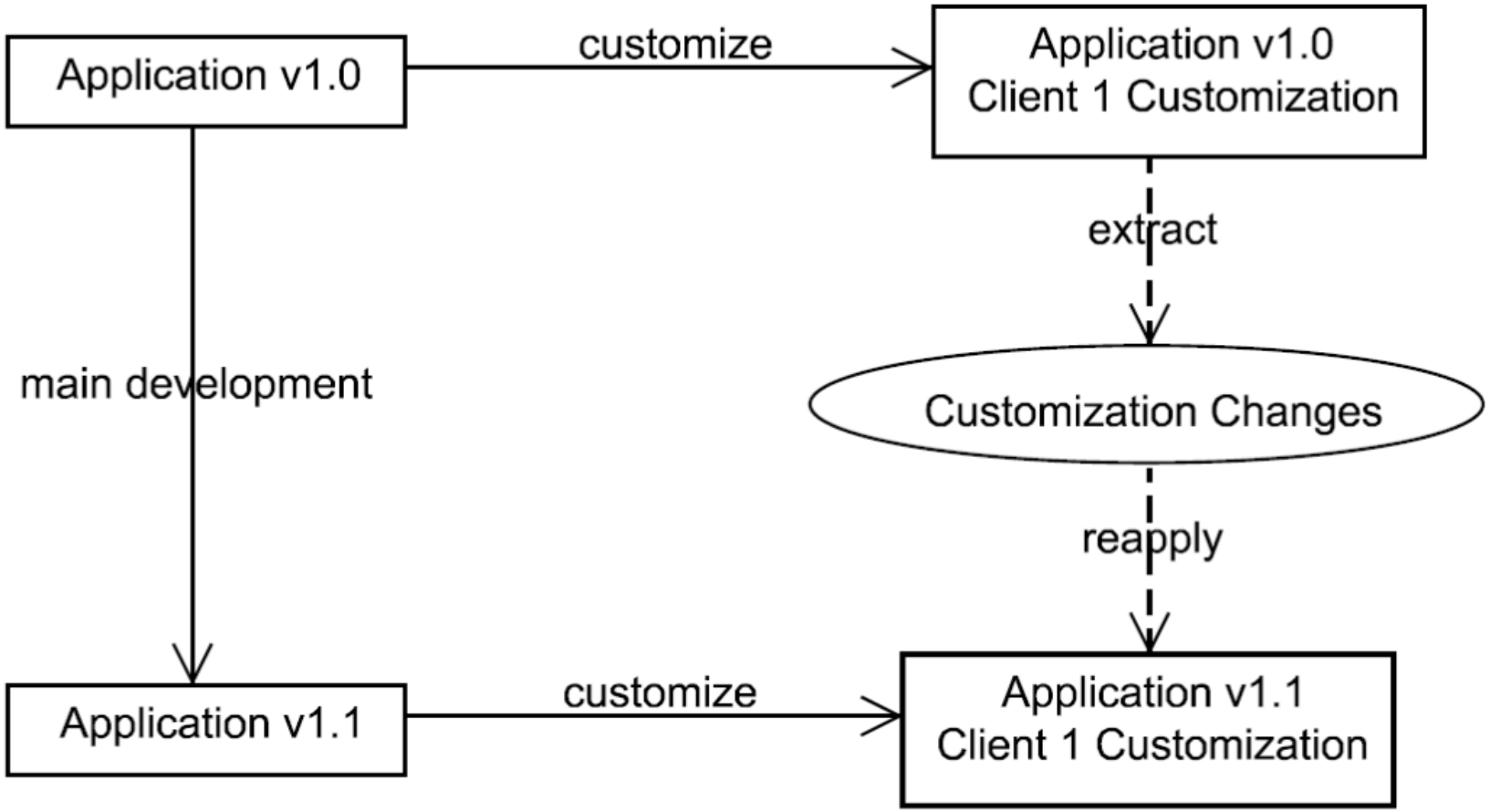
Aspect Oriented Change Realization



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

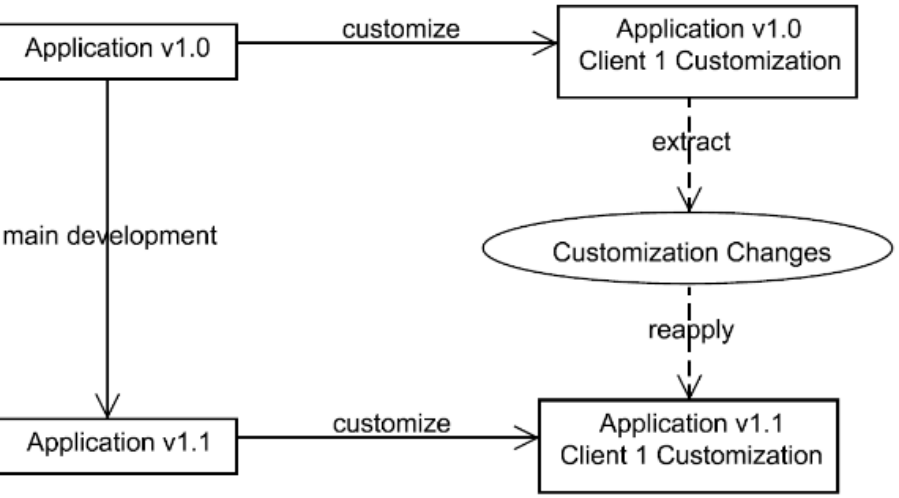


Source: Vranić, Valentino. (2010). Aspect-Oriented Change Realization



Aspects

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



Aspects

Aspect-oriented change realization

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

Examples From Tests

Test 2019 /2020

Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmlkd4p2texXMqXPrKjR15ljkIAmMU5

Systém pozostáva z detekčných zariadení reprezentovaných triedou `Detector` a regulačných zariadení reprezentovaných triedou `Regulator`. Každé detekčné zariadenie má svoje číslo uložené v atribúte `n`. Ak identifikuje určitú udalosť, detekčné zariadenie pošle signál jednému z regulačných zariadení zavolaním metódy `act(int signal)` tohto regulačného zariadenia vo svojej metódy `regulate()`, kde `signal` je identifikačné číslo signálu. Ak sa ho daný signál týka, regulačné zariadenie vykoná príslušnú akciu. Inak pošle signál všetkým regulačným zariadeniam, s ktorými je spojené, zavolaním ich metód `act(int id)`.

Občas sa niektoré detekčné zariadenie pokazí. Zabezpečte, aby sa bezo zmien v triedach `Detector` a `Regulator` signály takého detekčného zariadenia ignorovali. Pre jednoduchosť, môžete pokladať, že je číslo pokazeného zariadenia dané ako konšanta.

Prvky tried `Detector` a `Regulator` významné z hľadiska riešenia úlohy sú sumarizované v nasledujúcom kóde:

```
public class Detector {
    public int n;

    public regulate() {
        ...
    }

    ...
}

public class Regulator {
    public void act(int signal) {
        ...
    }

    ...
}
```

1. (2 b) Uveďte príslušný analytický model v notácii Theme/Doc v pohľade tém a vzťahov (základný pohľad). Transformujte tento pohľad do pohľadu pretínajúcich tém.

2. (3 b) Uveďte príslušný návrhový model v notácii Theme/UML. Môžete využiť jej rozšírenie o nepriame vzťahy z notácie JPDD.

3. (5 b) Uveďte kód výsledného aspektu v jazyku AspectJ. Uplatnite pritom najvhodnejší z týchto aspektovo-orientovaných návrhových vzorov: Worker Object Creation, Wormhole a Cuckoo's Egg. Vysvetlite, ako je vzor uplatnený.

Systém pozostáva z detekčných zariadení reprezentovaných triedou `Detector` a regulačných zariadení reprezentovaných triedou `Regulator`. Každé detekčné zariadenie má svoje číslo uložené v atribúte `n`. Ak identifikuje určitú udalosť, detekčné zariadenie pošle signál jednému z regulačných zariadení zavolaním metódy `act(int signal)` tohto regulačného zariadenia vo svojej metódy `regulate()`, kde `signal` je identifikačné číslo signálu. Ak sa ho daný signál týka, regulačné zariadenie vykoná príslušnú akciu. Inak pošle signál všetkým regulačným zariadeniam, s ktorými je spojené, zavolaním ich metód `act(int id)`.

Občas sa niektoré detekčné zariadenie pokazí. Zabezpečte, aby sa bezo zmien v triedach `Detector` a `Regulator` signály takého detekčného zariadenia ignorovali. Pre jednoduchosť, môžete pokladať, že je číslo pokazeného zariadenia dané ako konšanta.

Prvky tried `Detector` a `Regulator` významné z hľadiska riešenia úlohy sú sumarizované v nasledujúcom kóde:

```
public class Detector {
    public int n;

    public regulate() {
        ...
    }

    ...
}

public class Regulator {
    public void act(int signal) {
        ...
    }

    ...
}
```

Requirement

Theme 1

Theme 2 crosscutting

Test 2019 /2020

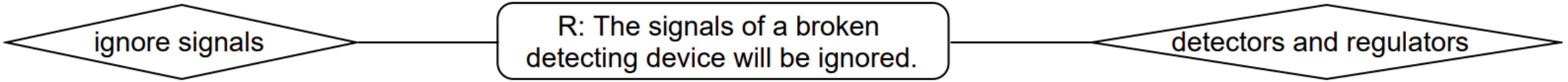
Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmlkd4p2texXMqXPrKjR15ljkIAmMU5

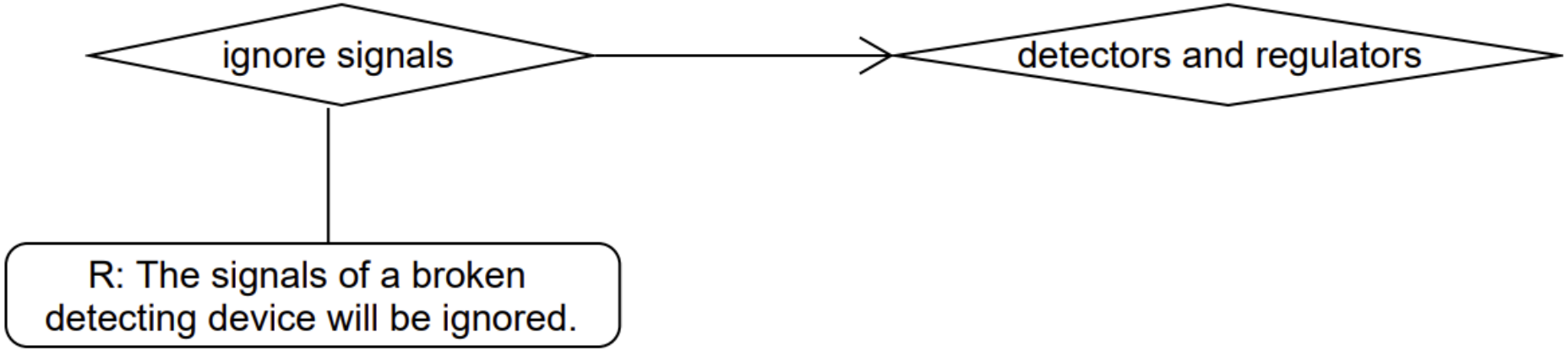
1. (2 b) Uveďte príslušný analytický model v notácii Theme/Doc v pohľade tém a vzťahov (základný pohľad). Transformujte tento pohľad do pohľadu pretínajúcich tém.

2. (3 b) Uveďte príslušný návrhový model v notácii Theme/UML. Môžete využiť jej rozšírenie o nepriame vzťahy z notácie JPDD.

3. (5 b) Uveďte kód výsledného aspektu v jazyku AspectJ. Uplatnite pritom najvhodnejší z týchto aspektovo-orientovaných návrhových vzorov: Worker Object Creation, Wormhole a Cuckoo's Egg. Vysvetlite, ako je vzor uplatnený.

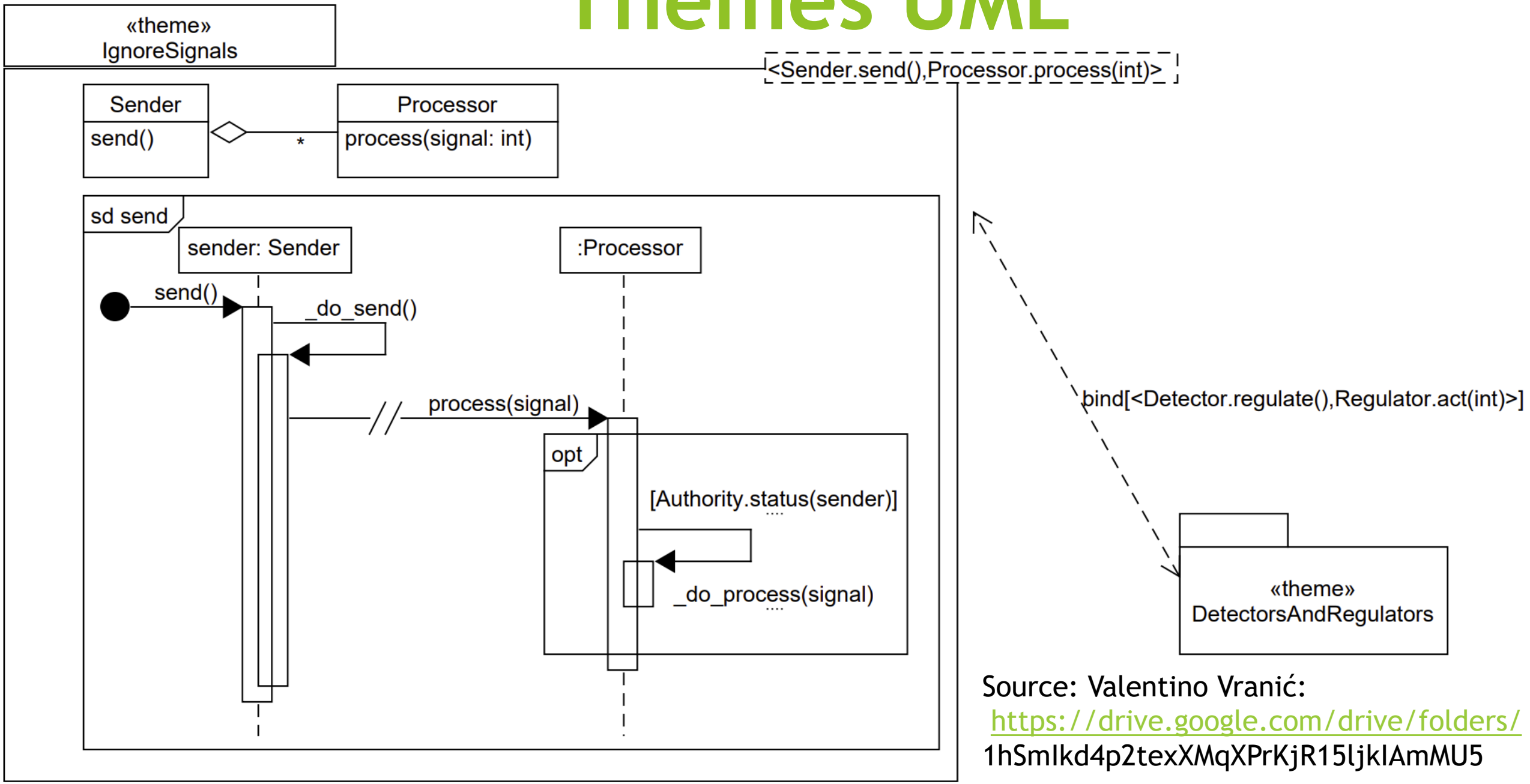
Themes DOC





Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljkIAmMU5

Themes UML



Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljkIAmMU5

Code

```
public class Detector {
    public int n;
    public regulate() {
        …
    }
    …
}

public class Regulator {
    public void act(int signal) {
        …
    }
    …
}
```

```
public aspect IgnoreSignals {

    pointcut regulations(Detector detector):
        execution(void Detector.regulate()) && this(detector); // callee space

    pointcut activations(Regulator regulator, int signal):
        call(void Regulator.act(int)) && this(regulator) && args(signal); // caller space

    pointcut activationsInRegulations(Regulator regulator, int signal, Detector detector):
        activations(regulator, signal) && cflow(regulations(detector)) // Wormhole

    void around(Regulator regulator, int signal, Detector detector):
                activationsInRegulations(regulator, signal, detector)) {

        if (Authority.status(detector))
            proceed(regulator, signal, detector);
    }
}
```

Application of Wormhole pattern

Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljkIAmMU5

Test 2018 /2019

Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljkIAmMU5

The space probe information system is equipped with a communication service the probe uses to send messages to the control center on Earth and to receive commands from there. This service is being activated by instantiating the corresponding class called `CommService`:

```
public class CommService {
    ...

    public CommService() {
        ...
    }

    public void send(String message) {
        ...
    }
    ...
}
```

That is, to send a message, one would first need to create a communication service object:

```
CommService commService = new CommService();
commService.send("Houston, we have a problem.");
```

To improve reliability, an additional communication service is being added to the space probe information system. This service, represented by the `CommServiceAdd` class:

```
public class CommServiceAdd extends CommService {
    ...

    public CommServiceAdd() {
        ...
    }

    public void send(String message) {
        ...
    }
    ...
}
```

is to be activated if the original communication service fails. Provide an aspect-oriented solution to this problem.

1. (2 b) Provide the corresponding analytic model in the Theme/Doc notation in the view of themes and relationships (the basic view). Transform this view into the crosscutting theme view.

2. (3 b) Provide the corresponding design model in the Theme/UML notation. You may use the indirect relationships known from the JPDD notation.

3. (5 b) Provide the code of the corresponding aspect in the AspectJ programming language. If appropriate, apply the Worker Object Creation, Wormhole, or Cuckoo's Egg design pattern. Explain your decision.

Test 2018 /2019

The space probe information system is equipped with a communication service the probe uses to send messages to the control center on Earth and to receive commands from there. This service is being activated by instantiating the corresponding class called `CommService`:

```
public class CommService {
    ...

    public CommService() {
        ...
    }

    public void send(String message) {
        ...
    }
    ...
}
```

Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljkIAmMU5

That is, to send a message, one would first need to create a communication service object:

```
CommService commService = new CommService();
commService.send("Houston, we have a problem.");
```

Activate CommServiceAdd = Communication service

To improve reliability, an additional communication service is being added to the space probe information system. This service, represented by the `CommServiceAdd` class:

```
public class CommServiceAdd extends CommService {
    ...

    public CommServiceAdd() {
        ...
    }

    public void send(String message) {
        ...
    }
    ...
}
```

Theme 1

Theme 2 crosscutting

is to be activated if the original communication service fails. Provide an aspect-oriented solution to this problem.

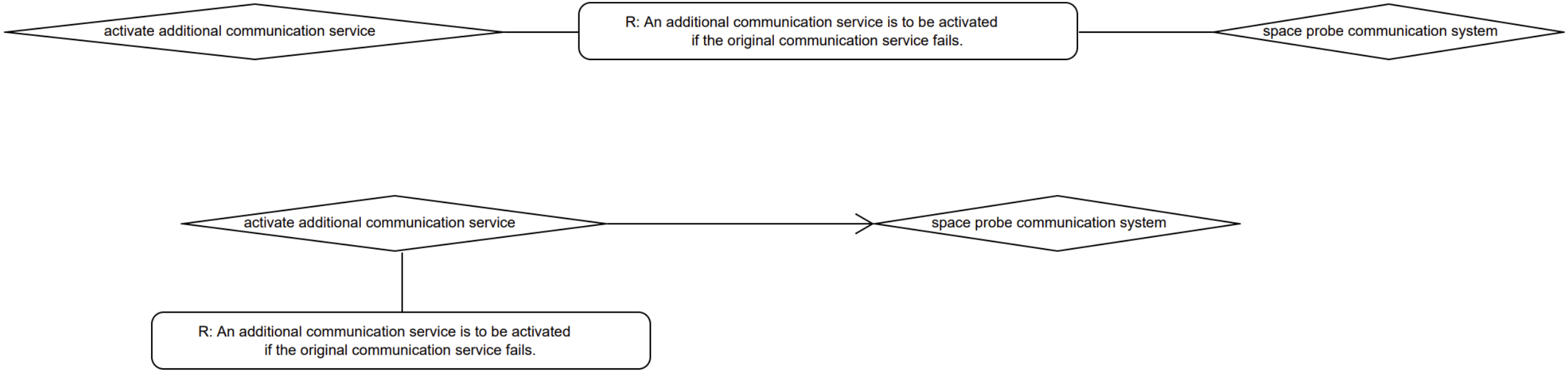
1. (2 b) Provide the corresponding analytic model in the Theme/Doc notation in the view of themes and relationships (the basic view). Transform this view into the crosscutting theme view.

2. (3 b) Provide the corresponding design model in the Theme/UML notation. You may use the indirect relationships known from the JPDD notation.

3. (5 b) Provide the code of the corresponding aspect in the AspectJ programming language. If appropriate, apply the Worker Object Creation, Wormhole, or Cuckoo's Egg design pattern. Explain your decision.

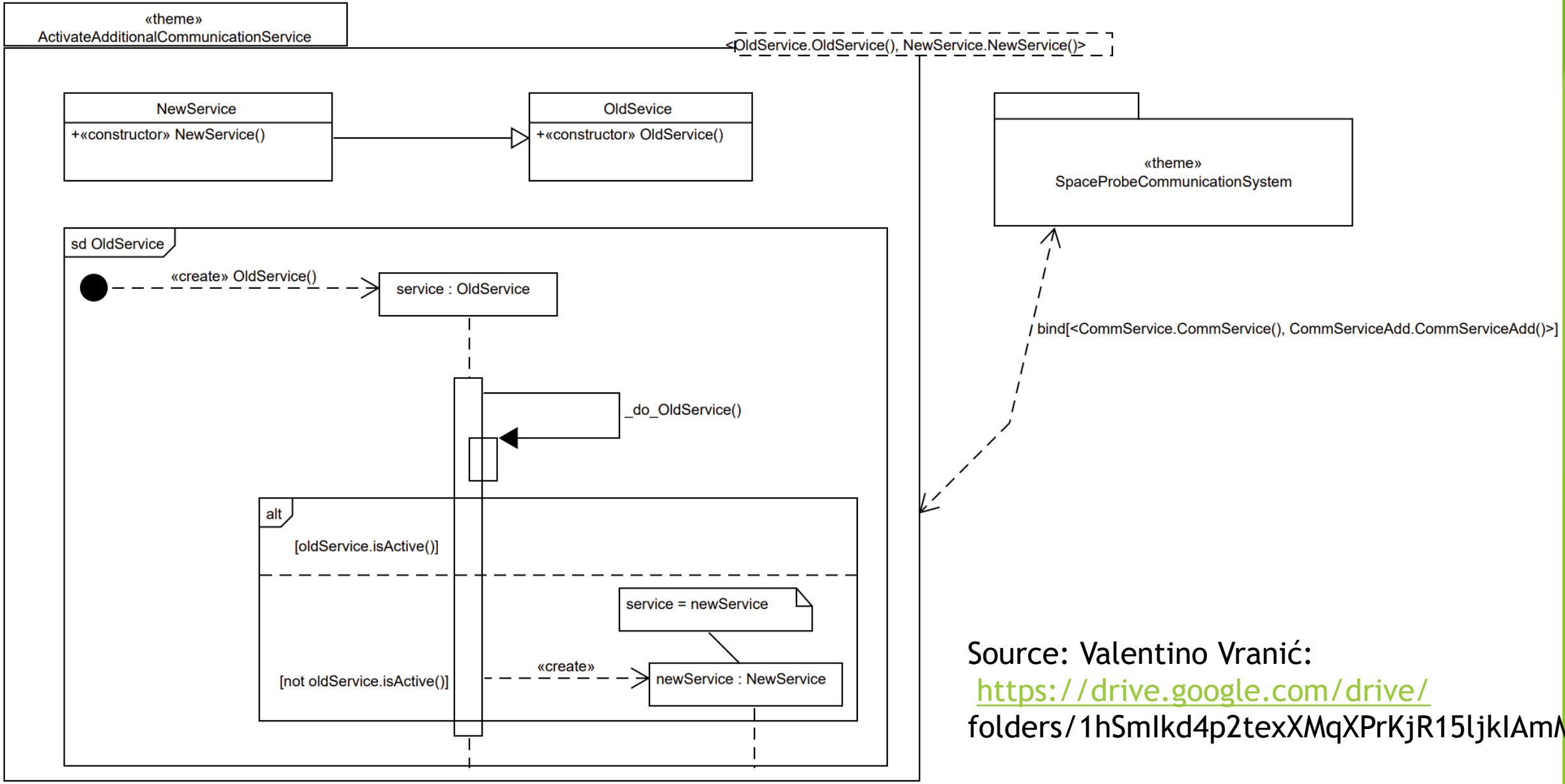
Problem = requirement

Themes DOC



Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljklAmMU5

Themes UML



Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljkIAmMU5

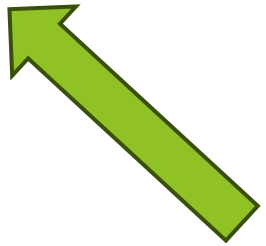
Code

```
public class CommService {
    public CommService() {
    }
    public void send(String message) {
    }
    public bool isActive() {
    }
}
```

```
public class CommServiceAdd extends CommService {
    public CommServiceAdd() {
    }
    public void send(String message) {
    }
}
```

```
public aspect ActivateAdditionalCommService {
    CommService around(): execution(void CommService.new()) {
        CommService commService = proceed();

        if (commService.isActive())
            return commService;
        else
            return new CommServiceAdd();
    }
}
```



Application of Cuckoo's egg pattern

Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljkIAmMU5

Test 2017 /2018

Do bankového systému je potrebné zaviesť biznis pravidlo, že sa pre bankové transakcie s hodnotou nad 10 000 eur vyžaduje dodatočné schválenie zo strany manažmentu. Potrebné je zabezpečiť, aby manažment mohol tieto transakcie dostávať na postupné schvaľovanie, ktoré nemusí prebehnúť bezprostredne po zadaní transakcie.

Transakcia sa realizuje volaním metódy `makeTransaction()`:

```
public class BankingSystem {
        ...
        public void makeTransaction(BankEntity source, BankEntity target,
                int amount, Currency currency) {
                ...
        }
}
```

Akékoľvek ďalšie triedy naznačte do úrovne, do akej je to nevyhnutné pre vyjadrenie riešenia. Samotný proces schvaľovania nie je predmetom úlohy.

1. (2 b) Uveďte príslušný analytický model v notácii Theme/Doc v pohľade tém a vzťahov (základný pohľad). Transformujte tento pohľad na pohľad pretínajúcich tém.

2. (3 b) Uveďte príslušný návrhový model v notácii Theme/UML. Môžete využiť jej rozšírenie o nepriame vzťahy z notácie JPDD.

3. (5 b) Uveďte kód výsledného aspektu v jazyku AspectJ. Ak by to bolo vhodné, uplatnite pritom aspektovo-orientovaný návrhový vzor Worker Object Creation, Wormhole alebo Cuckoo's Egg. Svoje rozhodnutie zdôvodnite.

Source: Valentino Vranić: https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljkIAmMU5

Test 2017 /2018

Theme 1

Theme 2 crosscutting

Do bankového systému je potrebné zaviesť biznis pravidlo, že sa pre bankové transakcie s hodnotou nad 10 000 eur vyžaduje dodatočné schválenie zo strany manažmentu. Potrebné je zabezpečiť, aby manažment mohol tieto transakcie dočasne dať na postupné schvaľovanie, ktoré nemusí prebehnúť bezprostredne po zadaní transakcie.

Transakcia sa realizuje volaním metódy `makeTransaction()`:

```
public class BankingSystem {
        ...
        public void makeTransaction(BankEntity source, BankEntity target,
                int amount, Currency currency) {
                ...
        }
}
```

Akékoľvek ďalšie triedy naznačte do úrovne, do akej je to nevyhnutné pre vyjadrenie riešenia. Samotný proces schvaľovania nie je predmetom úlohy.

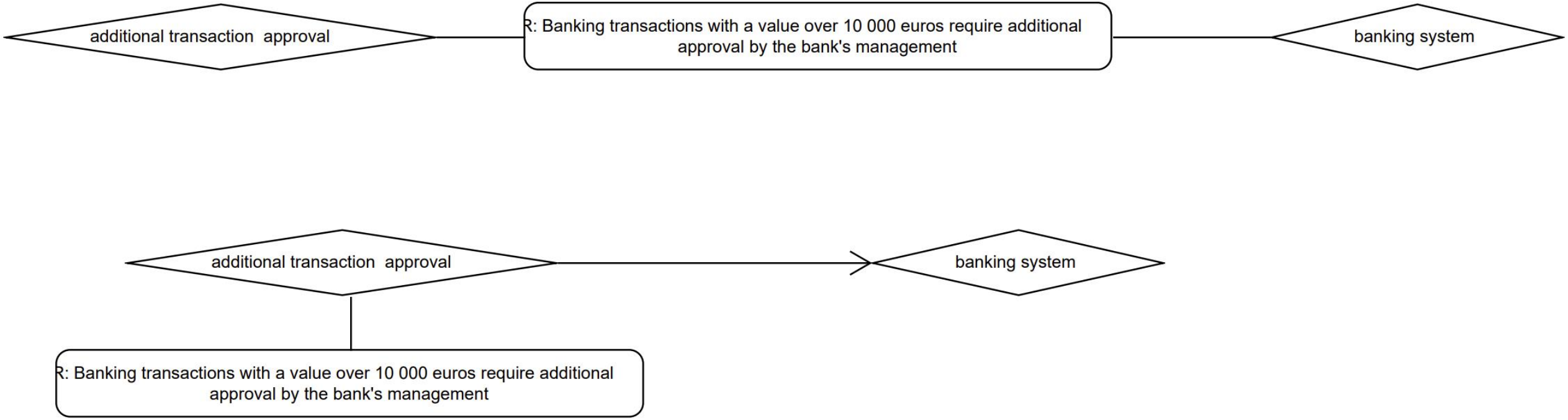
1. (2 b) Uveďte príslušný analytický model v notácii Theme/Doc v pohľade tém a vzťahov (základný pohľad). Transformujte tento pohľad na pohľad pretínajúcich tém.

2. (3 b) Uveďte príslušný návrhový model v notácii Theme/UML. Môžete využiť jej rozšírenie o nepriame vzťahy z notácie JPDD.

3. (5 b) Uveďte kód výsledného aspektu v jazyku AspectJ. Ak by to bolo vhodné, uplatnite pritom aspektovo-orientovaný návrhový vzor Worker Object Creation, Wormhole alebo Cuckoo's Egg. Svoje rozhodnutie zdôvodnite.

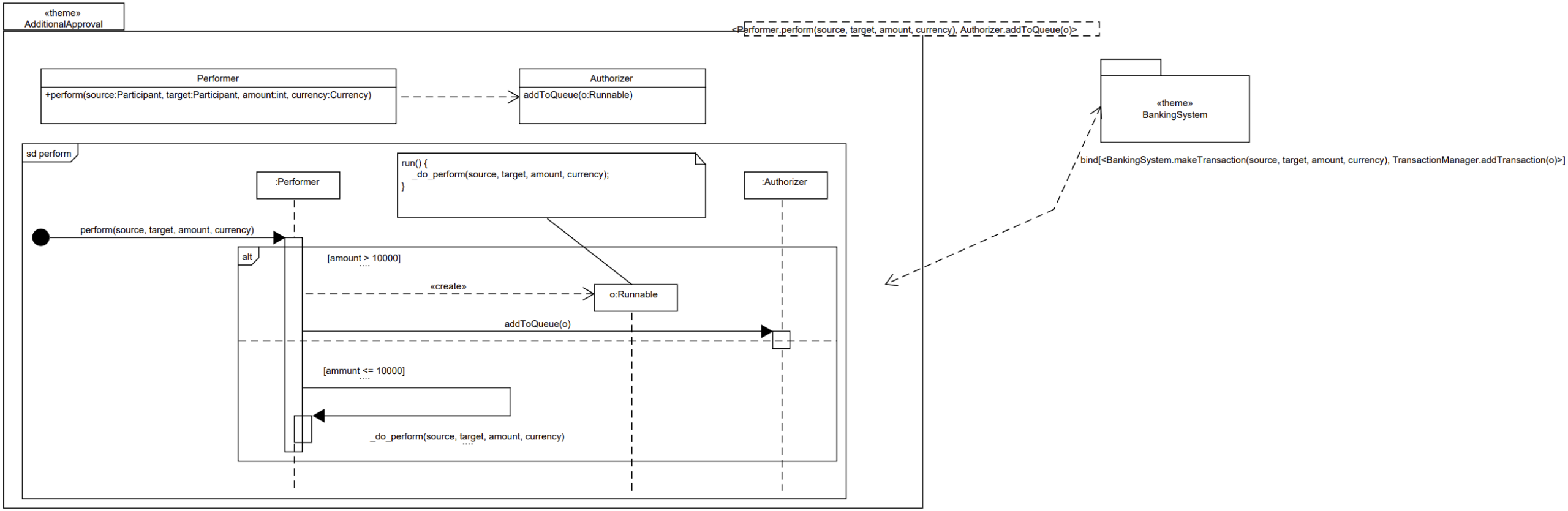
Source: Valentino Vranić: https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljkIAmMU5

Themes DOC



Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljkIAmMU5

Themes UML



Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljkIAmMU5

Code

```
public class BankingSystem {
    public void makeTransaction(BankEntity source, BankEntity target, int ammount, Currency currency) {
    }
}
public class TransactionManager {
    public List<Object> transactionsToBeAuthorized;
    TransactionManager getTransactionManager() {
    }
    void addTransaction(transaction) {
    }
}
```

Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmIkd4p2texXMqXPrKjR15ljkIAmMU5

```
public aspect TransactionAdditionalAuthorization {
    // other arguments might have been captured, too, so that the authorizer could make a qualified decision
    void around(int ammount): call(void BankingSystem.makeTransaction(..)) && args(*, *, ammount, *) {
        if (transaction.getAmmount() < 10000)
            proceed(ammount);
        else {
        TransactionManager.getTransactionManager().addTransaction(new Runnable() {
            public void run() {
                proceed(ammount);
            }
        });
    }
  }
}
```

Application of Proceed object pattern

Source: Valentino Vranić:
https://drive.google.com/drive/folders/1hSmlkd4p2texXMqXPrKjR15ljklAmMU5

JPDD

Join Point Designation Diagrams

- To determine join points
- Graphical notation to expressed join points
- Query models in broader sense:

  7V. Stricker, S. Hanenberg, and D. Stein. Designing Design Constriants in the UML Using Join Point Designation Diagrams.
  In Proc. of 47th Int. Conf. on Objects, Components, Models, and Patterns, TOOLS EUROPE 2009, Zurich Switzerland, June/July 2009, Springer

- For activity diagrams, state diagrams, structural and sequence diagrams

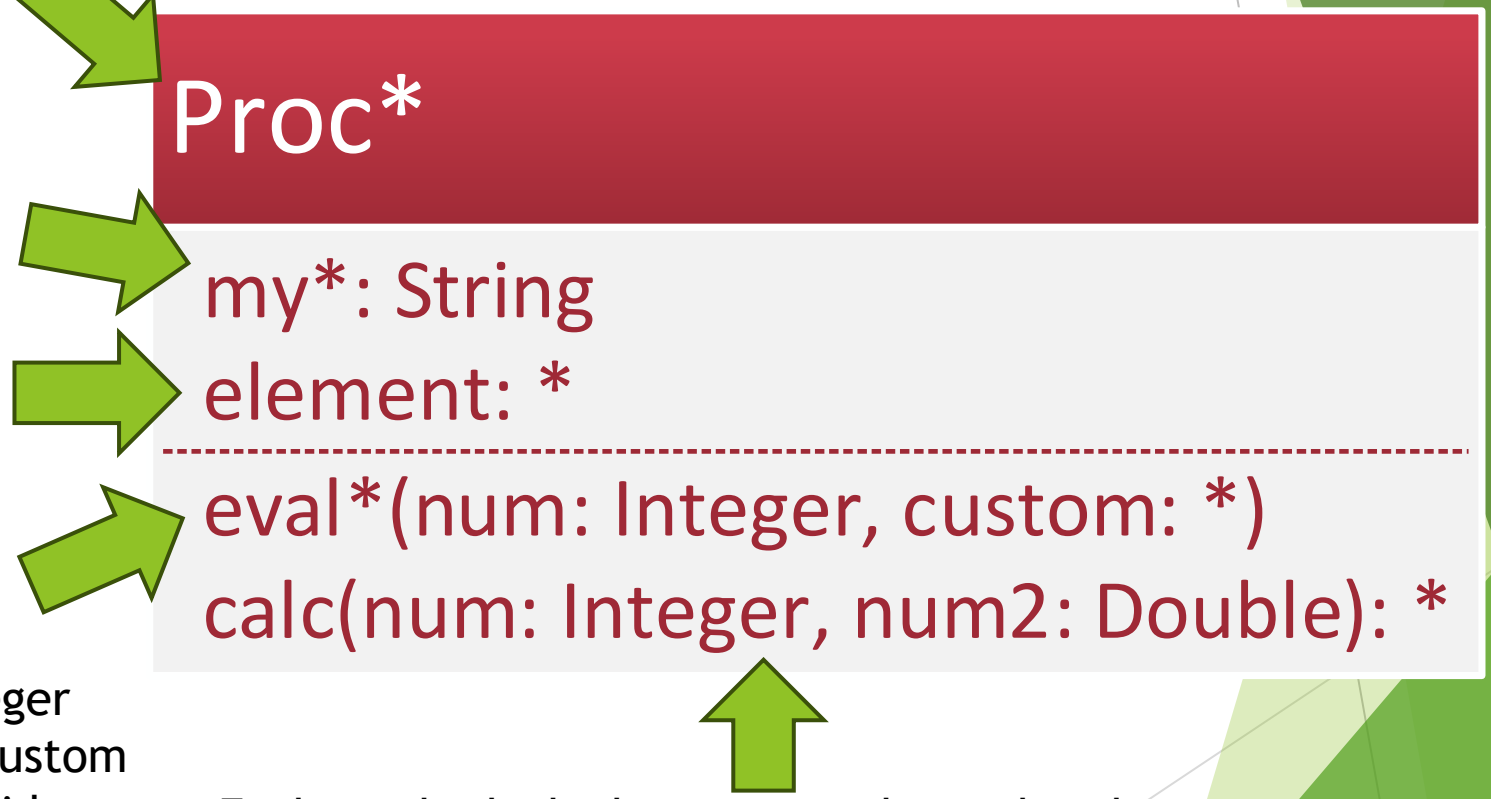
Lexical Correspondence

Each class that starts on Proc

Each variable with type String that starts on my

Each variable with name element of any type

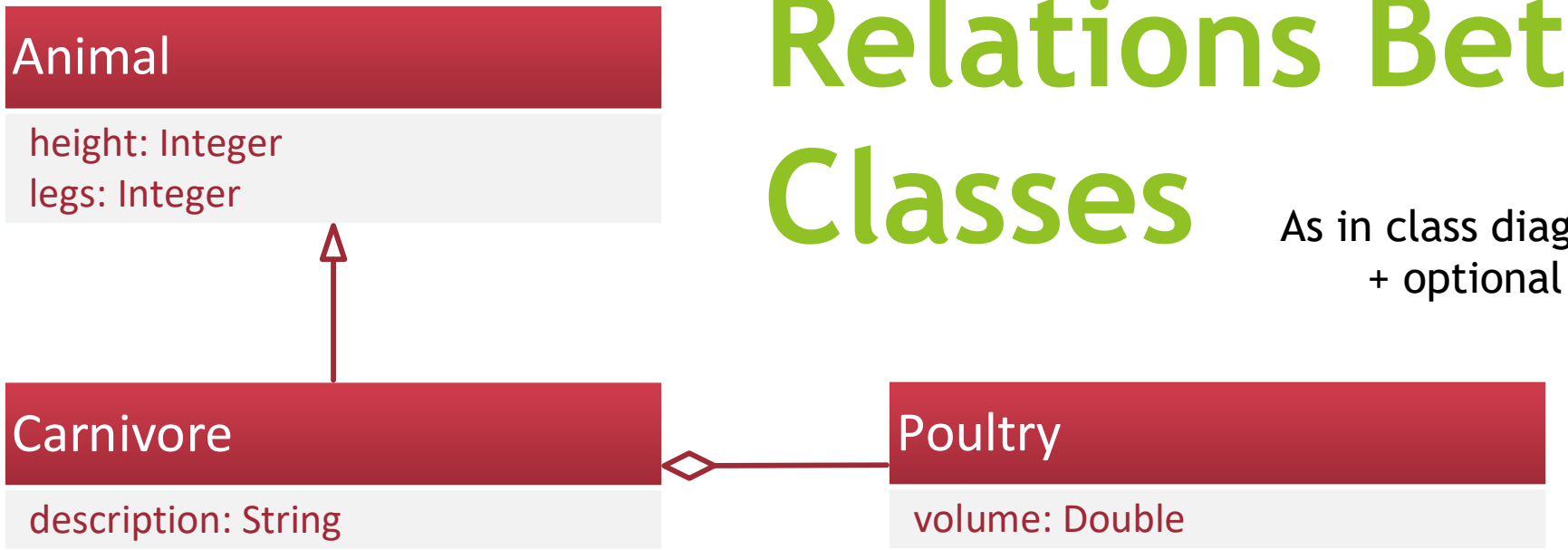
Each method which name is starting with eval, and with two arguments, where the first is Integer called num and second is called custom of any type, and return type is void

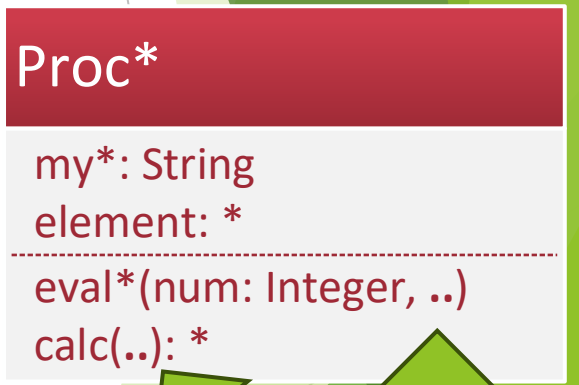


Each method which name is calc, and with two arguments, where the first is Integer called num and second is Double called num2, and return type can be any

Relations Between Classes

As in class diagram
+ optional JPDD relations

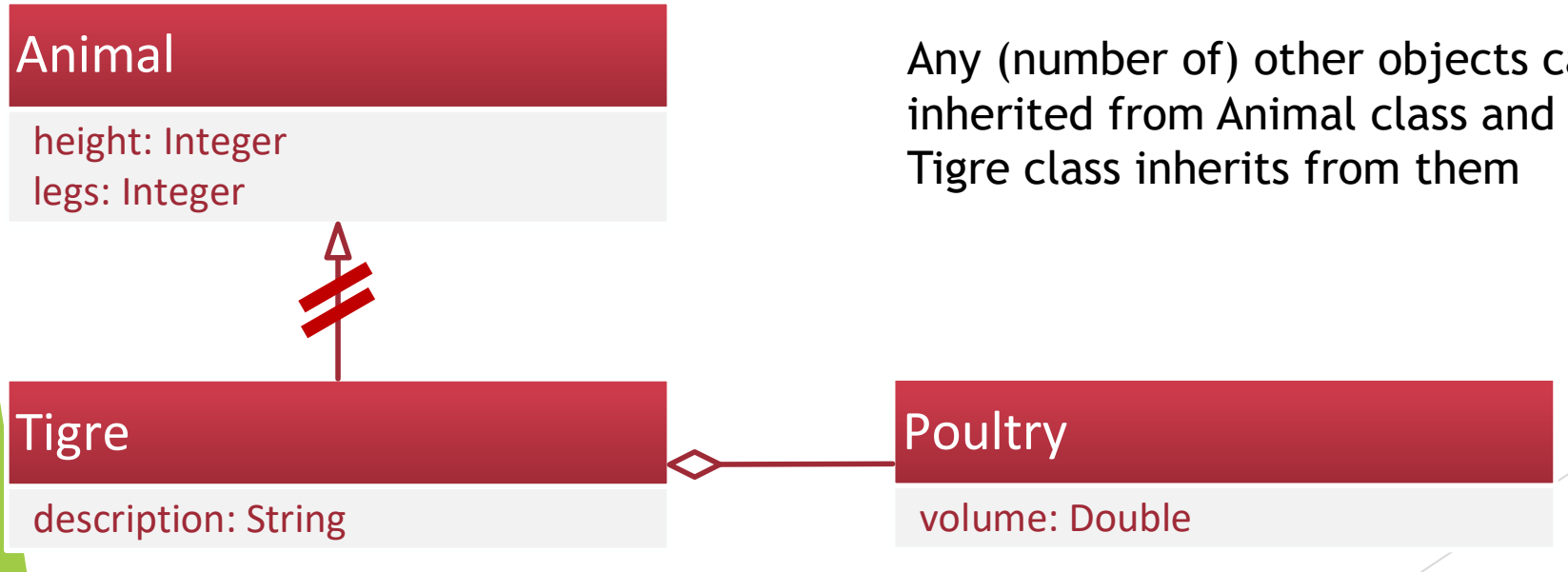




Any other arguments
in any number

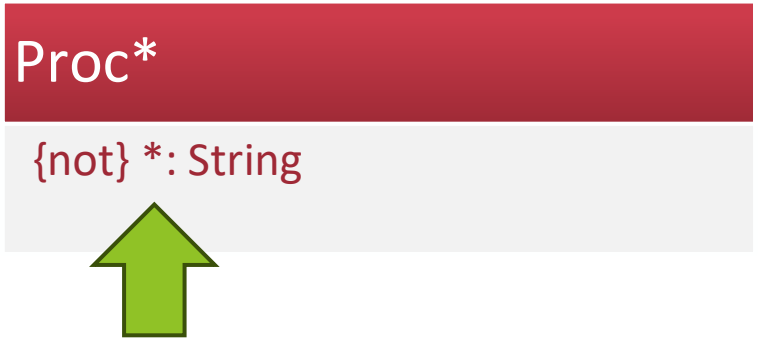
The use of indirect operator

Any (number of) other objects can be inherited from Animal class and still Tigre class inherits from them



Lexical correspondence

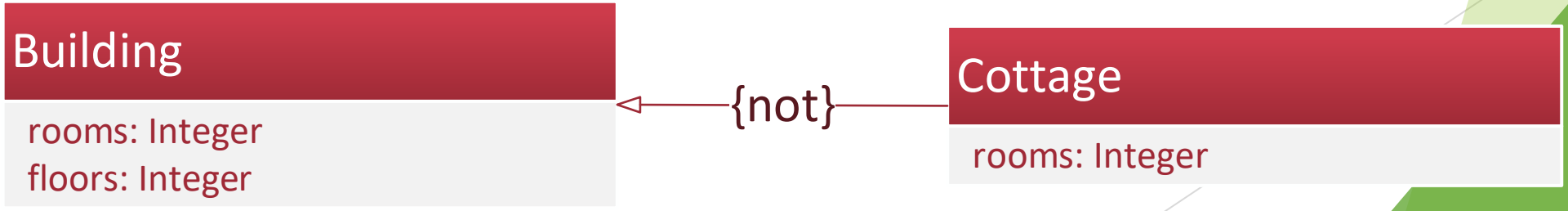
Negation:



Each variable that does not start on my

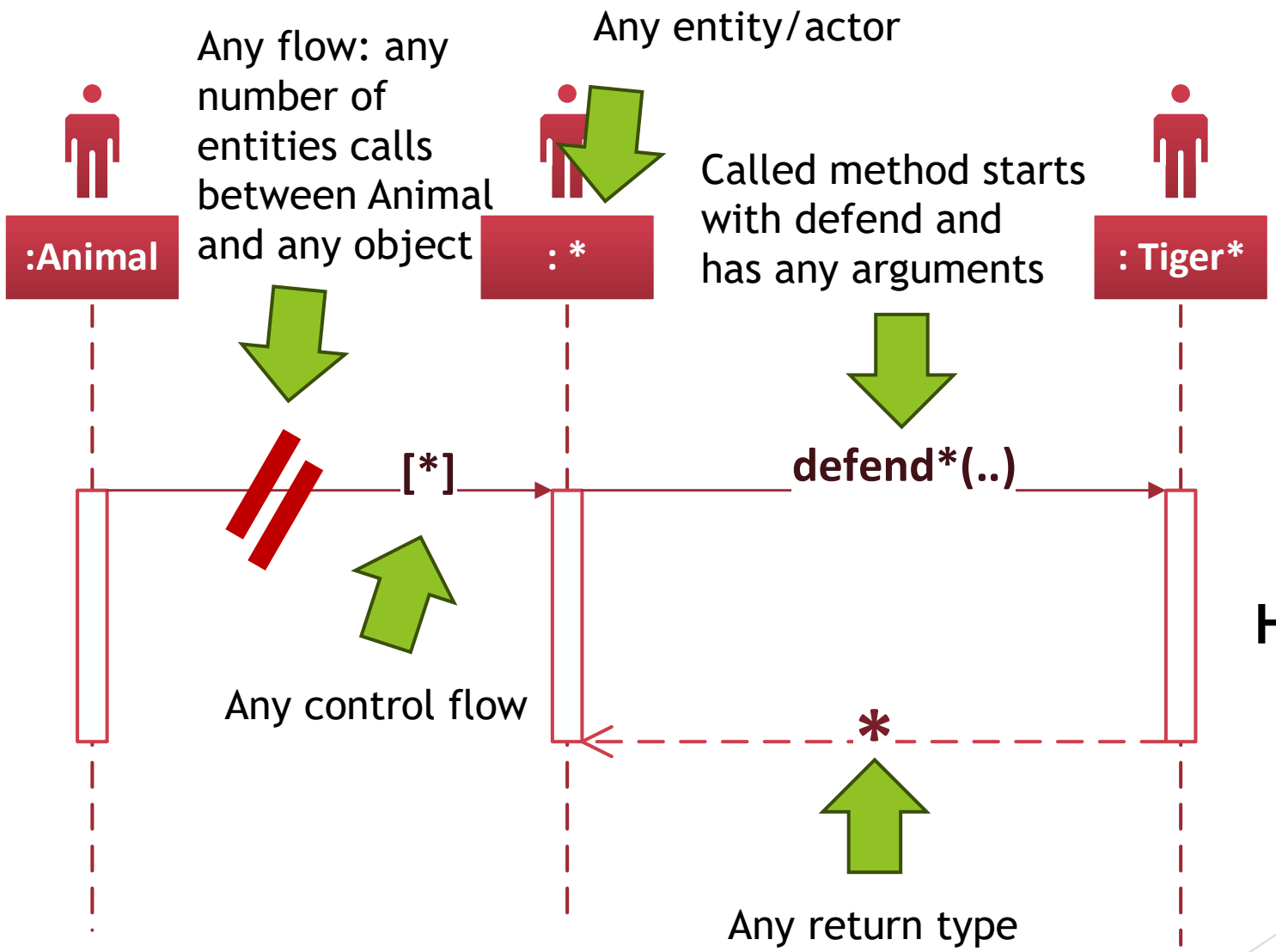
Each variable except those with type of String that does not start on my

Negation in relations between classes:



Sequence Diagrams

Sending messages

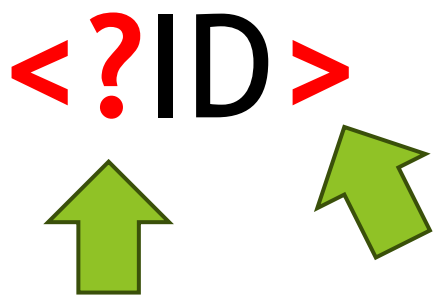


How to interpret diagram:

Trying to find all such configurations of sending messages that correspond to the diagram (this)

Parametrization

- Inside templates near the name of elements where they are positioned
- Can be used in references inside JPDD



Starts with ? question mark

Inside <> parenthesis

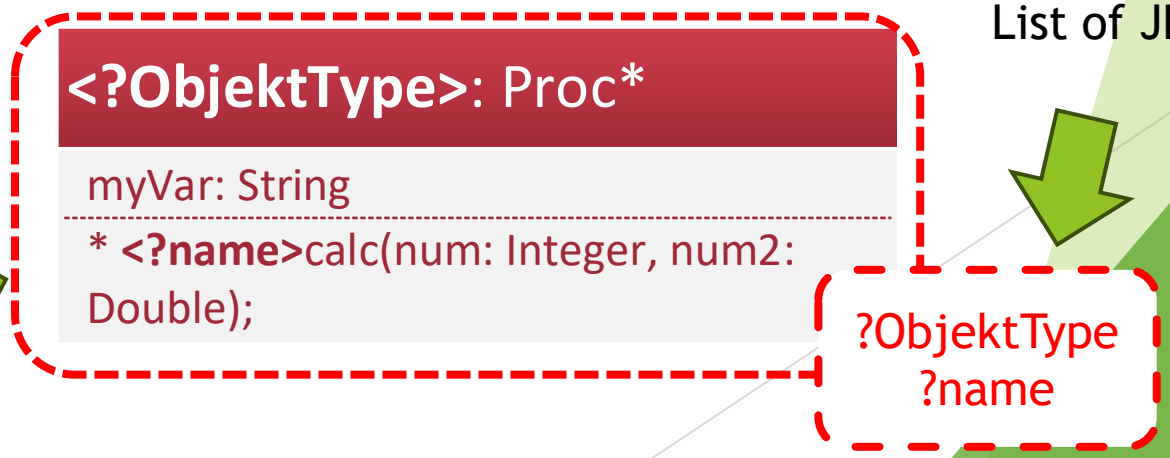
<?ObjektType>: Proc*

myVar: String

* **<?name>**calc(num: Integer, num2: Double);

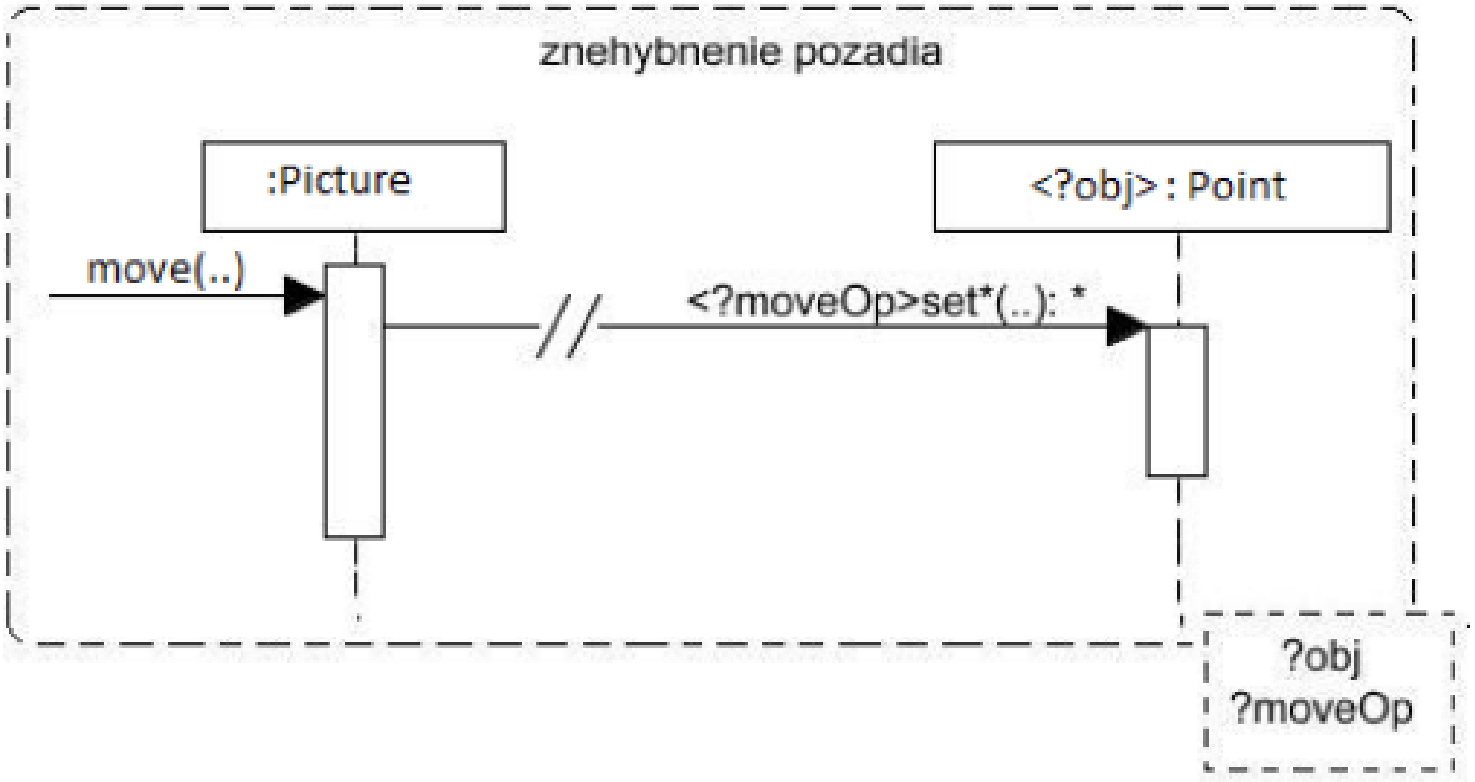
Parameters are located in bottom part:

List of JPDD parameters



JPDD is bounded in an ellipse

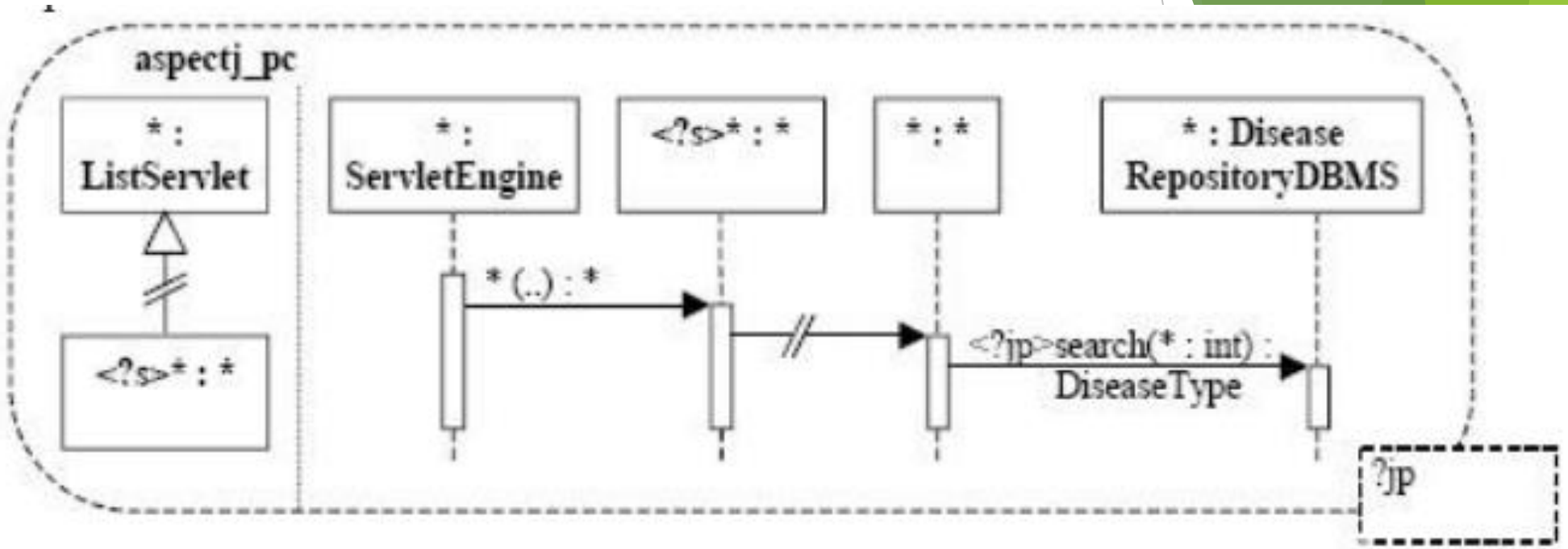
JPDD Examples



Source: Aspekty v analýze a návrhu – prístupy Theme a JPDD Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoja softvéru, Valentino Vranić, 2017

```
pointcut MovingOp(Point o):
    target(o) && call(* set*(..)) && cflow(call(* Picture.move(..)));
```

JPDD Examples




```
pointcut aspectj_pc():
  call(DiseaseType DiseaseRepositoryDBMS.search(int)) &&
  cflow(call(* ListServlet+.*(..)) && this(ServletEngine))
```

Source: Aspekty v analýze a návrhu
– prístupy Theme a JPDD Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoja softvéru, Valentino Vranić, 2017

Resources

Application of Themes approach: Bc. Pavol Michalco: PRÍPADY POUŽITIA A TÉMY V PRÍSTUPE THEME/DOC
D. Stein. **Join Point Designation Diagrams**: A Visual Design Notation for Join Point Selections in Aspect-Oriented Software Development. PhD. thesis, Universität Duisburg-Essen, 2010.
E. Baniassad and S. Clarke, **"Theme: an approach for aspect-oriented analysis and design,"** Proceedings. 26th International Conference on Software Engineering, Edinburgh, UK, 2004, pp. 158-167, doi: 10.1109/ICSE.2004.1317438.

AMPLE Project home page (Aspect-Oriented, Model-Driven Product Line Engineering: Web page *Rashid, A., Royer, J.C., Rummler, A. (eds.): Aspect-Oriented, Model-Driven Software Product Lines: The AMPLE Way (09 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*
Pointcut Language in AspectJ For Analogies: The AspectJ in Action *Laddad, Ramnivas, 2003. AspectJ in action: practical aspect-oriented programming. Greenwich, CT: Manning. ISBN 978-1-930110-93-9.*
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.
D. Stein, S. Hanenberg, and R. Unland. Query Models. In Proceedings of 7th International Conference on the Unified Modeling Language (UML 2004) – The Language and Its Applications, Lisbon, Portugal, October 2004, LNCS 3273, Springer.