Aspekty v analýze a návrhu – prístupy Theme a JPDD

Poznámky k prednáškam z predmetu Aspektovo-orientovaný vývoj softvéru

Valentino Vranić
http://fiit.sk/~vranic/, vranic@stuba.sk
Ústav informatiky, informačných systémov a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave

9. október 2017

Obsah

1 Úvod

1.1 Aspektovo-orientovaná analýza a návrh

- Analýza – návrh – modelovanie
- Ako vyvíjať systémy tak aby sa využili výhody aspektovo-orientovaného programovania
- Možno hľadať komplexnú podporu celého procesu alebo len jeho špecifických častí
- Štandardizácia objektovo-orientovanej analýzy a návrhu – pretrvávajú rozdiely
- Ale predsa máme UML ako de facto štandard
- Pre aspektovo-orientovanú analýzu a návrh UML nestačí
- V aspektovo-orientovanej analýze a návrhu je potrebné:[1]
  - reprezentovať oddelene modely pretínajúcich záležitostí
  - špecifikovať kompozíciu týchto modelov
  - umožniť túto kompozíciu a validáciu jej špecifikácie

1.2 Symetrickosť AOP

- Rozdiely medzi aspektovo-orientovanými prístupmi predstavujú dodatočný problém v hľadaní vhodného spôsobu aspektovo-orientovanej analýzy a návrhu
- Asymetrické aspektovo-orientované prístupy[2]
  - Rozlišuje sa medzi základnou dekompozíciou a pretínajúcimi záležitosťami (aspektami)
- Symetrické aspektovo-orientované prístupy
  - Program ako celok vzniká spájaním rozličných pohľadov (teda aspektov)
- Príklady:
  - AspectJ: asymetrický prístup
  - HyperJ: symetrický prístup

[1] R. E. Filman, T. Elrad, S. Clarke, M. Akşit. *Aspect-Oriented Software Development.* Addison-Wesley, 2004.

[2] W. H. Harrison, H. L. Ossher, and P. L. Tarr. Asymmetrically vs. Symmetrically Organized Paradigms for Software Composition. Technical report RC22685, IBM Research, 2002.

1.3 Prístupy k aspektovo-orientovanej analýze a návrhu

- Prístupy tesne spojené s programovacím jazykom (predovšetkým AspectJ)
- Pozrieme sa bližšie na Theme a JPDD
- Iné prístupy zahŕňajú:
  - Reusable Aspect Models (RAM)[3]
  - Prístup Elrada et al. – založený na štandardnom UML
  - Cosmos – modelovanie záležitostí
- Štúdie viacerých prístupov k aspektovo-orientovanému modelovaniu na spoločnom príklade v TAOSD[4]
- Správy AOSD-Europe (Moodle)[5]

2 Prístup Theme

2.1 Základné aktivity prístupu Theme

- Analýza (Theme/Doc)
  - Identifikácia a spresnenie tém a ich vzťahov v požiadavkách
  - Určenie základných a pretínajúcich tém
- Návrh (Theme/UML)
  - Návrh identifikovaných základných tém a aspektových (pretínajúcich) tém
  - Návrh aspektových tém, ktoré sa objavia počas návrhu
- Kompozícia (Theme/UML)
  - Špecifikácia a návrh kompozície tém: prekrývanie a pretínanie

2.2 Theme

- Theme – spojenie dvoch prístupov:[6] [7] [8]
  - Theme/Doc – aspektovo-orientovaná analýza: identifikácia aspektov v požiadavkách

[3] `http://www.cs.mcgill.ca/~joerg/SEL/RAM.html`

[4] Transactions on Aspect-Oriented Software Development VII: A Common Case Study for Aspect-Oriented Modeling, Springer, 2010.

[5] A. Rashid et al. Aspect-Oriented Software Development in Practice: Tales from AOSD-Europe. Computer, 43(2): 19–26, 2010. `http://dx.doi.org/10.1109/MC.2010.30`

[6] S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach.* Addison-Wesley, 2005.

[7] E. Baniassad and S. Clarke. Theme: An Approach for Aspect-Oriented Analysis and Design. ICSE 2004. `http://www.cse.cuhk.edu.hk/~elisa/papers/theme.pdf`

[8] S. Clarke and R. J. Walker. Towards a Standard Design Language for AOSD. In Proc. of 1st Int. Conf. on AOSD, Enschede, The Netherlands, 2002. `http://pages.cpsc.ucalgary.ca/~rwalker/publications/clarke2002a.pdf`

  - Theme/UML – aspektovo-orientovaný návrh v UML
- Prístupy sú založené na pojme *témy* (theme)
- Téma predstavuje záležitosť alebo koncept vo všeobecnosti
- Rozdiel oproti OO: nepredurčujeme (aspoň v prvom priblížení) ako bude daný koncept implementovaný

3 Analýza tém

3.1 Pretínajúce záležitosti v požiadavkách

- AOP nemá byť používané len dodatočne na vyriešenie problémov základnej dekompozície
- Už na úrovni špecifikácie požiadaviek možno identifikovať pretínajúce záležitosti
- Skoré aspekty (early aspects)
- Jeden z prístupov k AO analýze požiadaviek je Theme/Doc

3.2 Identifikácia tém

- Základná množina tém sa identifikuje v požiadavkách
- Dôležitý krok – stratégie identifikácie
- Najčastejšie sa jednoducho v špecifikácii požiadaviek hľadajú identifikovateľné časti funkcionality
- Tri pohľady na témy:
  - pohľad tém a vzťahov
  - pohľad pretínajúcich tém
  - individuálny pohľad

3.3 Generovanie pohľadov...

- Pohľady generuje nástroj na základe zadaných údajov (ešte stále nie je verejne dostupný[9])
- Dostupná je však podpora pre Theme/UML[10] [11]
  - Theme/UML profil (XMI)
  - Nástroj na podporu Model-Driven Theme/UML
- Na našej fakulte bol vytvorený Theme profil pre IBM RSA (aj Theme/Doc, aj Theme/UML)[12]

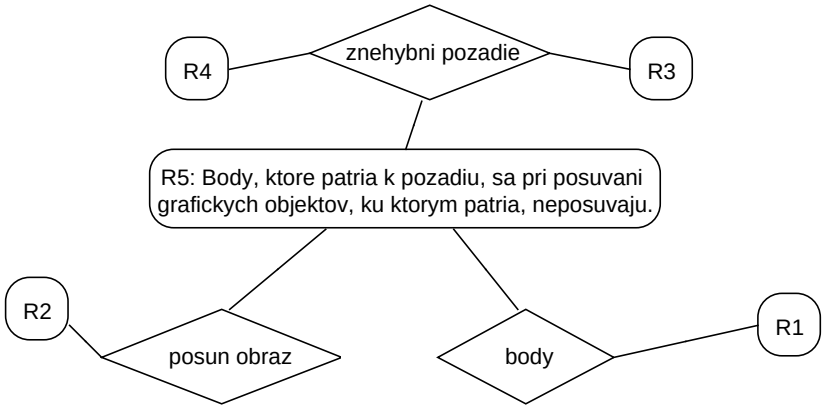
[9] `http://www.thethemeapproach.com/downloads.html`
[10] `https://www.dsg.cs.tcd.ie/aspects/themeUML/`
[11] `http://www.scss.tcd.ie/Eamonn.Linehan/lero/handbook_dev.pdf`
[12] B. Kuliha. Realizing Changes by Aspects at the Design Level. Diplomová práca, Slovenská technická univerzita v Bratislave, 2010. (dokumentový server AIS)

3.4 Pohľad tém a vzťahov

- Vzťahy medzi témami, ale len cez požiadavky
- Témy sú opísané požiadavkami – nemajú priamy opis
- 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.

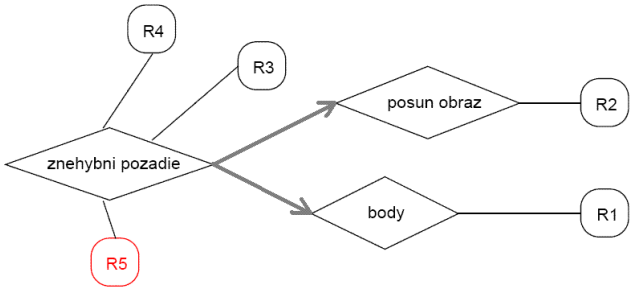
  ...

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



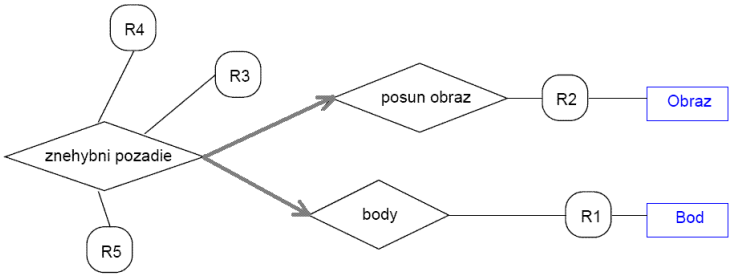
3.5 Pohľad pretínajúcich tém

- Identifikujeme požiadavky, ktoré sa vzťahujú na viac než jednu tému
- Pokúsime sa tieto požiadavky prepísať tak, aby sa vzťahovali len na jednu tému – ako keby sme sa snažili takto témam nájsť opis (témy nie sú opísané explicitne – len prostredníctvom požiadaviek)
- Požiadavky za týmto účelom možno aj rozdeľovať
- Niektoré požiadavky napriek tomu zostanú spoločné
- V takom prípade sa určí téma, ktorá je pre danú požiadavku dominantná
- Takáto téma je potom pretínajúca – aspektová
- Pretínanie znázorníme orientovanou hranou



3.6 Individuálny pohľad

- Pohľad pretínajúcich tém pre jednu tému
- Zahŕňa identifikované entity



3.7 Práca s témami

- Pri reálnych systémoch vznikajú zložité siete tém
- Prvotne identifikované témy možno rozdeľovať, zlučovať, rušiť
- Prístup je iteratívny: bežné je, že vznikajú ďalšie požiadavky, jestvujúce sa menia alebo zanikajú atď.

3.8 A simple retail support application: požiadavky

A simple retail support application: requirements

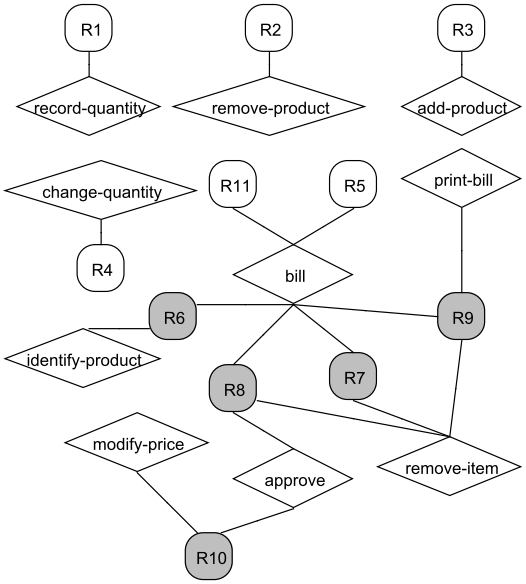
1. The application will record and maintain the product quantity in the stock in the central database.
2. The storekeeper can remove products from the database.
3. The storekeeper can add products into the database.
4. The storekeeper can change the product quantity in the database.
5. The cashier can bill the item by manually entering the bar code or with a bar code reader.
6. Only the products recorded in the database can be billed.
7. The billed items can be removed from the bill until it has been closed.

8. The billed item removal must be approved by a store manager by entering his authentication data.

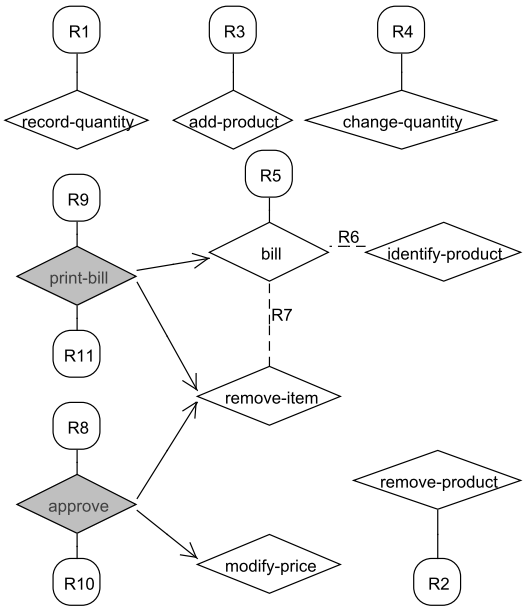
9. The billed items will be printed on the cash desk bill as they are entered. The bill will embrace the store name, billed items, information on removed billed items, the total amount of money to be paid, and date and time.

10. The product price can be entered or modified only by a properly authenticated store manager.

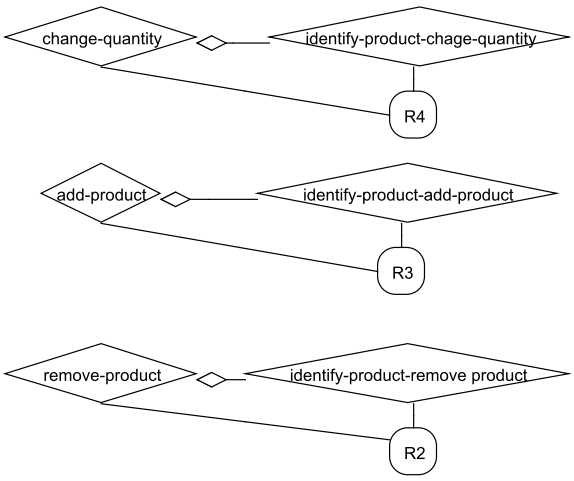
3.9 A simple retail support application: témy a vzťahy



3.10 A simple retail support application: pretínajúce témy



3.11 A simple retail support application: individuálne témy



- V pohľade individuálnych tém môžu vystupovať aj podtémy

4 Návrh tém

4.1 Theme/UML

- Rozširuje UML

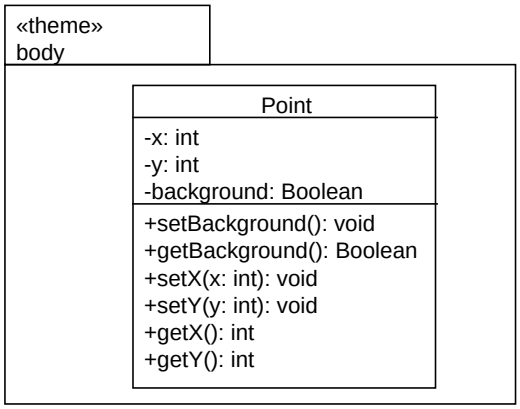
- Nezávislý od programovacieho jazyka
- Theme/UML podporuje aj symetrické, aj asymetrické uvažovanie
- Definované je zobrazenie na konštrukcie jazyka AspectJ
- Pretínajúce témy sú špecifikované pomocou šablón (*template*) balíkov v UML

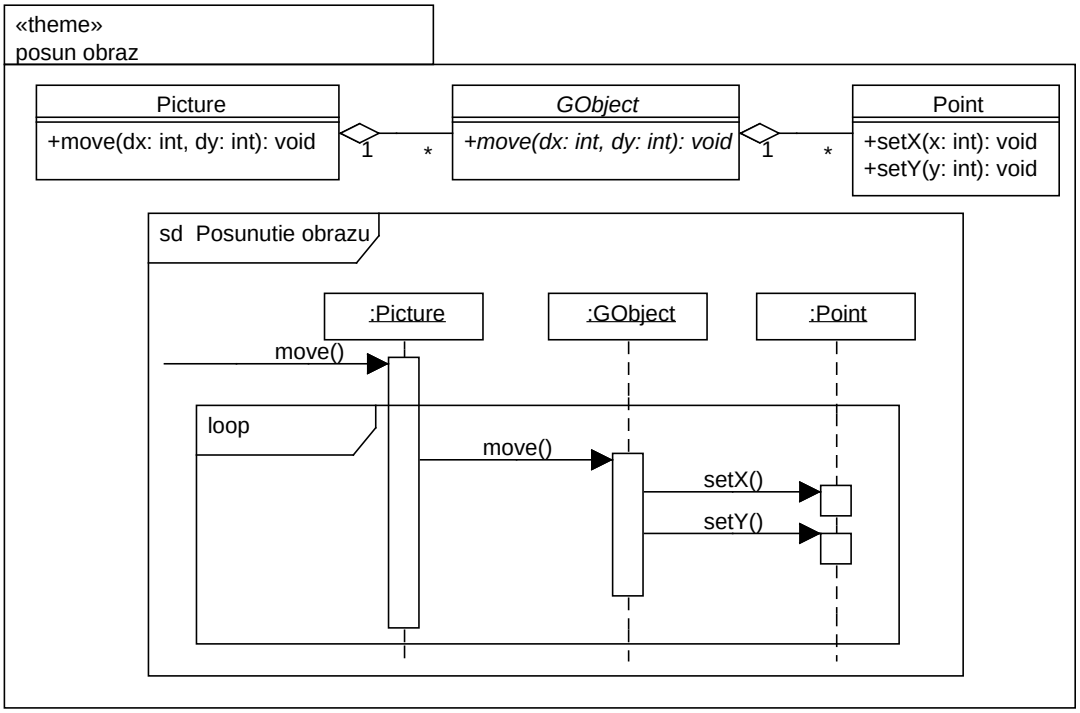
4.2 Témy

- Každý pohľad na systém sa modeluje nezávisle ako téma
- Následne sa špecifikuje kompozícia tém
- Témy sú reprezentované balíkmi (*package*) v UML
- Balíky obsahujú rôzne pohľady a špecifikujú aj štruktúru, aj správanie
- Dva druhy tém: základné a pretínajúce (ako na konci Theme/Doc)

4.3 Návrh základných tém

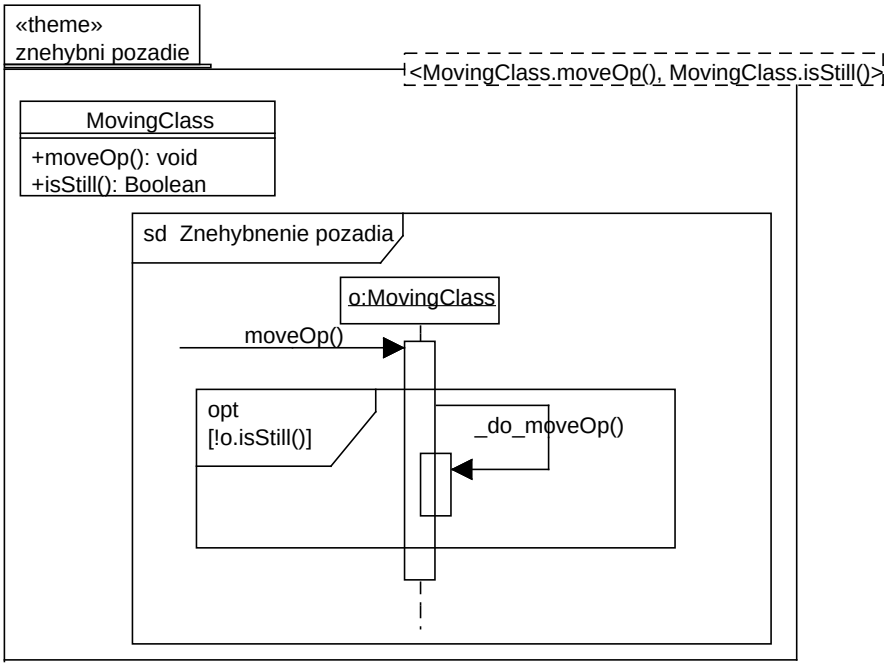
- Témy navrhujeme samostatne
- Nemôžeme to robiť úplne oddelene – pri spájaní by sme museli riešiť strašne veľa konfliktov





4.4 Návrh pretínajúcich tém

- Pretínajúce témy sú reprezentované šablónami balíkov
- Pretínanie sa reprezentuje diagramami sekvencií v rámci týchto balíkov
- Pre každý diagram sekvencií sa v lomených zátvorkach (<>) uvedú parametre šablóny balíka (v čiarkovanom obdĺžniku umiestnenom v hornom pravom rohu)
- Prvý parameter v skupine vymedzenej lomenými zátvorkami je spúšťací
- Samotná spúšťacia operácia je dostupná pod jeho názvom s prefixom `_do_`



5 Kompozícia tém

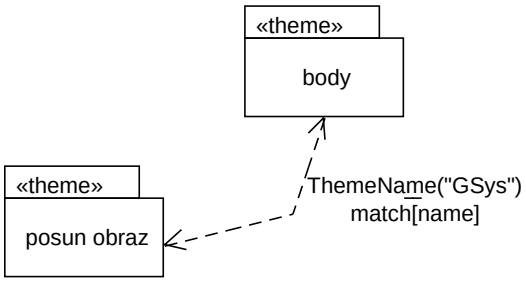
- Výsledná aplikácia vzniká kompozíciou tém
- Rozlišuje sa medzi:
  - kompozíciou základných tém (zobrazením)
  - kompozíciou pretínajúcich tém (viazaním)

5.1 Kompozícia základných tém

- Kompozícia základných tém sa uskutočňuje definovaním zobrazenia (matching/mapping)
- Zobrazenie sa definuje graficky – hranou
- Kompozíciou základných tém vzniká nová téma, ktorej názov sa definuje výrazom `ThemeName` na hrane
- Dá sa robiť kompozícia viacerých tém naraz
- Je možné definovať rôzne zobrazenia pre rôzne aplikácie
- Zobrazenie obvykle prebieha na základe pomenovaní – `match[name]`
- Dá sa určiť aj explicitne (graficky)
- Dva typy kompozície:
  - merge – spojenie všetkých prvkov daných tém (obojsmerná šípka)

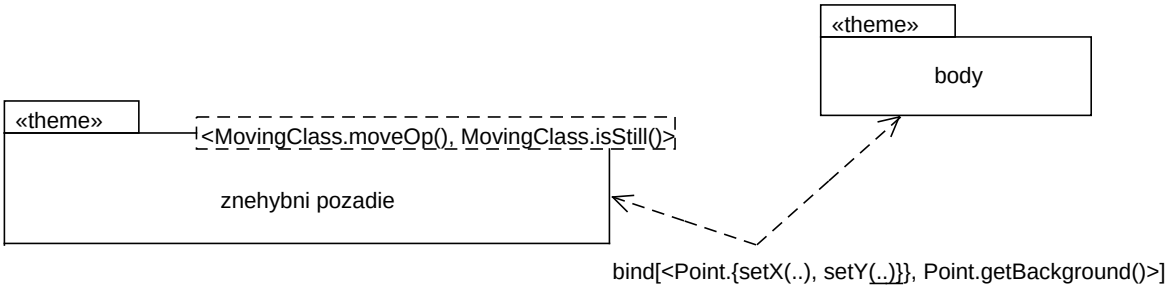
  - override – prvky prekonávajúcej témy nahrádzajú rovnomenné prvky v prekonávanej téme (musí byť práve jedna) a ostatné sa pridajú (jednosmerná šípka)

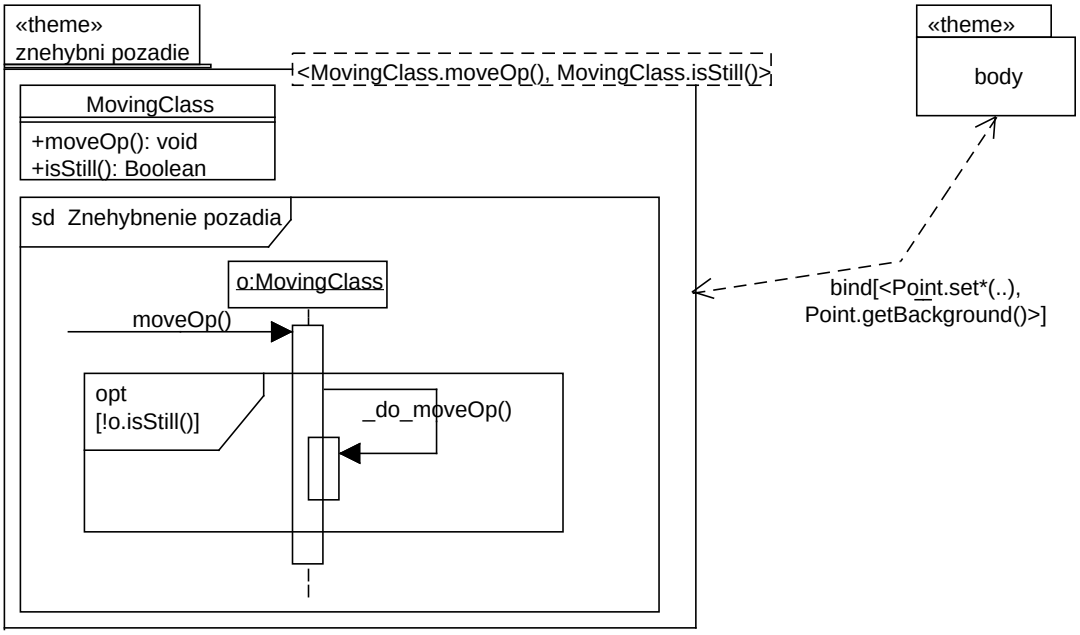
- Riešenie konfliktov – explicitným uvedením jednej z možnosti (`resolve`())
- Implementácia v AspectJ – väčšinou pomocou medzitypových deklarácií, ale aj pomocou videní (override)



5.2 Kompozícia pretínajúcich tém

- Pretínajúce témy sa skladajú so základnými témami
- Viazanie sa definuje spojením tém hranou a výrazom `bind`
- Vo výraze `bind` sa vymenujú zodpovedajúce operácie, ktoré sa naviažu na parametre šablóny balíka
- Možno uviesť aj viac operácií pre jeden parameter – uvedú sa ako množina (v skupinových zátvorkách)
- Možno použiť aj náhradné znaky podobne ako v AspectJ
- Na diskusiu:
  - Kompozícia dvoch alebo viacerých pretínajucich tém
  - Kompozícia základnej a pretínajúcej témy zobrazením



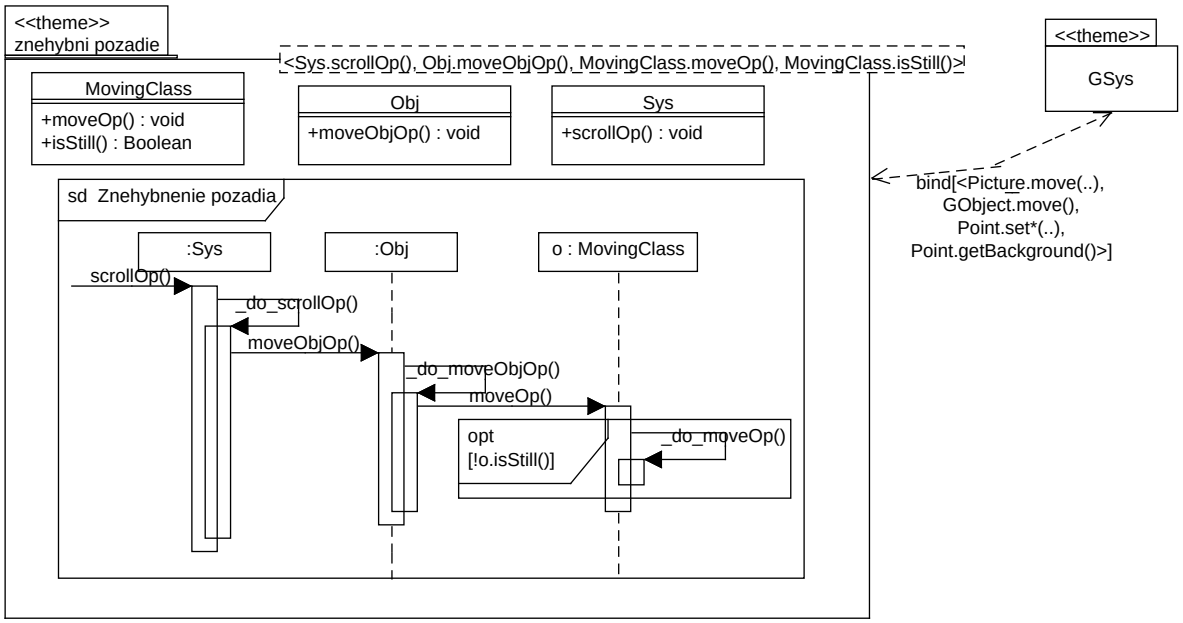


pointcut `MovingOp(Point o)`: **target**(o) && **call**(∗ **set**∗(..));

- Takto blokujeme posúvanie všetkých bodov pozadia.
- Chceme však blokovať posuny bodov pozadia len v rámci posunu obrazu:

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

- Posun obrazu sa realizuje metódou `Picture.move`(), ale samotné volania `Point.setX`() a `Point.sety`() sa môžu vyskytovať v metódach `GObject.move`() volaných v metóde `Picture.move`()
- Metóda `GObject.move`(**int** `dx`, **int** `dy`) posunie všetky body objektu o zadanú vzdialenosť
- Predpokladajme, že sú korektné aj objekty so zmiešanými druhmi bodov – chceme zabrániť len posunu bodov pozadia
- Ako toto vyjadríme v Theme/UML?



6 Zobrazenie do kódu

- Jestvujú už definované postupnosti krokov pre zobrazenia do AspectJ, AspectWerkz (v súčasnosti tzv. @notácia v AspectJ) a Concern Manipulation Environment (CME)
- Základ pretínajúcej témy sa implementuje abstraktným aspektom
- Viazanie (binding) sa definuje v konkrétnom aspekte
- Pretínajúca téma sa tak dá použiť v rôznych viazaniach
- Nasleduje kód pre návrh pretínajúcej témy zo slajdu 9

6.1 Abstraktný aspekt reprezentuje pretínajúcu tému

```
abstract aspect BackgroundStill {
   interface MovingClass {
      boolean isStill();
   }

   abstract pointcut MovingOp(MovingClass o);

   void around(MovingClass o): movingOp(o) {
      if (!o.isStill())
         proceed(o);
   }
}
```

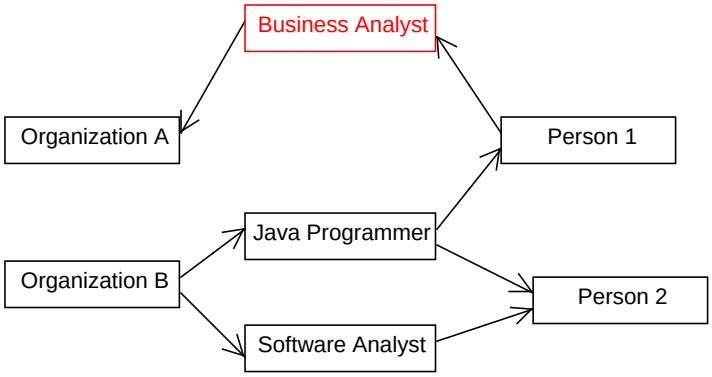
6.2 Konkrétny aspekt definuje viazanie

```
aspect GSysBackgroundStill extends BackgroundStill {
   declare parents: Point implements MovingClass;
   public boolean Point.isStill() {
      return getBackground();
   }
   pointcut movingOp(MovingClass o): target(o) && call(* set*(..));
}
```

7 OOram – aspektovo-orientovaná dekompozícia a kompozícia

- Object-Oriented Role Analysis Method[13]
- Trygve Reenskaug, 1995 (objavil vzor MVC)
- Základná abstrakcia: rola
- Odľahčená metóda (lightweight method): nestanovuje prísne procesy a predpokladá prispôsobenie
- Na základnej úrovni bola potvrdená reverzibilita transfromácie medzi modelom v OOram a Theme/UML[14]

7.1 Príklad: pracovné ponuky

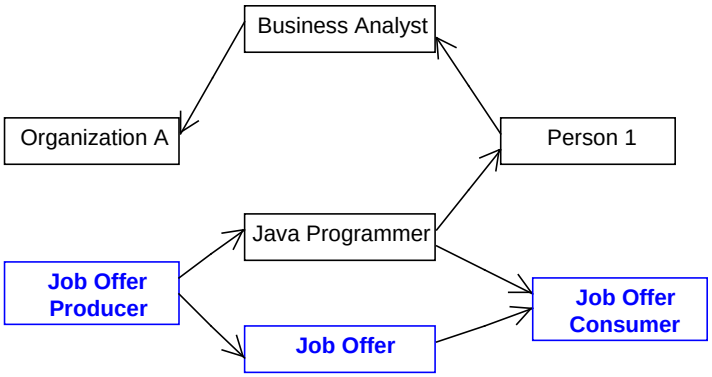


- Jedna z možných konfigurácií objektov
- Aj uchádzač, aj organizácia môžu hrať dve roly: producenta a konzumenta ponuky práce

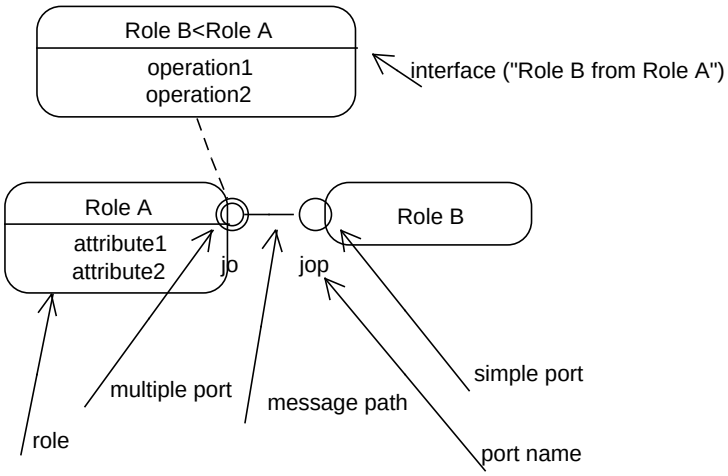
[13] T. Reenskaug. Working with Objects: The OOram Software Engineering Method. Manning, 1995. http://heim.ifi.uio.no/~trygver/1996/book/WorkingWithObjects.pdf

[14] M. Laslop. Aspects and Roles in Software Modeling. Diplomová práca, FIIT STU, 2012.

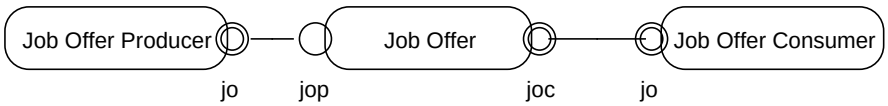
7.2 Od objektov k rolám



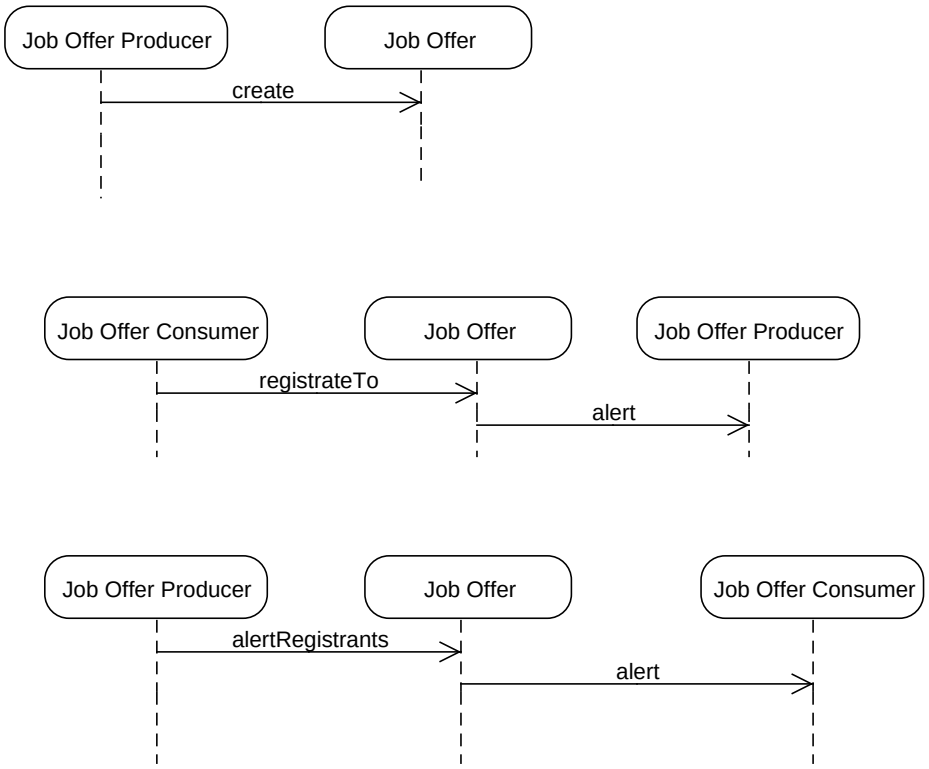
7.3 Základné prvky notácie pohľadu kolaborácie



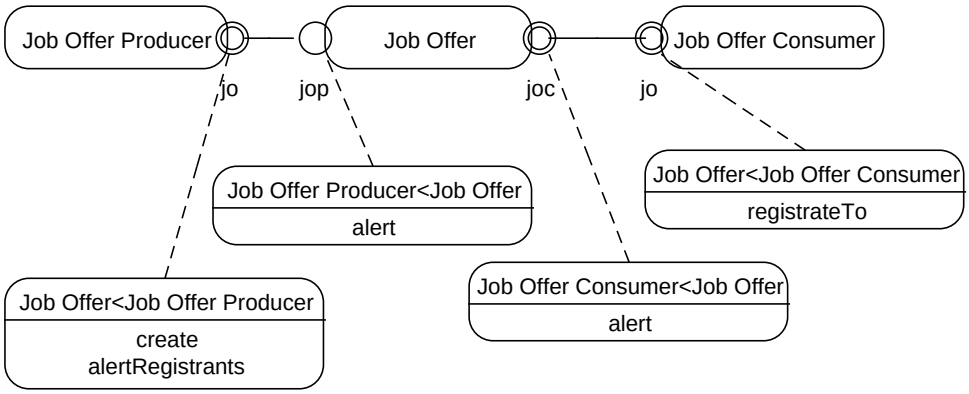
7.4 Pohľad kolaborácie



7.5 Pohľad scenárov



7.6 Pohľad kolaborácie s rozhraniami

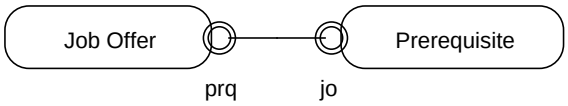


- Rola definuje rozhranie, ktoré má splniť jej klient:

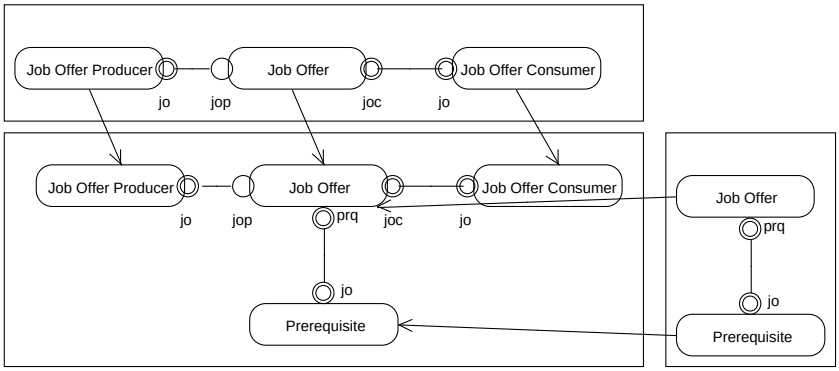
  `Job Offer<Job Offer Producer` = rozhranie, ktoré poskytuje `Job Offer` pre `Job Offer Producer`

7.7 Ďalší model

- Pracovné ponuky môžu byť detailnejšie analyzované vo vlastnom modeli
- Napr. pracovná ponuka môže určovať predpoklady, ktoré uchádzač má spĺňať

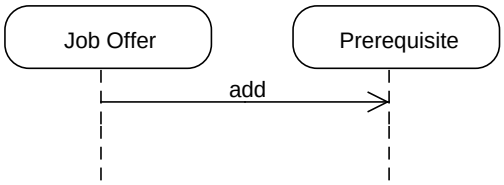


7.8 Kompozícia modelov rolí

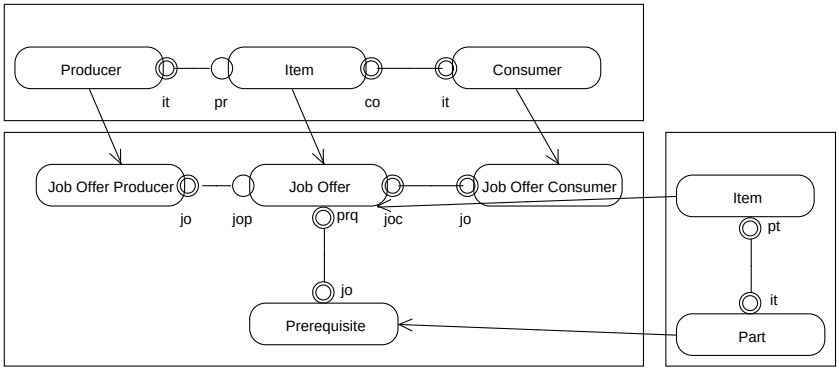


7.9 Kompozícia na úrovní scenárov

- Model pracovnej ponuky dopĺni do integrovaného modelu scenár pridania predpokladu
- Symetrická aspektovo-orientovaná dekompozícia a kompozícia



7.10 Zovšeobecnenie rolí

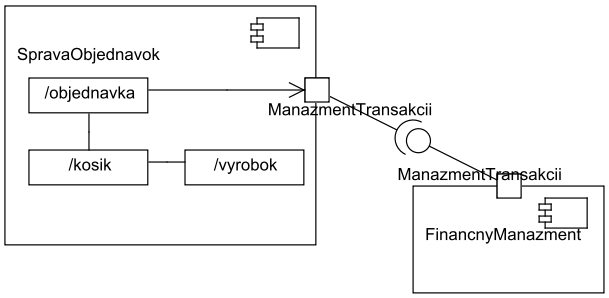


7.11 Vplyv OOram na UML

- Roly v UML
- Diagramy kompozitnej štruktúry
- Porty komponentov – porty rolí
- Cez porty komponentov sa exportujú rozhrania

7.12 Roly v UML

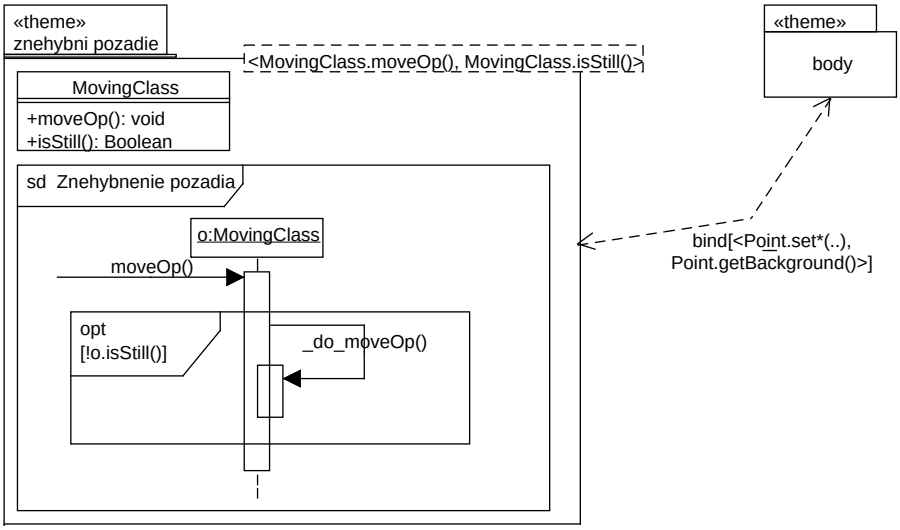
- V diagramoch kompozitnej štruktúry sa nemusíme viazať na presné typy – môžeme tiež modelovať pomocou rolí
- Ako v OOram: dynamická štruktúra najprv – typy budú spresnené po jej preskúmaní



8 Body spájania v Theme

- Vo výraze `bind` sa vymenujú zodpovedajúce operácie, ktoré sa naviažu na parametre šablóny balíka
- Možno uviesť aj viac operácií pre jeden parameter – uvedú sa ako množina (v skupinových zátvorkách)

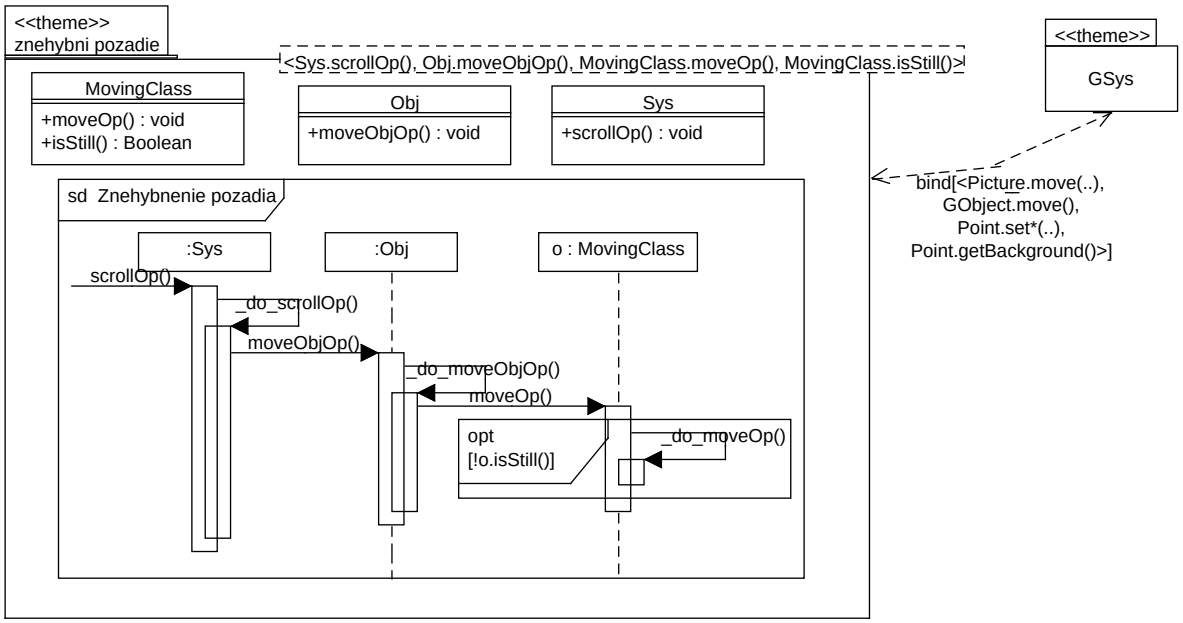
- Možno použiť aj náhradné znaky podobne ako v AspectJ
- Body spájania však určuje téma ako taká – vo všeobecnom zmysle



pointcut `MovingOp`(`Point o`): **target**(`o`) && **call**(∗ **set**∗(..));

8.1 Zachytenie bodov spájania v toku riadenia

- Chceme blokovať posuny bodov pozadia len v rámci skrolovania
- Skrolovanie sa realizuje metódou `Picture.move`(), ale samotné volania `Point.setX`() a `Point.sety`() sa môžu vyskytovať v metódach `GObject.move`() volaných v metóde `Picture.move`()
- Metóda `GObject.move`(**int** `dx`, **int** `dy`) posunie všetky body objektu o zadanú vzdialenosť
- Predpokladajme, že sú korektné aj objekty so zmiešanými druhmi bodov – chceme zabrániť len posunu bodov pozadia
- Ako toto vyjadríme v Theme/UML?



8.2 Bodový prierez toku riadenia v Theme

- V Theme dochádza k prepleteniu špecifikácie bodov spájania a funkcionality videnia
- Funkcionalita videnia je následne len obmedzene zonovupoužiteľná

8.3 Zodpovedajúci bodový prierez v AspectJ

- Uvedený diagram zodpovedá nasledujúcemu bodovému prierezu v AspectJ:

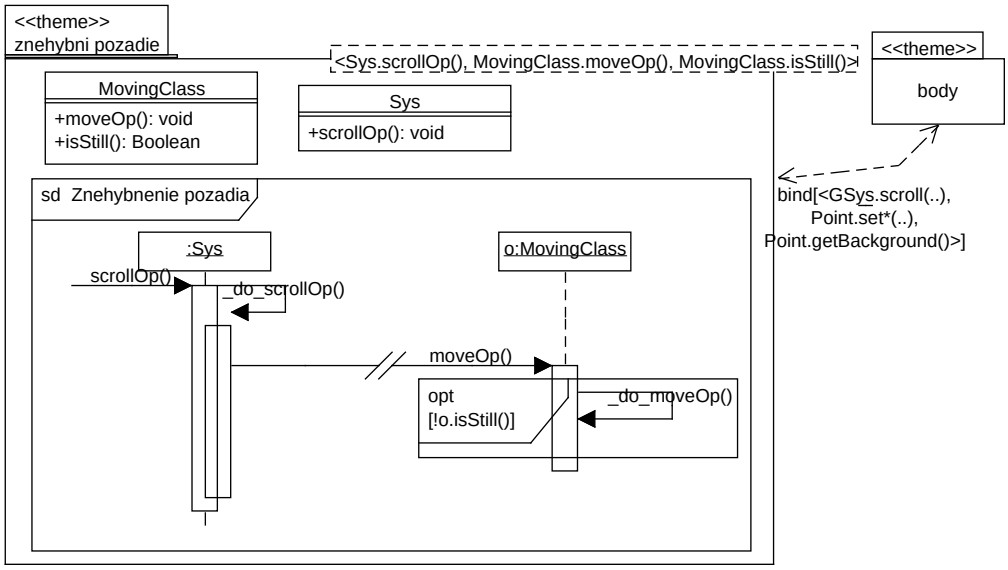
```
pointcut MovingOp(MovingClass obj):
  target(obj) && call(* set*(..)) && cflow(call(* Picture.move(..)));
```

- Tento bodový prierez je však všeobecnejší:
  - v Theme sme síce zachytili volania `MovingClass.moveOp()` v rámci `Sys.scrollOp()`, ale len cez `Obj.moveObjOp()`
  - V AspectJ zachytávame volania operácií posunov bodov do ľubovoľnej hĺbky vnorenia v toku riadenia operácie skrolovania

8.4 Nepriame volanie

- Čo sme vlastne chceli dosiahnuť je *nepriame volanie*
- Nechceme závislosť od nejakej sprostredkovateľskej triedy, o ktorej ani nemusíme vedieť
- Theme/UML však nepodporuje nepriame volania

- Keby Theme/UML podporoval nepriame volania:



9 Join Point Designation Diagrams

- Diagramy určenia bodov spájania – Join Point Designation Diagrams (JPDD)[15] [16]
- Grafické vyjadrenie bodov spájania
- V širšom kontexte predstavujú modely dopytov (query models) – použitie v MDA alebo na špecifikovanie návrhových ohraničení za účelom prevencie chýb v návrhu[17]
- JPDD možno aplikovať aj na diagramy aktivít a stavové diagramy
- Prejdeme len základy notácie JPDD pre diagramy štruktúry a diagramy sekvencií

9.1 Lexikálna korešpondencia

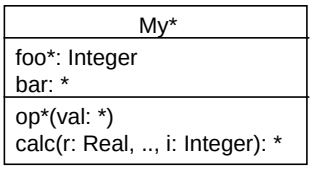
- Používajú sa šablóny názvov (name patterns) podobne ako v AspectJ
- Príklad: výber klasifikátora

[15] `https://www.dawis.wiwi.uni-due.de/forschung/forschungsschwerpunkte/aspektorientierte-softwareentwicklung/join-point-designation-diagrams/`

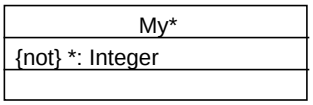
[16] D. Stein. Join Point Designation Diagrams: A Visual Design Notation for Join Point Selections in Aspect-Oriented Software Development. PhD. thesis, Universität Duisburg-Essen. `http://duepublico.uni-duisburg-essen.de/servlets/DocumentServlet?id=24134&lang=en`

[17] V. 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.

– Vyberú sa všetky klasifikátory, ktoré obsahujú uvedené prvky
– Prvky, ktoré nie sú uvedené, nás nezaujímajú

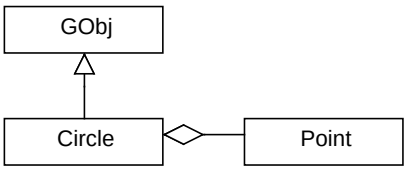


- Môžeme použiť aj negáciu – napr. chceme vybrať triedy bez celočíselných atribútov:

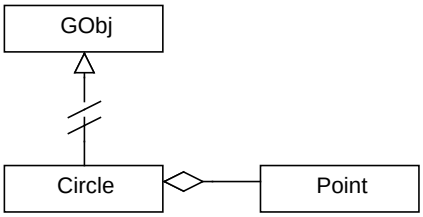


9.2 Asociácie

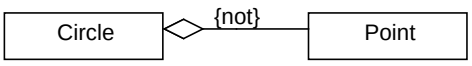
- V JPDD vzťahy, ktoré musia jestvovať, znázorníme v diagrame tried



- Môžeme použiť aj „operátor“ nepriamosti: dve čiary cez hranu, ktorá je medzi nimi prerušená

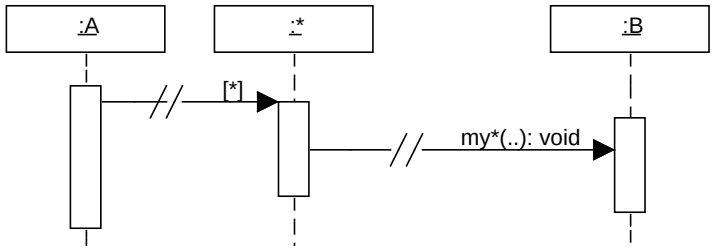


- Negácia sa môže použiť aj pri vzťahoch medzi triedami



9.3 Posielanie správ

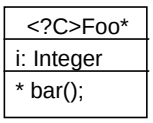
- JPDD môžu zachytiť aj dynamické body spájania – pomocou diagramov sekvencií
- Aj v diagramoch sekvencií môžeme mať nepriame volania



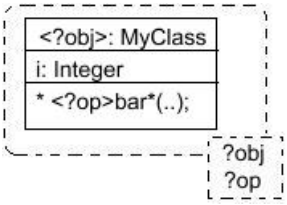
- $[*]$ označuje ľubovoľný tok riadenia
- Interpretácia: hľadáme všetky konfigurácie posielania správ, ktoré zodpovedajú diagramu
- Čo vlastne zachytávame?

9.4 Značenie vybraných prvkov

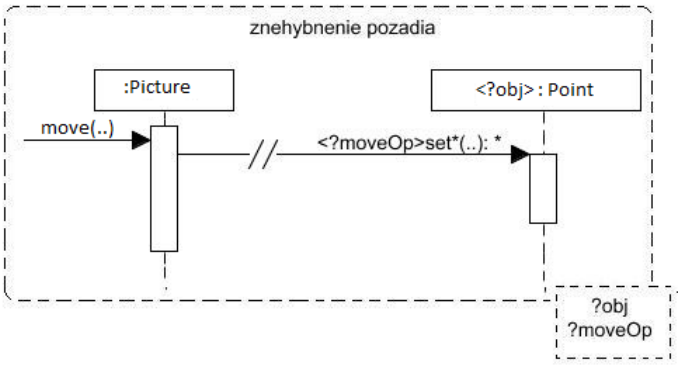
- Prvky, ktoré chceme zachytiť, označíme špeciálnymi identifikátormi
  - Začínajú otáznikom
  - Sú v lomených zátvorkách
  - Uvádzajú sa pri šablónach názvov prvkov, na ktoré sa vzťahujú



- Tieto identifikátory neovplyvňujú šablóny názvov
- Možno ich použiť aj na odkazy vnútri JPDD
- Tieto identifikátory deklarujeme v zozname parametrov JPDD
- JPDD je vymedzený oválom
- Parametre sa uvádzajú podobne ako pri šablónach v UML, ale v spodnej časti



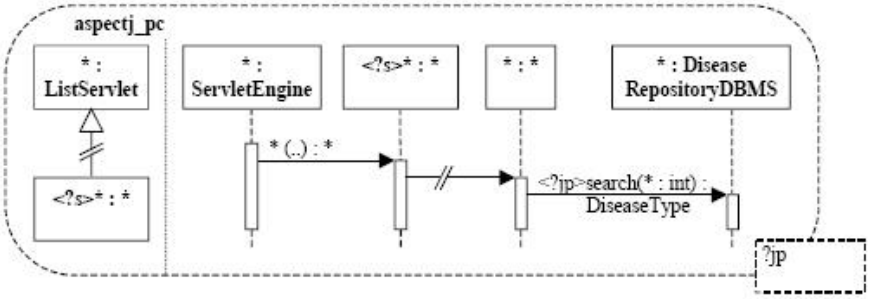
9.5 Príklad JPDD




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

9.6 Ďalší príklad JPDD

Ďalší príklad JPDD od D. Steina et al.[18]




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

9.7 Nástroje na podporu JPDD

- Informácie o dvoch prototypoch na webovej stránke venovanej výskumu JPDD:[19]

[18] D. Stein et al. Query Models. In Proc. of 7th International Conference on the Unified Modeling Language (UML 2004)–The Language and Its Applications, Lisbon, Portugal, October 2004.

[19] `https://www.dawis.wiwi.uni-due.de/forschung/forschungsschwerpunkte/aspektorientierte-softwareentwicklung/join-point-designation-diagrams/`

- Nástroj na tvorbu JPDD v prostredí Eclipse: M4JPDD[20] [21]
- Nástroj na generovanie bodových prierezov z JPDD (v prostredí Squeak)[22]

10 Theme a JPDD

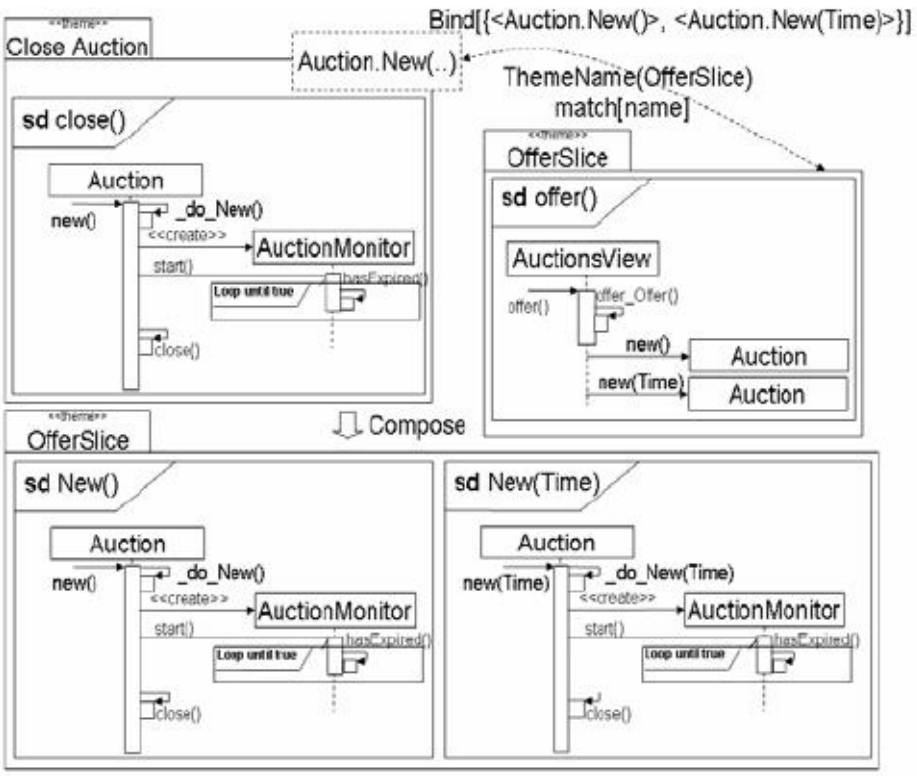
- Zaujímavé v Theme: modelovanie štruktúry, základné témy a ich spájanie
- Theme však má obmedzené možnosti určenia bodov spájania navyše spojené s definíciou pretínajúcej funkcionality
- Toto je evidentné pri práci s dynamickým kontextom bodov spájania
- Dá sa použiť technika JPDD na zachytenie bodov spájania, a Theme na definovanie pretínajúcich záležitostí[23]
- Nasledujúci príklad je z citovaného článku

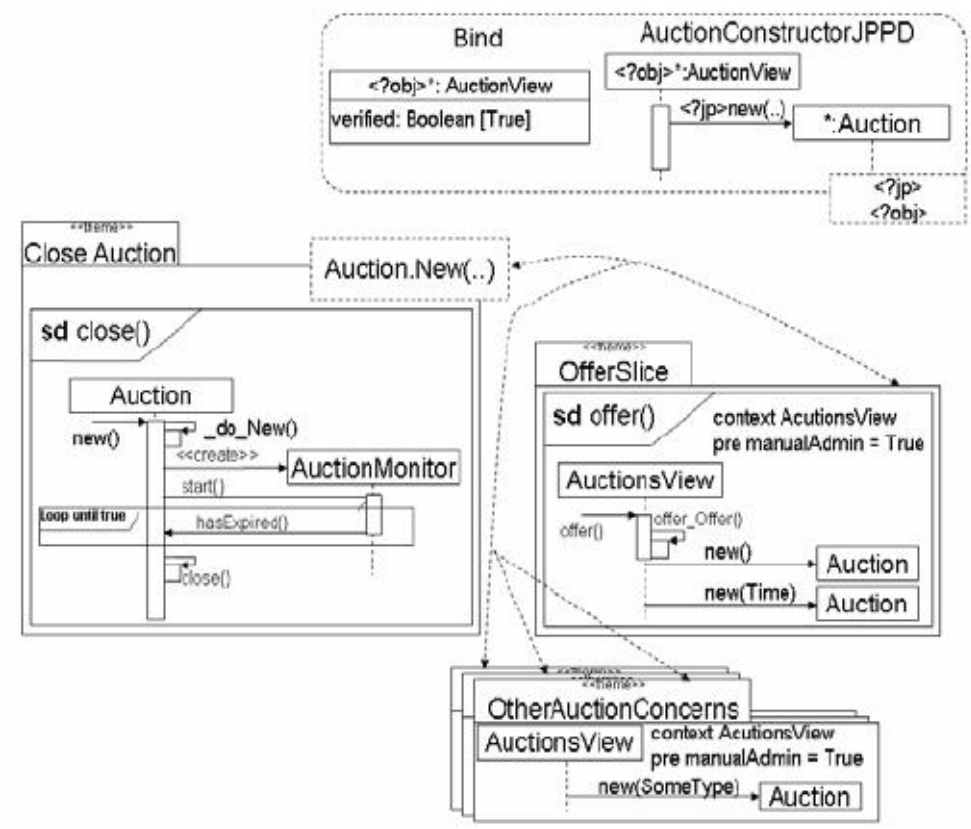
[20] Stein, D., Hanenberg, S.: M4JPDD—Tool-Support for Modeling Join Point Designation Diagrams. Demo at AOSD 2008. Brussels, Belgium, 2008.

[21] J. Bartelheimer. Implementation of a Modeling Tool for Join Point Designation Diagrams Based on the Eclipse Frameworks EMF, GEF, and the UML2-Ecore-Meta-Model. BSc. thesis, Universität Duisburg-Essen, 2006.

[22] S. Hanenberg, D. Stein, and R. Unland. From aspect-oriented design to aspect-oriented programs: tool-supported translation of JPDDs into code. In Proc. of 6th Int. Conf. on Aspect-Oriented Software Development, AOSD 2007, Vancouver, Canada, 2007. ACM.

[23] A. Jackson and S. Clarke. Towards the Integration of Theme/UML and JPDDs. In 8th International Workshop on Aspect-Oriented Modeling, held in conjunction with 5th International Conference on Aspect-Oriented Software Development (AOSD'06), March 2006, Bonn, Germany. `http://dawis2.icb.uni-due.de/events/AOM_AOSD2006/papers.shtml`


Bind[{<Auction.New()>, <Auction.New(Time)>}]
Close Auction
Auction.New(..)
ThemeName(OfferSlice)
match[name]
sd close()
Auction
new()
_do_New()
<<create>>
AuctionMonitor
start()
Loop until true
hasExpired()
close()
OfferSlice
sd offer()
AuctionsView
offer()
offer_Offer()
new()
Auction
new(Time)
Auction
Compose
OfferSlice
sd New()
Auction
new()
_do_New()
<<create>>
AuctionMonitor
start()
Loop until true
hasExpired()
close()
sd New(Time)
Auction
new(Time)
_do_New(Time)
<<create>>
AuctionMonitor
start()
Loop until true
hasExpired()
close()



11 Sumarizácia

- Aspekty možno identifikovať už v požiadavkach a to aj v asymetrickej dekompozícii
- Na návrh aspektov sú potrebné jednotky schopné zachytiť príslušnú štruktúru a správanie a definovať ich kompozíciu: asymetricky a symetricky
- Aspekty možno chápať ako spoluprácu rolí
- Definovanie pretínania nad grafickými modelmi možno robiť pomocou grafických dopytovacích jazykov