Prednáška 6: Metóda OOram

Metódy a prostriedky špecifikácie 2013/14

Valentino Vranić

Ústav informatiky a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave

29. október 2013

Obsah prednášky

Úvod

Triedne-orientované modelovanie?

- Kde sú objekty v UML diagramoch?
- Je *objektovo*-orientované modelovanie mylne pomenované?
- Je známe, že štruktúra má byť odvodená zo správania
- Ale aj tak – neurčujeme štruktúru predsa príliš skoro?
- Príliš skoro stanovujeme pevné hierarchie – v prirodzených systémoch zriedkavé
- Hierarchie potrebujeme pre poriadok, ale až v implementácii

Objekty najprv!

- Prečo bolo potrebné OOP a vôbec OO vývoj softvéru?
- Potrebujeme decentralizáciu – systém ako spolupráca objektov, z ktorých je každý autonómny a stará sa o vlastné veci
- Kedy možno nájsť objekty? Počas behu programu.
- Treba modelovať najprv dôležité konfigurácie objektov počas behu programu
- Ako modelovať objekty a neviazť sa na typy?
- Objekty v nich hrajú rôzne *roly*

Modelovanie a implementácia

- UML je preťažený detailami implementačnej povahy
- Programovacie jazyky sú aj tak lepšie v ich vyjadrovaní
- Potrebujeme výrazné oddelenie záležitostí
- Potrebujeme oddelené parciálne modely, v ktorých objekty hrajú rôzne roly
- Potrebujeme kompozíciu týchto pohľadov, v ktorej z rolí vzniknú objekty

Charakteristika metódy OOram

Metóda OOram

- Object-Oriented Role Analysis Method
- Trygve Reenskaug, 1995 (vynašiel vzor MVC)
- Základná abstrakcia: rola
- Odľahčená metóda (lightweight method): nestanovuje prísne procesy a predpokladá prispôsobenie

Proces

- Základný rámec metódy:
  1. System user model
  2. System requirement model
  3. System design model
  4. System implementation
  5. System of objects
- Proces je oportunistický, iteratívny a inkrementálny
- Prebieha zhora nadol, zdola nahor a zvnútra von

OOram model

- S modelom pracujeme prostredníctvom *pohľadov* (view) – nie len v metóde OOram
- Pohľady možno uplatniť pre jednu alebo viac *perspektív* (perspective)[1]
- Perspektíva má význam výhľadu (vtáčia perspektíva/výhľad) – z akého priblíženia sa pozeráme na model

[1] Aj keď sú perspektíva a pohľad synonymá, v OOram majú rozdielny význam.

Perspektívy – rozsah

1. Environment perspective – rozsah systému; sleduje sa interakcia systému s rolami prostredia
2. External perspective – interakcia rolí; nepriamo určuje štruktúru rolí (nimi definovaných objektov)
3. Internal perspective – vnútro roly; implementácia roly

Pohľady (1)

1. Area of concern view – opis textom
2. Stimulus-response view – opis tabuľkou (Stimulus message, Response messages, Comments)
3. Role list view – zoznam rolí s ich opisom a atribútmi (text)
4. Semantic view – diagram, ktorý opisuje význam rolí a ich vzťahov (notácia založená na entitno-relačných diagramoch)
5. Collaboration view – vzory (usporiadania) rolí a toky správ medzi nimi

Pohľady (2)

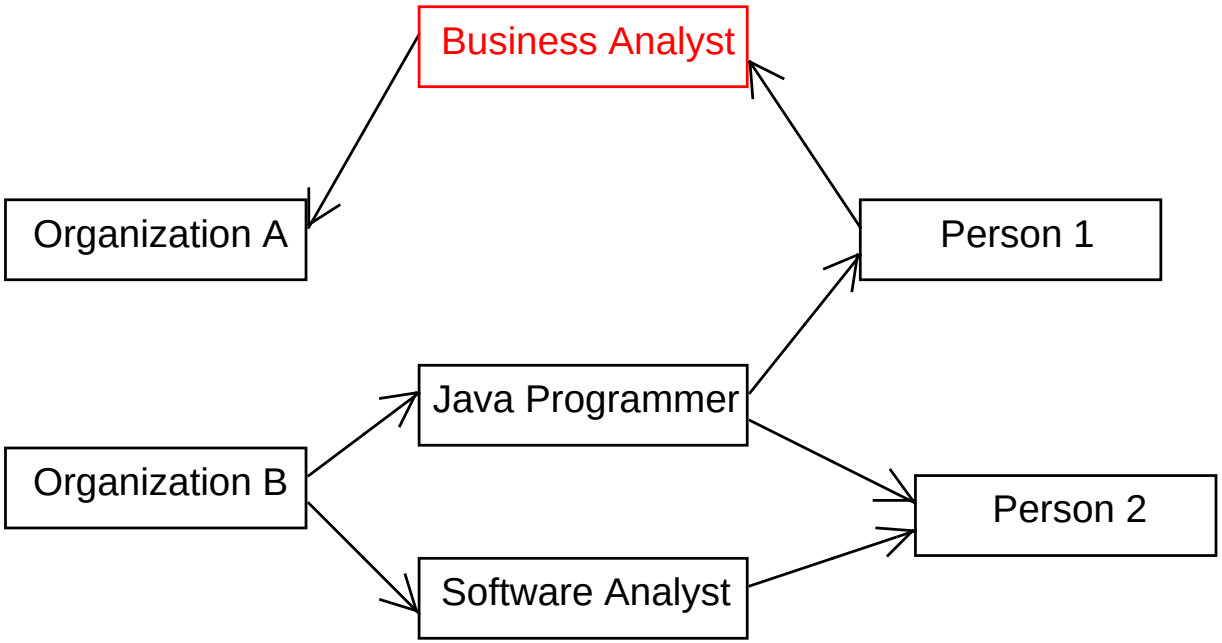
6. Interface view – opis rozhraní a správ, ktoré definujú (text)
7. Scenario view – príklady postupností správ medzi rolami (ako diagramy sekvencií)
8. Process view – toky údajov medzi rolami a akcie, ktoré roly realizujú (notácia blízka diagramom aktivít)
9. State diagram view – stavový diagram, ktorý opisuje stavy danej roly
10. Method specification view – volania, ktoré sa realizujú v rámci danej metódy (rovnaká notácia ako pri scenároch – ako diagramy sekvencií)

Pohľady (3)

- Pozrieme pomocou príkladov bližšie základy:
  - collaboration view
  - scenario view
  - kompozície
  - implementácie

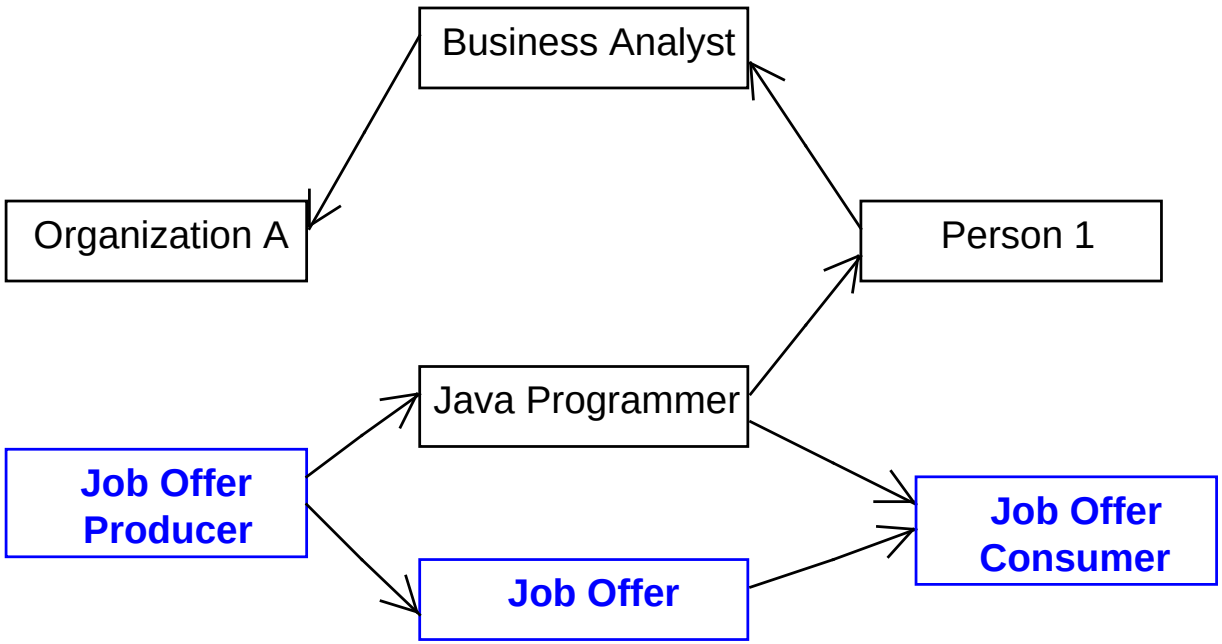
Kolaborácia rolí, scenáre a kompozícia

Príklad: pracovné ponuky

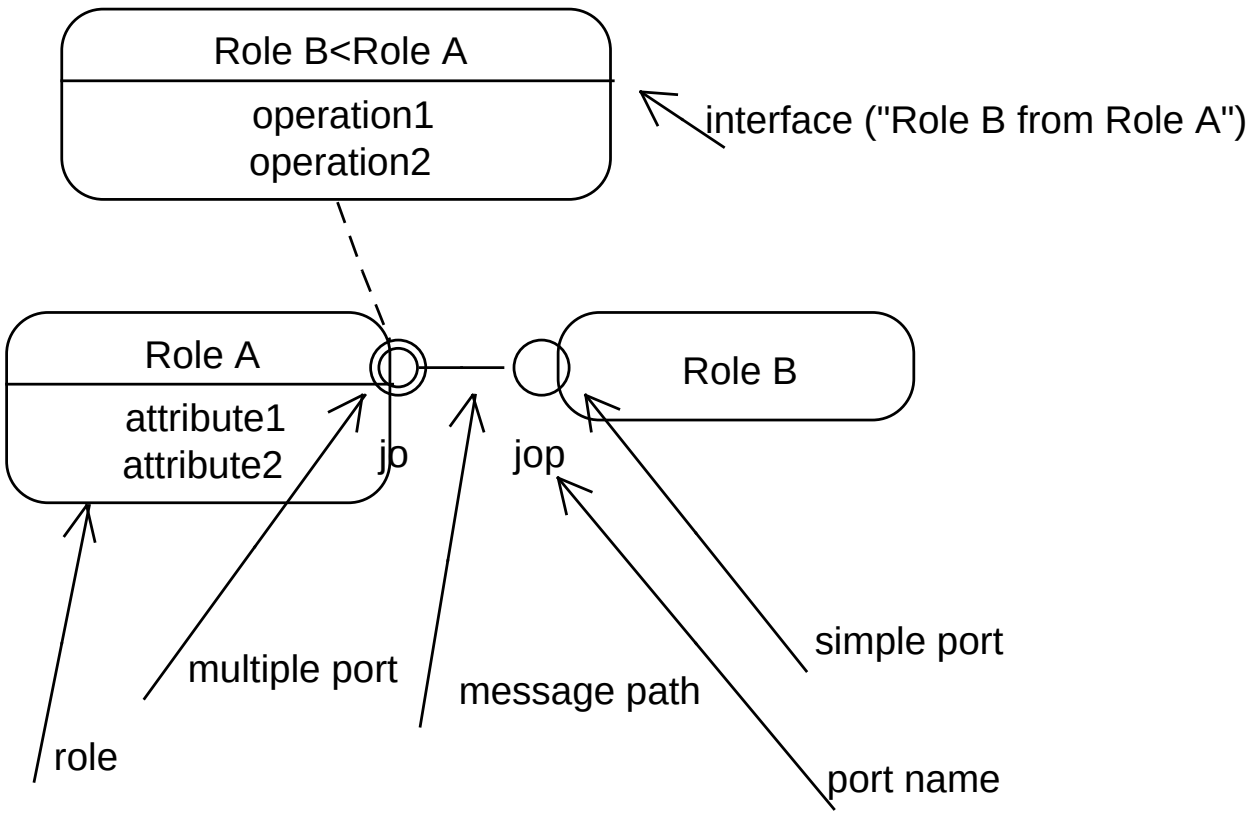


- Jedna z možných konfigurácií objektov
- Aj uchádzač, aj organizácia môžu hrať dve roly: producenta a konzumenta ponuky práce

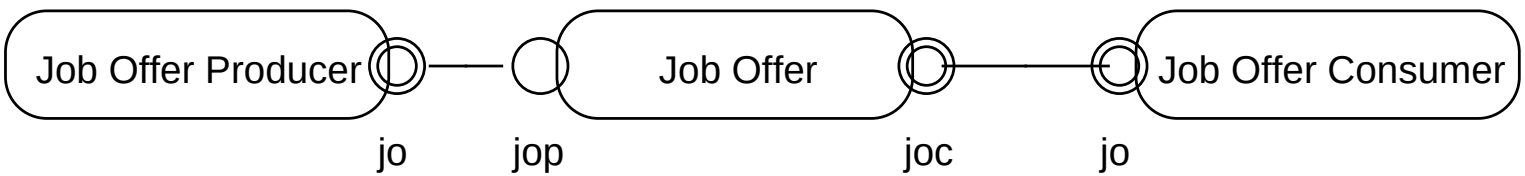
Od objektov k rolám



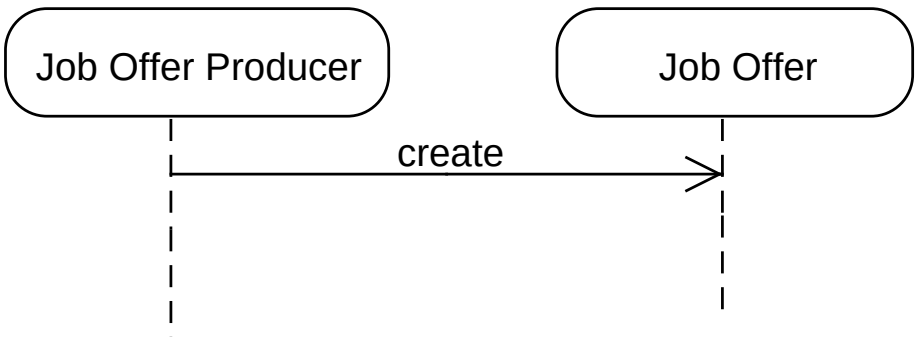
Základné prvky notácie pohľadu kolaborácie



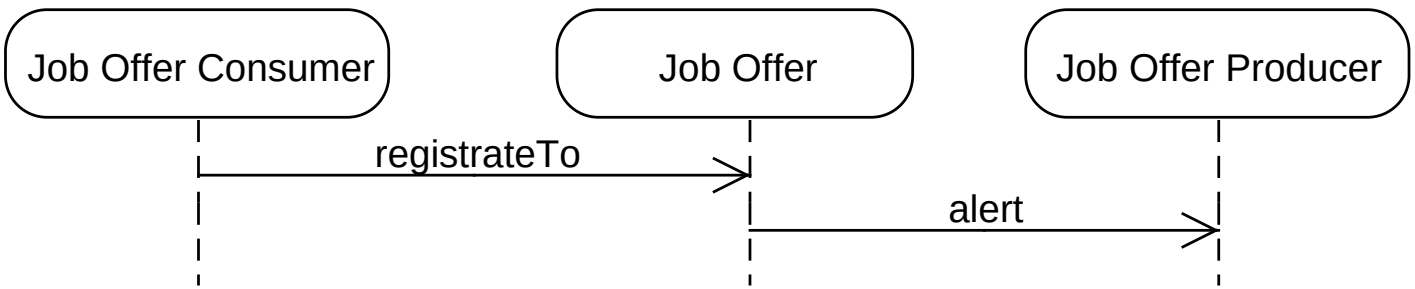
Pohľad kolaborácie



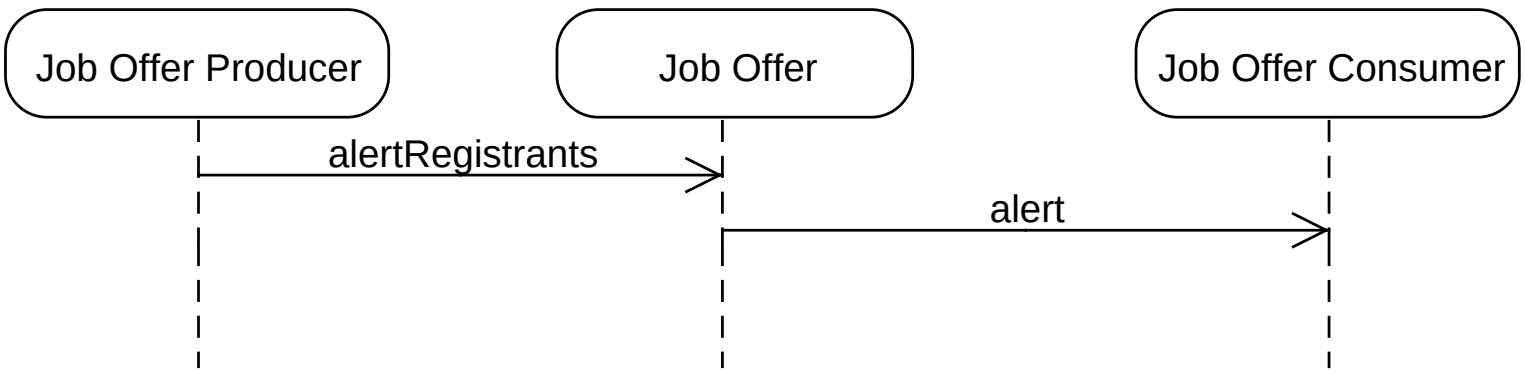
Pohľad scenárov (1)



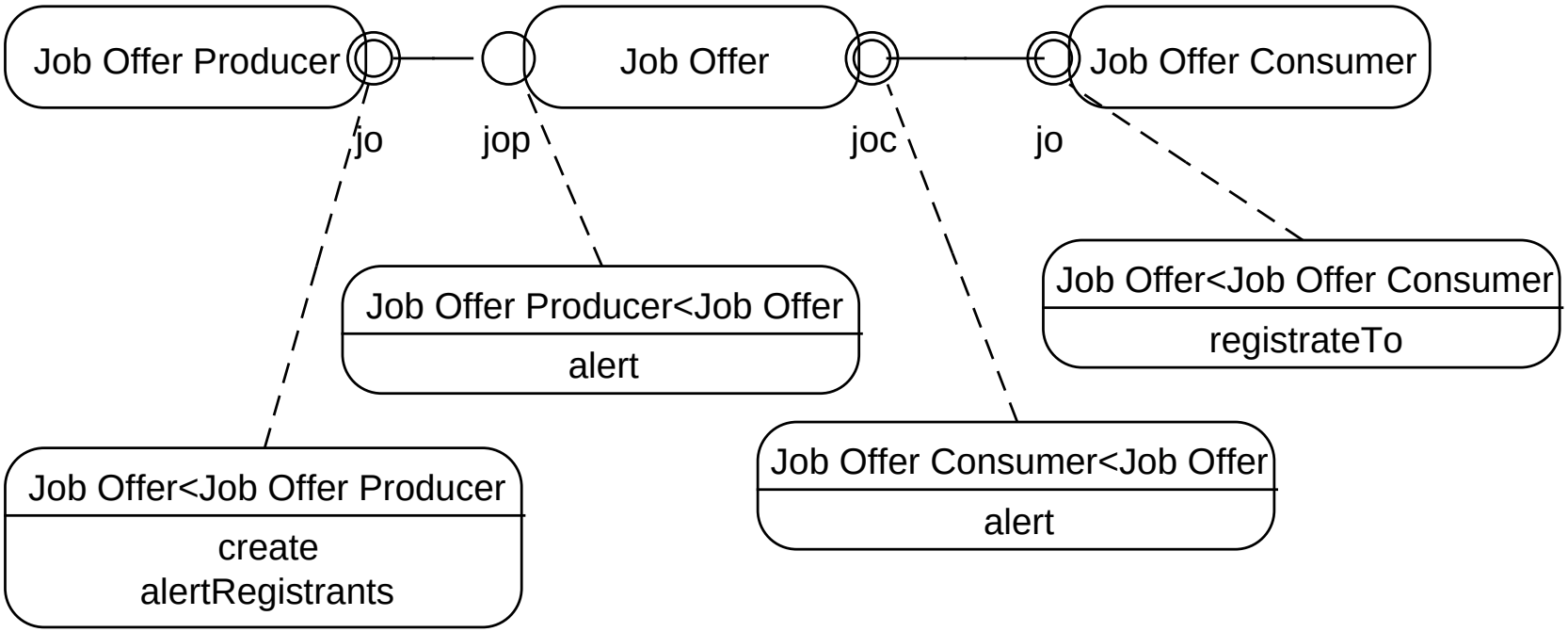
Pohľad scenárov (2)



Pohľad scenárov (3)

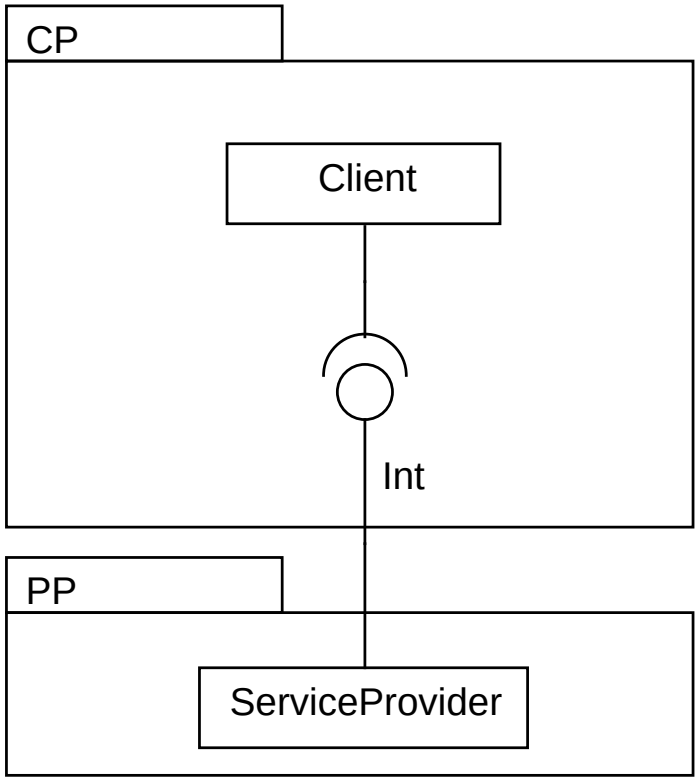


Pohľad kolaborácie s rozhraniami



- Rola definuje rozhranie, ktoré má splniť jej klient: `Job Offer<Job Offer Producer` = rozhranie, ktoré poskytuje `Job Offer` pre `Job Offer Producer`

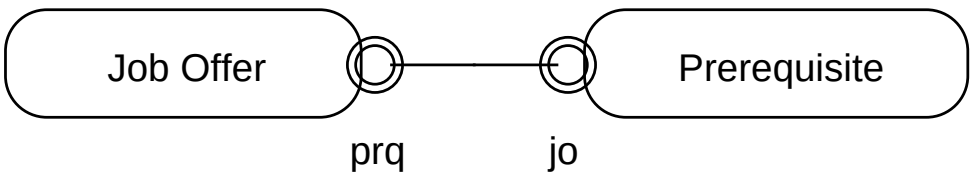
Digresia: definícia požadovaného rozhrania na strane klienta



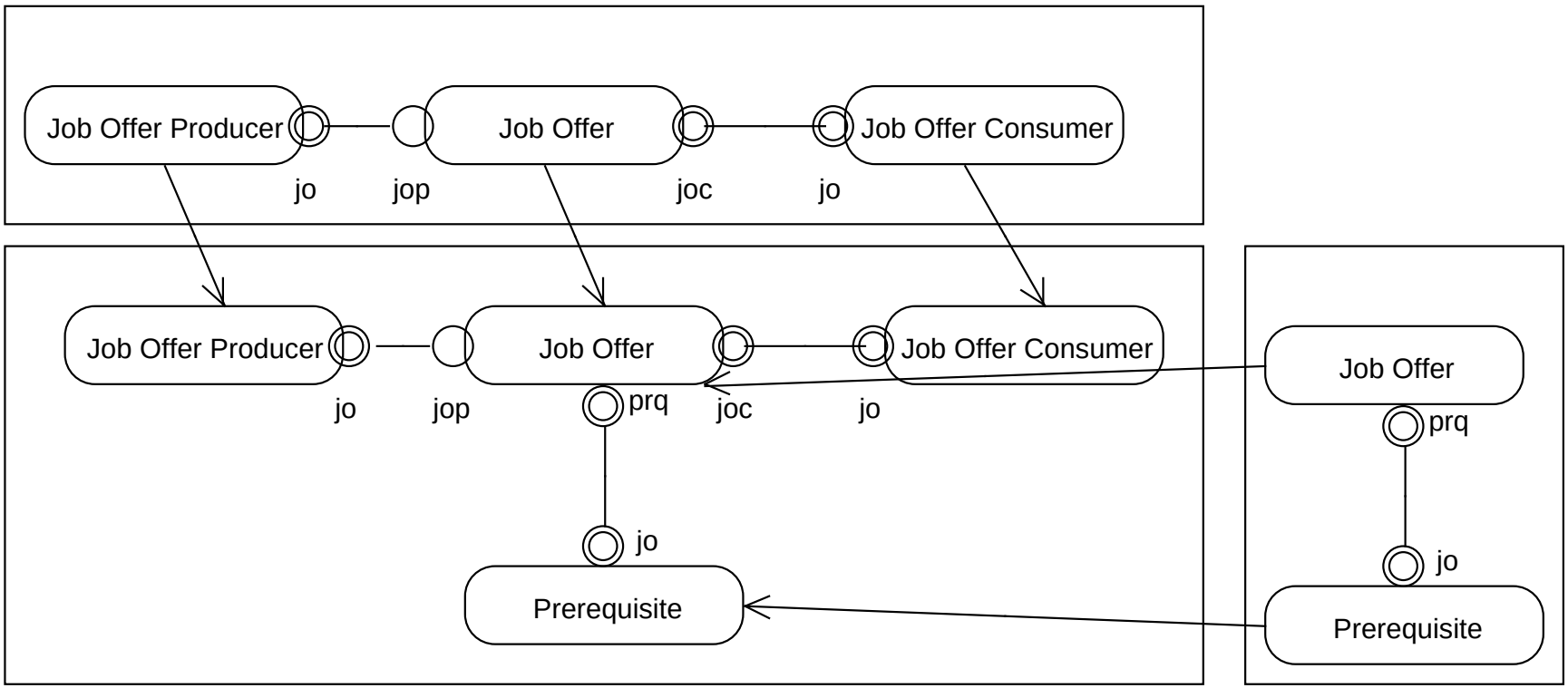

```
package CP;
class Client { ... }
interface Int { ... }
\\ klient predpisuje rozhranie

_____________________________

package PP;
\\ rozhranie implementuje
\\ poskytovateľ z iného balíka
class ServiceProvider implements Int { ... }
```

Ďalší model

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

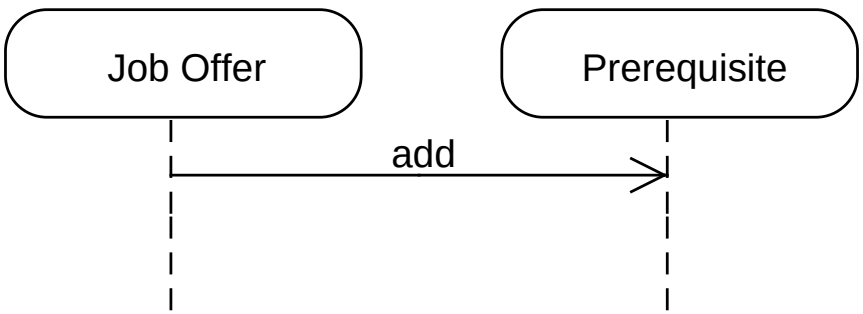


Kompozícia modelov rolí

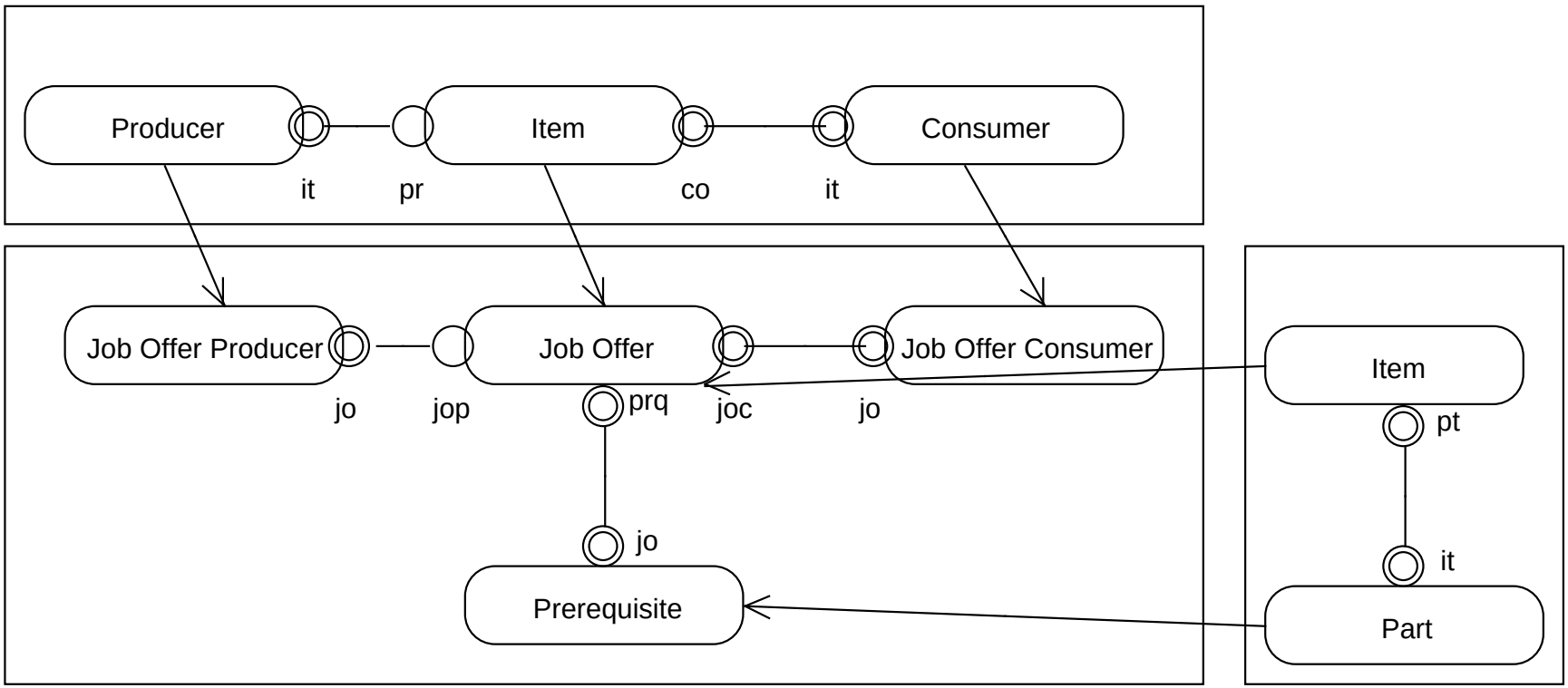


Kompozícia na úrovní scenára

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



Zovšeobecnenie rolí

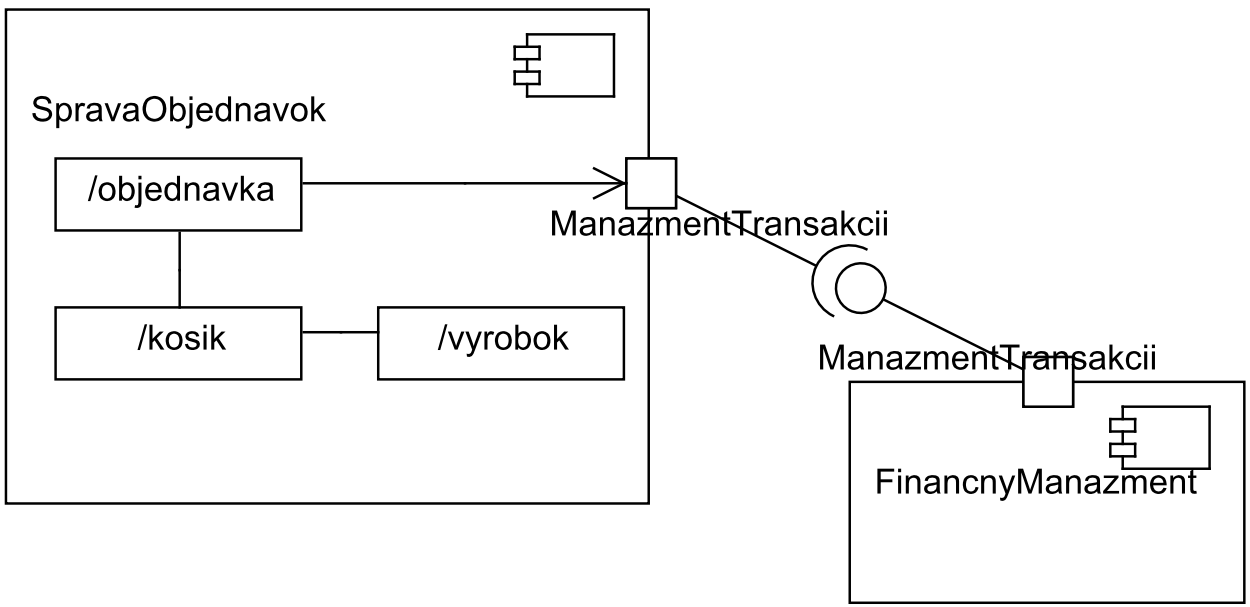


Vplyv OOram na UML

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

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



Implementácia OOram modelu

Implementácia OOram modelu

OOram	Smalltalk	C++	Java
Role Model	-	-	-
Role	Object	Object	Object
Object Specification, Type	Class	Class	Class
Port	Variable	Pointer data member	Reference
Interface	Protocol	Abstract class or protocol class	Interface
Message	Message	Function call	Method call
Method	Method	Member function	Method
Derived model	Subclass	Derived class	Subclass
Base model	Superclass	Base class	Superclass

Implementácia – rozhrania

```
interface JobOfferAlert {
    void alertParticipants();
}

interface JobOfferProducer {
    void alertRegistrnats();
}

interface JobOfferConsumer {
    void registrateTo();
}
```

- Metóda `create()` z rozhrania `Job Offer Producer` bude implementovaná konštruktormi tried

Implementácia – triedy

- Hierarchia dedenia tried – teda dedenia *štruktúry* – nemusí sledovať roly: Company a Society napr. nemajú spoločnú nadtriedu

```
class Company implements JobOfferProducer, JobOfferConsumer { ... }

class PrivateCompany extends Company { ... }

class PublicCompany extends Company { ... }

class Society implements JobOfferProducer, JobOfferConsumer { ... }

class Person implements JobOfferProducer, JobOfferConsumer { ... }

class JobOffer implements JobOfferAlert { ... }
```

- Porty budú implementované ako atribúty
- Viacnásobné porty musia byť implementované pomocou zoskupení

Sumarizácia

Sumarizácia

- OOram – OO metóda vývoja softvéru založená na abstrakcii roly
- Nepredstavuje hlavný prúd, ale mala vplyv na UML
- Dá sa pre ňu použiť významná časť jazyka UML
- Čo ďalej: prístup Domain, Context and Interaction (DCI) – James O. Coplien a Trygve Reenskaug[2]

[2] J. O. Coplien and G. Bjørnvig. Lean Architecture: for Agile Software Development. Wiley, 2010.

Čítanie

- Trygve Reenskaug. Working with Objects: The OOram Software Engineering Method. Manning, 1995. `http://heim.ifi.uio.no/~trygver/1996/book/WorkingWithObjects.pdf` – aspoň časť 1.2
- Øredev – Developer Conference, 2010, Malmö, Sweden:
  - Jim Coplien – DCI: Practical Tips and Lessons for Nerds `http://vimeo.com/8235574`
  - Trygve Reenskaug – DCI: Re-thinking the foundations of object orientation and of programming `http://vimeo.com/8235394`