

Prednáška 4: Modelovanie štruktúry v UML

Metódy a prostriedky špecifikácie 2012/13

Valentino Vranič

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

16. október 2012

Obsah prednášky

- 1 Úvod
- 2 Diagram tried
- 3 Diagram balíkov
- 4 Diagram komponentov
- 5 Diagram kompozitnej štruktúry

Úvod

Štruktúra softvérového systému

- Čo je štruktúra softvérového systému
- Na základe čoho ju modelovať
- Ako ju vyjadriť v jazyku UML

Štruktúra softvérového systému

- Štruktúra softvérového systému: časti, z ktorých sa softvérový systém skladá
- Program je kód, ale aj jeho vykonávanie
- Štruktúra je najpresnejšie vyjadrená kódom
- Ale aj program vo vykonávaní má štruktúru
- Conwayov zákon:

Organizácie, ktoré navrhujú systémy, sú ohraničené tak, aby vyrábali systémy, ktoré sú kópiami komunikačných štruktúr týchto organizácií.

Conway, M. E. "How do Committees Invent?", *Datamation*, (14) 4, April 1968.¹

¹http://egov.blogs.com/eaglossary/2004/06/conways_law.html

Štruktúra má byť odvodená od správania

- Správanie je zachytené v prípadoch použitia
- Štruktúru možno spoznať priamo v tokoch prípadov použitia
- Vhodným medzikrokom je pokus o vyjadrenie správania technikou, ktorá vynucuje exponovanie štruktúry – diagramy sekvencií
- Diagramy aktivít (ako alternatíva) to neumožňujú v dostatočnej miere – zachytávajú (ak vôbec) len údajové objekty

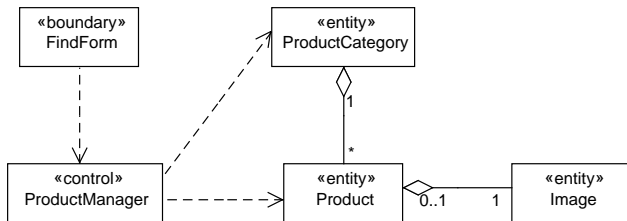
Diagramy štruktúry v UML

- Diagram tried – class diagram
- Diagram objektov – object diagram
- Diagram kompozitnej štruktúry – composite structure diagram
- Diagram rozloženia – deployment diagram
- Diagram komponentov – component diagram
- Diagram balíkov – package diagram

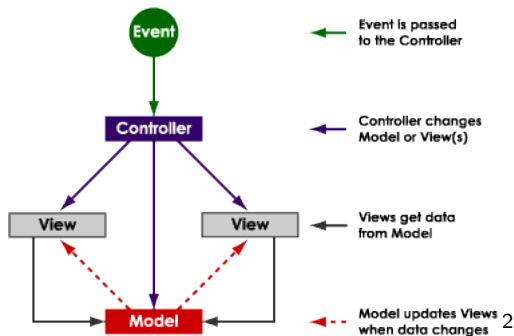
Unified Process stereotypy tried (1)

- Stereotypy umožňujú rozširovať jazyk UML o nové prvky – odvodené od jestvujúcich
- Boundary – triedy rozhrania – používateľské rozhranie (typicky formuláre GUI), ale aj technické rozhranie
- Control – riadiace prvky; triedy s metódami, ktoré implementujú kľúčovú funkcionálnosť
- Entity – údajové prvky; entity, s ktorými sa v systéme pracuje – väčšinou jednoduchšie metódy (zobrazené do databázových tabuliek: object-relational mapping)

Unified Process stereotypes tried (2)



Vzor Model-View-Controller (MVC)



²<http://ootips.org/mvc-pattern.html>

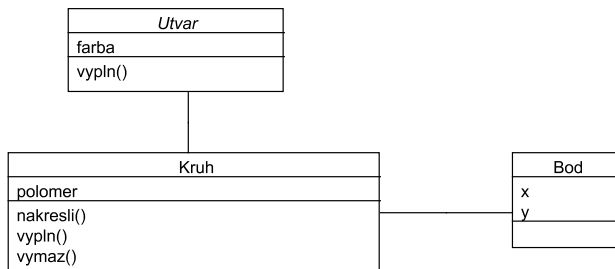
Diagram tried

Diagram tried

- Znázorňuje (predovšetkým) triedy a vzťahy medzi nimi
- Môže obsahovať aj rozhrania, balíky a objekty
- Vzťahy medzi triedami:
 - Asociácia – všeobecný vzťah
 - Agregácia
 - hodnotou (kompozícia)
 - referenciou
 - Generalizácia/špecializácia – dedenie
 - Závislosť

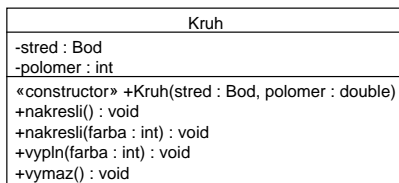
Asociácia

- Všeobecná asociácia je zvlášť užitočná keď začíname skúmať vzťahy medzi triedami
- Vtedy ešte nevieme ich presný význam



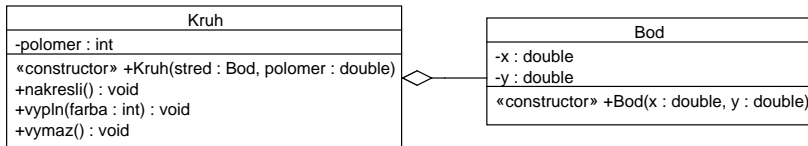
Viditeľnosť atribútov a operácií

- Viditeľnosť atribútov a operácií – ako modifikátory prístupu v Jave:
 - + public
 - # protected
 - - private
 - ~ package
- Detaily triedy Kruh



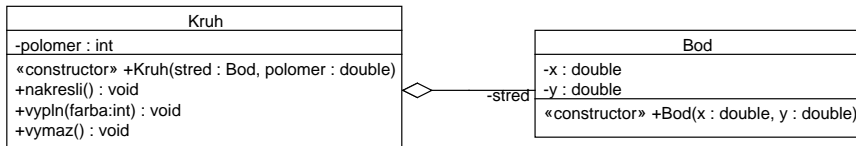
Agregácia

- Agregácia
 - plný kosoštvorec – hodnotou
 - prázdny kosoštvorec – referenciou
- Presný význam:
 - plný kosoštvorec – agregácia, pri ktorej agregujúci objekt nesie zodpovednosť za existenciu a uloženie agregovaných objektov (composite)
 - prázdny kosoštvorec – zdieľaná agregácia, pri ktorej aj iné objekty môžu agregovať rovnaký objekt (shared)



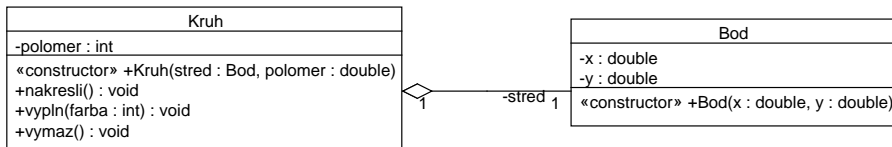
Asociačná rola

- Asociačná rola (association role) – rola triedy vo vzťahu k inej triede
 - Inštancia triedy **Bod** v inštancii triedy **Kruh** hrá rolu stredu



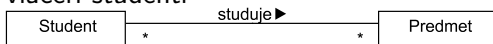
Násobnosť

- Násobnosť (*multiplicity*) vzťahu – koľko inštancií triedy pripadá na výskyt inštancie inej triedy
 - Kruh má práve jeden stred, a ten stred patrí len jemu
- Násobnosť sa často označuje ako kardinalita

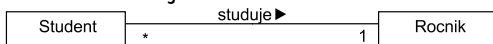


Príklady násobnosti

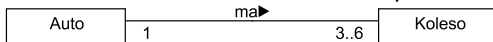
- Študent študuje viac predmetov, a ten istý predmet študujú viaceri študenti



- Študent študuje práve v jednom ročníku, a v tom istom ročníku študujú viaceri študenti

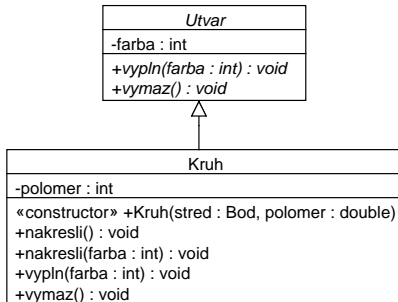


- Auto má 3 až 6 kolies, a koleso patrí len jednému autu



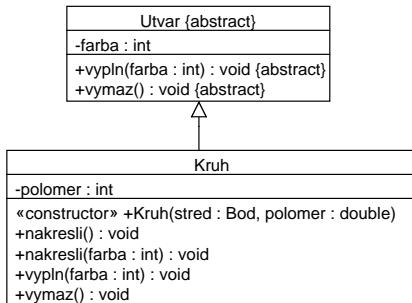
Generalizácia/špecializácia

- Generalizácia/špecializácia – dedenie
 - Trieda Kruh je špecializáciou triedy Utvar
- Trieda Utvar je abstraktná – názov kurzívou



Značenie abstraktných tried a operácií

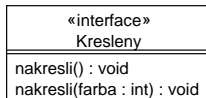
- Pre zvýšenie čitateľnosti možno použiť označenie {abstract}



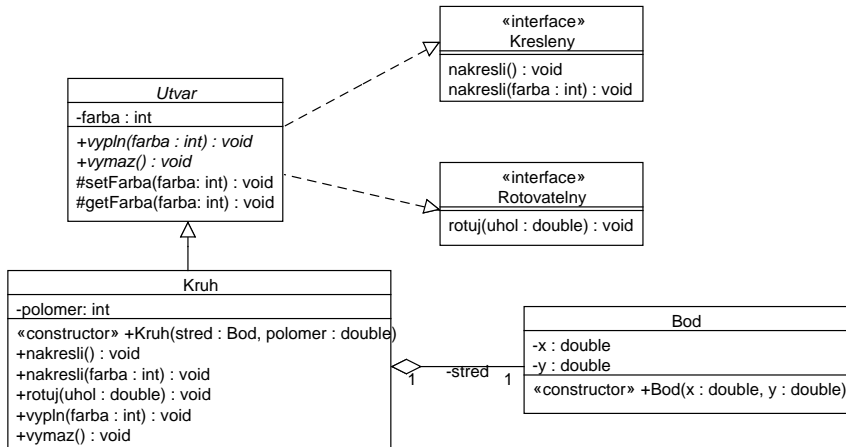
Realizácia rozhrania (1)

- Rozhrania – stereotyp «interface»
- Trieda Utvar realizuje (implementuje) rozhranie Kresleny
- Tento vzťah tiež predstavuje formu dedenia – dedenie správania
- Rozhrania v UML nemôžu mať atribúty (narozdiel napr. od Javy)
 - Uvedenie atribútu v rozhraní v UML znamená, že ho rozhranie predpisuje – tak ako metódu
 - Triedy, ktoré také rozhranie realizujú, musia predpísané atribúty implementovať

Realizácia rozhrania (2)

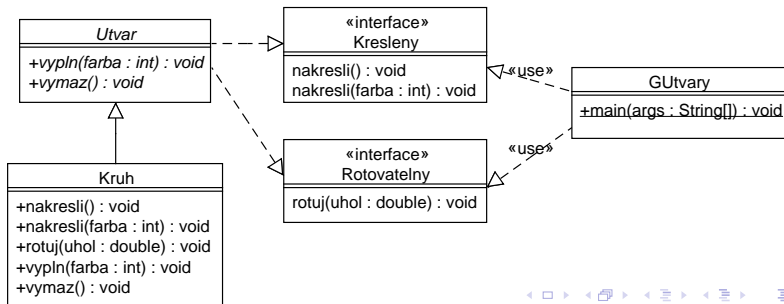


Detailný diagram tried



Použitie rozhrania a väzba závislosti

- Použitie rozhrania predstavuje väzbu závislosti – niekedy sa používa stereotyp «use»
- Pomocou tejto väzby sa dá vyjadriť hocijaká závislosť
- Závislý je prvok, z ktorého vychádza šípka – klient (client)
- Zmeny klienta nemajú vplyv na prvok, ku ktorému šípka smeruje – poskytovateľ (supplier)

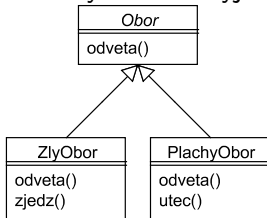


Obsah diagramu tried

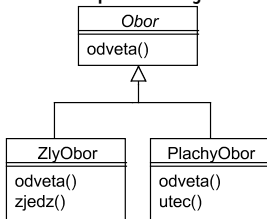
- Diagram tried nemusí obsahovať všetky details
- V jednom diagrame nemusia byť všetky triedy
- Nemusíme znázorniť všetky vzťahy medzi znázornenými triedami
- Každý diagram má niešť istý základný odkaz

Hierarchia dedenia

- Niekedy chceme vyjadriť len hierarchiu dedenia



- Možno použiť aj združenú šípku



Príklad s grafickými útvarmi (1)

```
class Bod {  
    private double x, y;  
    public Bod(double x, double y) { . . . }  
}
```

```
interface Kresleny {  
    void nakresli();  
    void nakresli(int farba);  
}
```

```
interface Rotovatelny {  
    void rotuj(double uhol);  
}  
. . .
```

Príklad s grafickými útvarmi (2)

```
...
abstract class Utvar implements Kresleny, Rotovatelny {
    private int farba;
    public void vypln(int farba) { ... }
    protected void setFarba(int farba) { ... }
    protected int getFarba() { ... }
}

class Kruh extends Utvar {
    private Bod c;
    private double r;
    Kruh(Bod c, double r) { ... }
    public void nakresli() { ... }
    public void nakresli(int farba) { ... }
    public void vypln(int farba) { ... }
    public void rotuj(double uhol) { ... }
}
```

Vhniezdené triedy

- UML umožňuje vyjadriť len statické vhniezdenie tried



- Na vyjadrenie anonymných tried v Java (druh vnútorných tried, teda dynamické vhniezdenie) R. C. Martin navrhuje použiť stereotyp «anonymous»³, ale statickosť to nezmení

³R. C. Martin. UML for Java Programmers. Prentice Hall, 2003.

Diagram objektov

- Ďalší štruktúrálny pohľad – špeciálny prípad diagramu tried
- Znázorňuje vzťahy medzi inštanciami tried v určitom okamihu vykonávania programu
- Môže obsahovať hodnoty atribútov

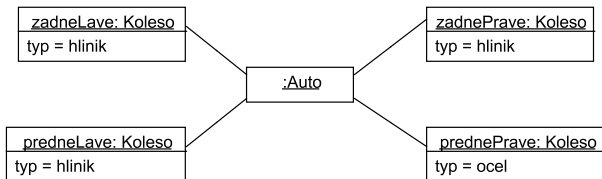
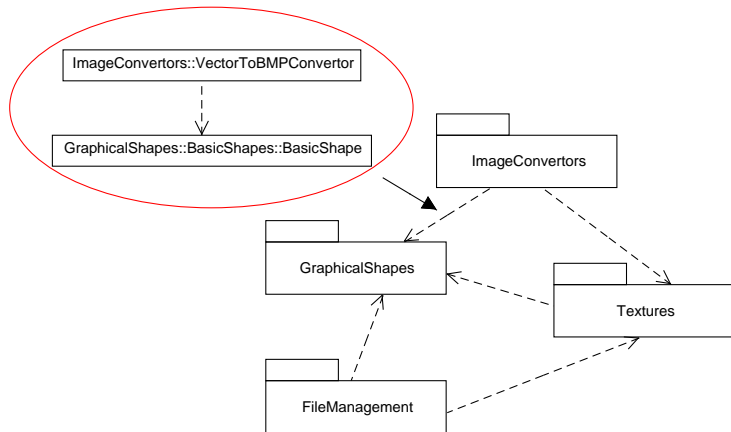


Diagram balíkov

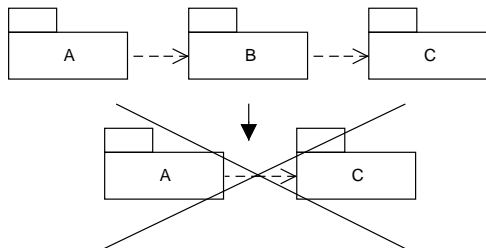
Balíky

- Package diagram
- Potreba za väčšími štruktúrnymi jednotkami
- Každý balík predstavuje priestor názvov (namespace)
- Kvalifikované názvy (::)
- Použitie:
 - vyjadrenie hierarchie štruktúry
 - vyjadrenie závislosti medzi štruktúrnymi jednotkami

Závislosti medzi balíkmi



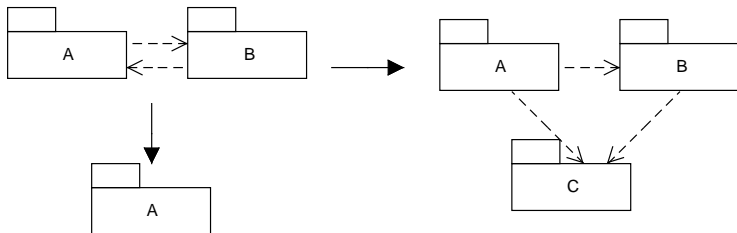
Závislosti nie sú tranzitívne



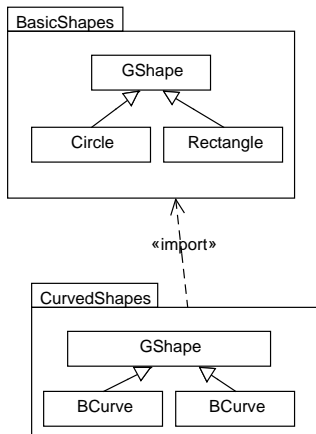
- Dôsledkom netranzitívnosti závislosti je to, že ak sa rozhranie balíka C zmení, pravdepodobne sa bude musieť zmeniť aj jadro balíka B, ale balík A sa nebude musieť meniť

Cirkulárne závislosti

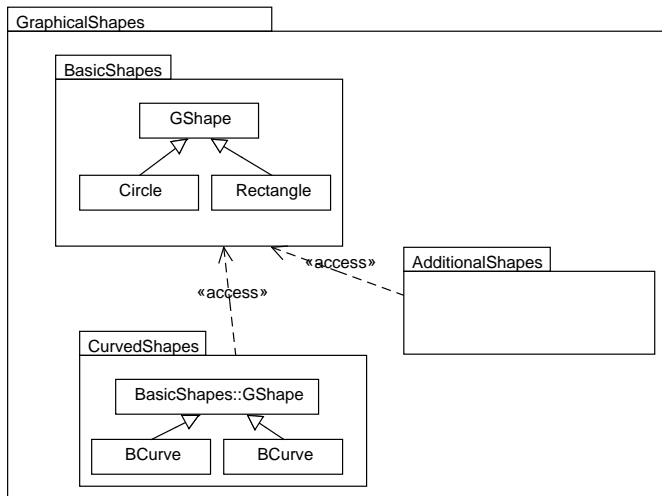
- Cirkulárne závislosti vyriešime reštrukturalizáciou balíkov
- Na analytickej úrovni nemusia prekážať



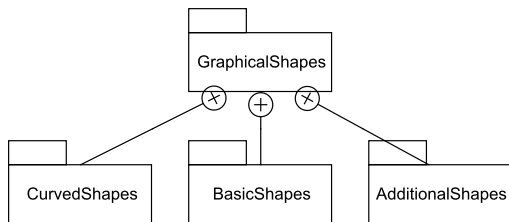
Import



Vhniezdenie a prístup



Vhniezdenie vyjadrené väzbou



Viditeľnosť obsahu balíka

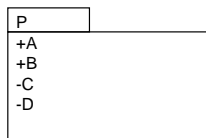
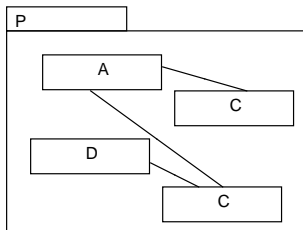
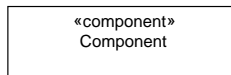
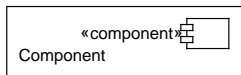
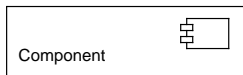


Diagram komponentov

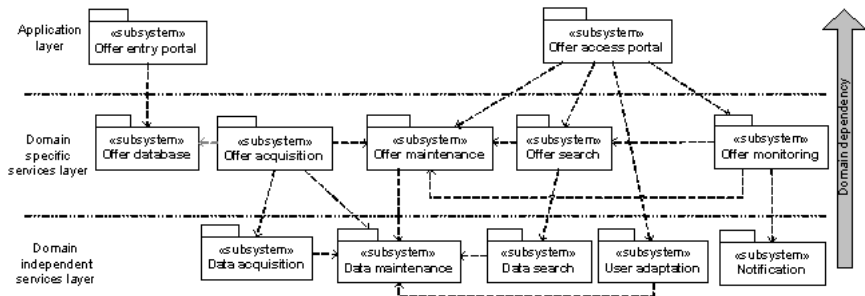
Komponenty

- Komponent podľa špecifikácie UML: *a modular unit with well-defined interfaces that is replaceable within its environment*
- Komponent poskytuje a požaduje rozhrania (provided/required interfaces), inak je zapuzdrený
- Môže predstavovať fyzickú alebo logickú jednotku

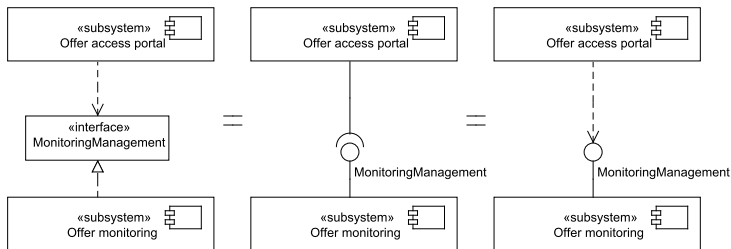


Podsystemy

- V UML 1.x sa podsystemy modelovali balíkmi – v UML 2 sa na to používajú komponenty

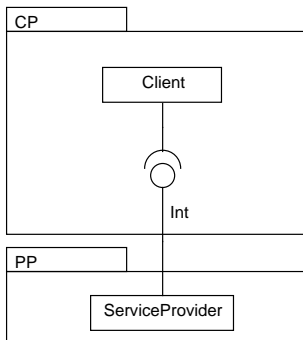


Realizácia a použitie rozhraní



- Takéto závislosti sa niekedy uvádzajú so stereotypom «use»
- Rovnakú schému možno použiť aj pri triedach

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 { ... }
```

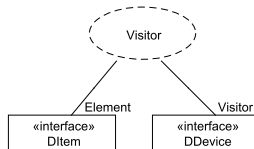
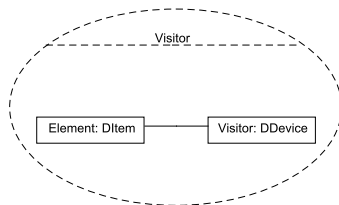
Diagram kompozitnej štruktúry

Kompozitná štruktúra

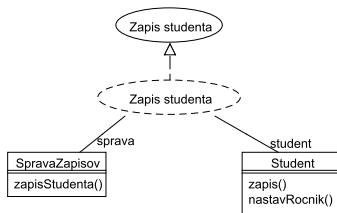
- Kompozitná štruktúra: štruktúra v zmysle kompozície prepojených prvkov, ktoré predstavujú inštancie v čase vykonávania
- Doteraz sme používali lexikálne vnhiezdenie – balík v balíku, triedy v balíku a pod.

Kolaborácia

- Kolaborácia: štruktúra spolupracujúcich (kolaborujúcich) prvkov (rolí), ktoré spoločne zabezpečujú určité správanie
- Takto sa napr. dajú vyjadriť návrhové vzory



Kolaborácia ako realizácia prípadu použitia

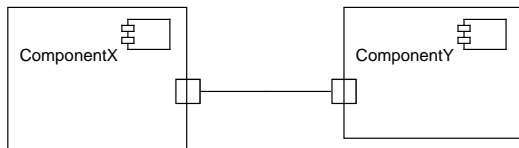


Štruktúra komponentov

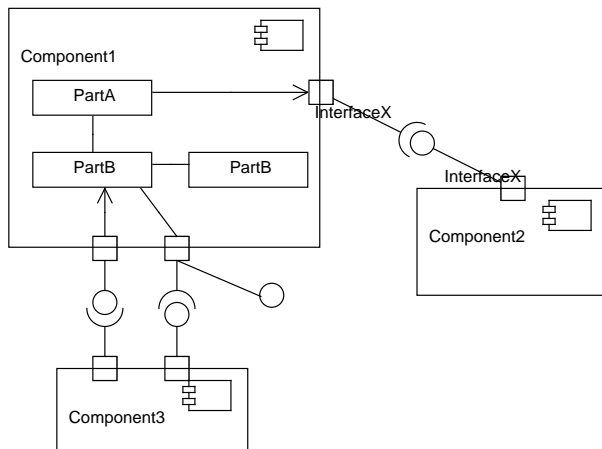
- Chceme vyjadriť časti, z ktorých pozostáva komponent, ako sú prepojené a ako a ktoré sú exponované
- Dá sa použiť aj keď sme ešte nestanovili statickú štruktúru (diagram tried)⁴
- To je v duchu odvodu štruktúry zo správania
- Prvky:
 - part – property
 - port
 - connector – delegate/assembly
 - provided/required interface

⁴ [http://rodin.cs.ncl.ac.uk/Publications/On UML's Composite Structure Diagram.pdf](http://rodin.cs.ncl.ac.uk/Publications/On%20UML's%20Composite%20Structure%20Diagram.pdf)

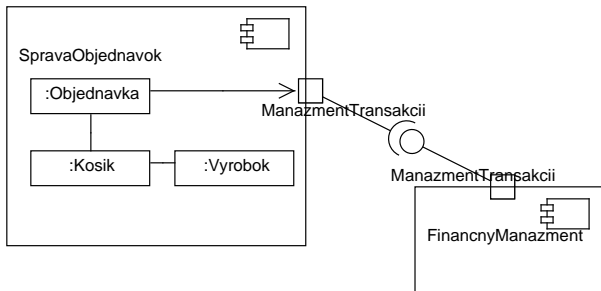
Spojenie komponentov – assembly



Spojenie komponentov – delegate

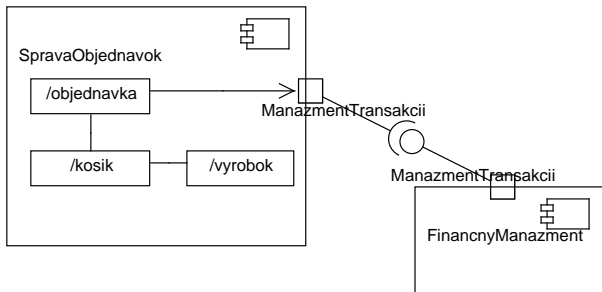


Príklad



Príklad – roly

- Nemusíme sa viazať na presné typy – môžeme modelovať pomocou rolí
- Dynamická štruktúra najprv – typy budú spresnené po jej preskúmaní



Sumarizácia

Sumarizácia

- Diagramy na modelovanie štruktúry:
 - diagram tried
 - diagram objektov
 - diagram balíkov
 - diagram komponentov
 - diagram kompozitnej štruktúry
- Nebol zahrnutý diagram rozloženia (deployment diagram)
- Kolaborácie
- Roly

Čítanie

- Martin Fowler. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley, 3rd edition, 2003.
- Jim Arlow and Ila Neustadt. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design. Addison-Wesley, 2nd edition, 2005.
- Agile Modeling Home Page, <http://www.agilemodeling.com/>
- Špecifikácia UML, http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML → Superstructure specification