SWI041:
Návrh

Z analytického konceptuálního modelu, kde je popsáno CO, musíme navrhnout JAK se to udělá

Nejprve trochu kontroly

Stav projektů

Kroky návrhu

- návrh architektury systému
- návrh uživatelského vzhledu
- návrh komponent
- návrh komunikace mezi komponentami
- návrh způsobu integrace komponent a testování celku

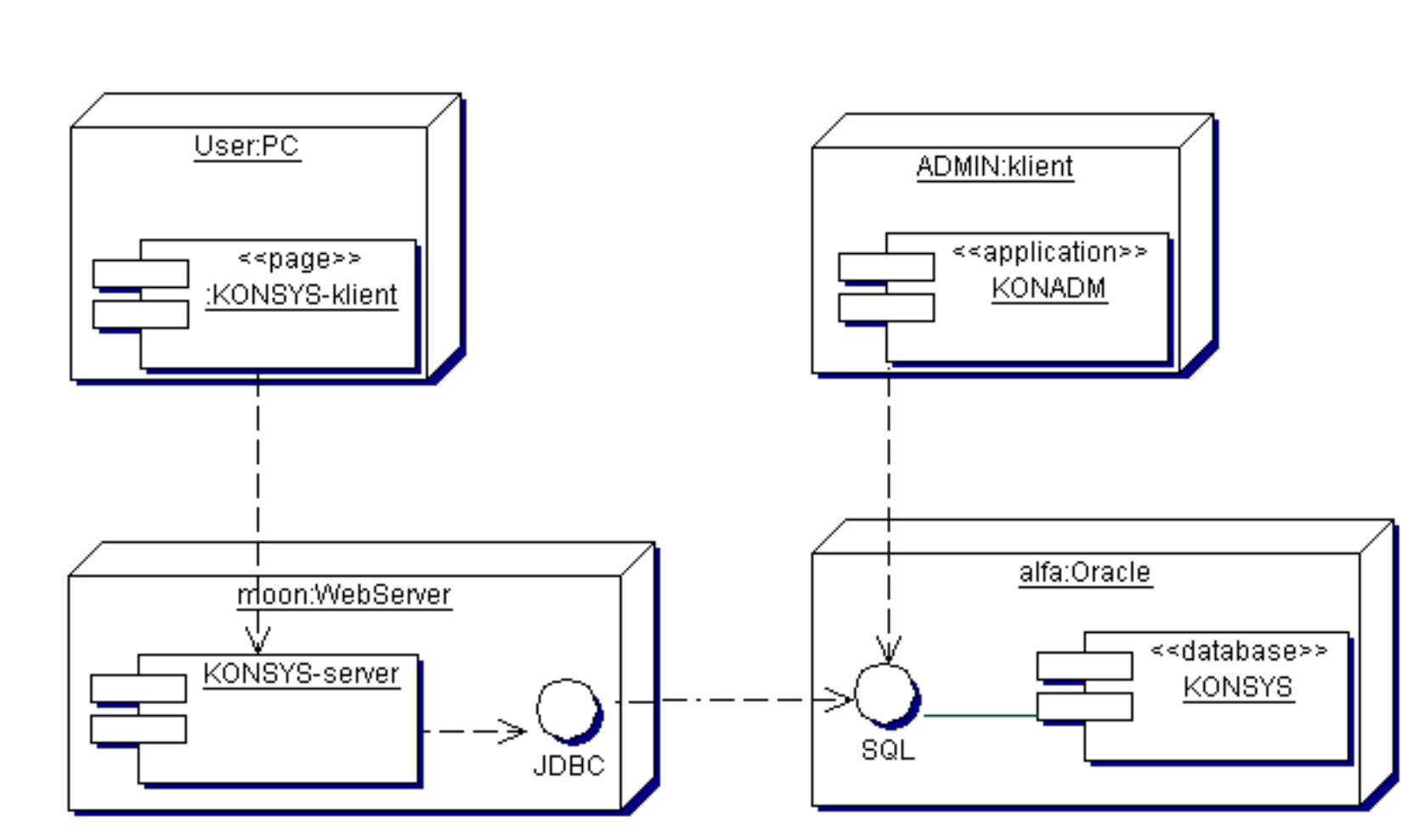
Základní technologická rozhodnutí ve fázi návrhu

- architektura systému
- datové zdroje, datové paměti, přístupové mechanismy k nim
- distribuce programových modulů, komunikační mechanismy
- typy a formy výstupů
- uživatelské rozhraní
- vývojové prostředí

Výstupní dokumenty návrhu

- Architektura systému (HW,SW)
- Popis implementace dat (logický datový model)
- Popis komponent (modulů)
- Projektová dokumentace návrhu

Příklad návrhu architektury



SWI041:
Návrh (datový model)

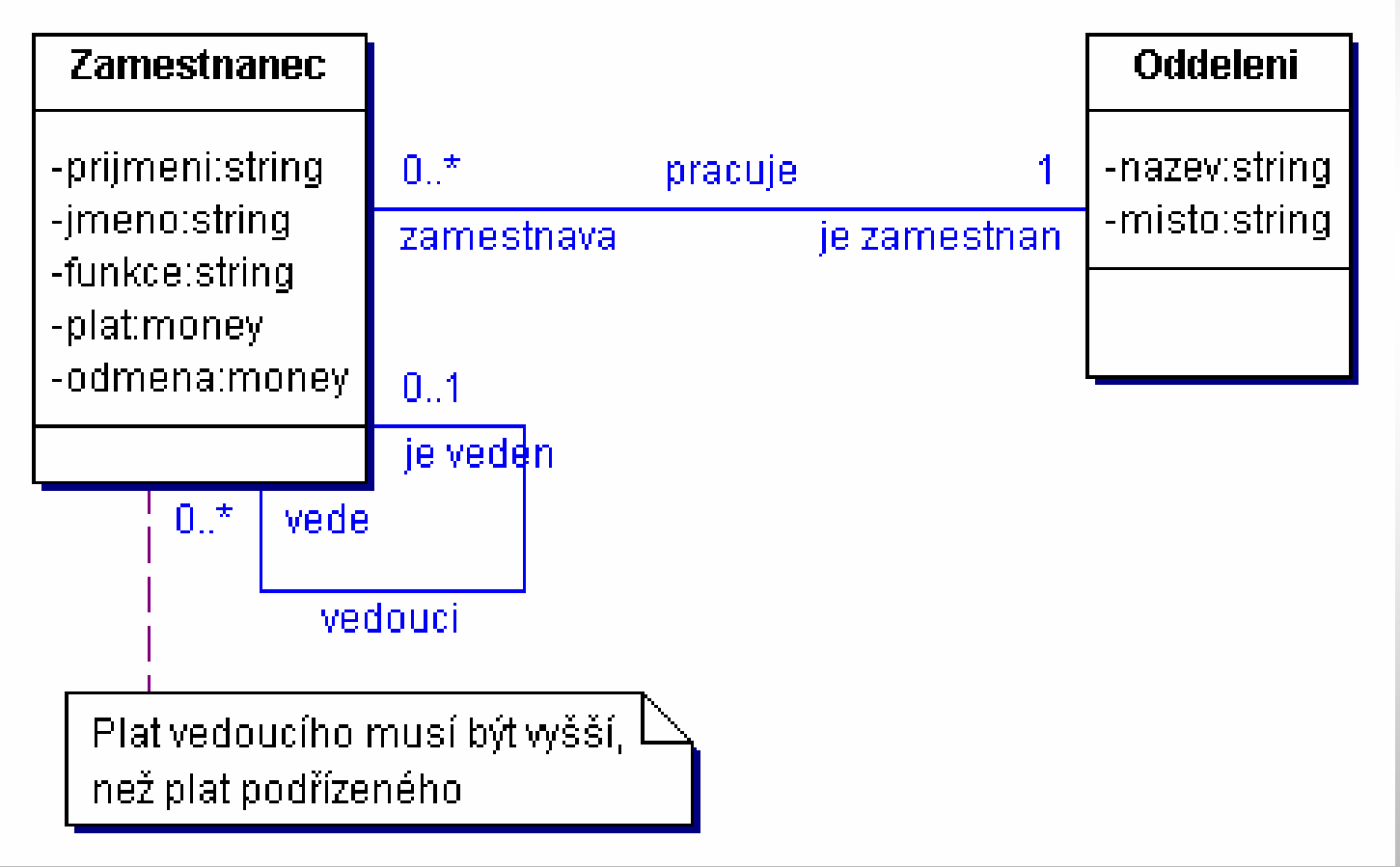
Z analytického konceptuálního
datového modelu musíme
navrhnout model logický

Konceptuální datový model specifikuje

- Typy dat (které entity, třídy, …)
- Vztahy mezi nimi
- Další logická omezení (integrity constraints)

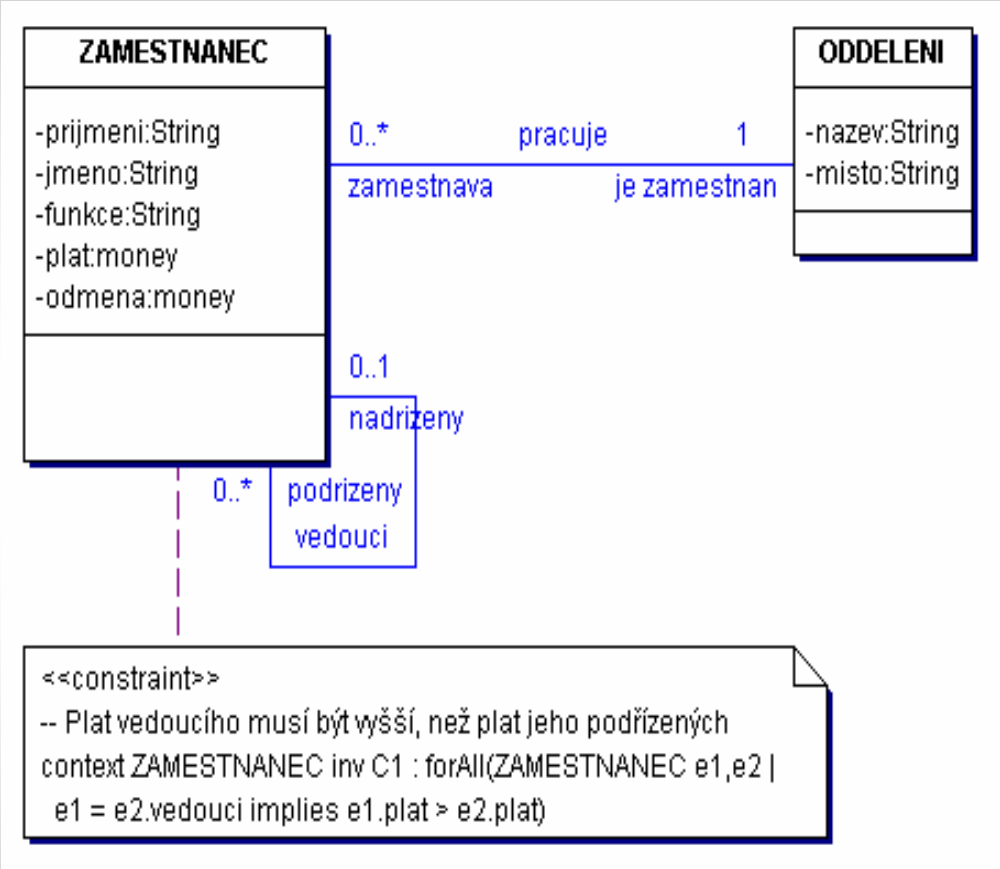
Pozn: Model tříd popisuje i operace

Příklad: Diagram tříd pro personální agendu



Integritní omezení jsou součástí specifikace

- Integritní omezení zajišťují, aby popisovaná data (data uložená v databázi) byla pokud možno korektní a úplná.
- V UML můžeme pro specifikaci integritních omezení použít jazyk OCL (Object Constraint Language).



Návrh reprezentace dat

- Pro každou datovou paměť (úložiště) musíme navrhnout způsob reprezentace - může to být systém ovládání souborů, systém řízení báze dat (relační, objektové, objektově-relační), speciální datový stroj (SW, SW+HW).
- Následuje převod konceptuálního modelu do logického.
- Součástí převodu je i návrh zajištění integrity dat
- Návrh zajištění konzistence dat, zálohování, archivace apod.

Návrh reprezentace dat pomocí relačního databázového systému

Vstup: konceptuální datový model (diagram tříd + popis integritních omezení)

Výstup: logický relační datový model (SQL-1999), včetně návrhu realizace integritních omezení

Pozn.: Výstupem je obecné SQL, při skutečné implementaci návrhu musí být ještě výstup přizpůsoben konkrétnímu stroji

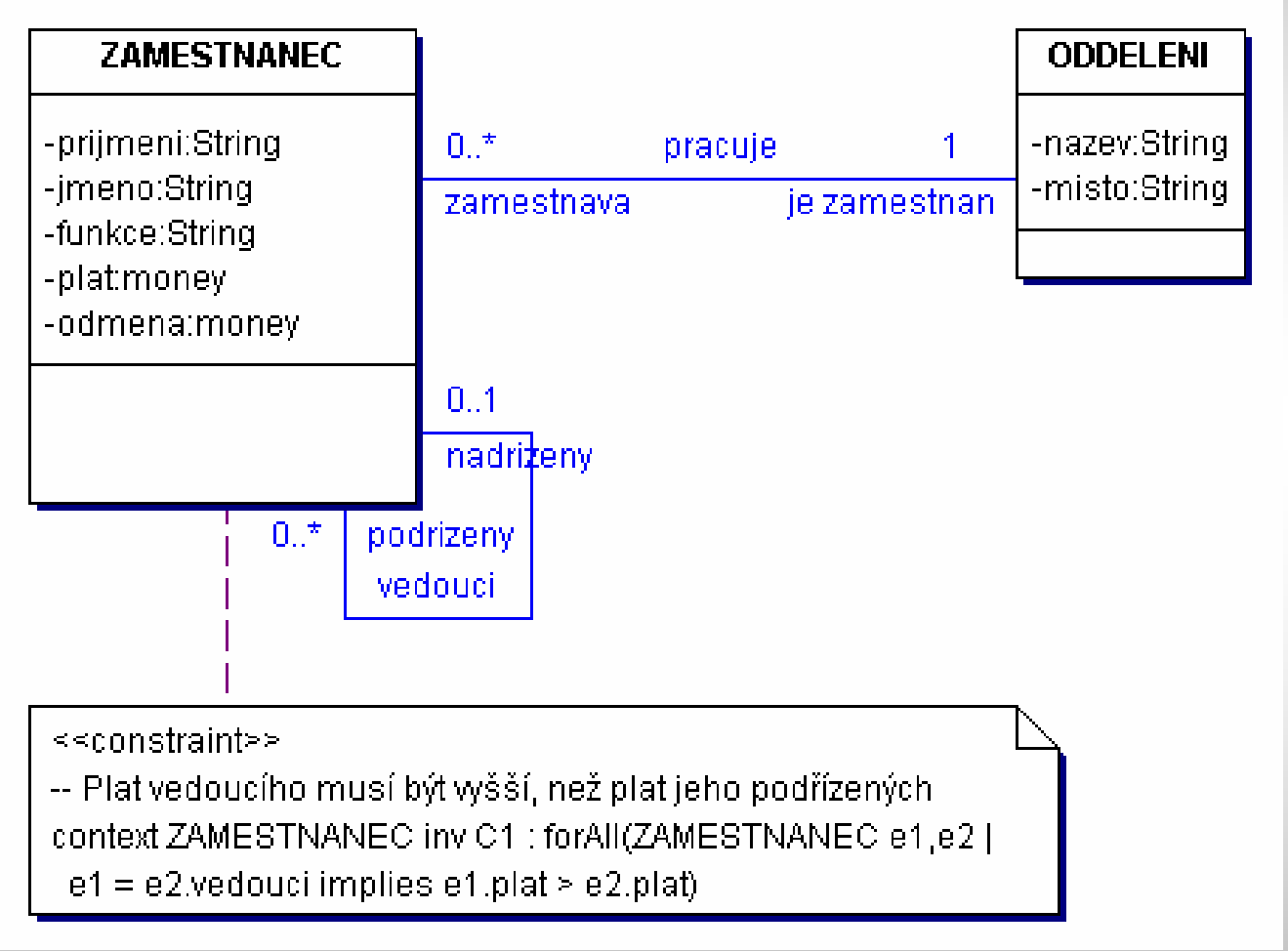
Postup návrhu

- Úprava (normalizace) konceptuálního modelu
- Návrh reprezentace typů (entit)
- Návrh reprezentace vztahů
- Návrh reprezentace integritních omezení

Úprava (normalizace) konceptuálního modelu

- Vyloučení multihodnotových a násobných atributů
- Vyloučení funkčních závislostí (odstranění redundance dat) – převod modelu do 3-NF (příp. 4-té, či 5-té normální formy)
- Náhrada nebinárních vztahů binárními
- Náhrada vztahů typu M:N přidruženými třídami

Normalizovaný datový model



Návrh reprezentace

- Pro každou jednoduchou entitu (typ) navrhneme tabulku, jméno tabulky bude množné číslo jména typu.
- Návrh jmen sloupců pro reprezentaci atributů a odpovídajících domén.
- Doplníme informace o volitelnosti formátu sloupců.
- Z nejčastěji používané unikátní identifikace vytvoříme primární klíč, nebo zavedeme nový identifikační sloupec (OID).

Návrh reprezentace (pokr.)

- Pro N-konce vztahů přidáme k tabulce jednoznačné identifikace z tabulky na 1-konci (volitelné vztahy indikují nepovinnost. Současně přidáme odpovídající cizí klíče.
- Pro každý vztah typu nadtyp/podtyp navrhneme reprezentaci (společná tabulka s rozlišovací položkou, samostatné tabulky).
- Pro každý vztah typu celek/část navrhneme reprezentaci (společná tabulka s rozlišovací položkou, samostatné tabulky).

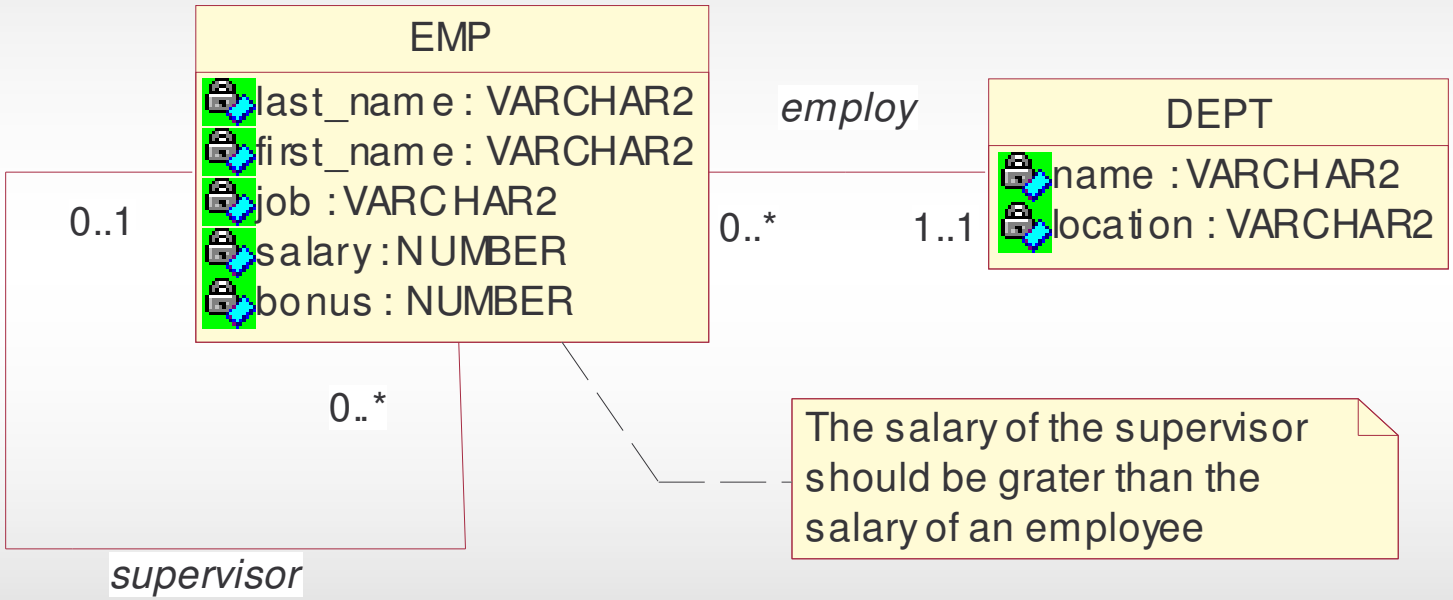
Návrh reprezentace (pokr.)

- Pro každý exkluzivní vztah (exkluzivní podtypy) rozhodneme, zda se má řešit společnou doménou, nebo explicitními cizími klíči.
- Doplníme sloupce odpovídající často používaným odvozeným atributům a navrhneme mechanizmus jejich údržby.
- Navrhneme indexy pro často využívané unikátní kombinace, které nejsou realizovány jako primární klíče. Indexy rovněž vytvoříme pro cizí klíče.

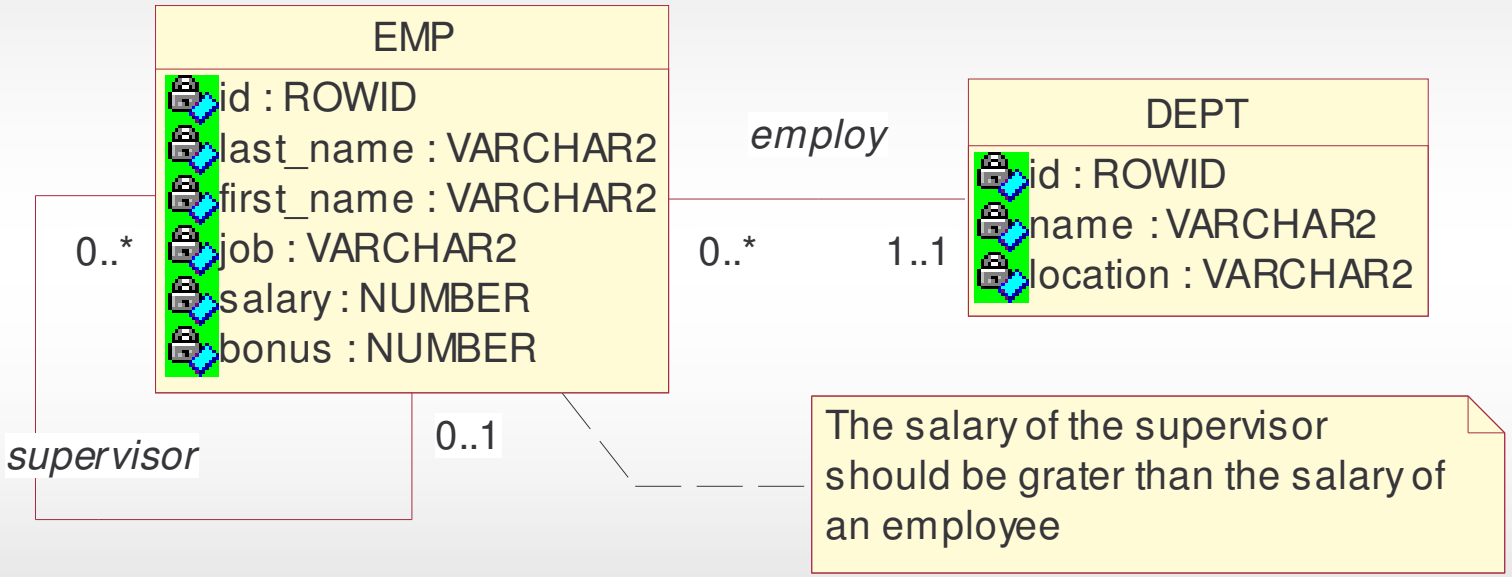
Návrh reprezentace (pokr.)

- Přidáme definice pohledů (zejména pro nadtypy, podtypy, celky a části).
- Pro generované primární klíče přidáme definice sekvencí pro jejich generování (může být implementačně závislé).
- Navrhneme řešení integritních omezení (použijeme deklarativní relační integritní omezení, nebo navrhneme „triggery“).

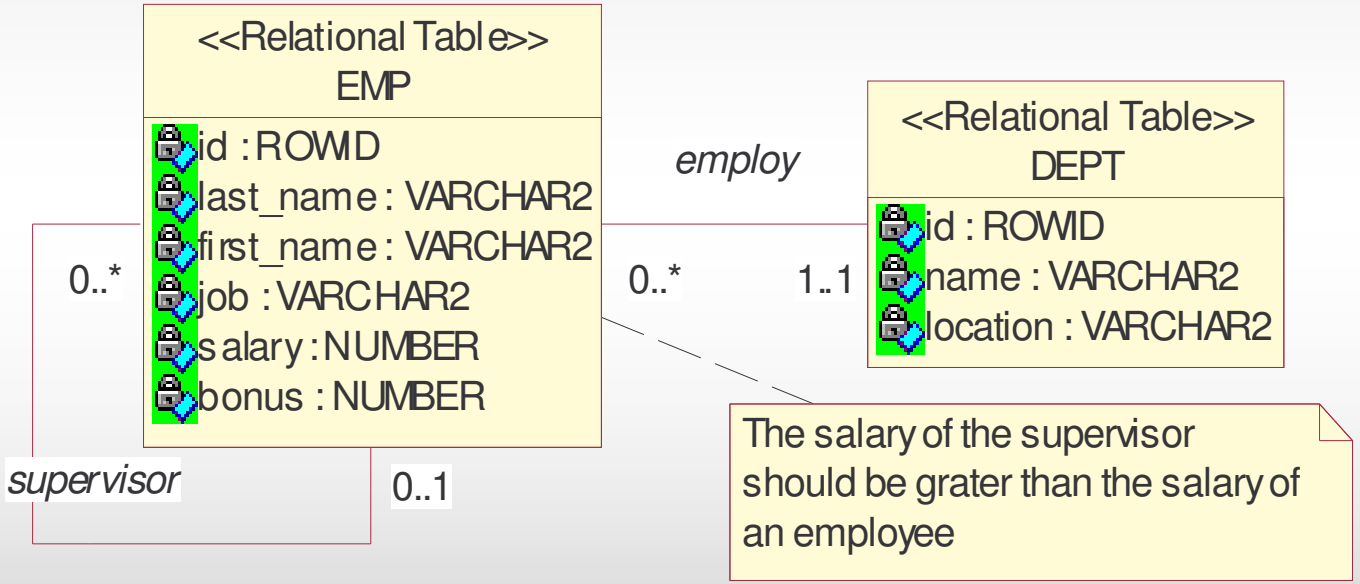
Jména a domény



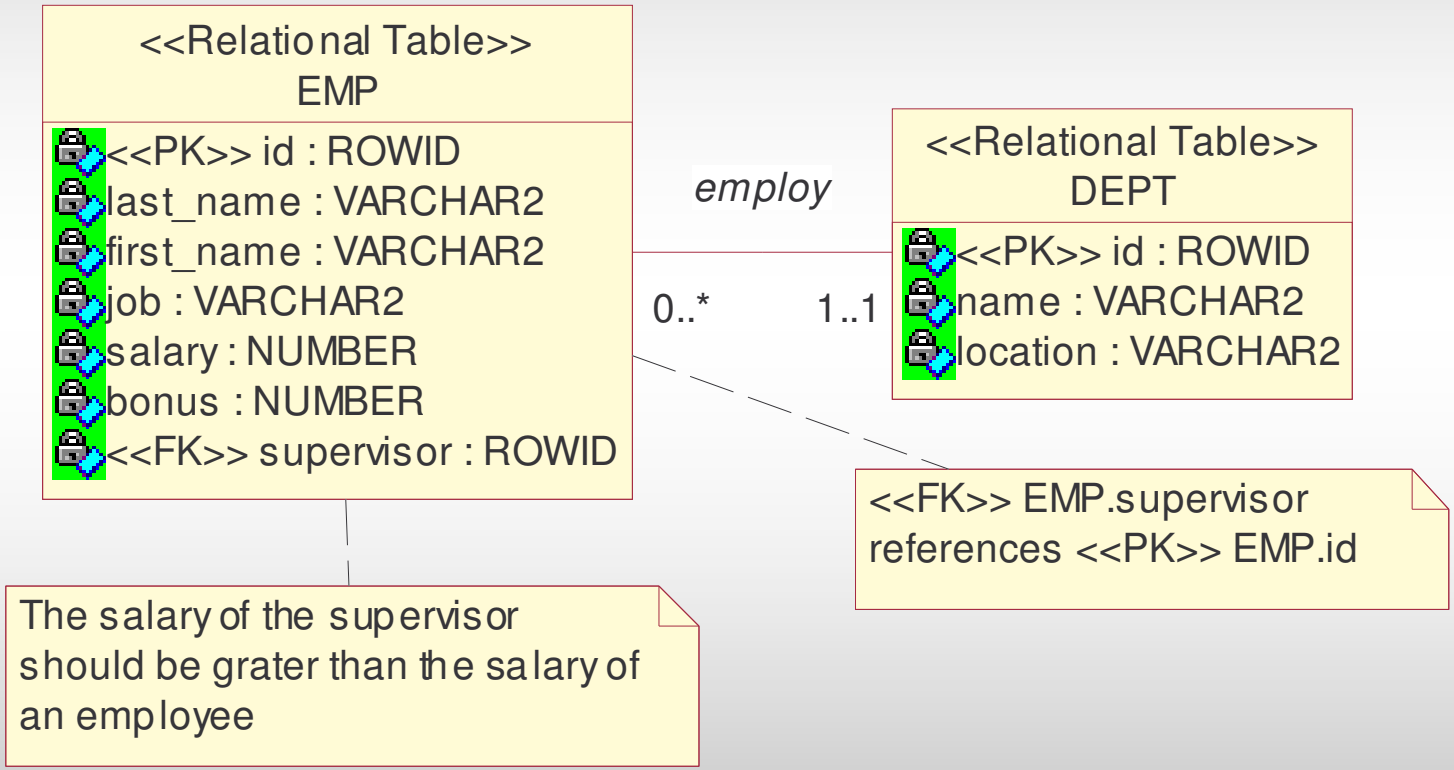
Primární identifikace



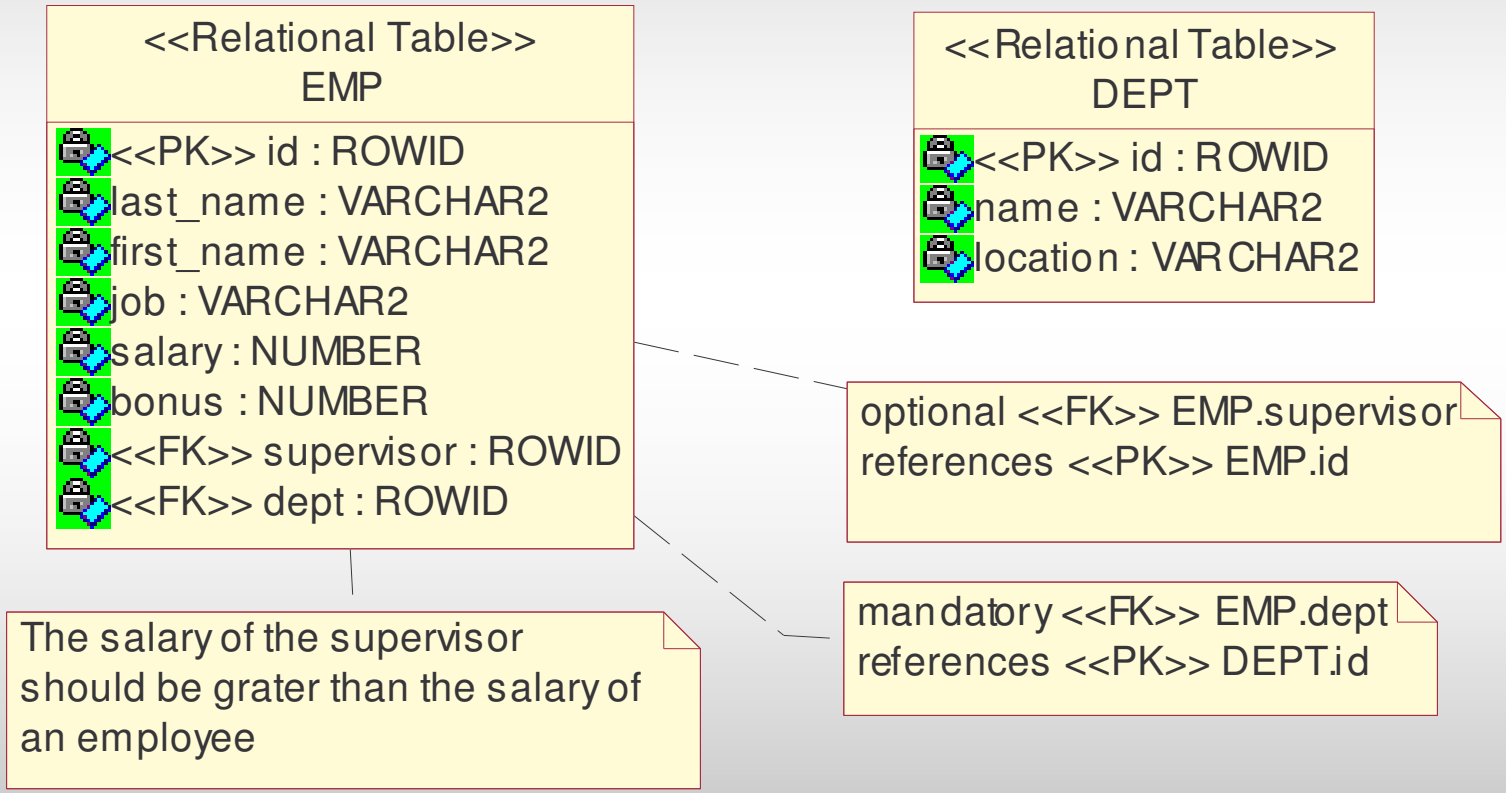
Vytvoříme tabulky



Cizí klíč pro “supervisora”



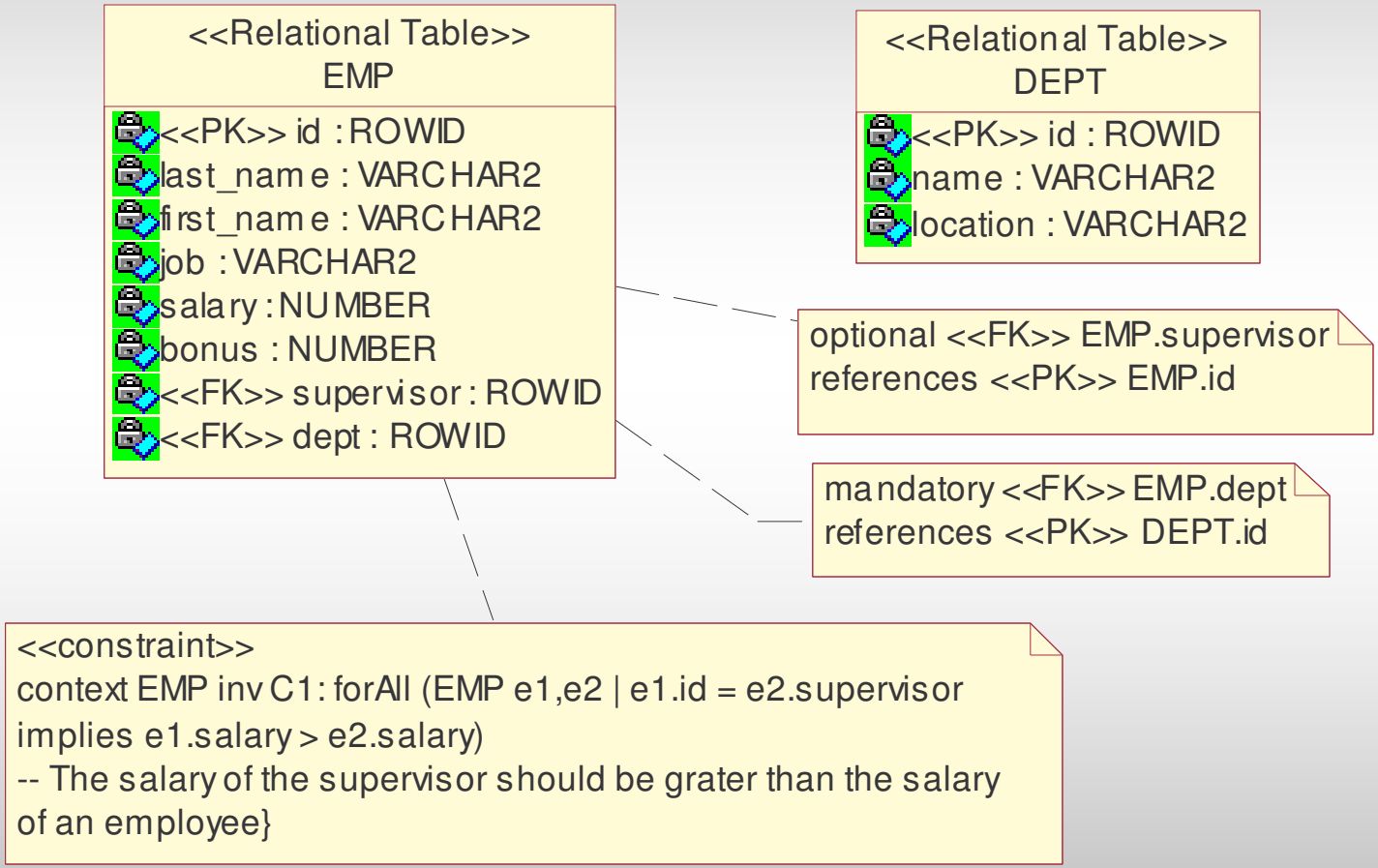
Cizí klíč pro “zaměstnance”



Návrh reprezentace integritních omezení

- Zkusíme vytvořit deklarativní omezení.
- Pokud by nefungovala, musíme navrhnout „triggery“.

Specifikace v OCL



Konverze integritních omezení

Příklad:

```
context EMP inv min_age:
      self.age >= 18
```

⇒

```
alter table EMP add constraint
 min_age check (self.age >= 18);
```

Příklad: Personální agenda

```
context EMP inv C1: forAll (EMP e1,e2 |
  e1.id = e2.supervisor implies
     e1.salary > e2.salary)
⇒
alter table EMP add constraint C1 check
  (not exists (select 'X'
                from EMP e1, EMP e2
                 where e1.id =
  e2.supervisor
                and e2.salary >=
  e1.salary));
```

Odvozené SQL I. (table DEPT)

```
create table DEPT (
   id ROWID primary key,
   name VARCHAR2(20),
   location VARCHAR2(20)
);
```

Odvozené SQL II. (table EMP)

```
create table EMP (
   id ROWID primary key,
   last_name VARCHAR2(35),
   first_name VARCHAR2(35),
   job VARCHAR2(10),
   salary NUMBER(9,2),
   bonus NUMBER(9,2),
   supervisor ROWID references EMP(id),
   dept ROWID not null references DEPT(id)
);
```

Odvozené SQL III. (ostatní)

```
alter table EMP add constraint C1
check (not exists
   (select 'X'
    from EMP e1, EMP e2
    where e1.id = e2.supervisor
    and e2.salary > e1.salary));
```

Ale problém – někde to nefunguje !!!

Zkusíme trigger !!!

Příklad (zjednodušeno)

```
create or replace trigger C1
before insert or update of salary on EMP e1
declare
  cnt INTEGER;
begin
  select count(*) into cnt from EMP e2
    where e1.id = e2.supervisor
    and e2.salary > e1.salary;
  if cnt = 0 then
    e1.salary := :new.salary
  end if;
end;
```

Příklad – jiné řešení

```
create or replace view EMP_C1 as
select *
from EMP e1 join EMP e2 on (e1.id =
  e2.supervisor)
where e2.salary >= e1.salary;
with check option;
```

Uživatelům zpřístupníme pouze EMP_C1

SWI041:
Návrh (funkční model)

Měl by odpovědět na otázku JAK, tj. jaké moduly, třídy, komponenty

Programování ve velkém

Dekompozice

Návrh - funkční model

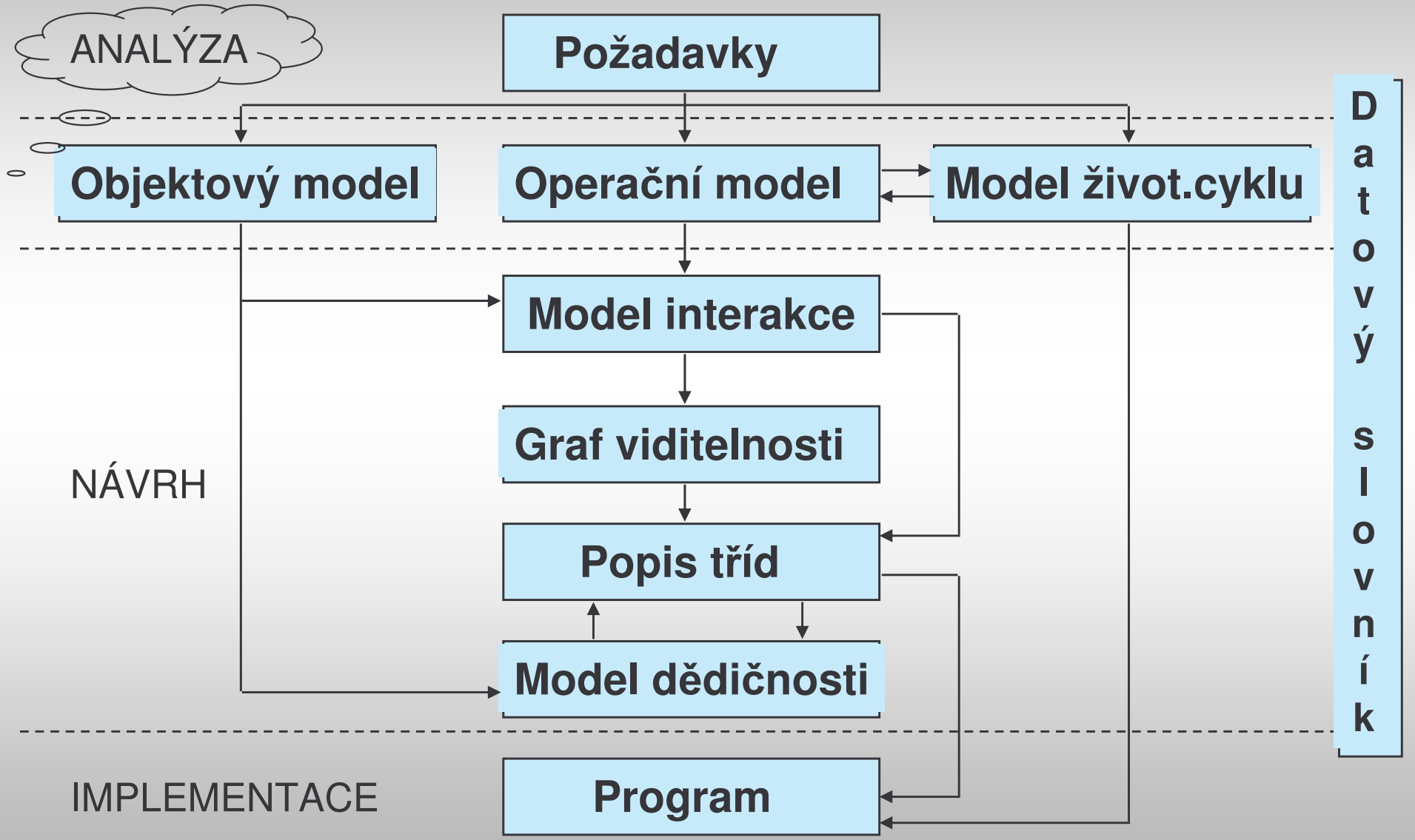
- Funkční model musíme rozložit na části zvládnutelné technikami programování (v malém)
- Takové části se obvykle nazývají moduly, komponenty, …
- Z analytického funkčního modelu, kde je popsáno CO (jaké služby), musíme navrhnout JAK se to udělá (který modul bude službu realizovat a jakými dalšími moduly bude muset spolupracovat)

Techniky návrhu modulů

- Objektový návrh (např. Coleman) – z analytických modelů vytvoříme popisy tříd
- Návrh na základě datových toků (např. Page-Jones) – z analytických diagramů datových toků vytvoříme diagramy struktury
- Návrh na základě datových struktur (např. Jackson) – z analytických popisů datových struktur vytvoříme popisy programů
- Návrh fukční dekompozicí – z analytických popisů funkcí vytvoříme přímou funkční dekompozicí dostatečně podrobné popisy funkcí

Příklad: Návrh dle Fusion

Model životního cyklu (Fusion)



Vstupní dokumenty pro návrh ve Fusion

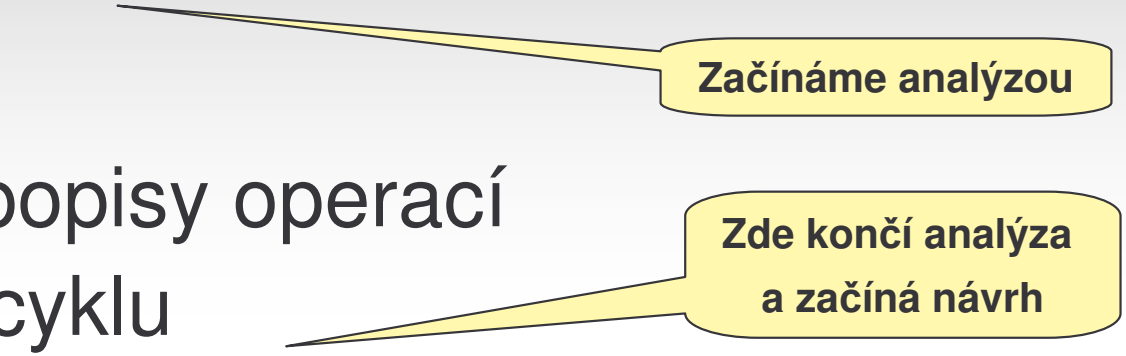
- Datový slovník
- Objektový model (ER-model)
- Operační model (kontext, model jednání, scénáře, popisy operací)
- Model životního cyklu

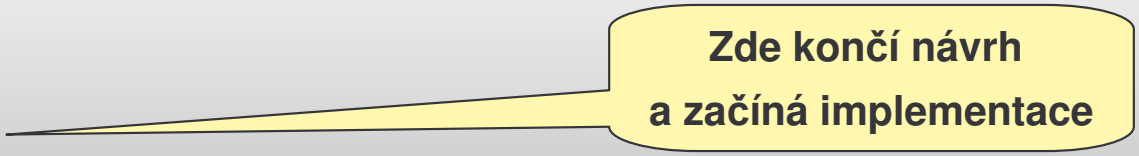
Výstupní dokumenty návrhu ve Fusion

- Datový slovník
- Architektura systému (HW,SW)
- Popis komponent jako popis tříd
- Model životního cyklu

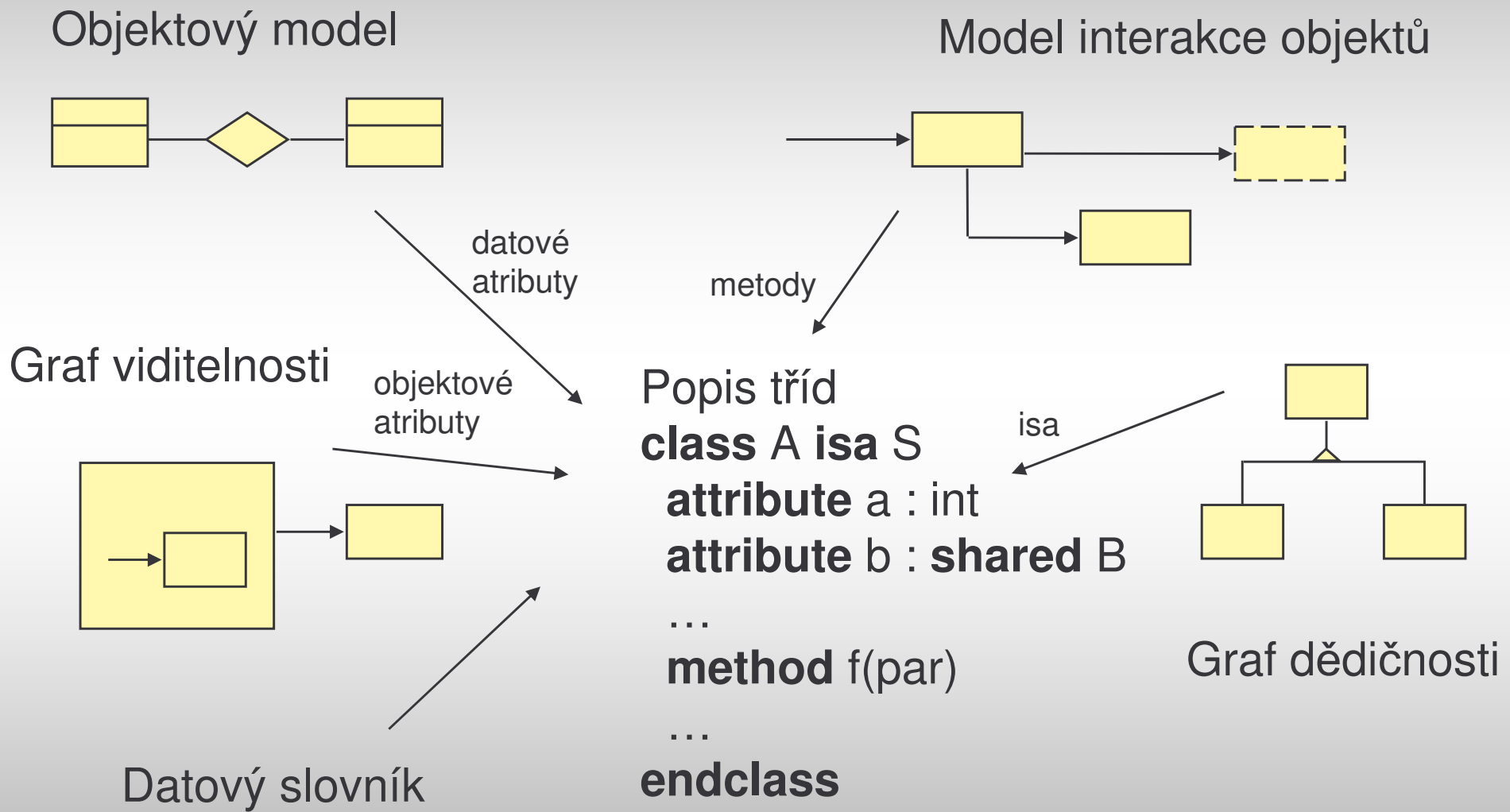
Postup dle Fusion

- datový slovník
- objektový model
- funkční model - popisy operací
- model životního cyklu
- model interakce - scénáře, diag.kolaborace
- graf viditelnosti
- popis tříd
- graf dědičnosti
- program





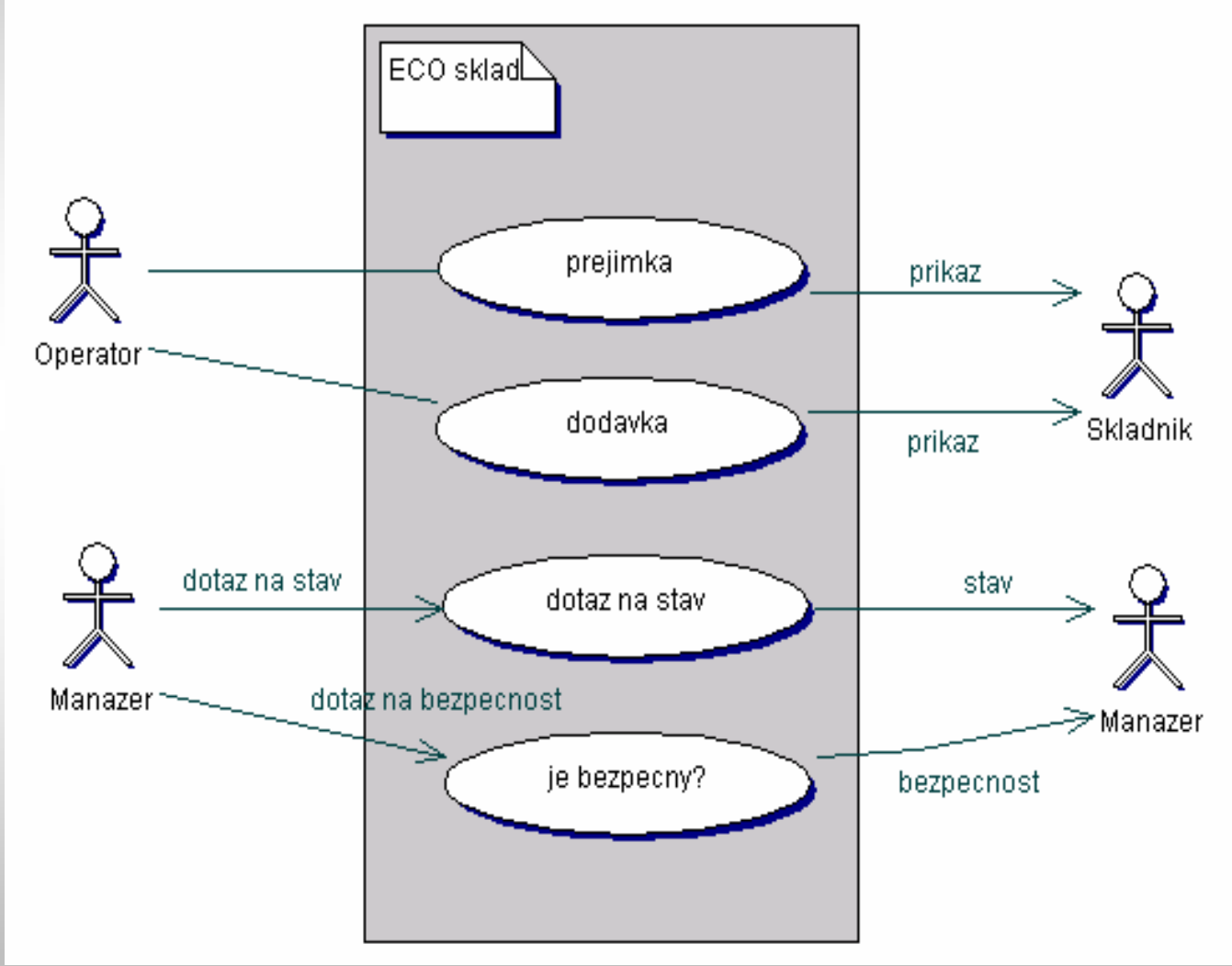
Princip návrhu ve Fusion

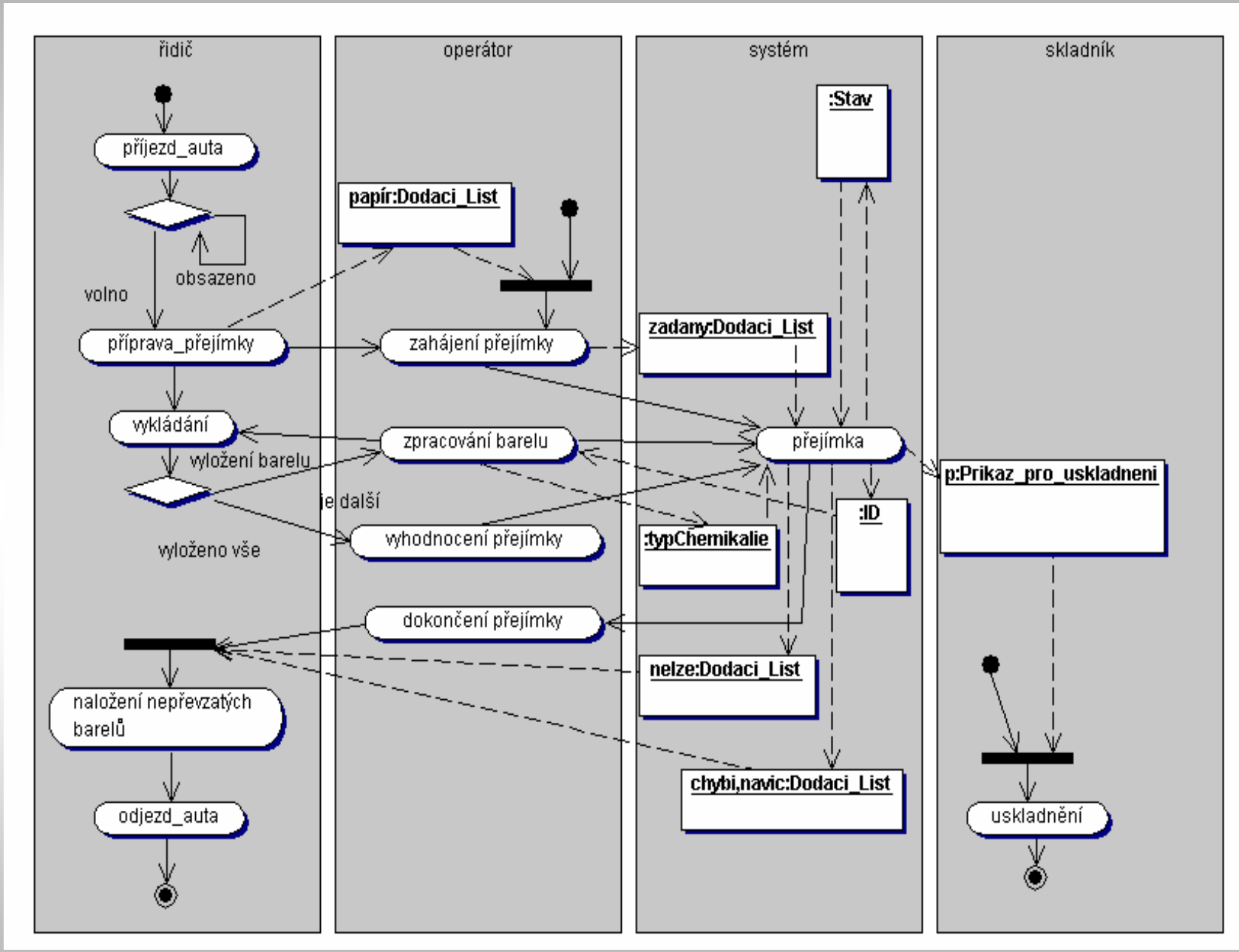


Postup návrhu ve Fusion

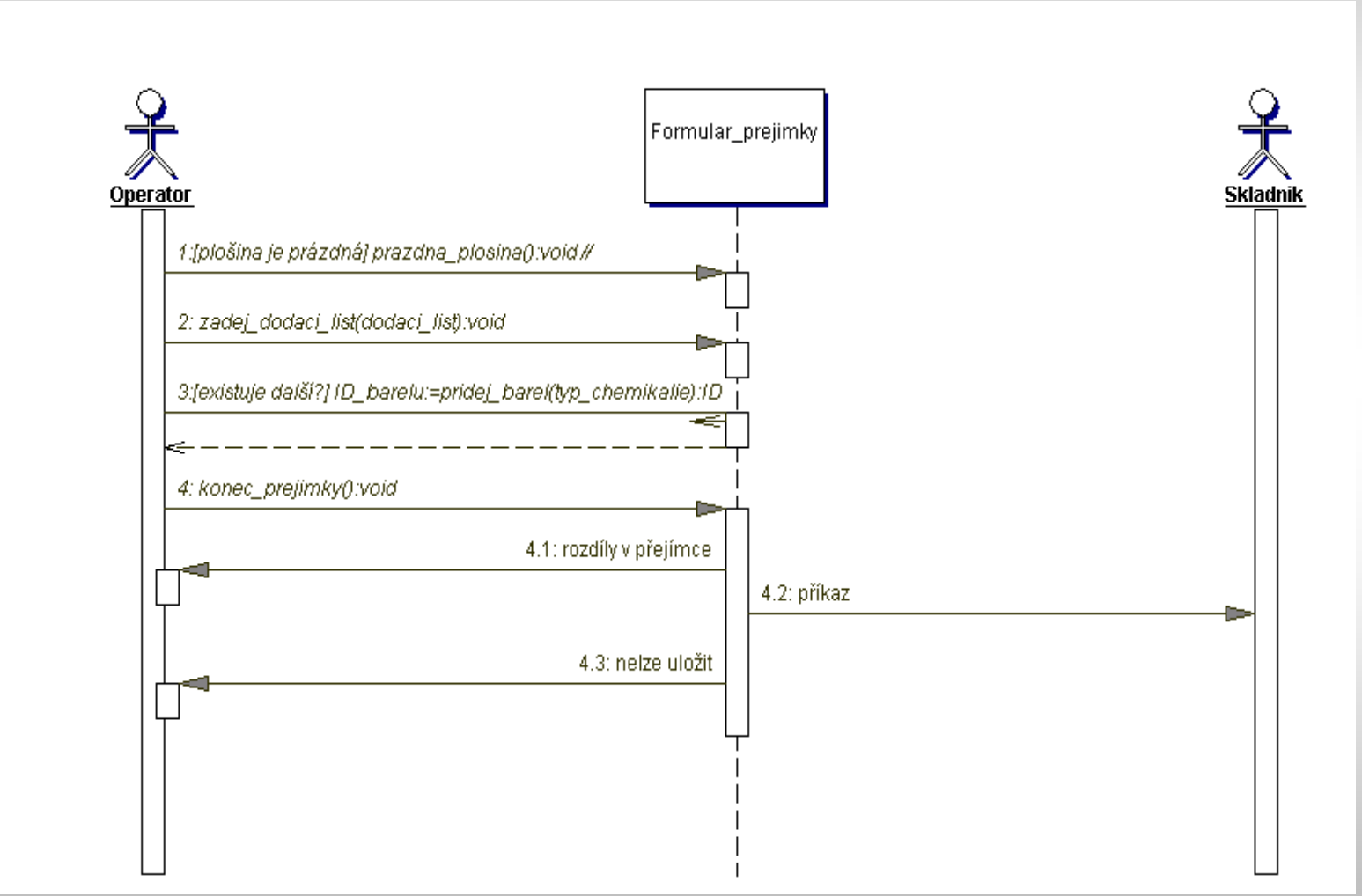
- Pro každou operaci funkčního modelu nakresli diagram interakce (včetně metod vzniklých při návrhu)
- Pro každou třídu datového modelu nakresli graf viditelnosti podle diagramů interakce
- Pro každou třídu vytvoř popis třídy
- Doplň návrh grafy dědičnosti

Př. ECO sklad


ECO sklad
Operator
prejimka
prikaz
Skladnik
dodavka
prikaz
Manazer
dotaz na stav
dotaz na stav
stav
Manazer
dotaz na bezpecnost
je bezpecny?
bezpecnost


řidič
operátor
systém
skladník
příjezd_auta
obsazeno
volno
příprava_přejímky
vykládání
vyložení barelu
vyloženo vše
naložení nepřevzatých barelů
odjezd_auta
papír:Dodaci_List
zahájení přejímky
zpracování barelu
vyhodnocení přejímky
dokončení přejímky
:Stav
zadany:Dodaci_List
přejímka
:typChemikalie
:ID
nelze:Dodaci_List
chybi,navic:Dodaci_List
p:Prikaz_pro_uskladneni
uskladnění

Scénář pro “přejímku” v UML

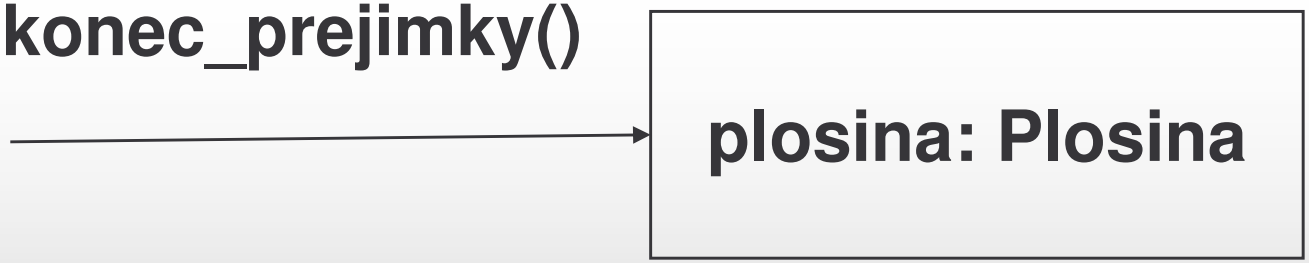


Postup návrhu ve Fusion

- Pro každou operaci funkčního modelu nakresli diagram interakce (včetně metod vzniklých při návrhu)
- **Pro každou třídu datového modelu nakresli graf viditelnosti podle diagramů interakce**
- **Pro každou třídu vytvoř popis třídy**
- **Doplň návrh grafy dědičnosti**

Dokumentace návrhu pro ECO

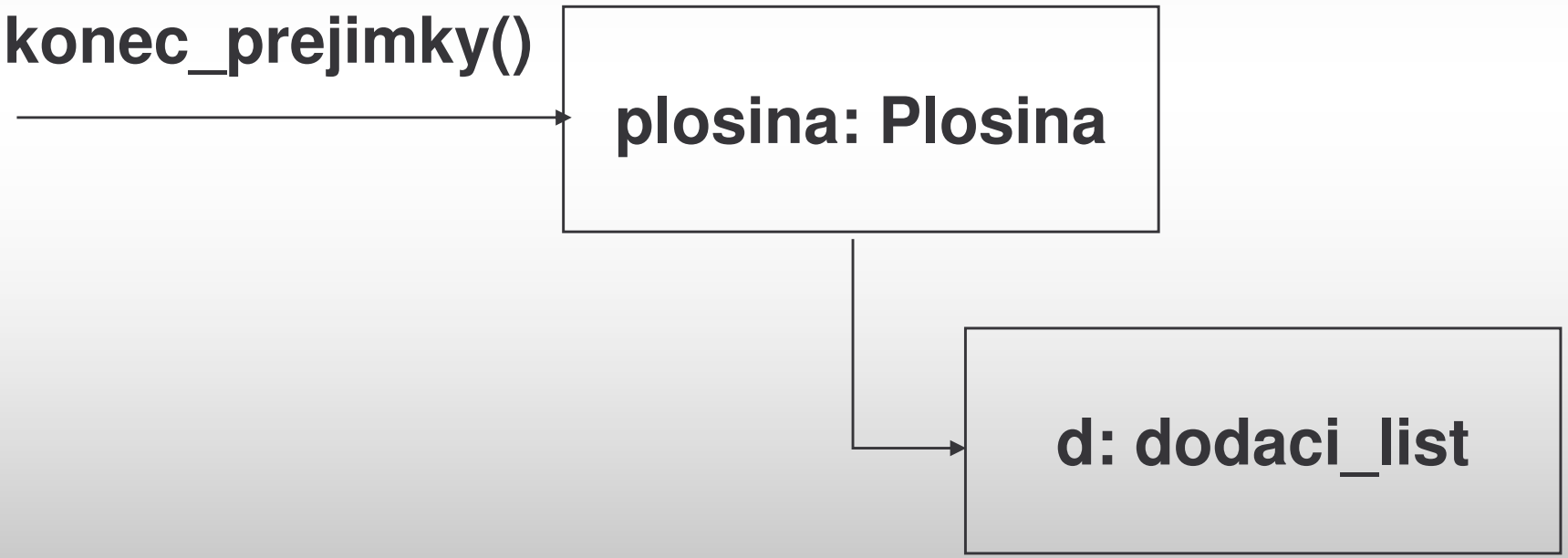
Diagram interakce pro “konec přejímky“



1. Výběr zodpovědného objektu

Dokumentace návrhu pro ECO

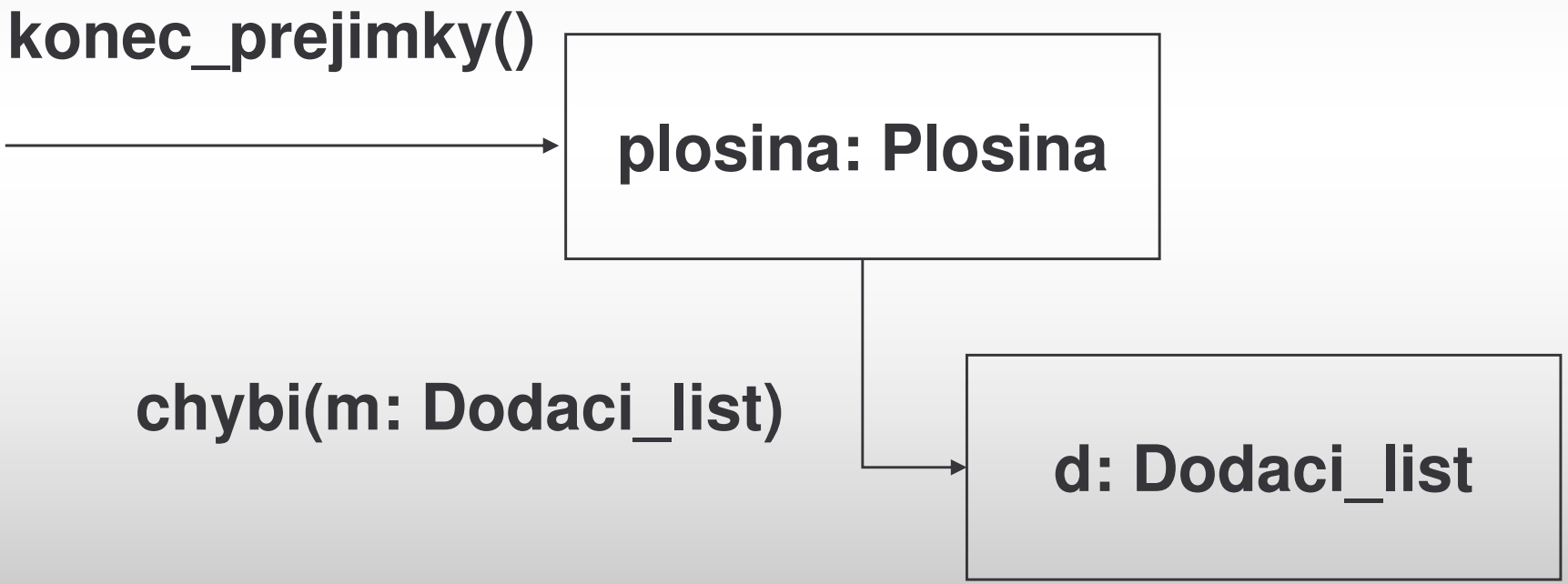
Diagram interakce pro “konec přejímky“



2. Výběr kooperujícího objektu

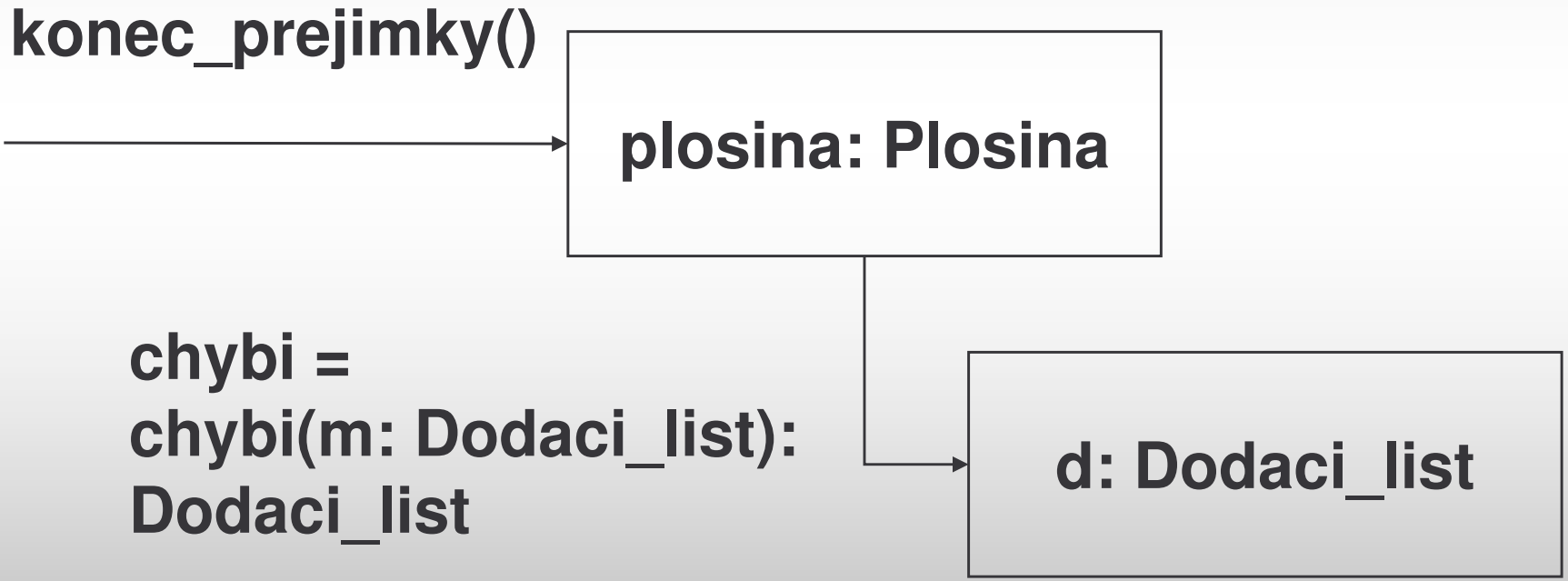
Dokumentace návrhu pro ECO

Diagram interakce pro "konec přejímky"



3. Výběr metody kooperujícího objektu

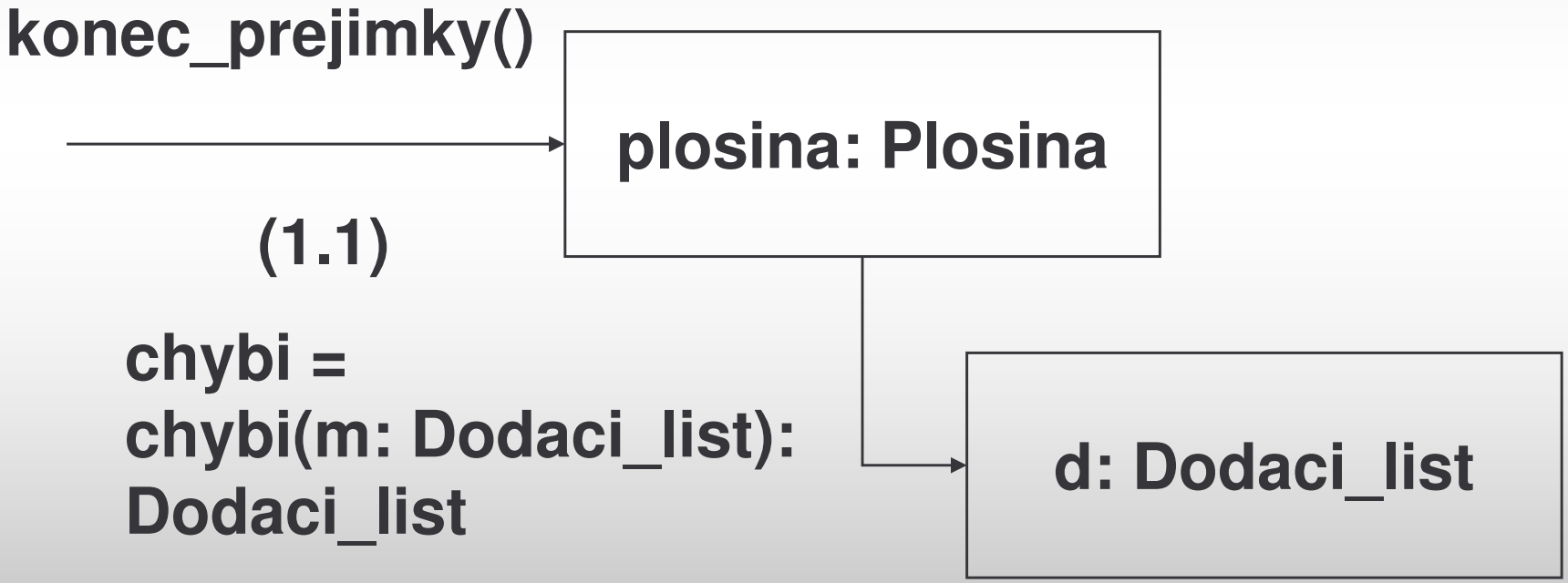
Diagram interakce pro "konec přejímky"



4. Bude něco vracet?

Dokumentace návrhu pro ECO

Diagram interakce pro "konec přejímky"



5. Stanovení pořadí

Popis pro „chybi“

- Operace „chybi“ nepochází z analýzy, vznikla při návrhu, ale je nutno ji rovněž popsat (aby pak programátor věděl, co má dělat)

Popis pro “chybi”

Dokumentace návrhu pro ECO

Operation: chybi

Description: **zjistí rozdíly v přejímce – co chybí**

Reads: *supplied* m: dodaci_list,
zadany_dodaci_list: dodaci_list

Changes: new chybi: dodaci_list

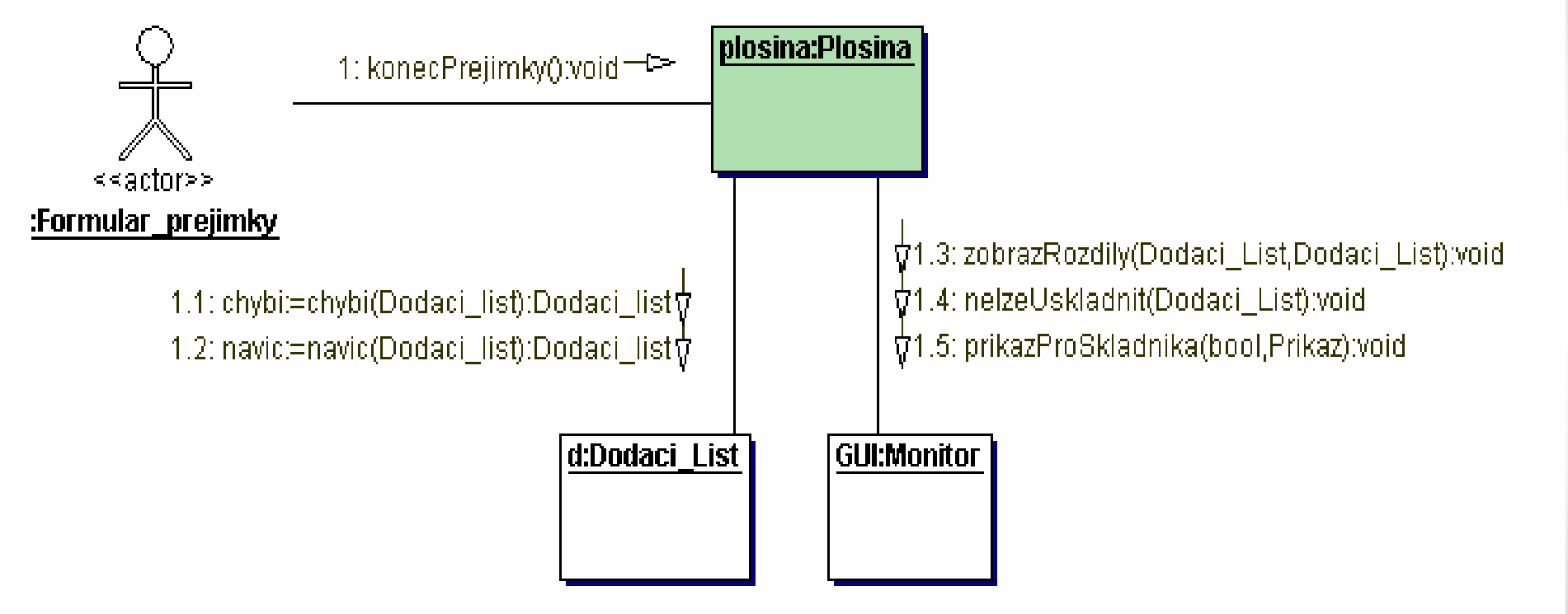
Sends:

Assumes:

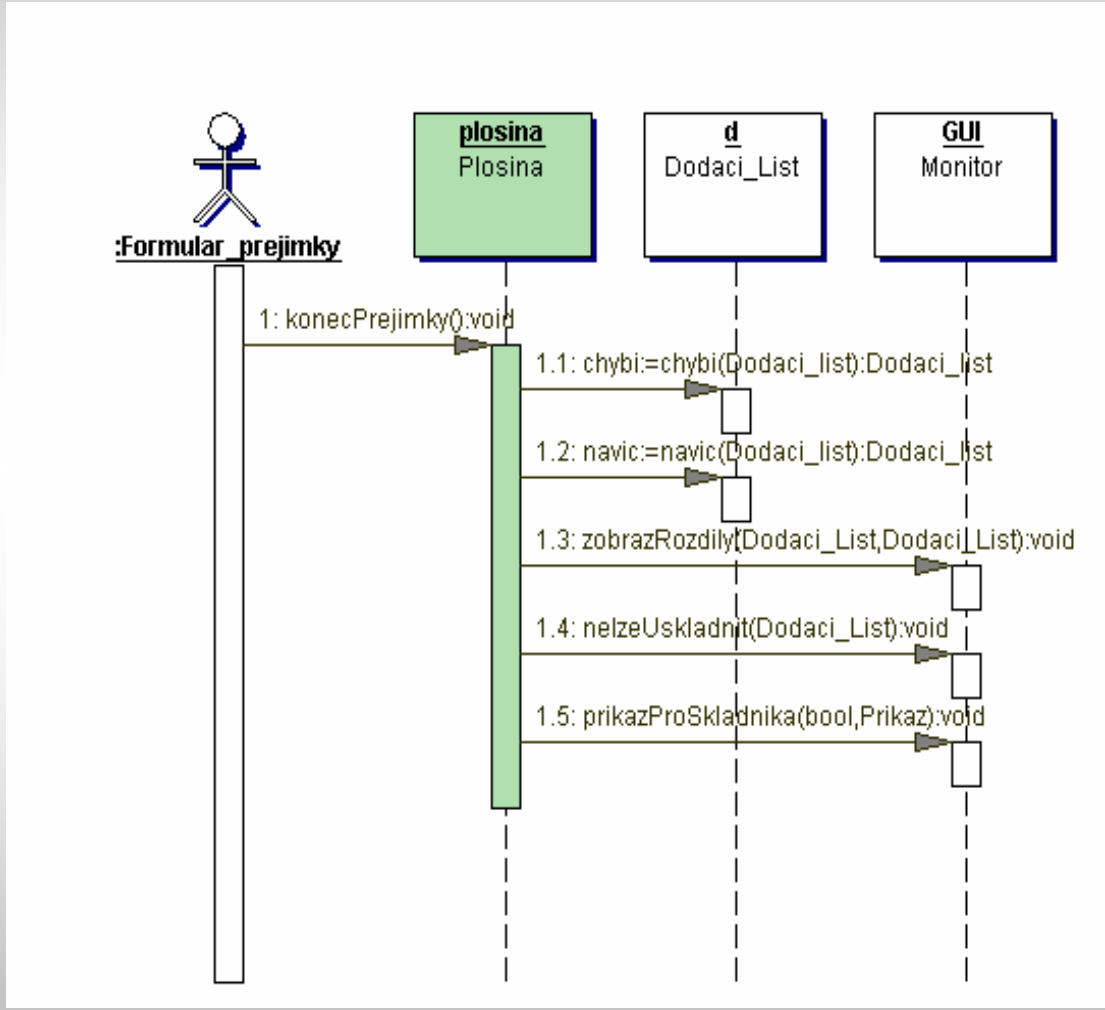
Results:

- porovná zadaný dodací list **m** s dodacím listem **zadany_dodaci_list**
- vytvoří objekt **chybi: dodací list**, obsahující barely které chybí

Výsledek


<<actor>>
:Formular_prejimky
1: konecPrejimky():void
plosina:Plosina
1.1: chybi:=chybi(Dodaci_list):Dodaci_list
1.2: navic:=navic(Dodaci_list):Dodaci_list
1.3: zobrazRozdily(Dodaci_List,Dodaci_List):void
1.4: nelzeUskladnit(Dodaci_List):void
1.5: prikazProSkladnika(bool,Prikaz):void
d:Dodaci_List
GUI:Monitor

Nebo jako scénář



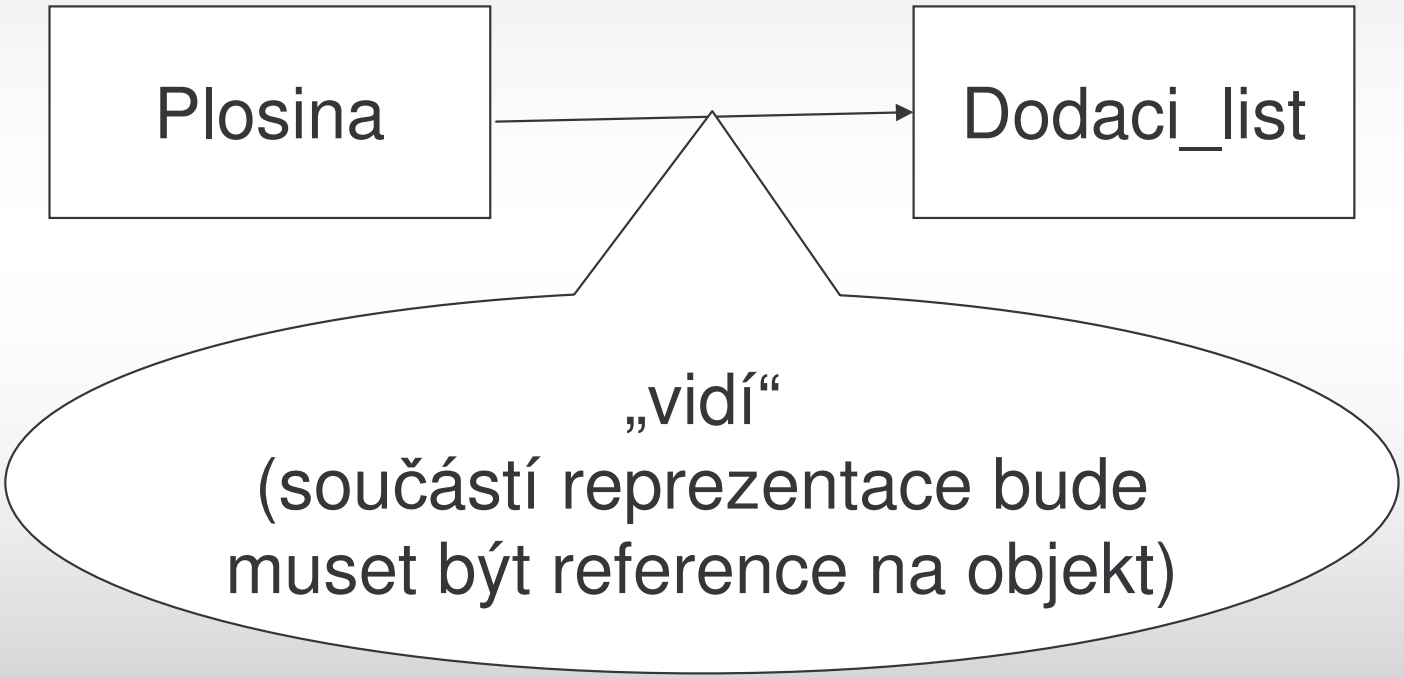
Postup návrhu ve Fusion

- Pro každou operaci funkčního modelu nakresli diagram interakce (včetně metod vzniklých při návrhu)
- Pro každou třídu datového modelu nakresli graf viditelnosti podle diagramů interakce
- Pro každou třídu vytvoř popis třídy
- Doplň návrh grafy dědičnosti

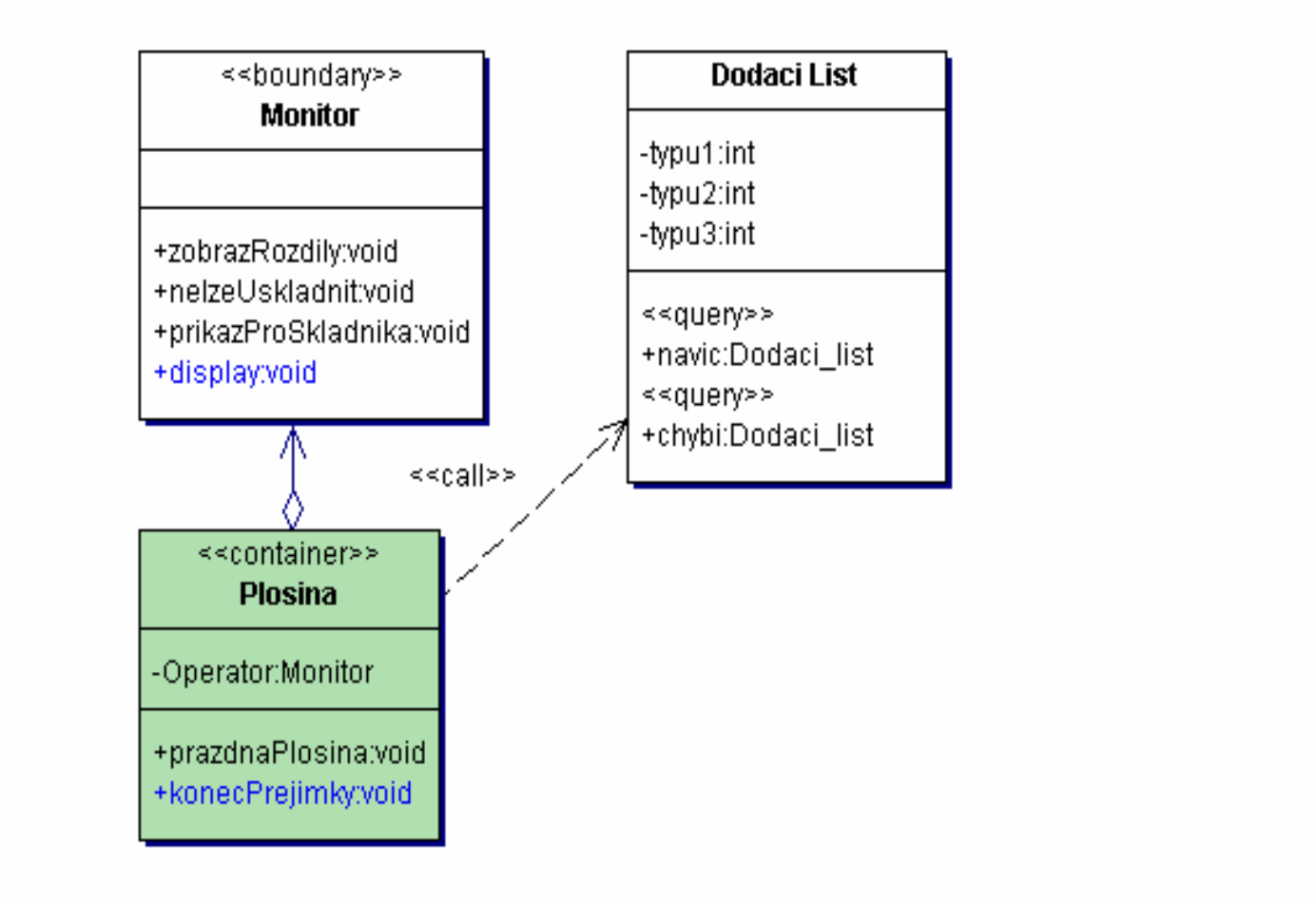
Grafy viditelnosti

- Popisují nutné vazby mezi třídami, které souvisí s potřebou viditelnosti (mám-li volat službu objektu jiné třídy, musím jej vidět)
- Definují potřebnou referenční strukturu (objektové atributy tříd)

Notace grafů viditelnosti



Viditelnost v UML



Postup návrhu ve Fusion

- Pro každou operaci funkčního modelu nakresli diagram interakce (včetně metod vzniklých při návrhu)
- Pro každou třídu datového modelu nakresli graf viditelnosti podle diagramů interakce
- Pro každou třídu vytvoř popis třídy
- Doplň návrh grafy dědičnosti

Popis tříd ve Fusion

```
class <jméno> [ isa <jméno> ]
// pro každý atribut
   [attribute][constant]<jméno>:[vazba]<typ>
   …
// pro každou metodu
   [method]<jméno>(<argumenty>)[:<typ>]
   …
endclass

vazba = [ref|(exclusive|shared)[bound]
argumenty = [<jméno>:<typ>](<jméno>:<typ>)*
typ = [col]<jméno>
```

Popis třídy “Plosina”

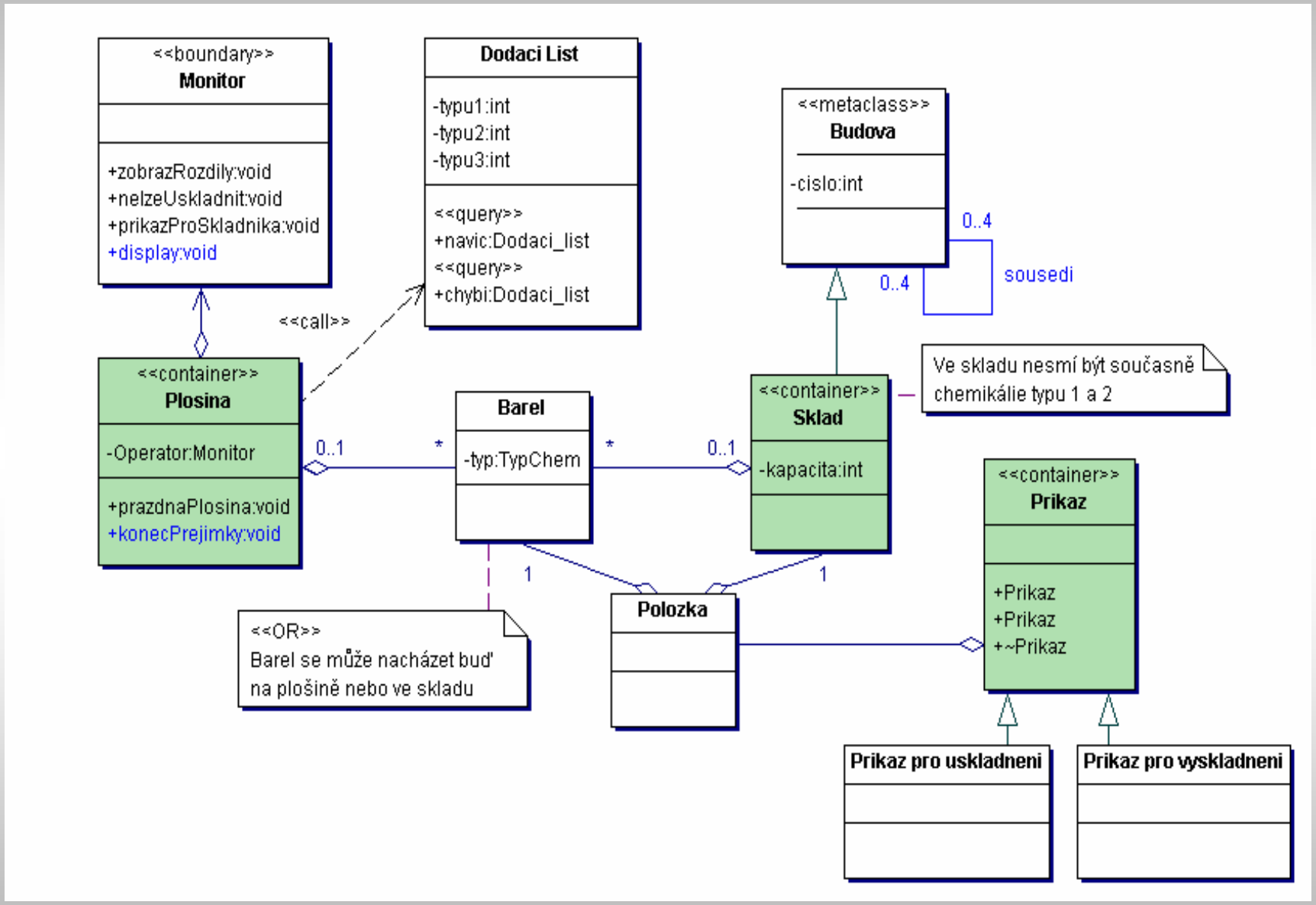
Dokumentace návrhu pro ECO

```
class Plosina
      attribute sklady:
            ref shared col Sklad
      attribute Operator:
            share bound Monitor
      attribute zadany_dodaci_list:
            ref shared Dodaci_list
      method konec_prejimky()
      method prazdna_plosina()
      method cti_obsah(): col Barel
      …
endclass
```

Popis třídy “Sklad”

Dokumentace návrhu pro ECO

```
class Sklad isa Budova
      attribute kapacita:int
      attribute sousedi:
            exclusive bound col Sklad
      attribute barely:
            exclusive bound col Barel
      method je_místo?(b: Barel): Bool
      method cti_cislo(): int
      method pridej(b: Barel)
      method kolik?(t: TypChem): int
      …
endclass
```


<<boundary>>
Monitor
+zobrazRozdily:void
+nelzeUskladnit:void
+prikazProSkladnika:void
+display:void
Dodaci List
-typu1:int
-typu2:int
-typu3:int
<<query>>
+navic:Dodaci_list
<<query>>
+chybi:Dodaci_list
<<metaclass>>
Budova
-cislo:int
0..4
0..4
sousedi
<<call>>
<<container>>
Plosina
-Operator:Monitor
+prazdnaPlosina:void
+konecPrejimky:void
0..1
*
Barel
-typ:TypChem
*
0..1
<<container>>
Sklad
-kapacita:int
Ve skladu nesmí být současně chemikálie typu 1 a 2
<<container>>
Prikaz
+Prikaz
+Prikaz
+~Prikaz
1
1
Polozka
<<OR>>
Barel se může nacházet buď na plošině nebo ve skladu
Prikaz pro uskladneni
Prikaz pro vyskladneni

Postup návrhu ve Fusion

- Pro každou operaci funkčního modelu nakresli diagram interakce (včetně metod vzniklých při návrhu)
- Pro každou třídu datového modelu nakresli graf viditelnosti podle diagramů interakce
- Pro každou třídu vytvoř popis třídy
- Doplň návrh grafy dědičnosti

Dva zdroje dědičnosti

- Společné a speciální vlastnosti dat odhalené při analýze
- Vlastnosti zděděné z použitých knihoven

Implementace třídy “Plošina”

Dokumentace implementace ECO

```
class Plosina {
protected:
      Set<Sklad &> sklady; // ref shared col
      Monitor *Operator; // share bound
      zadany_dodaci_list &dodaci_list; // ref shared
public:
      Plosina();
      ~Plosina();
      void konec_prejimky();
      void prazdna_plosina();
      List<Barel> &cti_obsah();
      …
endclass
```

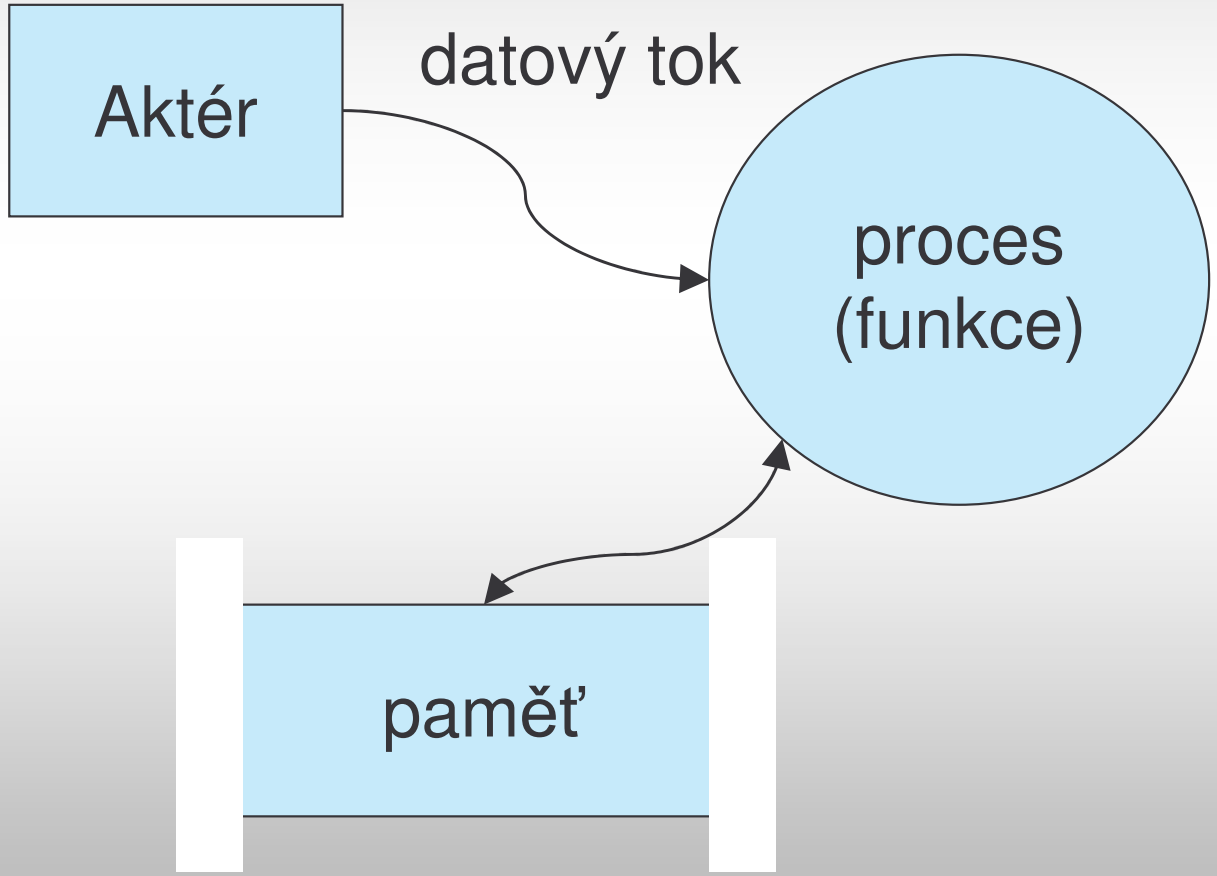
Návrh funkční dekompozicí

- Návrh fukční dekompozicí –
  z analytických popisů funkcí vytvoříme přímou funkční dekompozicí dostatečně podrobné popisy funkcí, generalizací navrhneme potřebné komponenty

Diagramy datových toků

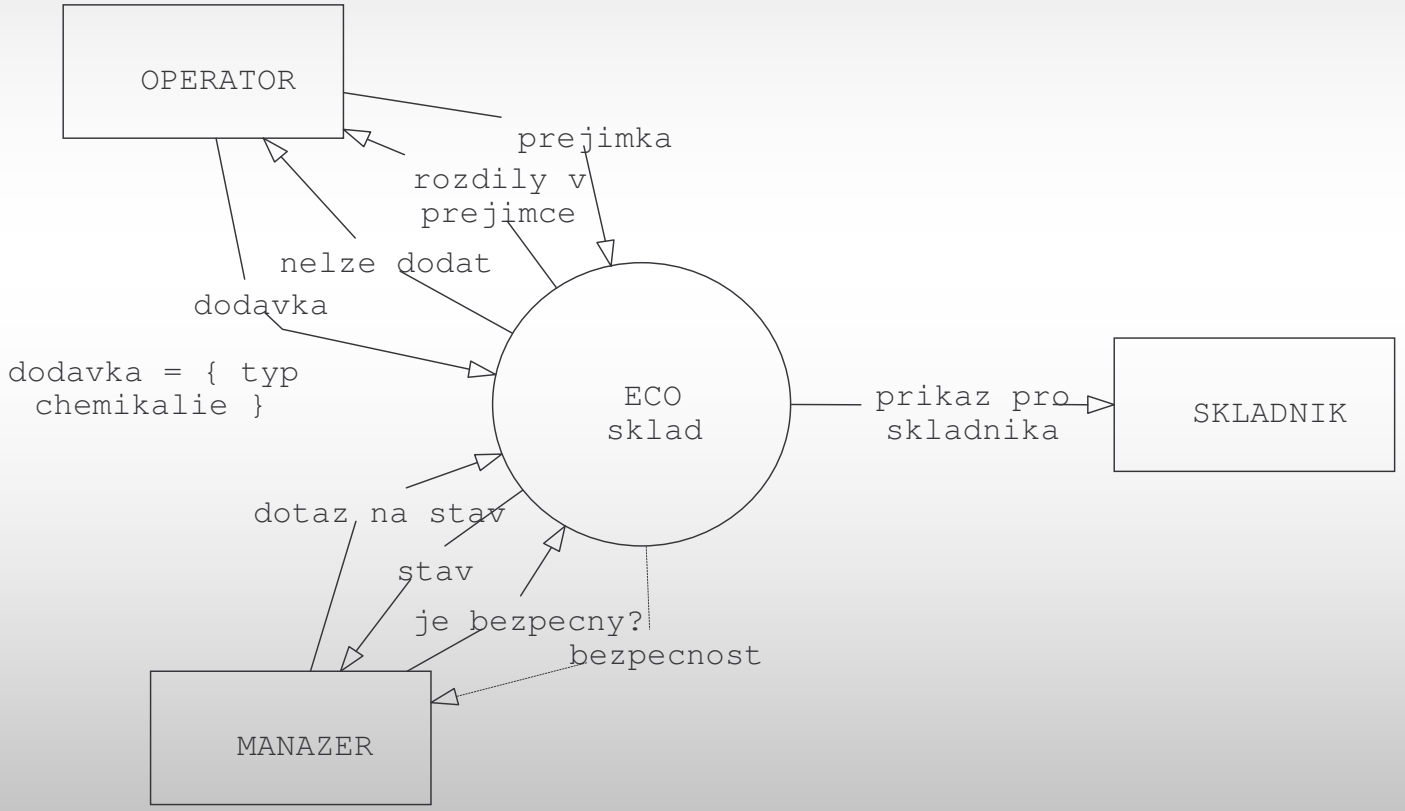
- Data Flow Diagrams (DFD)
- Hierarchická sada diagramů, jejíž vrchol tvoří diagram kontextu a jejíž listy představují elementární funkce, popsané již jiným způsobem, než dekompozicí (pomocí tzv. minispecifikací – popisů operací, metod)
- Dokumentace řešení technikou „divide at impera“

Notace DFD (Yourdon)



Kontext ECO

Dokumentace analýzy pro ECO



Postup návrhu dle Yourdona (MSA - Modern Structured Analysis)

- Pro každou událost zaveď funkci
- Doplň vstupní a výstupní toky
- Přidej paměti pro spolupráci
- Dej vše dohromady a technikou „zprostředka do krajů“ vytvoř hierarchii DFD – abstrakcí dojdi až ke kontextovému diagramu a dekompozicí k elementárním funkcím

Návrh na základě diagramů datových toků (DFD)

- Z analytického popisu funkčního modelu vytváříme strukturované návrhové modely - např. Page-Jones: Practical Guide to Structured Systems Design.
- Co to jsou diagramy datových toků?
- Co to jsou diagramy struktury?
- Jak se diagramy datových toků převedou na diagramy struktury?

Diagramy struktury

Prvky:

- proces (funkce)
- vztah typu „volá“
- data jako argumenty
- data jako produkt volání

Převod DFD na DS

- Transakční analýza (výběr transakcí v sadě DFD)
- Transformační analýza (výběr šéfa, překreslení do DS)
- Přidání transakčního monitoru (nástroj pro výběr transakce)

Návrh na základě datových struktur

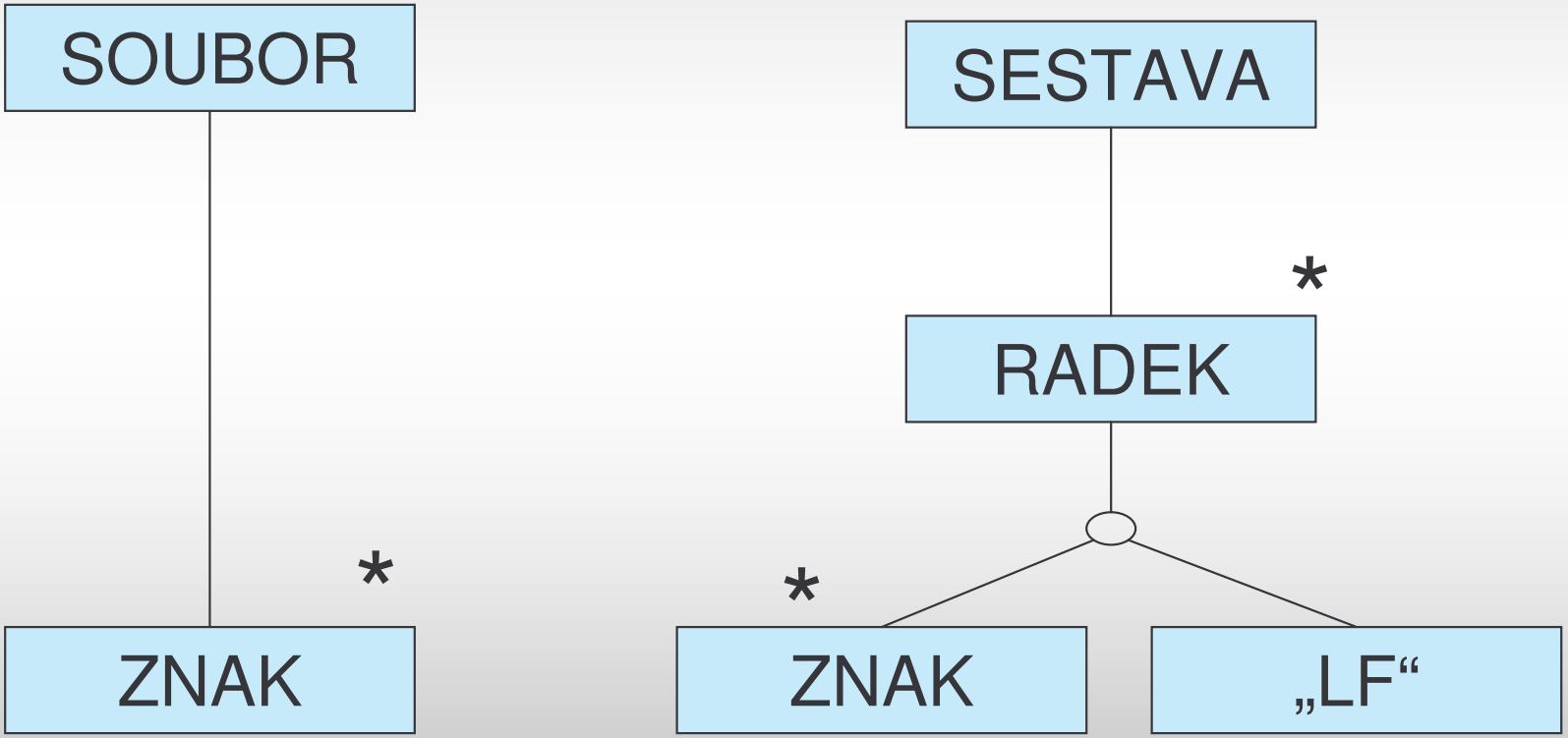
- Návrh na základě datových struktur (např. Jackson) – z analytických popisů datových struktur vytvoříme popisy programů

Vstupní a výstupní struktury

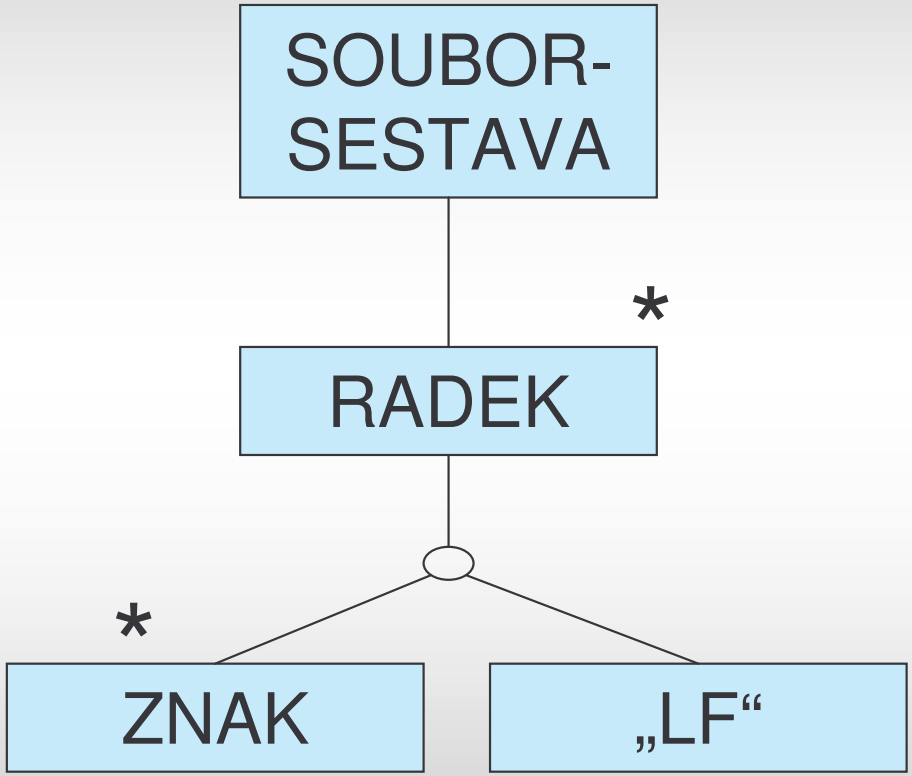
Prvky:

- element
- sekvence
- selekce
- iterace

Příklad: vstup - výstup



Odvozená struktura programu



Výstupní dokumenty návrhu

- Architektura systému (HW,SW)
- Popis implementace dat (logický datový model)
- Popis komponent (modulů)
- Projektová dokumentace návrhu

The End