X36SIN: Softwarové inženýrství

Karel Richta

2+2, kl.z.

Co to je „softwarové inženýrství“ ?

(definice IEEE 1993)

„Softwarové inženýrství je systematický, disciplinovaný a kvalifikovaný přístup k vývoji, tvorbě a údržbě softwaru.“

Proč se SI na FEL učí?

- Protože „softwarové inženýrství“ patří ke standardní výbavě absolventů universit.
- Absolventi FEL by neměli být pozadu a měli by se umět domluvit s absolventy jiných škol.
- Zdá se, že tato profese bude ještě dlouho žádaná.

Jak se SI na FEL učí?

- Projektová forma výuky
- Projekty do výuky patří (kromě klasické výuky)

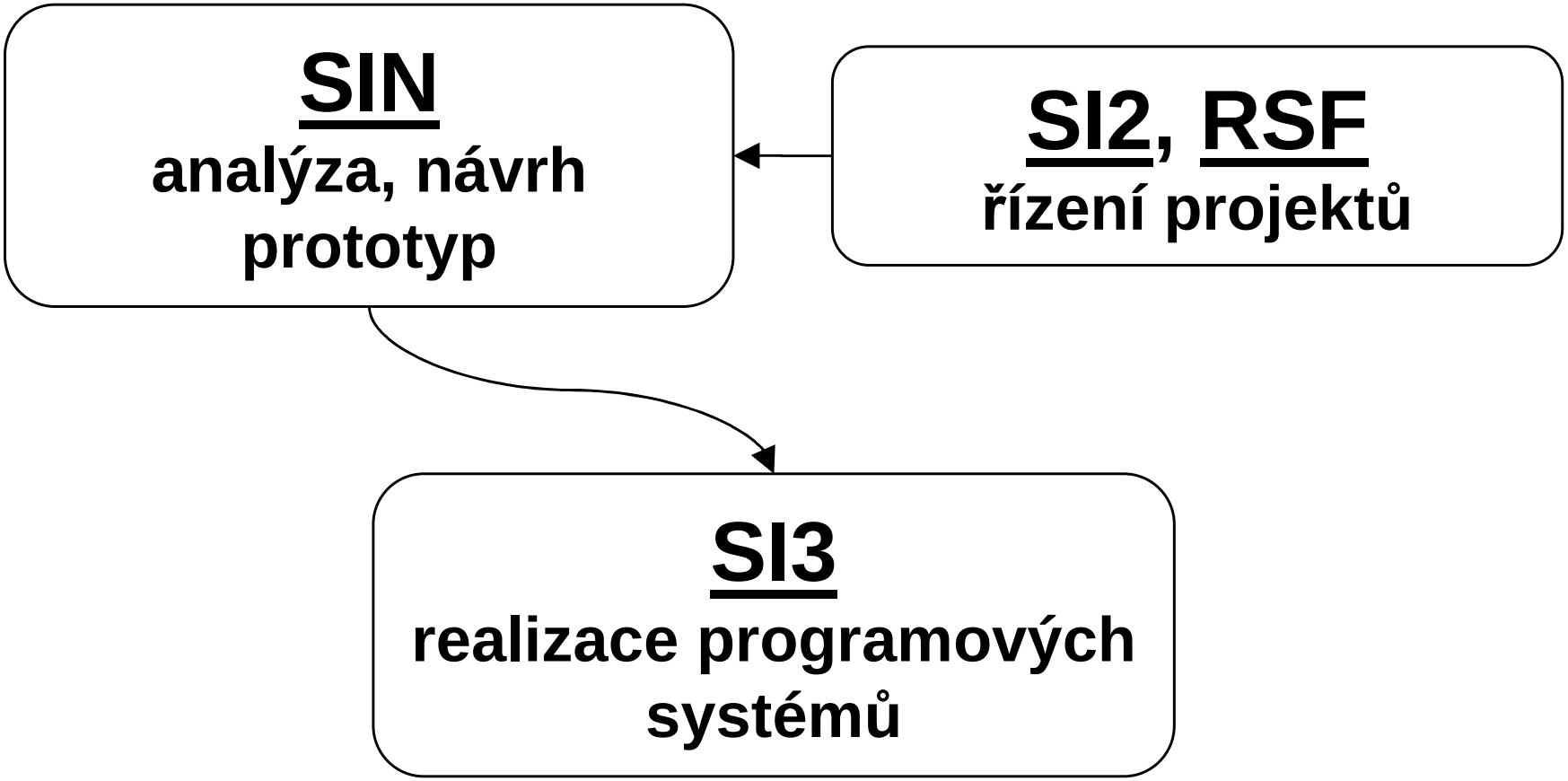
„Tradiční učení by nemělo být dominantní způsob výuky na universitě, která se chce zabývat výzkumem.“

MIT Educational Design Project, 1999

Smysl předmětů řady SI

- SIN je základní kurz softwarového inženýrství, který je určen pro pochopení discipliny, získání základních dovedností v analýze a návrhu, seznámení s používanými technikami a nástroji.
- V rámci cvičení se řeší menší projekty v týmech. Smyslem je vyzkoušet si práci na projektu řešeném v týmu. Výstupem projektů je předepsaná projektová dokumentace.
- Na předmět SIN (Softwarové inženýrství), který se zabývá zejména modelováním, navazuje předmět SI3 (Realizace programových systémů), který se zabývá realizací.
- Paralelně s SIN běží předmět SI2, resp. RSF (Řízení softwarových projektů), který se zabývá řízením projektů. Studenti SI2 řídí studenty SIN, studenti RSF provádějí audit projektů.

Návaznost předmětů SI



Co je to projekt?

- *„Projekt je dobře definovaná posloupnost činností, která je zaměřena na dosažení nejistého cíle, má určen začátek a konec, a je uskutečňována pomocí zdrojů – lidí a prostředků.“*
- *„Projekt je specifická nerutinní akce, která proto vyžaduje plánování.“*
- *„Čím složitější je projekt, tím více vyžaduje plánování.“*

Co je to softwarový projekt?

- *„Softwarový projekt je projekt, jehož cílem je vytvoření nebo využití programového díla.“*
- *„Při uskutečňování SW projektů se uplatní SW inženýři, kteří nabyli znalostí v předmětech SI (mimo jiné také v předmětu SIN).“*

Jaké jsou softwarové projekty v předmětu SIN?

- Větší projekt se za semestr nestihne (zkušenost).
- Projekt představuje hodně práce – projekty je třeba řešit v týmech (navíc se učíme týmovou práci a zodpovědnost).
- Projekt je třeba ukončit prezentací a obhajobou (prezentace práce představuje důležitou praktiku).
- Také metodika posouzení jiného projektu přináší nové poznatky.

Co z toho vyplývá?

- Studenti jsou rozděleni do týmů. Studenti SIN představují řešitele. Studenti předmětu SI2 fungují jako manažeři projektů, testéři kvality a pod. Studenti RSF provádějí audit.
- Vedoucí projektu (cvičící) může na základě osobnostního testu definovat strukturu týmu. Případně podle dohody volí pouze šéfa týmu a ostatní role v týmech definuje šéf.
- Tým řeší projekt, který si zvolí a dohodne, nebo který mu byl přidělen.
- Cvičení jsou projektová – konají se tak, jak to odpovídá potřebě projektů, příp. tak, jak stanoví cvičící dle potřeby projektů!

SIN – Jak získat známku

- Předmět SIN je zakončen klasifikovaným zápočtem.
- Zápočet uděluje cvičící - vedoucí projektu, příp. přednášející na základě doporučení vedoucího, pokud vedoucí nemá příslušné oprávnění.
- Zápočet se uděluje na základě akceptovaného projektu a písemných testů.
- Známka je určena kvalitou projektu (0 až 100 bodů), výsledky testů (20 až 50 bodů) a osobním hodnocením (-20 až 20 bodů).

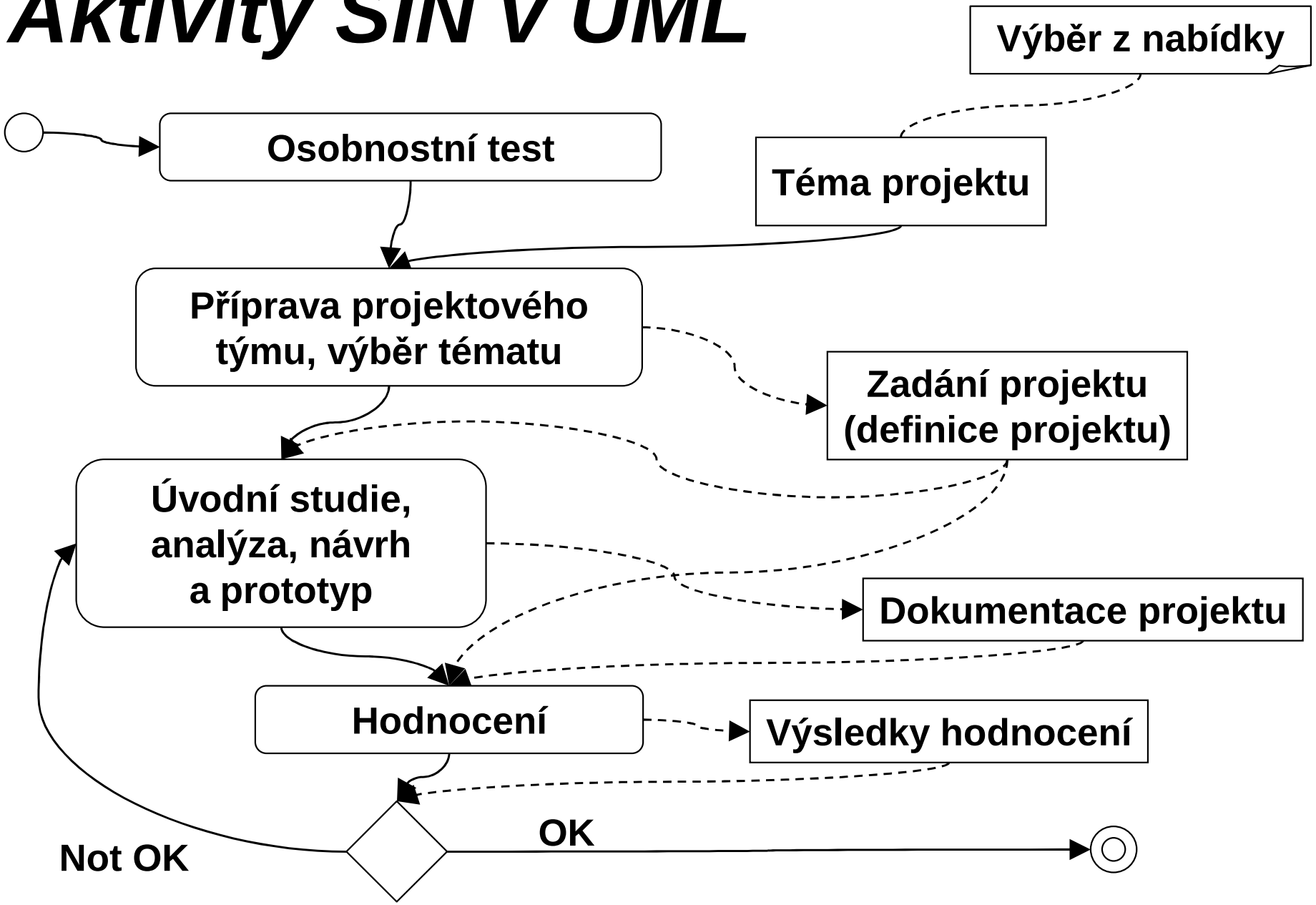
Pro získání zápočtu je třeba:

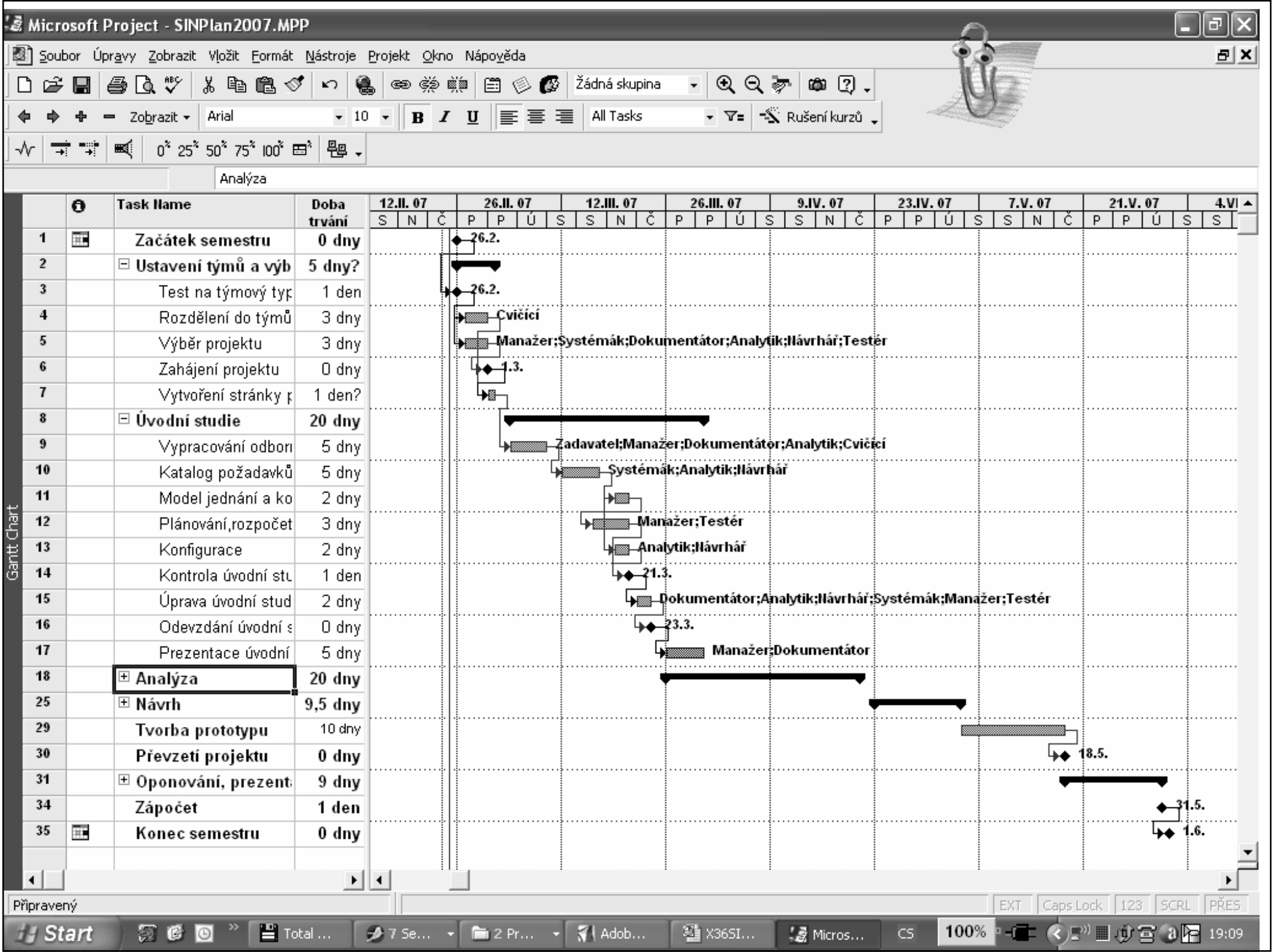
- Vytvořit projekt, který projde hodnocením (s kladným výsledkem). Projekty hodnotí:
  - Cvičící
  - Zadavatel
  - Nezávislý hodnotitel
  - Kdokoliv
- Ohodnotit jiný projekt
- Absolvovat 2 testy alespoň s minimálním hodnocením (10 bodů pro každý test).

Stupnice:

Body od	Body do	Známka
-20	69	-
70	89	3
90	109	2
110	160	1

Aktivity SIN v UML

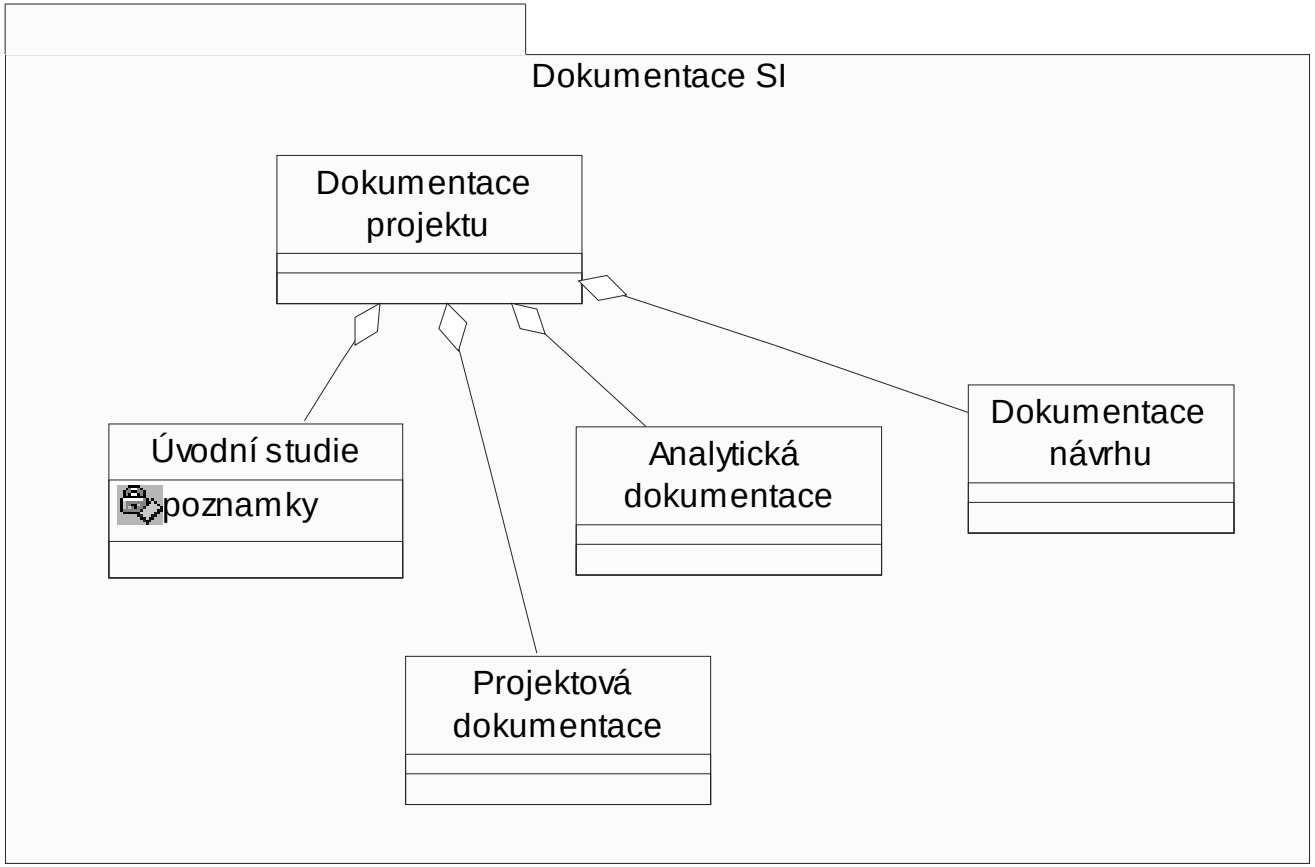



Microsoft Project - SINPlan2007.MPP
Soubor Úpravy Zobrazit Vložit Formát Nástroje Projekt Okno Nápověda
Žádná skupina
Zobrazit Arial 10 All Tasks Rušení kurzů
Analýza
Gantt Chart
Task Name
Doba trvání
1 Začátek semestru 0 dny
2 Ustavení týmů a výb 5 dny?
3 Test na týmový typ 1 den
4 Rozdělení do týmů 3 dny
5 Výběr projektu 3 dny
6 Zahájení projektu 0 dny
7 Vytvoření stránky p 1 den?
8 Úvodní studie 20 dny
9 Vypracování odbor 5 dny
10 Katalog požadavků 5 dny
11 Model jednání a ko 2 dny
12 Plánování,rozpočet 3 dny
13 Konfigurace 2 dny
14 Kontrola úvodní stu 1 den
15 Úprava úvodní stud 2 dny
16 Odevzdání úvodní s 0 dny
17 Prezentace úvodní 5 dny
18 Analýza 20 dny
25 Návrh 9,5 dny
29 Tvorba prototypu 10 dny
30 Převzetí projektu 0 dny
31 Oponování, prezent 9 dny
34 Zápočet 1 den
35 Konec semestru 0 dny
12.II. 07
26.II. 07
12.III. 07
26.III. 07
9.IV. 07
23.IV. 07
7.V. 07
21.V. 07
26.2.
Cvičící
Manažer;Systémák;Dokumentátor;Analytik;Návrhář;Tester
1.3.
Zadavatel;Manažer;Dokumentátor;Analytik;Cvičící
Systémák;Analytik;Návrhář
Manažer;Tester
Analytik;Návrhář
21.3.
Dokumentátor;Analytik;Návrhář;Systémák;Manažer;Tester
23.3.
Manažer;Dokumentátor
18.5.
31.5.
1.6.
Připravený
EXT Caps Lock 123 SCRL PŘES
Start
Total ...
7 Se...
2 Pr...
Adob...
X36SI...
Micros...
CS
100%
19:09

Co to znamená:

- Realizovat projekt znamená:
  - ***prostudovat téma***
  - ***vypracovat popis problému***
  - ***vytvořit potřebnou dokumentaci***
  - ***vytvořit prototyp produktu***
  - ***absolvovat požadované inspekce***
  - ***prezentovat projekt***
- Ohodnotit jiný projekt znamená:
  - ***být přítomen při inspekcích cizího projektu***
  - ***prostudovat dokumentaci cizího projektu***
  - ***sepsat posudek***

Obsah dokumentace SIN



Přibližná osnova cvičení:

- **Osobnostní test, rozdělení do týmů, témata projektů, výběr projektu**
- **Vypracování zadání projektu - odborný článek**
- **Vypracování úvodní studie (CIM), plán řešení**
- **Inspekce úvodní studie, příp. přepracování**
- **Vypracování analytické specifikace (PIM)**
- **Inspekce analytické specifikace**
- **Návrh (architektura, uživatelský vzhled, dekompozice na komponenty - PSM, plán realizace projektu)**
- **Vytvoření prototypu**
- **Prezentace**
- **Posouzení cizího projektu**

Důležité termíny

- 1.3. – zahájení projektu
- 23.3. – úvodní studie (CIM)
- 20.4. – analýza (PIM)
- 1.5. – návrh (PSM)
- 18.5 - prototyp, odevzdání projektu
- 31.5. - odevzdání posudku, zápočet

Stránky předmětu SIN:

https://service.felk.cvut.cz/courses/X36SIN

http://ocw.cvut.cz/moodle/

Literatura

- Tyto přednášky
- Richta, Sochor: Softwarové inženýrství I. Skripta FEL ČVUT, Praha 1998 (bohužel již vyprodána).
- Vrana, I., Richta, K.: Zásady a postupy při zavádění podnikových informačních systémů. Grada, Praha 2004.
- Arlow, Neustadt: UML a unifikovaný proces vývoje aplikací. Computer Press, Praha 2003.
- Schmuller: Myslíme v jazyce UML. Grada, Praha 2001.
- Kanisová, Müller: UML srozumitelně. Computer Press, Brno 2004.
- Šešera, Mičovský, Červeň: Datové modelování v příkladech. Grada 2001.

Další zdroje

- Bieliková: Softvérové inžinierstvo - Princípy a manažment. Skripta STU, Bratislava 2000.
- Král: Informační systémy. Science, Veletiny 1997.
- Sommerville: Software Engineering. Addison-Wesley, 2000.
- Pressman: Software Engineering. McGraw-Hill, 1994.
- Booch G., Rumbaugh J., Jacobson I.: The Unified Modeling Language User Guide, Addison Wesley Longman, 1999.
- Unified Modeling Language Specification, OMG, http://www.uml.org/
- http://www.rational.com/uml/
- http://interval.cz/

Osnova přednášek

- Úvod do softwarového inženýrství, plánování projektů
- Modelování požadavků (CIM)
- Analýza (PIM)
- Architektura SW, MDA
- Návrh (PSM)
- Návrhové vzory
- Metodiky
- Testování
- Další novinky pro vývoj: WS, SOA, EJB

Úvod do softwarového inženýrství

Úvodem trochu historie I.

- Termín „**software**“ zavedl v roce 1958 statistik John Tukey (také autor termínu „bit“).
- Za okamžik zrození termínu „**softwarové inženýrství**“ se obvykle považuje rok 1968, kdy NATO sponzoruje první konferenci s tímto názvem a na toto téma.
- V roce 1969 na ni navázala konference „Techniky softwarového inženýrství“.
- V roce 1972 vychází první časopis „Transactions on Software Engineering“ (IEEE Computer Society).
- V roce 1976 vytváří IEEE Computer Society první komisi, která by měla definovat obsah oboru „softwarové inženýrství“.

Proč to vůbec vzniklo?

- Něco bylo špatně ☹
- Počítačů přibývalo, přibývalo i softwarových projektů, ale ubývalo úspěšně dokončených projektů
- Někdy to došlo až na hranici únosnosti – software byl, nebo mohl být, příčinou havárií (např. Mariner I.)

Další historie II.

- Organizátoři první konference „**Softwarové inženýrství**“ v roce 1968 zvolili termín „softwarové inženýrství“úmyslně jako provokativní – naznačující, že produkce software musí přejít na jiné postupy a být podložena teoretickými disciplinami, podobně, jako je tomu u inženýrského přístupu v jiných oborech.
- Konala se ve Spolkové republice Německo ve známém středisku Garmisch-Partenkirchen a řídil ji profesor Bauer z Mnichovské techniky.
- Účastnilo se jí asi 50 odborníků z různých oblastí, z praxe i ze škol. Její účastníci formulovali směry, kterými by se výzkum v oboru SI měl ubírat.

Termín softwarové inženýrství

- ***„Softwarové inženýrství je disciplina, která se zabývá zavedením a používáním řádných inženýrských principů do tvorby software tak, abychom dosáhli ekonomické tvorby software, který je spolehlivý a pracuje účinně na dostupných výpočetních prostředcích.“***

Konference „Softwarové inženýrství 1968“

Termín softwarové inženýrství

- ***„Softwarové inženýrství je disciplina, která se zabývá zavedením a používáním řádných inženýrských principů do tvorby software tak, abychom dosáhli ekonomické tvorby software, který je spolehlivý a pracuje účinně na dostupných výpočetních prostředcích.“***

Konference „Softwarové inženýrství 1968“

Termín softwarové inženýrství

- ***„Softwarové inženýrství je disciplina, která se zabývá zavedením a používáním řádných inženýrských principů do tvorby software tak, abychom dosáhli ekonomické tvorby software, který je spolehlivý a pracuje účinně na dostupných výpočetních prostředcích.“***

Konference „Softwarové inženýrství 1968“

Další historie III.

- V roce 1979 vytváří Fletcher Buckley standard IEEE 370 pro vytváření plánů zajišťujících kvalitu software.
- V roce 1986 vzniká standard IEEE 1002, který definuje taxonomii softwarově inženýrských standardů.
- 1990 – 1995 vznikají standardy pro proces životního cyklu software (Standard for Software Life Cycle Processes) - standard ISO/IEC12207, vychází z DoD Std 498.
- V roce 1993 vznikají komise IEEE a ACM, které ústí do společného úsilí definovat softwarové inženýrství jako disciplinu.

Další historie IV.

- V roce 1998 společná komise IEEE a ACM definuje profesi softwarového inženýra.
- Nakonec v roce 2004 vzniká společný návrh „curricula“ pro výuku tohoto oboru, označovaného SE2004.
- Tím se završilo uznání softwarového inženýrství jako discipliny, podobně jako CS1991 završilo uznání informatiky.
- Softwarové inženýrství je definované jako standard IEEE 610.12.

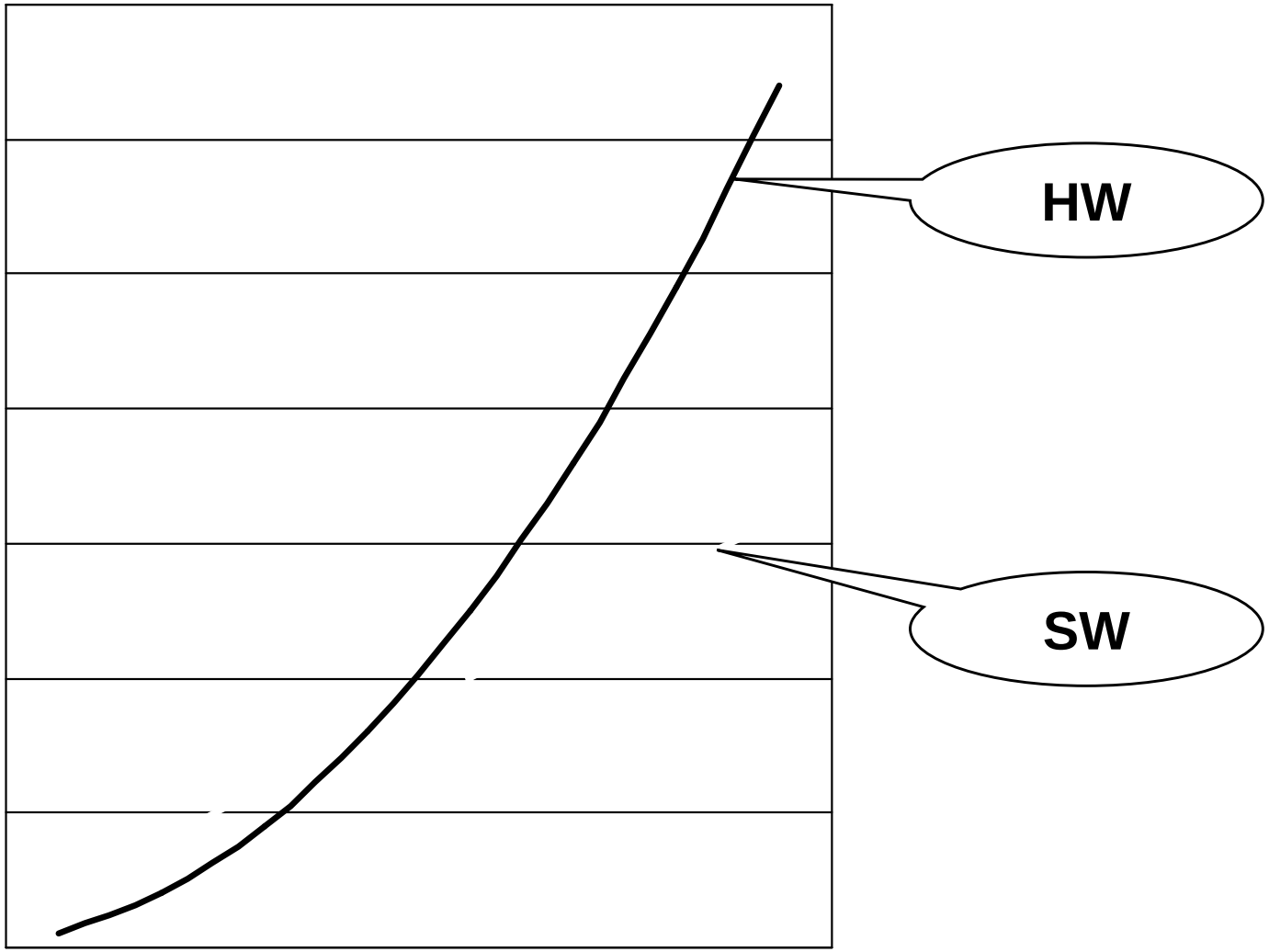
Příčina vzniku SI?

- Obvykle se říká, že to způsobil jev nazývaný „**softwarová krize**“.
- Dokud výkon počítačů nepřesáhl určitý rozměr, bylo možno se spolehnout na programátorské „hvězdy“.
- Často se počítače se využívaly pro vědecko-technické výpočty, kde záleželo spíše na preciznosti řešení, než na efektivitě tvorby programů.

Moorův zákon:

- ***„Výkon hardwaru vzrůstá zhruba dvakrát za dva roky“.***
  - Přestože sám autor prohlásil svou extrapolaci jako „pěkně divokou“, zákon zhruba platí dodnes.
  - Firma Intel nedávno zveřejnila výsledky výzkumné zprávy uvádějící, že Moorův zákon pravděpodobně přestane platit až kolem roku 2021 (křemík se dostane na hranici svých možností).

Softwarová krize



Edsger W. Dijkstra:

- ***„Hlavní příčinou softwarové krize byl nárůst výkonu hardware. Jinak řečeno, programování nemělo problémy, dokud neexistovaly počítače. Dokud jsme měli slabé počítače, mělo programování jen snesitelně těžké problémy. Nyní máme gigantické počítače a k nim gigantické problémy se softwarem“.***

Příčiny softwarové krize

- Příčinou softwarové krize vždy byl nesoulad mezi složitostí vytvářeného produktu a relativní nedostatečností a nezkušeností softwarové profese. Tento rozdíl způsobuje rozevírání nůžek a důsledkem pak jsou softwarové krize.

- *Pozn. Tento jev asi není omezen na software, ale zdá se mít univerzálnější platnost.*

Projevy softwarové krize

- Projekty překračují rozpočet.
- Projekty překračují čas.
- Software nemá dostatečnou kvalitu.
- Software neodpovídá požadavkům.
- Projekt není dobře řiditelný a software je obtížně udržovatelný.

Skončila softwarová krize?

- Při pohledu na předchozí body a současné projekty se zdá, že softwarová krize trvá stále.
- Svůj vliv zde mají též možnosti softwarových expertů, kteří jakkoli jsou geniální, mohou přímo mentálně obsáhnout jen určitý rozsah problémů.
- Řešení velkých projektů nelze proto nechat pouze na nich. Je nutno použít standardní techniku řešení obtížných problémů – „**rozděl a panuj**“ – velký problém je třeba rozdrobit na problémy menší.

Zkušenost (Brooks):

- ***„Přidání nových kapacit na zpožděný projekt prodlouží jeho řešení“.***

Tvorba software ↔ inženýrství

- Všechny tyto problémy související se softwarovou krizí vedly tedy nakonec k pokusu udělat z vývoje produkovaného nadšenci inženýrskou disciplinu.
- V 70-tých letech dochází k formulaci základních principů tohoto oboru.
- Vzniká také první generace nástrojů pro podporu této discipliny, které jsou označovány jako **CASE** (Computer Aided Software Engineering).

Co dělá inženýr?

- Než něco vyrábí, tak si to nejdříve nakreslí, namodeluje.
- K tomu potřebuje zavedenou notaci a vědecky podložené metody a postupy.
- Snaží se používat vhodné technologie tak, aby se dílo vytvořilo spolehlivě, efektivně a aby vydrželo.
- Srovnáme-li softwarového inženýra s inženýrem stavebním, pak stavební inženýr realizuje stavbu podle modelu, programátor programuje podle modelu.
- Model stavebnímu inženýrovi navrhl architekt (územní rozhodnutí, stavební povolení, realizace stavby), programátorovi softwarový architekt (úvodní studie, návrh architektury), analytik (konceptuální model) a návrhář (logický model).

První studijní programy

- Prvý inženýrský program byl vypsán již v roce 1979 na universitě v Seattlu, kde v roce 1982 udělili prvý titul v tomto oboru.
- Prvý bakalářský program v oboru SI vypsalo v roce 1996 vysoké učení technické v Rochesteru. Zpočátku byl odmítnut, ale později akreditaci získal v roce 2003 spolu s inženýrskou školou v Milwaukee.

Definice IEEE 610.12:

- ***„Softwarové inženýrství je aplikace systematického, disciplinovaného, kvantifikovatelného přístupu k vývoji, provozu a údržbě softwaru, tj. aplikace inženýrství na software. Také je to studium přístupů dle výše uvedeného.“***

Základní znalostní oblasti SI

- Správa požadavků (Software requirements)
- Softwarový návrh (Software design)
- Tvorba softwaru (Software construction)
- Testování softwaru (Software testing)
- Údržba softwaru (Software maintenance)
- Správa konfigurací (Software configuration management)
- Řízení vývoje (Software engineering management)
- Softwarový proces (Software engineering process)
- Nástroje a metody softwarového inženýrství (Software engineering tools and methods)
- Kvalita softwaru (Software quality)

Co nám SI přineslo?

Řadu novinek, namátkou některé:

- Softwarové profese a týmy.
- Modely životního cyklu.
- Oddělení správy dat od správy funkčnosti.
- Strukturované metody.
- Objektově-orientované metody.
- Komponentové metody.
- Nové programovací jazyky.
- Softwarové zápisy – unifikovaná notace UML.
- Metodiky tvorby softwaru.
- Plánování, metriky, organizace.

Co nám SI odneslo?

Pocit opravdových programátorů:

- Software vytvářejí specializovaní odborníci, kteří jediní znají postupy, nástroje, metody a techniky.
- Plánování a podobné hlouposti sem nepatří.
- Dokumentaci ať si pořizují ti, kteří neumějí číst programy přímo.

Neznámý programátor:

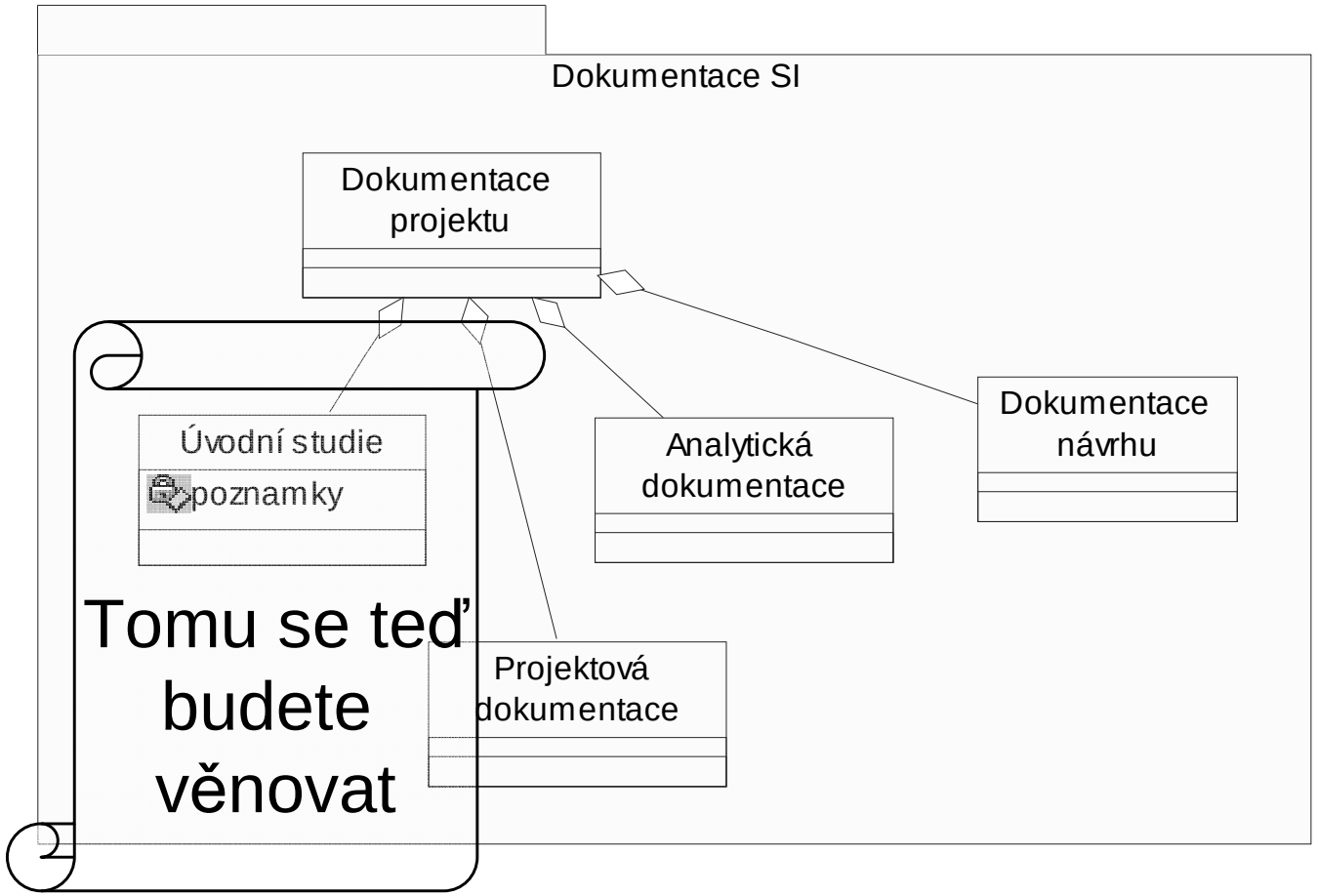
- *„Softwarové inženýrství znamená zavedení discipliny do volné tvorby software. Žádný opravdový programátor ho proto nemůže mít příliš v lásce, neboť jej nutí vytvářet nesmyslnou dokumentaci a další podobné artefakty.“*

Úvodní studie (feasibility study)

Odpověď na otázku ZDA a PROČ

Sběr požadavků na SW produkt

Obsah dokumentace SI



SIN: Zadávací dokumentace

- Měla by představovat zadání projektu pro řešitelský tým
- Měla by proto obsahovat deklaraci záměru a odborný článek
- Bude podkladem pro úvodní studii, kterou vypracují řešitelé – ta musí odpovědět na otázku “vyplatí se projekt řešit?”

Úvodní studie

Měla by odpovědět na otázku PROČ?

- Musí odpovědět na otázku: “vyplatí se projekt řešit?”
- Musí odpovědět na otázku: “je projekt uskutečnitelný?” (feasibility study)
- Musí proto vymezit hranici projektu
- Musí odpovědět na otázku: “kdo a co bude k řešení zapotřebí?”

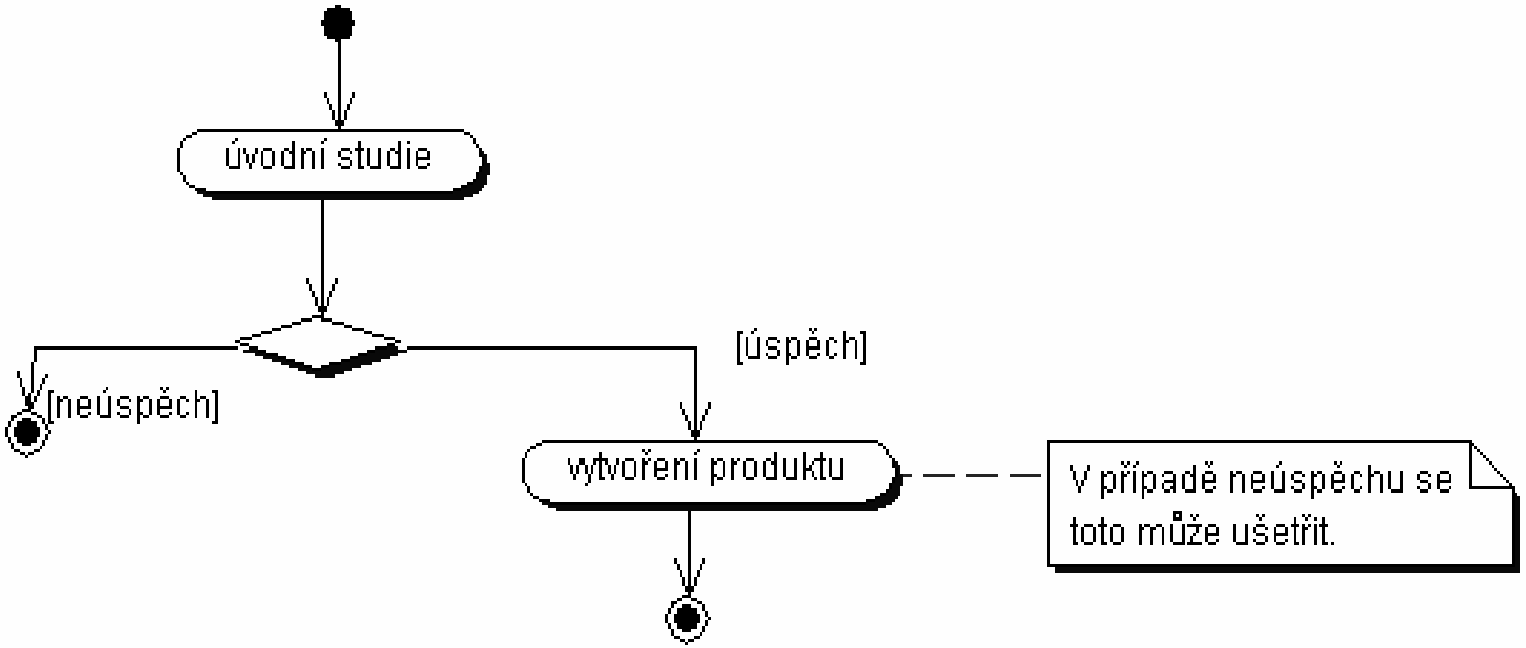
Vstupy úvodní studie

- Požadavky na systém
  - zadání projektu, deklarace záměru, vize projektu, odborný článek, tj. všechny dokumenty, které mají k řešenému problému nějaký vztah

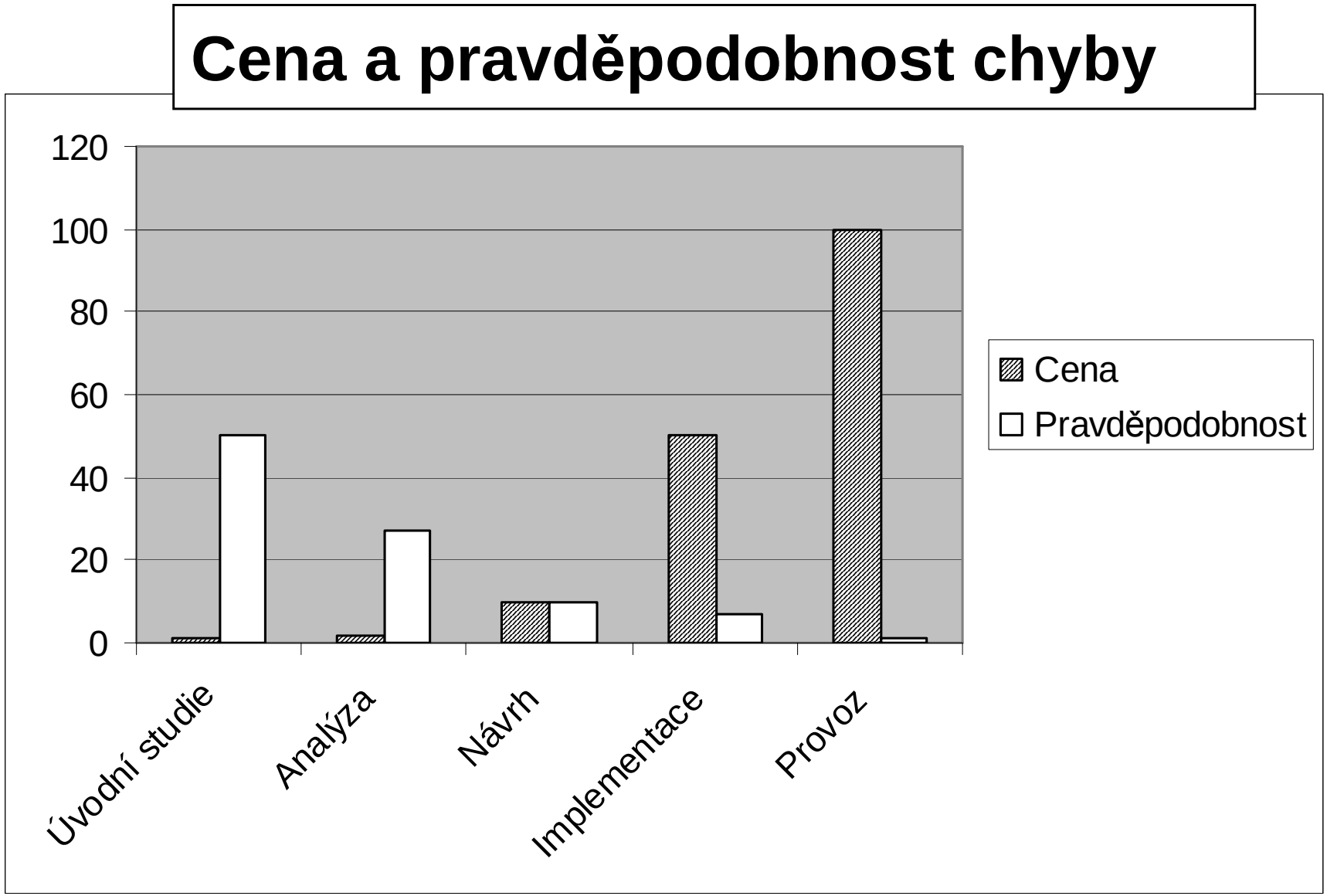
Výstupy úvodní studie

- Definice systému
  - katalog požadavků, definice hranice systému (diagram kontextu, model jednání), datový (pojmový) slovník, ...
- Projektová dokumentace
  - Řešitelský tým (funkce, zodpovědnosti).
  - Návrh řešení: HW, SW, komponenty.
  - Seznam úloh a harmonogram řešení.
  - Rozpočet: - cena HW, cena licencí na SW, cena vývoje SW a HW (COCOMO).

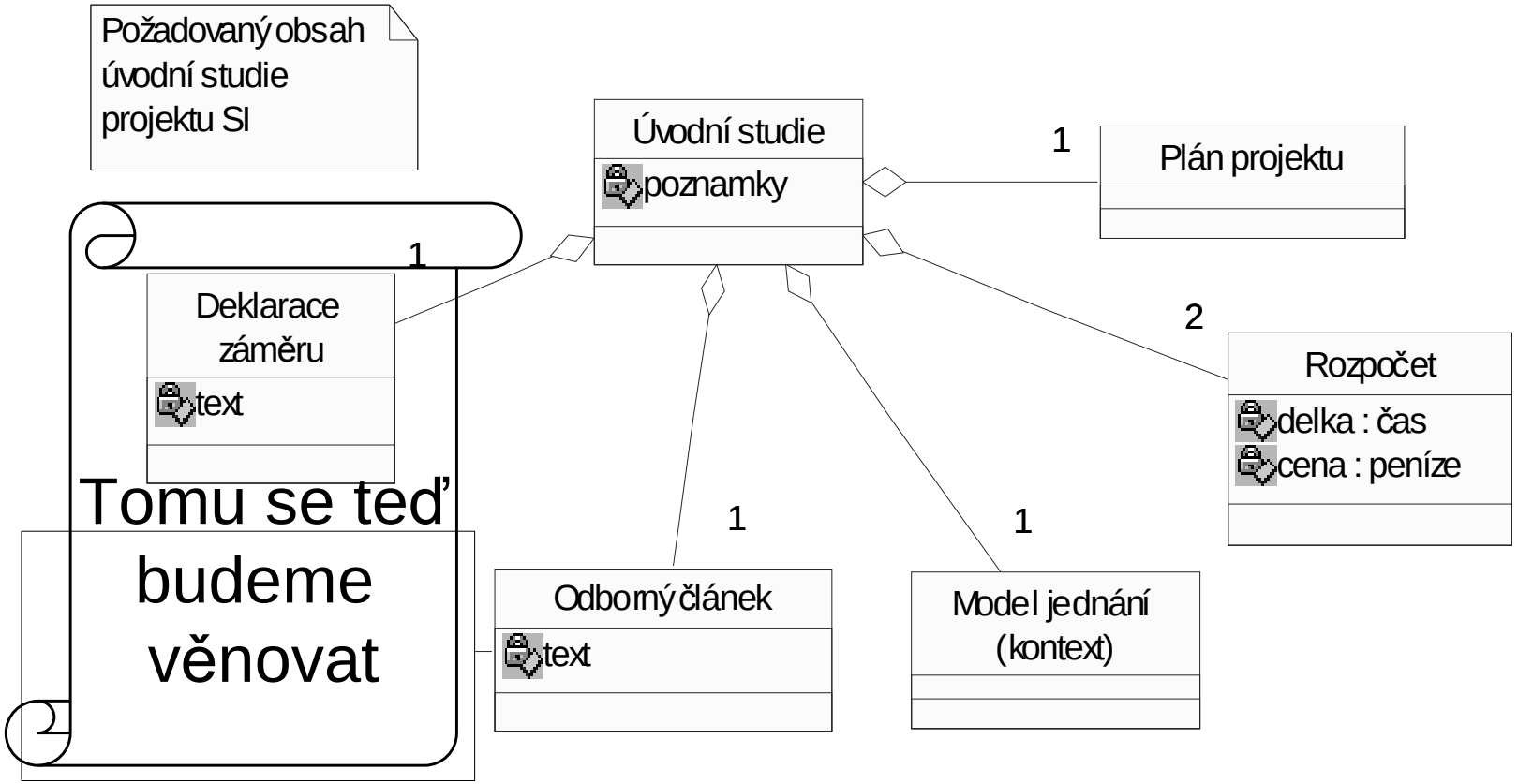
Úvodní studie může něco ušetřit



Má úvodní studie a analýza smysl?



Obsah úvodní studie



Deklarace záměru

- Krátký výstižný text se stručnými informacemi o projektu - jaké služby poskytuje, pro koho je určen a jaká předpokládá omezení.
- Měla by posloužit pro odpověď na otázku **“co ano, a co ne?”**.
- Je obvykle základem budoucího prospektu pro vytvořený produkt.

Deklarace záměru pro “Výtah”

(slouží pro odpověď na otázku “co ano, a co ne?”)

Systém “Výtah” slouží pro logické řízení obsluhy výtahu s jednou či více šachtami. Systém “Výtah” reaguje na požadavky uživatelů a dále registruje signalizaci ze spínačů v patrech a indikace ze senzorů přetížení. Systém “Výtah” ovládá klece výtahů pomocí povelů pro motory výtahů. Systém “Výtah” se nezabývá havarijním tlačítkem STOP, rovněž otevírání a zavírání dveří jde mimo systém (kvůli bezpečnosti).

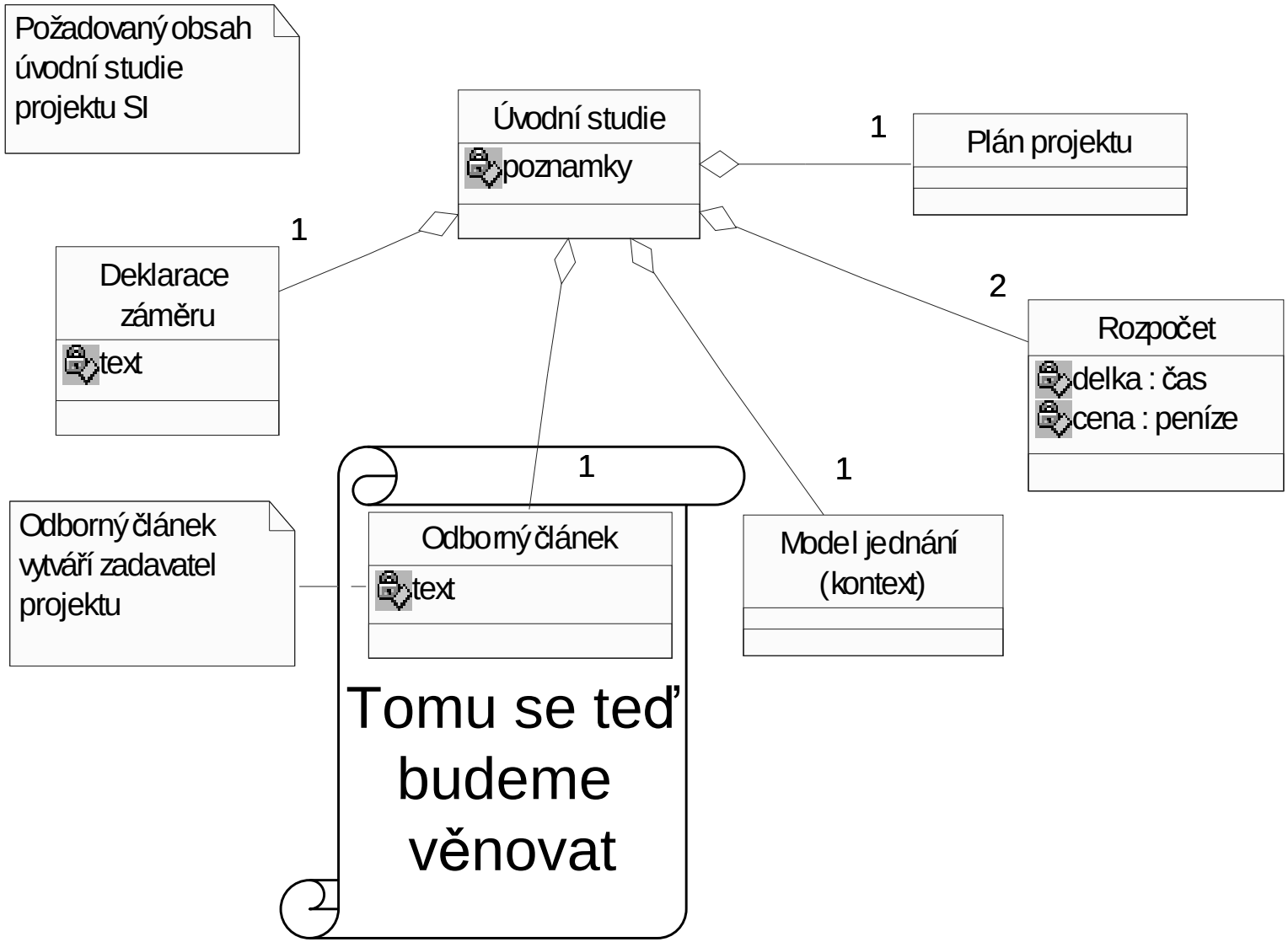
Příklad: Popis problematiky pro Systém pro přidělování úkolů (SPU)

- Ve firmě se přidělují úkoly
- Vykazuje se práce
- Je nutno vytvářet statistiky (kvůli rozhodování)
- Potřeba formalizace (kvůli automatizaci a spolehlivosti)
- Z toho plyne potřeba IS, který by to umožňoval
- Komerční systémy jsou ale drahé a složité

Deklarace záměru pro “SPU”

Systém SPU slouží pro zajištění podpory přidělování úkolů. Nadřízený pracovník přiděluje úkoly podřízeným pracovníkům a ti je plní. SPU poskytuje levné řešení tohoto problému, aby si je firmy s cca 100 zaměstnanci mohly dovolit. Jako komunikační médium využívá síť, což umožňuje kontrolu výkazů a zadávání úkolů i mimo firmu (např. ze služební cesty). Dále poskytuje různé statistiky činností.

Obsah úvodní studie



Odborný článek

- Všechny informace, které lze o projektu sehnat (články, interview, předpisy, …).
- Označení „**odborný článek**“ má vystihovat představu, že se jedná o texty v přirozeném jazyce, které sepsal odborník na řešenou problematiku. Informatik ji bude analyzovat a vytvoří popis přesnější. Někdy se nazývá **vize projektu**.
- Někdy se odborný článek nazývá „**katalog požadavků**“, ale my budeme takto označovat strukturovanou verzi odborného článku, kterou již tvoří informatik.

Odborný článek pro „Výtah“

(textový popis požadavků)

Systém “Výtah” slouží pro logické řízení obsluhy výtahu s jednou či více šachtami (předpokládají se 4 šachty a 40 úrovní). Systém zajišťuje efektivní plánování sběru a odvozu pasažérů mezi obsluhovanými patry podle požadavků (požadavek na přivolání výtahu pro jízdu směrem nahoru nebo dolů, požadavek na dopravení do určitého patra). Směr jízdy se nemění, dokud výtah nesplní objednávky v daném směru (výtah neví o pasažérech – neexistuje indikace prázdnosti klece). Přeplněný výtah nereaguje na výzvy (existuje indikace přetížení). Pro každou šachtu existuje samostatný motor ovládaný signály (povely UP, DOWN a STOP). Povel STOP způsobí zastavení výtahu v nejbližším patře v daném směru a otevření dveří výtahu (dveře se dají otevřít až v patře). Uvnitř klece je panel s tlačítky pater, indikace aktuální polohy a tlačítko STOP. Tlačítko STOP zabrání zavření dveří (jde mimo systém). Rovněž otevírání a zavírání dveří jde mimo systém (kvůli bezpečnosti). Příkazy pro systém jsou akceptovány až po zavření dveří. Operátor výtahu má k dispozici tlačítko ON/OFF, kterým zadává požadavek na zastavení pohybu výtahů.

Chyby v odborném článku

- Je příliš krátký a nepostihuje některé charakteristiky systému.
- Je příliš dlouhý a zabývá se problémy, které s popisem systému nesouvisí.
- Není z něj zřejmé, jaká data bude systém zpracovávat, jaké služby bude poskytovat, jak se budou vlastnosti systému měnit v čase či jako důsledek nějakých (popsaných) okolností.
- Neobsahuje některý požadavek.

Katalog požadavků

- Strukturovaná verze odborného článku.
- Odborný článek je předzpracován tak, aby tvořil strom požadavků.
- Požadavky jsou očíslovány a přes čísla se na ně lze odvolávat.

Katalog požadavků pro „Výtah“

(strukturovaný textový popis požadavků)

1. Systém “Výtah” slouží pro logické řízení obsluhy výtahu.

1.1 Výtah může mít jednu či více šachet (předpokládají se 4 šachty).

1.2 Výtah může mít dvě a více úrovní - pater (předpokládá se 40 úrovní).

2. Systém zajišťuje efektivní plánování sběru a odvozu pasažérů mezi obsluhovanými patry podle požadavků.

2.1 Požadavek na přivolání výtahu pro jízdu směrem nahoru nebo dolů (vzniká v patře).

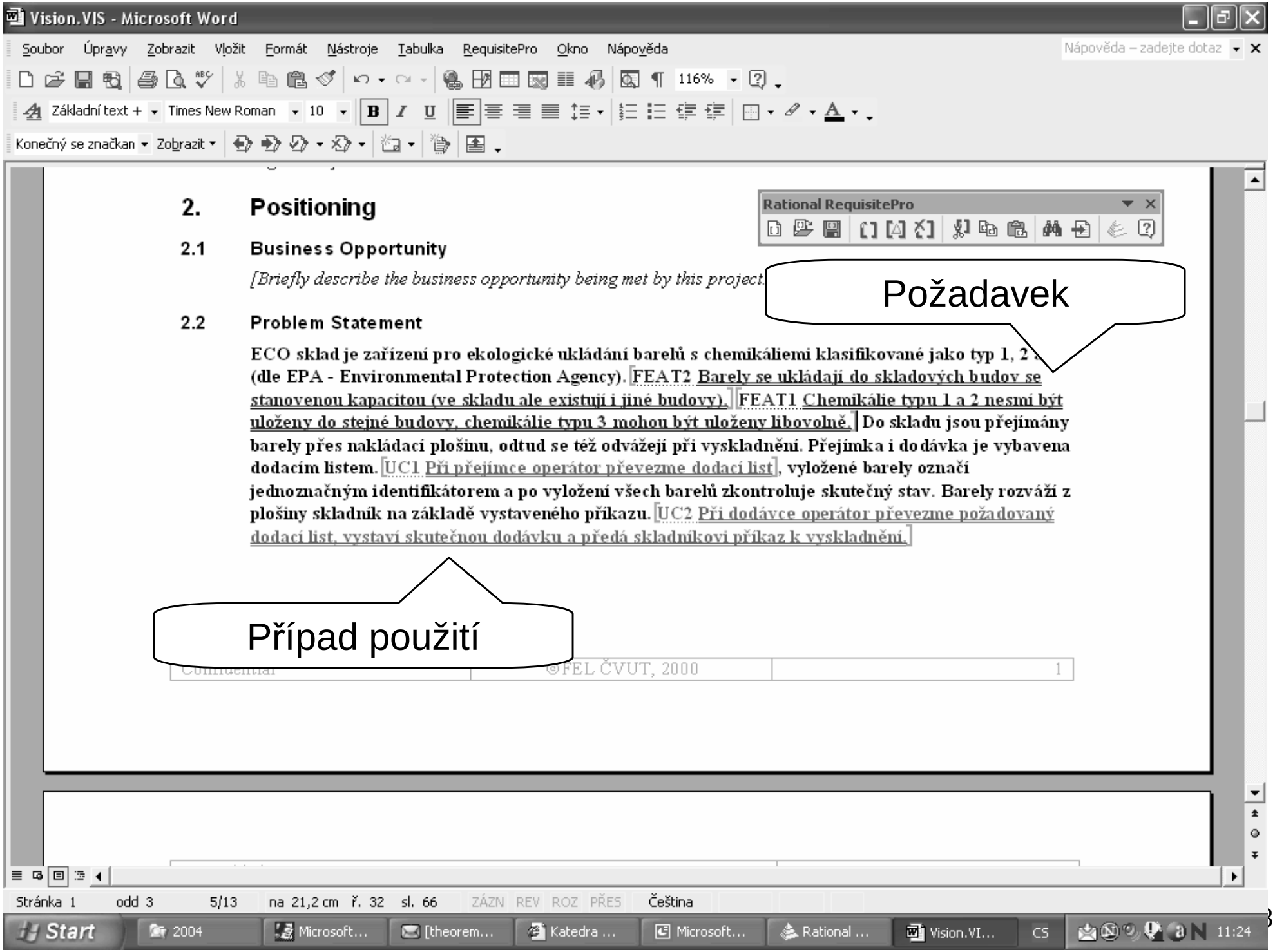
2.2 Požadavek na dopravení do určitého patra (vzniká v kleci výtahu).

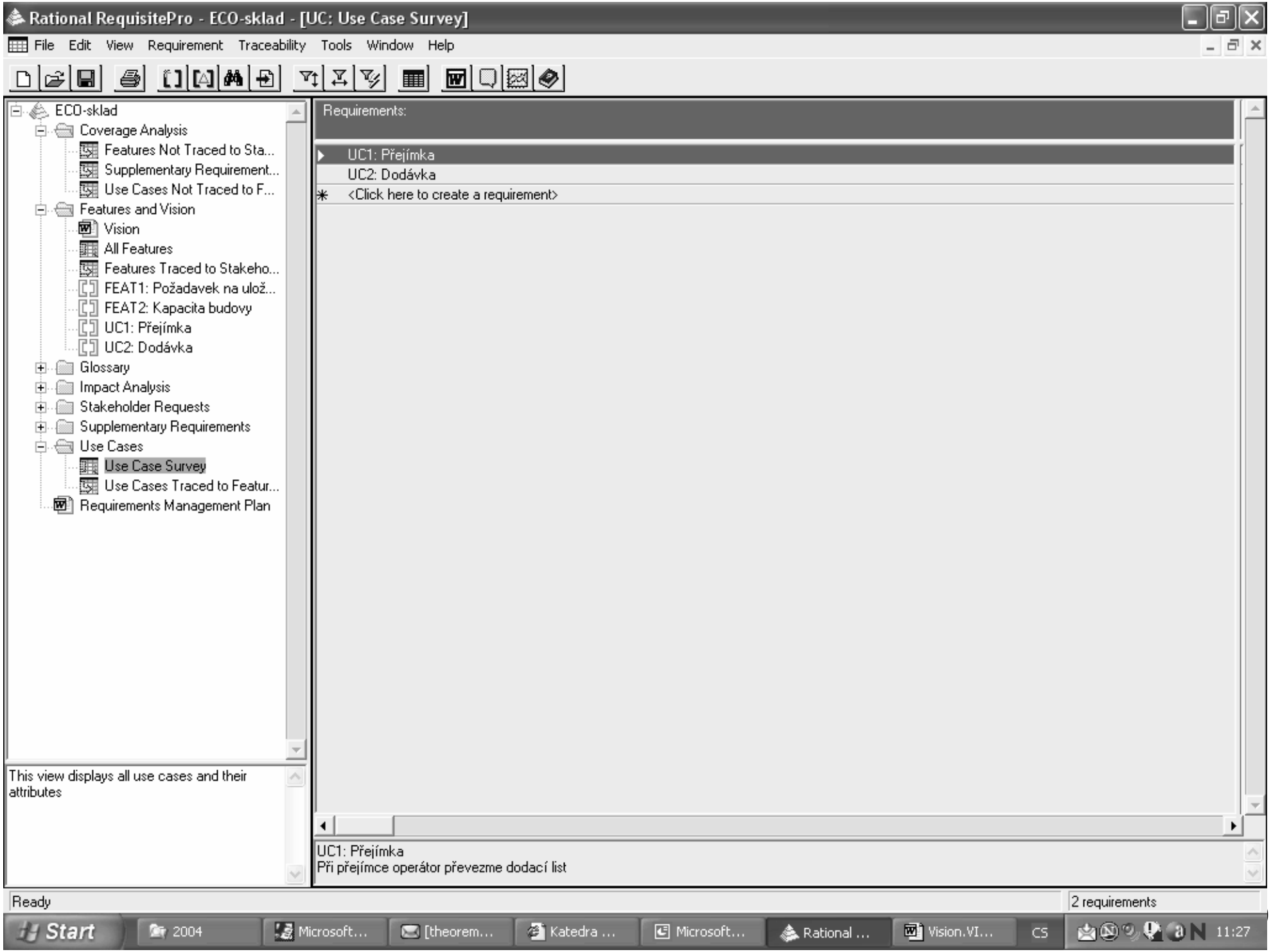
3. Směr jízdy se nemění, dokud výtah nesplní objednávky v daném směru (výtah neví o pasažérech – neexistuje indikace prázdnosti klece).

4. Přeplněný výtah nereaguje na výzvy (existuje indikace přetížení).

……

n. Pravděpodobnost chyby by měla být menší než 1 chyba za 10 let (příklad nefunkčního požadavku, který ale musíme též evidovat).


Vision.VIS - Microsoft Word
Soubor Úpravy Zobrazit Vložit Formát Nástroje Tabulka RequisitePro Okno Nápověda
Nápověda – zadejte dotaz
Základní text + Times New Roman 10
Konečný se značkan Zobrazit
Rational RequisitePro
2. Positioning
2.1 Business Opportunity
[Briefly describe the business opportunity being met by this project.
2.2 Problem Statement
ECO sklad je zařízení pro ekologické ukládání barelů s chemikáliemi klasifikované jako typ 1, 2 a 3 (dle EPA - Environmental Protection Agency). FEAT2 Barely se ukládají do skladových budov se stanovenou kapacitou (ve skladu ale existují i jiné budovy). FEAT1 Chemikálie typu 1 a 2 nesmí být uloženy do stejné budovy, chemikálie typu 3 mohou být uloženy libovolně. Do skladu jsou přejímány barely přes nakládací plošinu, odtud se též odvážejí při vyskladnění. Přejímka i dodávka je vybavena dodacím listem. UC1 Při přejímce operátor převezme dodací list, vyložené barely označí jednoznačným identifikátorem a po vyložení všech barelů zkontroluje skutečný stav. Barely rozváží z plošiny skladník na základě vystaveného příkazu. UC2 Při dodávce operátor převezme požadovaný dodací list, vystaví skutečnou dodávku a předá skladníkovi příkaz k vyskladnění.
Požadavek
Případ použití
Confidential
©FEL ČVUT, 2000
1
Stránka 1 odd 3 5/13 na 21,2 cm ř. 32 sl. 66 ZÁZN REV ROZ PŘES Čeština
Start
2004
Microsoft...
[theorem...
Katedra ...
Microsoft...
Rational ...
Vision.VI...
CS
11:24


Rational RequisitePro - ECO-sklad - [UC: Use Case Survey]
File Edit View Requirement Traceability Tools Window Help
ECO-sklad
Coverage Analysis
Features Not Traced to Sta...
Supplementary Requirement...
Use Cases Not Traced to F...
Features and Vision
Vision
All Features
Features Traced to Stakeho...
FEAT1: Požadavek na ulož...
FEAT2: Kapacita budovy
UC1: Přejímka
UC2: Dodávka
Glossary
Impact Analysis
Stakeholder Requests
Supplementary Requirements
Use Cases
Use Case Survey
Use Cases Traced to Featur...
Requirements Management Plan
This view displays all use cases and their attributes
Requirements:
UC1: Přejímka
UC2: Dodávka
<Click here to create a requirement>
UC1: Přejímka
Při přejímce operátor převezme dodací list
Ready
2 requirements
Start
2004
Microsoft...
[theorem...
Katedra ...
Microsoft...
Rational ...
Vision.VI...
CS
11:27

Význam termínů

- Všechny termíny v dokumentaci by měly být zaneseny ve **významovém slovníku** (technický termín je **datový slovník – Data Dictionary**).
- Je to proto, aby se termíny používané v dokumentaci interpretovaly stejně – např. „formulář 501“ může být termín běžný pro zadavatele, ale rozumět mu musí i řešitel - objednávka je obecně srozumitelný pojem, co ale má skutečně obsahovat?

Datový slovník (dle Yourdona)

Metaznak	Význam	Příklad	Jak se to čte
=	skládá se z	X =Y	X se skládá z Y
+	a	Z = X + Y	Z se skládá z X a Y
( )	může chybět	Z = X + (Y)	Z se skládá z X a příp. Z Y
{ }	opakování	Z = { X }	Z se skládá z několika X
[ ]	jeden z možných	Z = [X | Y]	Z se skládá buď z X nebo z Y (implicitní položku lze podtrhnout)
**	komentář	*toto je komentář*	
@	klíčová položka	Z = @X+Y	Z se skládá z X a Y, kde X je klíčová položka
@<číslo>	část složeného klíče	Z = @1X+@2Y	X a Y tvoří klíč (v tomto pořadí)

Datový slovník pro “Výtah”

(významy použitých termínů)

šachta = celé číslo *rozsah 1..4*
patro = celé číslo *rozsah 1..40*
tlačítko přivolání = patro + směr
směr = [UP | DOWN]
tlačítko patra = šachta + patro
stisk tlačítka = [tlačítko patra | tlačítko přivolání]
signalizace spínače patra = šachta + patro
signalizace přetížení = šachta
řídicí povel pro motor = šachta + povel
povel = [UP | DOWN | STOP]
indikace patra = šachta + patro
indikace přivolání = patro + směr
indikace = [indikace patra | indikace přivolání]

Př.: Rozhovor na téma „jméno“

- Člověk: My lidé se nazýváme jmény.
- Marťan: A co je to jméno?
- Člověk: Jméno je posloupnost znaků.
- Marťan: Takže „a1234“ je správné jméno?
- Člověk: Ve jménech používáme pouze písmena.
- Marťan: Takže „X“ je správné jméno?
- Člověk: Teoreticky ano, ale obvykle používáme jména, která obsahují nejméně dvě písmena. Navíc mají lidé většinou více jmen – jméno je rozděleno na části, kterým se říká „první jméno“, „příjmení“, apod.
- Marťan: …?

Datový slovník pro “Jméno”

celé jméno = { tituly před } + první jméno + { prostřední jméno } +
příjmení + { čárka + tituly za }

tituly před = [pan | paní | slečna | ing. | RNDr. | doc. | prof. | …]

první jméno = jméno

příjmení = jméno

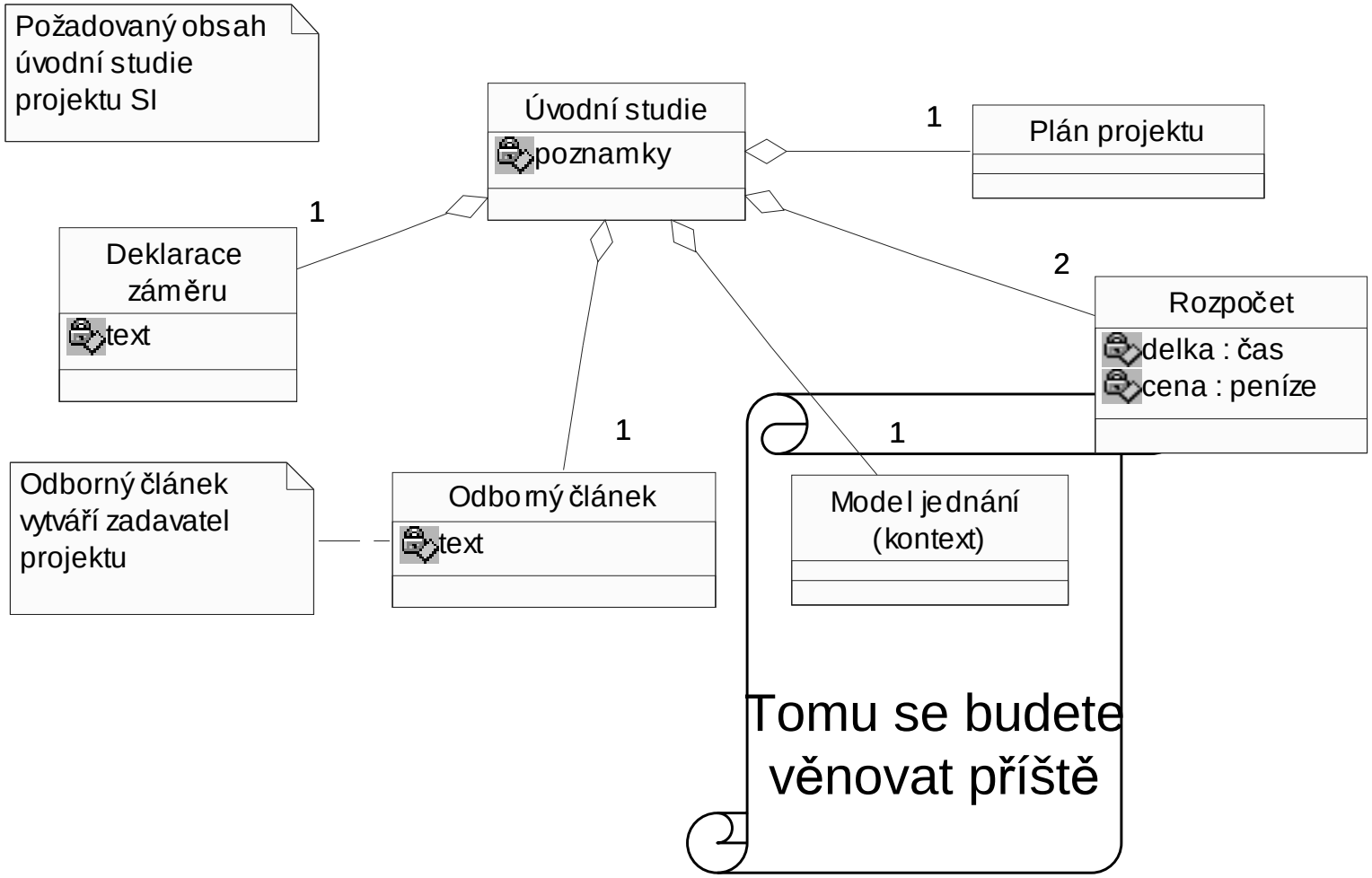
prostřední jméno = jméno

jméno = velké písmeno + 1{ malé písmeno }
písmeno = [malé písmeno | velké písmeno]
malé písmeno = [a | á | b | c | …] *písmena lokální abecedy*
velké písmeno = [A | Á | B | C | …] *písmena lokální abecedy*

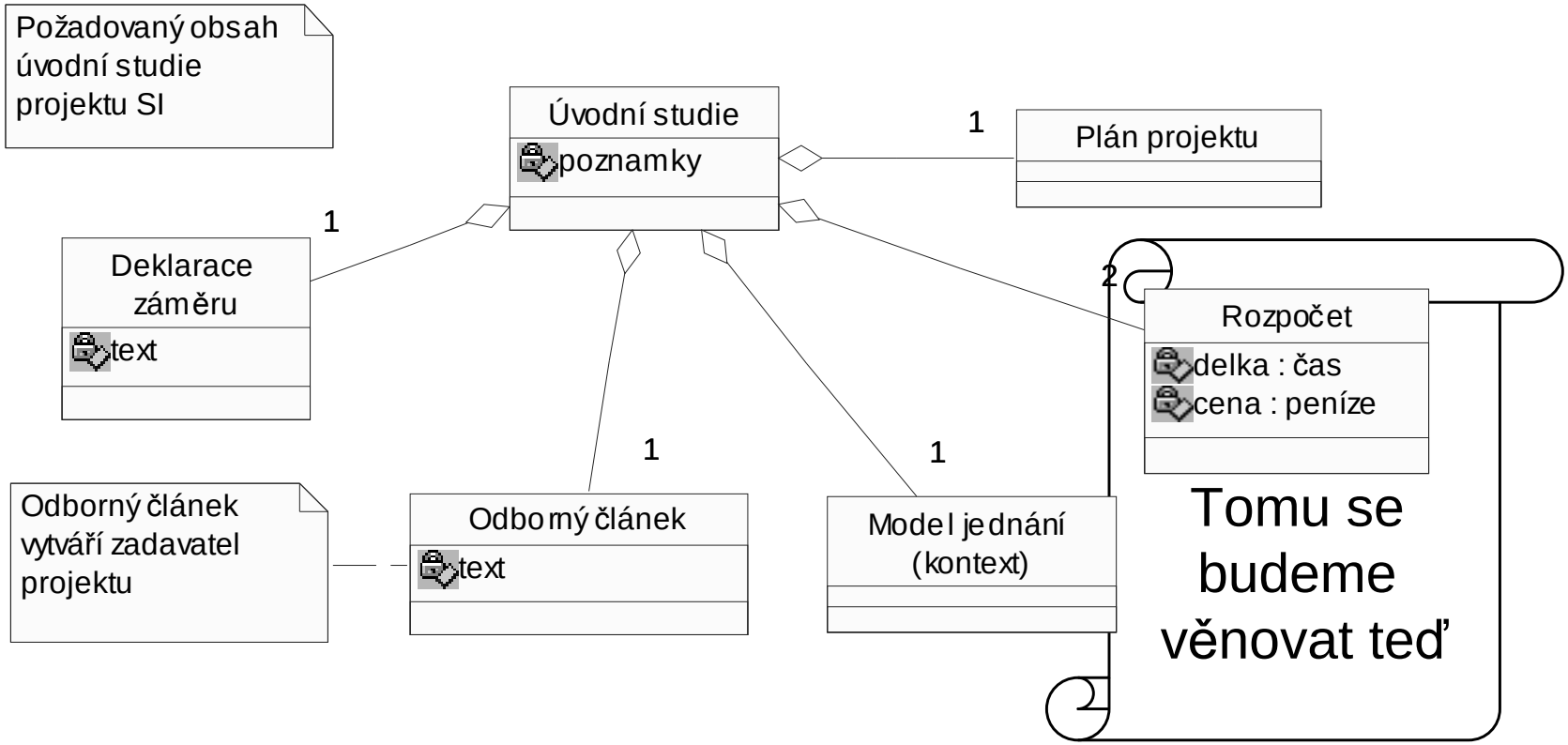
čárka = ,

tituly za = [CSc. | PhD. | DrSc. | prom.mat. | …]

Obsah úvodní studie



Obsah úvodní studie

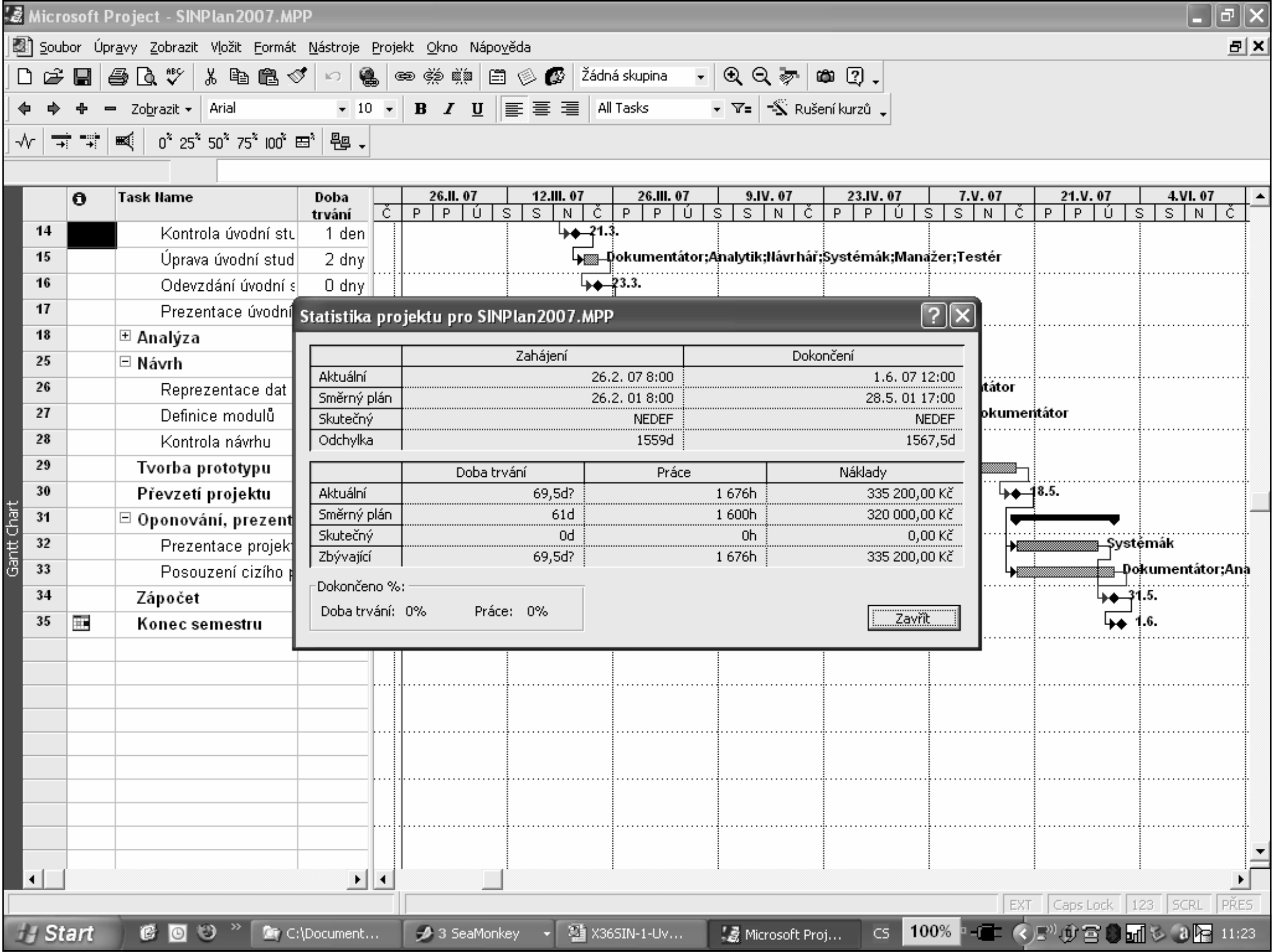


Odhad nákladů na projekt

- Odhad na základě zkušenosti z minula
  - již jsme něco podobného řešili a údaje o nákladech jsme si schovali
- Odhad na základě dekompozice problému na odhadnutelné složky
  - klasické řešení problému technikou „divide-et-impera“
- Odhady na základě výpočtu z odhadu rozsahu
  - odhad se obvykle řídí odhadem rozsahu kódu (LOC, KLOC, FP)

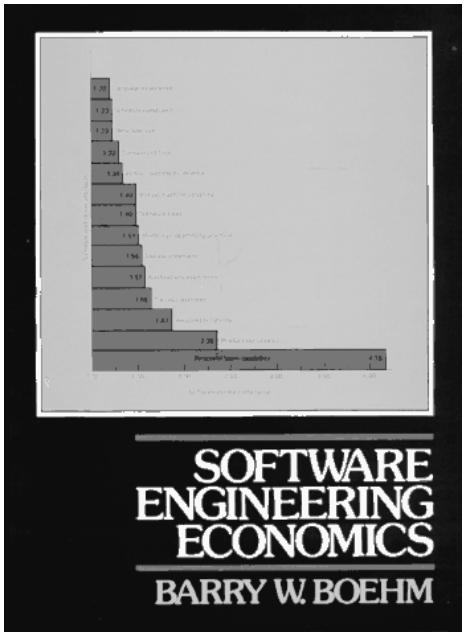
Náklady pomocí dekompozice na úlohy

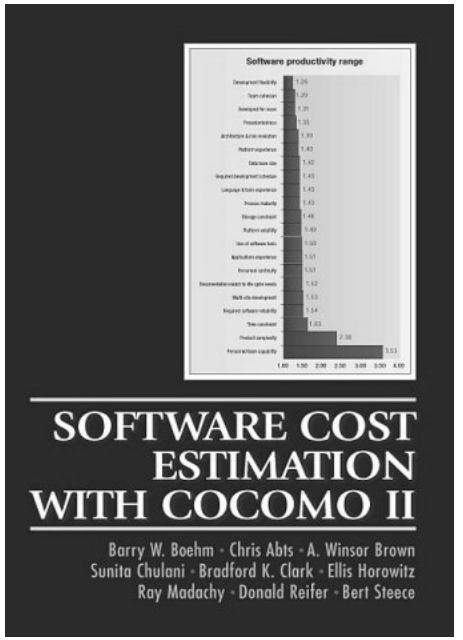
- Cena projektu = $\sum$ cen úloh
- Cena úlohy =
  - fixní náklady + cena za použití zdrojů
- Cena za použití zdroje =
  - odměna za práci v normální pracovní době +
  - odměna za práci přesčas +
  - fixní náklady na použití zdroje
- Práce = jednotky * délka
  - Práce je dána součinem počtu jednotek zdroje, které na úloze pracují a délky úlohy


Microsoft Project - SINPlan2007.MPP
Soubor Úpravy Zobrazit Vložit Formát Nástroje Projekt Okno Nápověda
Žádná skupina
Zobrazit Arial 10 All Tasks Rušení kurzů
Task Name
Doba trvání
Gantt Chart
14 Kontrola úvodní stu 1 den
15 Úprava úvodní stud 2 dny
16 Odevzdání úvodní s 0 dny
17 Prezentace úvodní
18 Analýza
25 Návrh
26 Reprezentace dat
27 Definice modulů
28 Kontrola návrhu
29 Tvorba prototypu
30 Převzetí projektu
31 Oponování, prezent
32 Prezentace projekt
33 Posouzení cizího p
34 Zápočet
35 Konec semestru
26.II. 07
12.III. 07
26.III. 07
9.IV. 07
23.IV. 07
7.V. 07
21.V. 07
4.VI. 07
21.3.
Dokumentátor;Analytik;Návrhář;Systémák;Manažer;Tester
23.3.
18.5.
Systémák
Dokumentátor;Ana
31.5.
1.6.
Statistika projektu pro SINPlan2007.MPP
Zahájení
Dokončení
Aktuální 26.2. 07 8:00 1.6. 07 12:00
Směrný plán 26.2. 01 8:00 28.5. 01 17:00
Skutečný NEDEF NEDEF
Odchylka 1559d 1567,5d
Doba trvání
Práce
Náklady
Aktuální 69,5d? 1 676h 335 200,00 Kč
Směrný plán 61d 1 600h 320 000,00 Kč
Skutečný 0d 0h 0,00 Kč
Zbývající 69,5d? 1 676h 335 200,00 Kč
Dokončeno %:
Doba trvání: 0%
Práce: 0%
Zavřít
EXT Caps Lock 123 SCRL PŘES
Start
C:\Document...
3 SeaMonkey
X36SIN-1-Uv...
Microsoft Proj...
CS
100%
11:23

Jiné metody odhadu

- COCOMO (Constructive Cost Model) - Barry Boehm





- http://sunset.usc.edu/research/COCOMOII/

Odhad rozpočtu dle COCOMO

- Vstup: Rozsah produktu v KLOC (KLines of Code)
- Náročnost = 2.94 * (Rozsah) $^{0.91}$
  - (udává se v člověko-měsících)
- Čas = 3.97 * (Náročnost) $^{0.28}$
- Cena = Čas * Plat
- Koeficienty se mění dle typu projektu a korekcí (cca 0.5 ÷ 2.0)

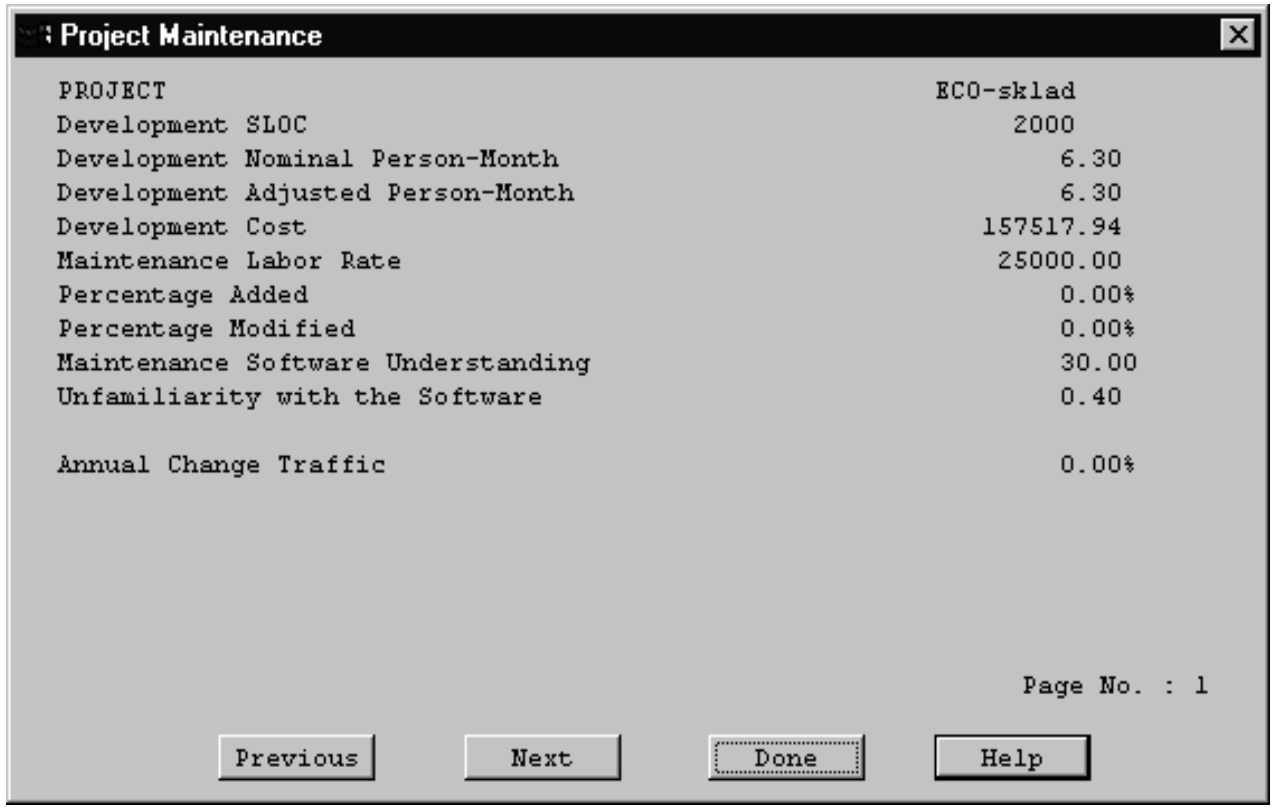
Příklad: Sestavovací program

- **Velikost: 32 KLOC (KDSI)**
- **Náročnost (Effort): 121.77 ČM**
- **Čas: 15.50 měsíců**
- **Lidí: 7.856**

(organic mode – jednodušší známé projekty, spočteno přes COCOMO kalkulátor)

http://sunset.usc.edu/research/COCOMOII/cocomo81_pgm/cocomo81.html

Příklad výpočtu (COCOMO II)



Př.: Náklady na projekt SPU

- Náklady dle MS-Project: 428.640,-
- Náklady dle COCOMO II: 443.000,-

  (člověkohodina je 200 Kč, parametry:

  SIZE = 5000, MODE = 1.05, DATA = 0.94, CPLX = 0.85, tj.

  náročnost = 13.86 člověko-měsíců,

  potřebný čas = 6.79 měsíce)

Zdroj: Projekt SPU

Výnosy pro projekt SPU

- Tento produkt by měl být distribuován jako krabicové řešení. Budeme-li předpokládat že se nám projekt podaří nasadit do 10ti firem (+ do jedné zdarma jako reklama), tak odhadovaná cena pro jednu kopii by měla být 50 000 Kč.
- Výnosy: 10 x 50.000,-

  tj. 450.000,-

Zdroj: Projekt SPU

Zhodnocení pro projekt SPU

- Obě metody odhadly cenu projektu na přibližně 450 000 Kč.
- Myslíme si, že cena produktu 50 000 Kč (+ DPH) by pro koncového zákazníka (firma s deseti až sto zákazníky) by mohla být přijatelná. Domníváme se, že nejméně 10 firem by si produkt zakoupilo, každý další prodej by znamenal zisk.
- Proto navrhujeme do projektu investovat.

Zdroj: Projekt SPU

Karnerova metoda odhadu

- **Jiná metoda odhadu nákladů, založená na modelu jednání**
- **Spočítejte aktéry, každého aktéra zařaďte do kategorie:**
  - **jednoduchý (např. jiný systém komunikující přes API) – váha 1,**
  - **střední (např. uživatel se znakovým terminálem nebo jiný systém komunikující přes TCP/IP) – váha 2, nebo**
  - **složitý (např. osoba komunikující přes GUI nebo Web) – váha 3.**
- **Sečtěte váhy všech aktérů a získáte neupravenou váhu aktérů - UAW (Unadjusted Actor Weights).**

Karnerova metoda odhadu (pokr.)

- **Rozdělte případy použití do kategorií podle odhadu počtu potřebných transakcí:**
  - **jednoduchý (méně než 4 transakce) – váha 5,**
  - **střední (4-7 transakcí) – váha 10, nebo**
  - **složitý (více než 7 transakcí) – váha 15.**
- **Sečtěte váhy všech případů užití a získáte neupravenou váhu případů užití - UUCW (Unadjusted Use Case Weight).**

Karnerova metoda odhadu (pokr.)

- **Sečtěte obě váhy a získáte neupravenou váhu modelu jednání – UUCP (Unadjusted Use Case Points)**
- **Adjustujte takto spočtenou váhu technickými faktory (TCF) a faktory prostředí (EF). Faktor má hodnotu 0 (žádný vliv) až 5 (silný vliv). Koeficient se spočítá:**

  TCF = 0.6+0.01*Technický faktor

  EF = 1.4-0.03*Faktor prostředí
- **Získáte tak upravenou váhu modelu jednání – UCP (Use Case Points)**

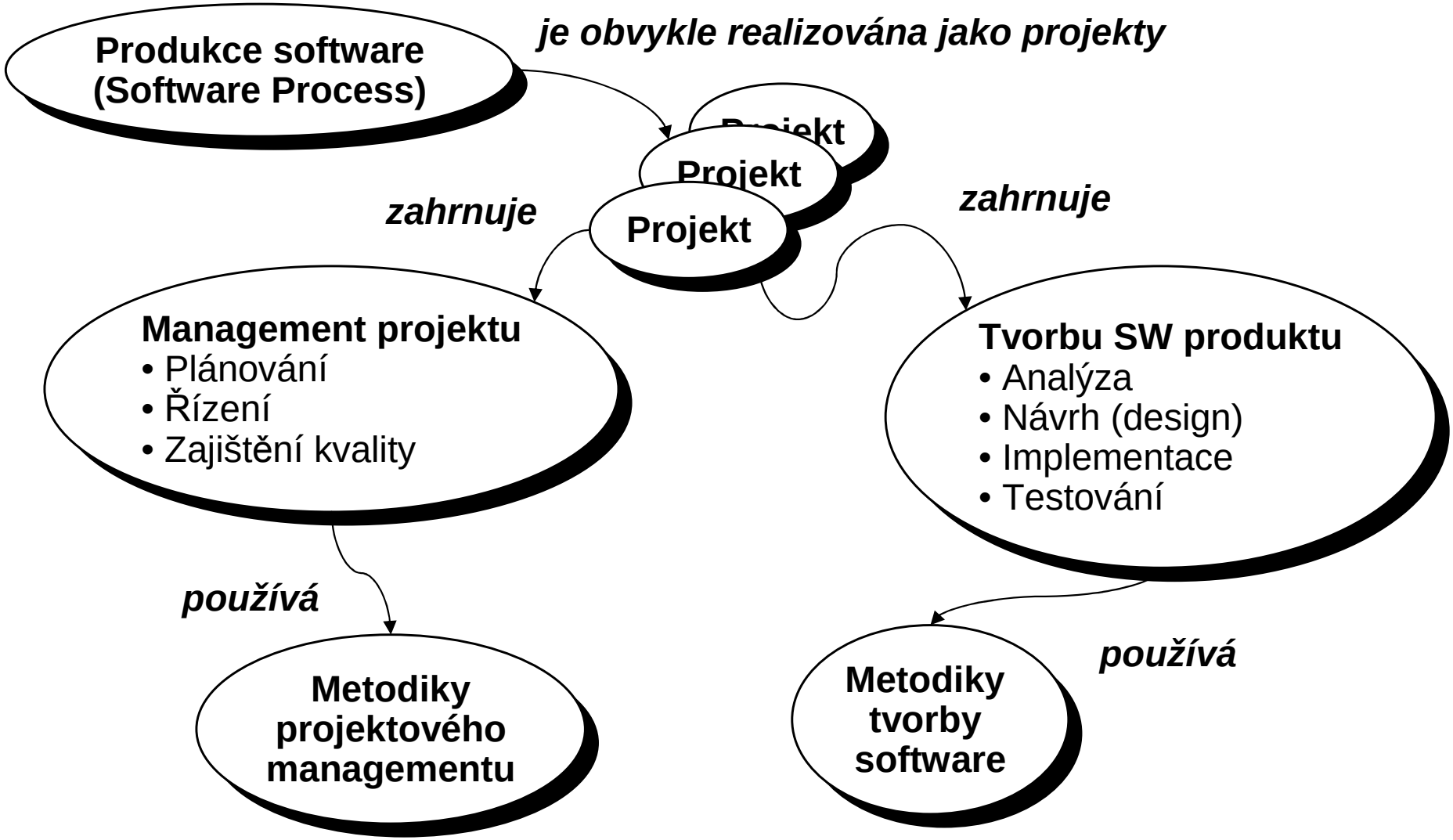
  UCP = (UAW+UUCP)*TCF*EF
- **Vynásobte UCP předpokládanou pracností jednoho případu užití (cca 15 – 30 hod, Karner doporučuje 20 hod). Získáte pracnost v člověko-hodinách.**

Příklad (Zdroj: www.komix.cz)

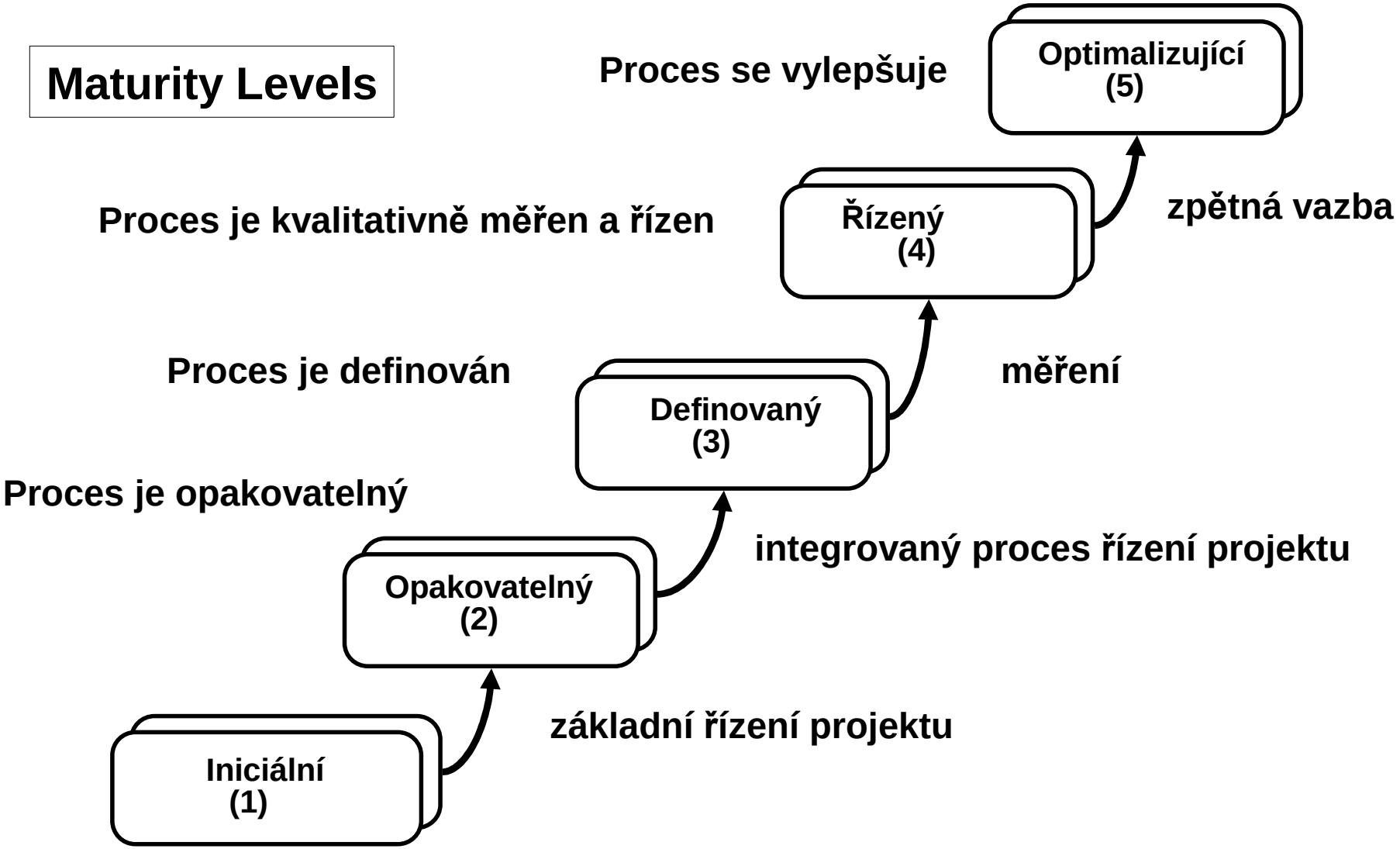
Co od Vás budeme chtít:

- Naučit se číst a vytvářet plány.
- Prostudovat si a upravit plán SINPlan2007.mpp (to je plán Vaší práce).
- Vytvořit plán práce pro Vaše následníky (pro ty, kteří budou projekt implementovat) a odhadnout z něj cenu.
- Pro ověření odhadnout cenu ještě jiným způsobem – např. pomocí COCOMO, nebo Karnerovou metodou.

Produkce software



Úroveň procesu tvorby software



The End

SOFTWAROVÉ INŽENÝRSTVÍ

ÚVODNÍ FÁZE VÝVOJE

Martin Komárek

MDD – Model Driven Development

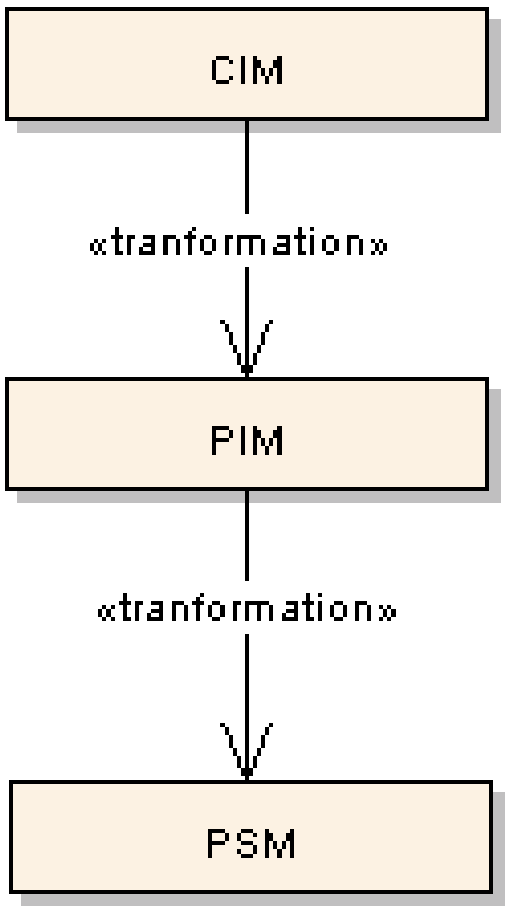
- Model vzniká vždy. Někdy bohužel jen v hlavách vývojářů.
- MDA Guide (www.omg.org)
- Model je kombinací textu a diagramů.
- Model by měl být „čitelný“ i bez diagramů.
- Diagramy „pouze“ usnadňují pochopení základní myšlenky popisovaného
- Každý digram by měl obsahovat pouze jednu myšlenku
- vytváření modelu je tedy převážně mentální úsilí vyúsťující v psaní textu

Proč modely vytvářet?

„Zaměstnanci přicházejí a odcházejí, modely zůstávají.“

- Znovupoužitelnost (firemní know-how)
- Sledovatelnost realizace požadavků
- Zvýšení udržovatelnosti produktu

MDD



- Computation Independent Model
- Platform Independent Model
- Platform Specific Model

CIM

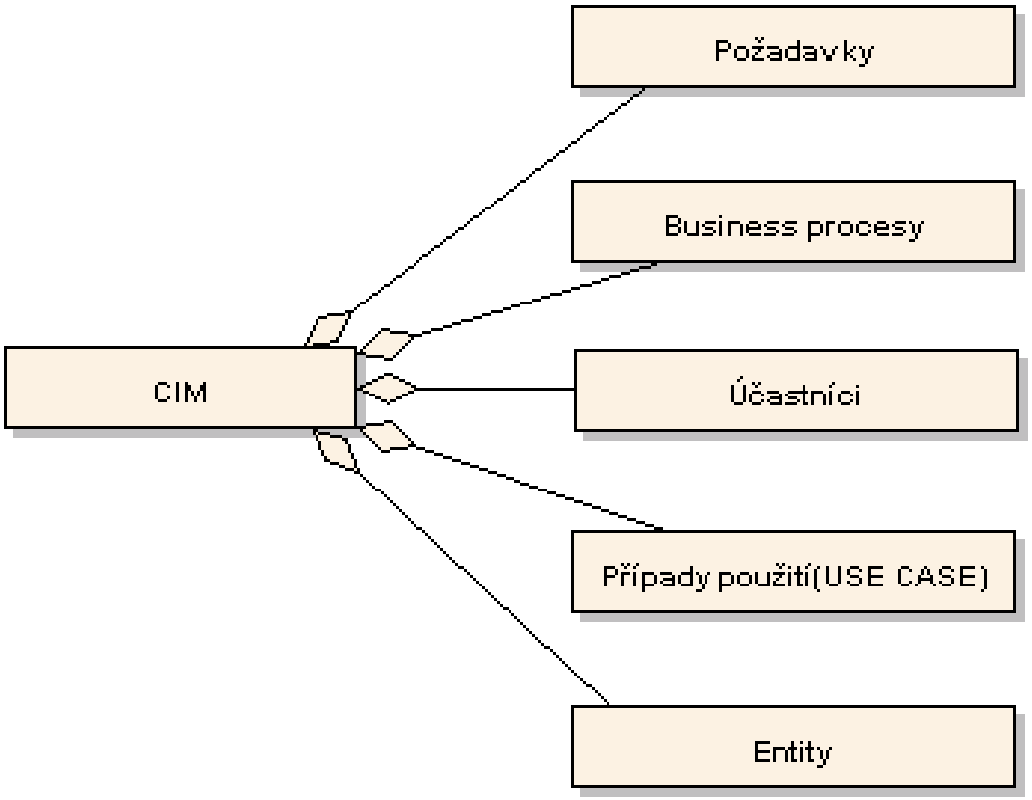
- Computation Independent Model ~ Domain Model ~ Conceptual Model
- strategický materiál pro odhad ceny
- strategický materiál pro vypracování nabídky
- podklad pro zavření smlouvy
- zákazníkovi srozumitelný a neobsahuje žádné implementační detaily

CIM

Podkladové materiály pro tvorbu jsou rozmanitého druhu:

- záznamy z jednání,
- normy, směrnice, zákony,
- stávající systémy,
- deklarace záměru,
- rešerše,
- .....

CIM


Požadavky
Business procesy
CIM
Účastníci
Případy použití(USE CASE)
Entity

UML – Unified Modeling Language

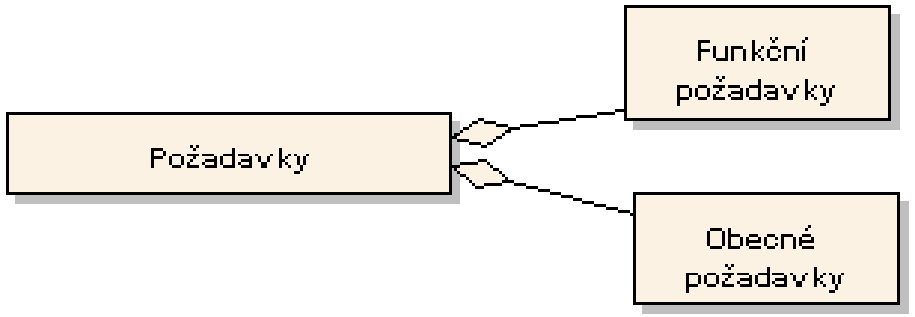
- Standard (www.omg.org)
- Syntaxe
- Sémantika

OOA/D - Objektově orientovaná analýza a návrh

- Co to je OO(A/D) ?
- Rozdíl mezi analýzou a návrhem
- Entity
- Business třídy
- Návrhové třídy

Požadavky

- Zachycují očekávání zákazníka
- mají jasné identifikátory (číslují se)
- hierarchicky děleny (zde na "funkční" a "obecné", ale možno i jinak)



- jeden požadavek=pouze jedno měřitelné "očekávání„
- obsahují (pokud možno) minimální množství informací o implementaci

Funkční požadavky

- definují co bude systém umožňovat
- měly by mít stanoveny priority

Př.:

1.1 Systém bude evidovat(minimálně po dobu pěti let) kdo a kdy měnil emailovou adresu na kterou je uživatelům zasíláno heslo.

1.2. Systém bude vždy při změně emailové adresy, na kterou je uživatelům zasíláno heslo, o této změně uživatele informovat a to tak, že mu na původní i novou emailovou adresu pošle zprávu o tom, kdo a kdy změnil jeho adresu na zasílání hesla.

Obecné požadavky

- vztahují se k celému systému
- spíše omezují způsob, jak bude systém navržen
- většinou se realizují vhodnou volbou jádra systému

Typové příklady obecných požadavků

- požadavky na parametry RAMS (relaibility, availability, maintainability, safety/security)
- požadavky na výkon
- požadavky na použitou platformu (operační systém, hardware, ...)
- požadavky na rozhraní s uživately i jinými systémy (vícejazyčnost, WEB/WAP/SMS brány/... , ...)
- požadavky na dokumentaci (analytická / návrhová / programátorská / uživatelská)
- požadavky související s právními aspekty (výhradní / nevýhradní / otevřená licence, otevřený kód, standardy atd.)

Příklad obecného požadavku

6.5. Systém bude spolehlivý.

Takovýto požadavek může být v závěru projektu oběma stranami interpretován velmi rozdílně. Proto je nutné požadavek upřesnit třeba takto:

6.5. Systém bude spolehlivý.

6.5.1. Střední doba do výpadku systému bude maximálně 20 dní.

6.5.2. Střední doba do opravy systému bude maximálně 12 hodin.

Business procesy - Co to je ?

- Zachycují i "akce", které se souvisejí s nasazením systému jen nepřímo

Business procesy – proč popisovat?

- Pro potřeby tvorby nabídky

  Model (business) procesů slouží jako podklad pro odsouhlasení rozsahu aplikace klientem.

- Pro potřeby nasazení systému

  a) pro testery - tester může testovat i business logiku

  b) pro tutory - školitelé musí vědět nejen co dělá které menu, ale i k čemu systém slouží

Business procesy – příklad

Př. "Přichází zákazník k přepážce banky" -> "Přijetí žádosti o hypotéku" -> atd... .

I bez znalosti bankovní problematiky dovodíme, že žádost bude pravděpodobně zadána do systému bankovní úřednicí.

Pokud bychom ale neměli v modelu zachyceno, že zákazník "Přichází k přepážce" mohli bychom si myslet, že zákazník může podat žádost o hypotéku i přes webové rozhraní nebo po telefonu.

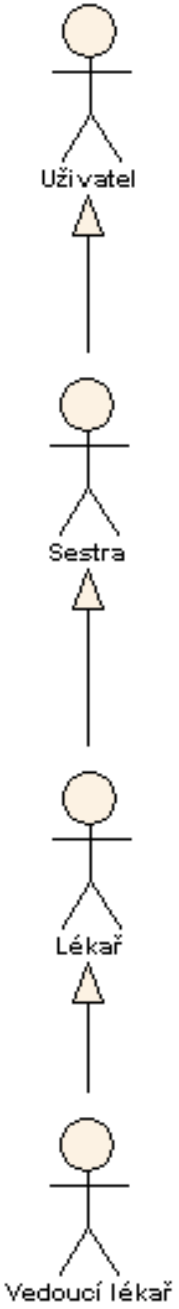
I z tohoto triviálního příkladu je zřejmá důležitost zachycení business procesů v CIM modelu.

Účastníci (=aktéři)

Jednotlivý účastník(actor)

- jsou vůči systému externími entitou, která systém využívá nebo ho ovlivňuje.
- většinou účastníkem reálná osoba(uživatel), ale může jím být například i "čas" (spouštění záloh atd..) nebo dokonce "blesk(=elektromagnetické rušení)"

Generalizace účastníků

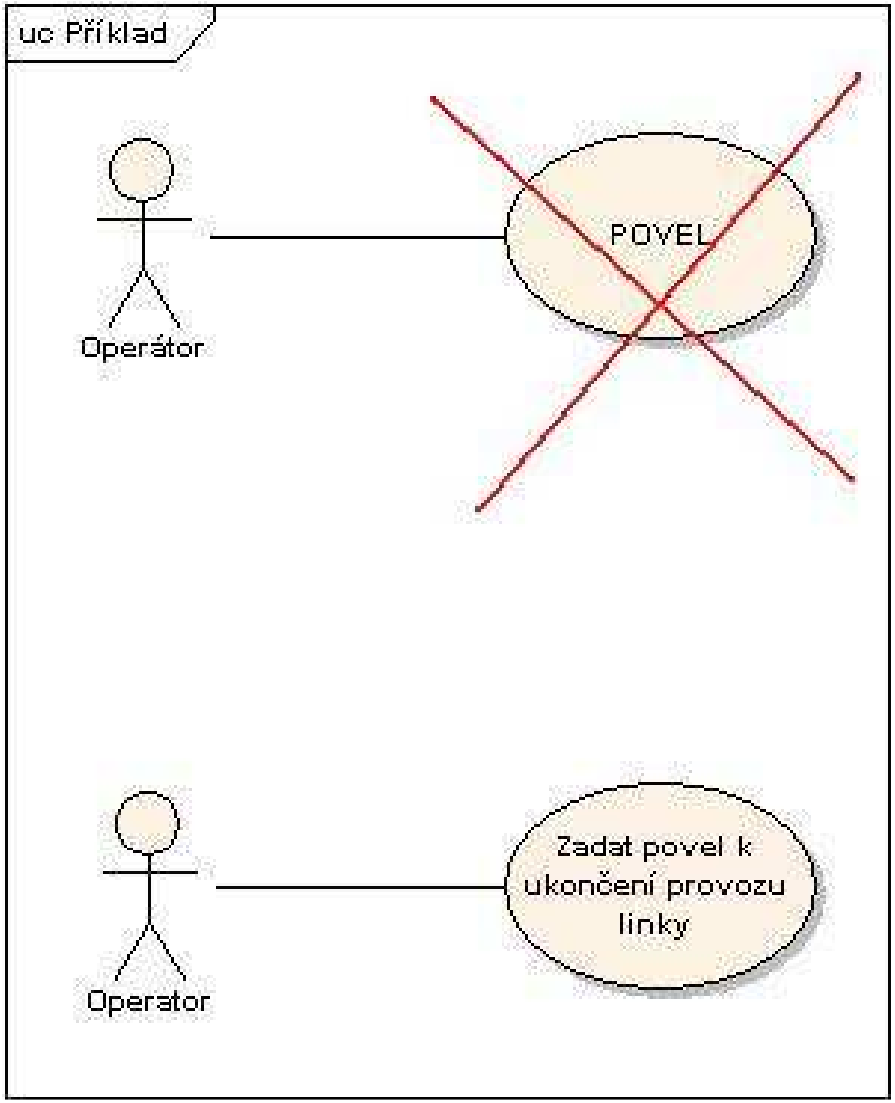


Případy použití systému (USE CASE)

“případ použití"~"případ užití"~"užitná činnost“~”USE CASE”

- Měl by popisovat jednu rutinní akci jednoho účastníka v jednu chvíli
- Je vždy iniciován účastníkem
- USE CASE diagram znázorňuje funkce systému z pohledu účastníků
- Název USE CASE má vždy *slovesnou vazbu*!!!

Název USE CASE !!! Slovesná vazba !!!



Vztahy mezi případy použití

- <<extend>>

  - pokud nějaký případ rozšiřuje chování (je zde možnost volby)

- <<include>>

  - pokud jeden případ zahrnuje případ jiný

  (např. přepočet ceny prodávaného zboží po změně kurzu EURO)

Entity systému

- většinou reálně existující „věci“, se kterými systém bude manipulovat
- seznam entit a jejich popis je „slovníčkem pojmů“
- ze seznamu "entit" se vychází při vytváření "analytických tříd" v PIM

Package (=balíček)

- Umožňují strukturovat modely
- Obdoba adresářů

Dotazy ?

X36SIN: Softwarové inženýrství

Modelování požadavků

Kontext

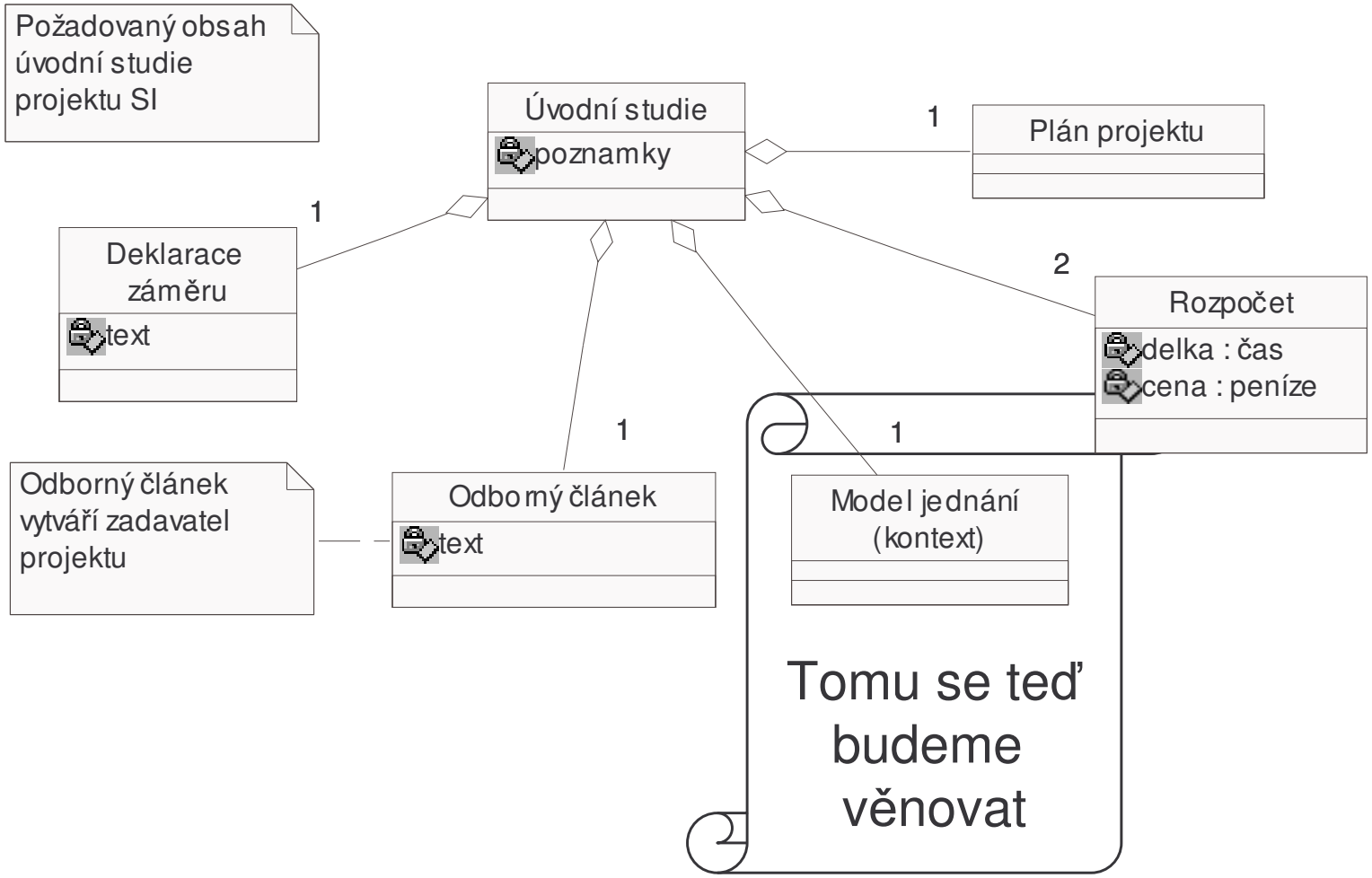
Dosud jsme probrali:

- Modely životního cyklu
- Co to je deklarace záměru, odborný článek, katalog požadavků
- Jak se plánuje a odhadují náklady

Dnes se budeme zabývat:

- Sběrem a modelováním požadavků
- Seznam aktérů a seznam událostí vyjadřujeme pomocí modelu jednání (use case model)
- Slovní popis případů užití a grafické vyjádření

Obsah úvodní studie

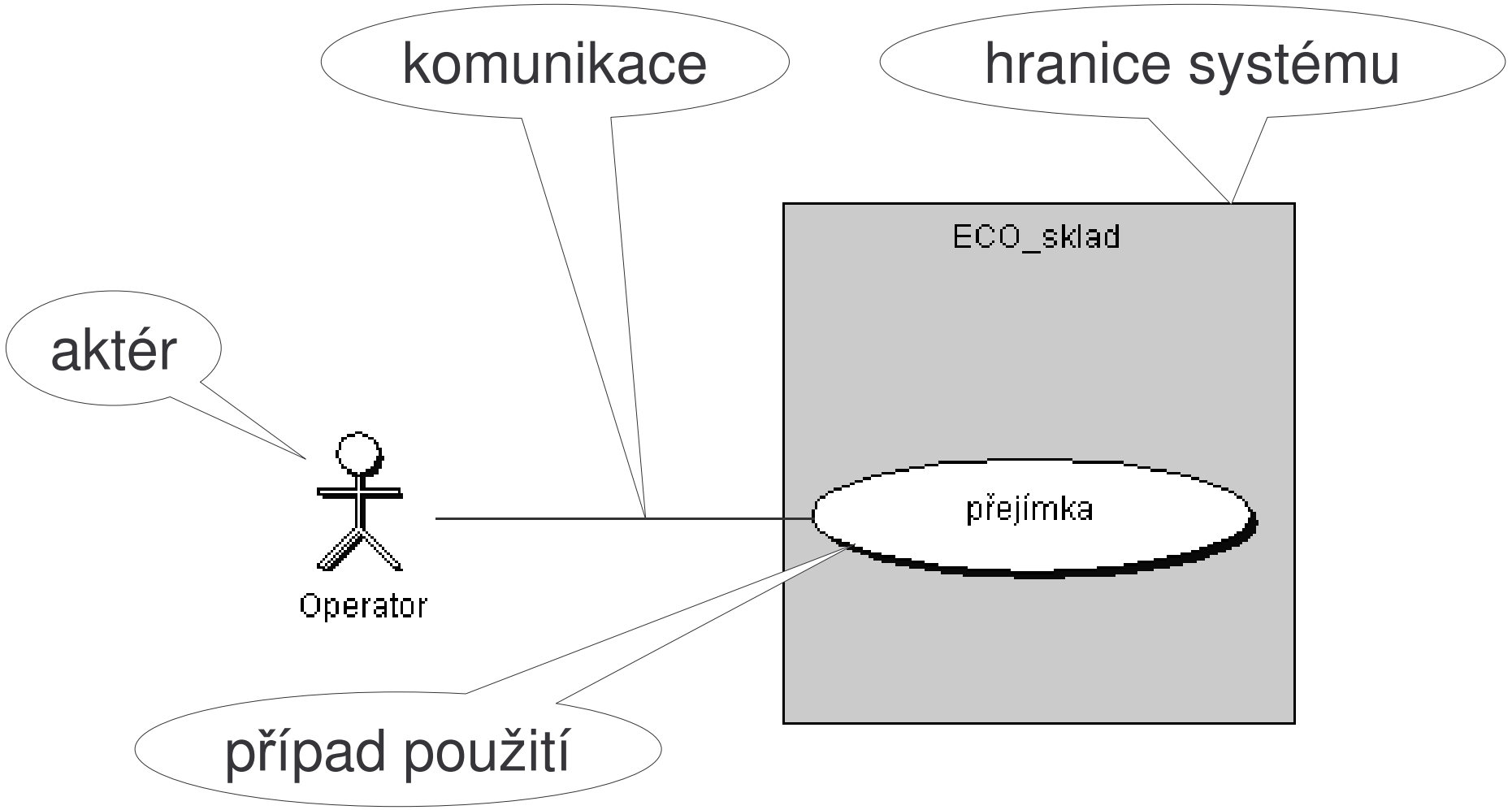


Model jednání (Use Case Model)

Prvky:

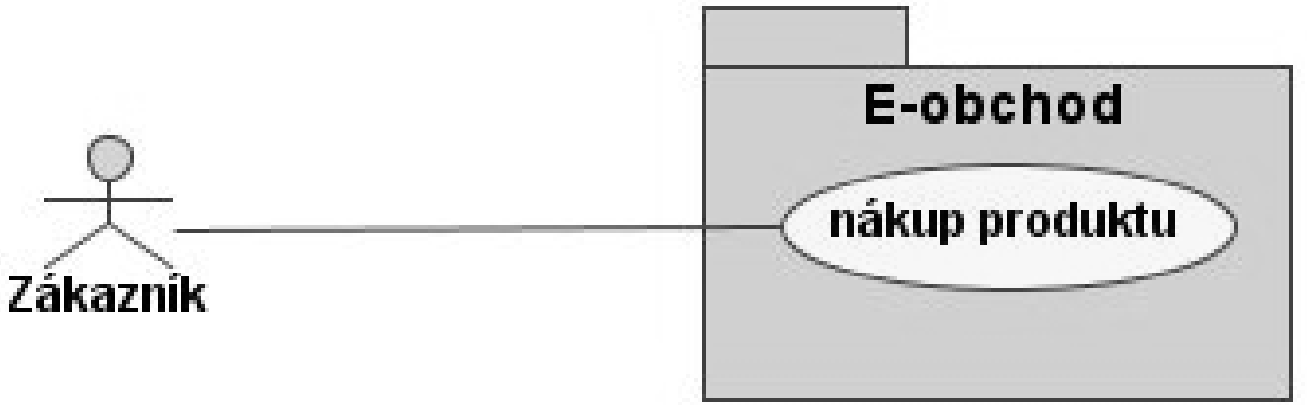
- **aktér** (actor) - uživatelská role nebo spolupracující systém
- **hranice systému** (systém boundary) - vymezení hranice systému
- **případ použití** (use case) - dokumentace události, na kterou musí systém reagovat
- **komunikace** - vazba mezi aktérem a případem použití (aktér komunikuje se systémem na daném případu)

Notace modelu jednání (UML)

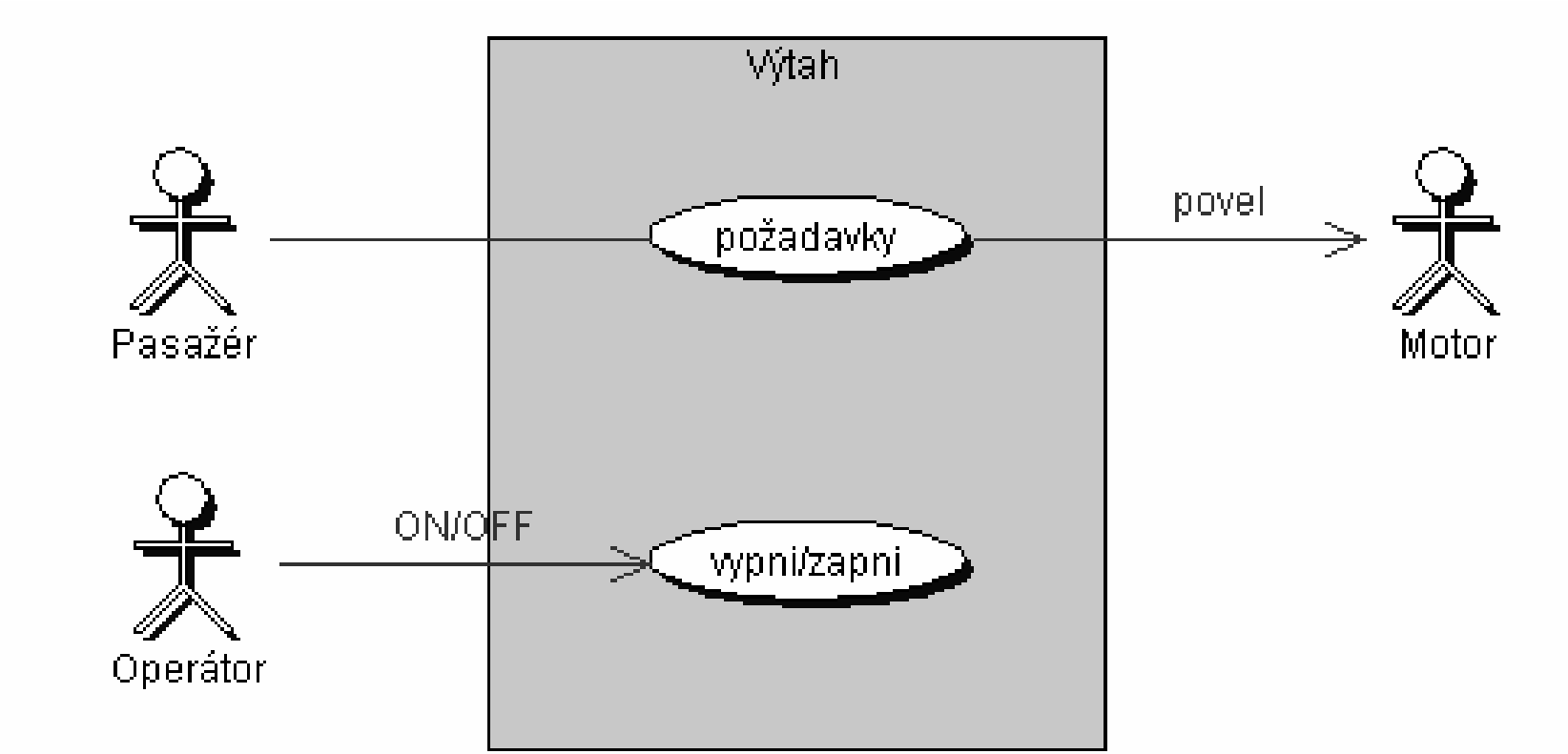


Příklad: „e-obchod“

- E-obchod poskytuje zákazníkům možnost nákupu produktů.



Příklad modelu jednání



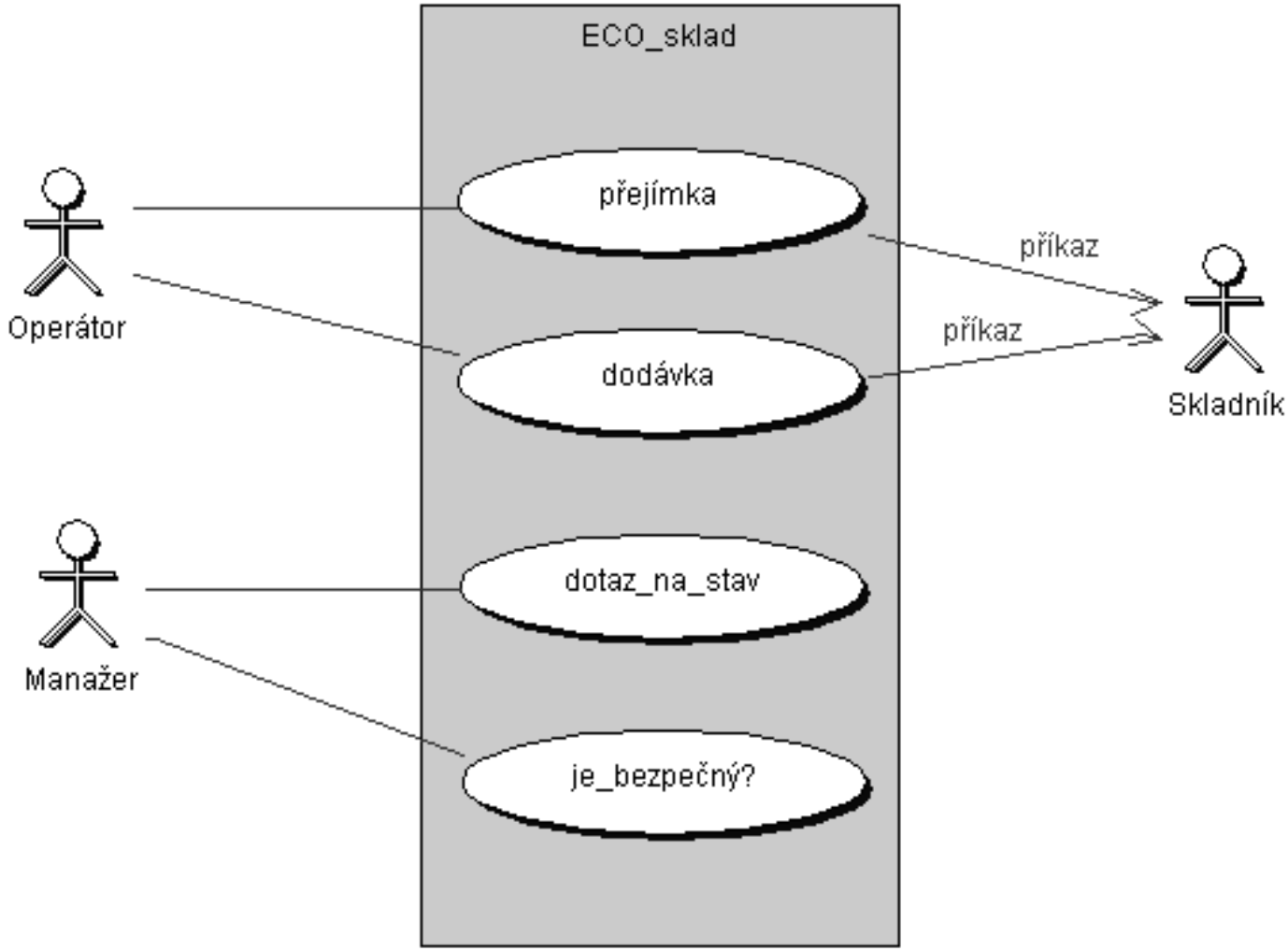
Chyby v modelu jednání

- Aktéři spolu komunikují mimo systém
- Není zdůrazněn dvojí výskyt aktéra
- Chybí případ použití (služba) pro některou událost
- Chybí některá reakce systému
- Případ použití není popsán v datovém slovníku
- Případ použití je popsán nevhodně (příliš obecně)
- Dva různí aktéři mají stejnou sadu událostí (pak to zřejmě nejsou různí aktéři)
- Za událost se považuje přihlášení do systému (zařazení do role jde mimo kontext)

Příklad: ECO-sklad

ECO sklad je zařízení pro ekologické ukládání barelů s chemikáliemi klasifikované jako typ 1, 2 a 3 (dle EPA - Environmental Protection Agency). Barely se ukládají do skladových budov se stanovenou kapacitou (ve skladu ale existují i jiné budovy). Chemikálie typu 1 a 2 nesmí být uloženy do stejné budovy, chemikálie typu 3 mohou být uloženy libovolně. Do skladu jsou přejímány barely přes nakládací plošinu, odtud se též odvážejí při vyskladnění. Přejímka i dodávka je vybavena dodacím listem. Při přejímce operátor převezme dodací list, vyložené barely označí jednoznačným identifikátorem a po vyložení všech barelů zkontroluje skutečný stav. Barely rozváží z plošiny skladník na základě vystaveného příkazu. Při dodávce operátor převezme požadovaný dodací list, vystaví skutečnou dodávku a předá skladníkovi příkaz k vyskladnění.

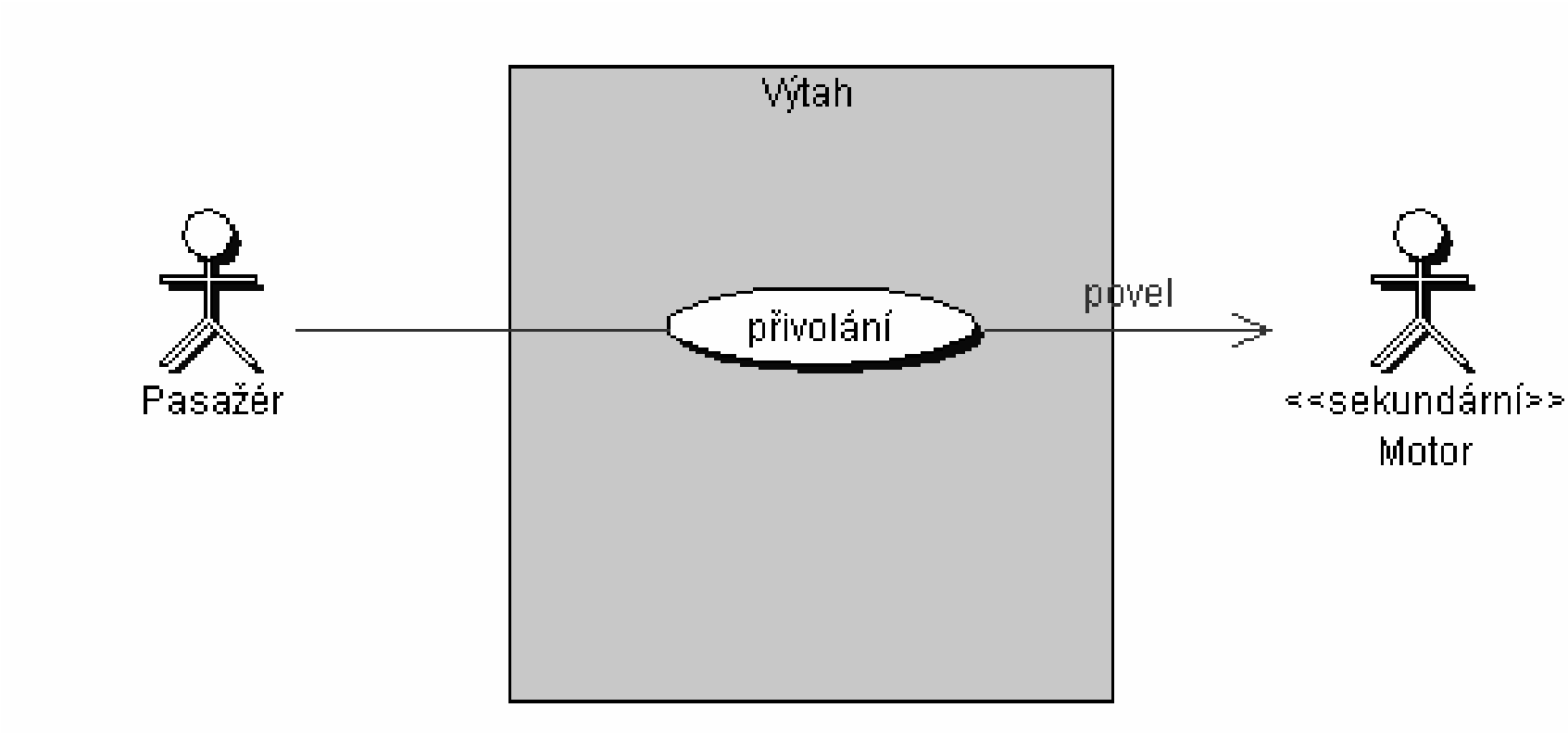
Model jednání pro ECO-sklad



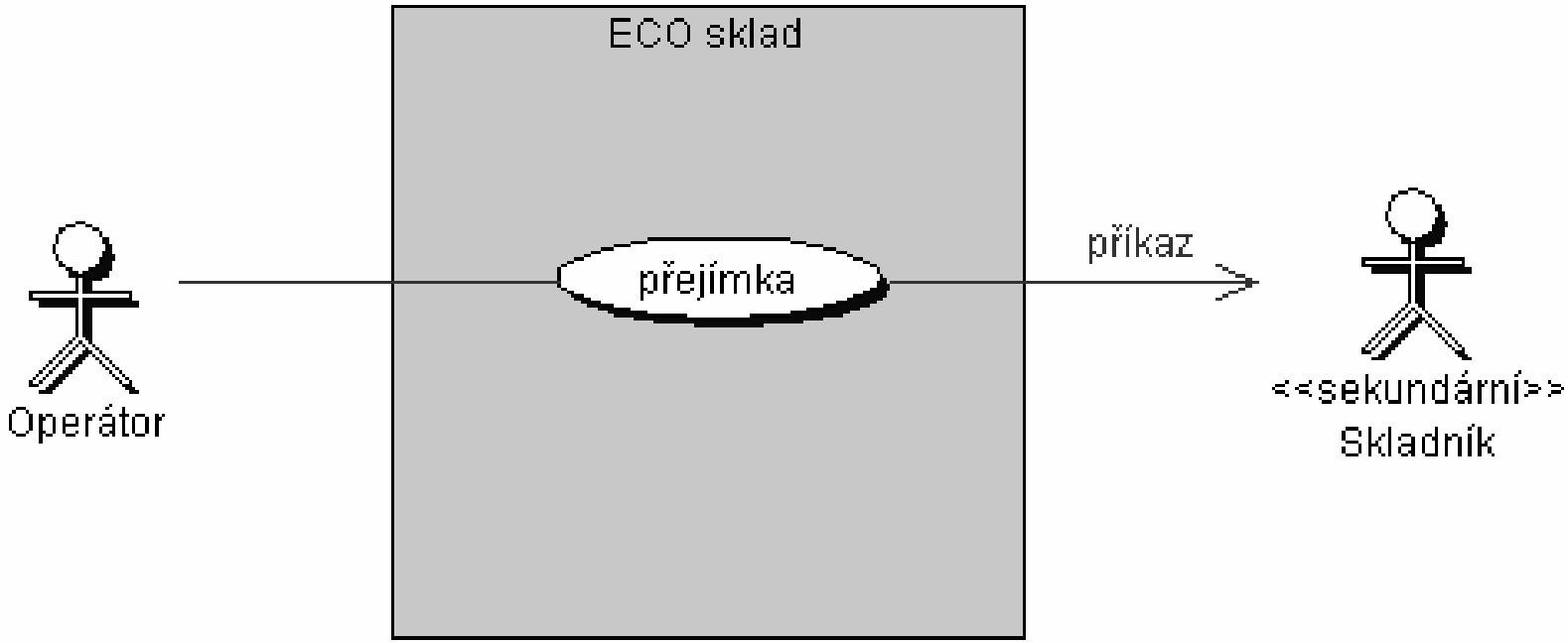
Doplňky k modelu jednání

- **sekundární aktér** - uživatelská role nebo spolupracující systém nutná pro činnost systému

Sekundární aktéři



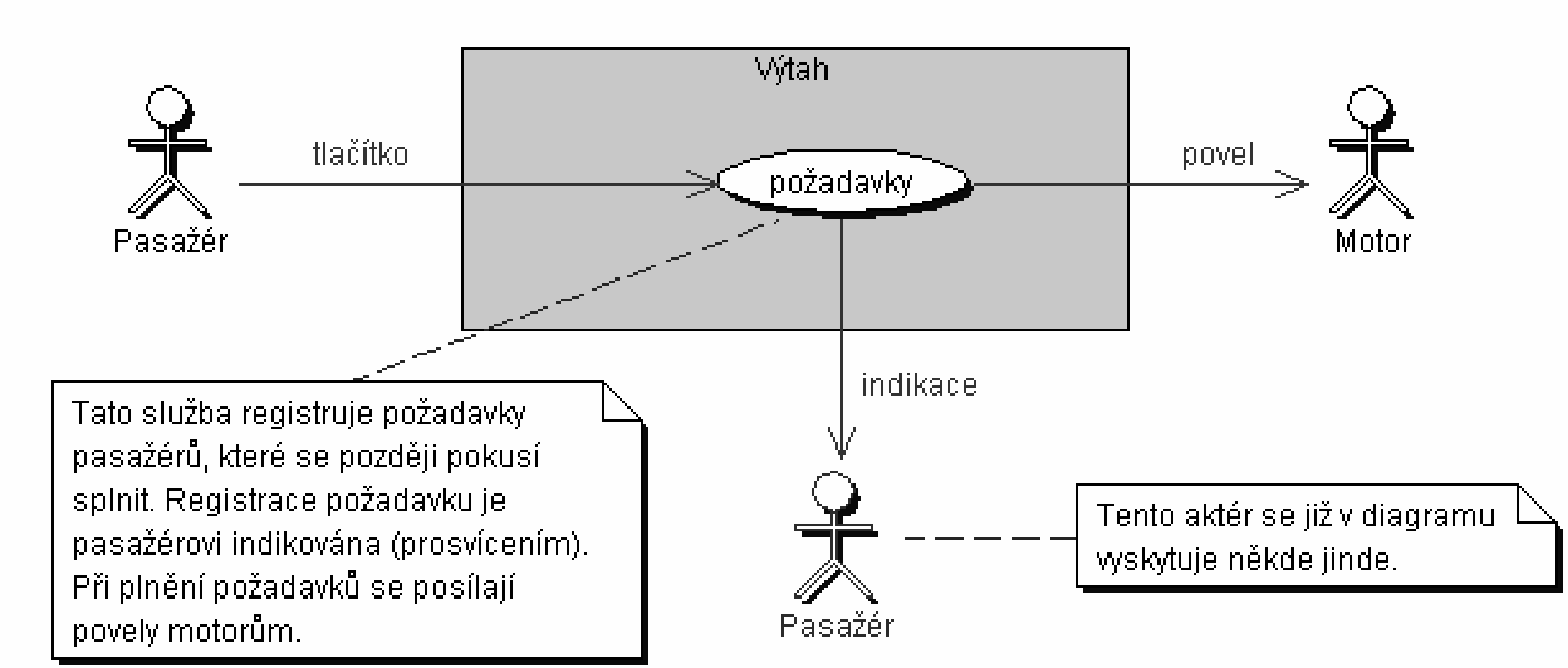
Sekundární aktéři



Doplňky k modelu jednání

- **orientovaná komunikace** - případ, kdy chceme vyznačit směr komunikace

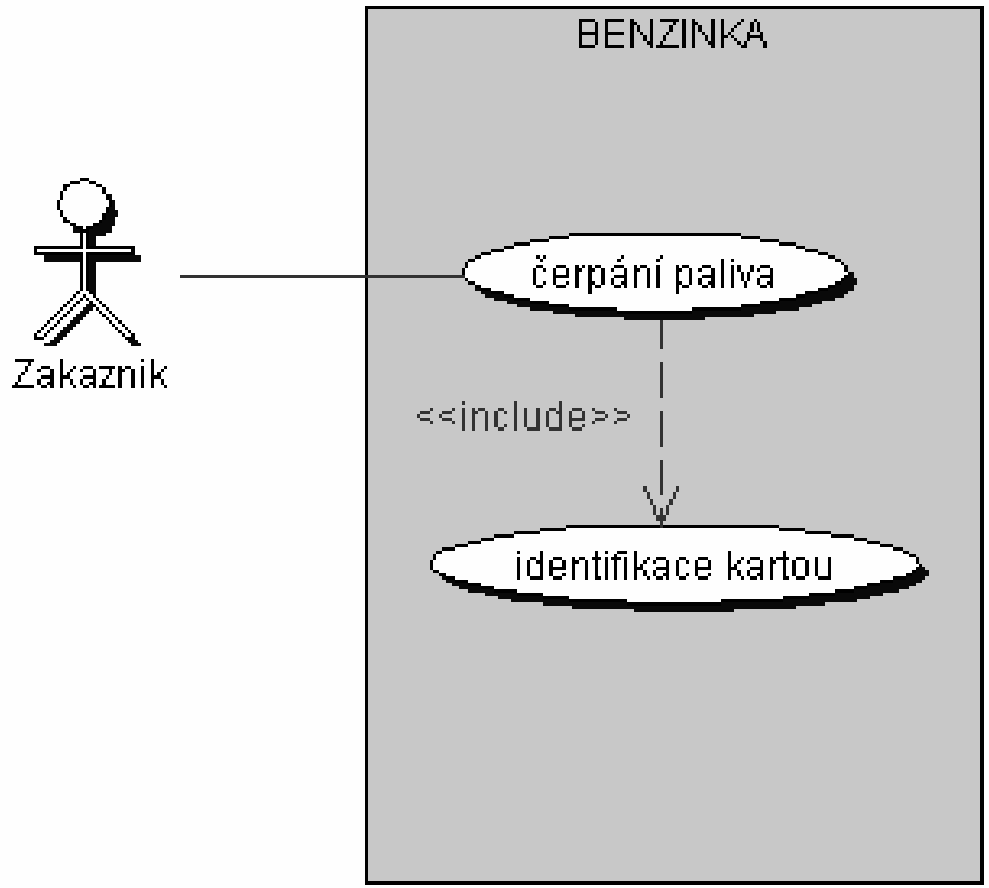
Orientovaná komunikace



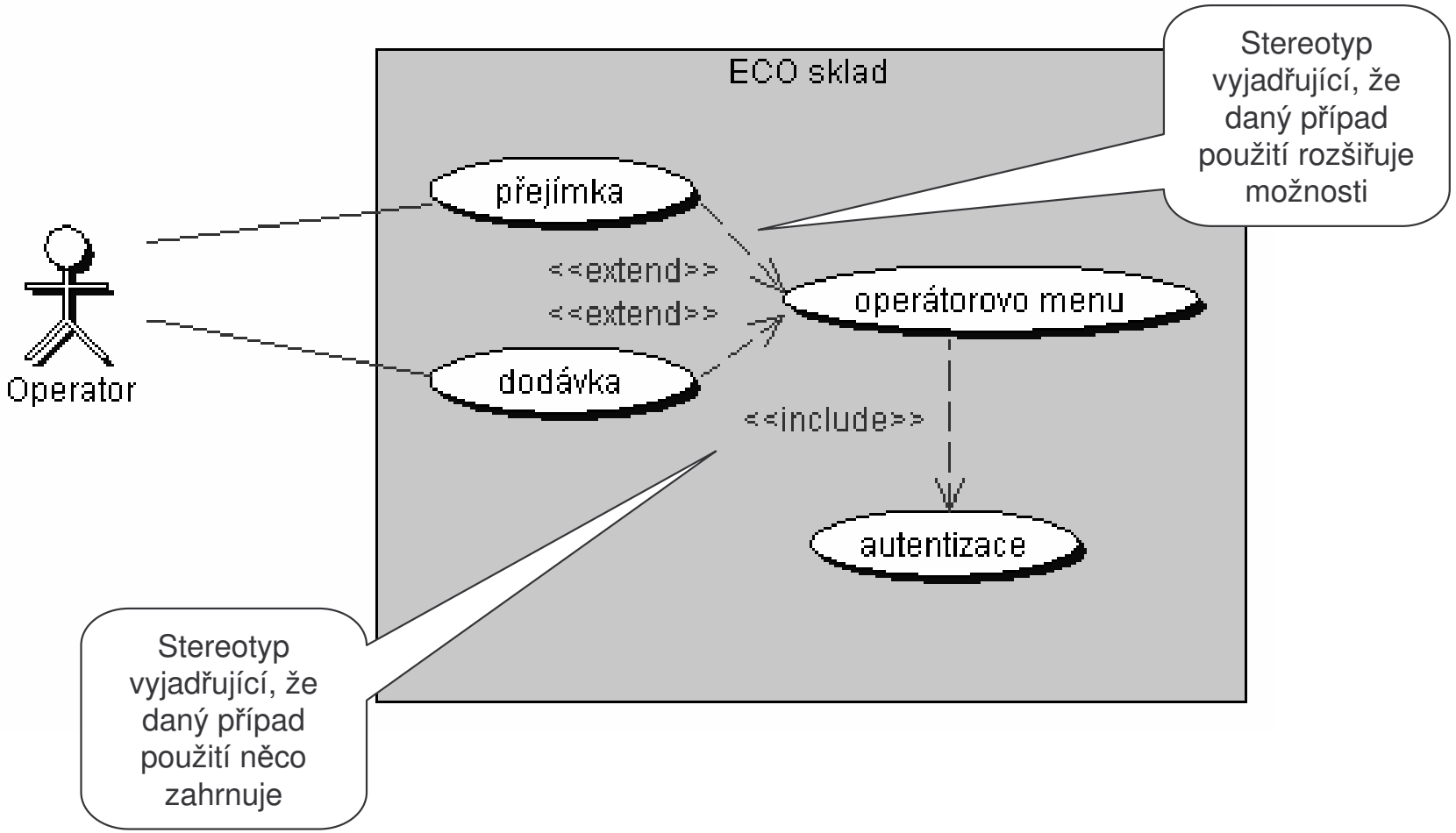
Doplňky k modelu jednání

- **vztahy mezi případy použití** - pokud chceme explicitně vyjádřit fakt, že takový vztah existuje
  - **<<include>>** - pokud jeden případ zahrnuje případ jiný (např. autentizace)
  - **<<extend>>** - pokud nějaký případ rozšiřuje chování (je zde možnost volby)
  - **generalizace/specializace**

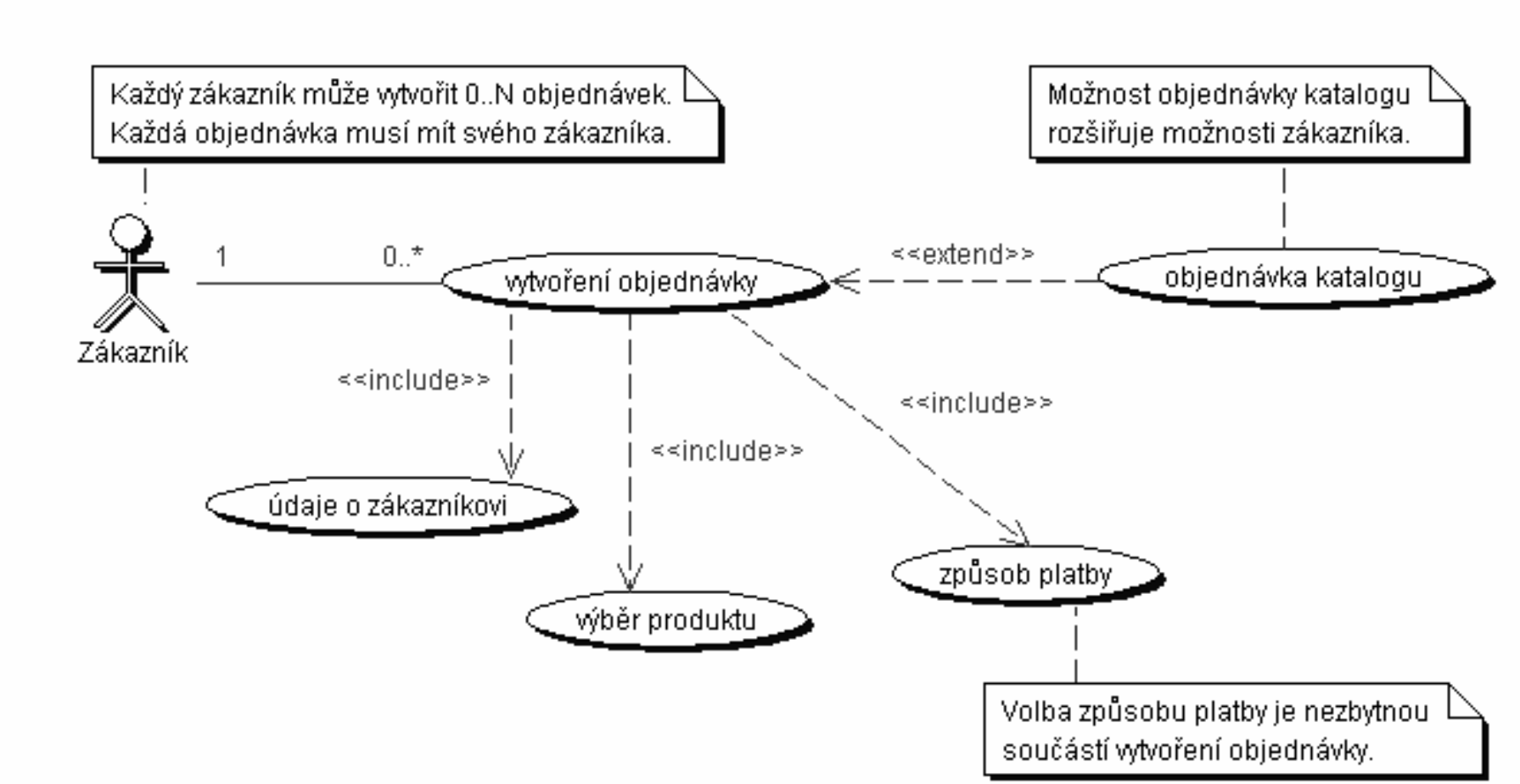
Vztahy mezi službami



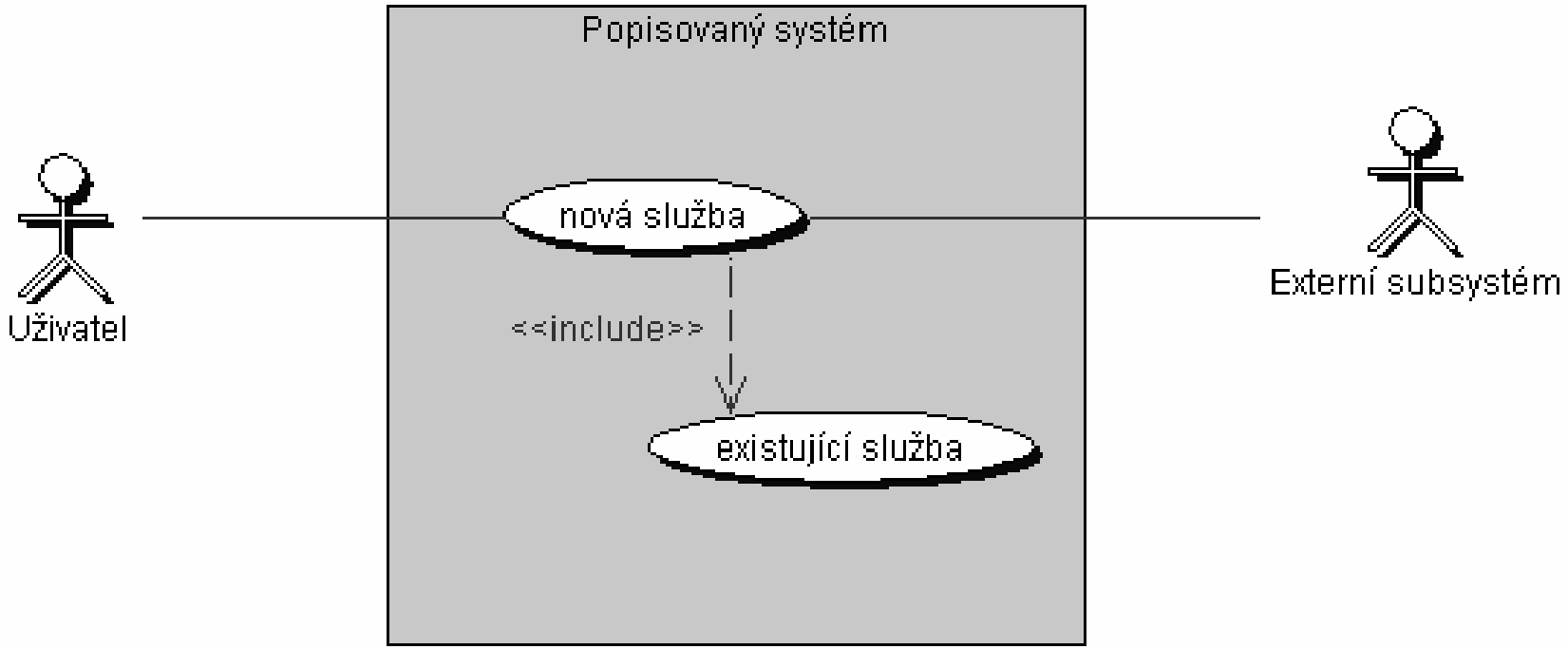
Vztahy mezi službami



Vztahy mezi službami

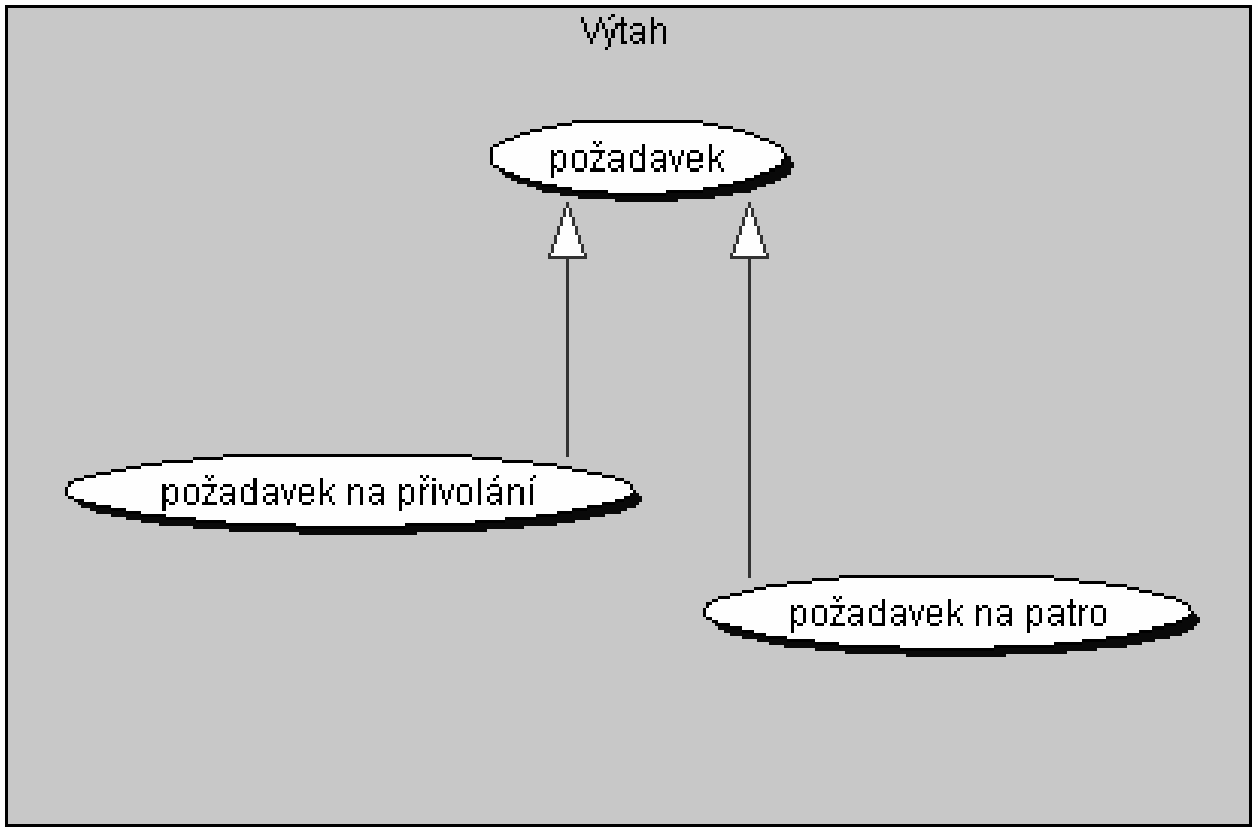


Kombinace různých prvků



Kombinace různých prvků v modelu jednání:
- nově vyvíjená část (nová služba)
- použité existující části (existující služba)
- část od jiného dodavatele (externí subsystém)

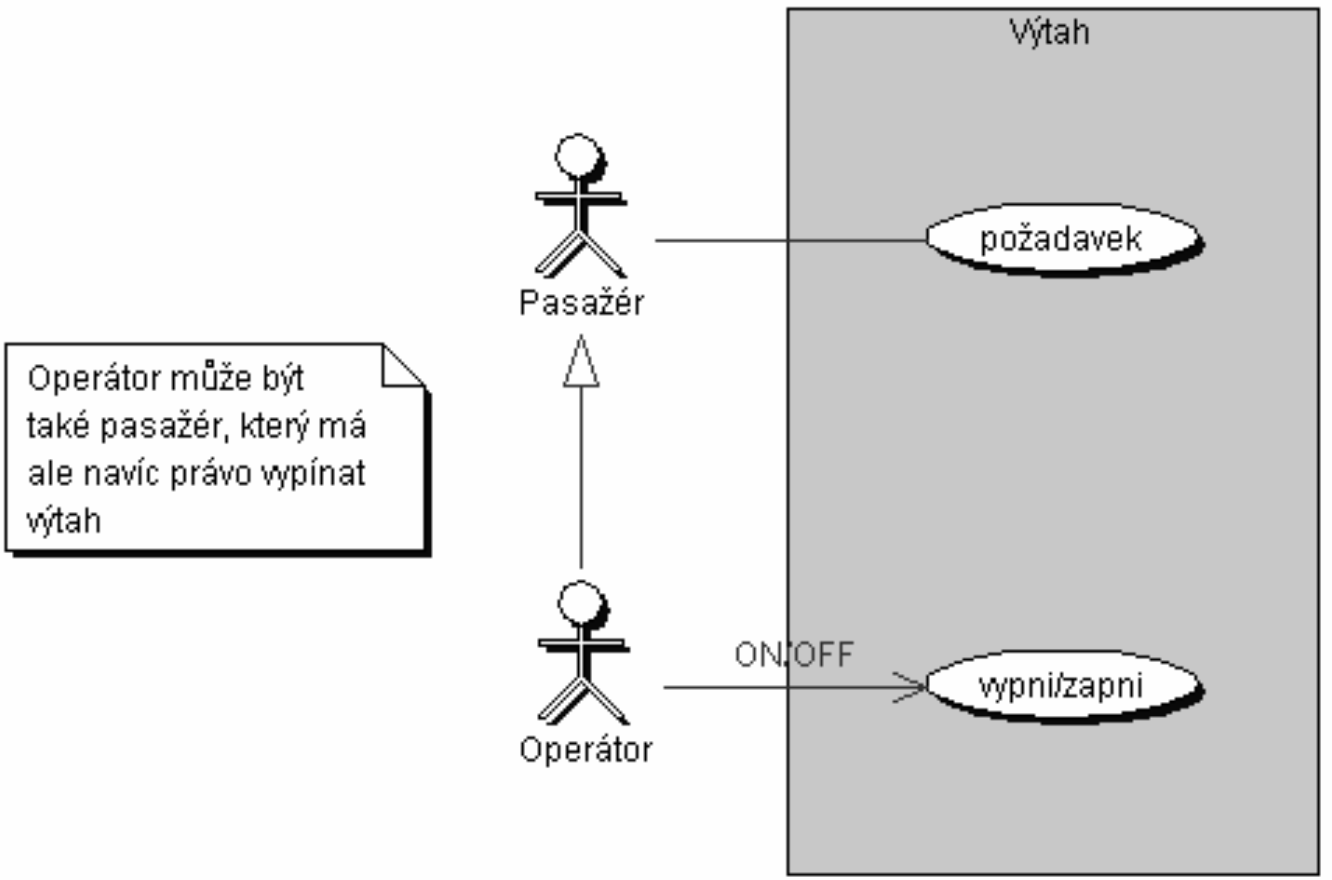
Generalizace služeb



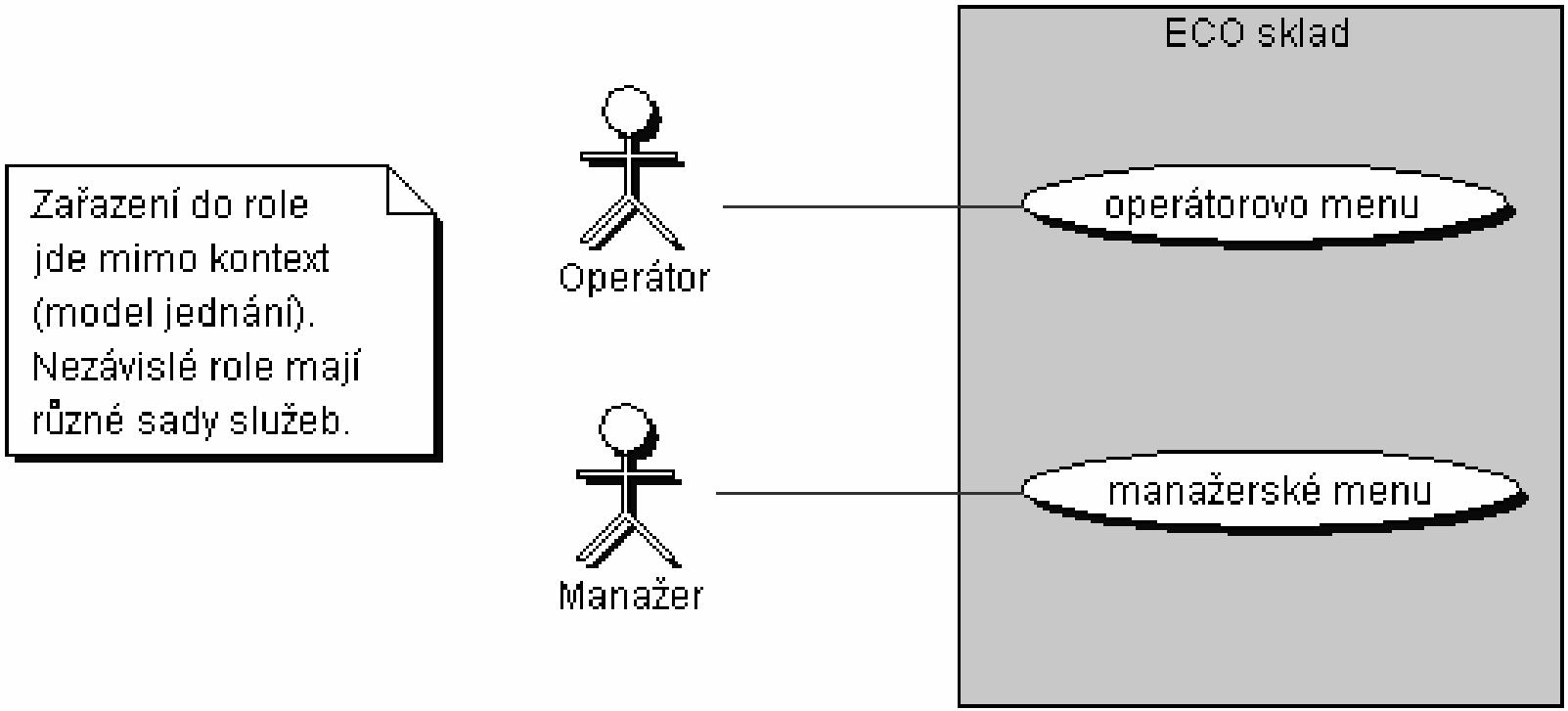
Doplňky k modelu jednání

- **vztahy mezi aktéry** - pokud chceme explicitně vyjádřit fakt, že takový vztah existuje
  - generalizace/specializace

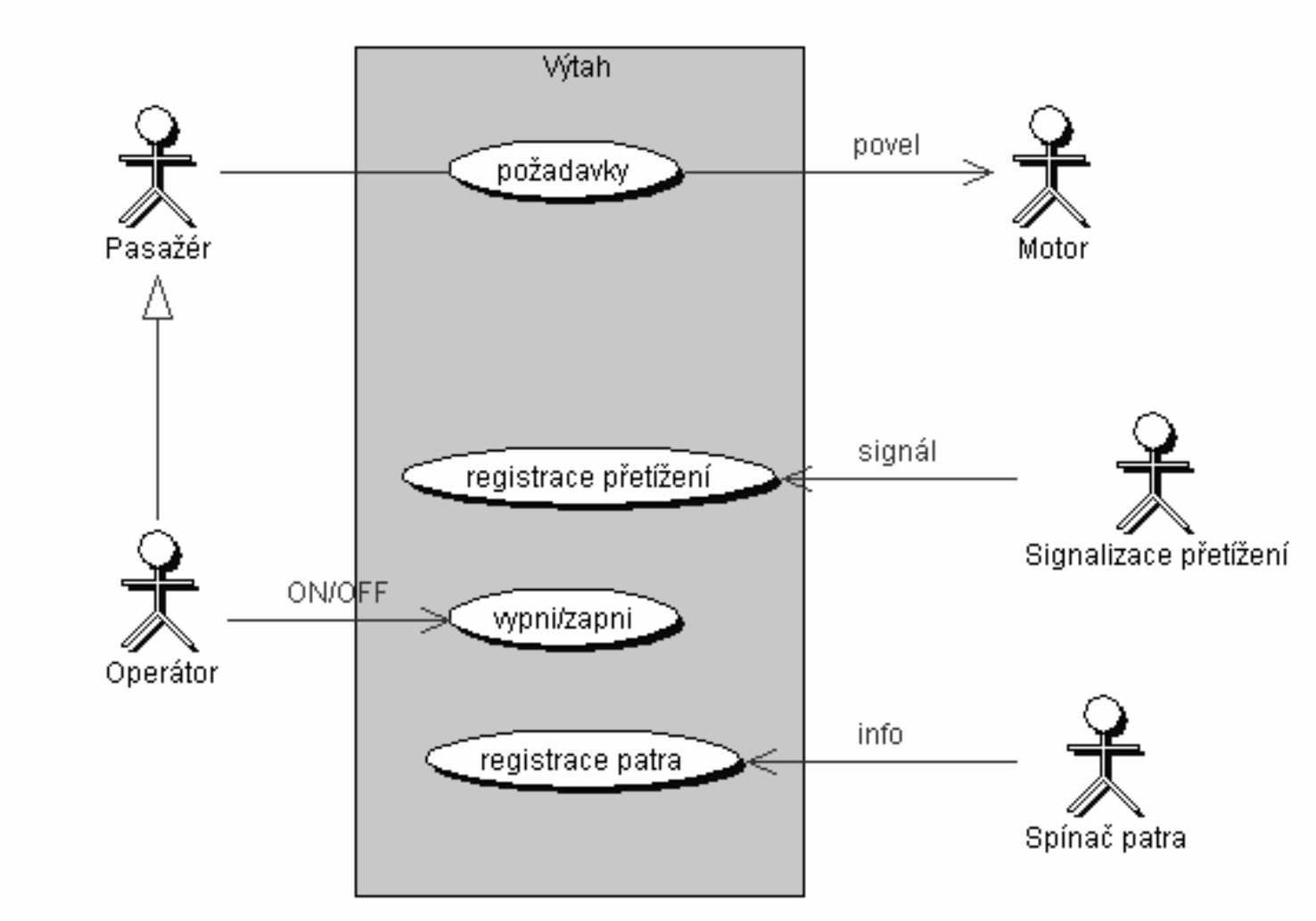
Generalizace aktérů



Autentizace do role

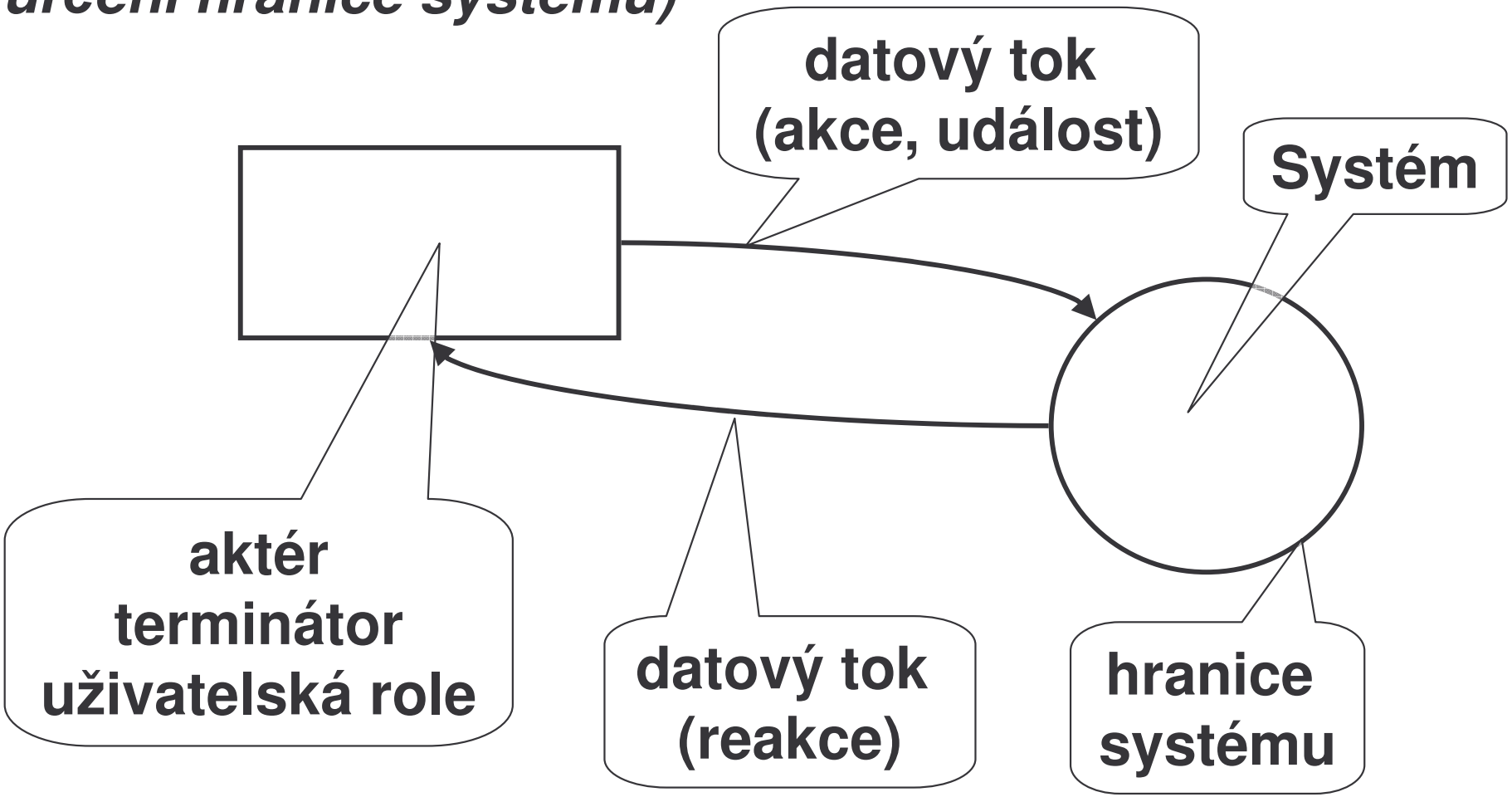


Model jednání pro Výtah



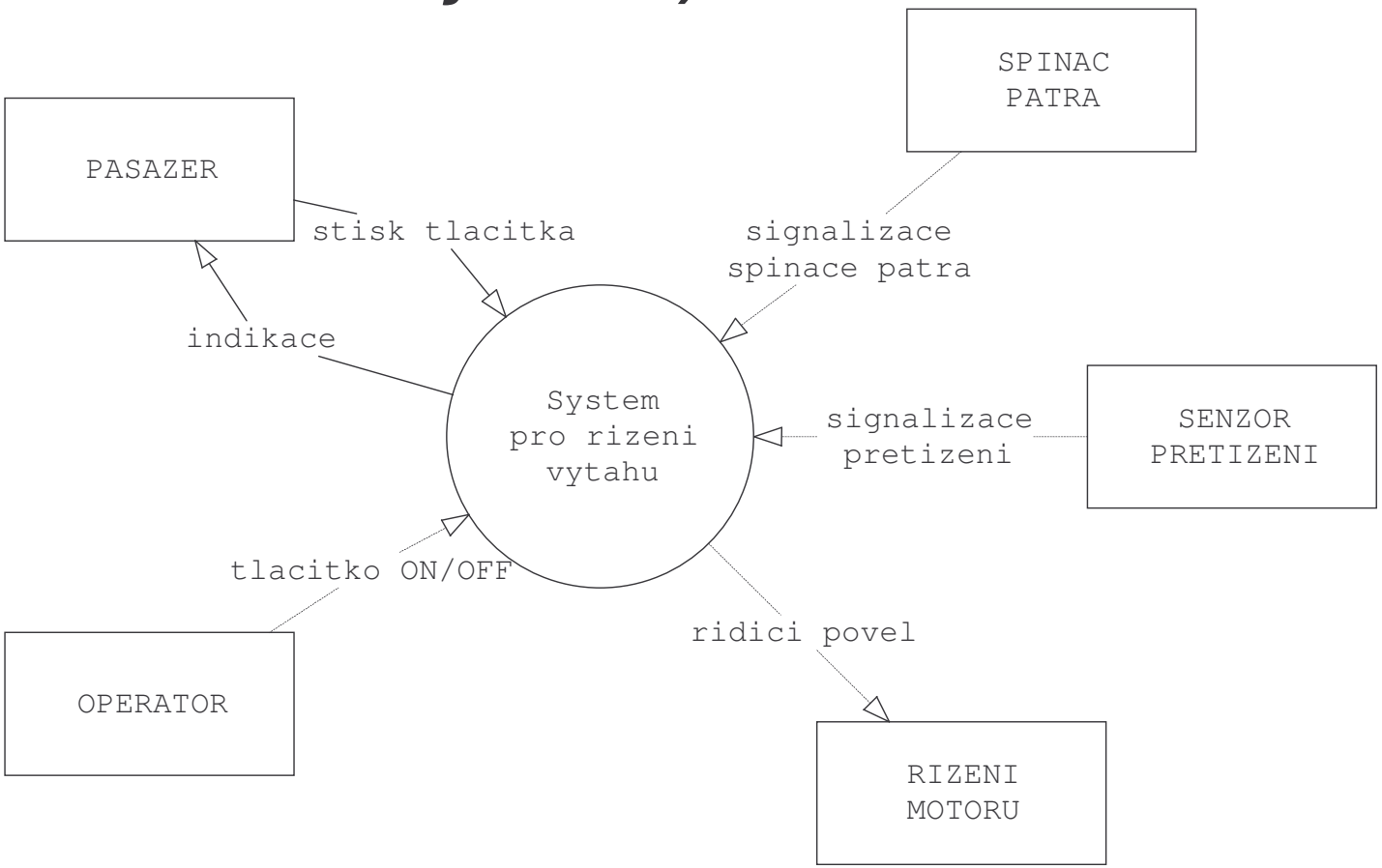
Kontextový diagram

(určení hranice systému)



Kontextový diagram pro “Výtah”

(určení hranice systému)



Chyby v definici kontextu

- Aktéři spolu komunikují mimo systém
- Není zdůrazněn dvojí výskyt aktéra
- Chybí datový tok pro některou událost
- Chybí datový tok pro některou reakci systému
- Datový tok není popsán v datovém slovníku
- Datový tok je popsán nevhodně (příliš obecně)
- Dva různí aktéři mají stejnou sadu událostí (pak to zřejmě nejsou různí aktéři)
- Za událost se považuje přihlášení do systému (zařazení do role jde mimo kontext)

Analýza

CO má systém umět

Návaznosti

- Dosud:
  - Úvodní studie, modelování požadavků
- Dnes:
  - Analýza
- Příště:
  - Architektura, modelem řízený vývoj

Analýza

Měla by odpovědět na otázku ***CO***?

- Musí proto definovat ***konceptuální model*** řešeného systému (***PIM – Platform Independent Model***)
- Musí definovat představu, s jakými ***daty*** bude systém pracovat, jaké ***služby*** bude systém poskytovat a jak se bude chování systému měnit - jaká bude ***dynamika*** systému
- Musí stanovit podmínky, za jakých je analytická dokumentace ***akceptovatelná***

Analytický (konceptuální, PIM) model

- (konceptuální) funkční model
  - Systém má poskytovat nějaké služby. V úvodní studii jsme se orientačně dohodli, jaké to služby budou, teď je třeba to říci přesně.
- (konceptuální) datový model
  - Aby bylo možno služby poskytovat, je potřeba pracovat s daty. V úvodní studii jsme se orientačně dohodli, jaká data to budou, teď je třeba to říci přesně.
- (konceptuální) dynamický model
  - Systém, nebo jeho prvky často vykazují dynamiku – jejich chování se mění na základě různých okolností. Teď je třeba říci přesně jak.

Datový model

- notace
- konceptuální model tříd nebo ER-model (diagramy + textový popis)
- další integritní omezení, která nejsou zachycena v diagramech
- datový slovník

Funkční model

- notace
- model jednání (seznam událostí, příp. scénáře)
- pokud scénář obsahuje složitější aktivity, pak dekompozice těchto aktivit na popisy jednodušší – mohou to být podrobnější scénáře, diagramy aktivit, hierarchická sada diagramů datových toků (DFD - kontextový diagram + diagramy úrovně 0,1,… + popis)
- minispecifikace elementárních operací
- datový slovník

Dynamický model

- notace
- scénáře životních cyklů
- stavové diagramy
- datový slovník

Postup při analýze I.

- Vstup:
  - úvodní studie, katalog požadavků - CIM
- Výstup:
  - analytická dokumentace - PIM
- Postup:
  - paralelně zpracuj koncept a projekt

Postup při analýze II.

Zpracování konceptu:

- Vstup: CIM
  - deklarace záměru, odborný článek, katalog požadavků, seznam aktérů, seznam událostí, model jednání, kontext, 1.verze datového slovníku (z úvodní studie)
- Výstup: PIM
  - konceptuální analytický model (datový, funkční a dynamický model, 2.verze datového slovníku)

Postup při analýze III.

Zpracování projektu:

- Vstup:
  - seznam úloh, harmonogram (z úvodní studie)
- Výstup:
  - projektová dokumentace (projektový deník, seznam zdrojů, matice zodpovědností, harmonogram, plán testů, akceptační test)

Úvod do notace UML

Unified Modeling Language

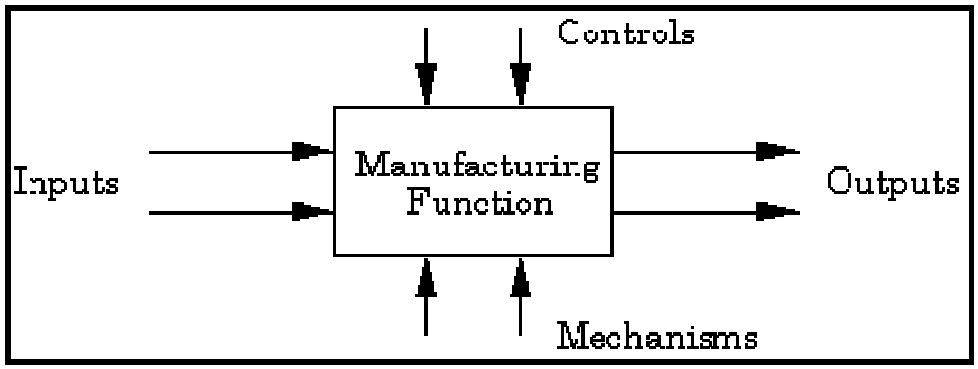
- jak se to zapisuje

Proč UML?

- UML je unifikovaný vyjadřovací prostředek pro dokumentaci obsahové části informatických projektů (dokumentace projektu obsahuje ještě tzv. projektovou dokumentaci, která popisuje projekt jako takový).
- UML umožňuje vyjádření dokumentace analýzy, návrhu i implementace.
- UML byl vyvinut na základě zkušeností s mnoha různými metodikami.
- UML umožňuje i vyjádření strukturovaného přístupu, ale byl určen zejména pro objektově-orientovaný přístup.

Alternativní notace

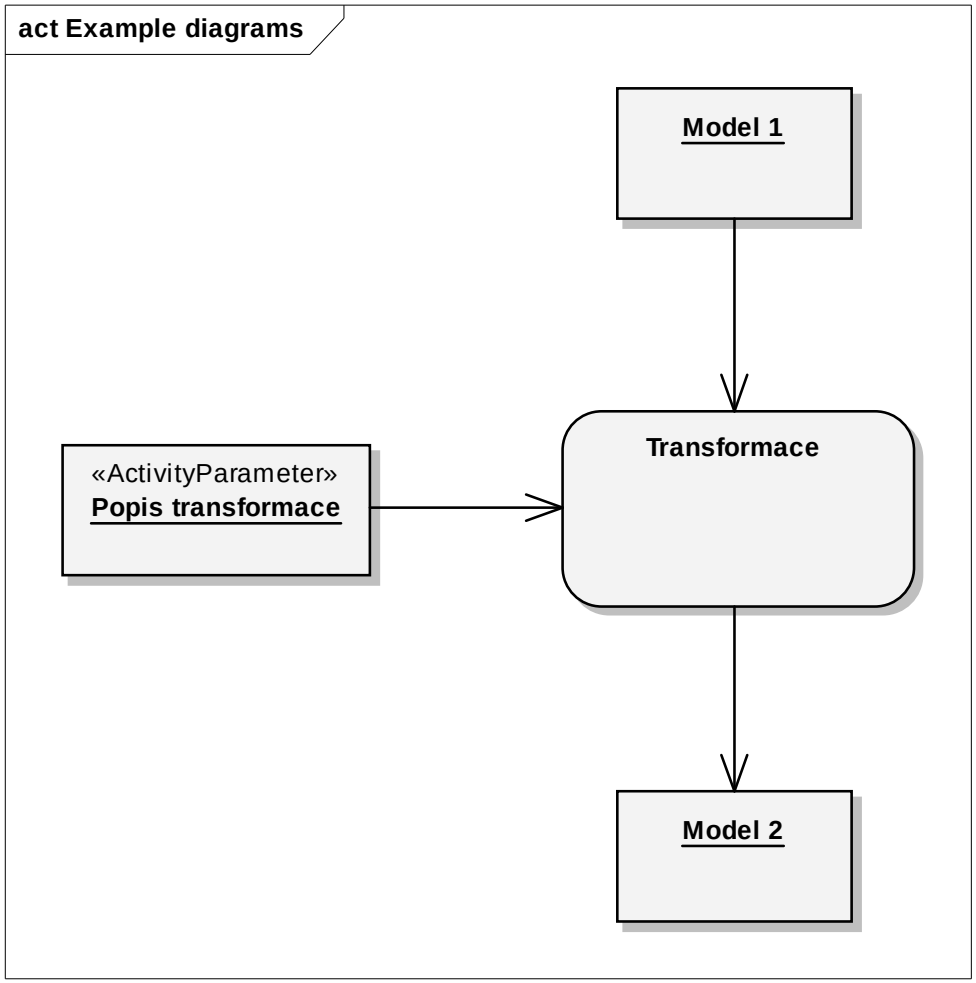
- Integrated Definition – IDEF (U.S. Air Force - http://www.idef.com)



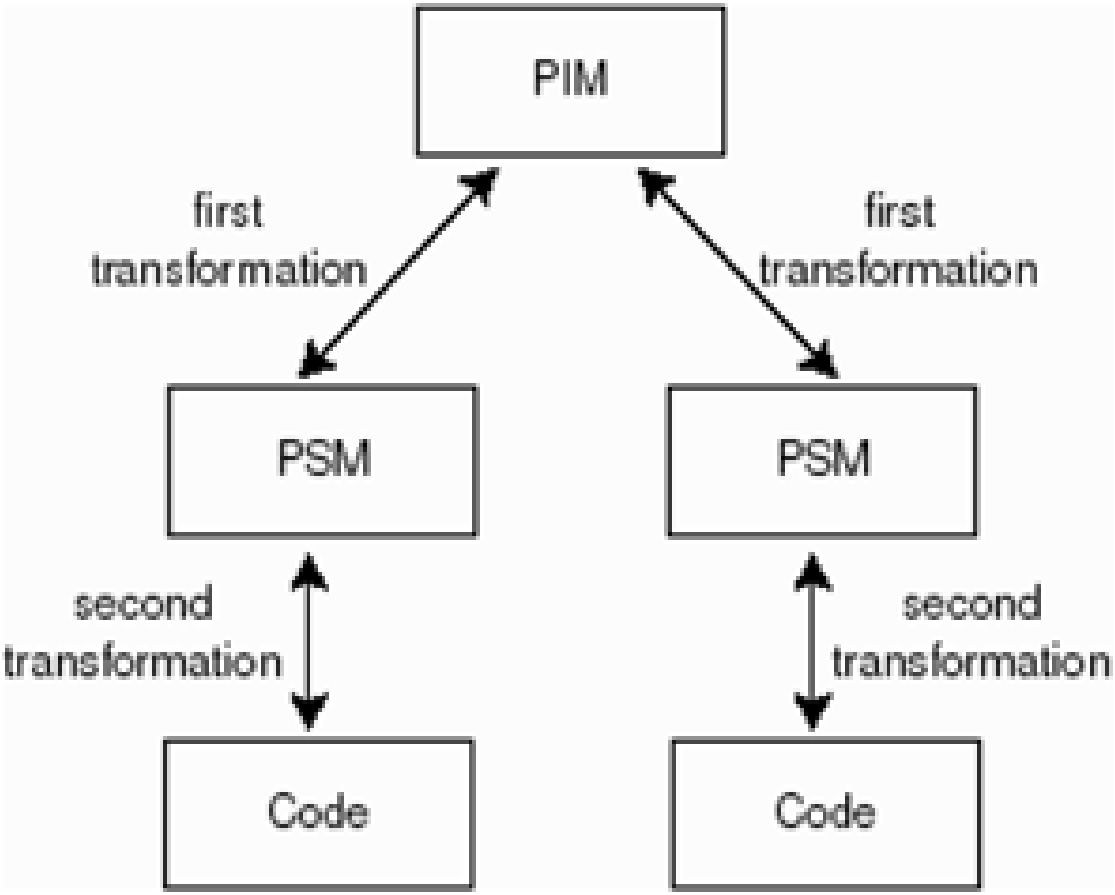
Alternativní notace

- Architecture of Integrated Information Systems – ARIS
  (prof. Scheer, SAP)
  - http://www.ids-scheer.com

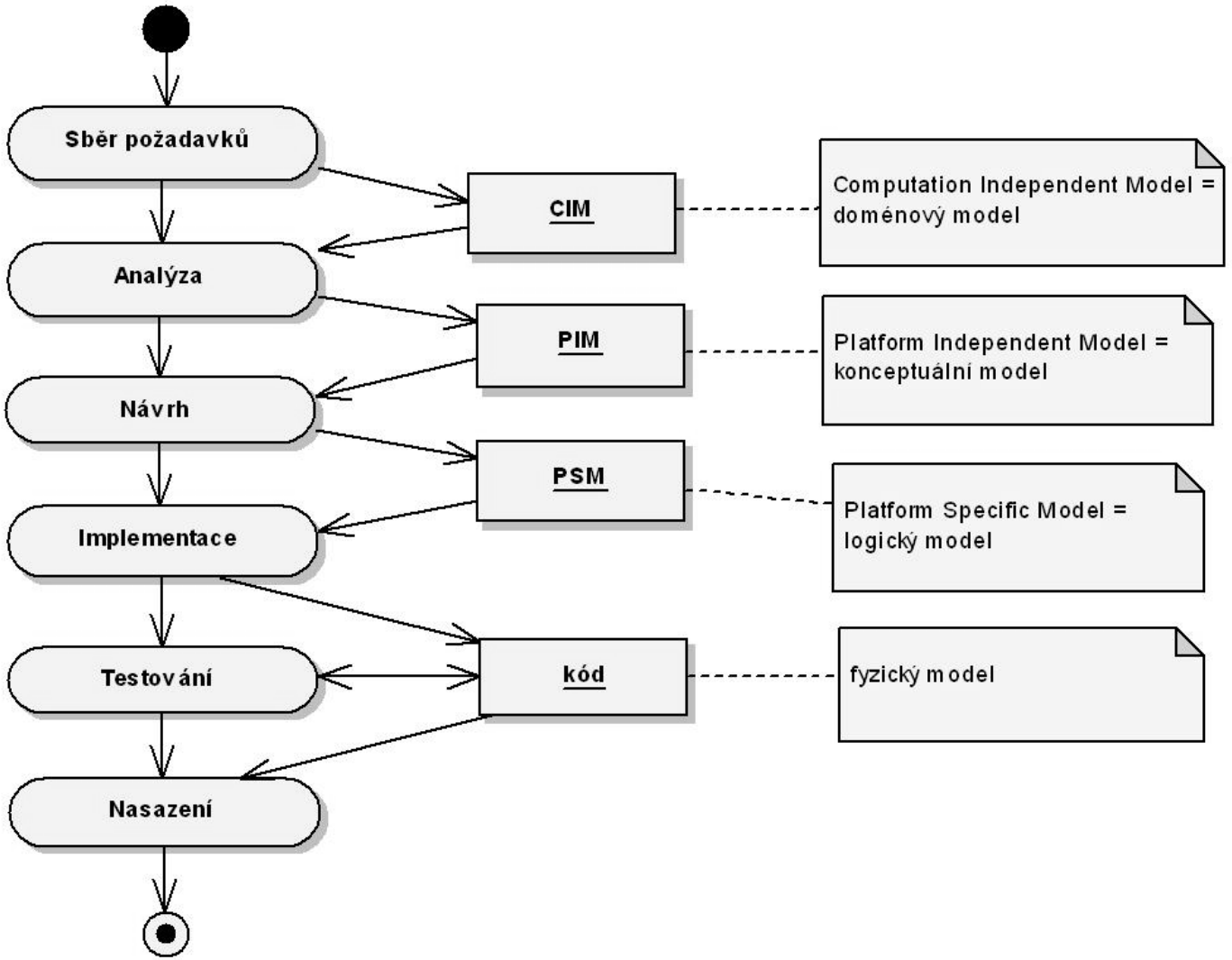
Vývoj software jako posloupnost transformací



Modelem řízený vývoj (MDD)



Modelem řízený vývoj v UML



6 dobrých zásad tvorby SW

- Vyvíjej iterativně a přírůstkově
- Řádně rozpoznej a zpracuj požadavky
- **Modeluj visuálně (=> UML)**
- Používej komponentové technologie
- Kontinuálně ověřuj kvalitu, testuj
- Snaž se být připraven na změny

Best practices
IBM Rational

Co to je UML?

- **UML (Unified Modeling Language)** je: *„Standard konsorcia OMG (Object Management Group) pro záznam, vizualizaci a dokumentaci artefaktů systémů s převážně softwarovou charakteristikou“.*

Co není UML?

- UML není žádná metodika – slouží pouze pro vyjádření artefaktů určitého typu, které metodiky požadují. Jsou metodiky, které UML intensivně využívají (např. UP, RUP), jiné nikoliv.
- UML není všespasitelné (U znamená unifikovaný, nikoli univerzální).

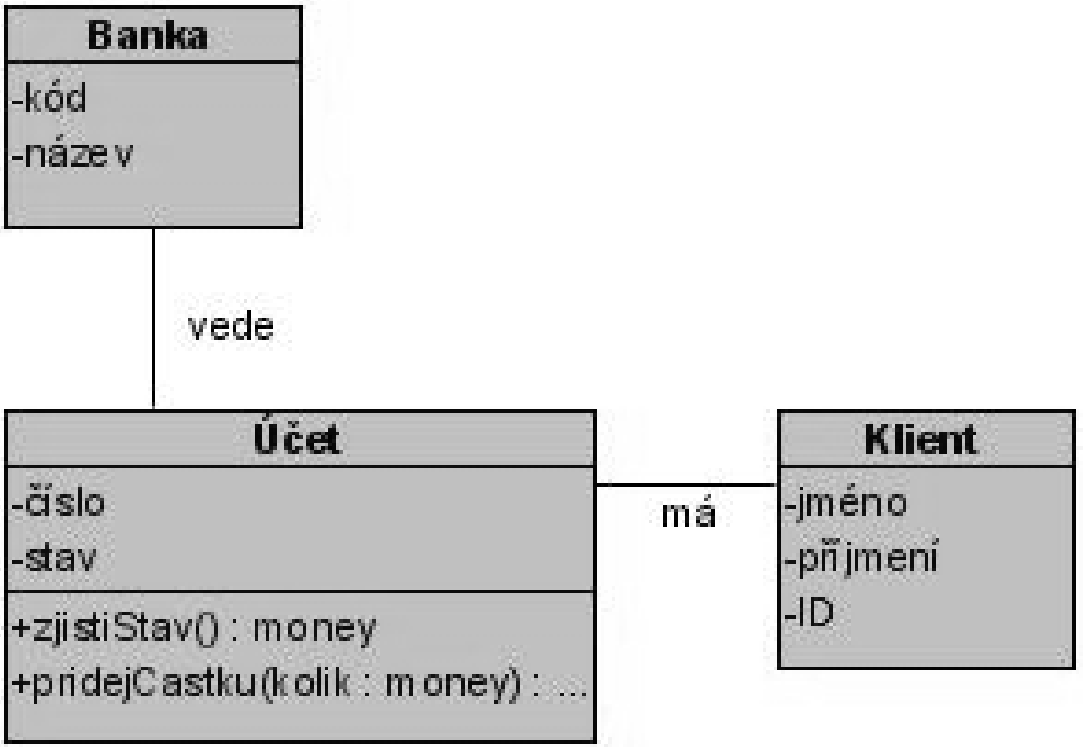
Tři úrovně využívání UML

- Jako **notaci** pro zachycení artefaktů, na kterých se je potřeba domluvit. Lze jej využít přímo (nakreslíme obrázek, abychom si upřesnili imlementaci), ale i zpětně (obrázek nám poslouží k pochopení existujícího kódu).
- Jako **dokumentační prostředek**, ve kterém dokumentujeme systém, či jeho části.
- Jako **programovací jazyk**, ze kterého se skutečná implementace (přesněji asi jen její kostra) generuje.

Základní diagramy UML (1.5)

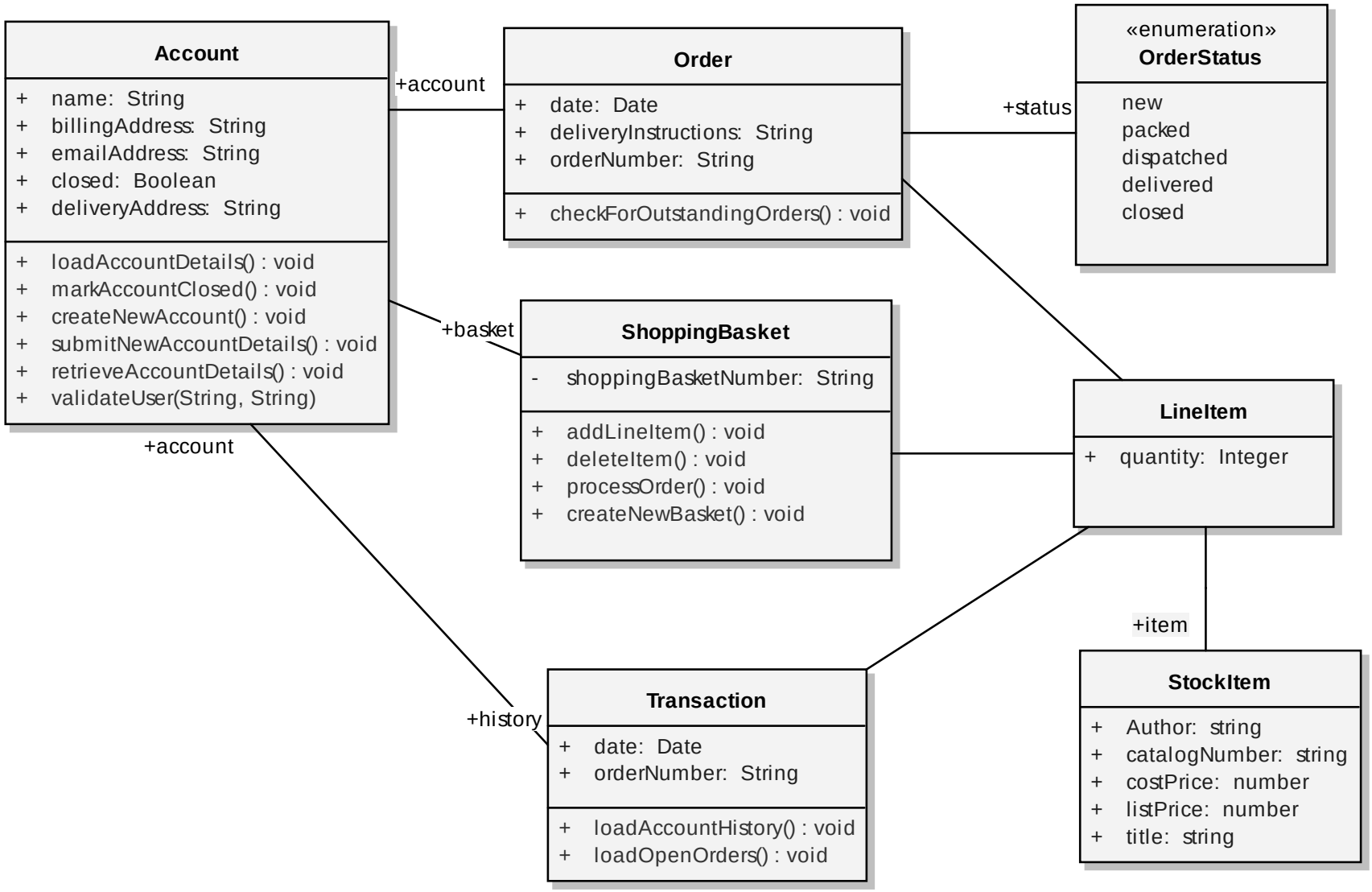
- diagramy tříd
- diagramy případů užití (model jednání)
- stavové diagramy
- scénáře (diagramy sekvencí)
- diagramy komunikace (spolupráce)
- diagramy aktivit (činností)
- diagramy komponent
- diagramy nasazení

Diagramy tříd

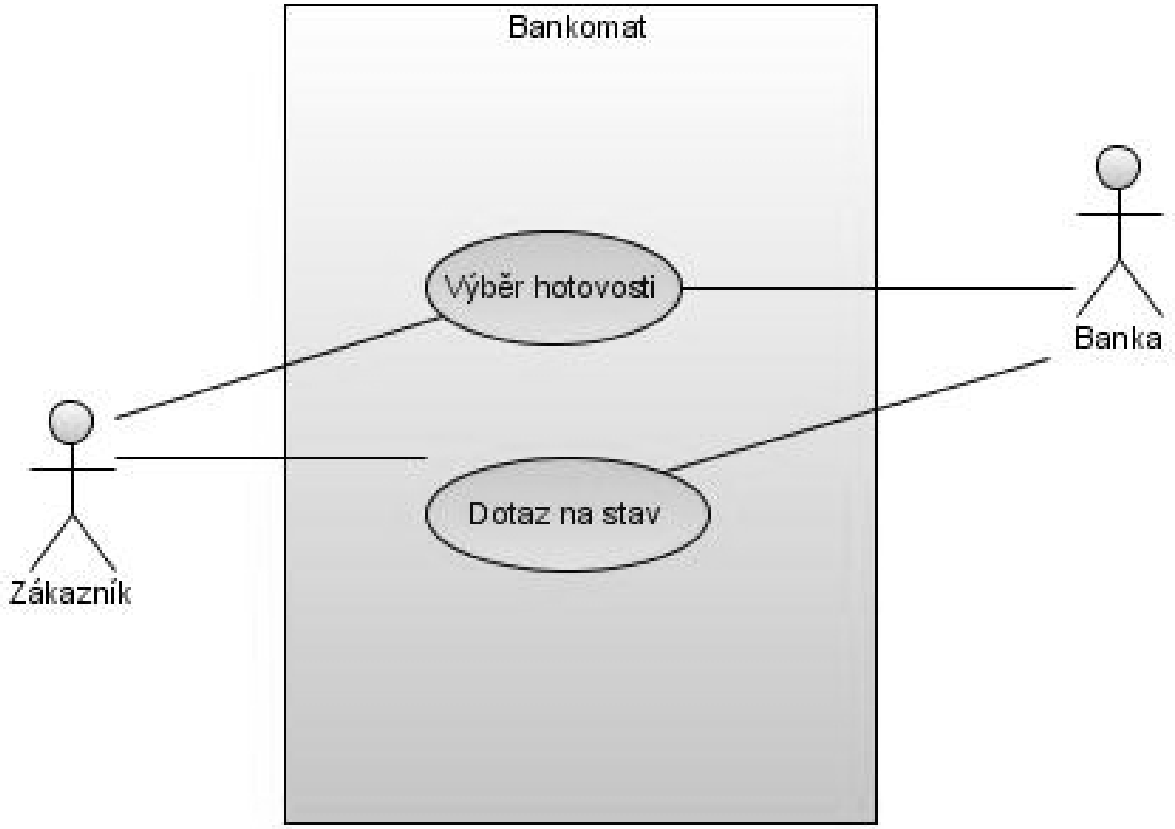


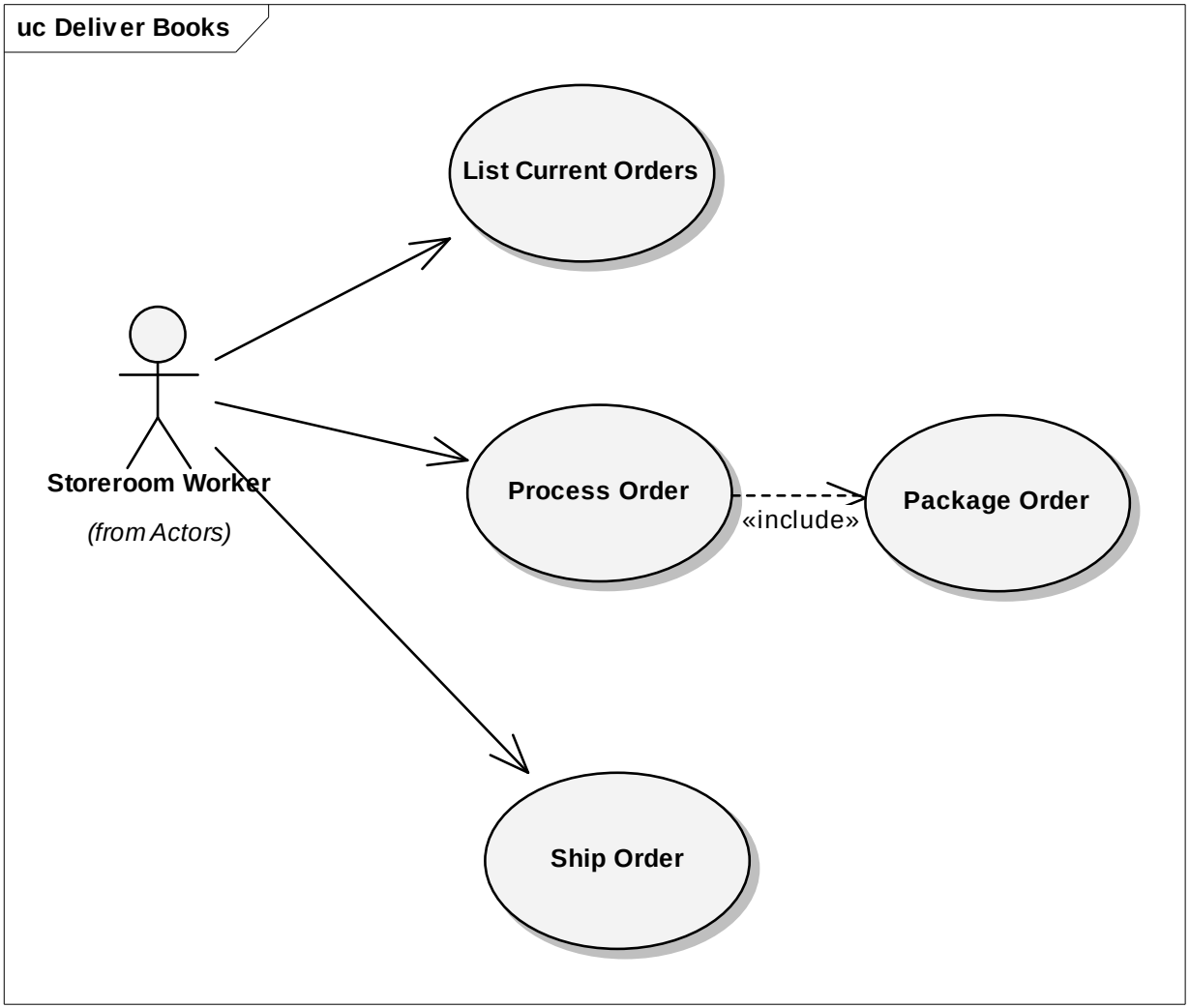
object Class Model

Konceptuální model pro elektronický obchod (PIM)

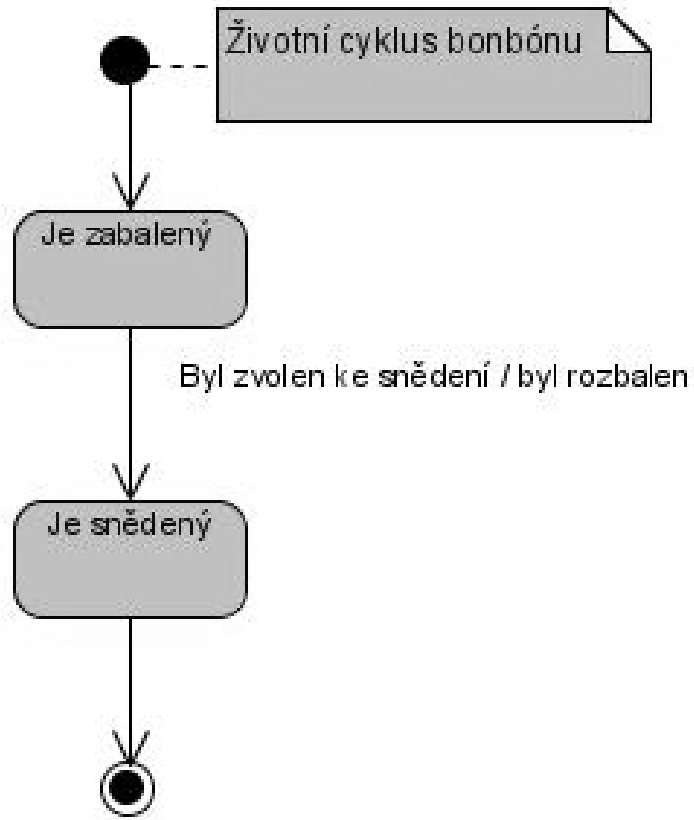


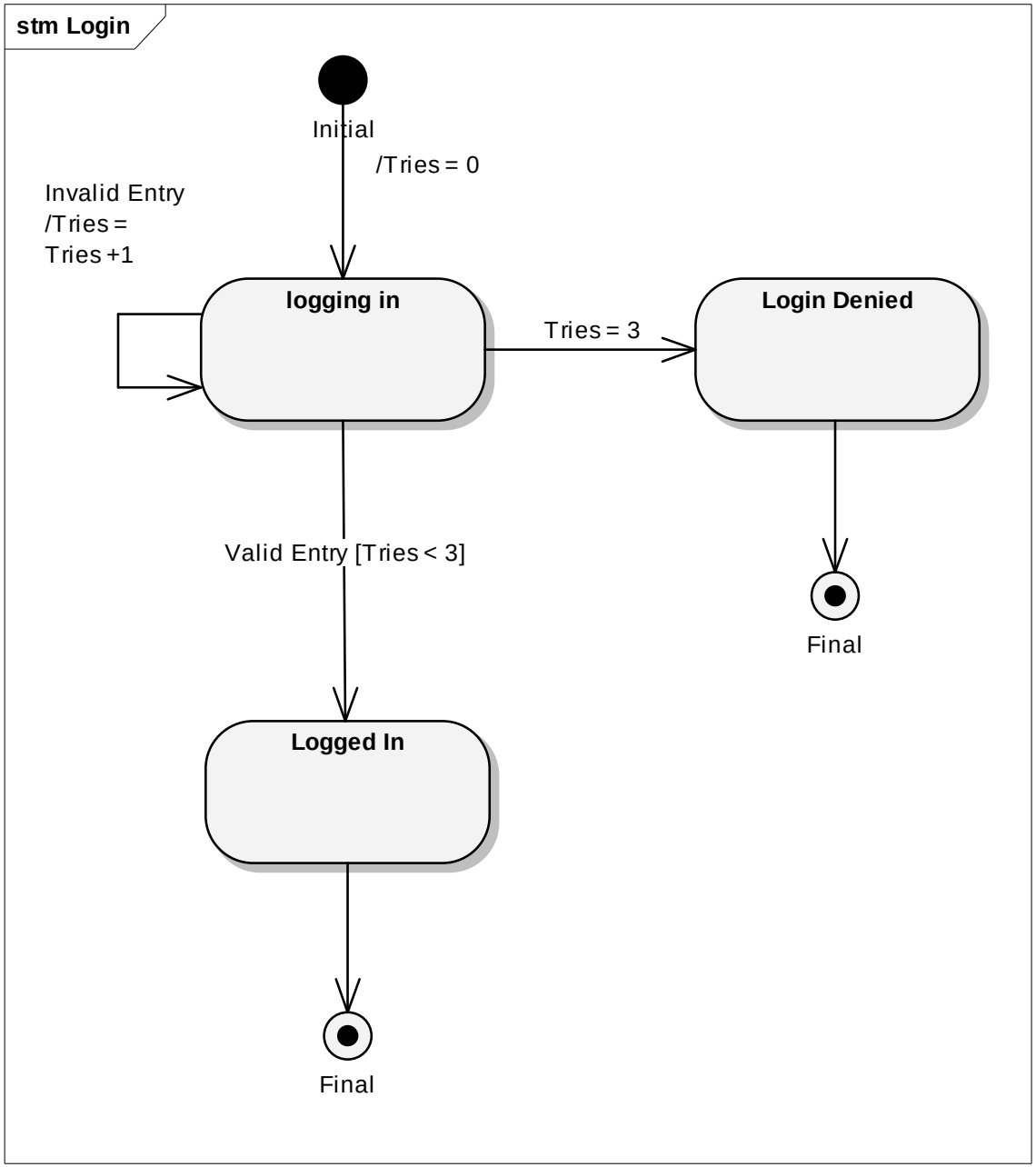
Diagramy případů užití (základ modelu jednání)



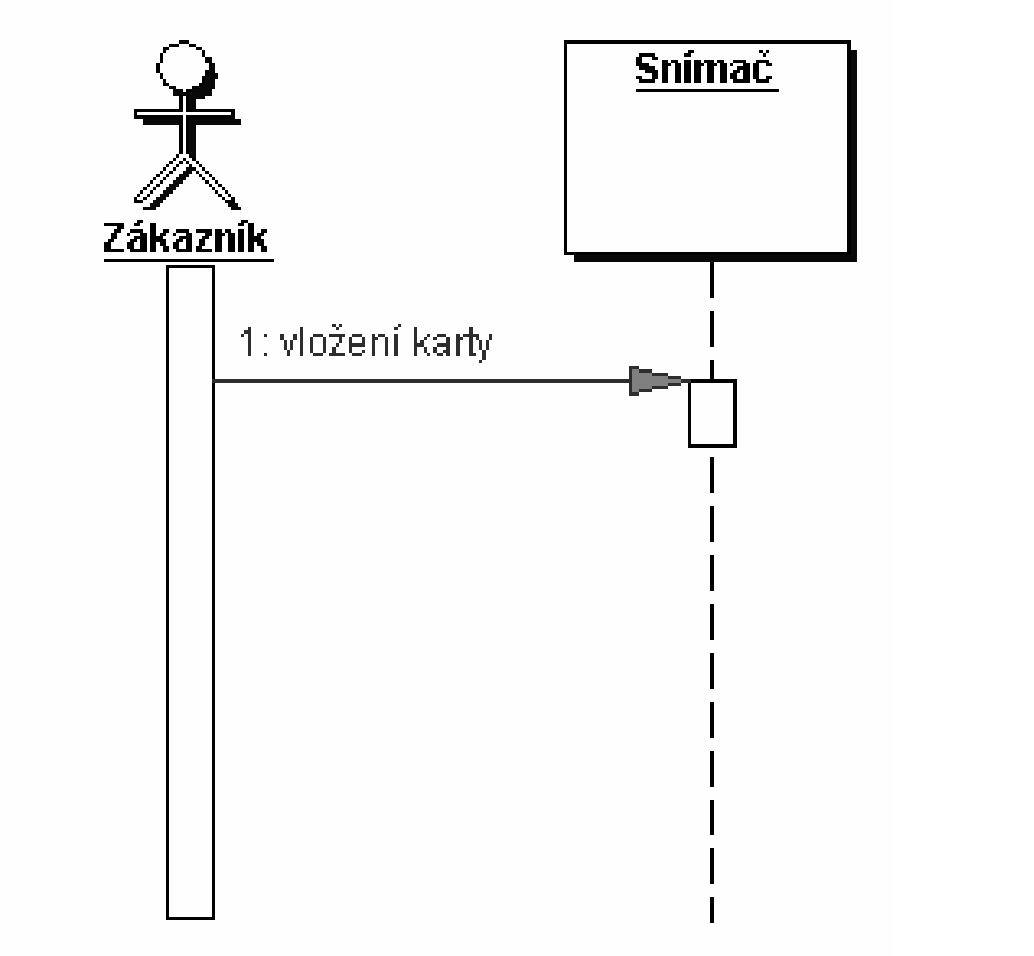

uc Deliver Books
List Current Orders
Storeroom Worker
(from Actors)
Process Order
«include»
Package Order
Ship Order

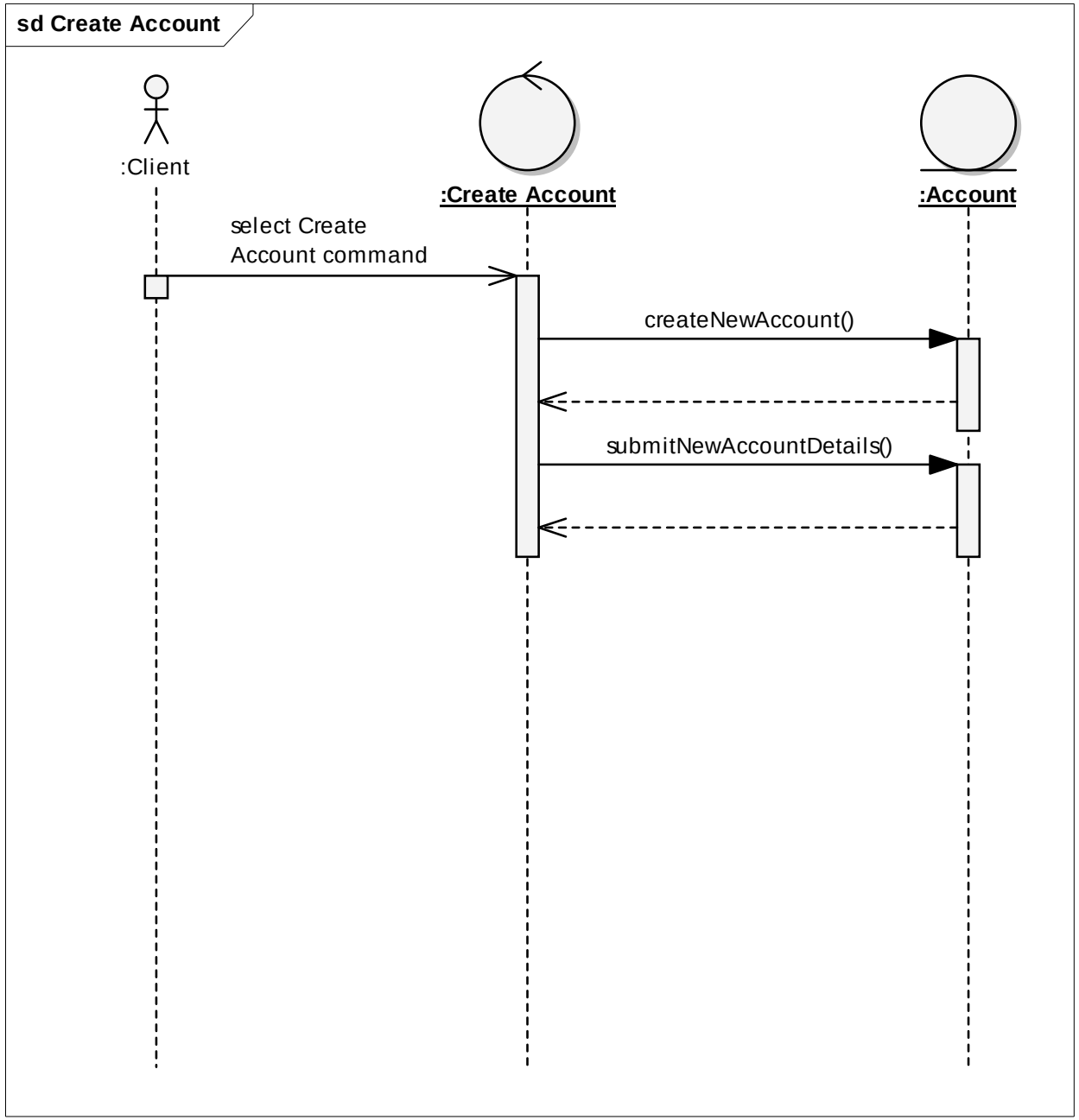
Stavové diagramy



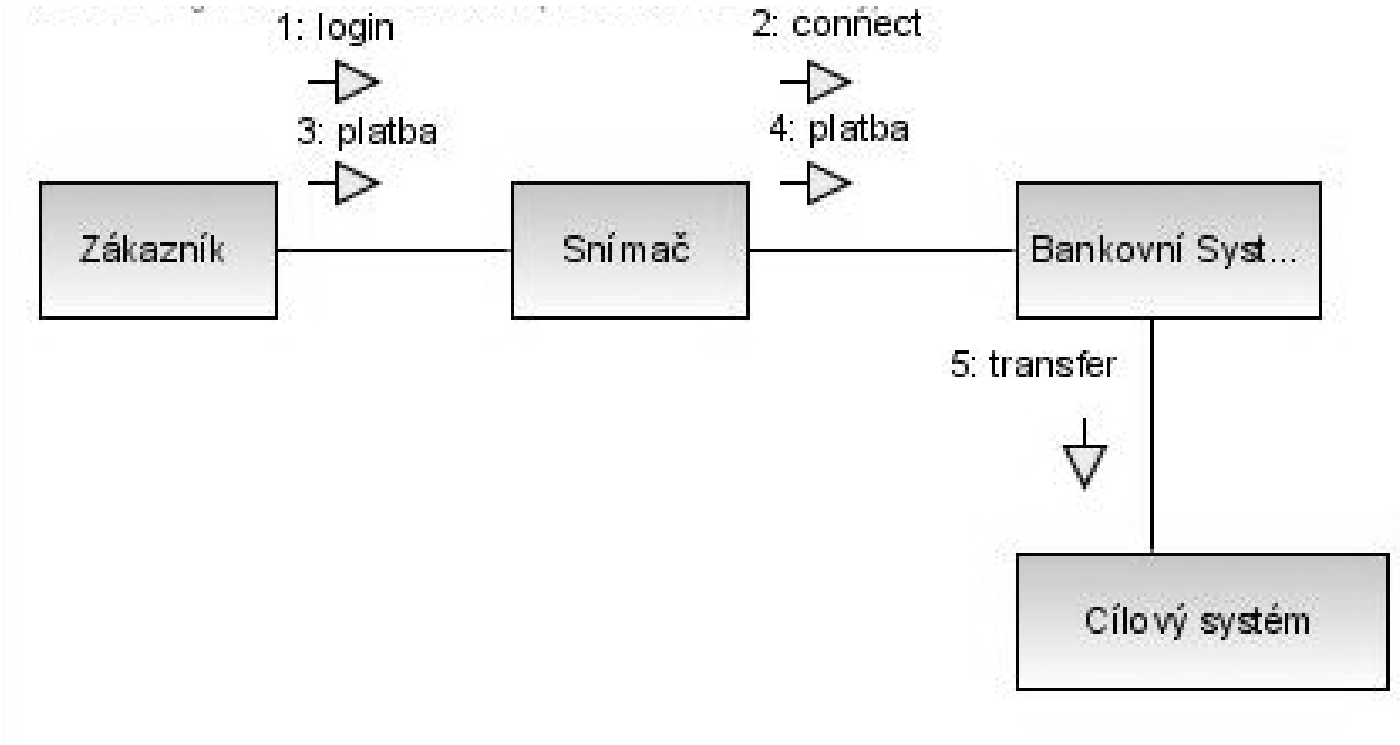

stm Login
Initial
/Tries = 0
Invalid Entry
/Tries =
Tries +1
logging in
Tries = 3
Login Denied
Valid Entry [Tries < 3]
Final
Logged In
Final

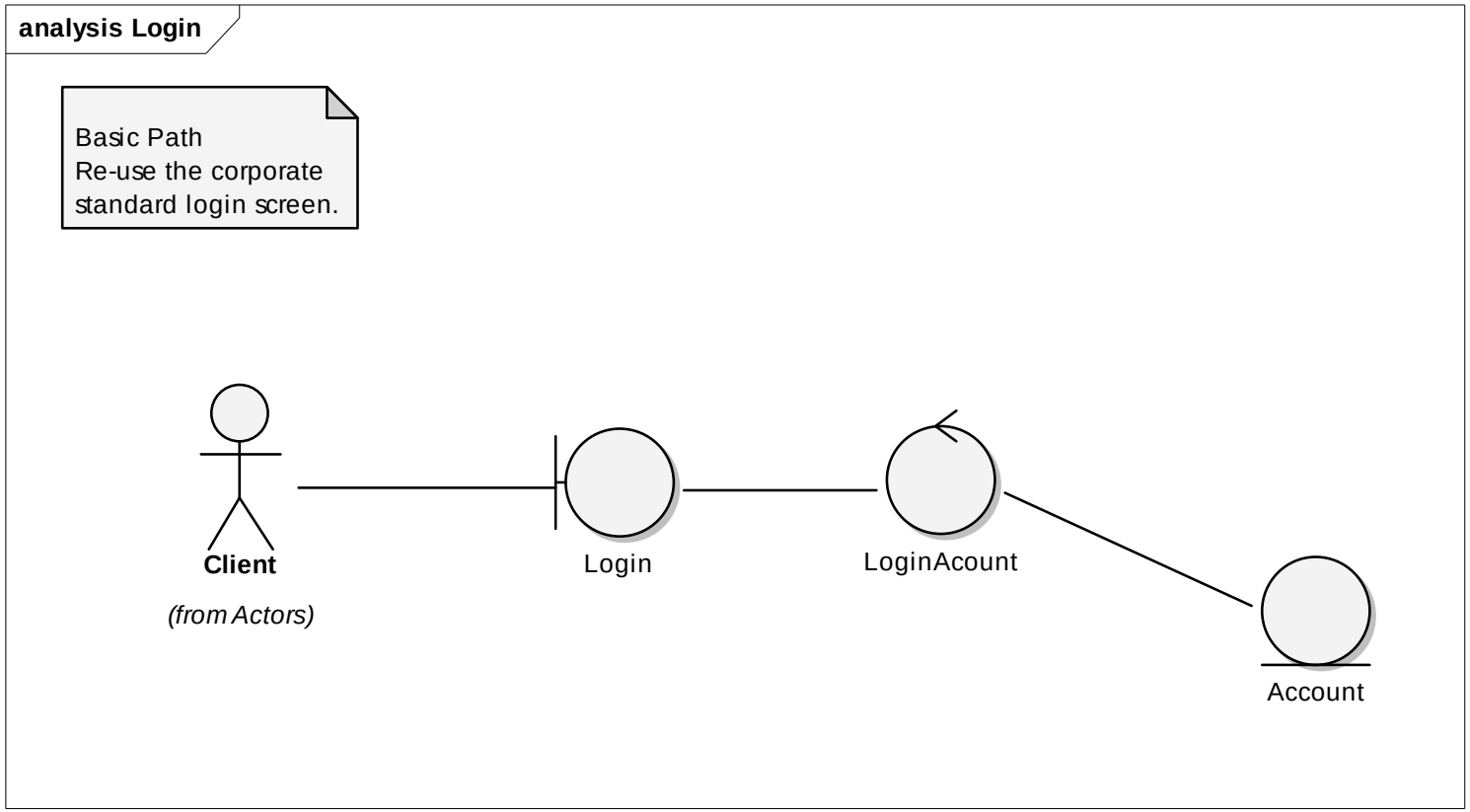
Scénáře (diagramy sekvencí)



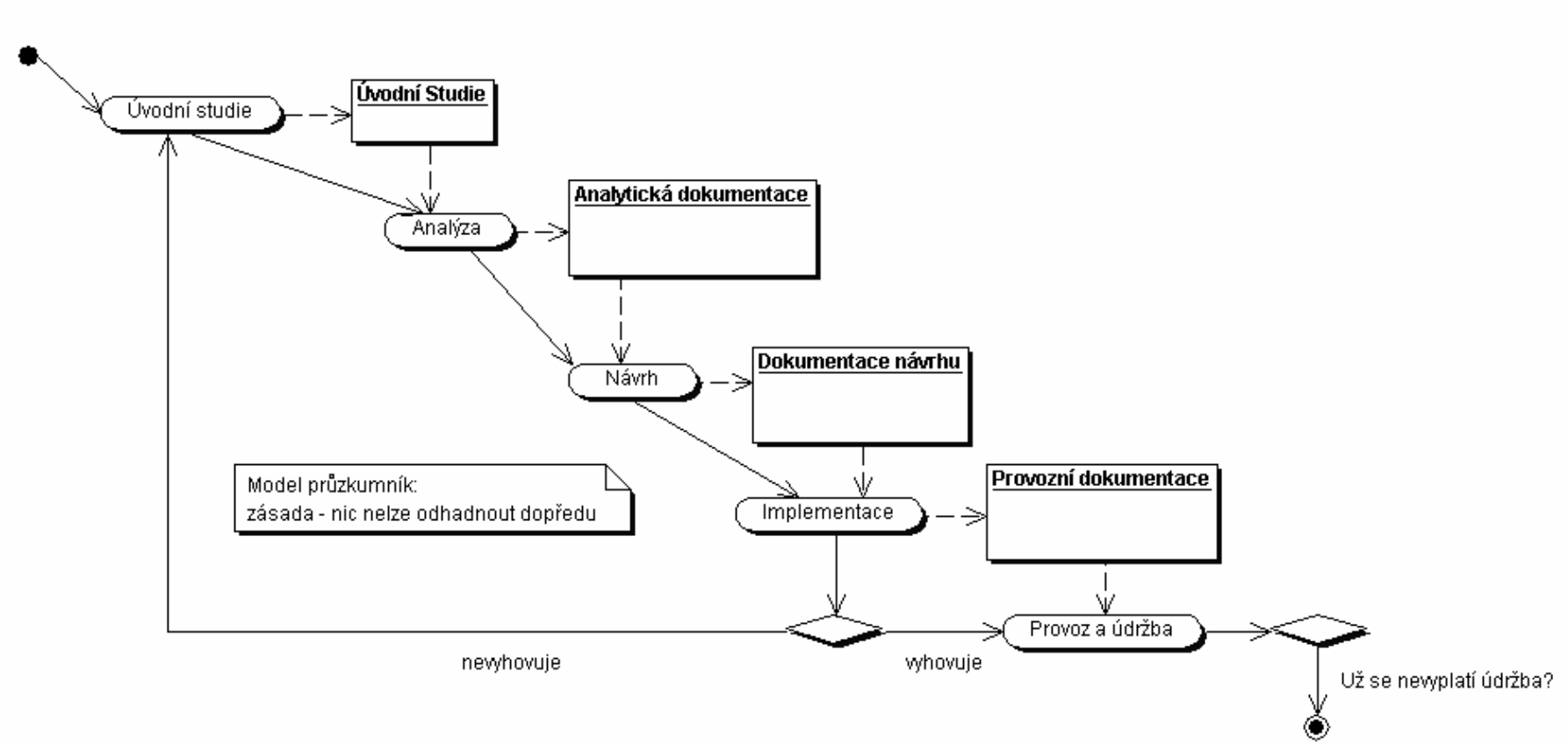

sd Create Account
:Client
:Create Account
:Account
select Create
Account command
createNewAccount()
submitNewAccountDetails()

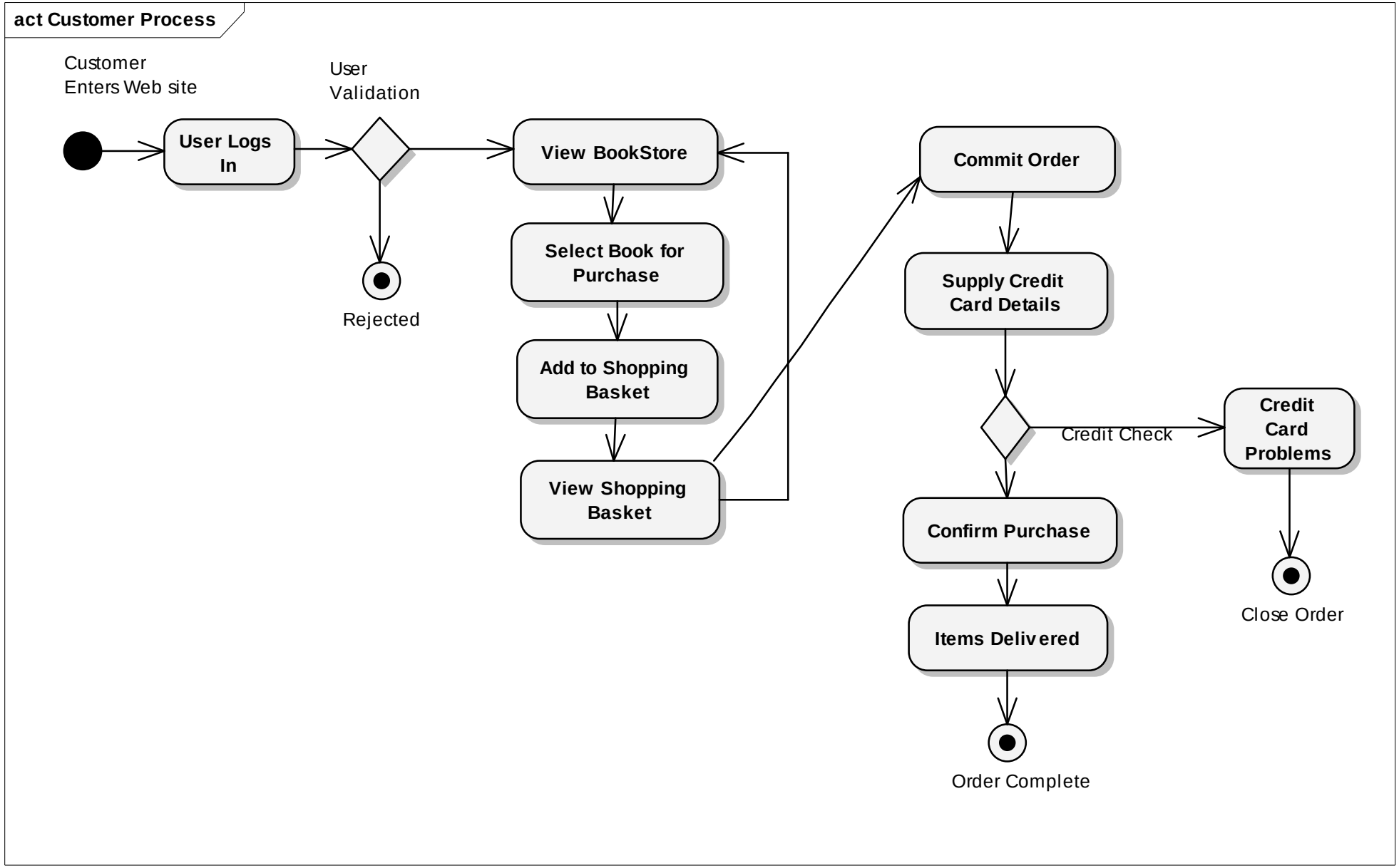
Diagramy komunikace (spolupráce)



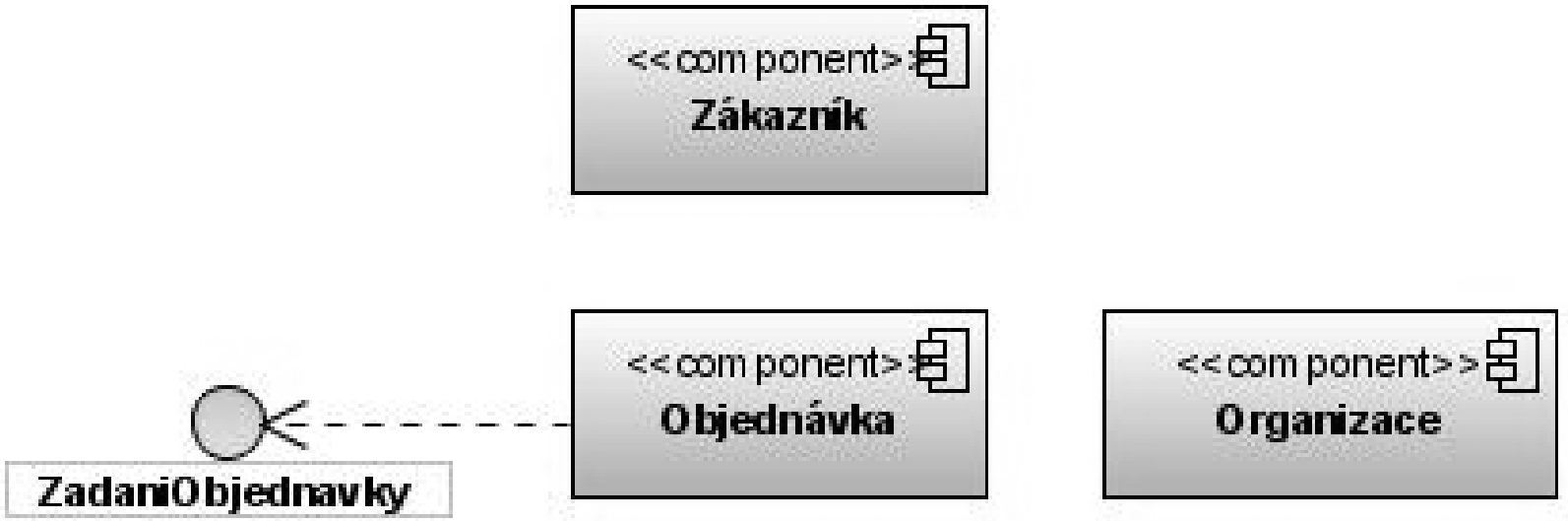

analysis Login
Basic Path
Re-use the corporate standard login screen.
Client
(from Actors)
Login
LoginAcount
Account

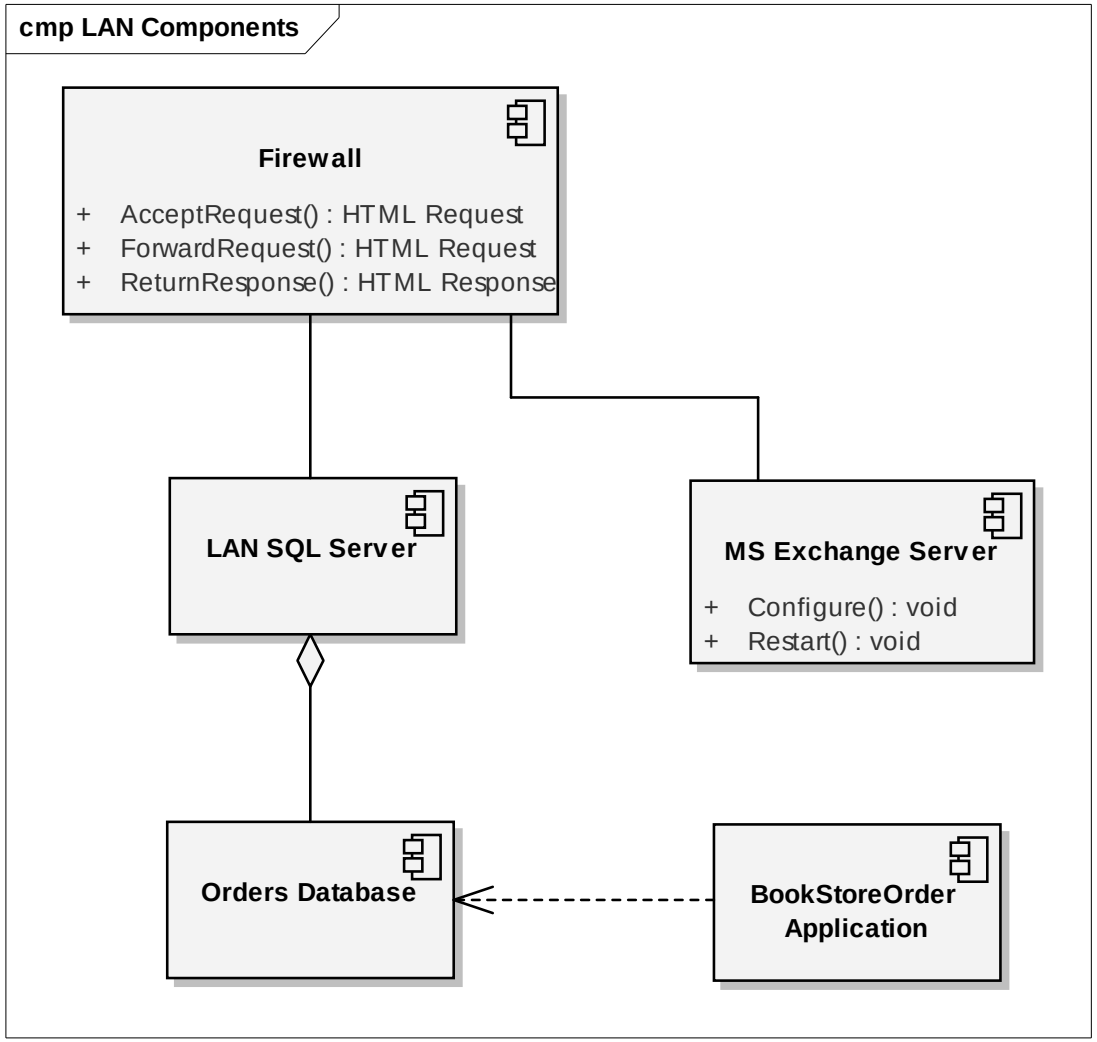
Diagramy aktivit (činností)



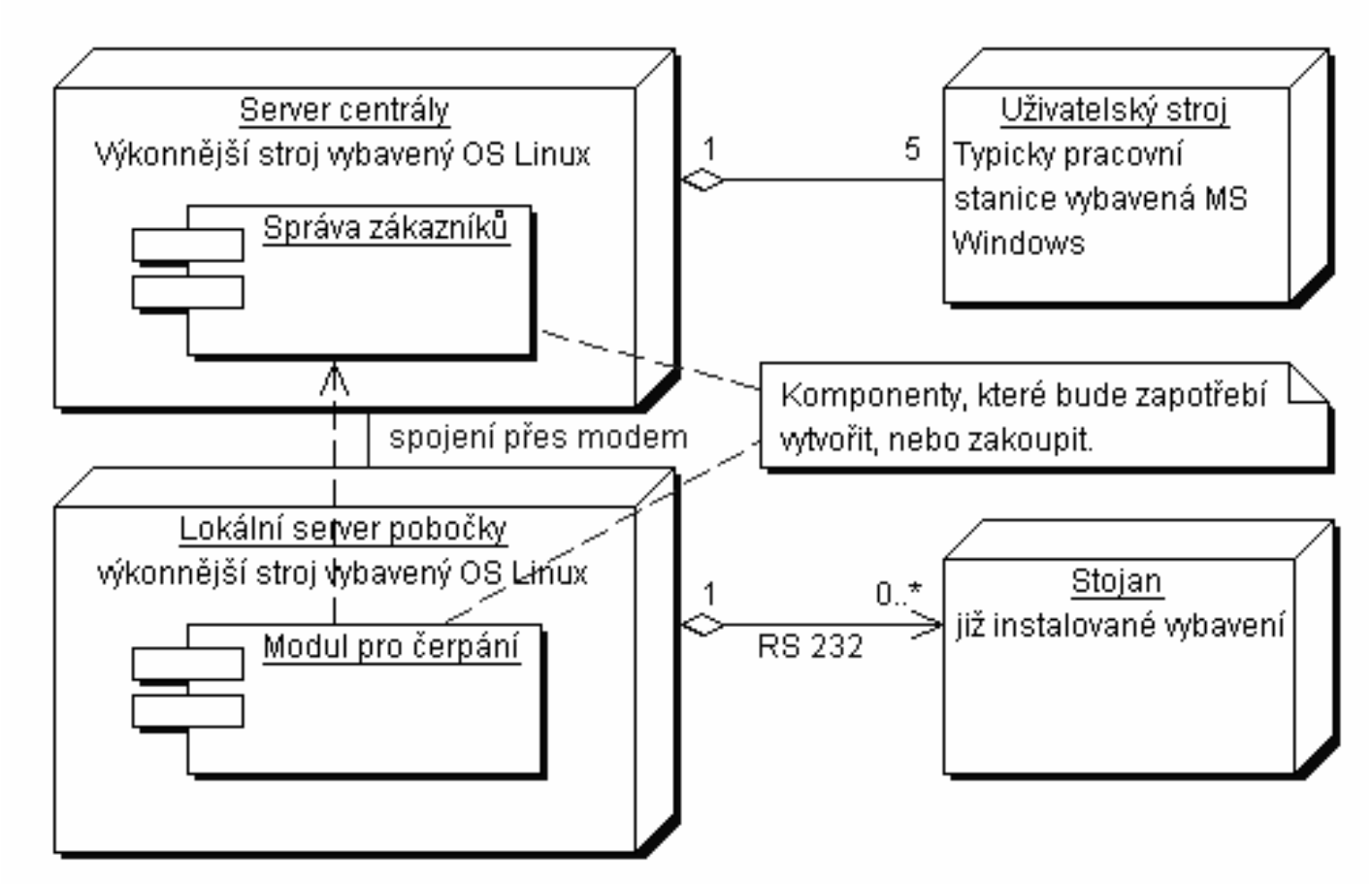

act Customer Process
Customer
Enters Web site
User
Validation
User Logs In
Rejected
View BookStore
Select Book for Purchase
Add to Shopping Basket
View Shopping Basket
Commit Order
Supply Credit Card Details
Credit Check
Credit Card Problems
Close Order
Confirm Purchase
Items Delivered
Order Complete

Diagramy komponent




cmp LAN Components
Firewall
+ AcceptRequest() : HTML Request
+ ForwardRequest() : HTML Request
+ ReturnResponse() : HTML Response
LAN SQL Server
MS Exchange Server
+ Configure() : void
+ Restart() : void
Orders Database
BookStoreOrder Application

Diagramy nasazení (deployment)



Historie UML

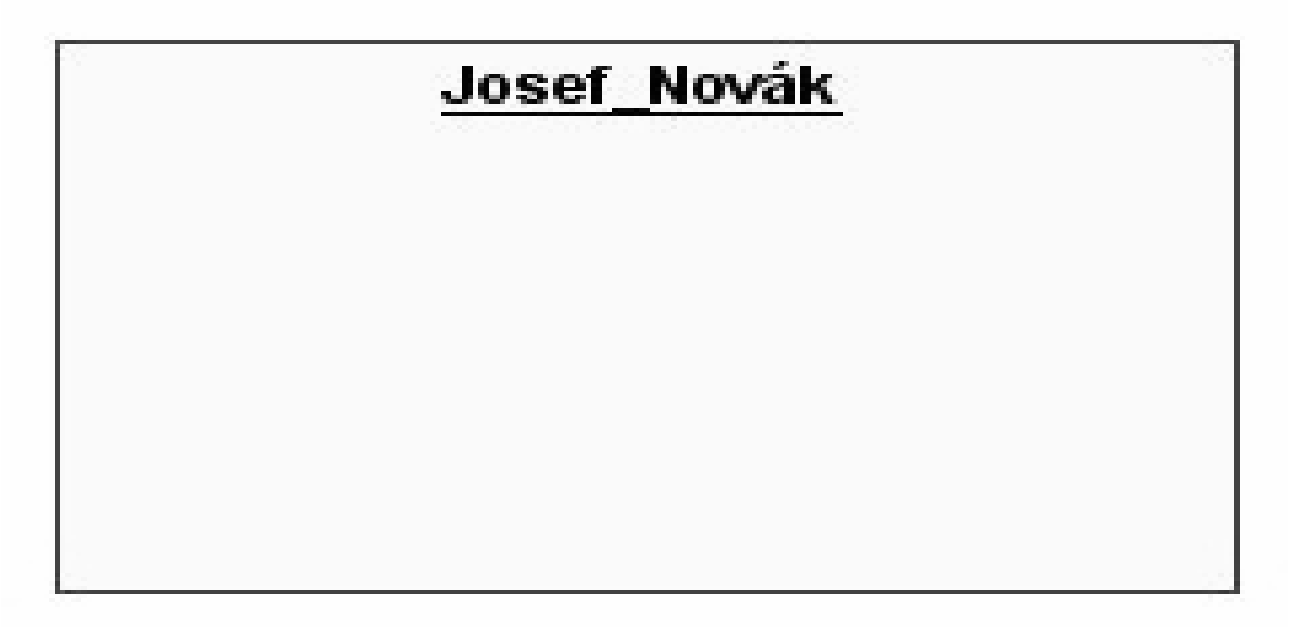
- **1994** – zahájení vývoje UML (Rumbaugh, Booch)
- **1995** – notace pro Unified Method 0.8, připojuje se Jacobson a model jednání, Rational Unified Process
- **1996** – UML 1.0 (zabudování připomínek, kooperace)
- **1997 – UML 1.1 (preciznější sémantika, přijat jako standard OMG)**
- **1999** – UML 1.3 (upřesněná verze 1.1)
- **2001** – UML 1.4 (komponenty)
- **2003 – UML 1.5 (diagramy aktivit, sémantika akcí)**
- **2005 – UML 2.0 (nové diagramy, změny)** - původně plánované ukončení květen 2002, ale ☺ (příliš mnoho hlav,obtížné)
- 2006 – UML 2.1 (další snaha o upřesnění sémantiky)

Složky UML

- Elementární prvky (infrastruktura)
- Diagramy UML (superstruktura)
- OCL (Object Constraint Language – jazyk pro popis omezení)
- Výměnný formát (buď jen model – XMI, nebo včetně diagramů konvertovaných do SVG)

Některé základní ikony

Ikona pro objekt

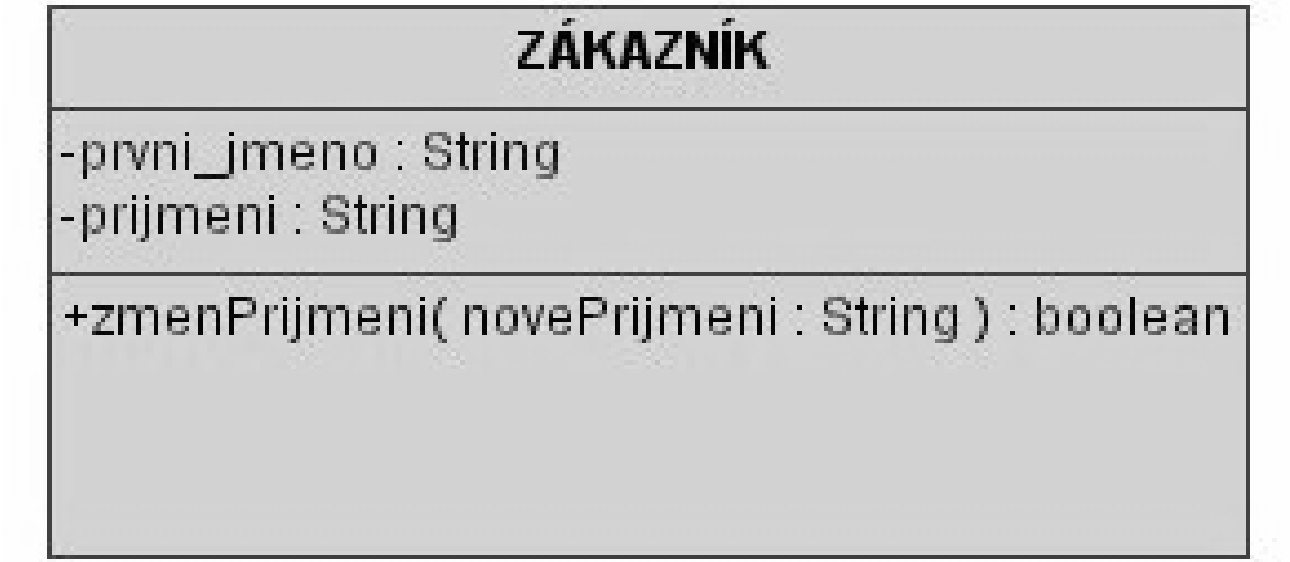


Jedinečný objekt identifikovaný svým jménem

Ikona pro třídu

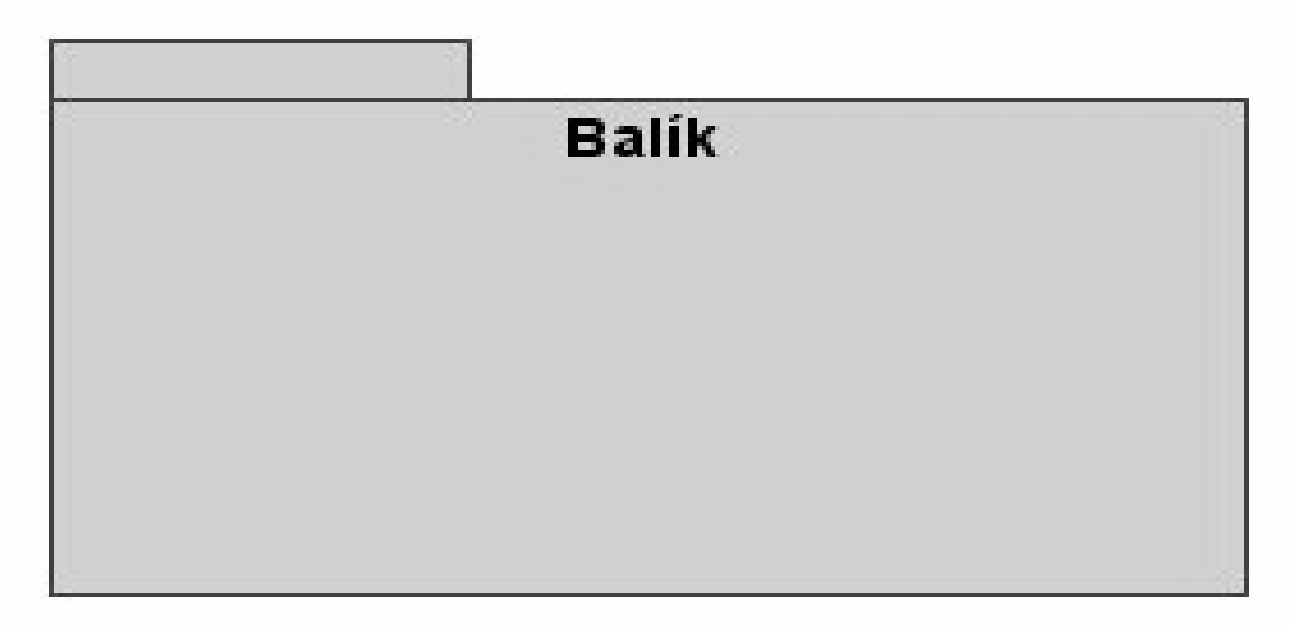
Třída zahrnuje objekty stejného typu – se stejnými atributy a operacemi

2D-symbol pro třídu



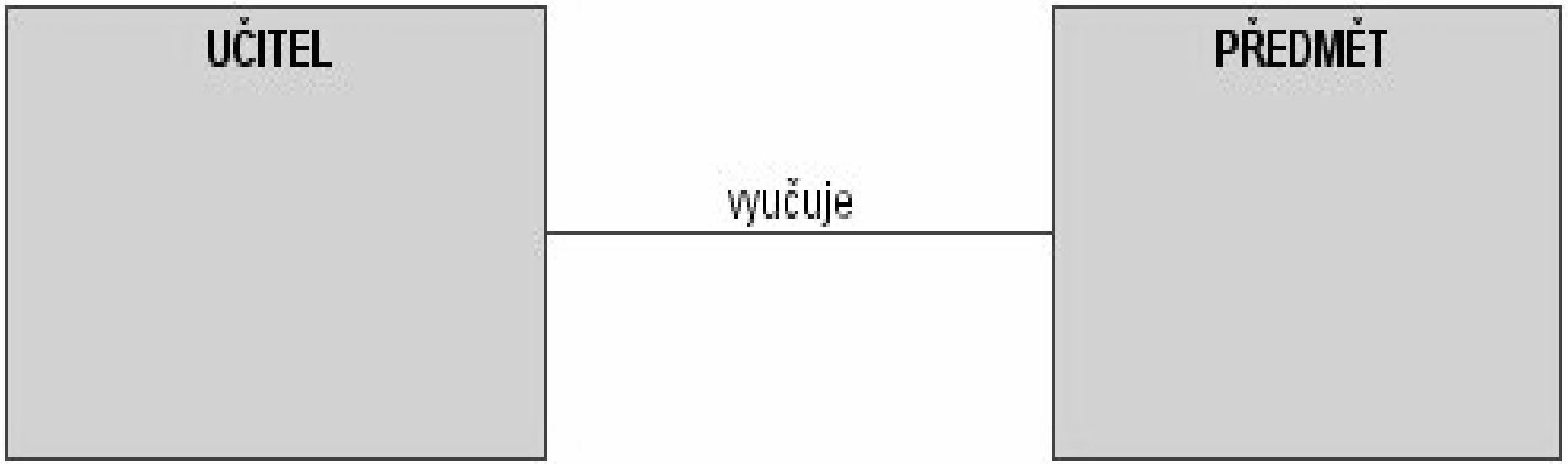
Třída s atributy a operacemi

Ikona pro balík (package)



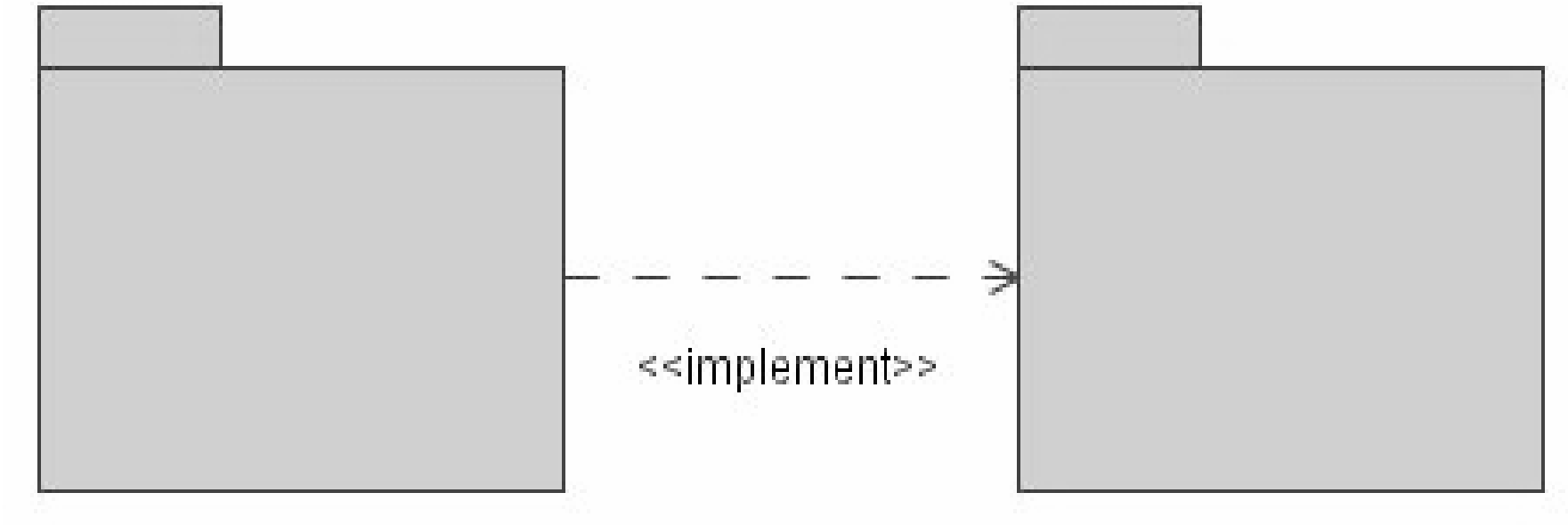
Logická skupina elementů

Spojnice



Vyjadřuje vztah (v tomto případě symetrický vztah mezi třídami)

Spojnice



Vyjadřuje závislost (v tomto případě, že něco implementuje něco jiného)

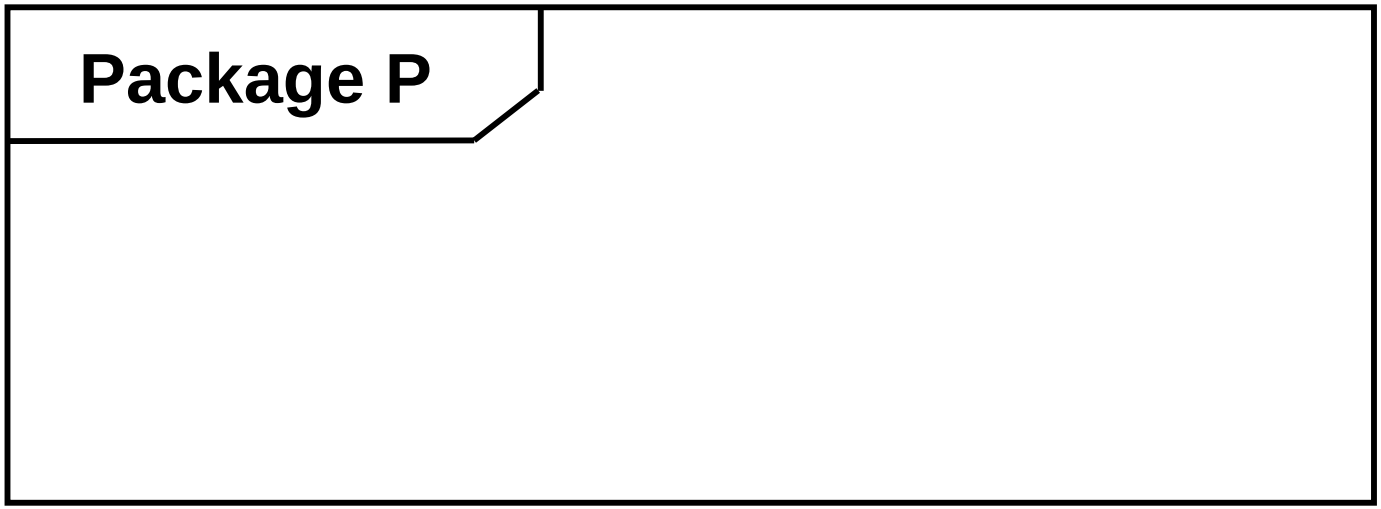
Ikona pro případ užití (use case)

Záznam o případu použití – vně systému může nastat událost, na kterou musí systém reagovat

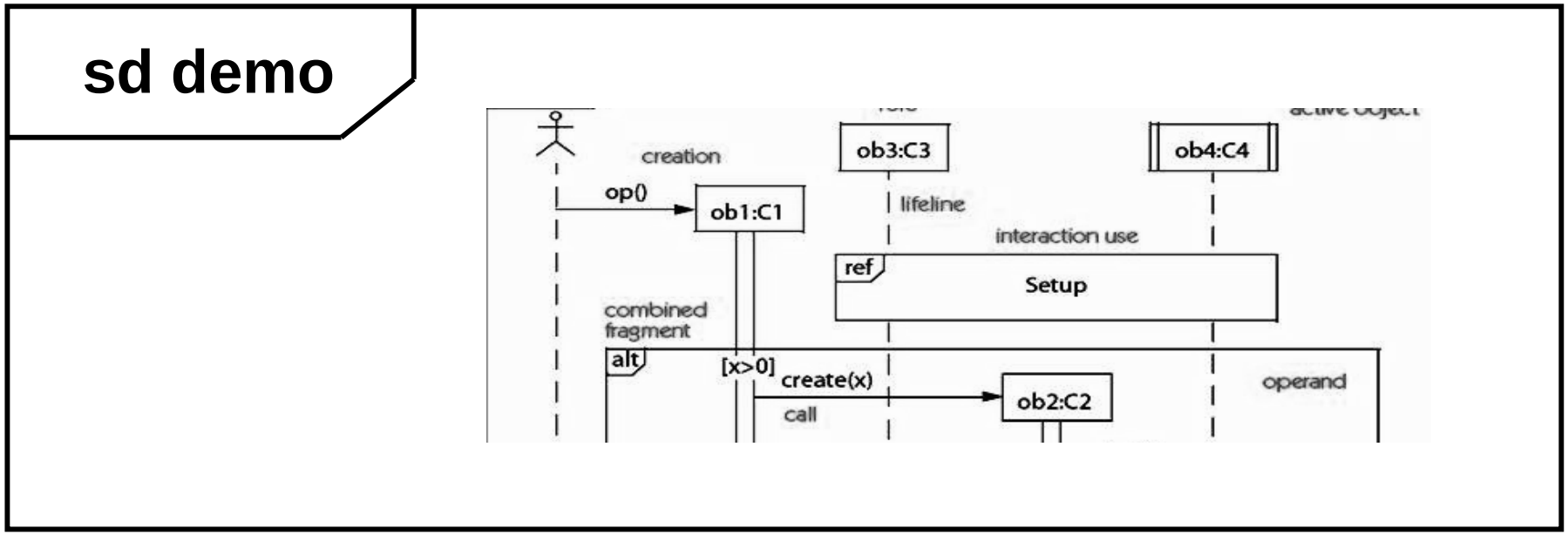
Ikona pro aktéra

Uživatelská role nebo spolupracující systém

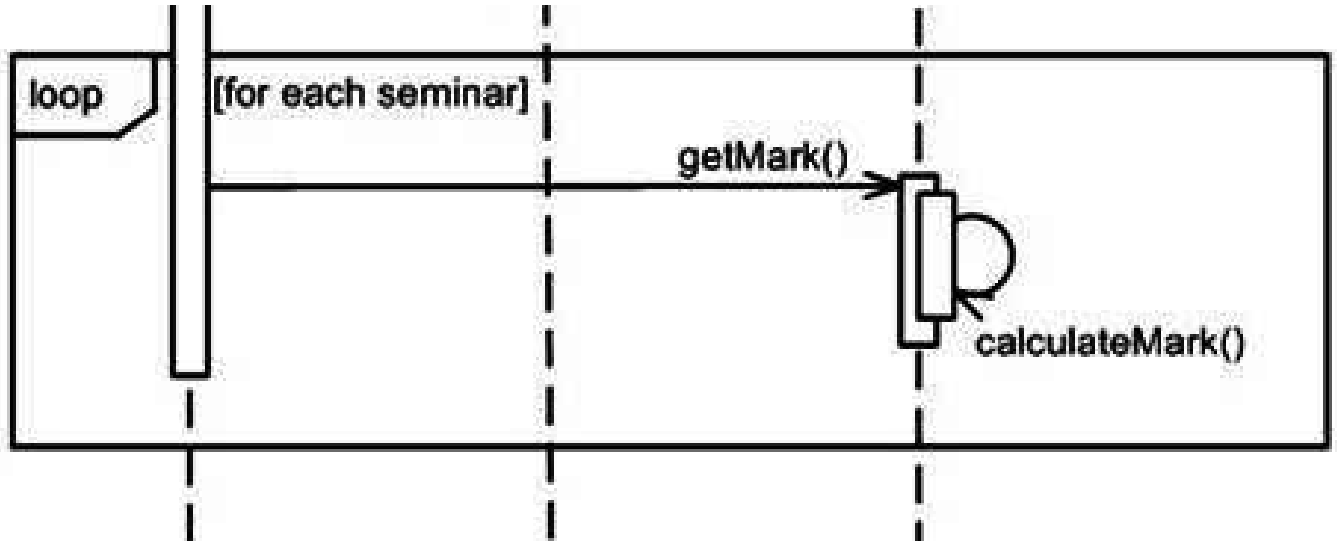
Nová ikona – rámec (frame)



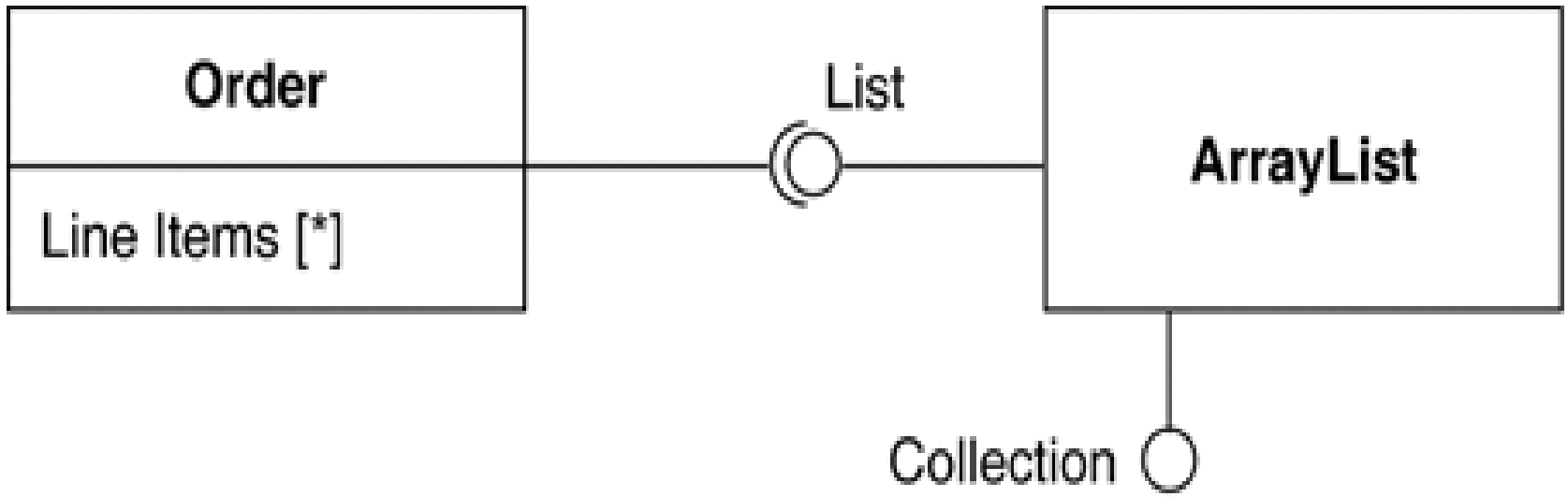
Příklad použití rámce pro diagram



Příklad použití rámce ve scénáři



Nová ikona pro „interface“



Artefakt

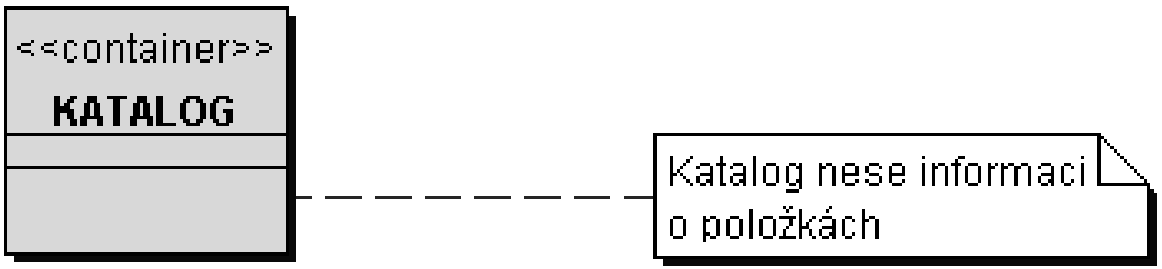
- Artefakt je specifikace fyzického kousku informace, která je použita nebo produkována při vývoji softwaru nebo při jeho nasazení a použití.
- Příkladem artefaktu může být modelový soubor, zdrojový soubor, skript, vykonatelný program, tabulka v databázi, libovolná součást dodávky, textový dokument, či zpráva.
- Instance artefaktu:

Použití artefaktu v diagramech

- Komponenta „Objednávka“ je „manifestována“ artefaktem „Objednávka.jar“

Poznámky

- Poznámka obsahuje libovolný text
- Zobrazuje se jako obdélník s „přehnutým rohem“
- Může být přidružena k elementu
- Může obsahovat stereotyp (např. **<<constraint>>** - pak se jedná o specifikaci omezení)



Struktura UML 2.0

Specifikace standardu UML 2.0 je rozdělena do 4 částí:

- **definice infrastruktury** – notace UML (syntax), definuje základní elementy, jádro UML společné pro UML a související standardy, např. MOF či CWM
- **definice superstruktury** - sémantika definovaná pomocí metamodelu, popis prvků metamodelu, konstrukty používané uživateli UML – elementy diagramů a diagramy
- **definice výměnné struktury** (diagram interchange) – pro export a import modelů a diagramů, formát pro výměnu dokumentů (diagramů) mezi různými nástroji, specifikace převodu do výměnných formátů (CORBA IDL, XML DTD)
- **Object Constraint Language** (OCL) – jazyk pro formálně přesný popis různých omezení platných v modelu

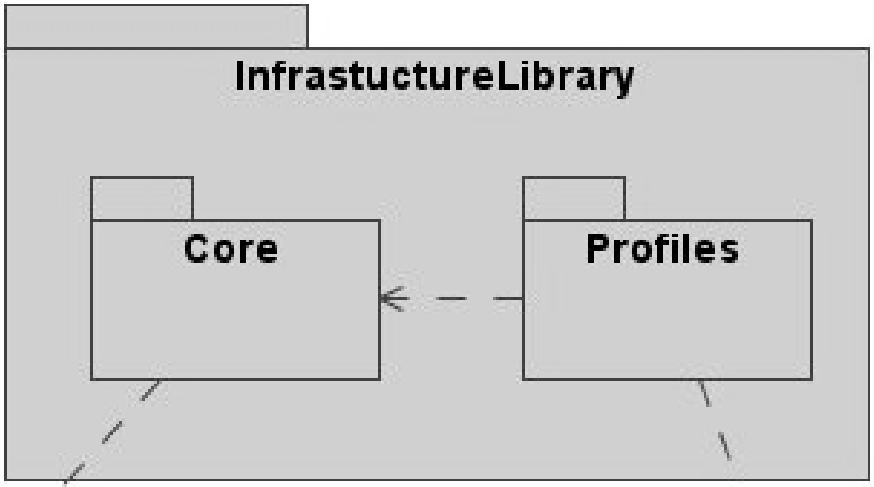
Aktuální verze UML 2.0

- **Superstructure Specification 2.0**: Revised Final Adopted Specification (formal/05-07-04) August 2005
- **Infrastructure Specification 2.0**: Revised Final Adopted Specification (formal/05-07-05) March 2006
- **Diagram Interchange Specification 2.0**: Convenience Document (ptc/05-06-04) June 2005
- **OCL 2.0**: Available Specification (formal/06-05-01) May 2006

`http://www.uml.org`

Infrastruktura UML (2.0)

Infrastruktura UML 2.0 je reprezentována balíkem InfrastructureLibrary.

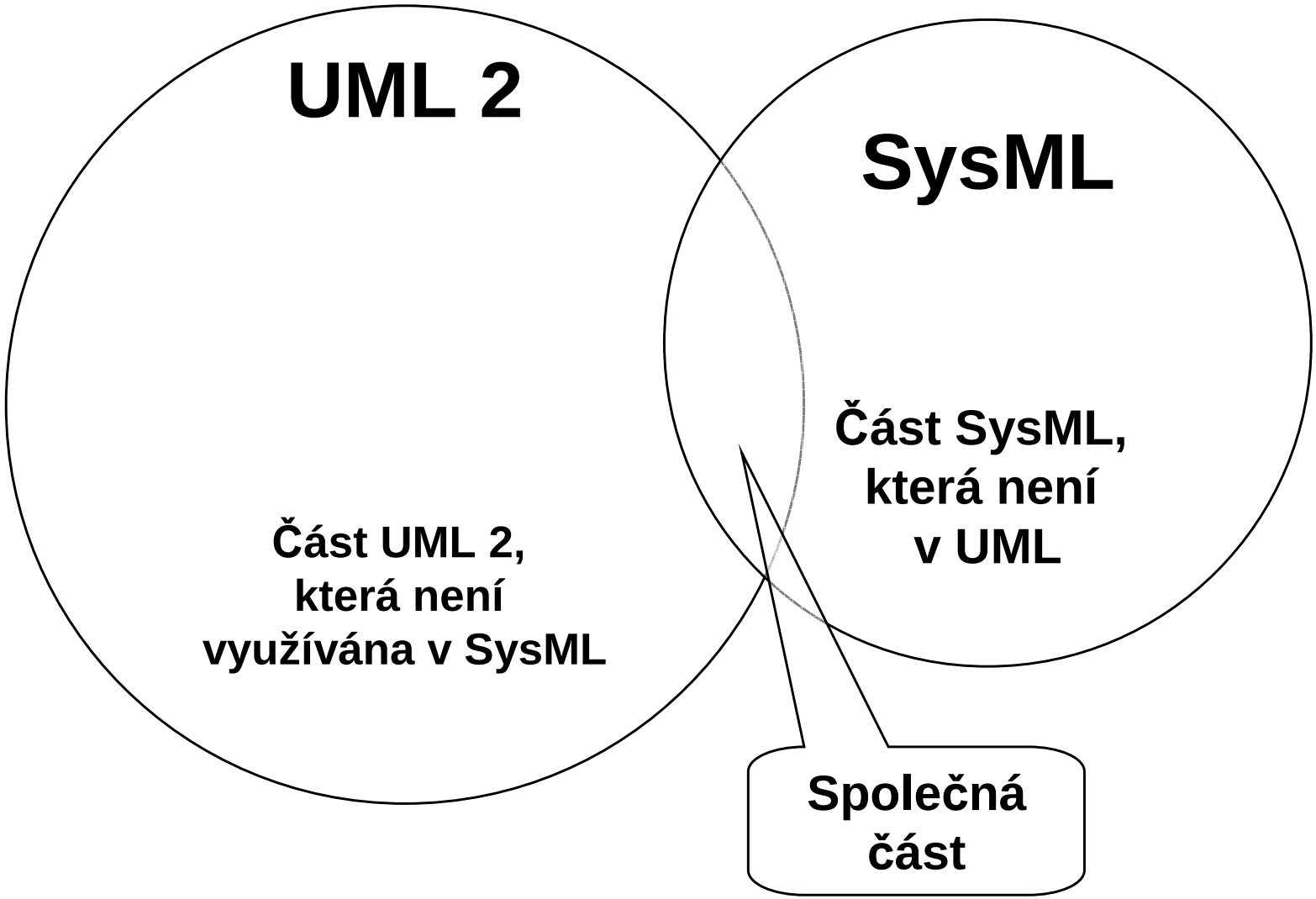


Balík Core představuje jádro definice infrastruktury. Jedná se o společný metamodel pro UML, MOF, CWM a Profily.

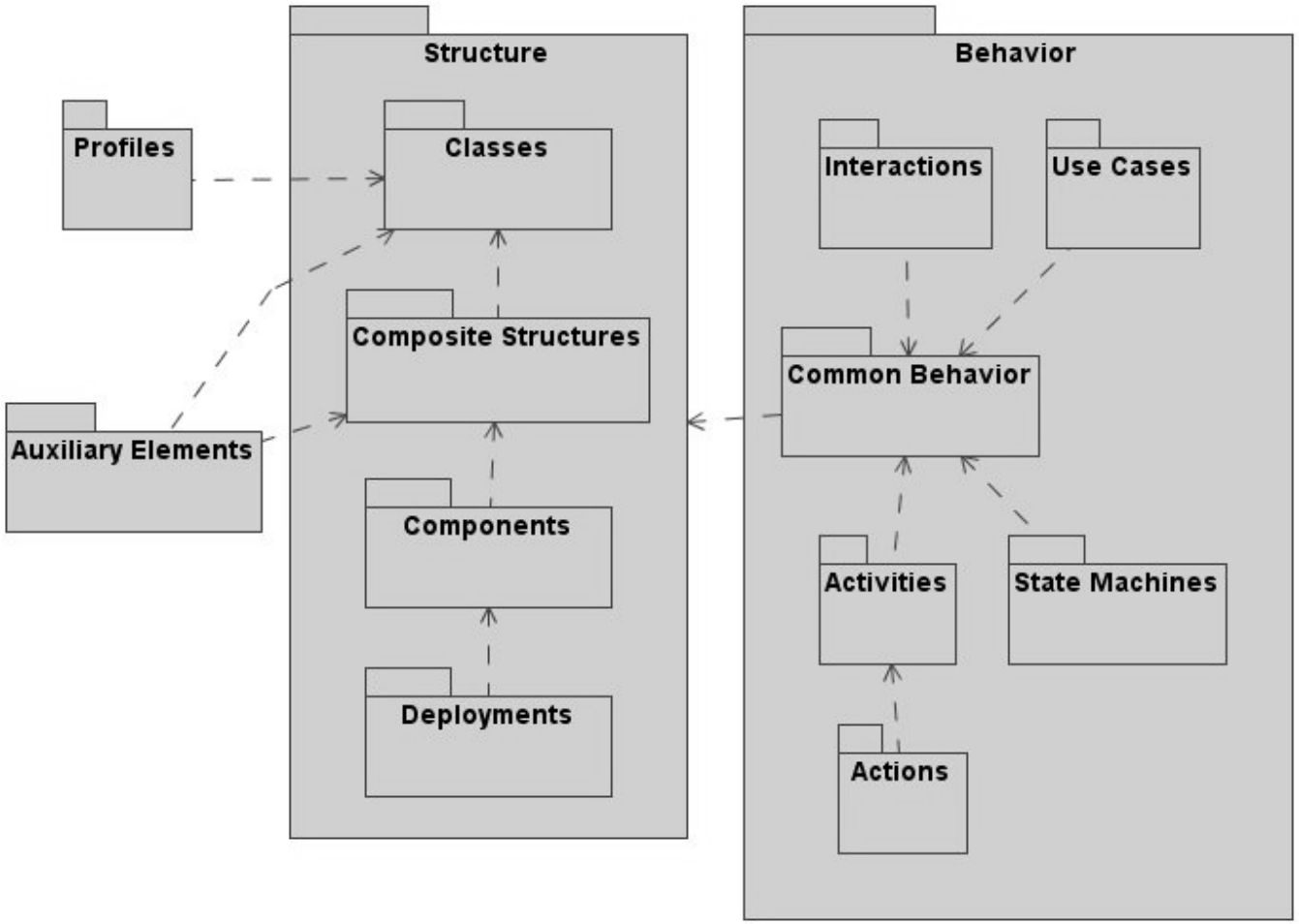
Profily slouží pro přzpůsobení jádra pro konkrétní domény (C++, EJB, softwarové modely, reálný čas, ...). Balík Profiles obsahuje mechanismy, které umožňují uzpůsobit prvky metamodelu pro konkrétní modely (z úrovně M3 na M2).

Profily v UML

Příklad: SysML je rozšíření UML, které využívá jen některé diagramy UML a přidává nové diagramy, které v UML nejsou.

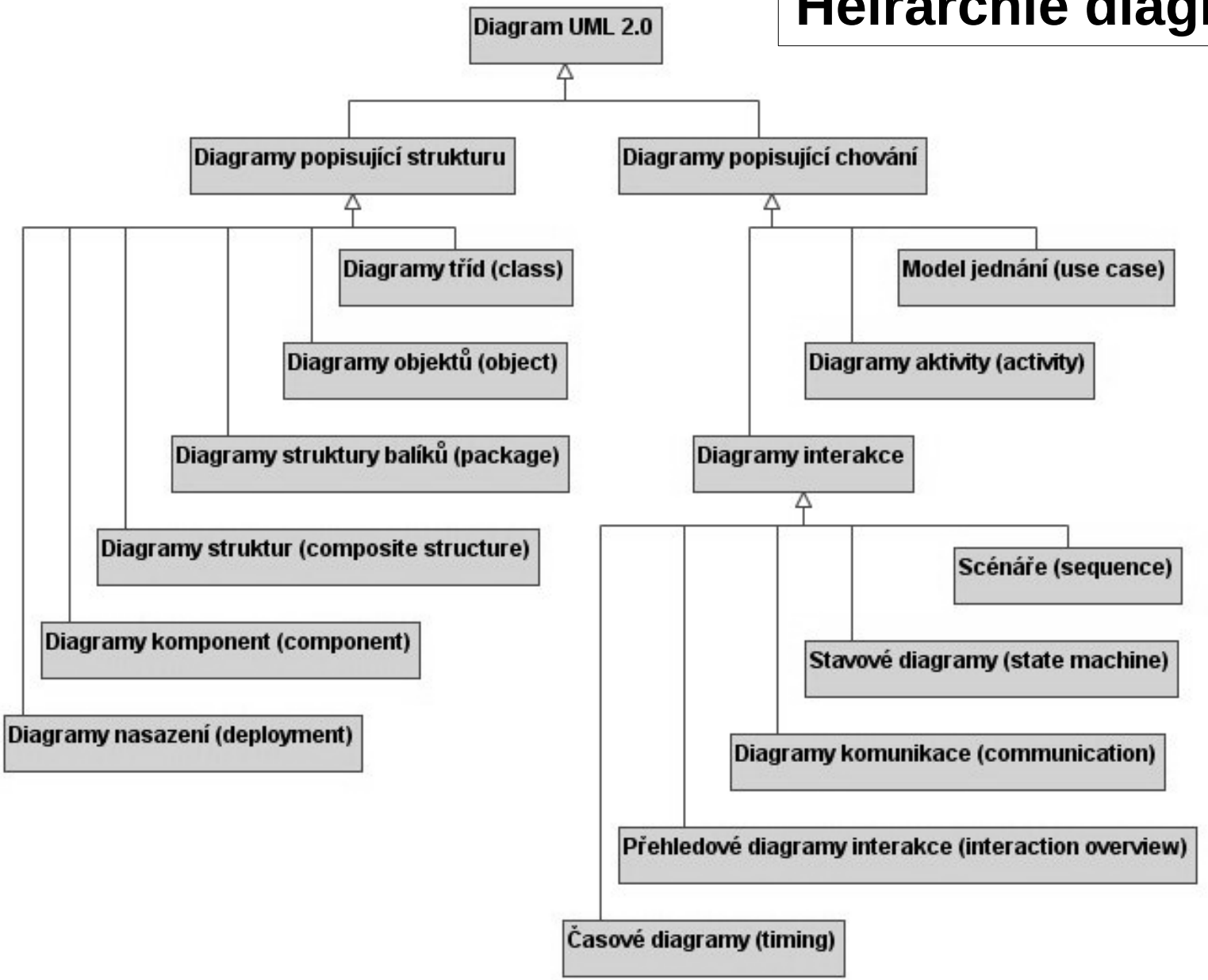

UML 2
SysML
Část SysML,
která není
v UML
Část UML 2,
která není
využívána v SysML
Společná
část

Superstruktura UML (2.0)



Diagramy verze 1.5 a 2.0

Heirarchie diagramů

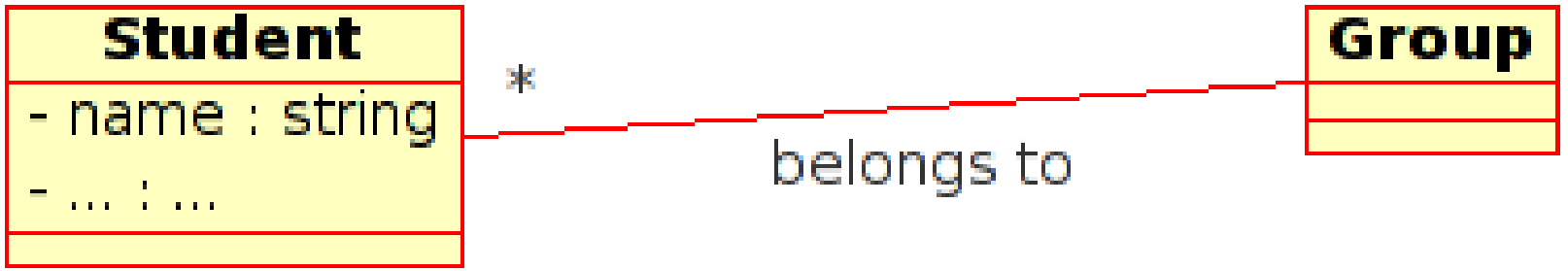


The End

Logický a fyzický model, MDA

Michal Píše
pisem1@fel.cvut.cz

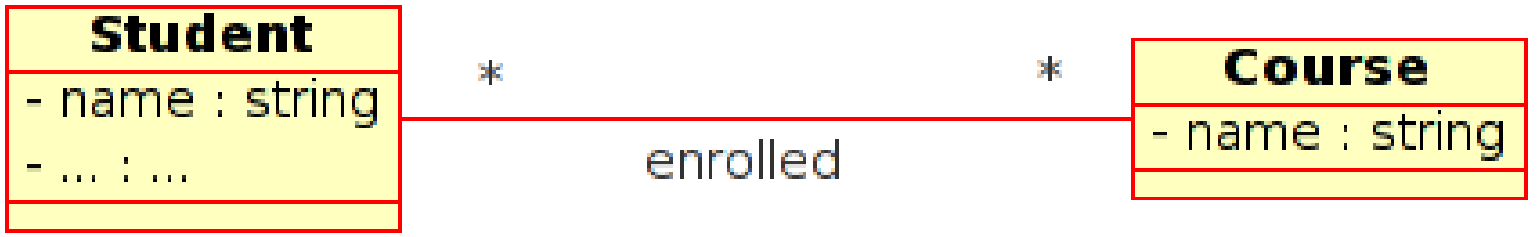
Jednoduchý model



Jednoduchý model - DB

- Tabulky student a group
- Každá tabulka má sloupce odpovídající jednotlivým atributům
- Tabulka student má navíc cizí klíč do tabulky group

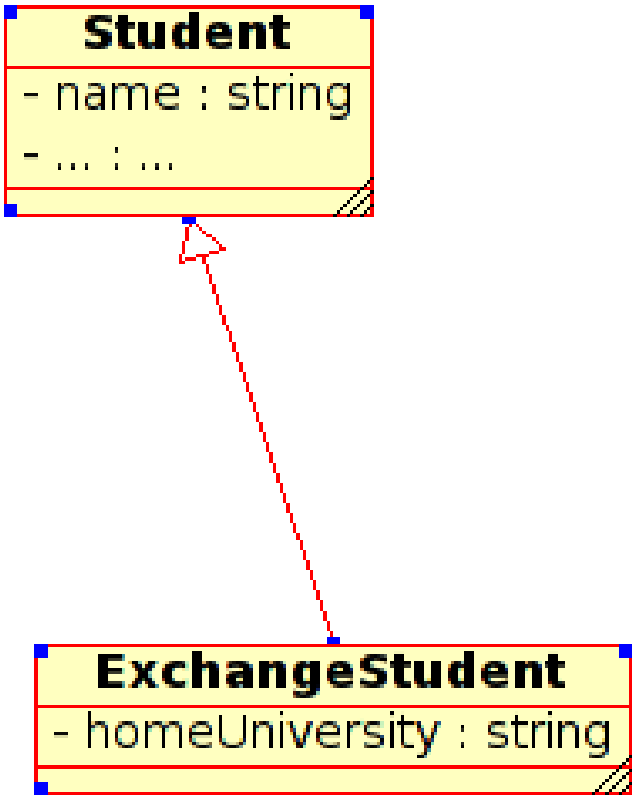
Lehce složitější model



Lehce složitější model - DB

- **Tři** tabulky: student, course a student_enrolled_course
- Tabulka student_enrolled_course na modelu není explicitně vidět, ale z multiplicity asociace enrolled se její existence dá odvodit
- Model stále obsahuje plnou informaci o jeho fyzické realizaci

Složitější model



Složitější model - DB

?

První možnost

- Dvě tabulky: student a foreign_student
- Každá tabulka má sloupce odpovídající atributům dané třídy
  - student má sloupec name
  - foreign_student má sloupce name a home_university
- Při hledání studenta (nebo studentů) se musíme podívat do obou tabulek

Druhá možnost

- Dvě tabulky: student a foreign_student
- Každá tabulka má sloupce odpovídající atributům přidaným v dané třídě
  - student má sloupec name
  - foreign_student má sloupec home_university
- Jedna instance třídy ForeignStudent je rozprostřená přes víc tabulek, takže při přístupu na přidané atributy musíme používat kartézský součin (join)

Další možnosti

- Různé kombinace předchozích řešení
- Zcela odlišné koncepce
- Zabývají se tím tvůrci objektově-orientovaných databází

Důsledek

- Původní model nenese celou informaci o implementaci
- Pokud do původního modelu informaci o implementaci zaneseme, model se tím znepřehlední
- V ideálním případě by model měl být přehledný a zároveň obsahovat plnou informaci
- Co s tím?

Řešení

- Dva modely, logický a fyzický
- Logický model je přehledný
- Fyzický model je odvozením (zpřesněním) fyzického a obsahuje plnou informaci
- Oba modely musí být synchronizovány!

Terminologie MDA

- Logický model se jmenuje PIM (Platform Independent Model)
- Fyzický model se jmenuje PSM (Platform Specific Model)
- Zpřesňování implementace může probíhat po krocích, takže fyzických modelů může být víc

Příklad zpřesňování

- Na úrovni PIM mám jenom třídy a asociace
- První zpřesnění: budu to psát v Javě
  - Takže vím, že mám třídy a interfacy
- Druhé zpřesnění: budu používat EJB
  - Entity Beans, Session Beans,...
- Třetí zpřesnění: budu používat Oracle a WebSphere
- ...

Výhody

- Různé úrovně abstrakce modelu
  - Analytik chce jinou úroveň abstrakce než programátor
- Zachování podstatné části investic
  - Při přechodu na jinou technologii se zahodí jen ty modely, které na dané technologii závisí
  - **Analýza se nemusí dělat znova!**

Výhody II

- Korespondence modelů
  - Specifičtější model může být poloautomaticky odvozen z toho abstraktnějšího
- Flexibilita
  - Do procesu odvození se dá kdykoli zasáhnout

Výhody III

- Generování kódu
  - Nejspecifičtější model nese tolik informace, že podstatná část kódu aplikace se z něj dá vygenerovat
- Snadná změna architektury aplikace
  - Změnou formy generovaného kódu změním architekturu celé aplikace

Pozor!

- Vývoj musí být “model-driven” tzn. mění se nejprve model, teprve potom se změny promítají do kódu
- Vývoj nemusí nutně probíhat vodopádově – i modely se dají vyrábět iterativně
- Diagram tříd popisuje jenom statickou strukturu aplikace, vývoj řízený modelem se týká i ostatních diagramů

Problémy

- Aplikace se neskládá jenom z “databázových” tříd, potřebovali bychom i třídy “paměťové” - jak je v modelu rozlišit?
  - Poznámkou?
- Java nemá jenom třídy, má i interfacy – jak je v modelu rozlišit?
  - Zase poznámkou?

Problémy II

- Java “umí” jednoduchou dědičnost tříd a vícenásobnou dědičnost interfaců – jak to v modelu ohlídat?
  - Dávat si na to pozor?
- Java “neumí” násobné atributy
  - Stačí je v UML nepoužívat?
- V databázi nemá smysl mluvit o modifikátorech přístupu – sloupce nejsou private, protected nebo public – co s tím?
  - Ignorovat?

Problémy III

- U “databázových” tříd bych potřeboval evidovat dvě jména – jméno v kódu a jméno v databázi – jak to do modelu dopsat?
  - Definovat mechanismus odvození jednoho jména z druhého?
- U tříd UI bych potřeboval definovat např. události, které v dané třídě mohou vznikat a kam se pošlou, jak?
  - V poznámce?

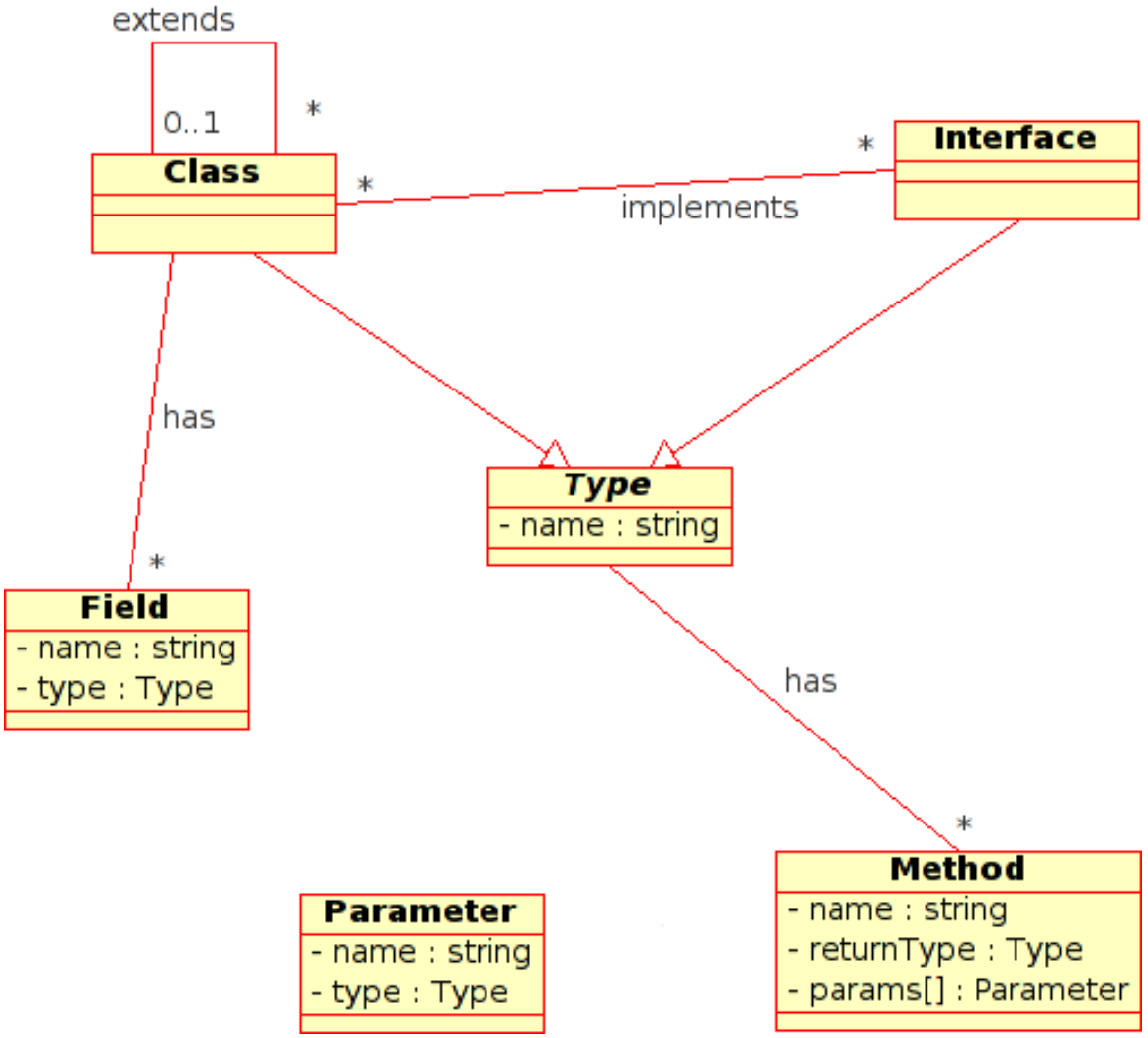
Problémy obecně

- UML někdy umí víc, než potřebujeme
- UML někdy umí méně, než potřebujeme
- UML by někdy mělo kontrolovat nějaká omezení

Řešení

- Popíšeme strukturu pojmů, se kterými pracujeme stejně, jako popisujeme strukturu objektů v aplikaci
  - Popíšeme, co je to v Javě třída, interface, atribut,...
  - Popíšeme, co je to tabulka, sloupec,...

Java


extends
0..1
*
Class
*
*
Interface
implements
has
*
Field
- name : string
- type : Type
Type
- name : string
has
*
Method
- name : string
- returnType : Type
- params[] : Parameter
Parameter
- name : string
- type : Type

Databáze

Zkuste si doma

Vrstvy návrhu

- Data – v návrhu se neobjevují
- Metadata = model – to, co se vytváří během analýzy a návrhu
- Metametadata = metamodel – definice pojmů, které se používají v modelu
- Metametamodel = definice pojmů, které se používají při definici pojmů, které se používají v modelu
- …

Jak to vyřeší naše problémy?

- Každý model bude navrhován v nějakém metamodelu
  - PIM v obecném
  - PSM např. v metamodelu Javy a J2EE
- UML nástroje se před tvorbou modelu zeptají na metamodel a nabídnou nám jenom to, co metamodel definuje
- MDA nástroje nám umožní popsat způsob transformace modelů

Terminologie MDA

- MOF – Metaobject Facility
  - Standard, který definuje metametamodel, tzn. pomocí kterého se vytvářejí metamodely
- UML Profile for Java
  - Zjednodušeně řečeno metamodel Javy

Terminologie MDA II

- XMI – XML Metadata Interchange
  - Formát založený na XML, ve kterém se ukládají modely a metamodely
- QVT – Queries/Views/Transformations
  - Jazyk umožňující popsat transformace modelů

Dotazy

?

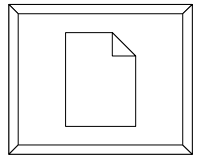
Analýza (pokračování)

Návaznosti

- Minule:
  - Architektura, modelem řízený vývoj
- Dnes:
  - Analýza (pokračování)
- Příště:
  - Návrh

Aktuální stav

- Všechny týmy by měly mít vypracován model jednání (příp. i kontext), detailní harmonogram práce s požadovanými zdroji a rozpočet – pozor harmonogramy jsou 2 – pro Vás a pro Vaše následníky.
- To vše by se mělo zobrazit na stránkách projektů a platí:
  - **Co není na service.felk.cvut.cz neexistuje!**
  - **Co není včas, neboduje se!**



Pár poznámek:

- Datový slovník popisuje pouze pojmy ze specifikované oblasti, nikoli např. co to je „CASE“.
- Datový slovník popisuje především strukturu pojmů, nikoli jejich význam – něco lze k termínům připojit formou poznámky.
- Př.

  ZÁKAZNÍK = @ID+příjmení+jméno+…

  evidovaný zákazník

Model jednání a kontext

- Model jednání (use case model) slouží pro evidenci aktérů a služeb systému.
- Kontextový diagram slouží pro evidenci aktérů a datových toků.
- Oba modely se tedy doplňují, ale představují pouze prvý krok popisu, který musí být doplněn podrobnějším popisem služeb a dat.
- Každý případ použití zastupuje sadu **aktivit**, které aktér se systémem provádí – sadu **scénářů**, jak komunikace se systémem probíhá.

Analýza

Měla by odpovědět na otázku ***CO***?

- Musí proto definovat ***konceptuální model*** řešeného systému (***PIM – Platform Independent Model***)
- Musí definovat představu, s jakými ***daty*** bude systém pracovat, jaké ***služby*** bude systém poskytovat a jak se bude chování systému měnit - jaká bude ***dynamika*** systému
- Musí stanovit podmínky, za jakých je analytická dokumentace ***akceptovatelná***

Funkční model

Funkčně orientovaná analýza

- Začínáme seznamem funkcí - modelem jednání
- Pro každý případ užití navrhneme scénář jednání (původce, událost, akce, participanti, výstupy - reakce), diagramy aktivit, příp. diagramy datových toků (návaznosti funkcí)
- Popis akcí (minispecifikace základních akcí)
- Identifikace objektů
- Identifikace vztahů mezi objekty
- Modelování životních cyklů objektů

Model jednání ECO-skladu

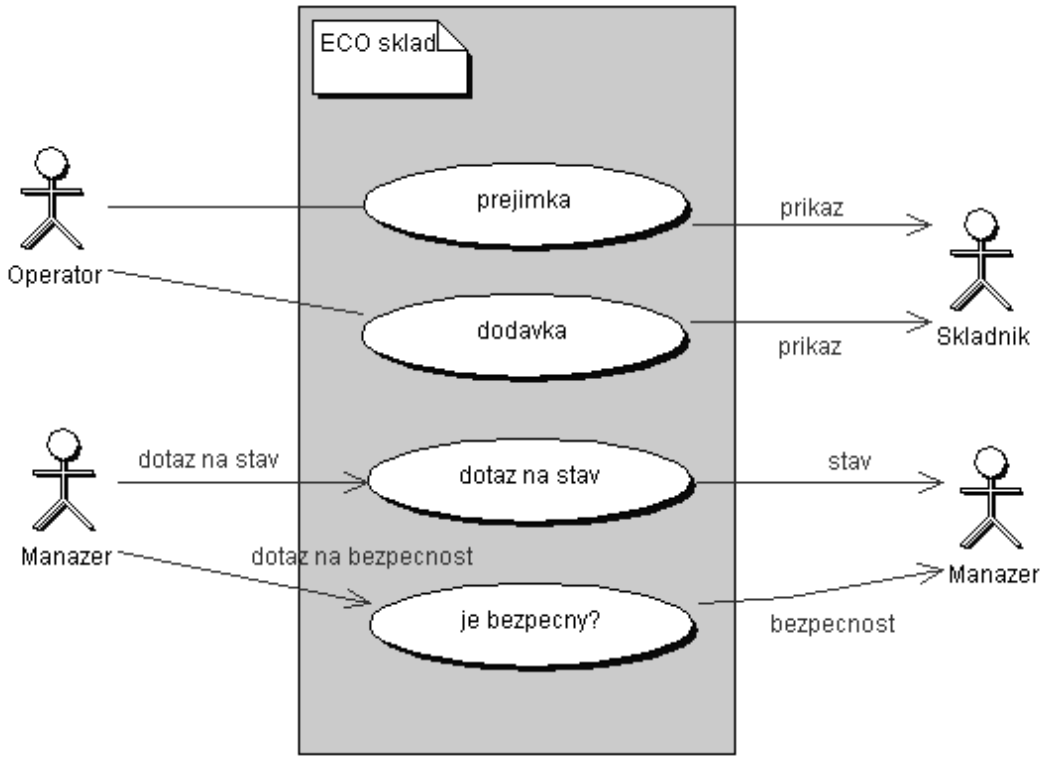
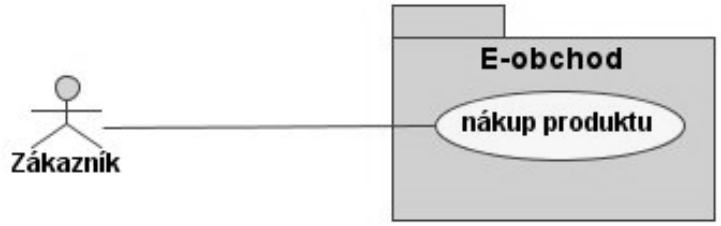


Diagram případů užití je pouhá evidence služeb, ty musí být popsány přesněji. Striktně řečeno - model jednání obsahuje diagram případů užití a jejich popis.

Jak lze služby evidované v modelu jednání popsat?

- Textovým popisem (to je podmínka nutná, nikoli postačující).
- Minispecifikací (strukturovaným popisem operace)
- Dekompozicí na služby jednodušší
  - pomocí scénáře
  - pomocí diagramu datových toků (DFD)
  - pomocí diagramu aktivity
  - pomocí diagramu komunikace
  - pomocí stavového diagramu



1. Zákazník prohlíží katalog a vybere si zboží k nákupu
2. Zákazník zvolí nákup
3. Zákazník vyplní dodací informace (adresa, expresní nebo standardní dodávka)
4. Systém zobrazí plnou cenu včetně ceny dodání
5. Zákazník vyplní platební informace (číslo kreditní karty)
6. Systém autorizuje platbu
7. Systém potvrdí prodej
8. Systém zašle potvrzovací e-mail zákazníkovi

Alternativy:

3a. Uživatel je pravidelným zákazníkem

- 3a1. Systém zobrazí naposled zapamatované dodací a platební informace
- 3a2. Uživatel může potvrdit, nebo změnit zobrazené informace a scénář pokračuje v kroku 6

6a. Systému se nepovedlo autorizovat platbu

- 6a1. Zákazník může opravit platební informace, nebo zrušit nákup

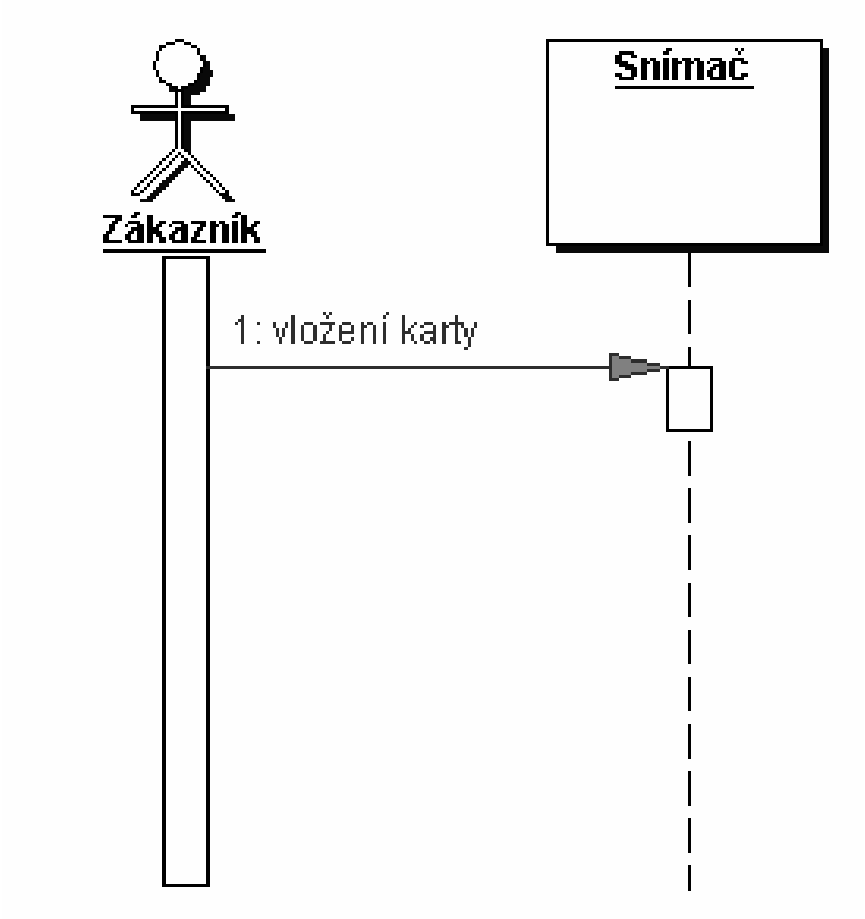
Scénáře (Sequence diagrams)

(zachycení sledu událostí)

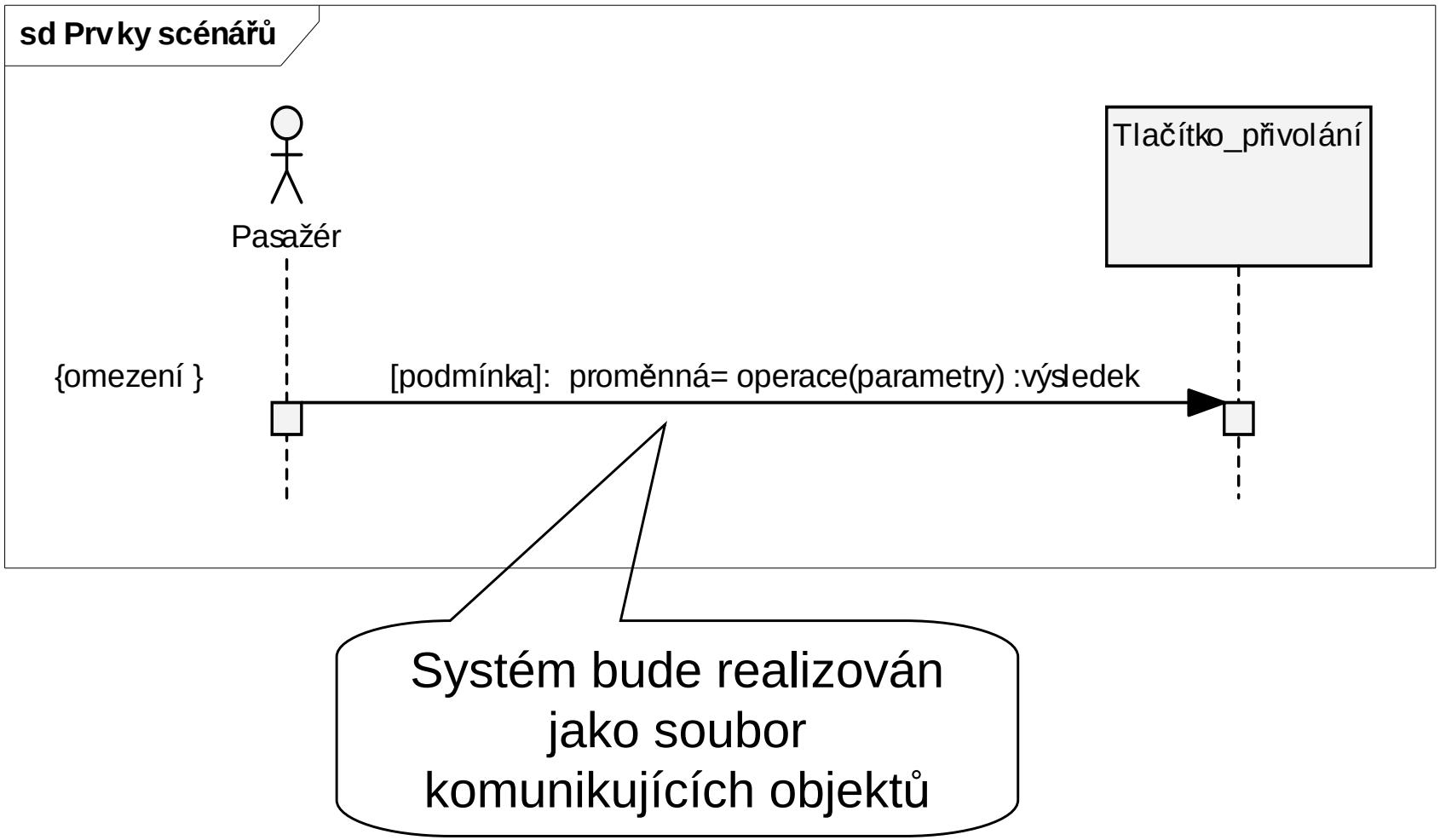
Prvky:

- **Objekty** - znázorněné obvykle jako sloupce
- **Interakce mezi objekty** (stimuly) - orientované šipky mezi objekty
- **Události** - události, které vyvolaly interakci
- **Reakce** - odezvy na události (výstupy)
- **Časová osa** - pro vyznačení sledu událostí

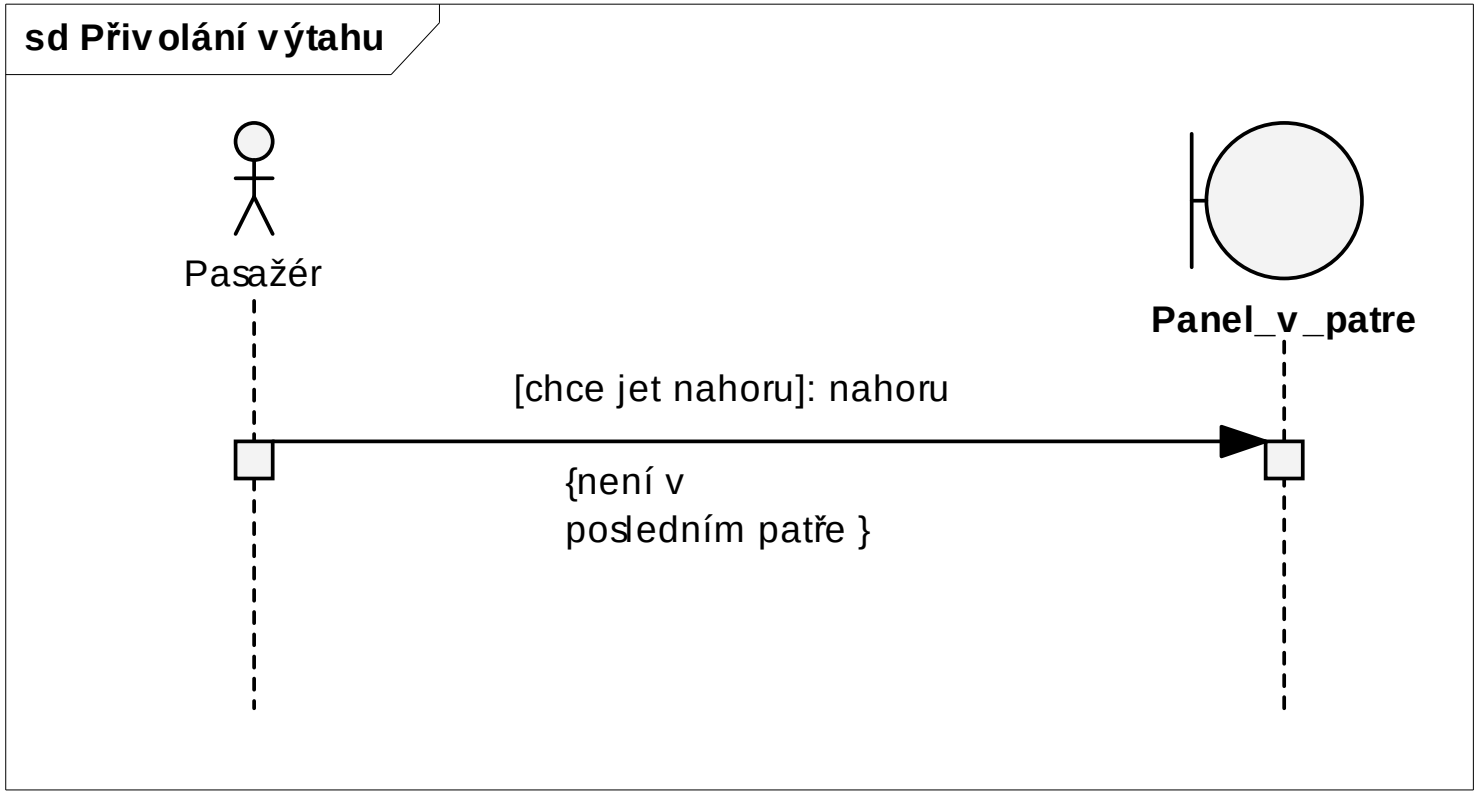
Zákazník se „autentizuje“



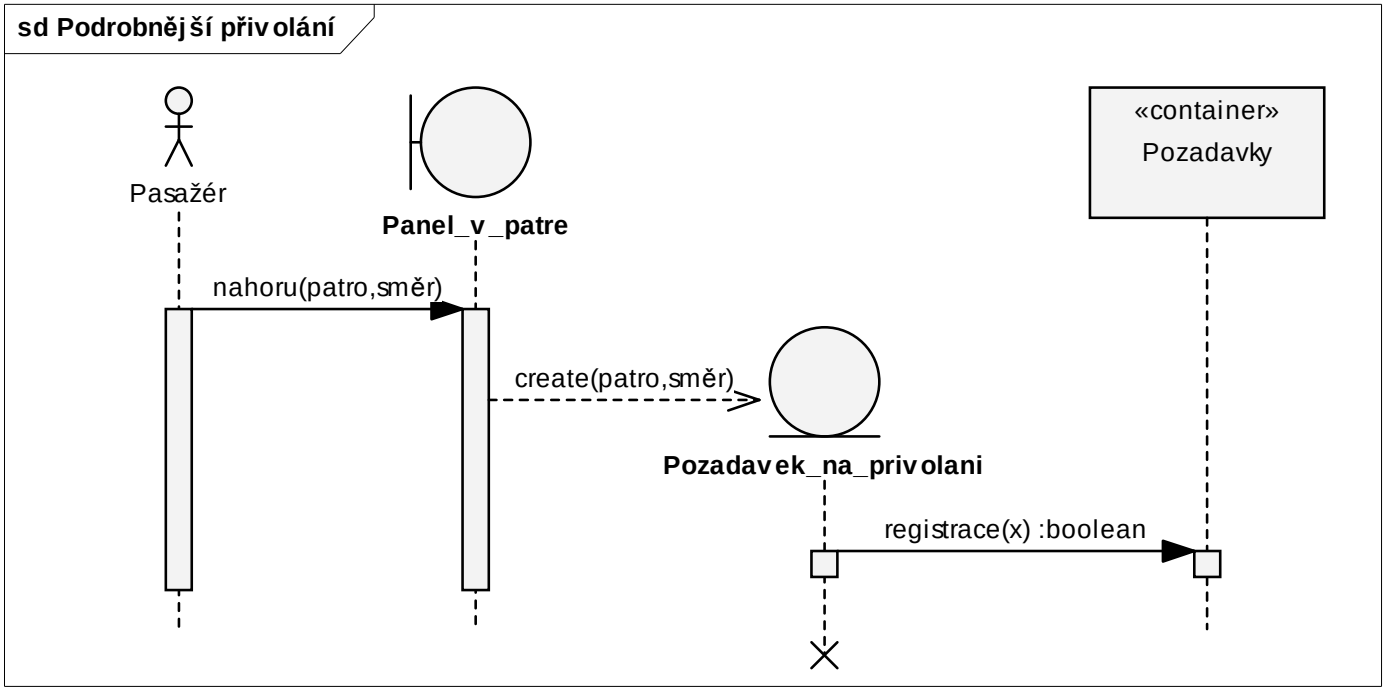
Základní princip scénáře



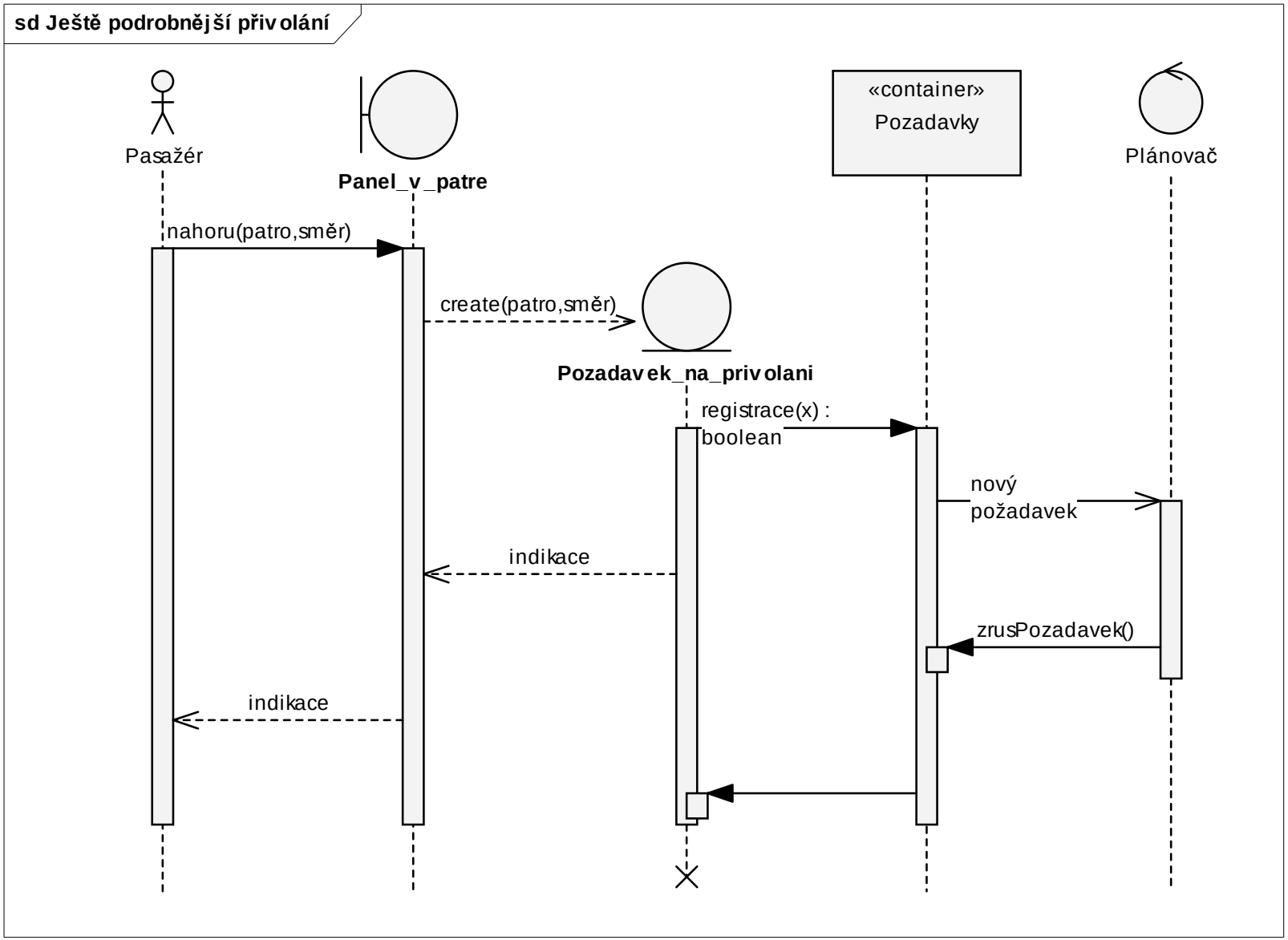
V konceptu nejprve pouze volíme metodu



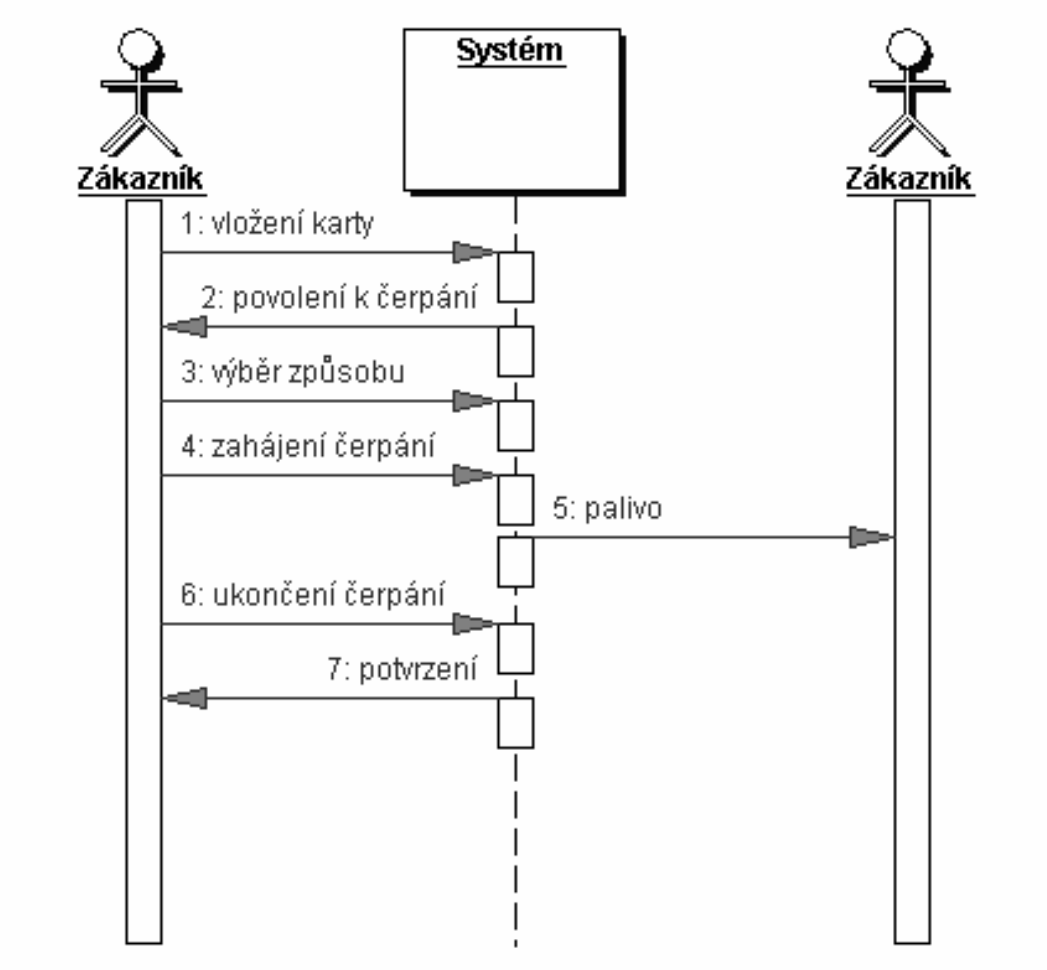
Později můžeme popsat činnost podrobněji, využít konstrukce a destrukce



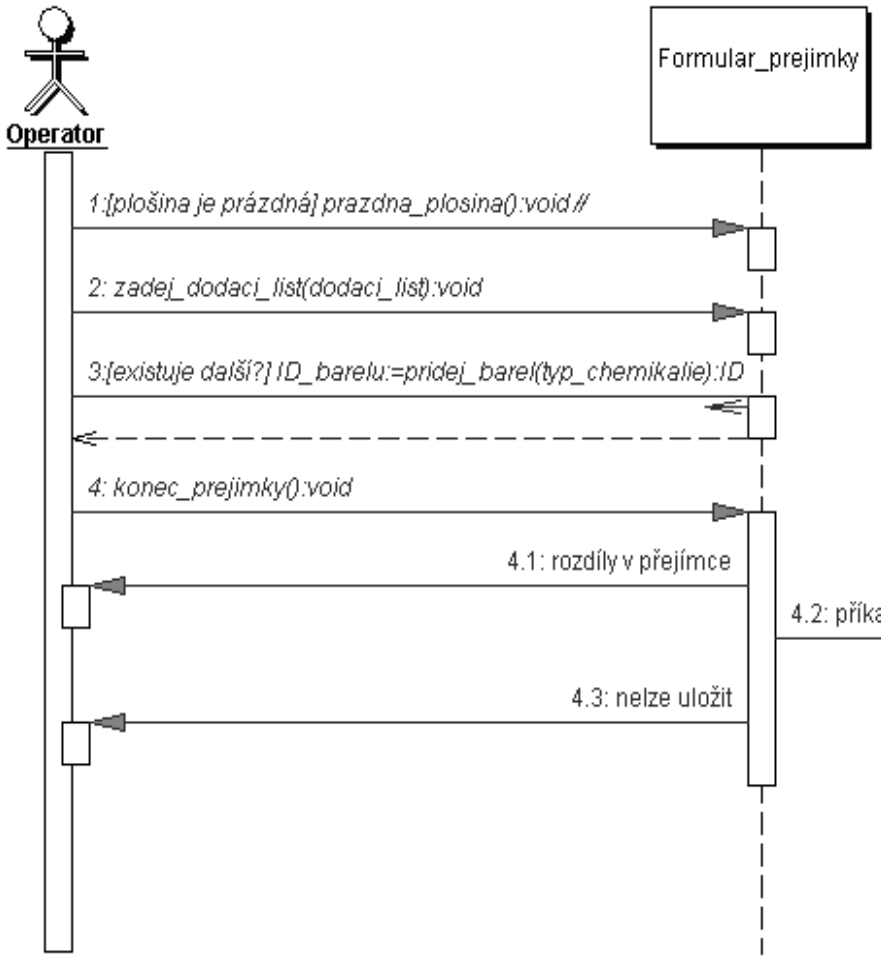
Reakce a návratové hodnoty



Hrubý scénář pro „čerpání“



Scénář pro “přejímku”



Scénář je pouze dekompozice, jeho složky musí být popsány přesněji – scénář, dekompozice, minispecifikace

Popis akce (operace, funkce)

Operation: název

Description: textový popis

Reads: jaká data jen čte

Changes: jaká data mění nebo vytváří

Sends: jaké reakce vyvolává (jaké zprávy posílá)

Assumes: co předpokládá

Results: co zajišťuje (zaručuje)

Popis pro “prázdná plošina”

Operation: prázdná plošina

Description: informuje systém, že nakládací plošina je prázdná

Reads:

Changes: plošina

Sends:

Assumes:

Results:

- vyprázdní v modelu nakládací **plošinu**
- uvolní identifikátory barelů, které jsou na **plošině**

Popis pro “dodací list”

Operation: dodací list

Description: zahájí přejímku a uloží informace z dodacího listu

Reads: *supplied* dodací_list

Changes: zadaný_dodací_list

Sends:

Assumes:

Results:

- vnitřní objekt **zadaný_dodací_list** je inicializován hodnotami z fyzického **dodacího_listu**

Popis pro “barel k zařazení”

Operation: barel k zařazení

Description: každý vyložený barel je jednoznačně identifikován

Reads: *supplied* typ_chemikálie

Changes: plošina, *new* b: Barel

Sends: operátor:{ID barelu}

Assumes:

Results:

- nakládací plošina obsahuje barel b
- operátor dostane identifikaci **ID barelu**
- atribut **b.typ** je nastaven na **typ_chemikálie**
- atribut **b.ID** je nastaven na identifikaci **ID barelu**

Popis pro “konec přejímky”

Operation: konec přejímky

Description: informuje systém, že již byly vyloženy všechny barely

Reads: plošina, zadaný_dodací_list

Changes: budovy ve skladu

Sends: operátor:{rozdíly v přejímce, nelze uložit}, skladník:{příkaz pro skladníka}

Assumes:

- **sklad** je bezpečný

Popis pro “konec přejímky” (pokr.)

Results:

- pro všechny barely, které lze do **skladu** umístit, přesune v modelu jejich umístění do vhodné **budovy** a vytvoří **príkaz pro skladníka(kam: alokační seznam)**
- pokud existují rozdíly mezi **zadaným_dodacím_listem** a skutečnou dodávkou, vytvoří se **rozdíly v přejímce(navíc, chybí: seznam barelů)**
- pro všechny barely, které nelze do skladu umístit vytvoří **nelze uložit(co: seznam barelů)**
- **sklad** je bezpečný

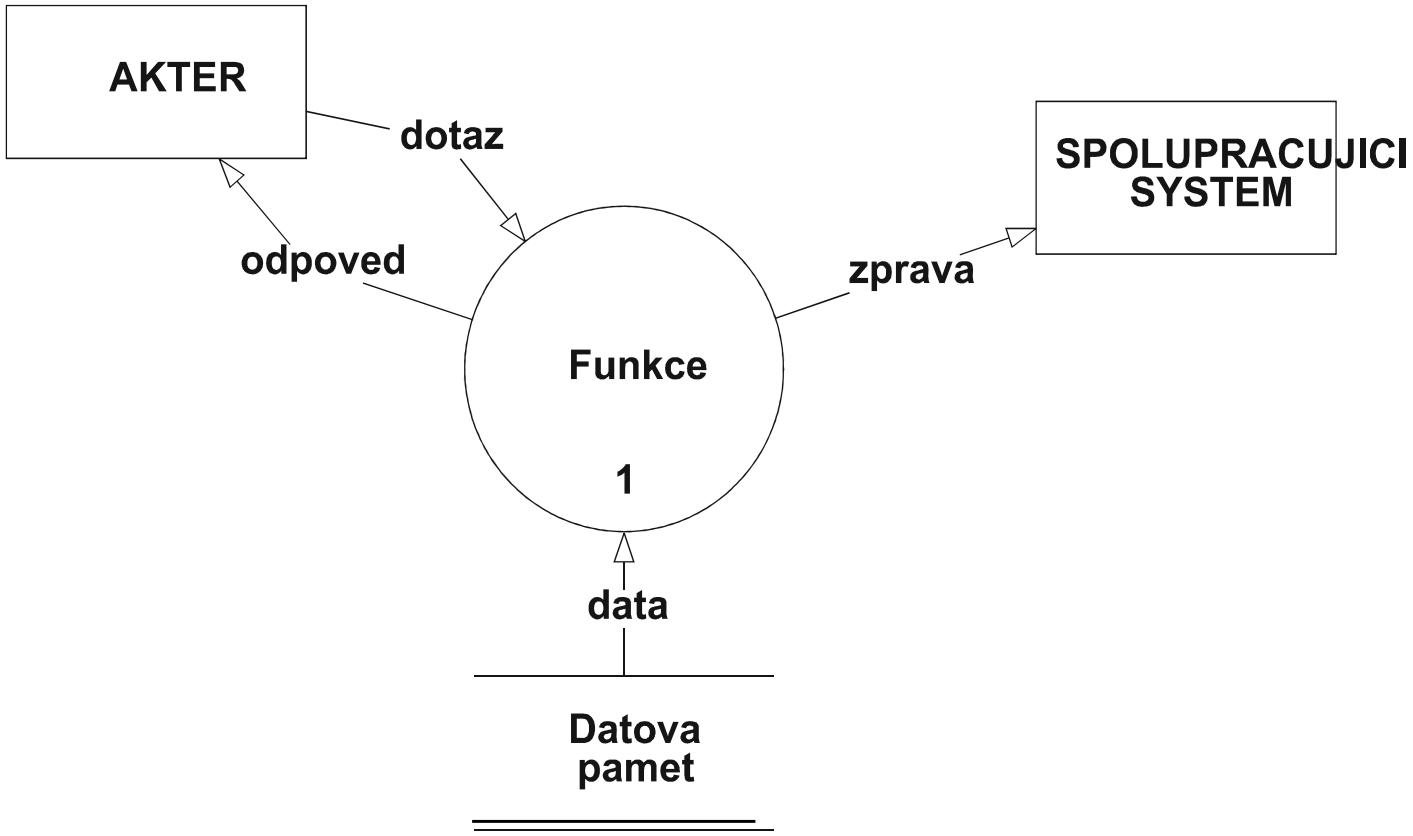
Diagramy datových toků (DFD – Data Flow Diagrams)

(zachycení vazeb funkcí a toků dat, dokumentace dekompozice)

Komponenty:

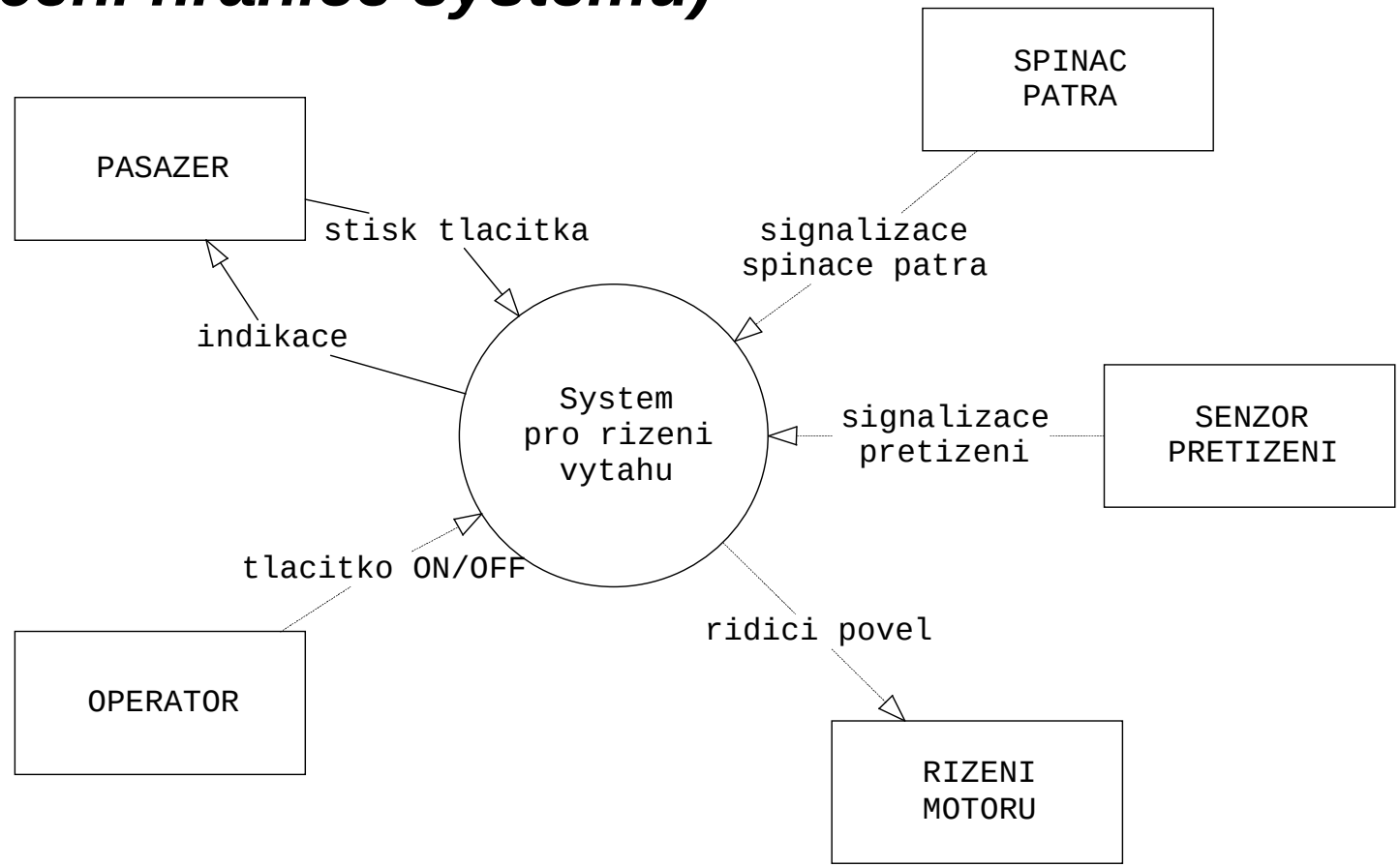
- funkce (procesy, akce)
- datové toky (data flows) - *orientované hrany vyznačující toky dat*
- datové paměti (data stores) - *místa, kde si potřebujeme něco pamatovat*
- aktéři (terminátory) - *uživatelské role nebo spolupracující systémy*

Notace DFD (Yourdon)

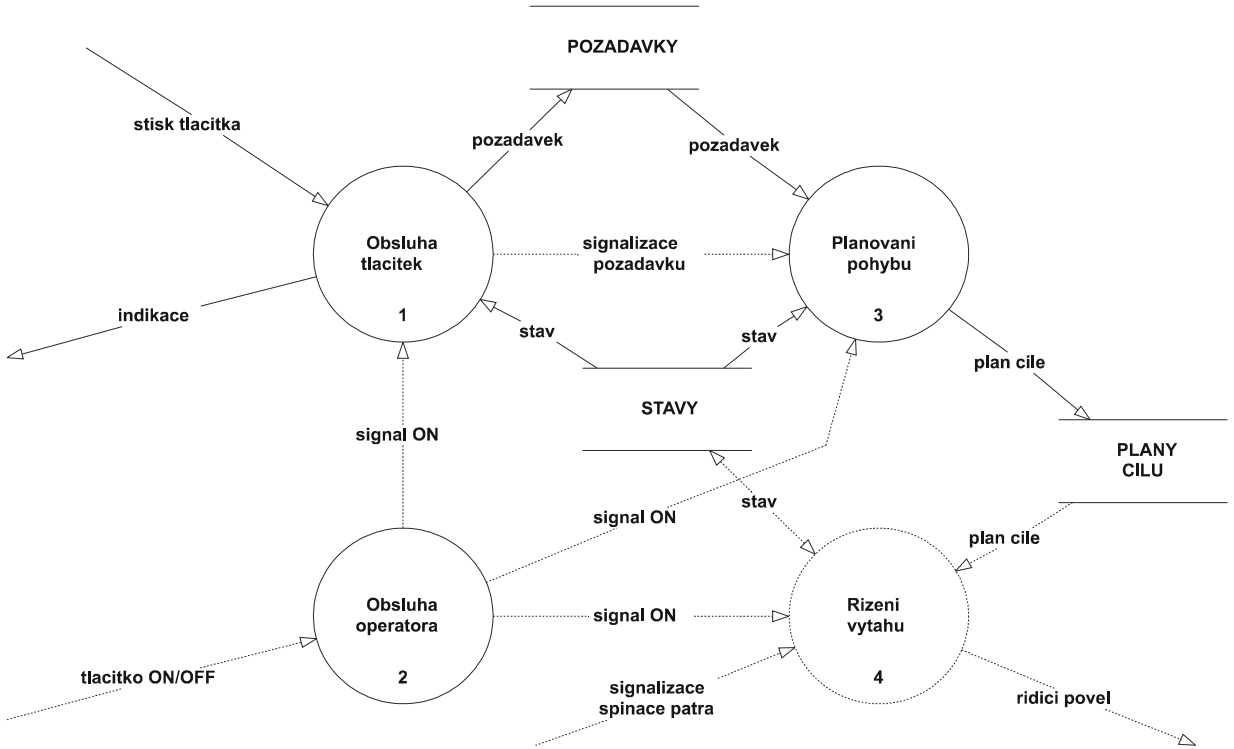


Kontextový diagram pro “Výtah”

(určení hranice systému)



DFD pro výtah (úroveň 0)



Diagramy aktivit

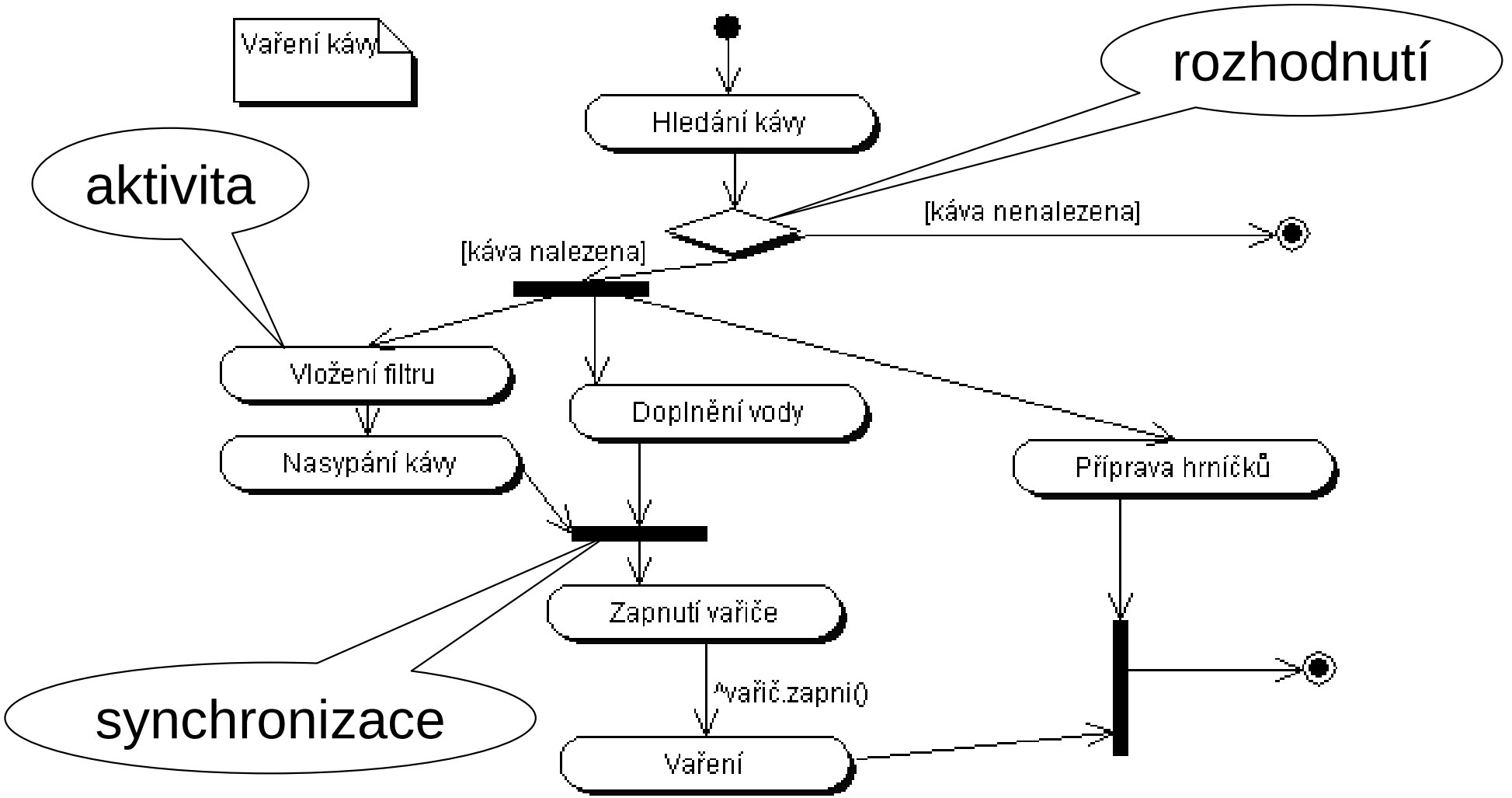
- Diagramy popisující aktivity (procesy) a jejich návaznosti.
- Přechody mezi aktivitami jsou vyvolány dokončením akce (jsou synchronní).
- Používají se pro dokumentaci tříd, metod, nebo případů použití (jako „workflow“).
- Obecně se mohou použít pro procesní modelování.
- Nahrazují v UML neexistující diagramy datových toků.

Diagramy aktivit (Activity diagrams)

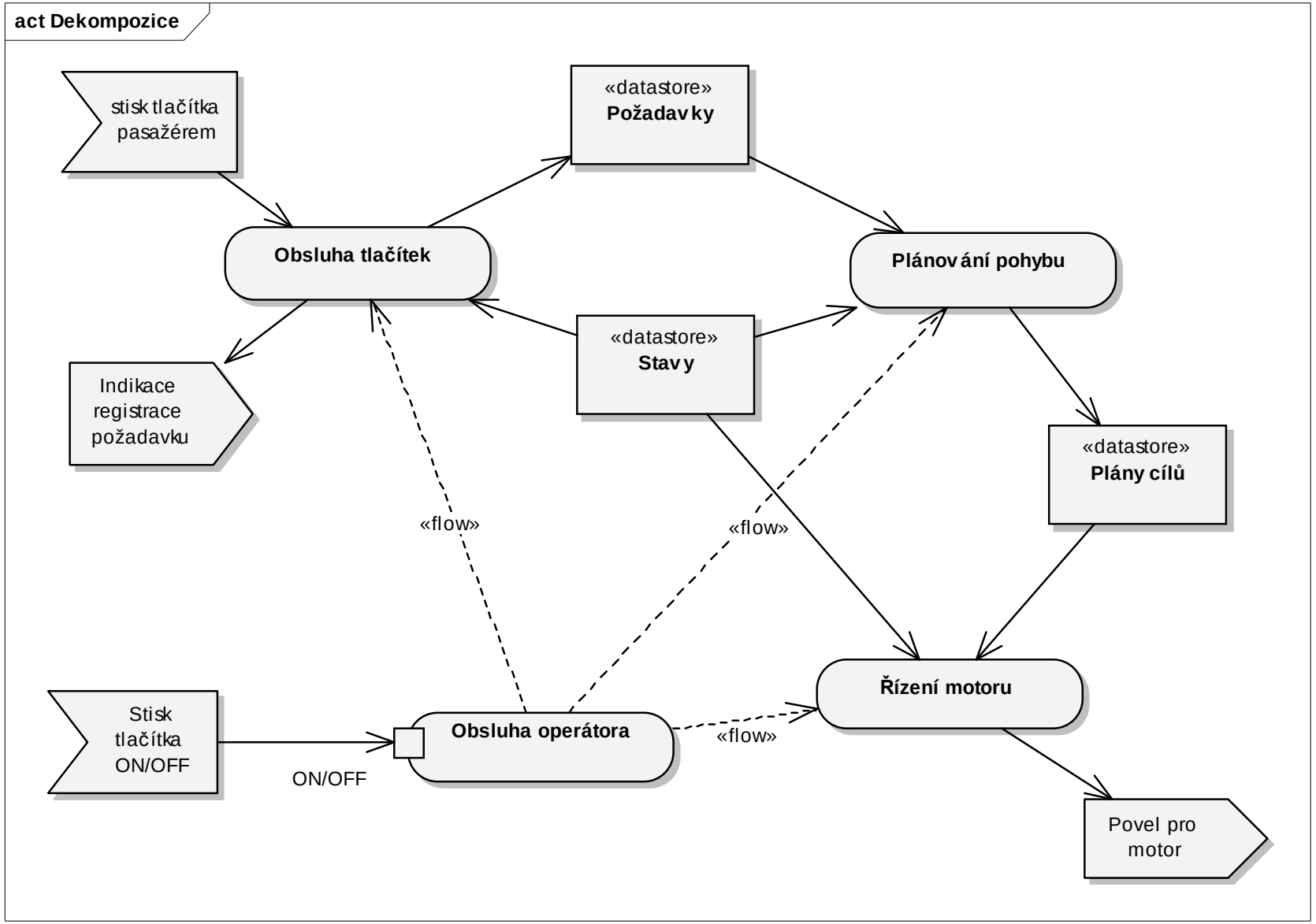
Prvky:

- **Aktivity** – činnosti, které modelujeme
- **Přechody** – po ukončení činnosti se přejde k činnosti jiné
- **Objekty** – s činností může souviset vytváření nebo konzumace objektů
- **Začátek**, **Konec**
- **Synchronizační značky** (rozvětvení a synchronizace)
- **Plavecké dráhy** – okruhy zodpovědností

Příklad diagramu aktivity



DFD v UML – diagram aktivity



Možný model vývoje, tj. rozklad produkce software na kroky a definice potřebných artefaktů

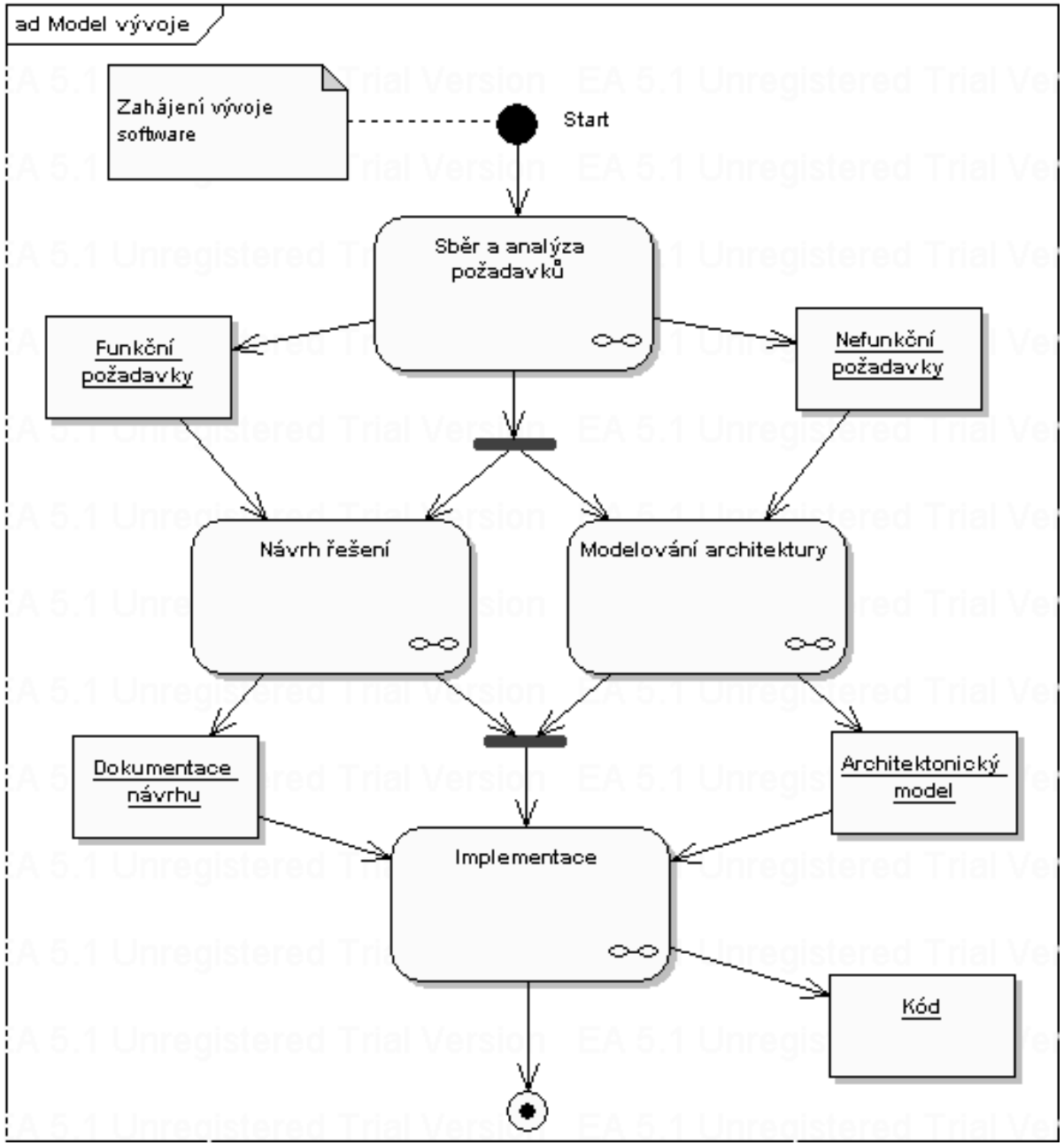


Diagram aktivity pro „přivolání výtahu“

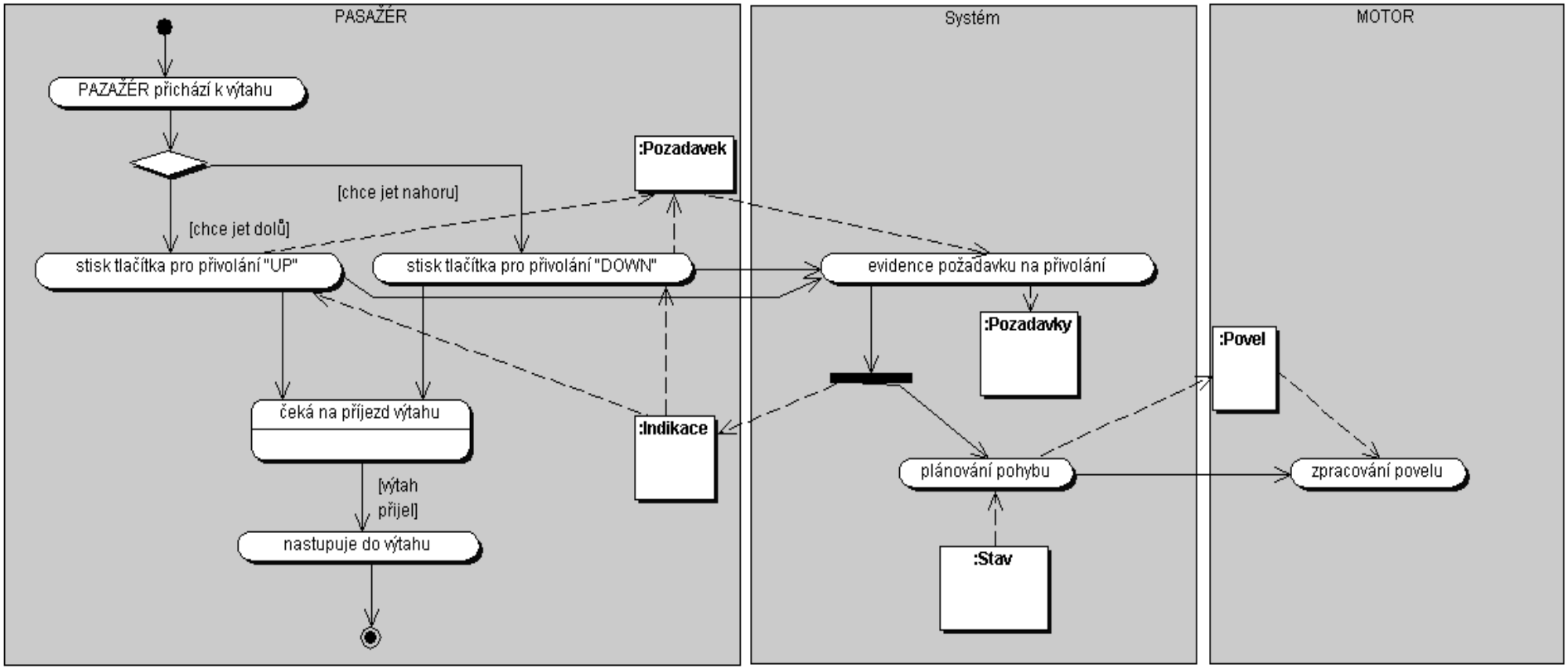
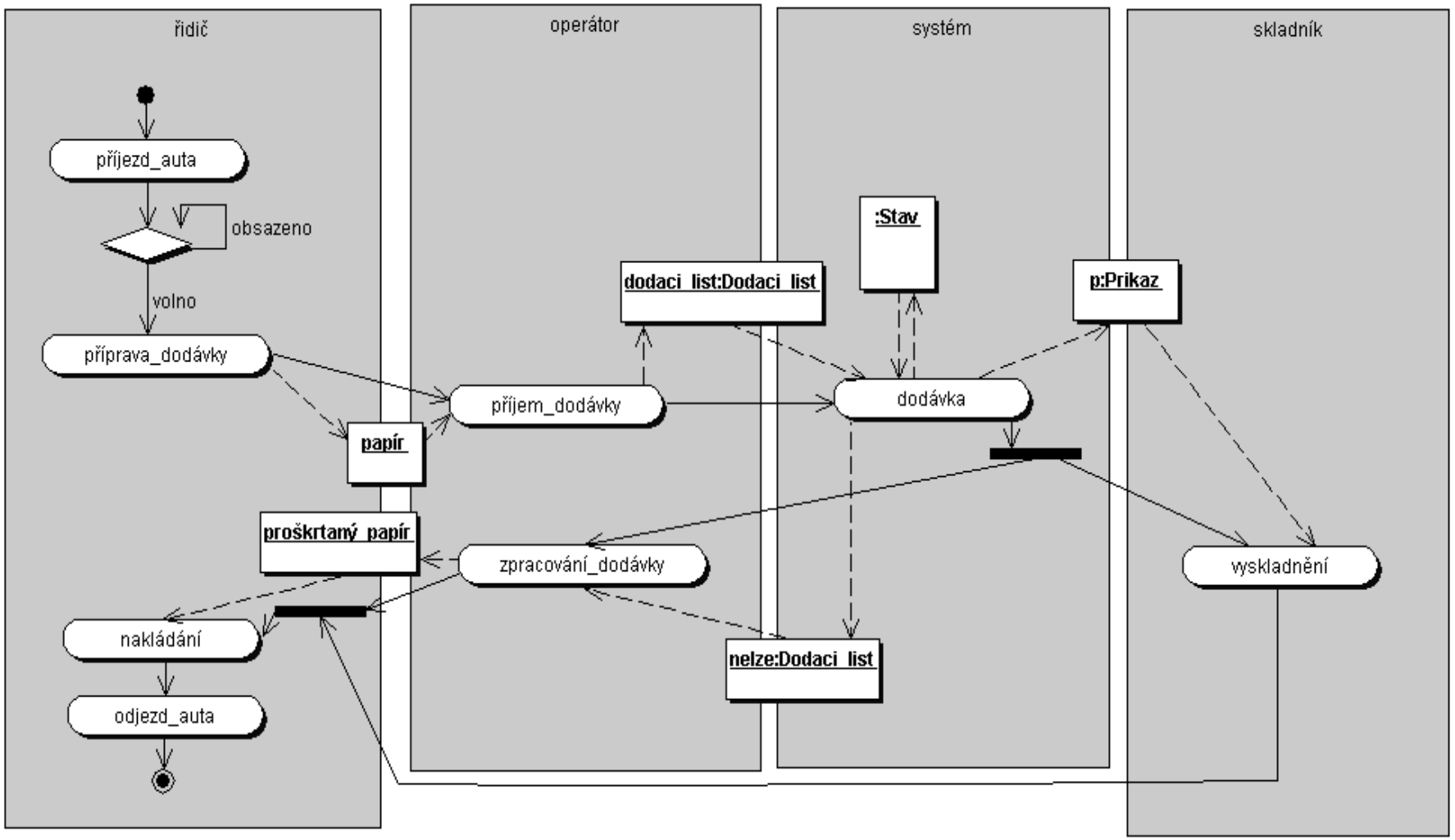


Diagram aktivity pro „dodávku“



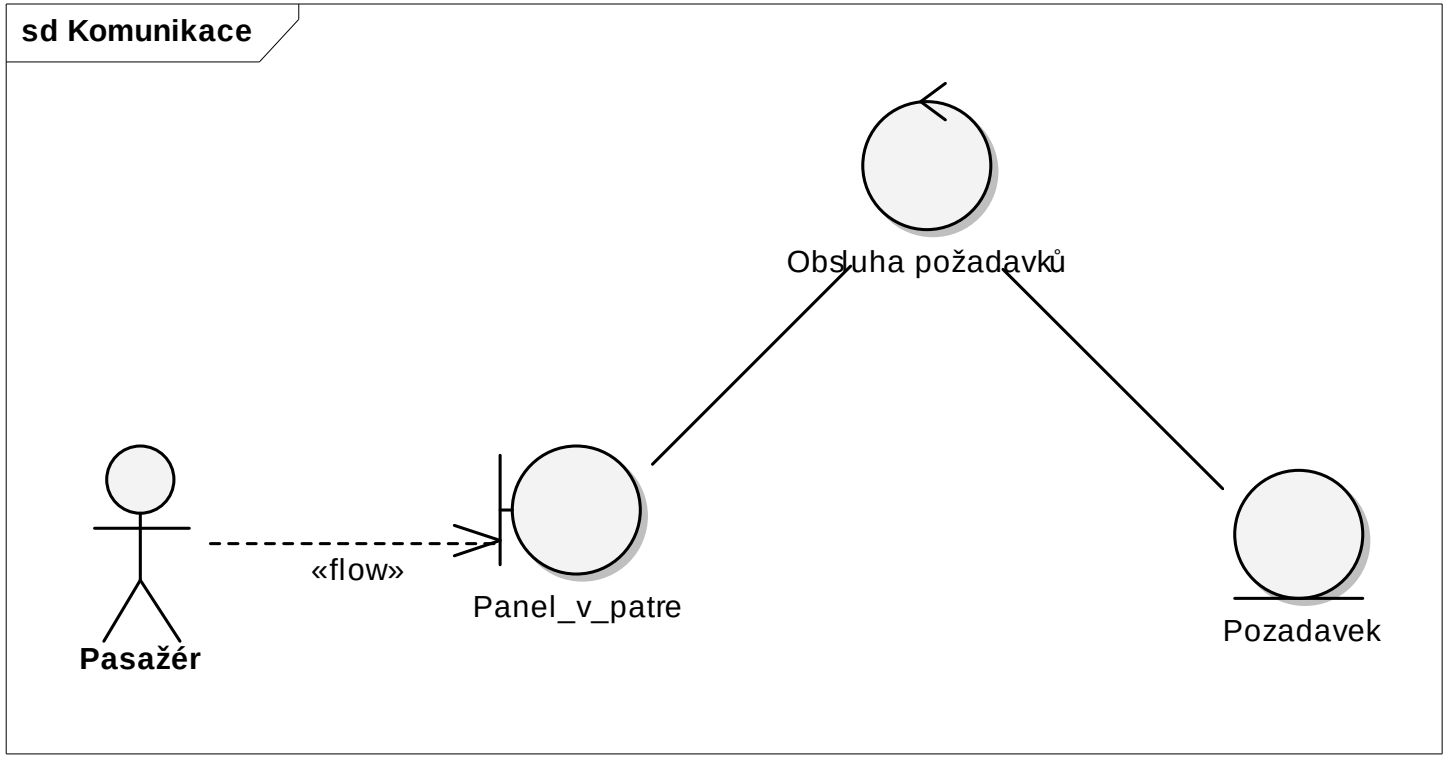
Diagramy komunikace (interakce)

(zachycení komunikace mezi objekty)

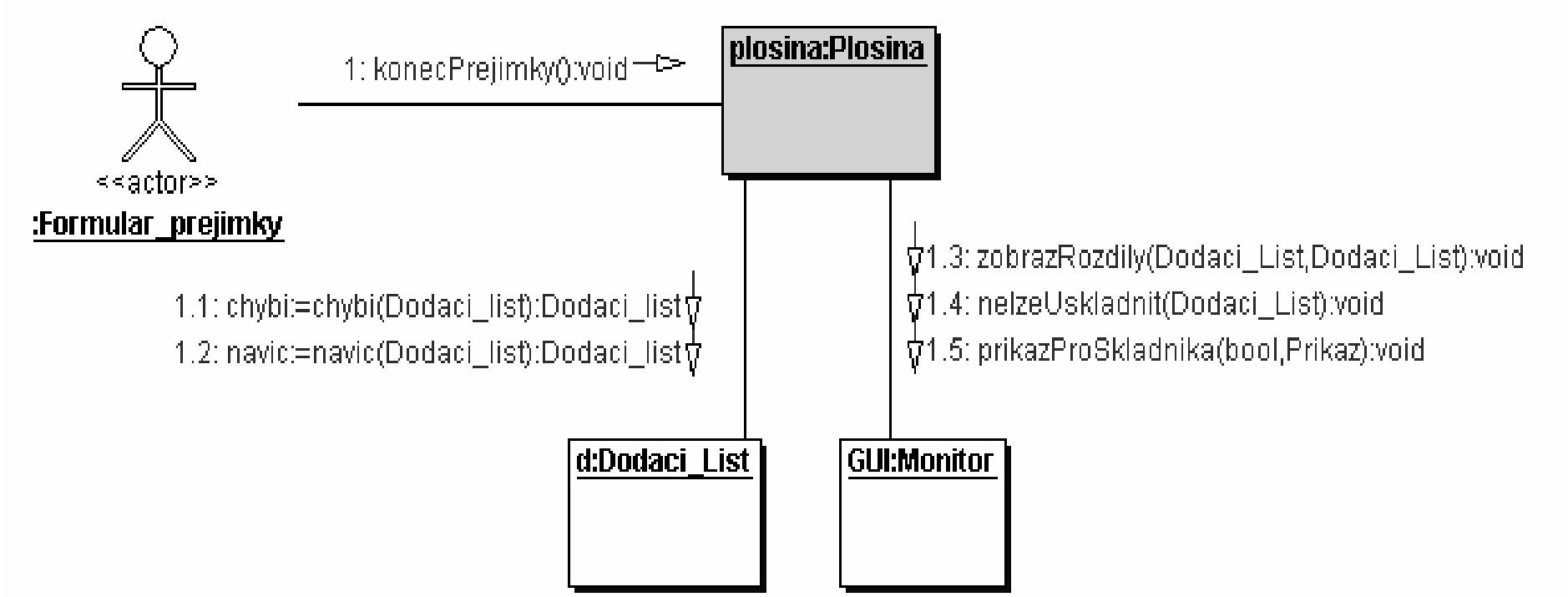
Prvky:

- **Objekty** - znázorněné jako obdélníky
- **Interakce mezi objekty** (stimuly) - orientovaná spojení mezi objekty
- **Zprávy** – dokumentace zpráv, které si objekty mezi sebou posílají, včetně parametrů a návratových hodnot - odezvy na události (výstupy)
- **Čas** – vyznačen číslováním

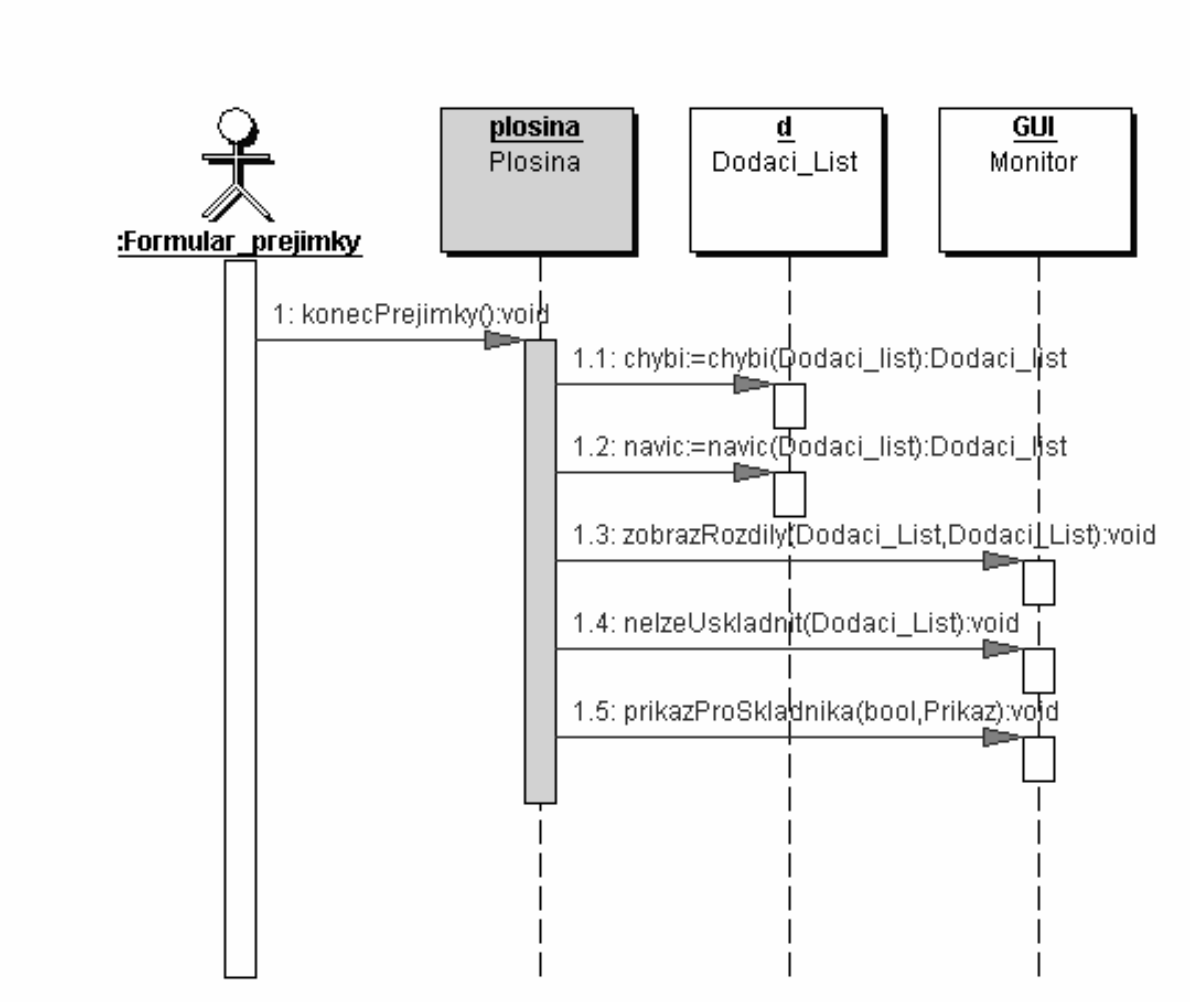
Triviálně


sd Komunikace
Obsluha požadavků
«flow»
Panel_v_patre
Pozadavek
Pasažér

Podrobněji

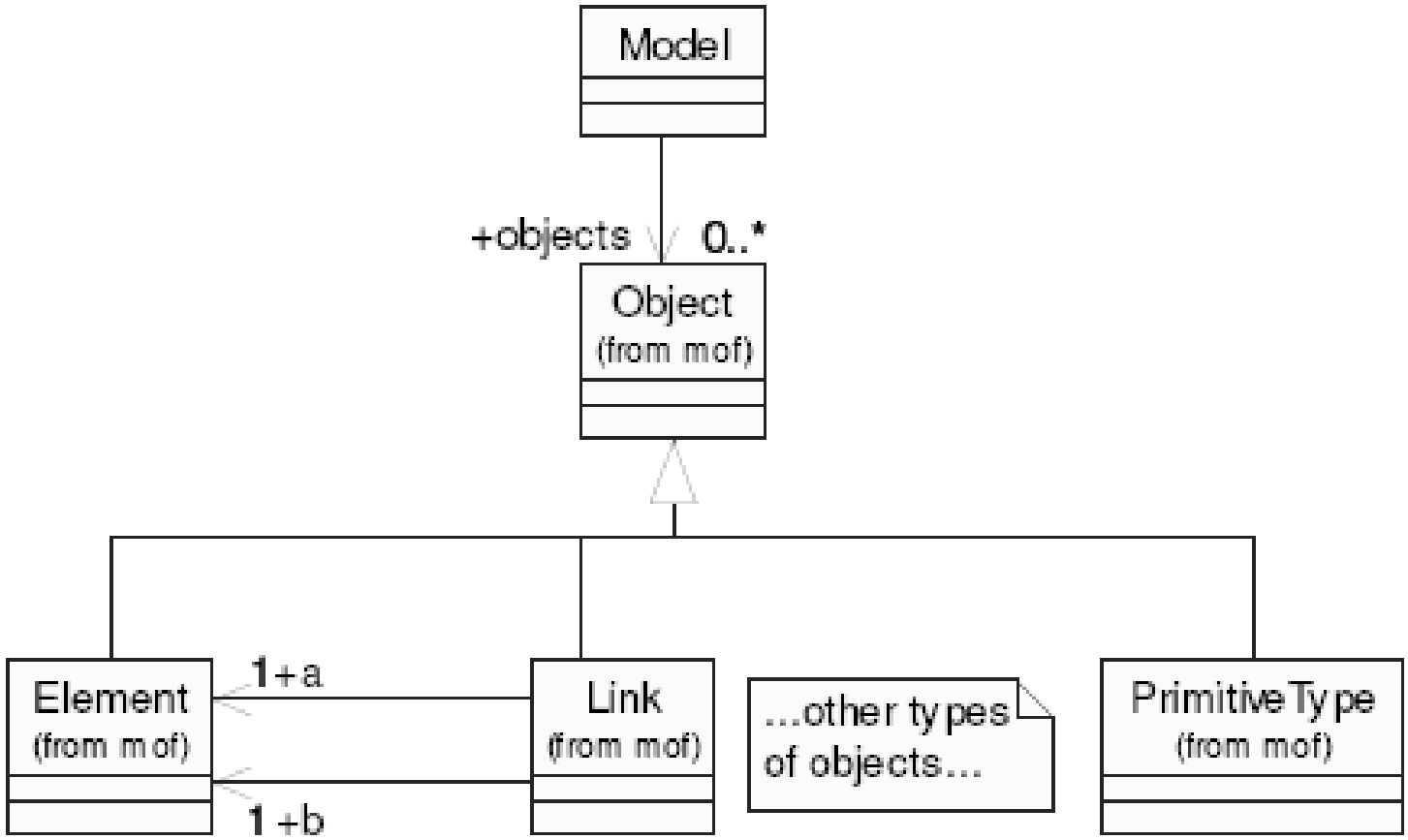

<<actor>>
:Formular_prejimky
1: konecPrejimky():void
plosina:Plosina
1.1: chybi:=chybi(Dodaci_list):Dodaci_list
1.2: navic:=navic(Dodaci_list):Dodaci_list
1.3: zobrazRozdily(Dodaci_List,Dodaci_List):void
1.4: nelzeUskladnit(Dodaci_List):void
1.5: prikazProSkladnika(bool,Prikaz):void
d:Dodaci_List
GUI:Monitor

Nebo jako scénář



Datový model

Př.: Model podle OMG MDA



Zdroj:http://www.omg.org/docs/ormsc/05-04-01.pdf

Datově orientovaná analýza

- Seznam událostí, kontext, datový slovník
- Identifikace dat, která s událostmi souvisí (identifikace základních objektů)
- Identifikace vztahů mezi objekty
- Scénáře jednání (původce, událost, akce, participanti, výstupy - reakce)
- Modelování životních cyklů objektů
- Popis akcí (minispecifikace základních akcí)

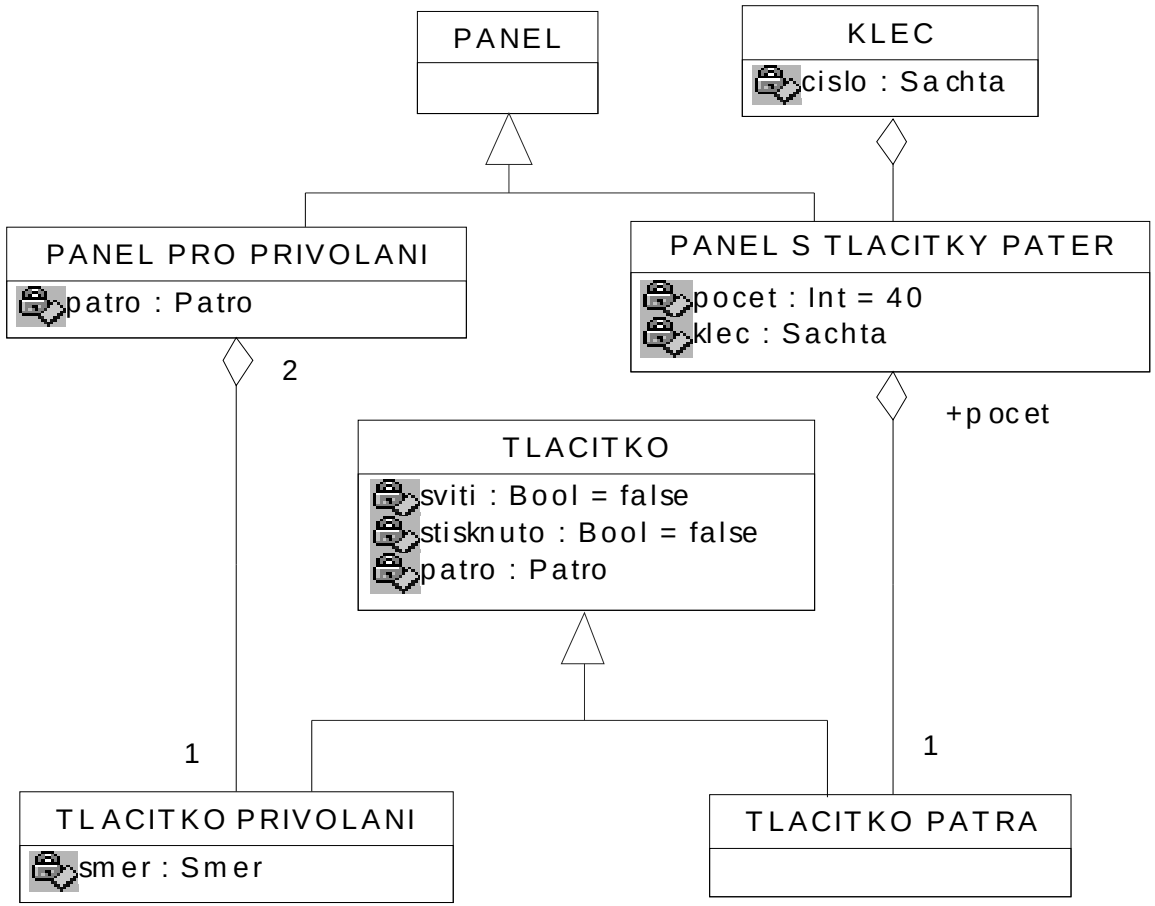
Datový model (konceptuální)

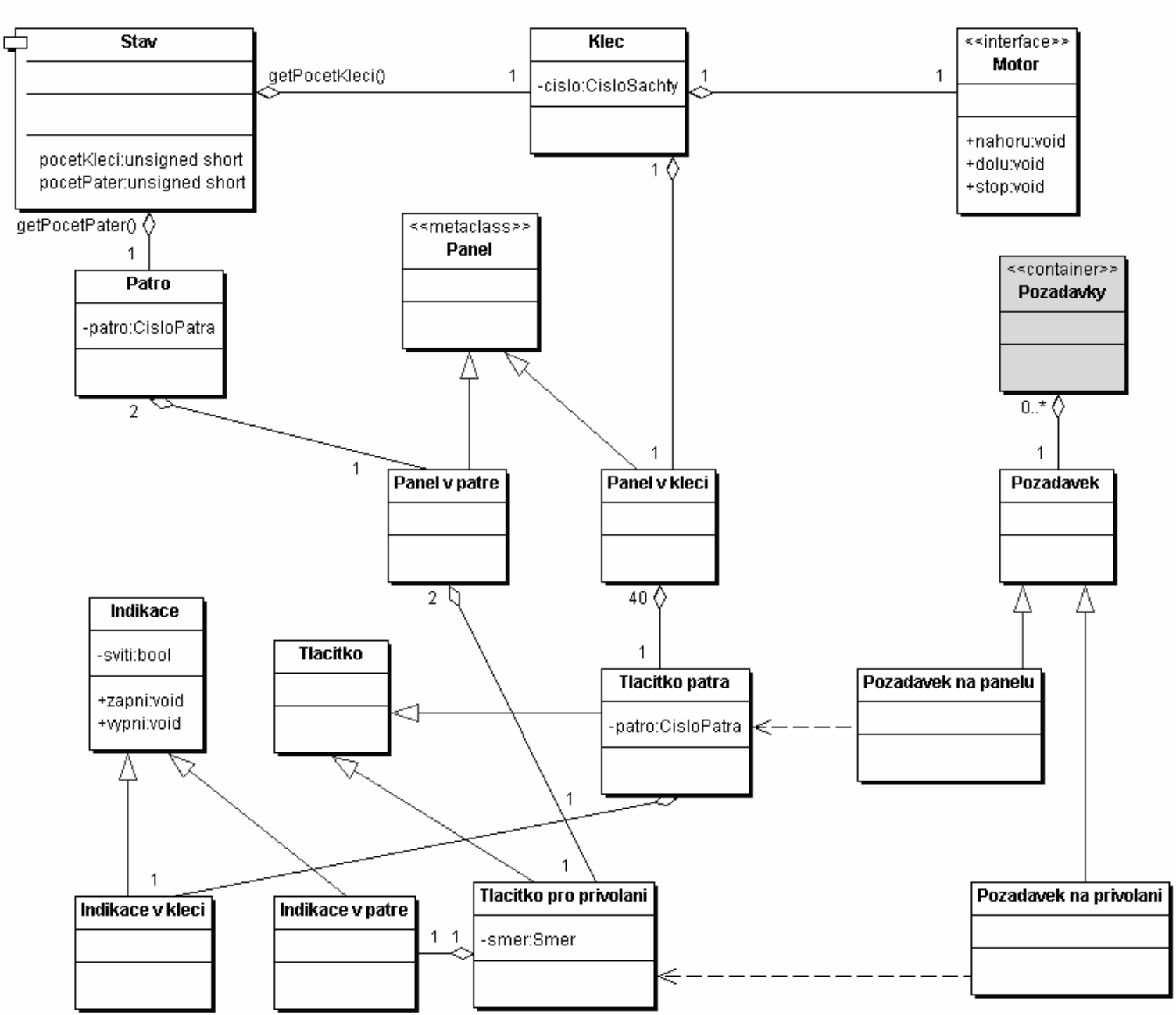
(zachycení analýzy dat)

Komponenty:

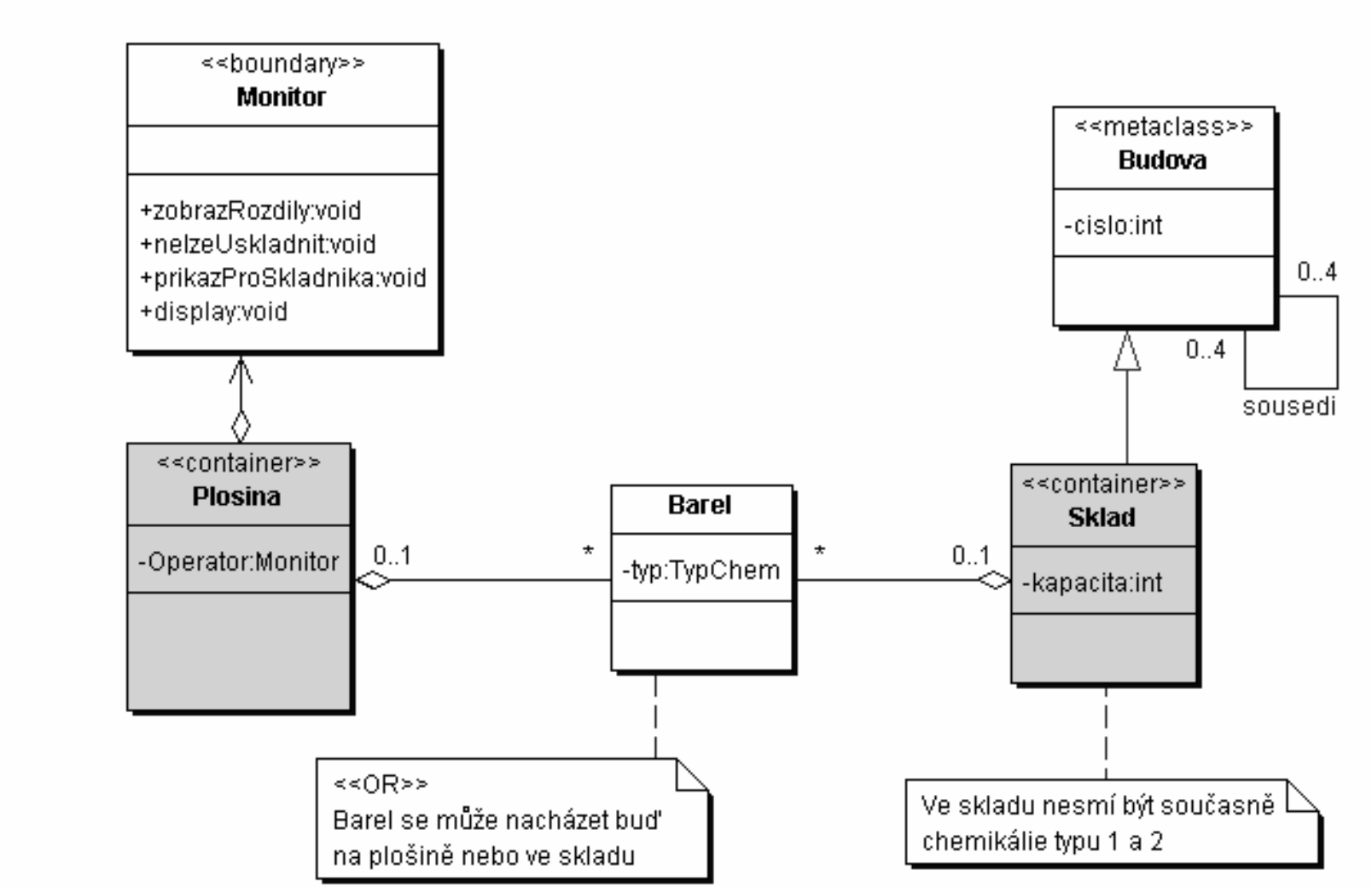
- **typy objektů (entity)** - entita = rozlišitelný identifikovatelný objekt
- **vztahy (relationships)** - množiny instancí reprezentujících vztahy mezi (2 a více) objekty
- **indikace přidružených objektů** - pro vztahy o nichž si potřebujeme něco pamatovat
- **indikace vztahů nadtyp-podtyp, celek-část** (gen-spec, whole-part) - vyjádření vztahu společný - speciální (dědičnost)

Datový model pro „Výtah“ (1.verze)




Stav
pocetKleci:unsigned short
pocetPater:unsigned short
getPocetKleci()
getPocetPater()
Klec
-cislo:CisloSachty
<<interface>>
Motor
+nahoru:void
+dolu:void
+stop:void
Patro
-patro:CisloPatra
<<metaclass>>
Panel
<<container>>
Pozadavky
Panel v patre
Panel v kleci
Pozadavek
Indikace
-sviti:bool
+zapni:void
+vypni:void
Tlacitko
Tlacitko patra
-patro:CisloPatra
Pozadavek na panelu
Indikace v kleci
Indikace v patre
Tlacitko pro privolani
-smer:Smer
Pozadavek na privolani

Datový model ECO (1.verze)



Datově orientovaná analýza

- Vychází z představy, že základem IS jsou data. Služby IS slouží pro pořízení a exploraci dat.
- Doporučuje proto nejprve analyzovat požadavky a definovat konceptuální datový model řešeného systému.
- Konceptuální datový model musí postihovat data přicházející přes hranici systému jako vstupní data související s událostmi, dále data, která se v systému ukládají a nakonec rovněž data, která systém produkuje na výstupu.
- Teprve později doplníme model o další části.

Postup datově orient. analýzy

1. Seznam událostí, kontext, datový slovník
2. Identifikace dat, která s událostí souvisí (základních objektů)
3. Identifikace vztahů mezi objekty
4. Scénáře jednání (původce, událost, akce, participanti, výstupy - reakce)
5. Modelování životních cyklů objektů
6. Popis akcí (minispecifikace základních akcí)

Jak hledat data?

Doporučení č.1:

- Analyzujeme odborný článek, vybereme všechna podstatná jména.
- Roztřídíme je do skupin:
  - kandidáti na typy objektů (entity),
  - kandidáti na vlastnosti objektů (atributy),
  - ostatní (kandidáti na aktéry, smetí).

Příklad: Odborný článek pro „Výtah“

Systém “Výtah” slouží pro logické řízení obsluhy výtahu s jednou či více šachtami (předpokládají se 4 šachty a 40 úrovní). Systém zajišťuje efektivní plánování sběru a odvozu pasažérů mezi obsluhovanými patry podle požadavků (požadavek na přivolání výtahu pro jízdu směrem nahoru nebo dolů, požadavek na dopravení do určitého patra). Směr jízdy se nemění, dokud výtah nesplní objednávky v daném směru (výtah neví o pasažérech – neexistuje indikace prázdnosti klece). Přeplněný výtah nereaguje na výzvy (existuje indikace přetížení). Pro každou šachtu existuje samostatný motor ovládaný signály (povely UP, DOWN a STOP). Povel STOP způsobí zastavení výtahu v nejbližším patře v daném směru a otevření dveří výtahu (dveře se dají otevřít až v patře). Uvnitř klece je panel s tlačítky pater, indikace aktuální polohy a tlačítko STOP. Tlačítko STOP zabrání zavření dveří (jde mimo systém). Rovněž otevírání a zavírání dveří jde mimo systém (kvůli bezpečnosti). Příkazy pro systém jsou akceptovány až po zavření dveří. Operátor výtahu má k dispozici tlačítko ON/OFF, kterým zadává požadavek na zastavení pohybu výtahů.

Zpracovaný článek

systém “Výtah”
logické řízení
šachta
úroveň
pasažér
patro
požadavek
požadavek na přivolání výtahu pro jízdu směrem nahoru
požadavek na přivolání výtahu pro jízdu směrem dolů
požadavek na dopravení do patra
směr jízdy
objednávka
indikace prázdnosti klece
výzva
indikace přetížení
motor
signál
povel UP
povel DOWN
povel STOP
dveře výtahu
klec
panel s tlačítky pater
indikace aktuální polohy
tlačítko STOP
příkaz pro systém
operátor výtahu
tlačítko ON/OFF
požadavek na zastavení pohybu

Kandidáti na aktéry

pasažér

indikace přetížení

motor

indikace aktuální polohy (patra)

tlačítko STOP

operátor výtahu

tlačítko ON/OFF

Kandidáti na typy dat

šachta (atribut klece)

úroveň alias patro

požadavek alias objednávka alias příkaz pro systém alias výzva

požadavek na přivolání výtahu pro jízdu směrem nahoru

požadavek na přivolání výtahu pro jízdu směrem dolů

požadavek na dopravení do patra

směr jízdy (atribut)

indikace prázdnosti klece (neexistuje)

indikace přetížení

signál alias povel (pro motor)

povel UP

povel DOWN

povel STOP

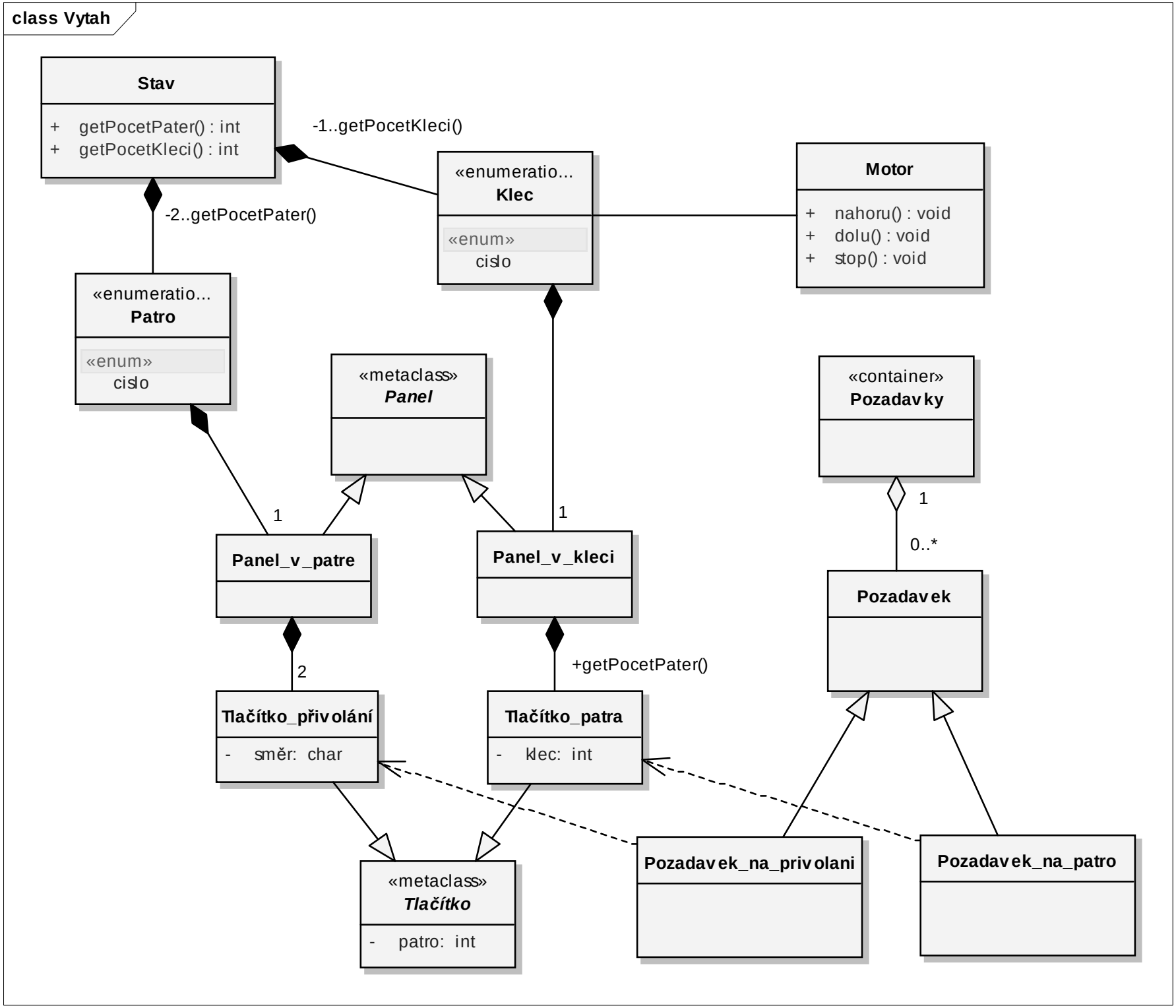
klec

panel s tlačítky pater

indikace aktuální polohy

tlačítko STOP (jde mimo systém)

tlačítko ON/OFF alias požadavek na zastavení pohybu

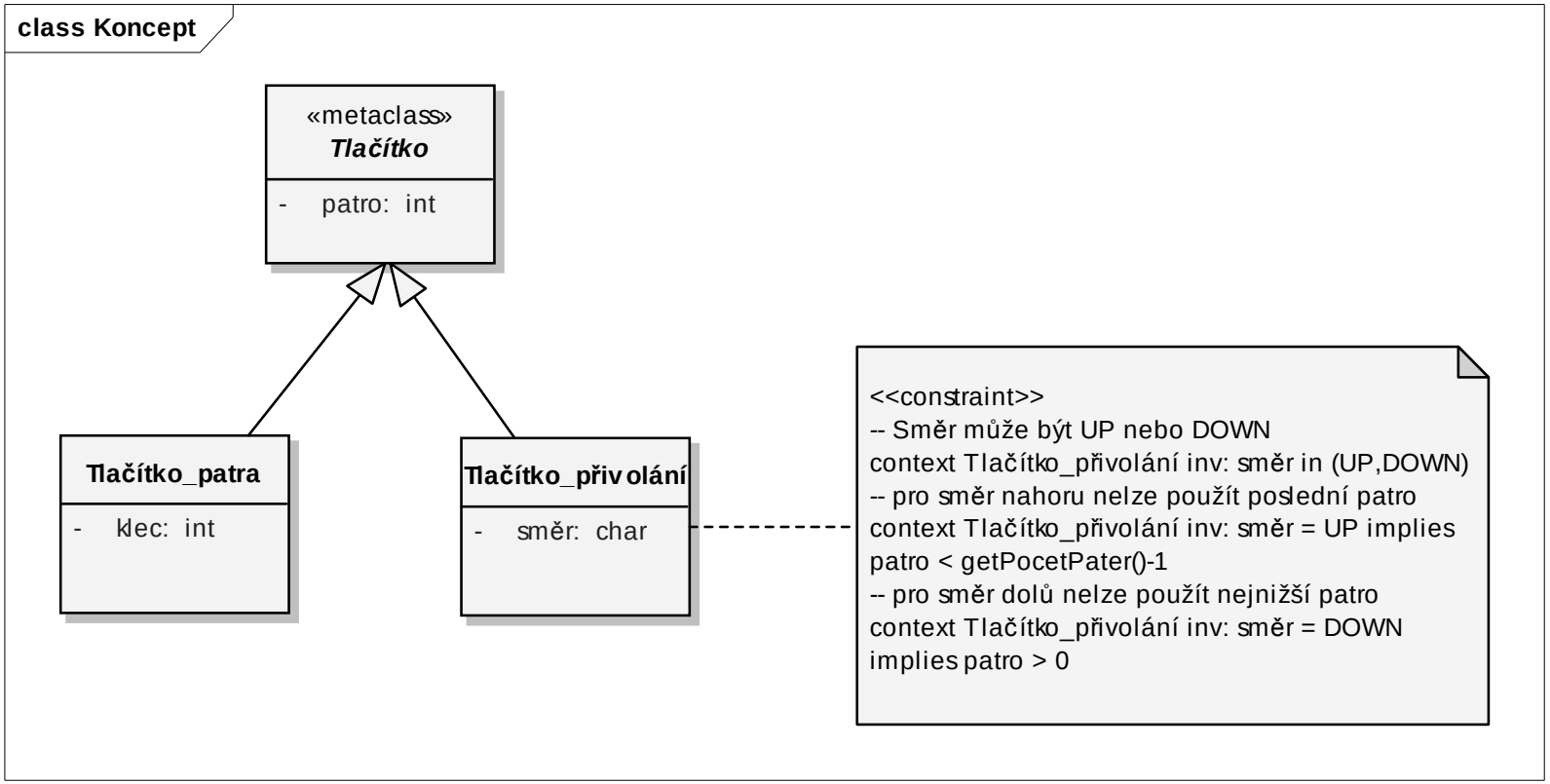

class Vytah
Stav
+ getPocetPater() : int
+ getPocetKleci() : int
-1..getPocetKleci()
-2..getPocetPater()
«enumeratio...
Klec
«enum»
cislo
Motor
+ nahoru() : void
+ dolu() : void
+ stop() : void
«enumeratio...
Patro
«enum»
cislo
«metaclass»
Panel
«container»
Pozadavky
1
1
1
0..*
Panel_v_patre
Panel_v_kleci
Pozadavek
2
+getPocetPater()
Tlačítko_přivolání
- směr: char
Tlačítko_patra
- klec: int
«metaclass»
Tlačítko
- patro: int
Pozadavek_na_privolani
Pozadavek_na_patro

Něco diagramem vyjádřit nelze

V příkladu systému Výtah je to např.:

- Tlačítko pro přivolání pro jízdu směrem nahoru na panelu v posledním patře, tj. když patro má hodnotu getPocetPater() neexistuje.
- Tlačítko pro přivolání pro jízdu směrem dolů na panelu v prvním patře neexistuje.

Připojení omezení k prvkům



Jak hledat data?

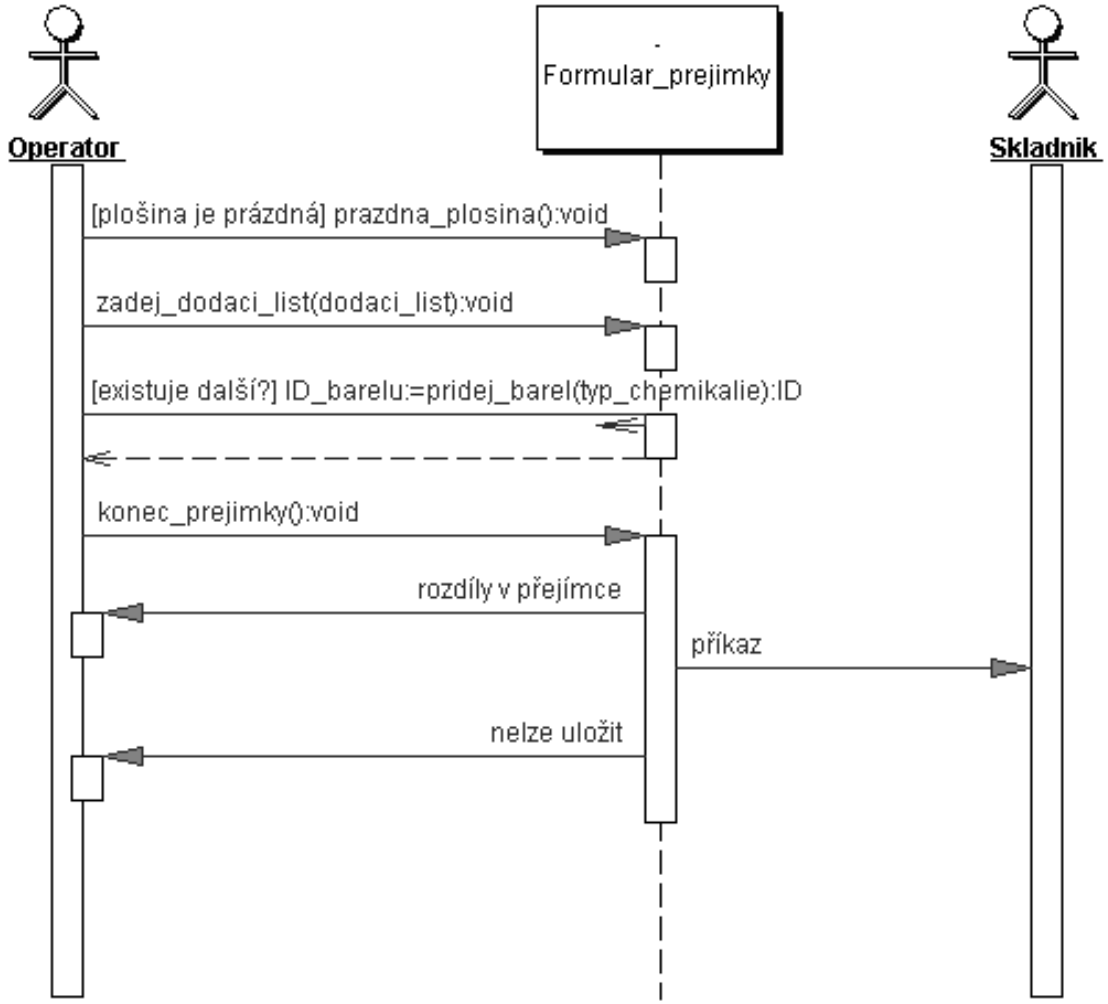
Doporučení č.2:

- Analyzujeme seznam událostí, rozpoznáváme data, která s událostmi souvisí.
- Roztřídíme je do skupin:
  - kandidáti na typy objektů (entity),
  - kandidáti na vlastnosti objektů (atributy).

Příklad: Události pro ECO sklad

- Operátor zahájil přejímku
- Operátor zahájil dodávku
- Manažer se ptá na stav skladu
- Manažer se ptá na bezpečnost skladu

Scénář pro přejímku



Kandidáti na typy dat

dodací list

barel

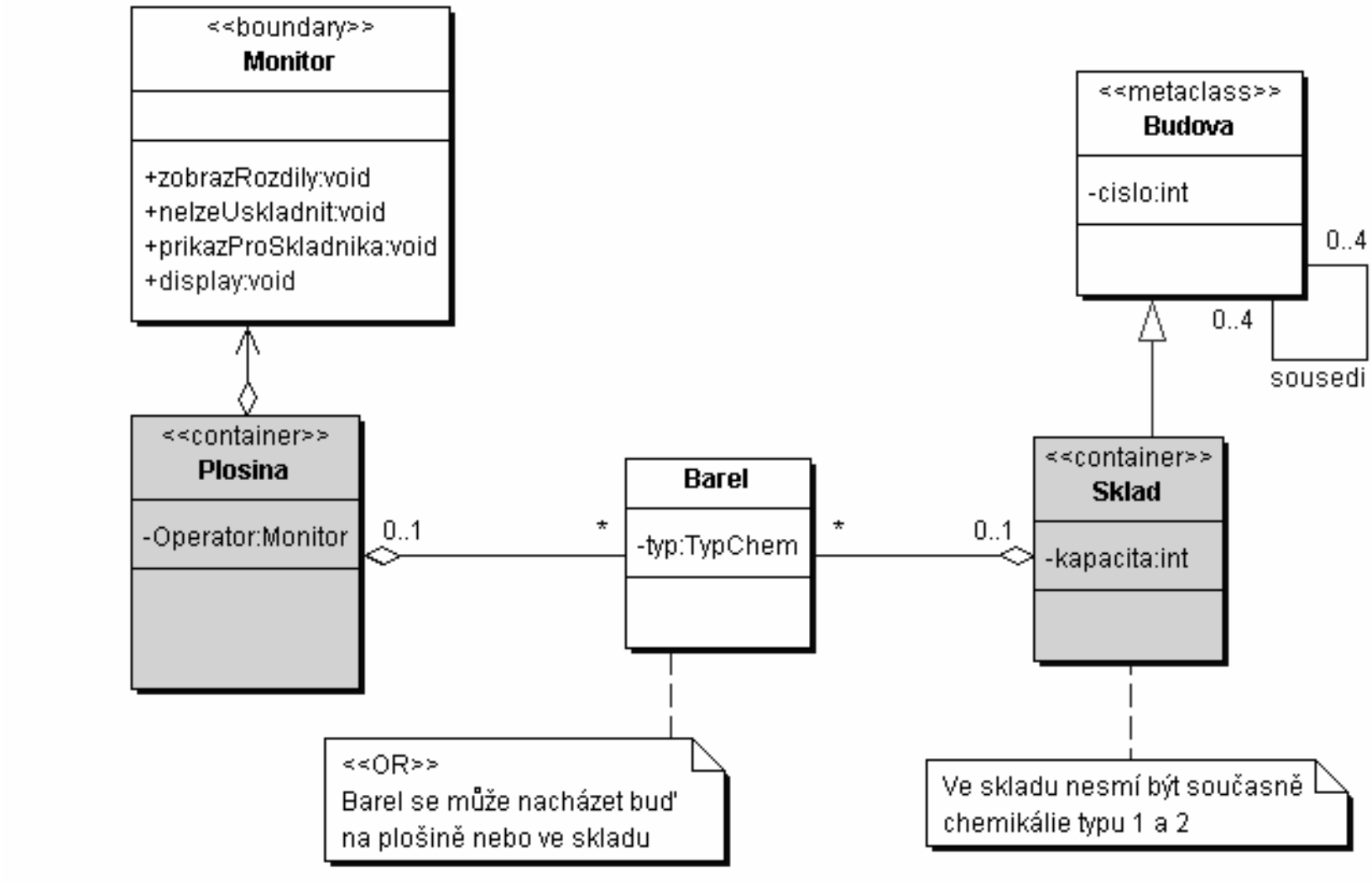
typ chemikálie

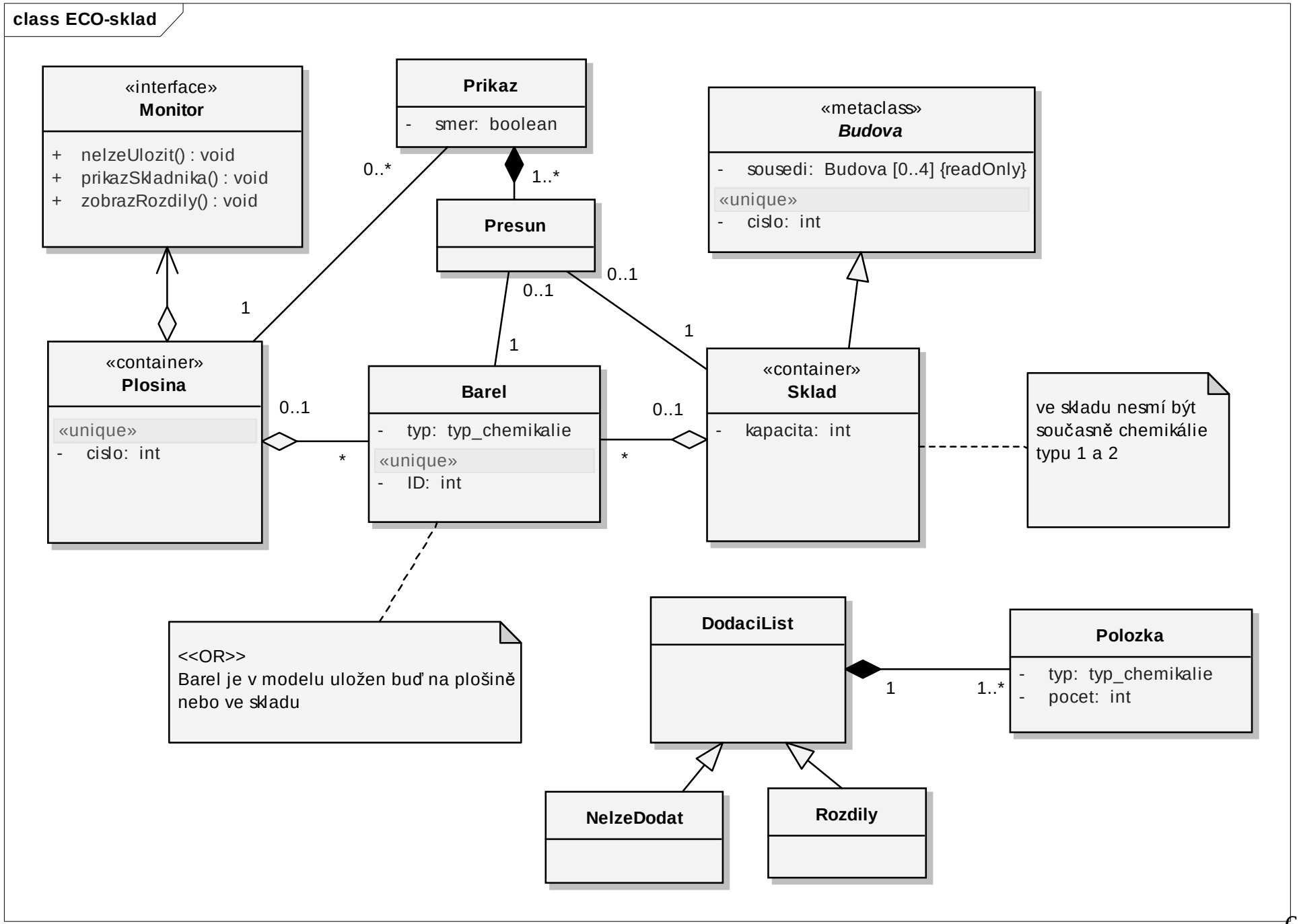
rozdíly v přejímce

nelze uložit

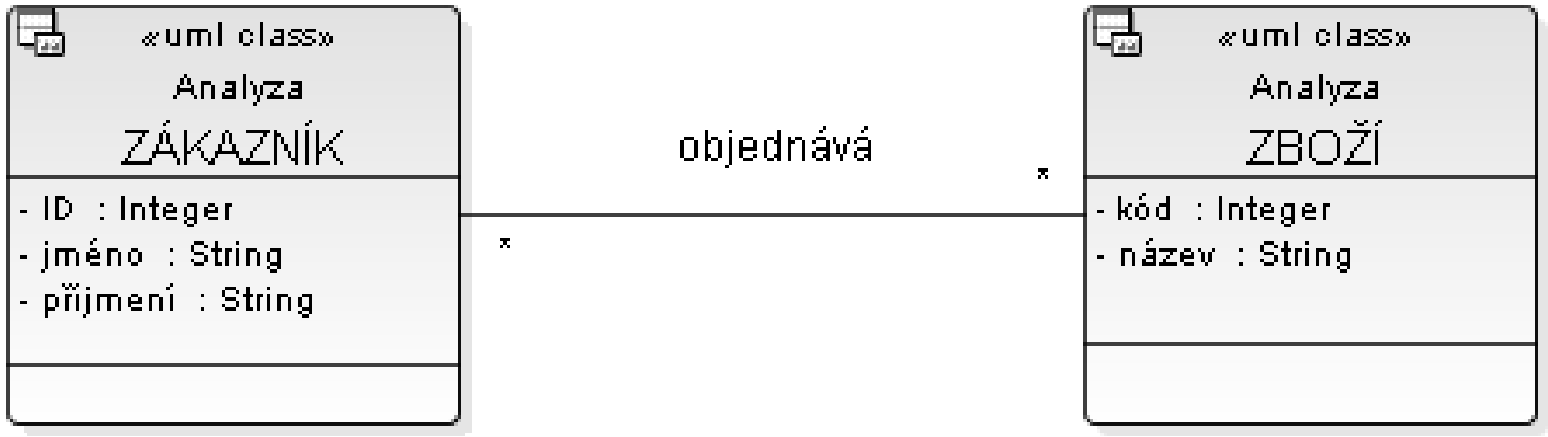
příkaz pro skladníka

Datový model pro ECO-sklad (1.)

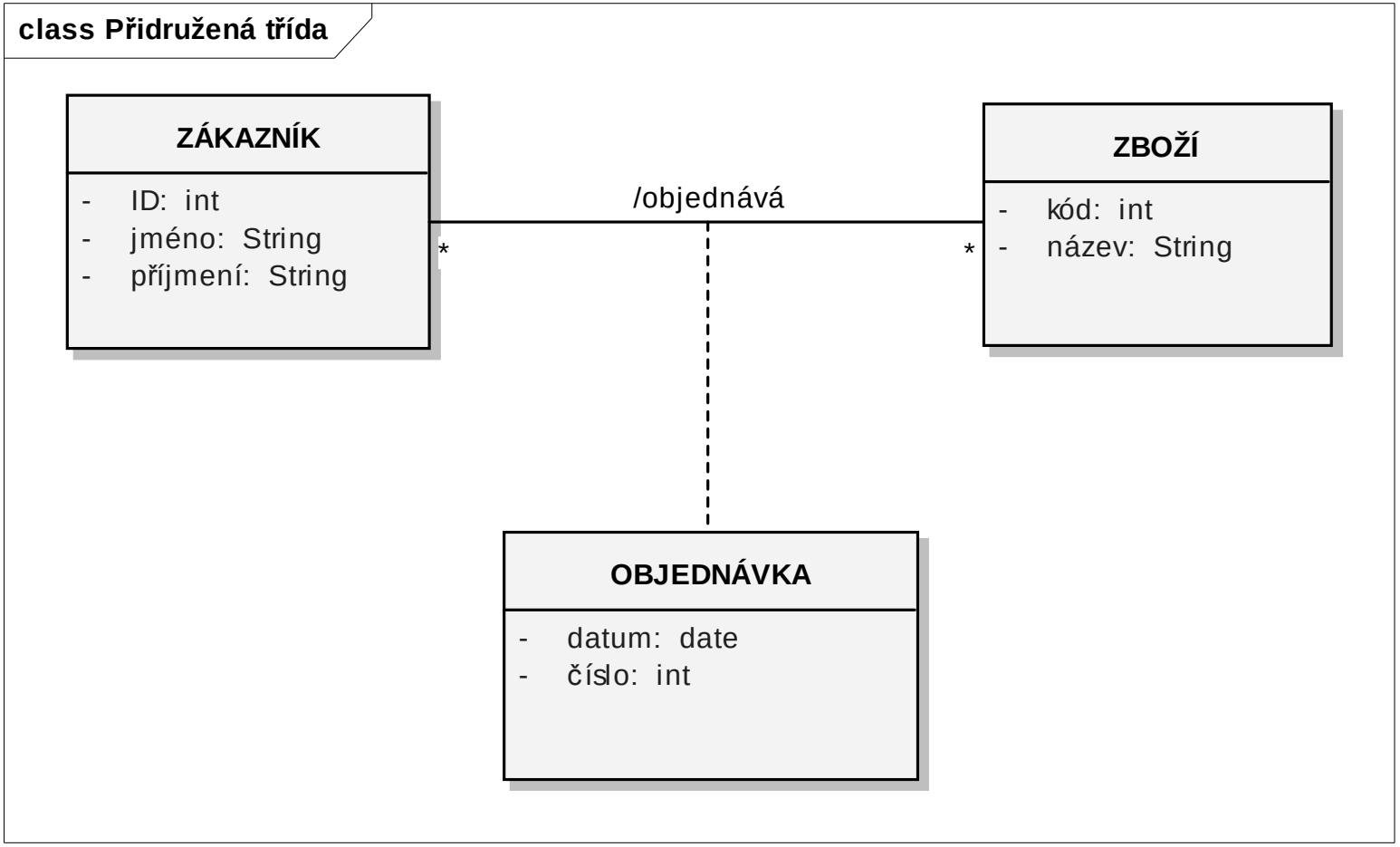



class ECO-sklad
«interface»
Monitor
+ nelzeUlozit() : void
+ prikazSkladnika() : void
+ zobrazRozdily() : void
Prikaz
- smer: boolean
0..*
1..*
Presun
«metaclass»
Budova
- sousedi: Budova [0..4] {readOnly}
«unique»
- cislo: int
0..1
0..1
1
1
1
«container»
Plosina
«unique»
- cislo: int
0..1
*
Barel
- typ: typ_chemikalie
«unique»
- ID: int
*
0..1
«container»
Sklad
- kapacita: int
ve skladu nesmí být
současně chemikálie
typu 1 a 2
<<OR>>
Barel je v modelu uložen buď na plošině
nebo ve skladu
DodaciList
1
1..*
Polozka
- typ: typ_chemikalie
- pocet: int
NelzeDodat
Rozdily

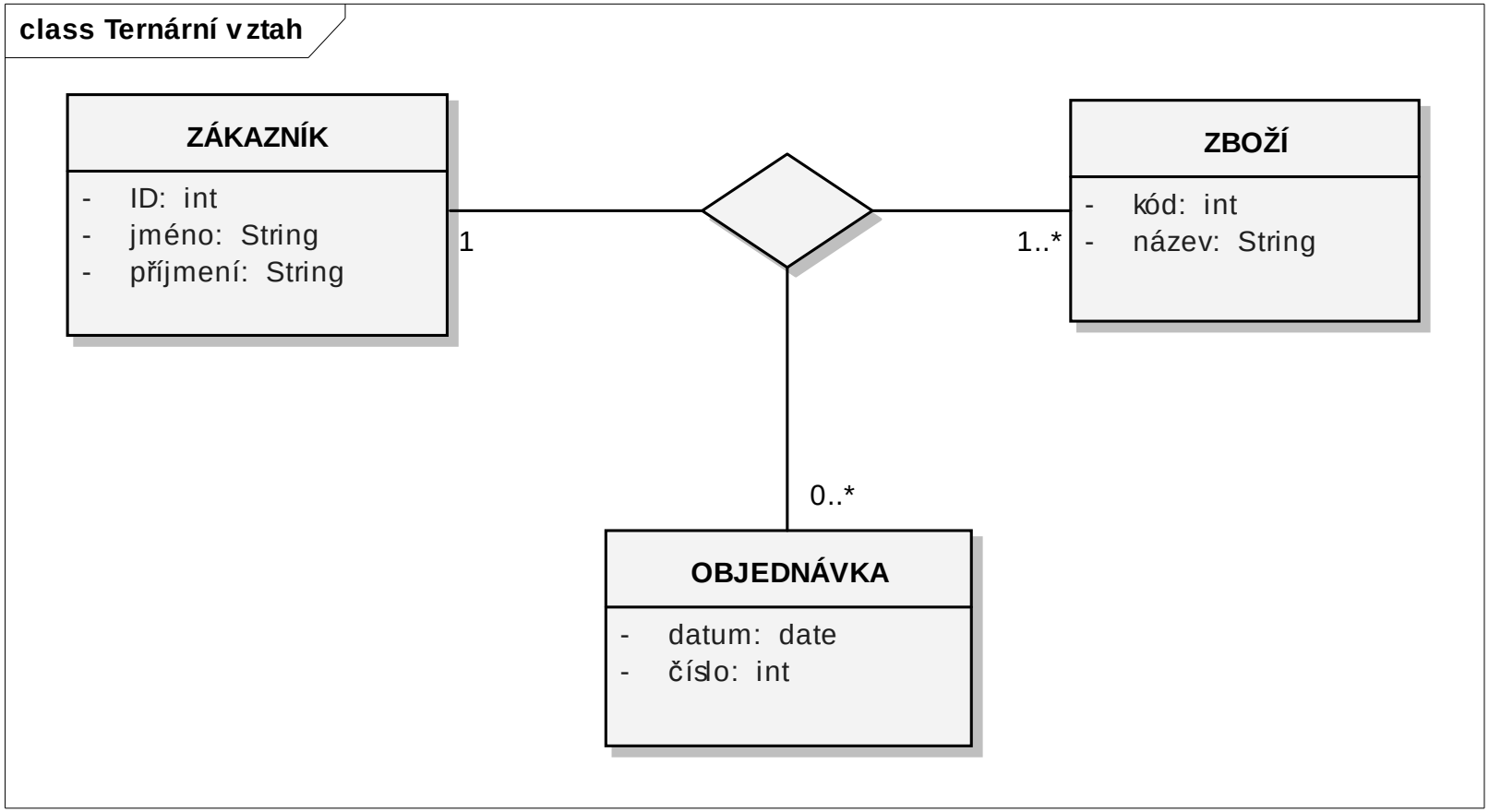
Pár poznámek – první model



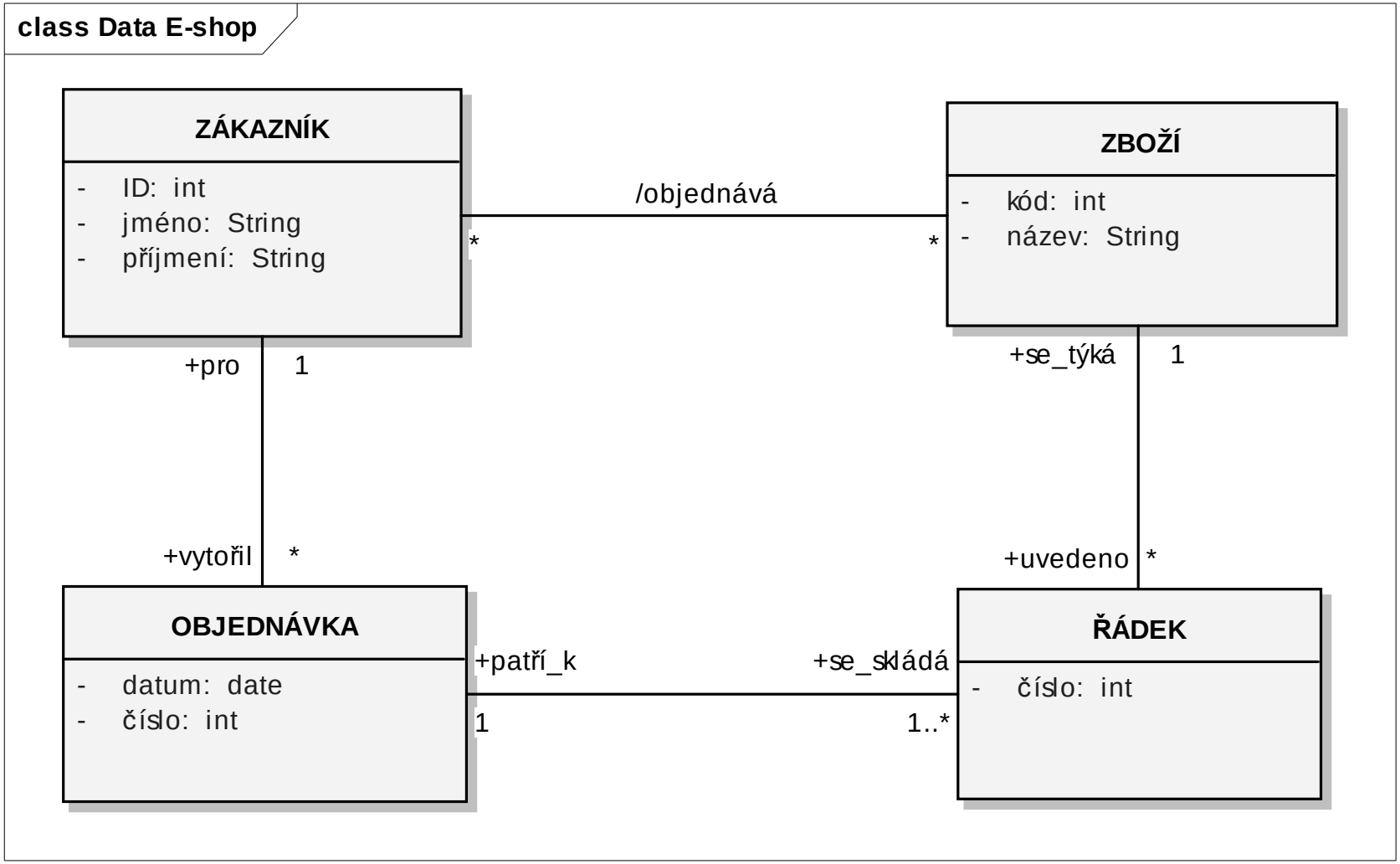
Přidružená třída



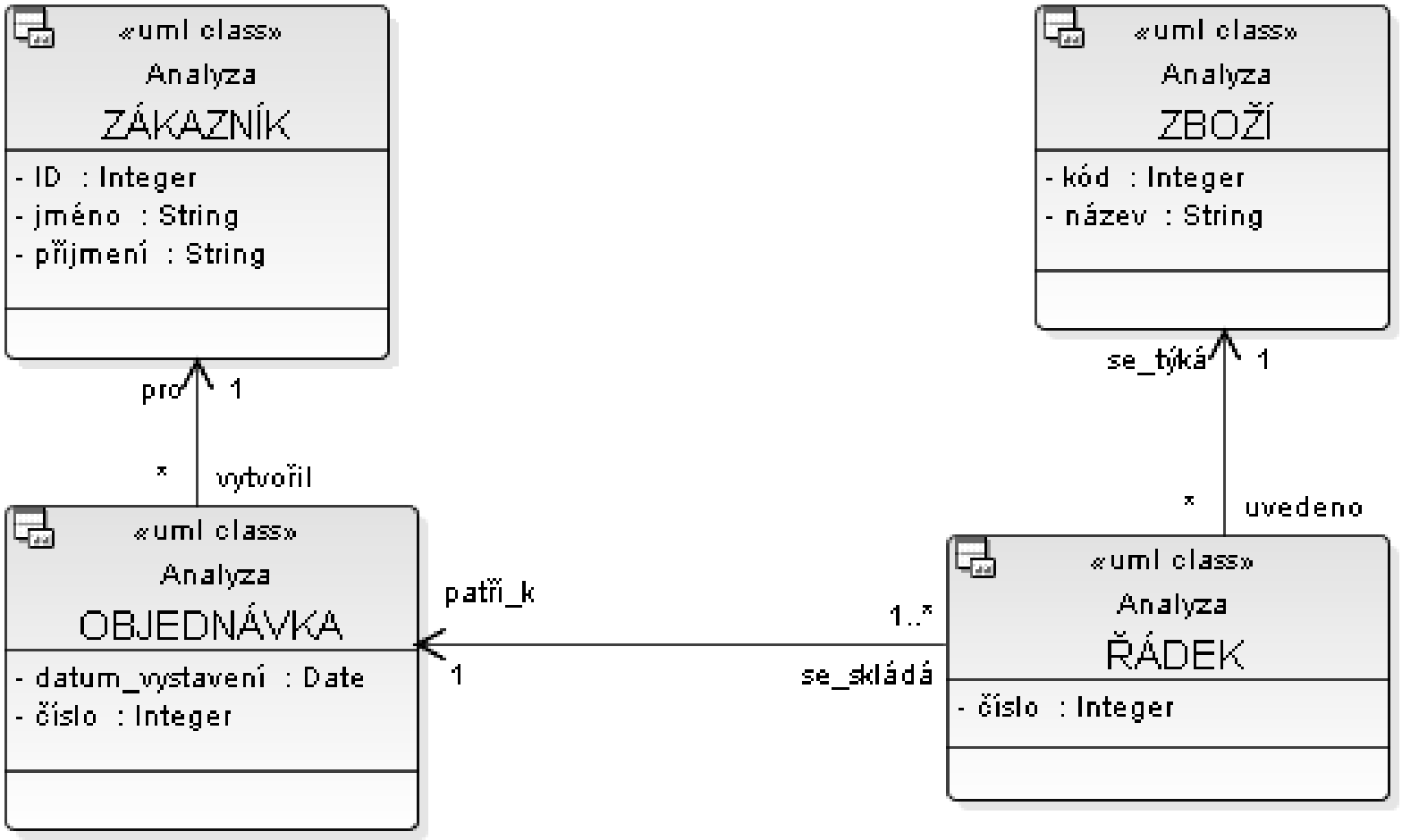
Nebo ternární vztah



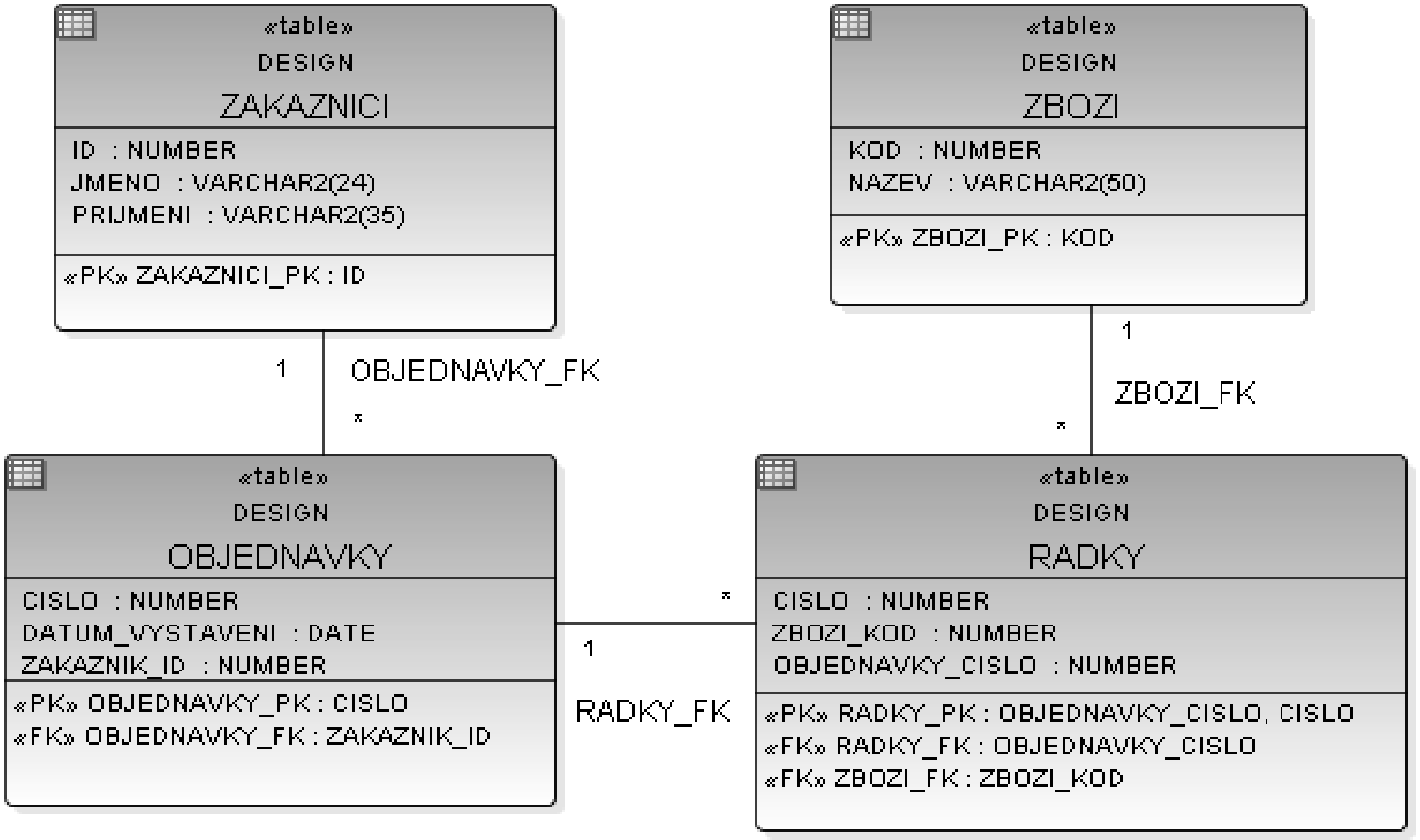
Podrobnější model



Používání šipek?



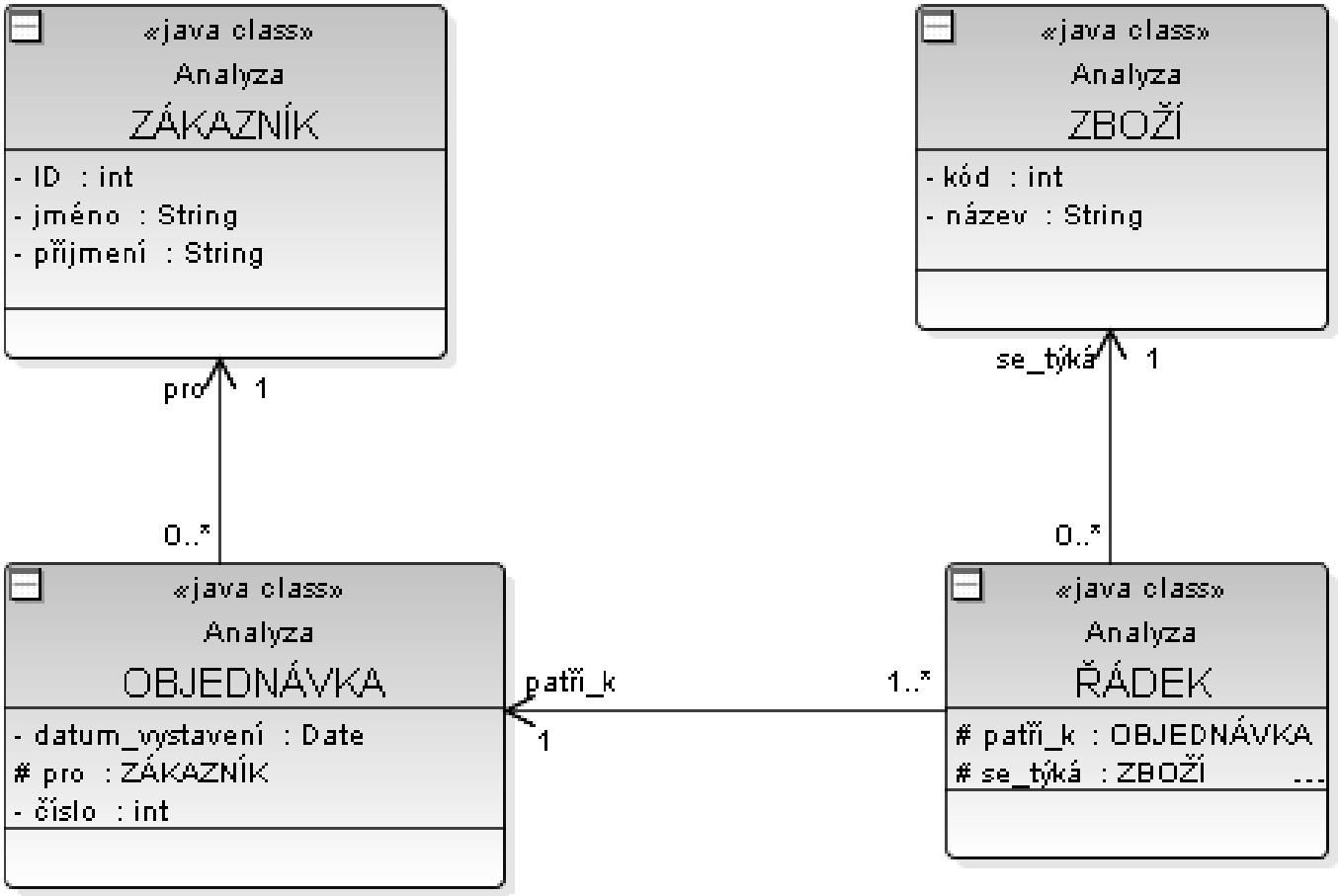
Generovaný logický model (SQL)



Generovaný kód

```
…
CREATE TABLE "RADKY„ (
  "CISLO" NUMBER NOT NULL,
  "ZBOZI_KOD" NUMBER NOT NULL,
  "OBJEDNAVKY_CISLO" NUMBER NOT NULL
);
ALTER TABLE "RADKY"
  ADD CONSTRAINT "RADKY_PK" PRIMARY KEY
    ("OBJEDNAVKY_CISLO","CISLO") ENABLE;
ALTER TABLE "RADKY"
  ADD CONSTRAINT "RADKY_FK" FOREIGN KEY
    ("OBJEDNAVKY_CISLO")
  REFERENCES "OBJEDNAVKY„ ("CISLO") ENABLE;
ALTER TABLE "RADKY"
  ADD CONSTRAINT "ZBOZI_FK" FOREIGN KEY ("ZBOZI_KOD")
  REFERENCES "ZBOZI„ ("KOD") ENABLE;
…
```

Generovaný logický model (Java)



Generovaný kód

```
…
package Analyza;

public class RADEK
{
  private int cislo;
  /* @label ObjednavkaZbozi
   * @clientCardinality 0..*
   */
  protected Analyza.ZBOZI se_tyka;
  /* @label obsahuje
   * @clientCardinality 1..*
   */
  protected Analyza.OBJEDNAVKA patri_k;
}
…
```

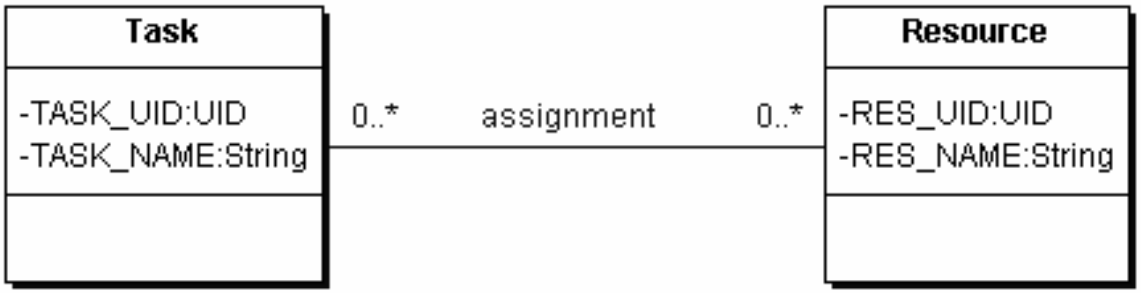
Příklad: MS Project

- Požadavky: aplikace bude pracovat s úlohami, zdroji a vztahy.
- Odtud kandidáti na entity (typy objektů, třídy):

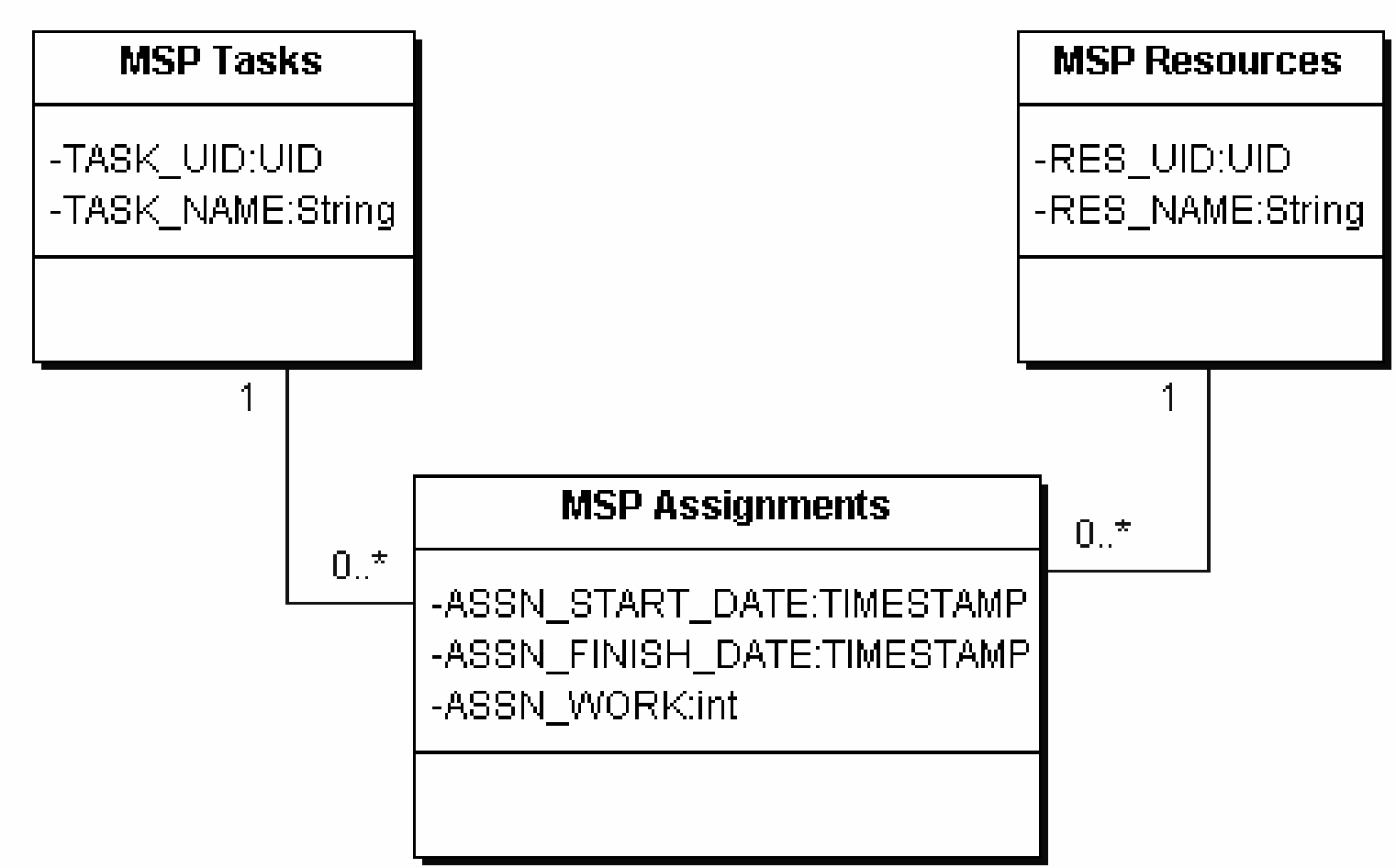
  Úloha

  Zdroj

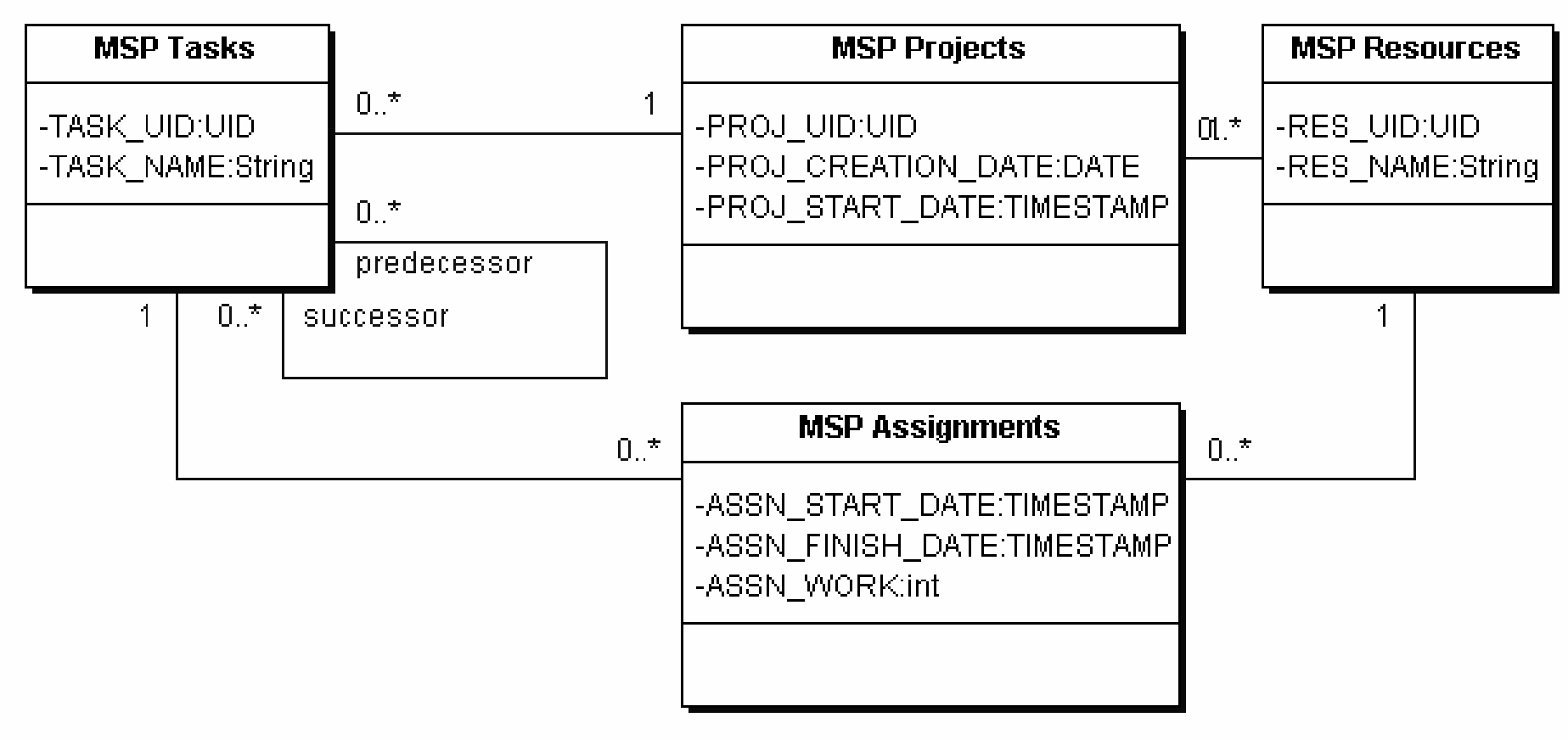
  Přiřazení



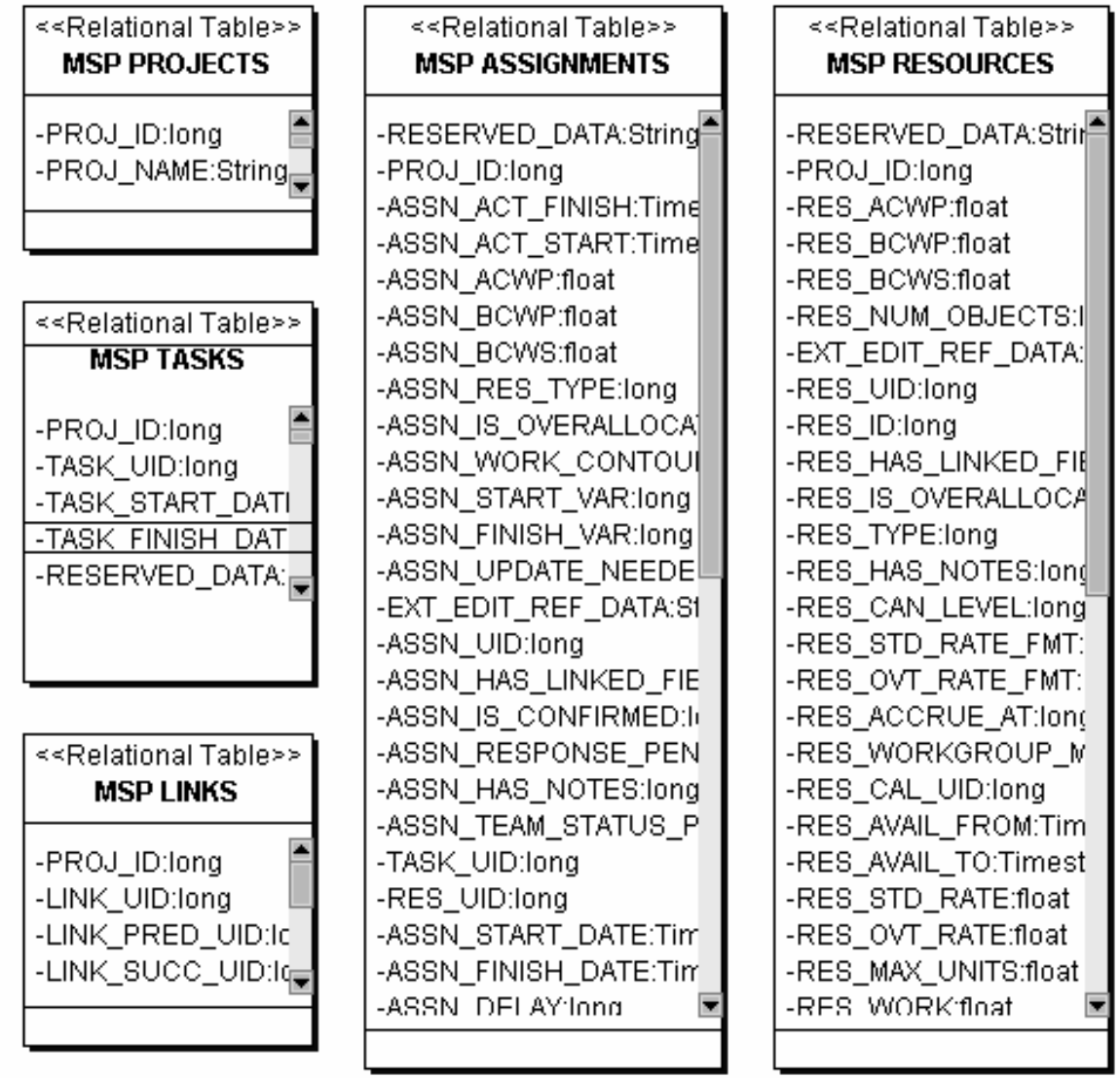
Podrobnější model



Ještě podrobnější model



Použijeme-li relační databázi (část)



Skutečná implementace

```
CREATE TABLE  MSP_TASKS (
PROJ_ID NUMBER(18,0),
TASK_UID NUMBER(18,0), … , PRIMARY KEY (PROJ_ID,TASK_UID)
);

CREATE TABLE MSP_RESOURCES (
PROJ_ID NUMBER(18,0),
RES_UID NUMBER(18,0),
RES_NAME VARCHAR2(255), … , PRIMARY KEY (PROJ_ID,RES_UID)
);

CREATE TABLE MSP_LINKS (
PROJ_ID NUMBER(18,0),
LINK_UID NUMBER(18,0),
LINK_PRED_UID NUMBER(18,0),
LINK_SUCC_UID NUMBER(18,0), … ,
FOREIGN KEY (PROJ_ID, LINK_PRED_UID)
REFERENCES MSP_TASKS (PROJ_ID, TASK_UID) …
);
```

Další postup

- Z datového modelu se snažíme odvodit funkce:
  - Vytvoříme matici CRUD (Create, Read, Update, Delete)
  - Zkoumáme, zda pro každý typ dat existuje odpovídající funkce
- Z datového modelu se snažíme odvodit dynamiku:
  - Pro každý typ dat zkoumáme, zda objekty nevykazují změny stavu

Matice CRUD

- Řádky odpovídají typům objektů.
- Sloupce odpovídají funkcím.
- V průsečíku je zapsáno zda funkce C,R,U a/nebo D odpovídající data.
- V každém řádku by mělo někde být vše (některá funkce musí objekt vytvářet, jiná využívat, či rušit).

Matice CRUD pro ECO sklad

	Prázdná plošina	Zadej dodací list	Zařaď barel	Konec přejímky	Dodávka	Zahájení práce systému ECO sklad	Ukončení práce systému ECO sklad
Plošina	U		U		U	C	D
Sklad				U	U	C,Get	D,Save
Monitor				U,Print	U,Print	C	D
Barel			C				
Dodací list		C		R,D			
Příkaz				C,Print	C,Print		

Co jsme zjistili?

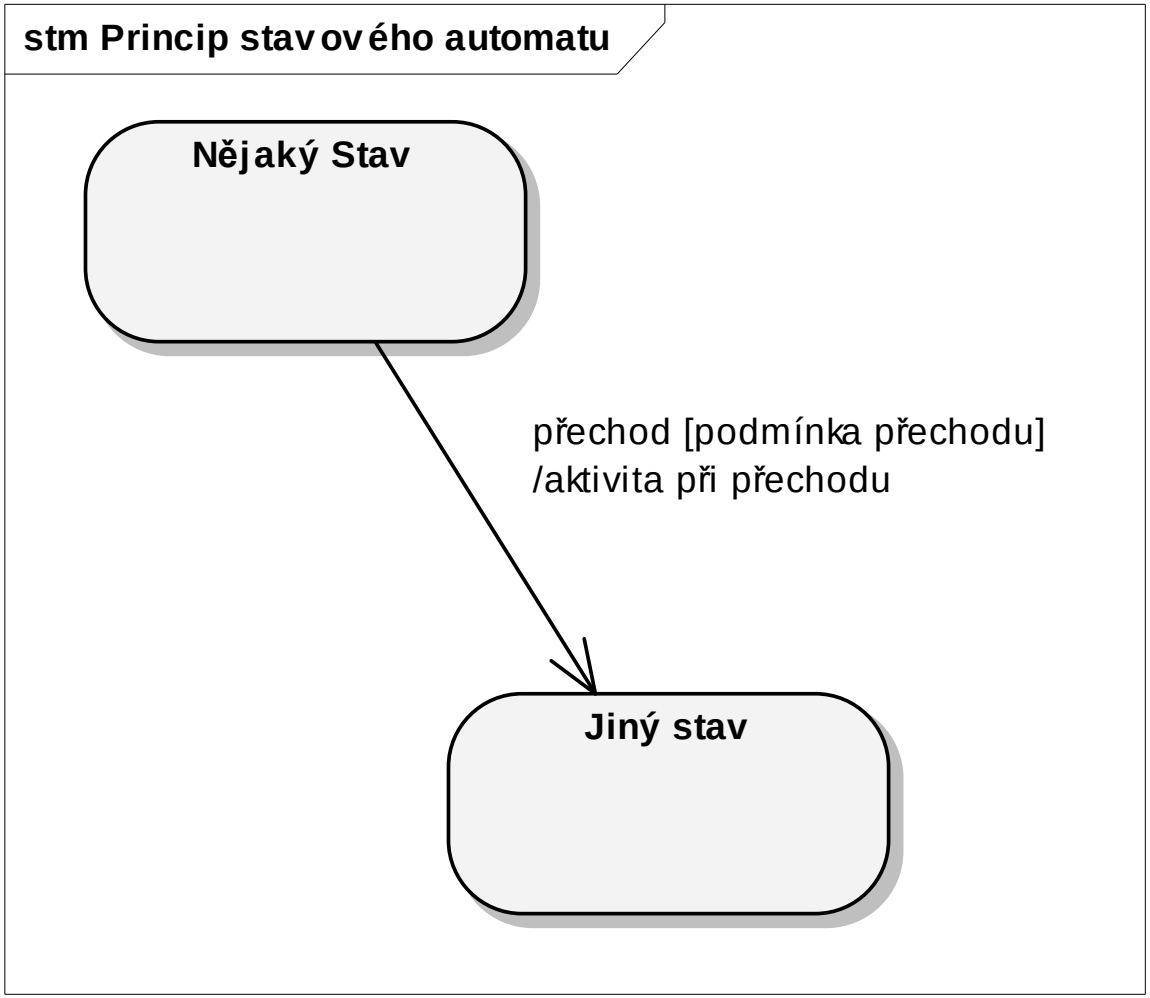
- Potřebujeme ještě v rámci nějaké funkce reprezentaci barelu zrušit.
- Mohla by to udělat funkce „dodávka“, neboť po vyskladnění barelu jeho životní cyklus končí.
- Doplníme tedy do popisu funkce dodávka požadavek „pokud v rámci dodávky využijeme některý barel, vymažeme jeho reprezentaci z obsahu skladu a zrušíme ji“.
- Do matice CRUD přidáme odpovídající D.

Dynamický model

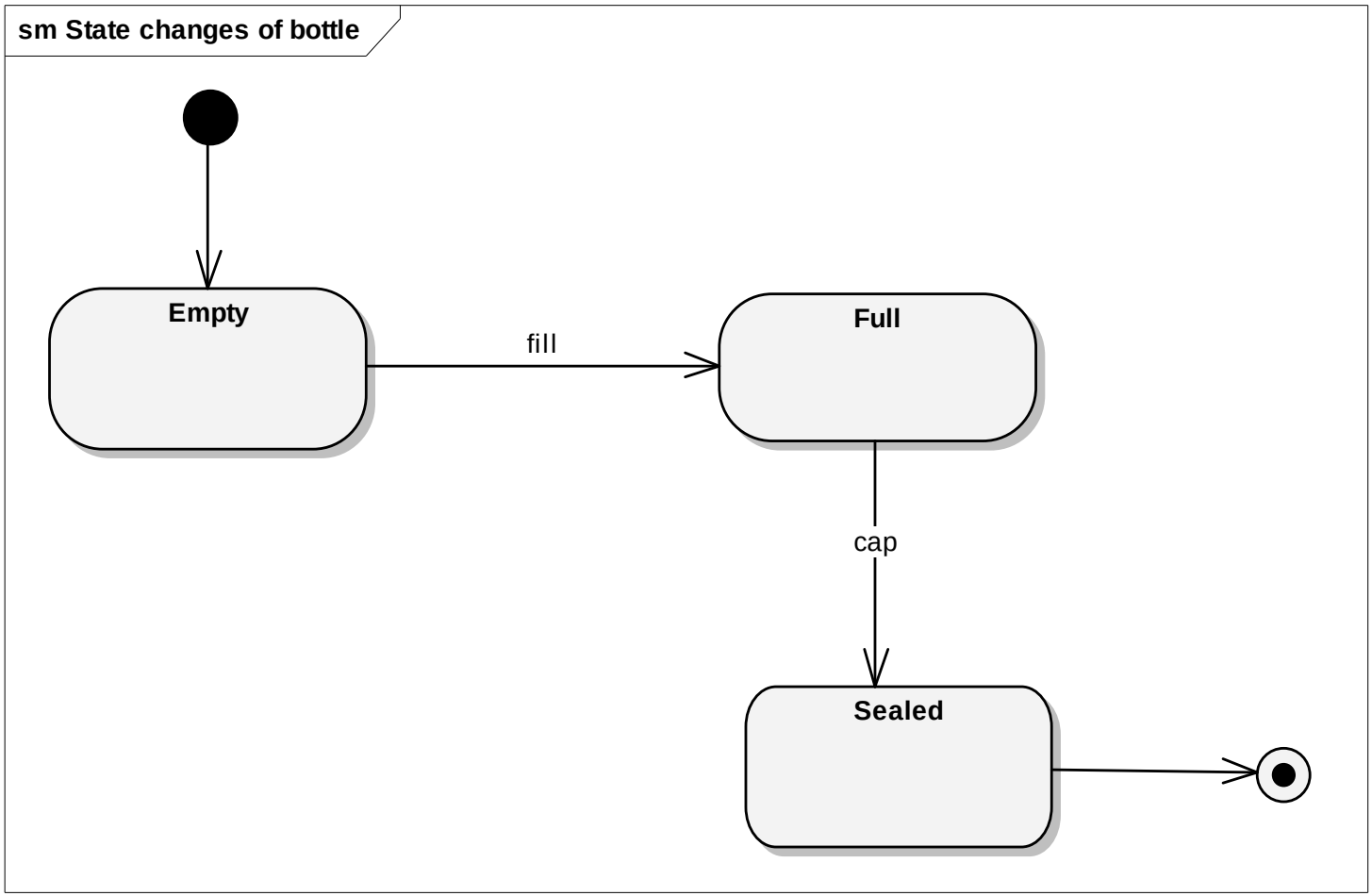
Stavové diagramy

- Slouží k popisu dynamiky systému
- Stavový diagram definuje možné **stavy**, možné **přechody** mezi stavy, **události**, které přechody iniciují, **podmínky** přechodů a **akce**, které s přechody souvisí
- Stavový diagram lze použít pro popis dynamiky objektu (pokud má rozpoznatelné stavy), pro popis metody (pokud známe algoritmus), či pro popis protokolu (včetně protokolu o styku uživatele se systémem)

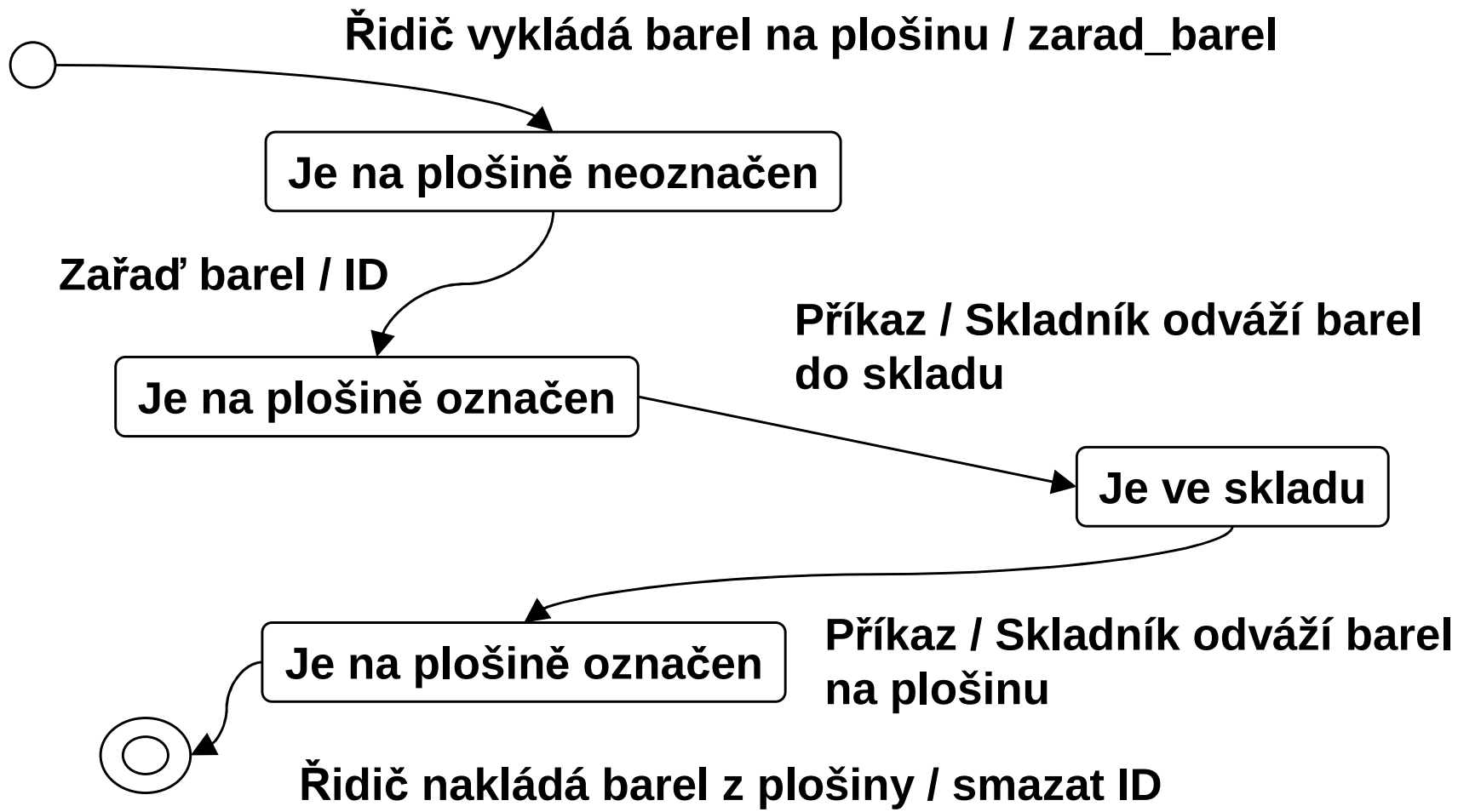
Princip stavového automatu



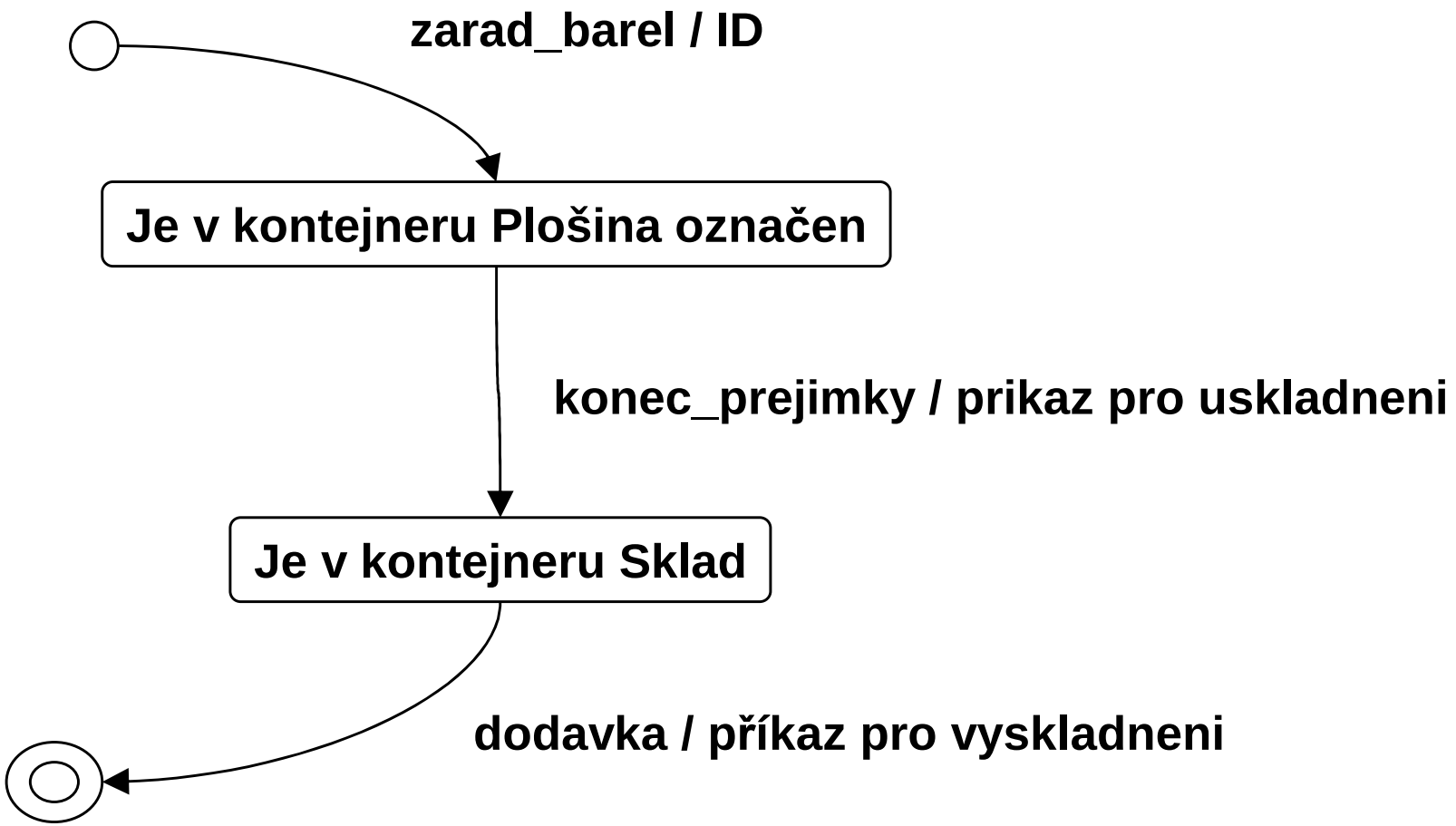
Životní cyklus jako stavový diag.



Životní cyklus skutečného „barelu”

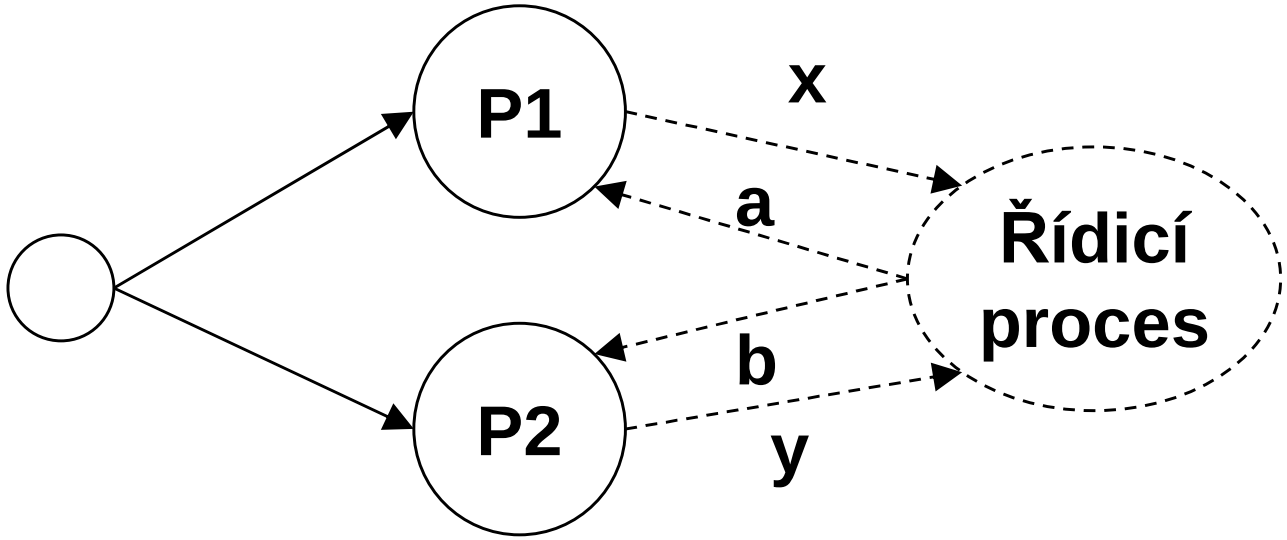


Životní cyklus entity „barel”

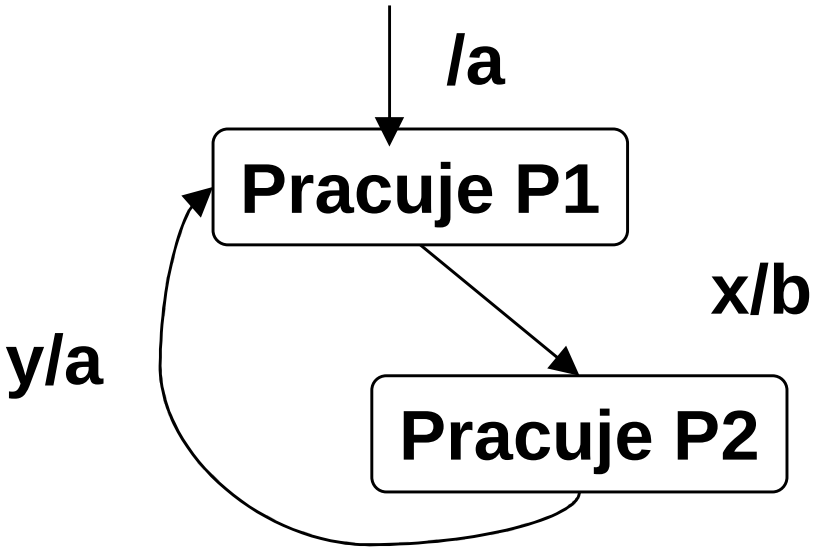


Popis řídicích procesů pomocí stavových diagramů

- Vstupy řídicího procesu lze modelovat pomocí událostí stavového diagramu.
- Výstupy řídicího procesu lze modelovat pomocí akcí stavového diagramu.
- Pak lze řídicí procesy modelovat stavovými diagramy.



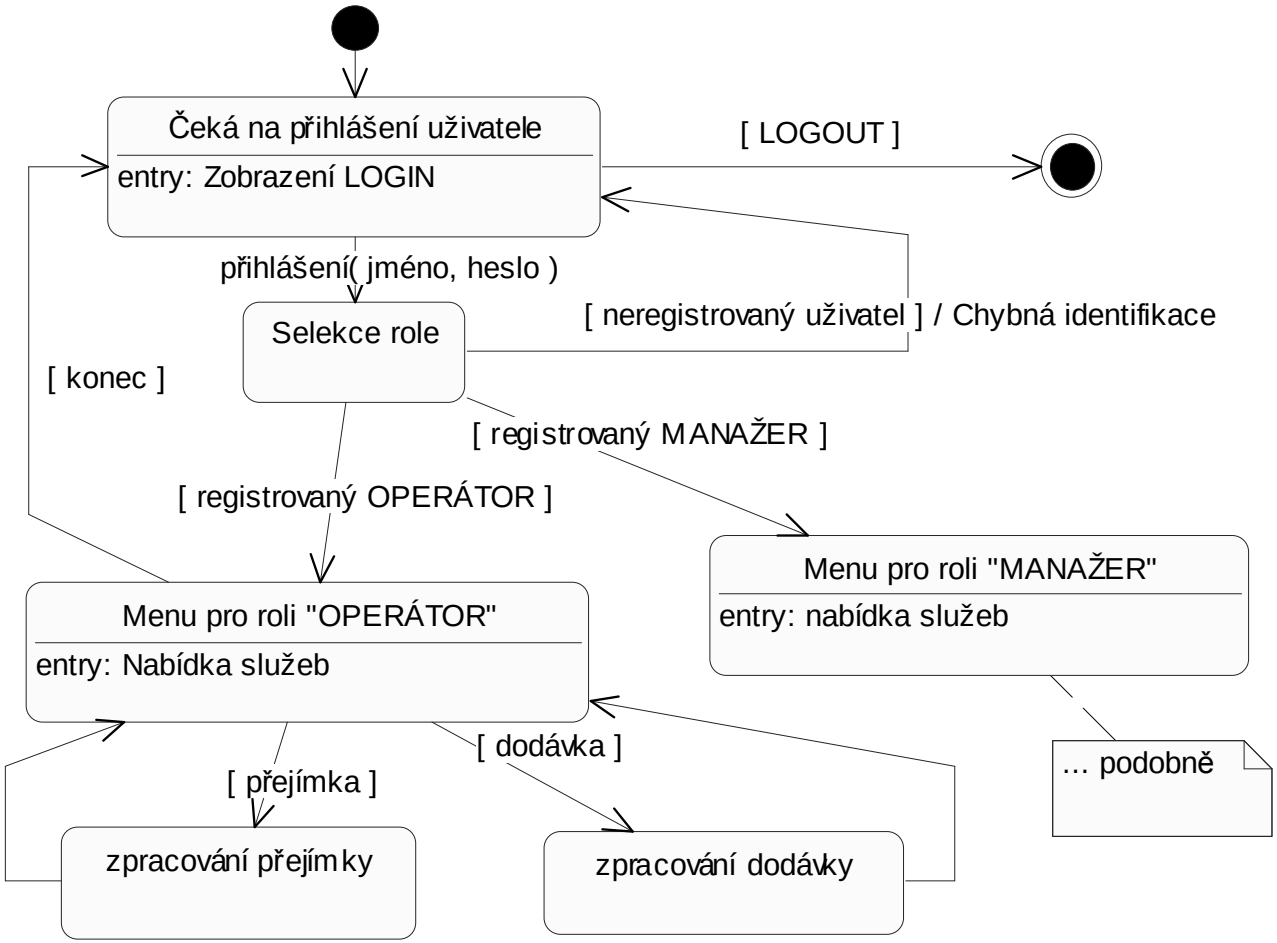
střídavě spouštíme P1 a P2



Životní cyklus systému

- Vyjádření souhrné dynamiky systému, která je zachycena ve scénářích
- Definuje povolené návaznosti akcí a reakcí
- Představuje hrubou “uživatelskou příručku” pro systém
- Definice systému jako “konečného automatu”

Životní cyklus ECO-skladu



Životní cyklus jako regulární výraz

<Životní cyklus> = Lifecycle <jméno objektu> : <regulární výraz>

<regulární výraz> =

<akce>	
| #<reakce>	
| <regulární výraz>. <regulární výraz>	sekvence
| [<regulární výraz>]	volitelně
| <regulární výraz>*	iterace
| (<regulární výraz> | <regulární výraz>)	selekce
| (<regulární výraz> || <regulární výraz>)	paralelně

<akce> = jméno události

<reakce> = jméno reakce

Životní cyklus "ECO-skladu"

Lifecycle ECO-sklad:

(dodávka | přejímka)* || (dotaz na stav | je bezpečný?)*

přejímka = prázdná plošina. dodací list.

(barel k zařazení. #ID barelu)*.

konec přejímky. [#rozdíly v přejímce].

#příkaz pro skladníka . [#nelze uložit]

dodávka = prázdná plošina.požadovaná dodávka.

#skutečná dodávka. #příkaz pro skladníka

dotaz na stav = …

je bezpečný? = ...

Životní cyklus entity „barel”

Lifecycle BAREL:

zarad_barel . #ID barelu . #příkaz pro uskladnění . dodávka . #příkaz pro vyskladnění

Kontroly analytických modelů

Výstup analýzy

Konceptuální model:

- datový model popisuje entity, atributy, vztahy, integritní omezení,
- funkční model popisuje služby, které systém poskytuje pro záznam, údržbu a využití dat,
- dynamický model popisuje možné stavy dat a jejich změny.

Kontrola výstupů analýzy

- kontrola jednotlivých modelů (pohledů)
- kontrola vzájemné konzistence modelů

Kontrola datového modelu

- je datový model úplný?
  - existuje entita pro každý typ objektu?
  - nejsou zde nadbytečné entity (entity tvořené pouze identifikací, entity s jedinou instancí, apod.)?
  - jsou zde zaneseny všechny vztahy (včetně generalizací a agregací)?
  - nejsou zde odvoditelné vztahy?
  - je model v normální formě?
  - jsou zanesena všechna integritní omezení?

Nadbytečné entity

- entity tvořené pouze identifikací
- entity s jedinou instancí
- entity s vazbou typy 1:1
- apod.
  - Dobrou technikou je představit si příklady entit a objektů?

Normalizace datového modelu

Předpoklad: všechny entity jsou jednoznačně identifikovatelné označenou kombinací atributů a/nebo vztahů

- 1.normální forma: entity neobsahují násobné atributy ani komponované atributy
- 2.normální forma (navíc): neklíčové atributy závisí pouze na celém klíči
- 3.normální forma (navíc): neklíčové atributy nejsou závislé na neklíčových položkách

Jsou zaneseny všechny vztahy?

- Nelze doplnit generalizace?
- Nelze doplnit agregace?
- Nelze model vylepšit?
- Příklad: Pro entitu „dodací list“ lze vymyslet pružnější model, který usnadní případné úpravy v budoucnosti

Nejsou zde odvoditelné vztahy?

- Zákazník si objednává zboží
- Zákazníkovi je vystavena faktura.
- Odebrané zboží je předmětem fakturace.
- ? Nejsou zde odvoditelné vztahy?
- Pozn.: Odvoditelné vztahy mohou v modelu být, ale musí být jako odvoditelné předznačeny znakem „/“ a doplněny způsobem odvození (formulí, popisem v OCL).

Jsou zanesena všechna integritní omezení?

Řadu vlastností dat nelze do diagramu zanést:

- Šéf musí mít větší plat než jeho podřízení.
- V jednom skladu nelze umístit chemikálie typu „1“ a „2“.

```
context s:Sklad inv : forAll(Barel x,y |
    s.obsahuje(x) and s.obsahuje(y)
    implies x.typ != 1 or y.typ != 2)
```

Vyvážení datového modelu

- datový model versus datový slovník
  - každá entita, atribut a vztah v DD
- datový model versus funkční dekompozice
  - každá paměť a datový tok obsahuje entitu, atribut nebo vztah (nebo jejich kombinaci)
- datový model versus minispecifikace
  - něco musí entity a vztahy vytvářet/rušit, číst/modifikovat (matice CRUD)

Kontrola funkčního modelu

- je funkční model úplný?
  - existuje funkce/metoda pro každou událost?
  - každá funkce/metoda musí být popsána dekompozicí, nebo mít minispecifikaci (vstupy a výstupy musí odpovídat)
  - nejsou zde nadbytečné funkce/metody?

Vyvážení funkčního modelu

- funkční model versus datový slovník
  - každá paměť a datový tok v DD
  - každý prvek DD se někde vyskytuje (jinak je zbytečný)
- funkční model versus datový model
  - každá data zmíněná ve funkce/metodě musí být popsána v datovém modelu
- funkční model versus dynamický model
  - každý řídicí proces má dynamický model (vstupy = podmínky, výstupy = akce)

Kontrola dynamického modelu

- je dynamický model úplný?
  - existuje model pro každou entitu, která může mít různé stavy?
  - existuje model pro každý řídicí proces?
  - existuje popis životního cyklu systému?

Životní cyklus "ECO-skladu"

Lifecycle ECO-sklad:

(dodávka | přejímka)* || (dotaz na stav | je bezpečný?)*

přejímka = prázdná plošina. dodací list.

(barel k zařazení. #ID barelu)*.

konec přejímky. [#rozdíly v přejímce].

#příkaz pro skladníka . [#nelze uložit]

dodávka = prázdná plošina.požadovaná dodávka.

#skutečná dodávka. #příkaz pro skladníka

dotaz na stav = …

je bezpečný? = ...

Životní cyklus pro „Výtah”

Lifecycle Výtah:

(požadavek)* || (ON/OFF)*

požadavek = [požadavek na přivolání | požadavek na patro]

požadavek na přivolání = [požadavek na jízdu dolů | požadavek na jízdu nahoru]

požadavek na jízdu dolů = tlačítko pro jízdu dolů . #indikace

požadavek na jízdu nahoru = tlačítko pro jízdu nahoru . #indikace

požadavek na patro = tlačítko patra . #indikace

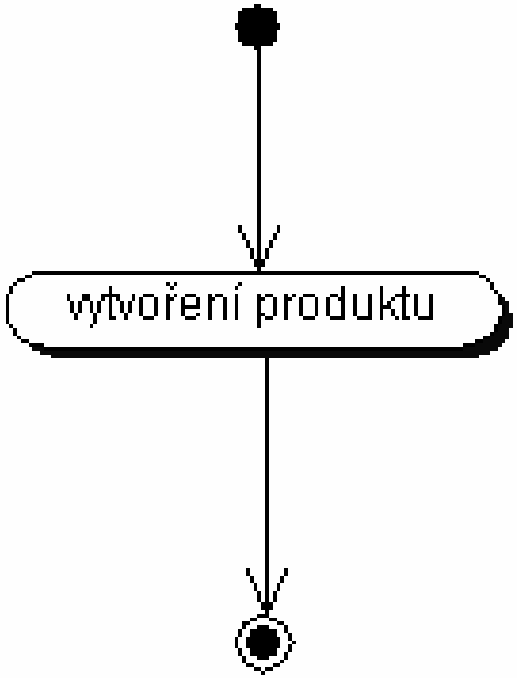
Poznámky k úvodní studii

Proč vytvářet úvodní studii?

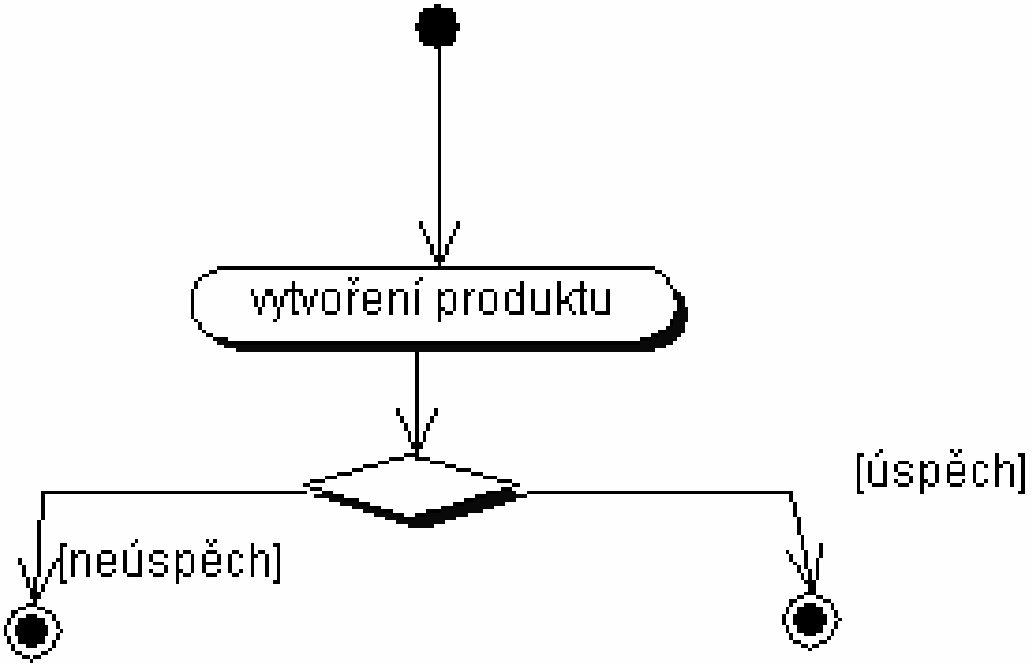
A. Protože to Richta chce.
B. Protože se to v komunitě informatiků sluší.
C. Protože to může ušetřit výdaje.

C je správně !!

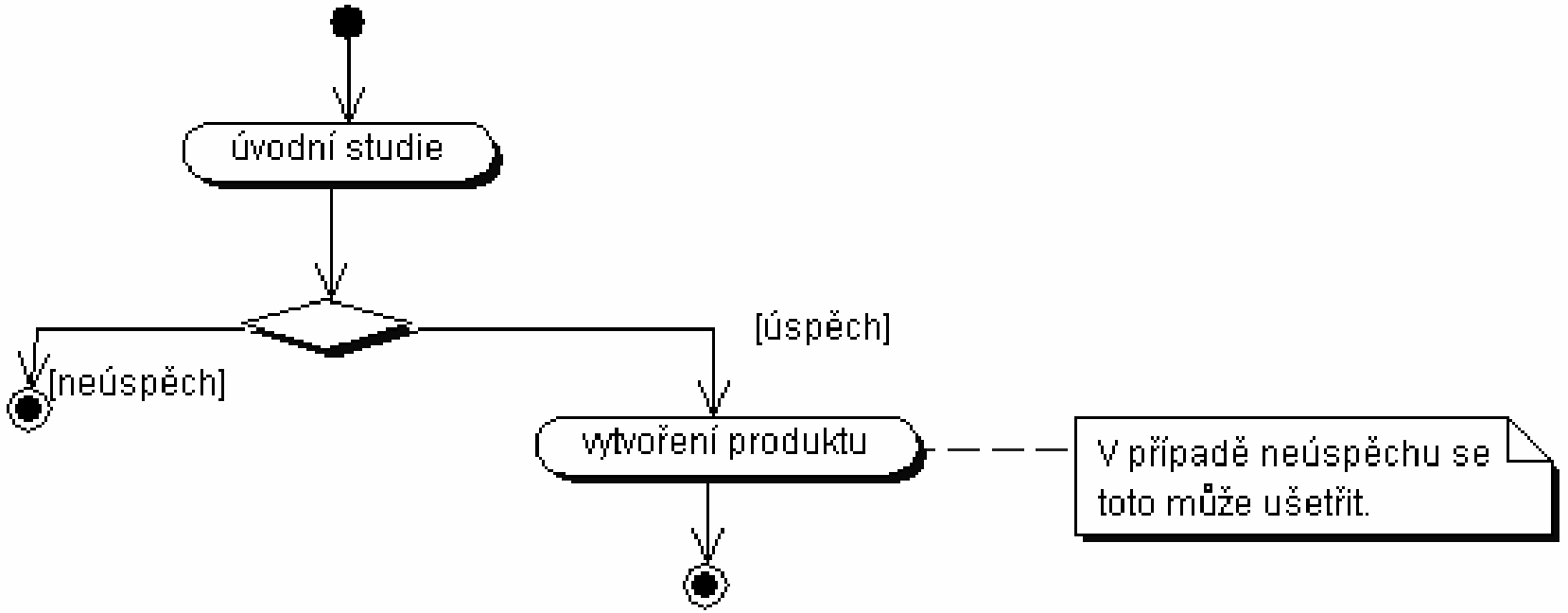
Jak to je ideální



Ale nemusí to vždy dopadnout



Úvodní studie může něco ušetřit



Jak se úvodní studie zakončí?

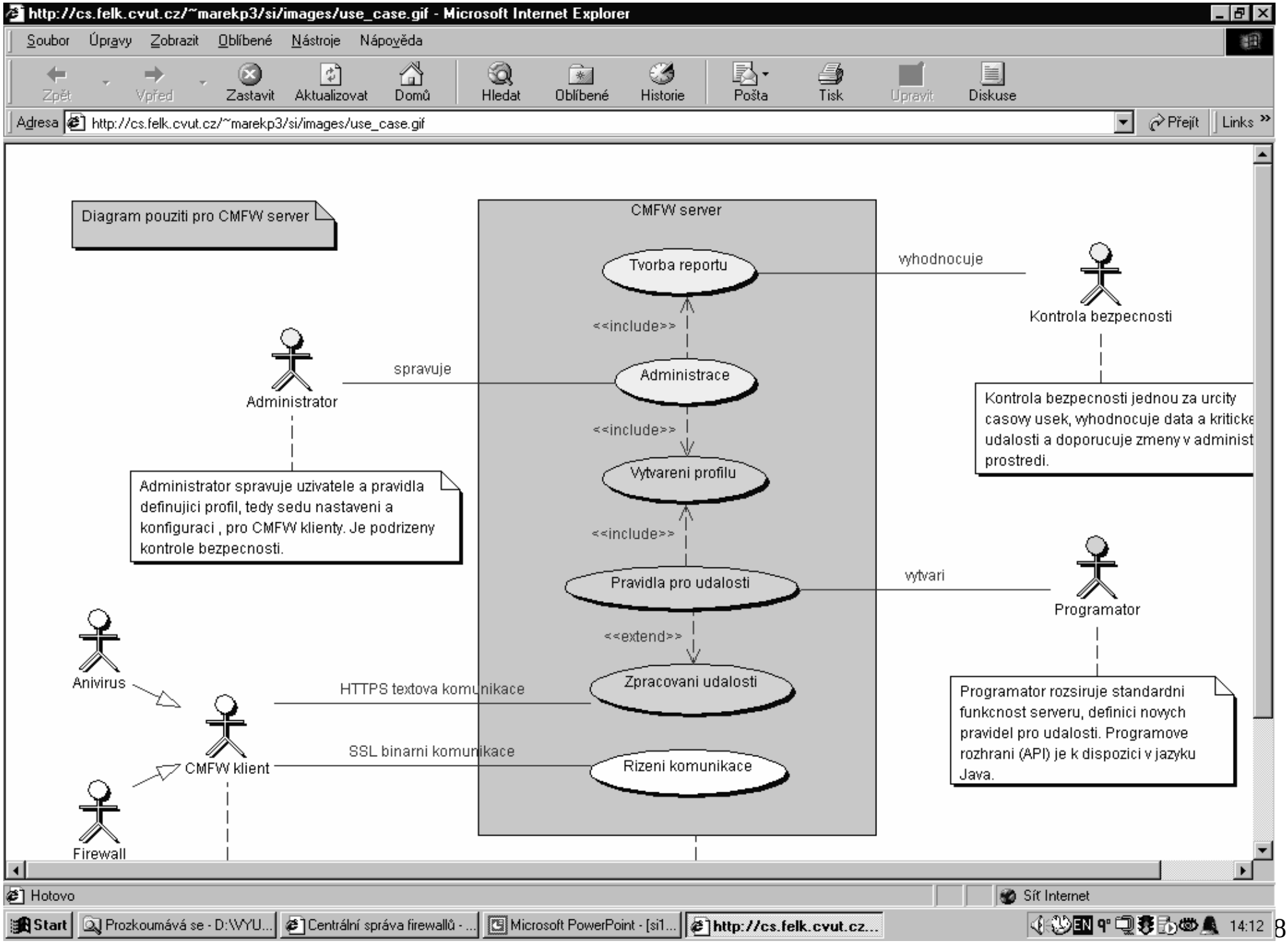
- Umístěním **dokumentace** na stránky projektu (deklarace záměru, odborný článek, model jednání/kontext, datový slovník, tým, kontakt, harmonogram (plán), matice zodpovědností, požadavky na HW a SW, rozpočet - 2x)
- Přípravou **prezentace** (umístí se rovněž na stránku projektu)
- Předvedením prezentace

Co to je prezentace úvodní studie?

- Stručný náznak o jaký produkt se jedná (definice hranice systému a poskytovaných služeb).
- Odhad nákladů a výnosů.
- Pokus o přesvědčení investora, že se vyplatí do projektu vložit peníze.
- Často je vhodné vymezit různé varianty řešení.

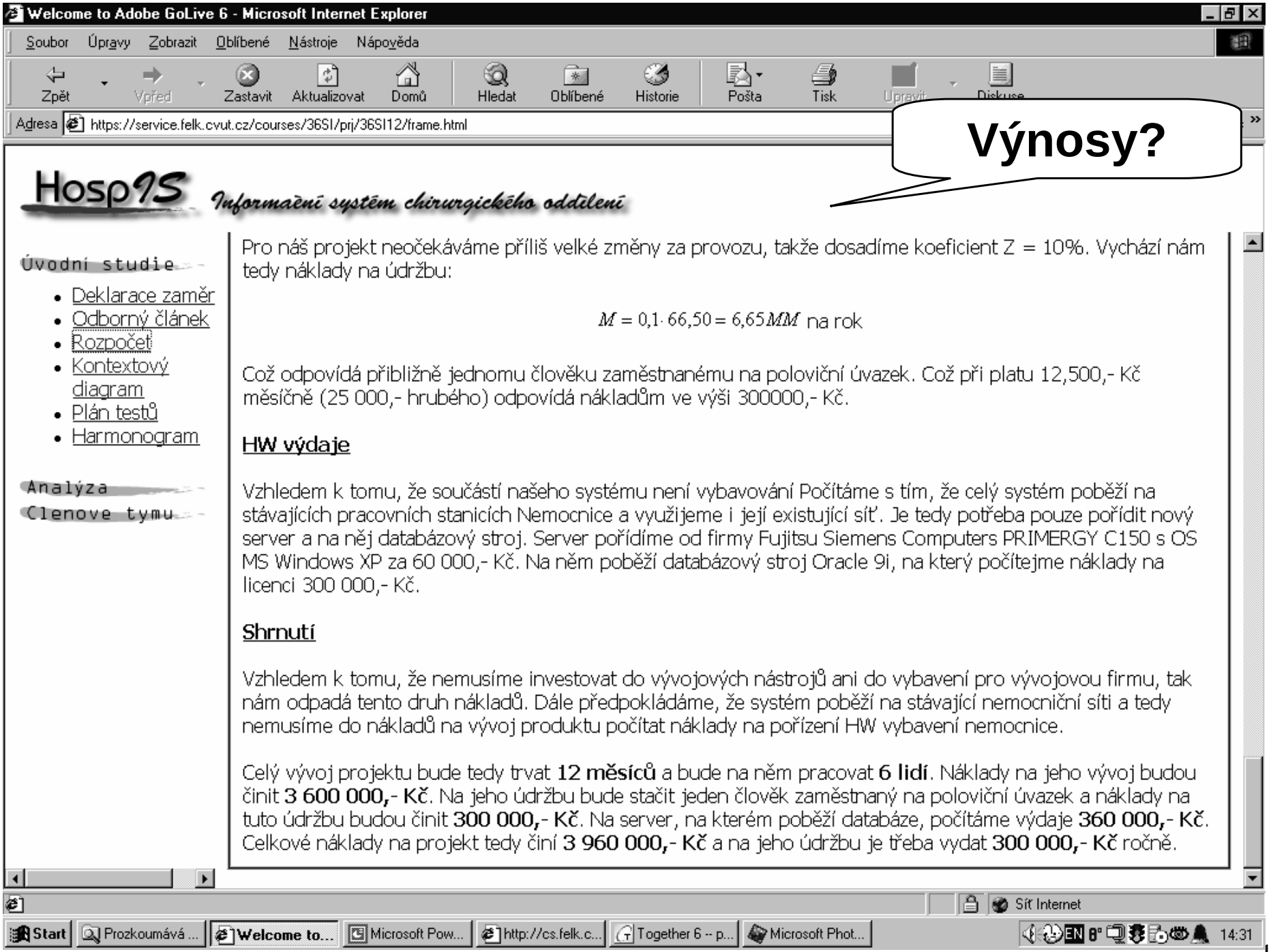
Stručná definice hranice systému

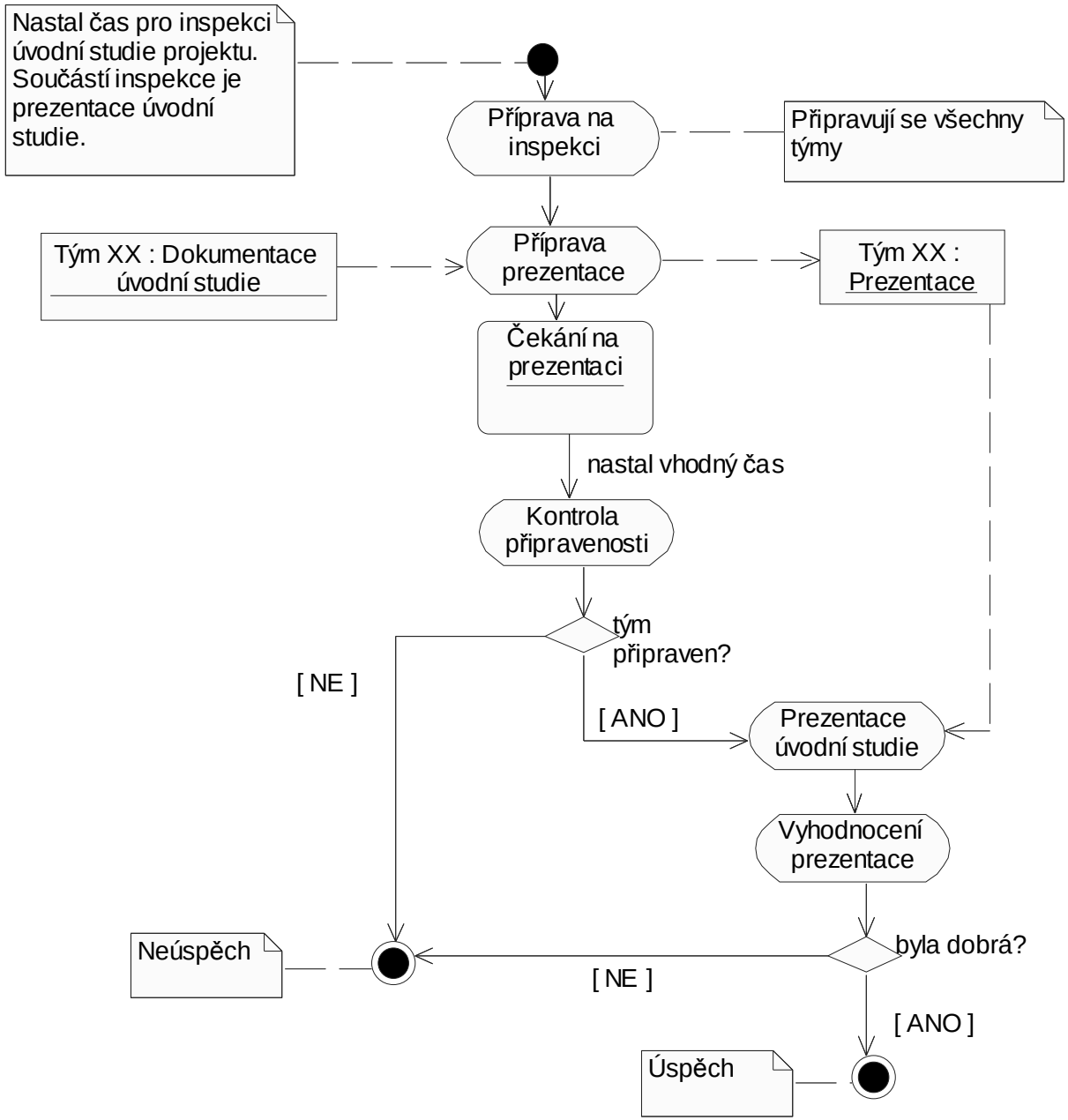
- Jen stručná informace, bez zbytečných podrobností (ty si každý může nalézt v úvodní studii, která je k dispozici na stránce projektu).


http://cs.felk.cvut.cz/~marekp3/si/images/use_case.gif - Microsoft Internet Explorer
Soubor Úpravy Zobrazit Oblíbené Nástroje Nápověda
Zpět Vpřed Zastavit Aktualizovat Domů Hledat Oblíbené Historie Pošta Tisk Upravit Diskuse
Adresa http://cs.felk.cvut.cz/~marekp3/si/images/use_case.gif
Přejít Links
Diagram pouziti pro CMFW server
CMFW server
Tvorba reportu
vyhodnocuje
Kontrola bezpecnosti
<<include>>
Administrace
spravuje
Administrator
<<include>>
Vytvareni profilu
Kontrola bezpecnosti jednou za urcity casovy usek, vyhodnocuje data a kritické udalosti a doporucuje zmeny v administ prostredi.
Administrator spravuje uzivatele a pravidla definujici profil, tedy sedu nastaveni a konfiguraci , pro CMFW klienty. Je podrizeny kontrole bezpecnosti.
<<include>>
Pravidla pro udalosti
vytvari
Programator
<<extend>>
Anivirus
Zpracovani udalosti
HTTPS textova komunikace
Programator rozsiruje standardni funkcnost serveru, definici novych pravidel pro udalosti. Programove rozhrani (API) je k dispozici v jazyku Java.
CMFW klient
SSL binarni komunikace
Rizeni komunikace
Firewall
Hotovo
Síť Internet
Start
Prozkoumává se - D:\VYU...
Centrální správa firewallů - ...
Microsoft PowerPoint - [si1...
http://cs.felk.cvut.cz...
14:12

Prezentace úvodní studie slouží zejména pro zviditelnění projektu a jako podklad pro rozhodování o financování projektu

hranice produktu – náklady - výnosy


Welcome to Adobe GoLive 6 - Microsoft Internet Explorer
Soubor Úpravy Zobrazit Oblíbené Nástroje Nápověda
Zpět Vpřed Zastavit Aktualizovat Domů Hledat Oblíbené Historie Pošta Tisk Upravit Diskuse
Adresa https://service.felk.cvut.cz/courses/36SI/prj/36SI12/frame.html
Výnosy?
HospIS
Informační systém chirurgického oddělení
Úvodní studie
Deklarace záměr
Odborný článek
Rozpočet
Kontextový diagram
Plán testů
Harmonogram
Analýza
Clenove tymu
Pro náš projekt neočekáváme příliš velké změny za provozu, takže dosadíme koeficient Z = 10%. Vychází nám tedy náklady na údržbu:
$M = 0{,}1 \cdot 66{,}50 = 6{,}65 MM$ na rok
Což odpovídá přibližně jednomu člověku zaměstnanému na poloviční úvazek. Což při platu 12,500,- Kč měsíčně (25 000,- hrubého) odpovídá nákladům ve výši 300000,- Kč.
HW výdaje
Vzhledem k tomu, že součástí našeho systému není vybavování Počítáme s tím, že celý systém poběží na stávajících pracovních stanicích Nemocnice a využijeme i její existující síť. Je tedy potřeba pouze pořídit nový server a na něj databázový stroj. Server pořídíme od firmy Fujitsu Siemens Computers PRIMERGY C150 s OS MS Windows XP za 60 000,- Kč. Na něm poběží databázový stroj Oracle 9i, na který počítejme náklady na licenci 300 000,- Kč.
Shrnutí
Vzhledem k tomu, že nemusíme investovat do vývojových nástrojů ani do vybavení pro vývojovou firmu, tak nám odpadá tento druh nákladů. Dále předpokládáme, že systém poběží na stávající nemocniční síti a tedy nemusíme do nákladů na vývoj produktu počítat náklady na pořízení HW vybavení nemocnice.
Celý vývoj projektu bude tedy trvat 12 měsíců a bude na něm pracovat 6 lidí. Náklady na jeho vývoj budou činit 3 600 000,- Kč. Na jeho údržbu bude stačit jeden člověk zaměstnaný na poloviční úvazek a náklady na tuto údržbu budou činit 300 000,- Kč. Na server, na kterém poběží databáze, počítáme výdaje 360 000,- Kč. Celkové náklady na projekt tedy činí 3 960 000,- Kč a na jeho údržbu je třeba vydat 300 000,- Kč ročně.
Síť Internet
Start Prozkoumává ... Welcome to... Microsoft Pow... http://cs.felk.c... Together 6 -- p... Microsoft Phot... 14:31



Prezentace úvodní studie

Kontrolní body pro úvod.studii

- Má projekt **kvalitní stránku**? Jsou zde uvedeni řešitelé a jejich e-maily? Je zde možnost poslat zprávu všem? Je na stránkách projektu uvedeno logo a název projektu? Je na stránkách projektu projektový deník?
- Je zpracována **deklarace záměru**?
- Je na stránkách projektu **odborný článek**?
- Existuje **katalog požadavků**? Kvalita funkčních požadavků, kvalita nefunkčních požadavků?
- Seznam a popis **případů užití** (USE-CASE), detailně rozpracované USE-CASE včetně scénářů?
- Je na stránkách k dispozici **harmonogram** práce týmu? Je na stránkách projektu **matice zodpovědnosti**? Jsou v úvodní studii specifikovány požadavky na HW a SW? Je v úvodní studii uveden rozpočet projektu dekomposicí (MS Project)? Je v úvodní studii uveden rozpočet projektu výpočtem (COCOMO)?
- Seznam a **popis entit systému** (= slovníček pojmů)? **Model entit systému** (= diagram)?
- Je k dispozici **prezentace** úvodní studie projektu?

Příklad

Prezentace úvodní studie projektu SPU

The End

Návrh

JAK se to udělá

Návaznosti

- Dříve:
  - Analýza
- Dnes:
  - Návrh
- Příště:
  - Návrhové vzory

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, přístupové mechanismy k nim
- distribuce programových modulů, komunikační mechanismy
- typy a formy výstupů
- uživatelské rozhraní
- vývojové prostředí

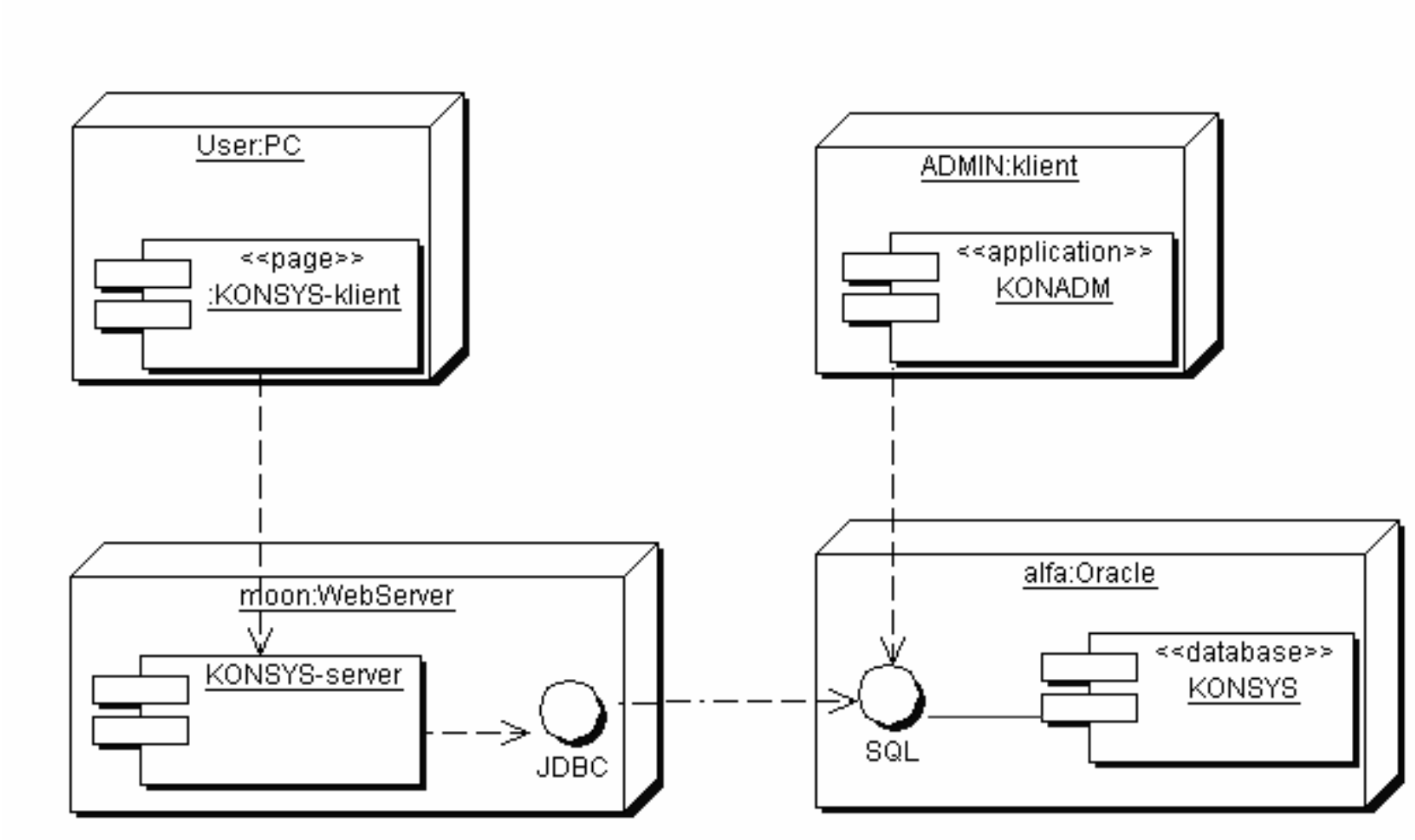
Vstupy pro návrh

- Analytický model chování systému

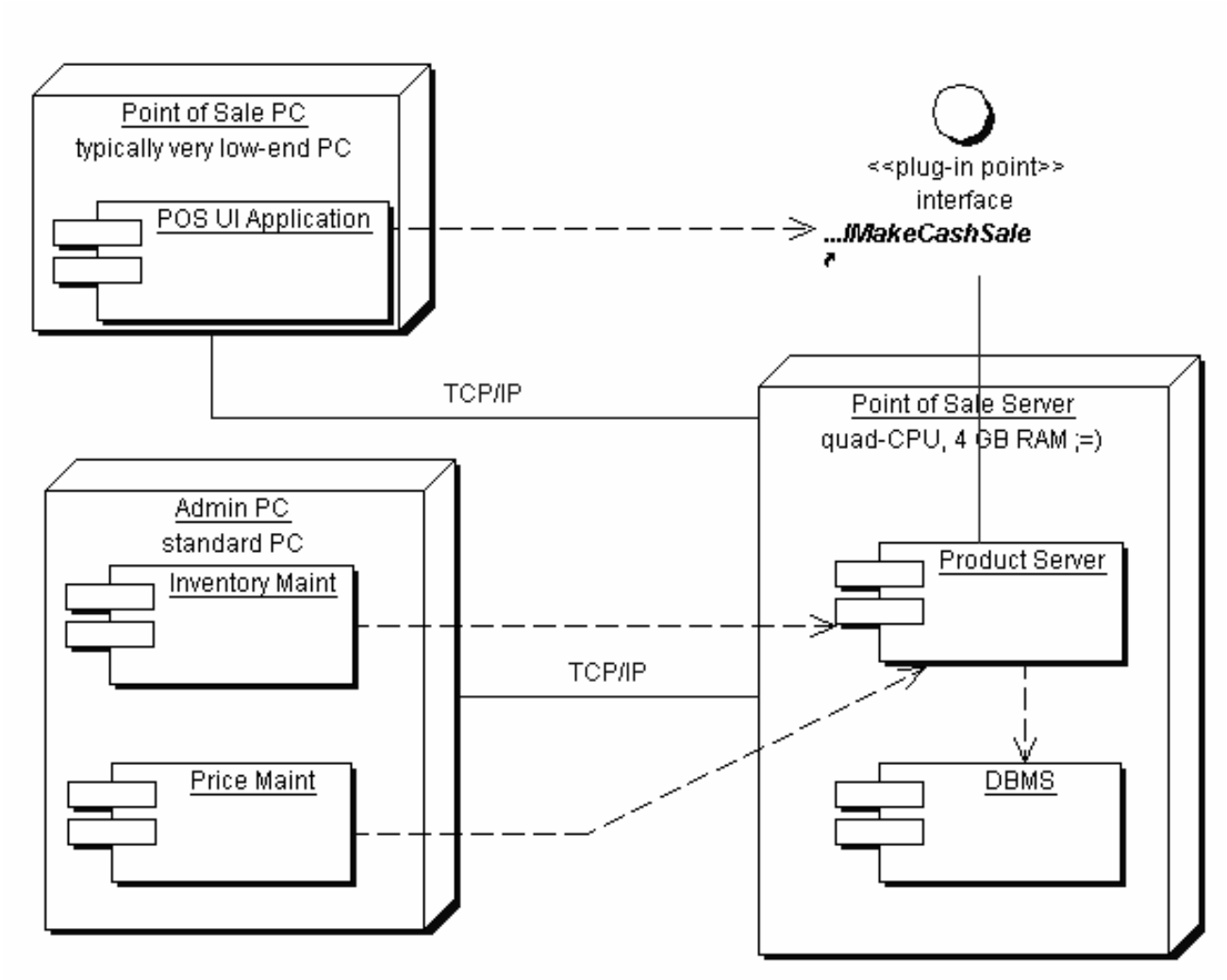
Výstupní dokumenty návrhu

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

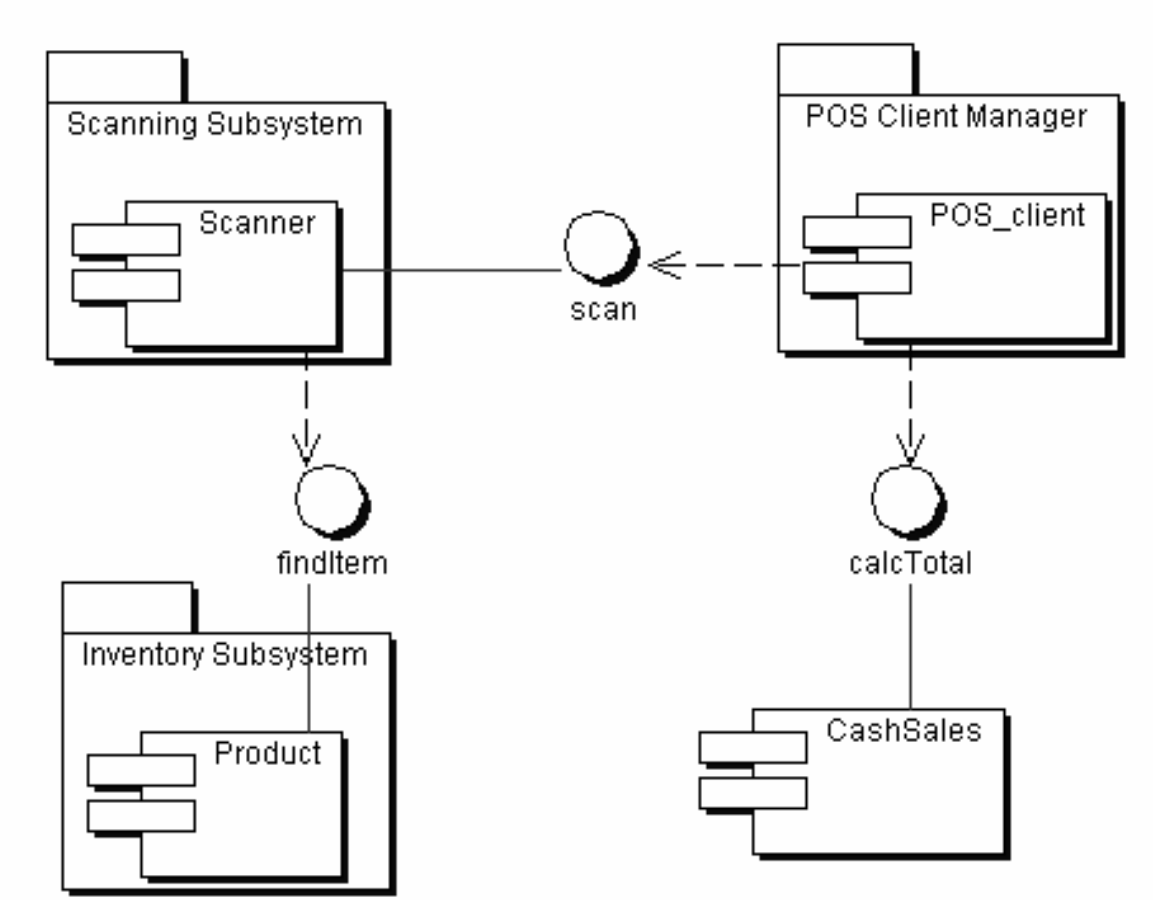
Příklad návrhu architektury



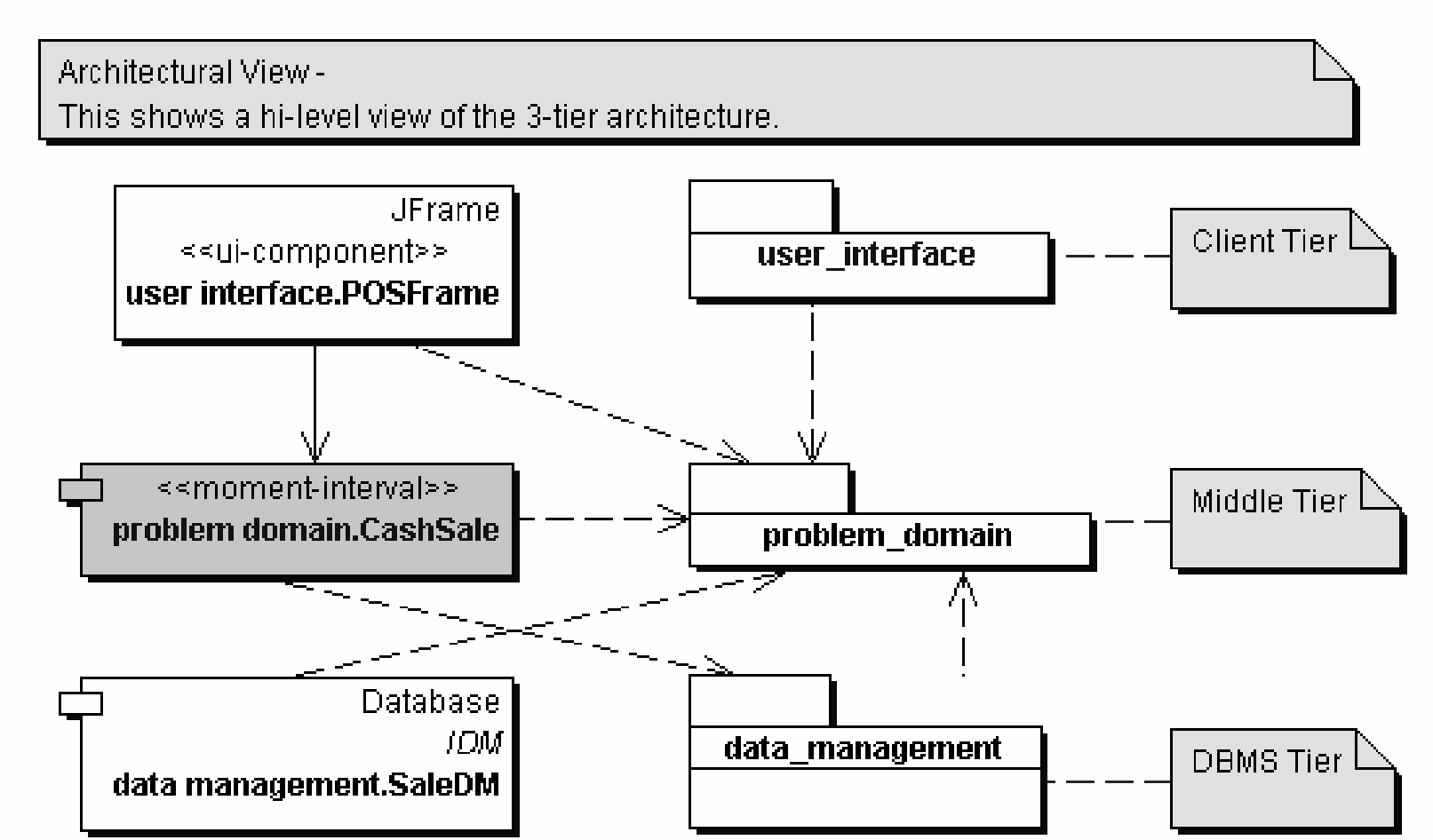
Jiný příklad: Elektronická pokladna



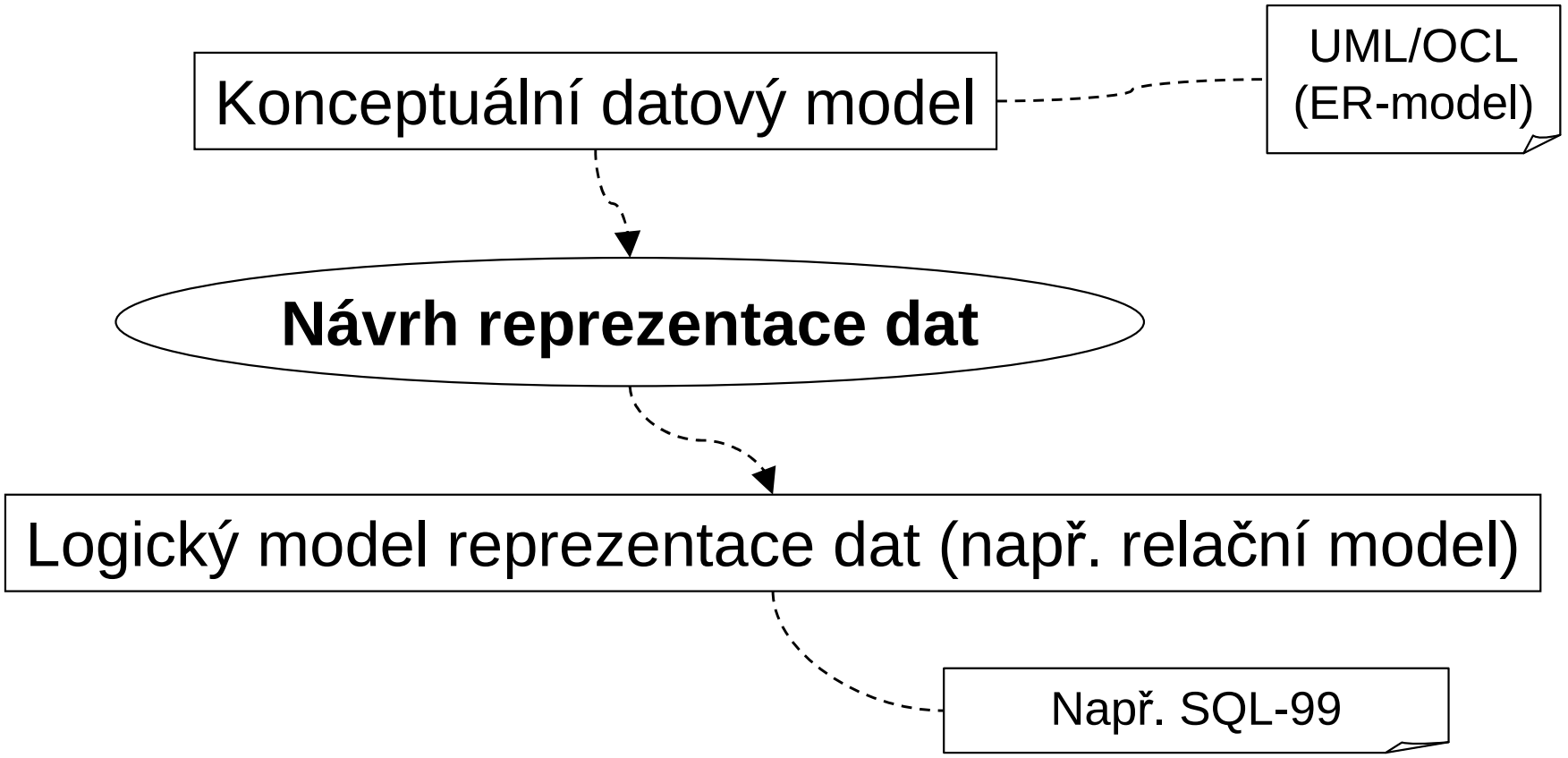
Elektronická pokladna (komponenty)



Pokladna (architektura)



Návrh reprezentace dat

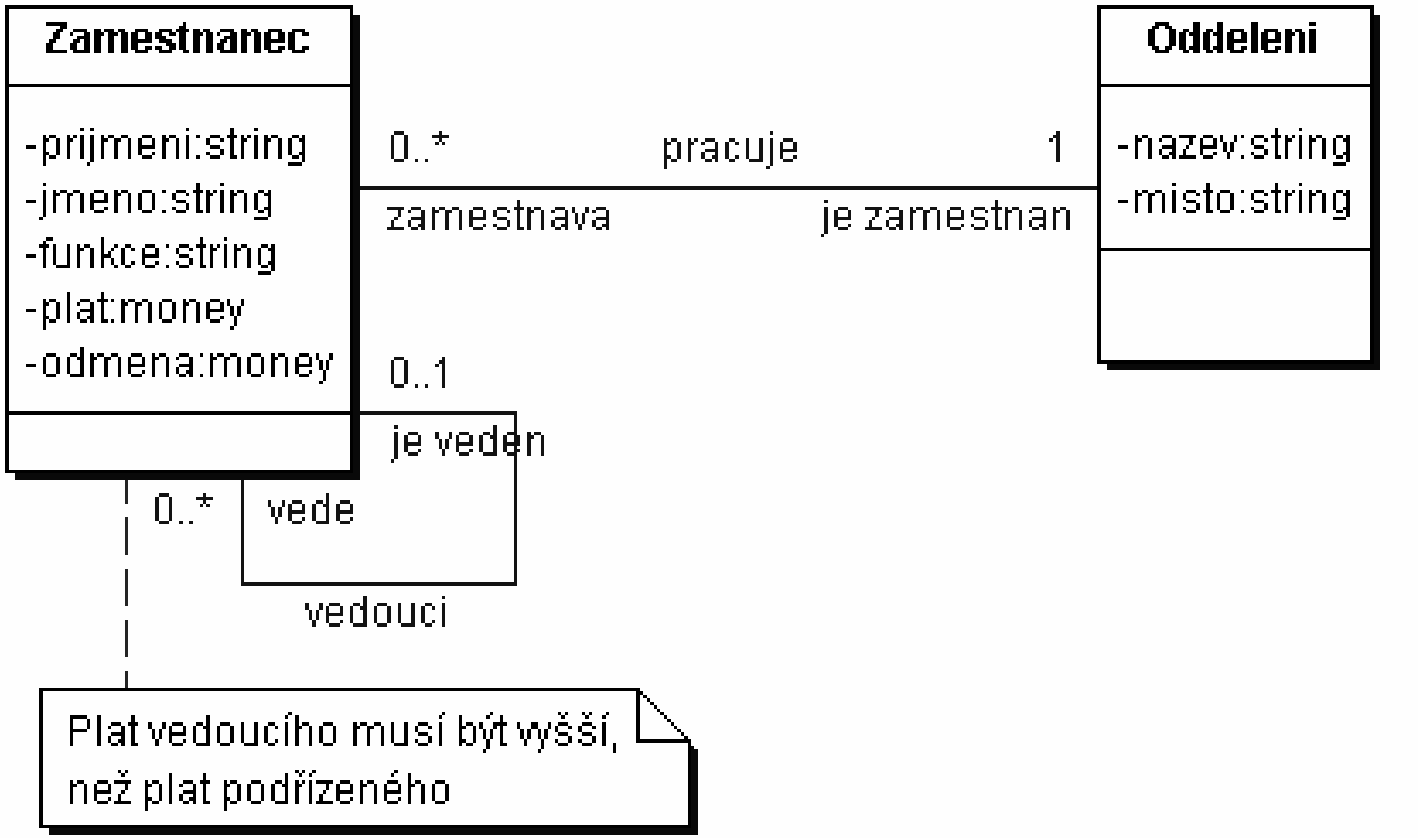


Konceptuální datový model specifikuje

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

Pozn: Model tříd popisuje i operace, ale ty se často k modelu připojí až v návrhu

Příklad: 1.verze dat.modelu



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).

Příklad

„Plat vedoucího musí být vyšší, než plat jeho podřízených“.

OCL:

```
context ZAMESTNANEC inv C1 :
  forAll(ZAMESTNANEC e1,e2 | e1 = e2.vedouci
    implies e1.plat > e2.plat)
```

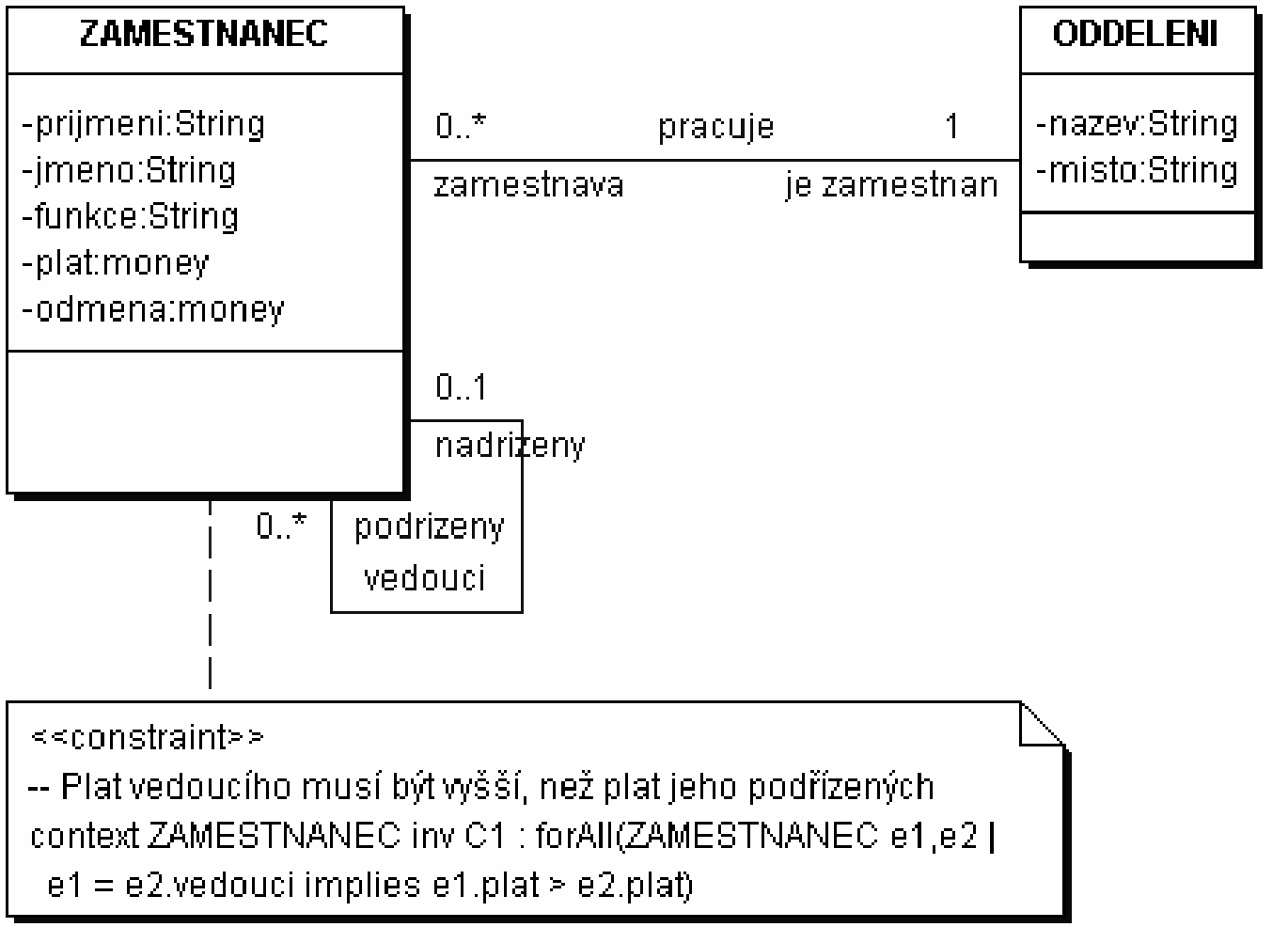
Ekvivalentní logická formule:

(C1) $\forall$**e1,e2** $\in$ **ZAMESTNANEC : e1 = e2.vedouci**

$\Rightarrow$

e1.plat > e2.plat

Příklad: 2.verze dat.modelu



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

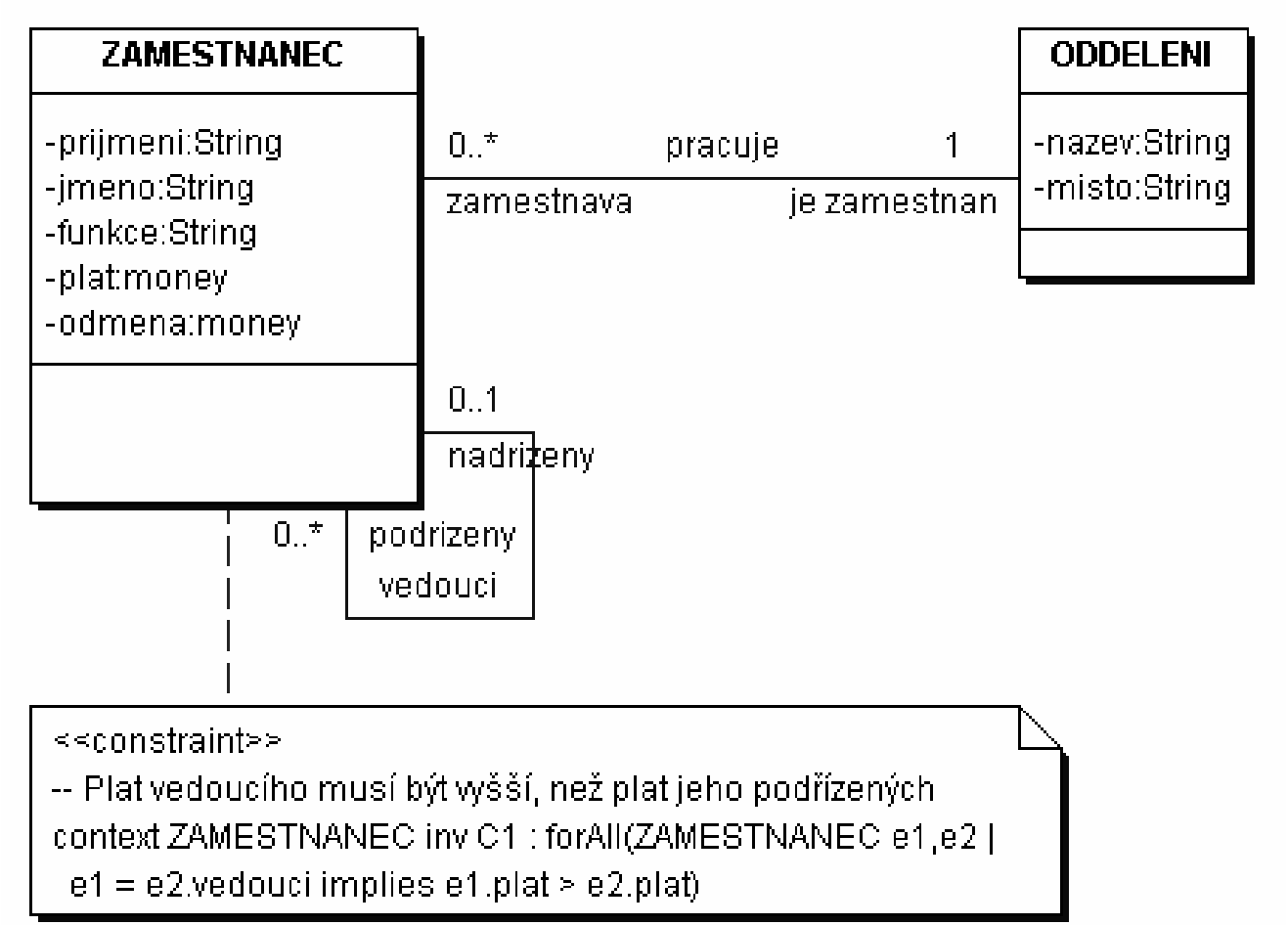
Postup návrhu

- 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).

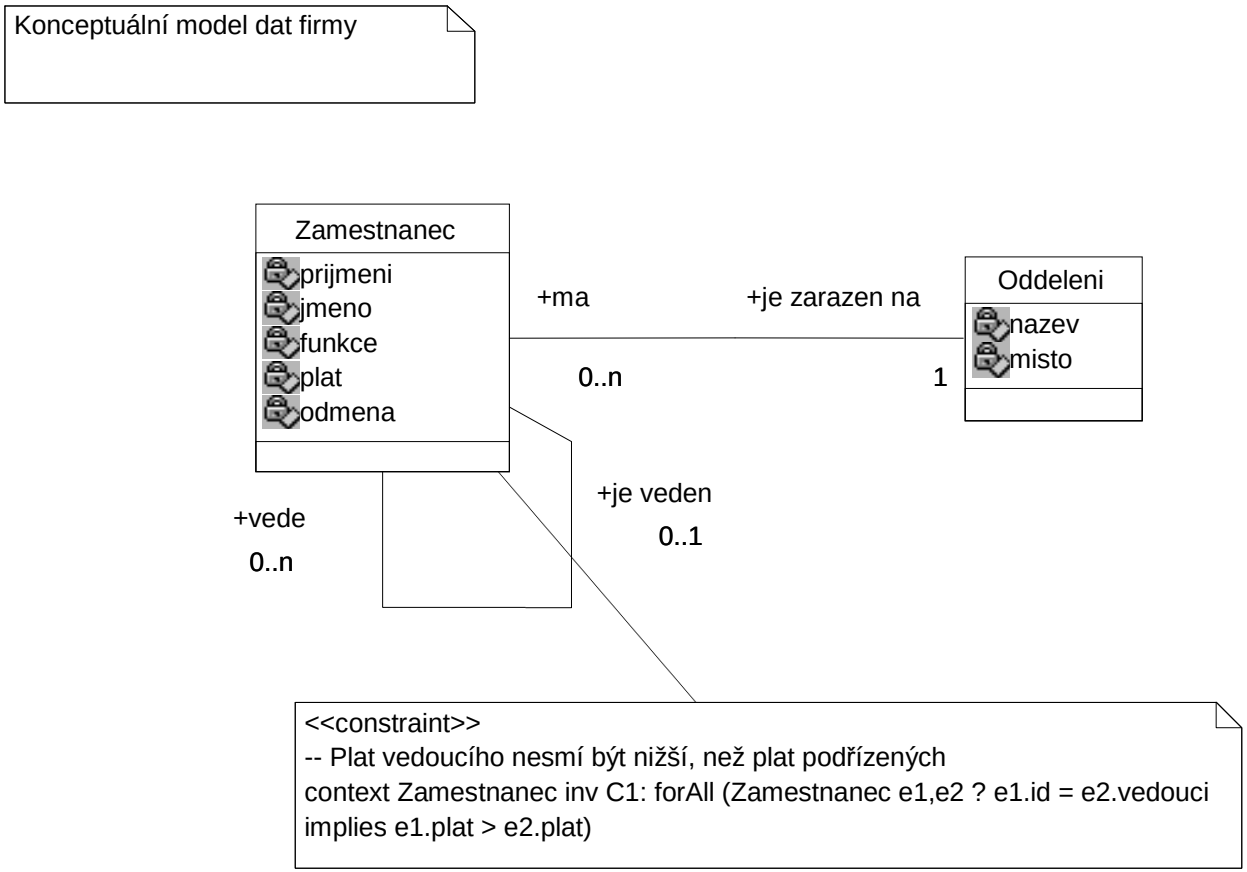
Návrh reprezentace (pokr.)

- 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.

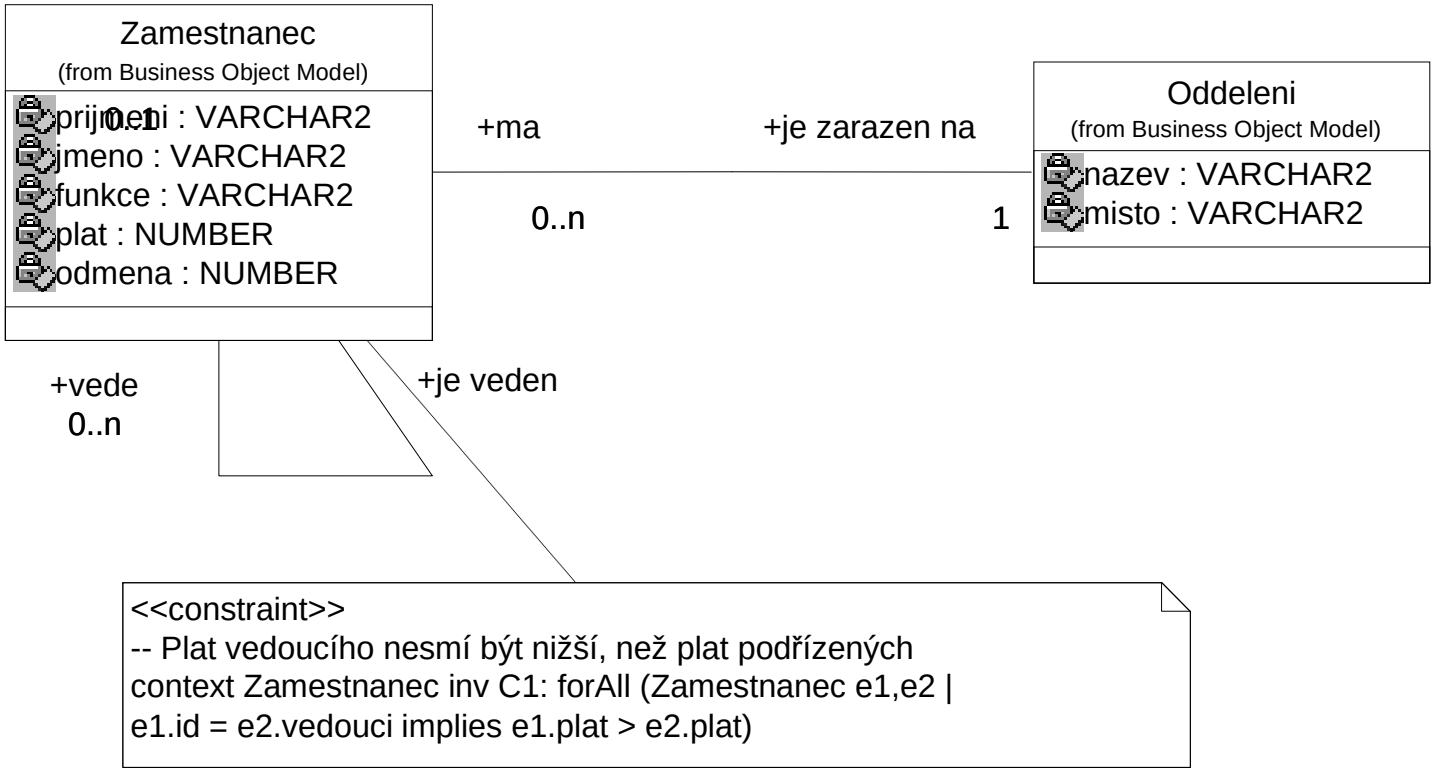
Návrh reprezentace (pokr.)

- 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“).

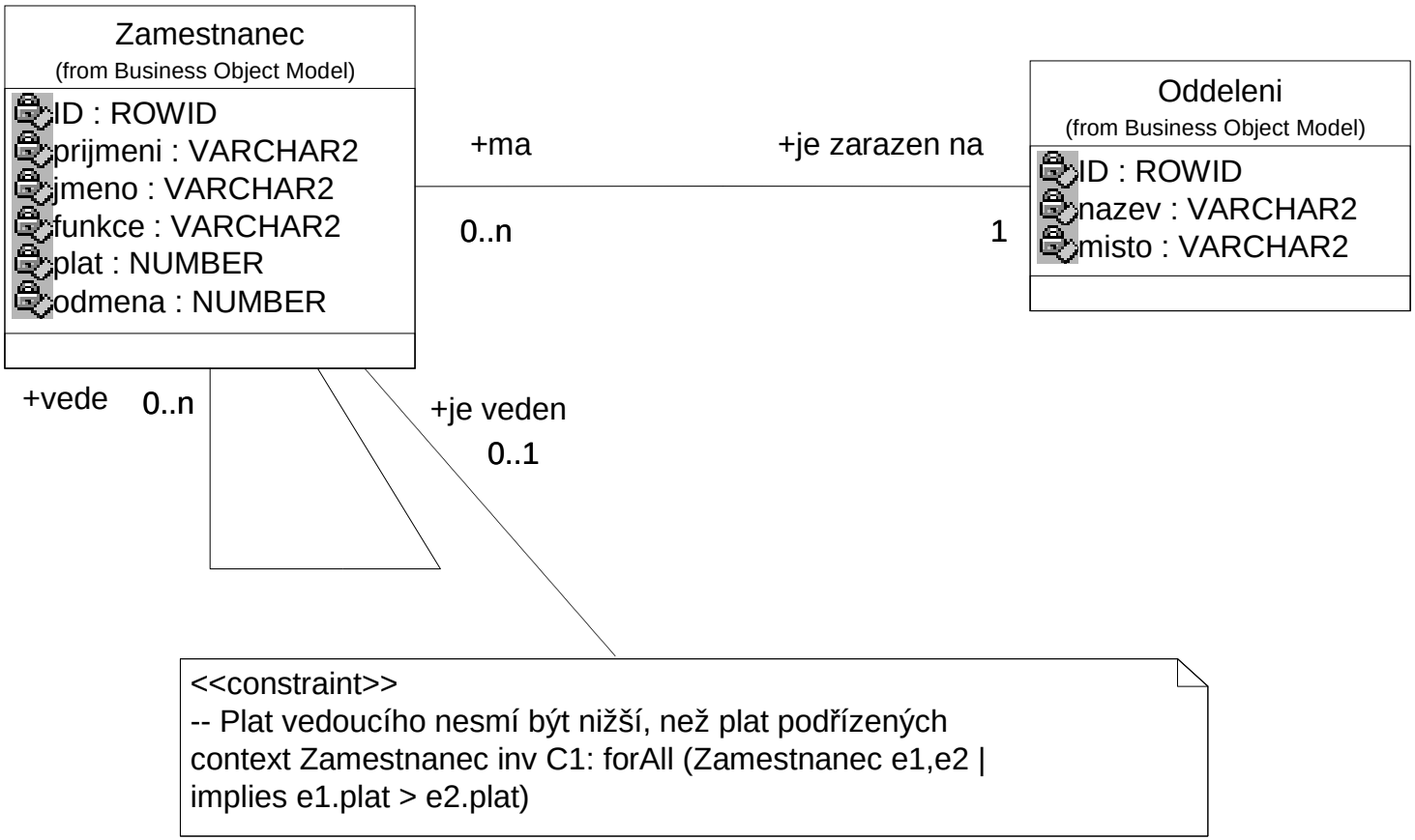
Konceptuální model



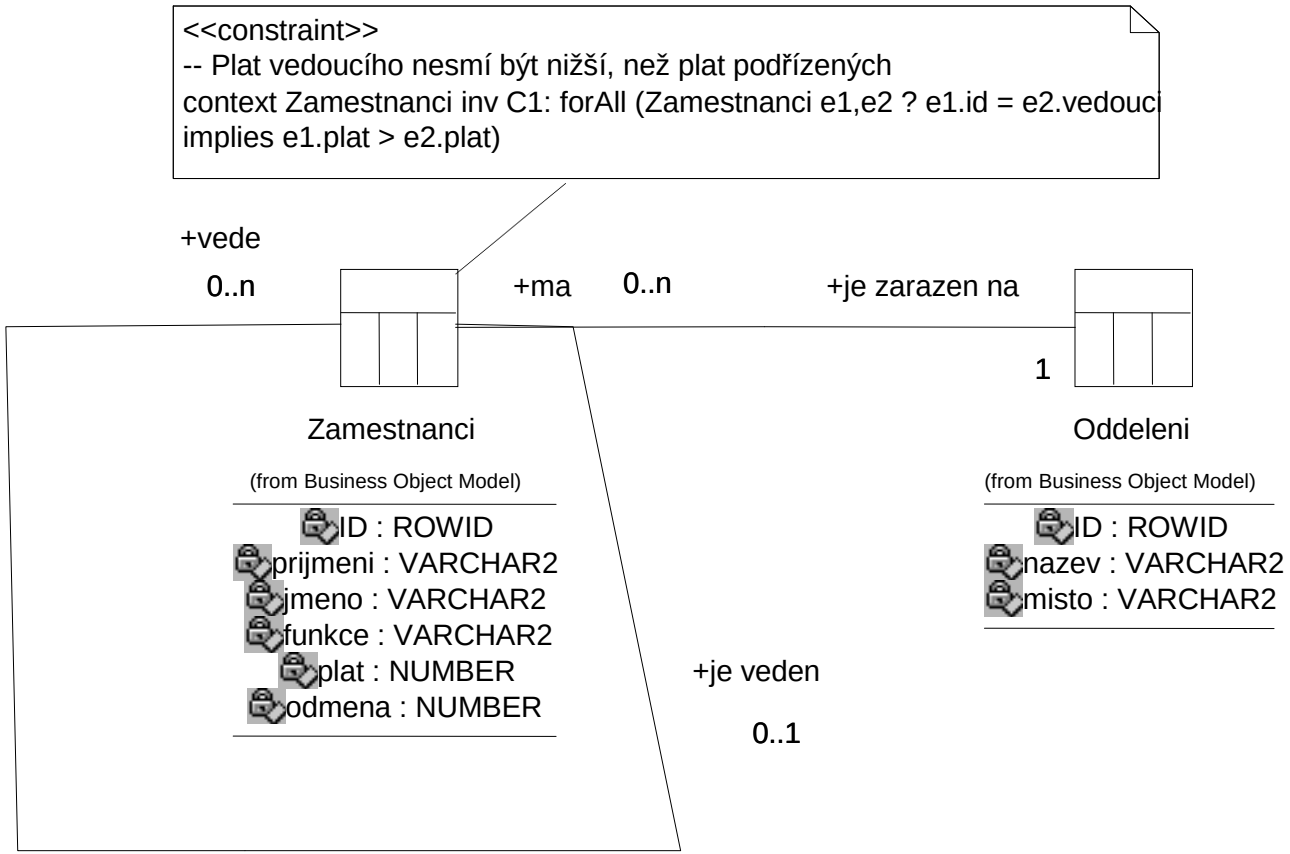
Jména a domény



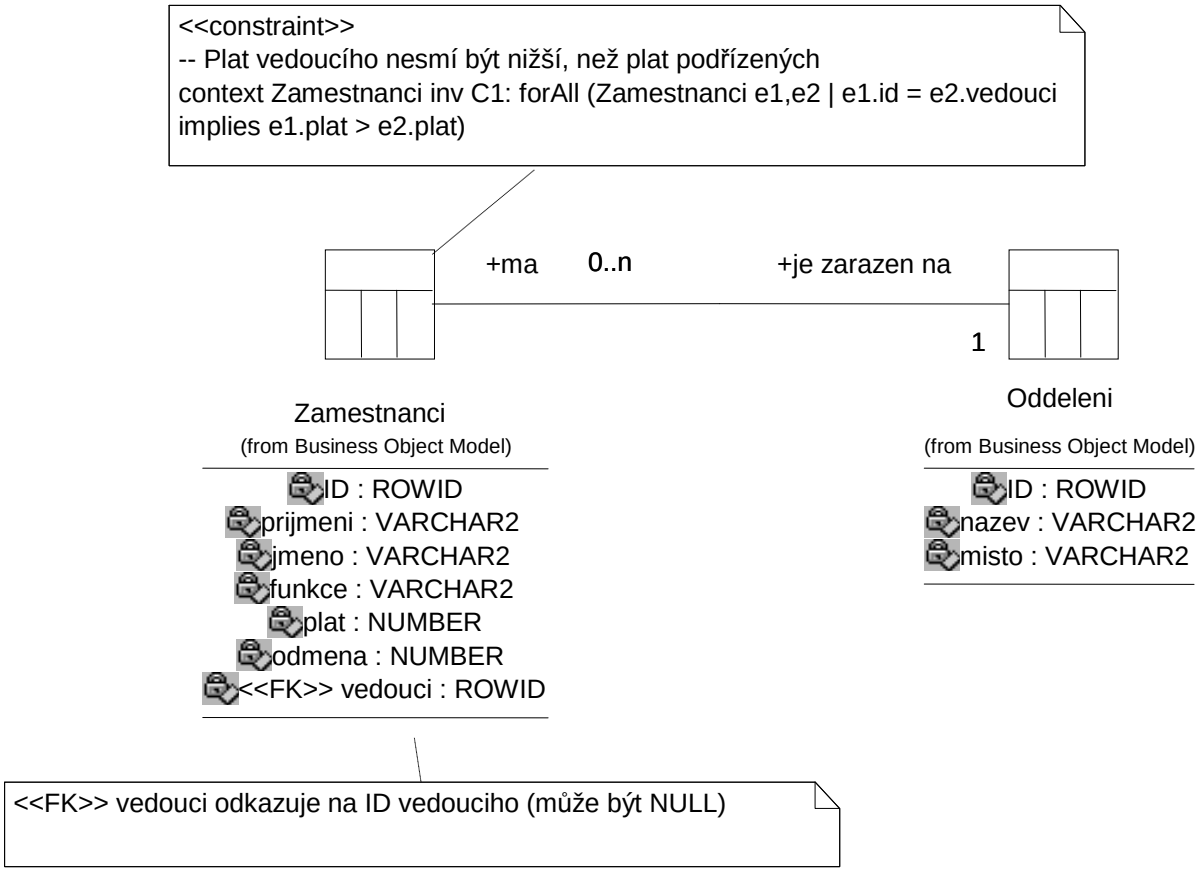
Primární identifikace



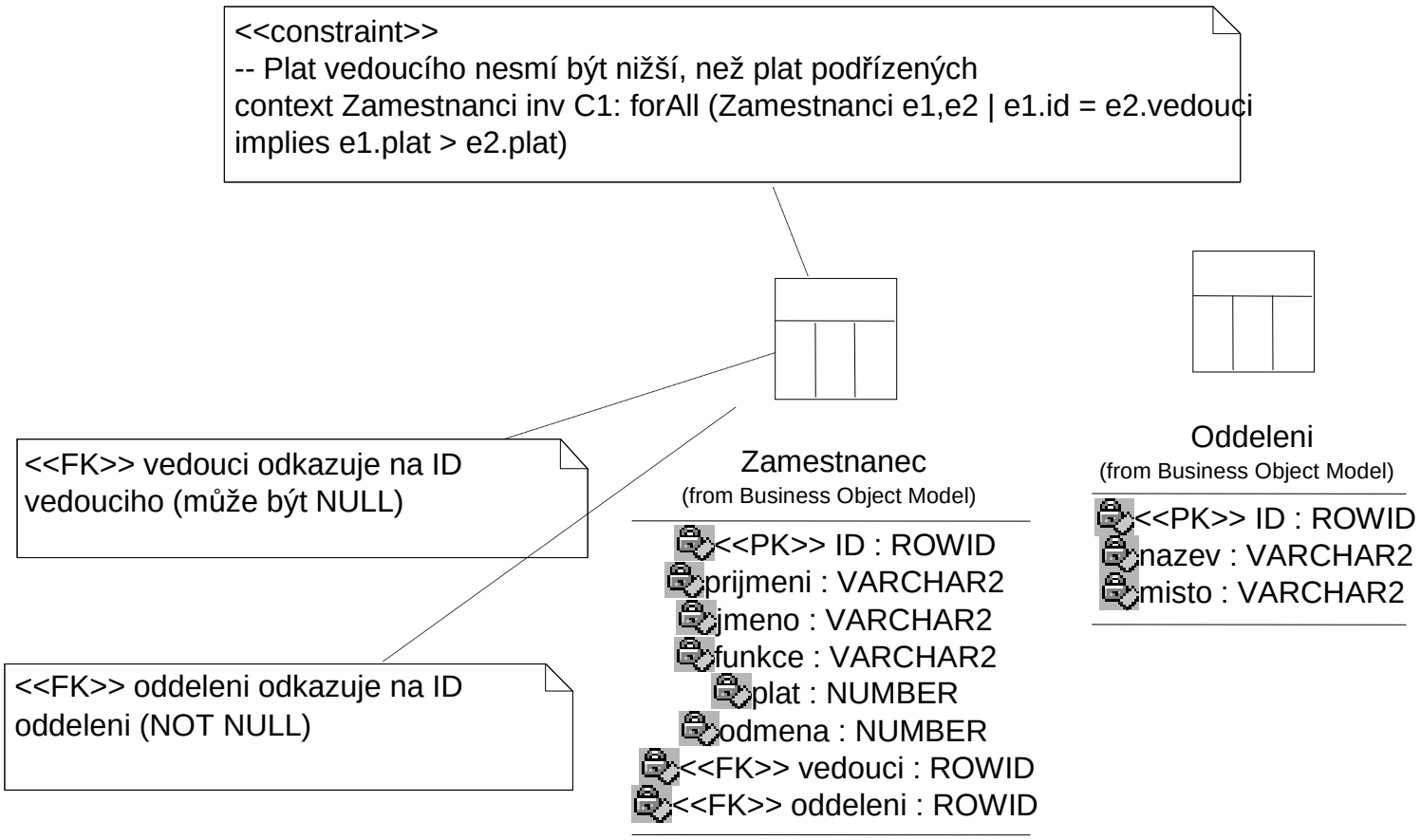
Vytvoříme tabulky



Cizí klíč pro “vedoucího”



Cizí klíč pro “zaměstnance”



Návrh reprezentace integritních omezení

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

Příklad: "forAll"

```
context Zamestnanci inv C1: forAll
  (Zamestnanci e1,e2 | e1.id =
  e2.vedouci implies e1.plat > e2.plat)
```

⇒

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

Odvozené SQL I. (tabulka Oddeleni)

```
create table Oddeleni (
  ID ROWID primary key,
  nazev VARCHAR2(20),
  misto VARCHAR2(20)
);
```

Odvozené SQL II. (table EMP)

```
create table Zamestnanci (
  ID ROWID primary key,
  prijmeni VARCHAR2(35),
  jmeno VARCHAR2(35),
  funkce VARCHAR2(10),
  plat NUMBER(9,2),
  odmena NUMBER(9,2),
  vedouci ROWID references EMP(id),
  oddeleni ROWID not null references
  Oddeleni(ID)
);
```

Odvozené SQL III. (ostatní)

```
alter table Zamestnanci add constraint C1
check (not exists
  (select ‘X’
   from Zamestnanci e1, Zamestnanci e2
   where e1.ID = e2.vedouci
   and e2.plat > e1.plat));
```

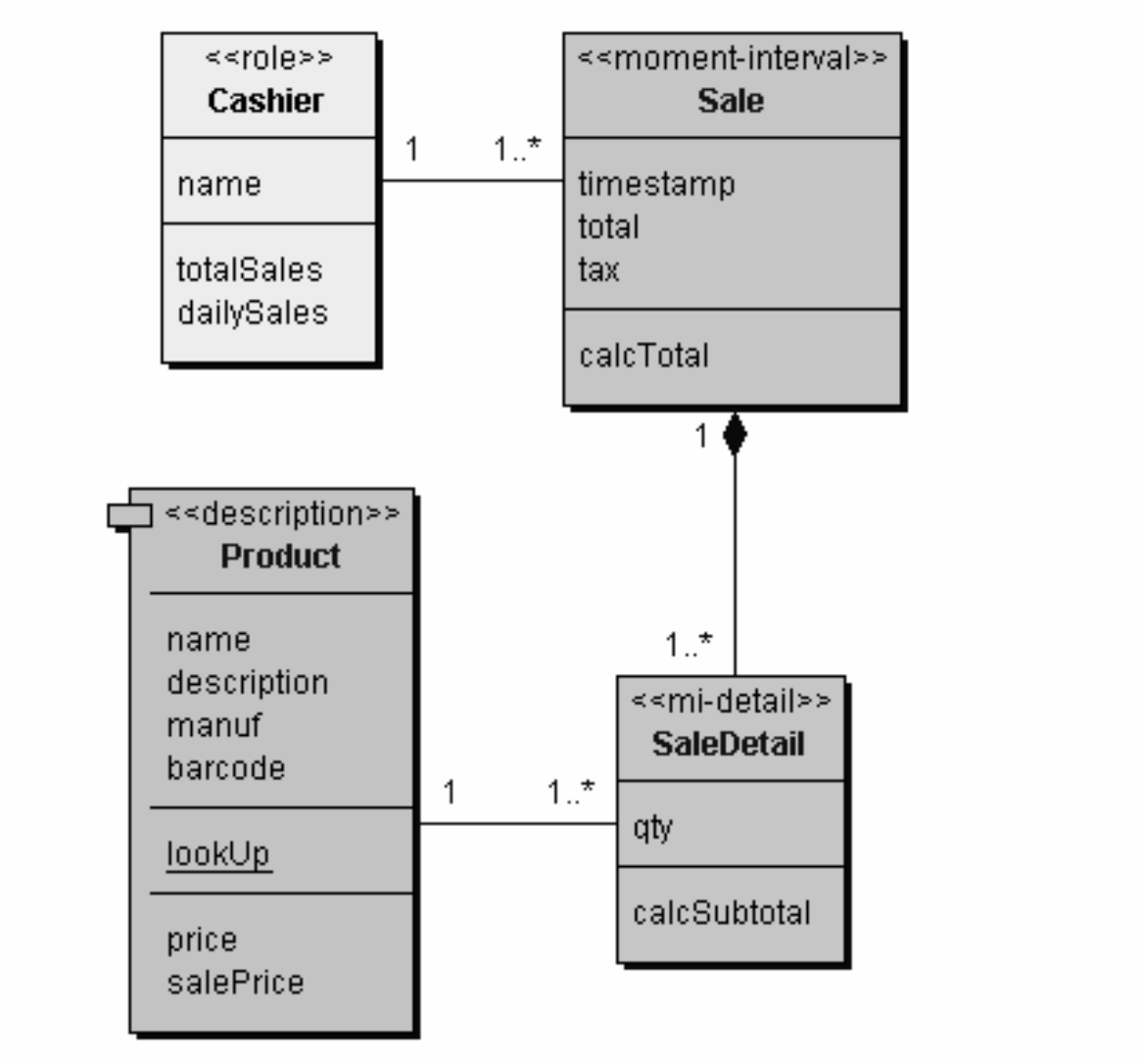
Ale problém – někde to nefunguje !!!

Zkusíme trigger !!!

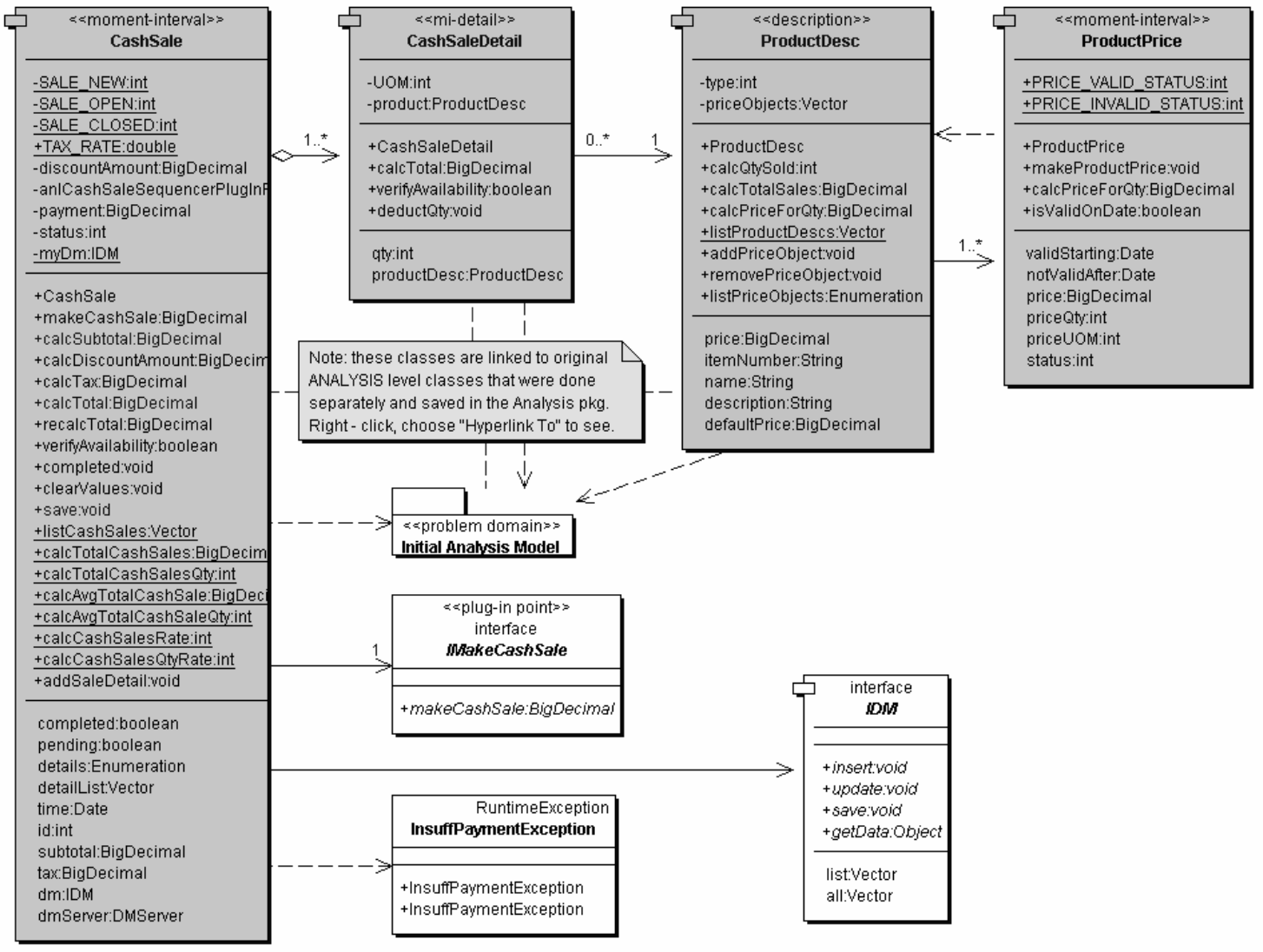
Příklad (zjednodušeno)

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

Pokladna (analytický model)

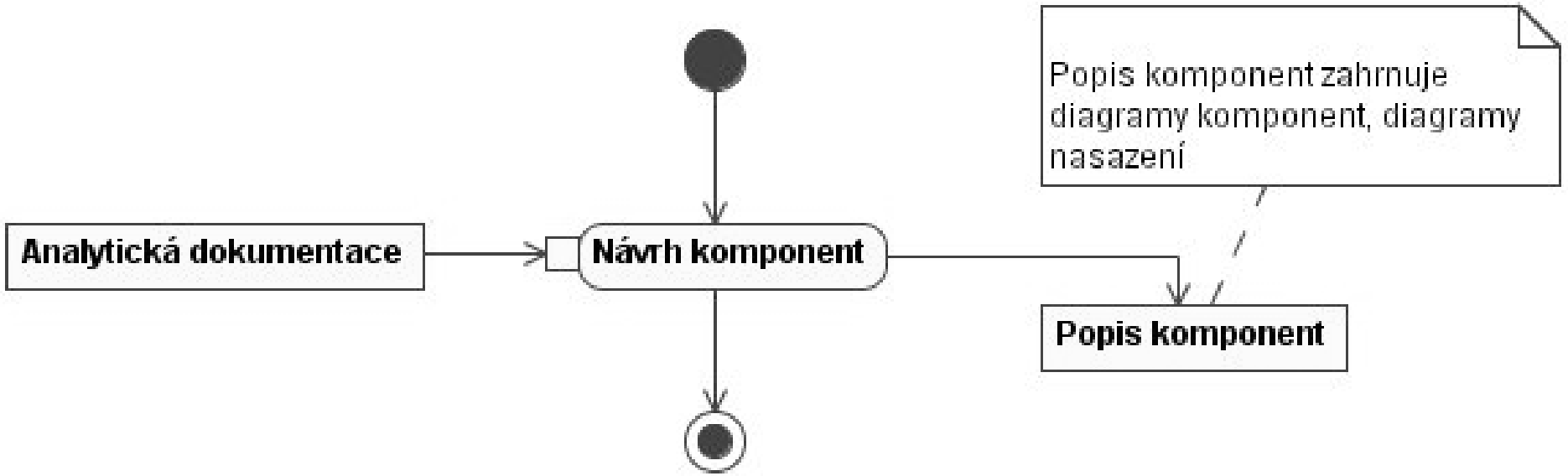


Pokladna (datový model po návrhu)



Návrh komponent

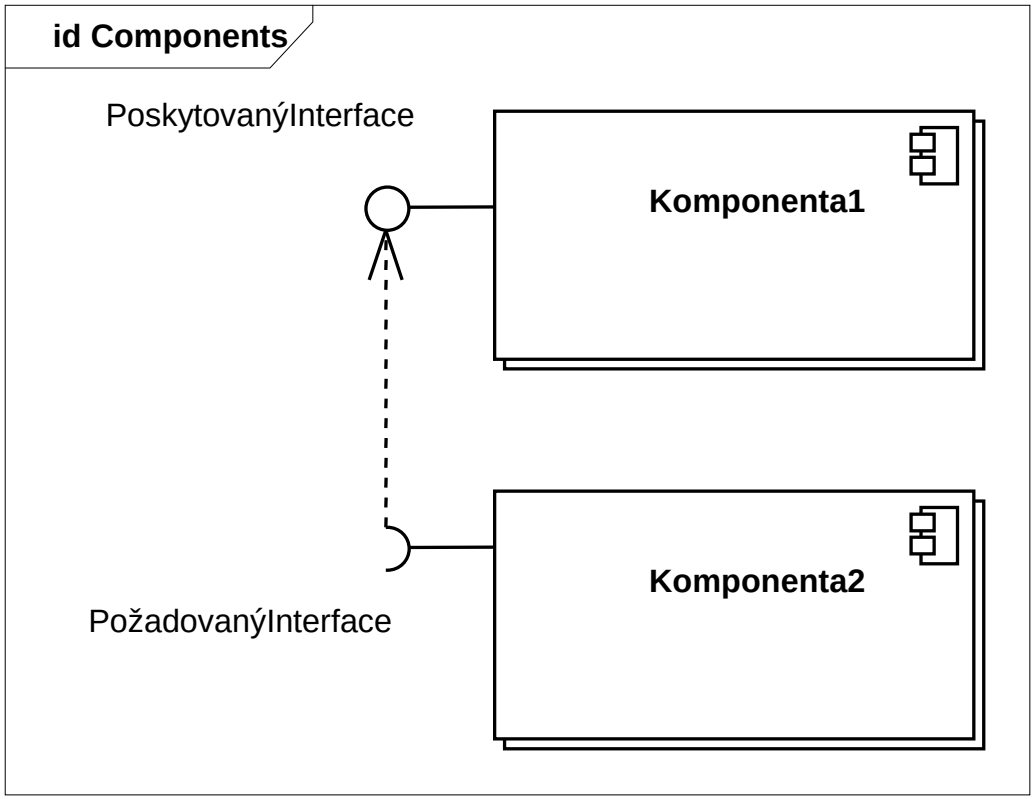
Návrh komponent



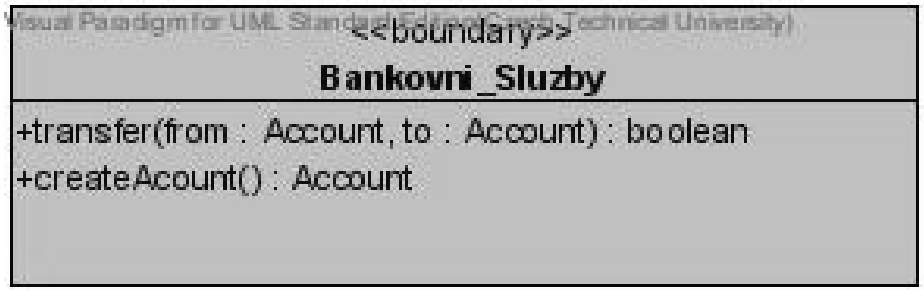
Co to je komponenta?

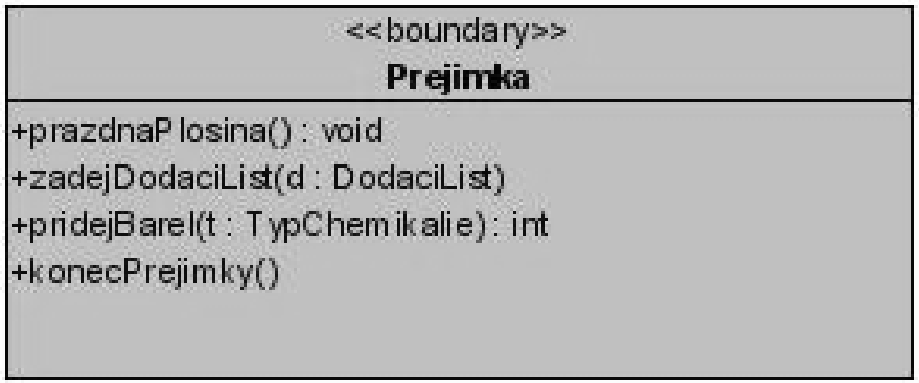
- Modulární, nasaditelná, nahraditelná a opakovaně použitelná část systému, která zapouzdřuje implementaci a zveřejňuje rozhraní.
- Komponenta je skupina objektů, které dohromady:
  - poskytují určitá rozhraní pro externí objekty
  - vyžadují určitá rozhraní od externích objektů (vyžadují součinnost)
- Na rozdíl od tříd, nemusí existovat instance komponenty (vyhýbají se nutnosti mít stav)
- Nedodává se obvykle ve zdrojovém tvaru, ale jako vykonatelný kód
- Často vyžaduje pro práci určité prostředí (např. kontejner)

Notace diagramů komponent I.



Interface vyžaduje popis





Komponenta se může skládat z dalších komponent

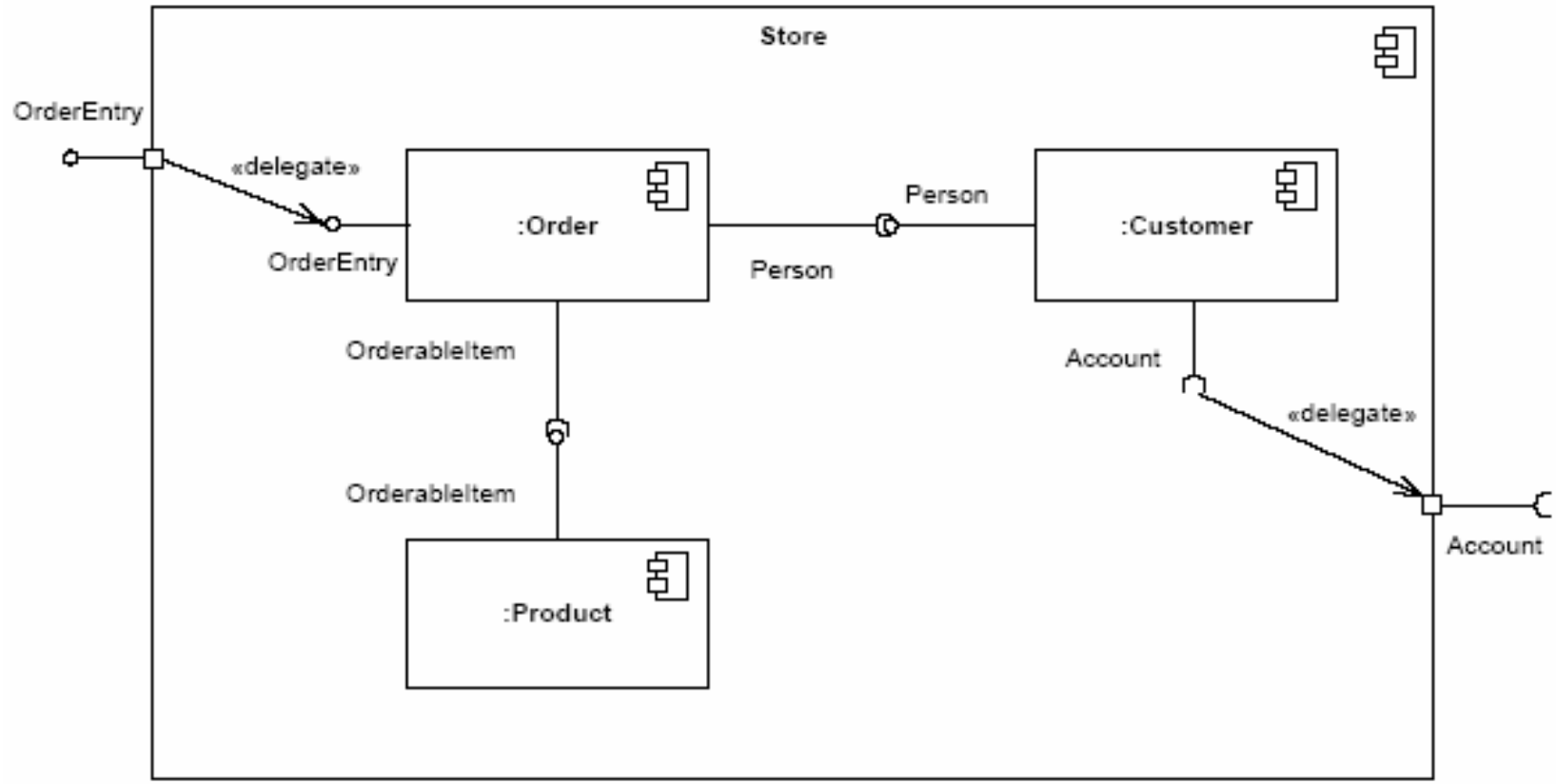
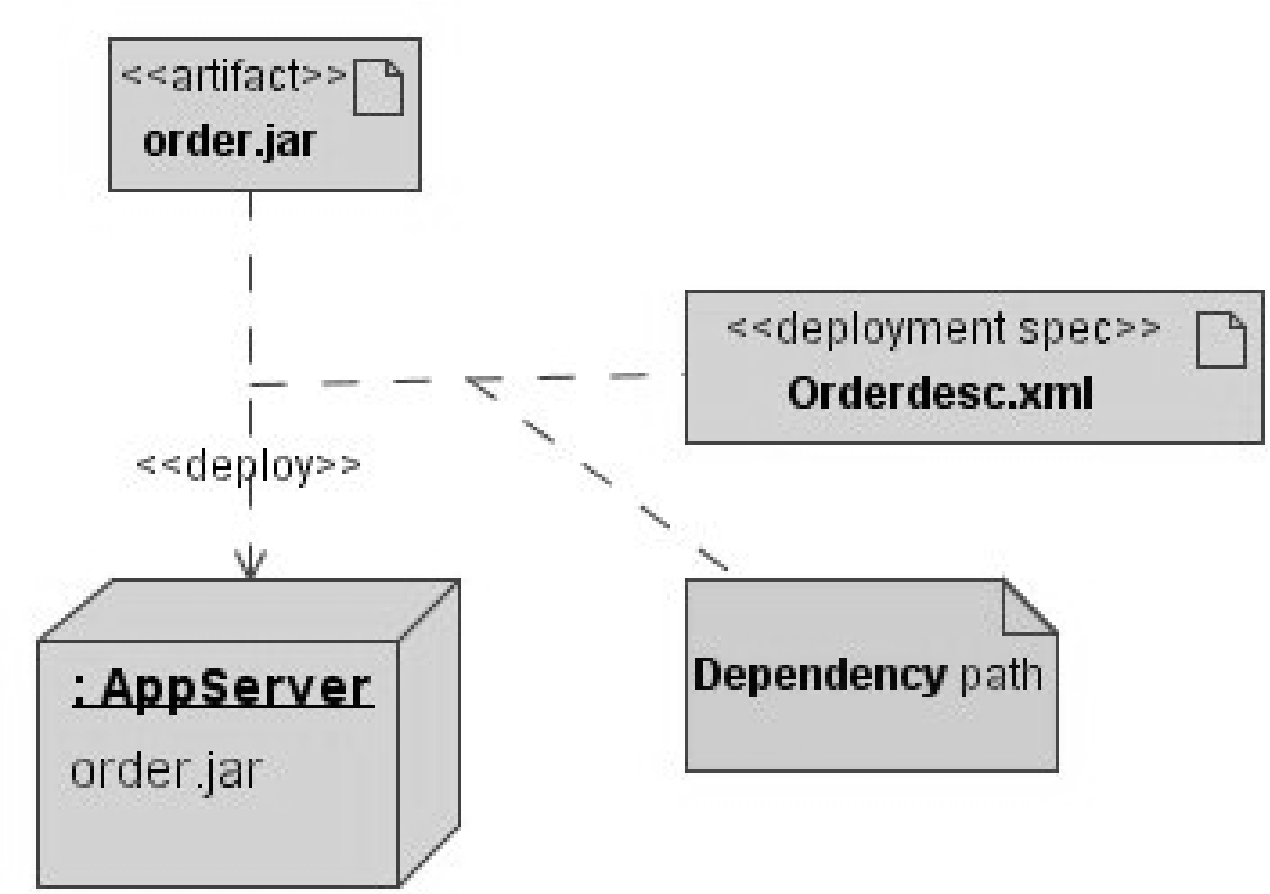
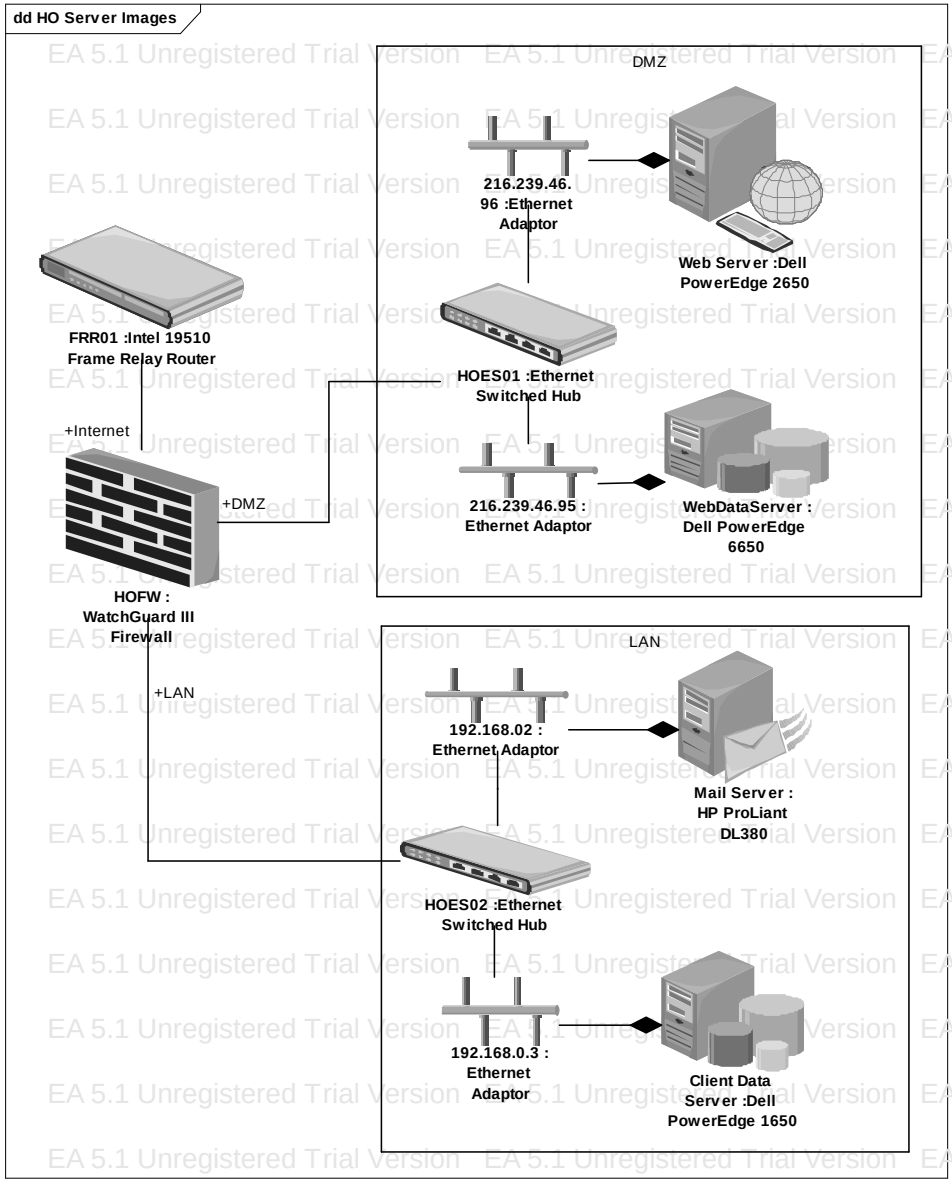


Diagram nasazení komponenty





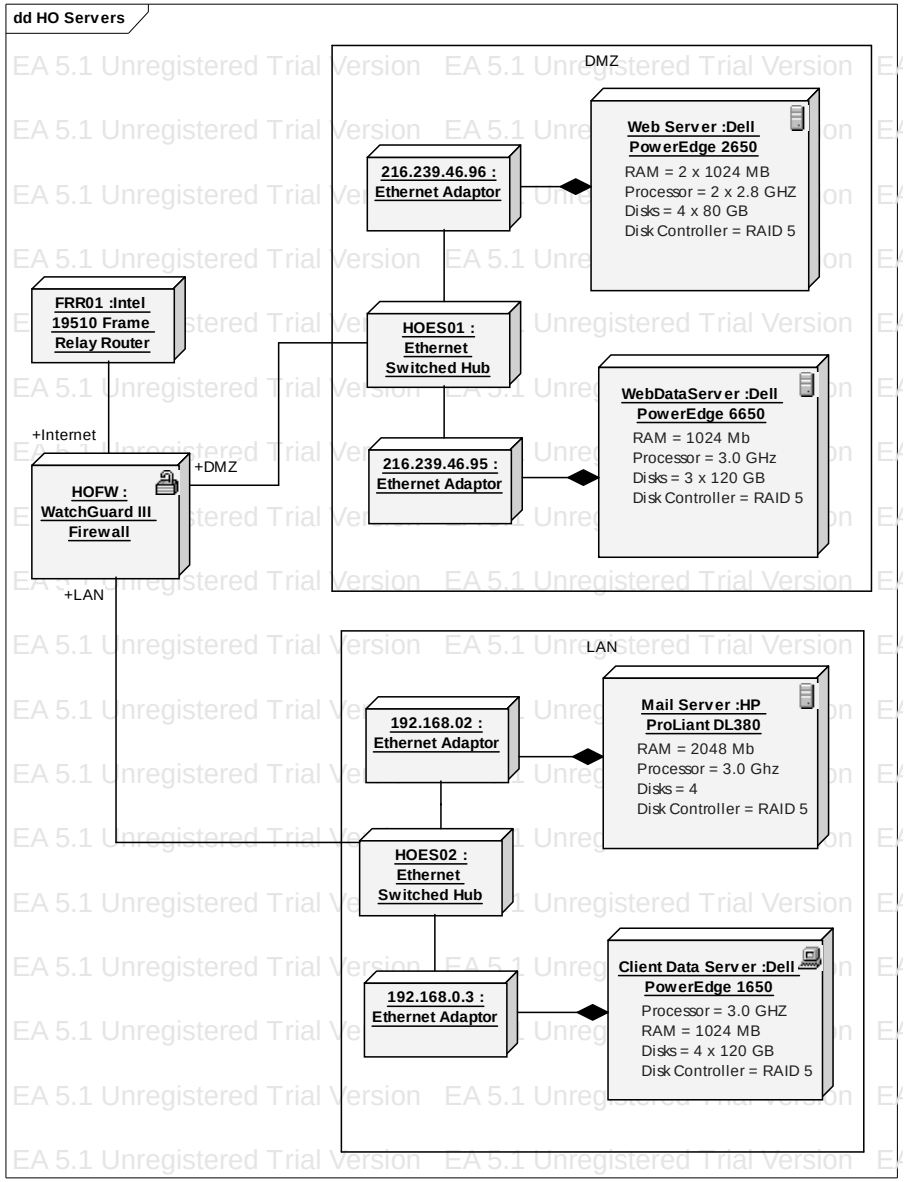
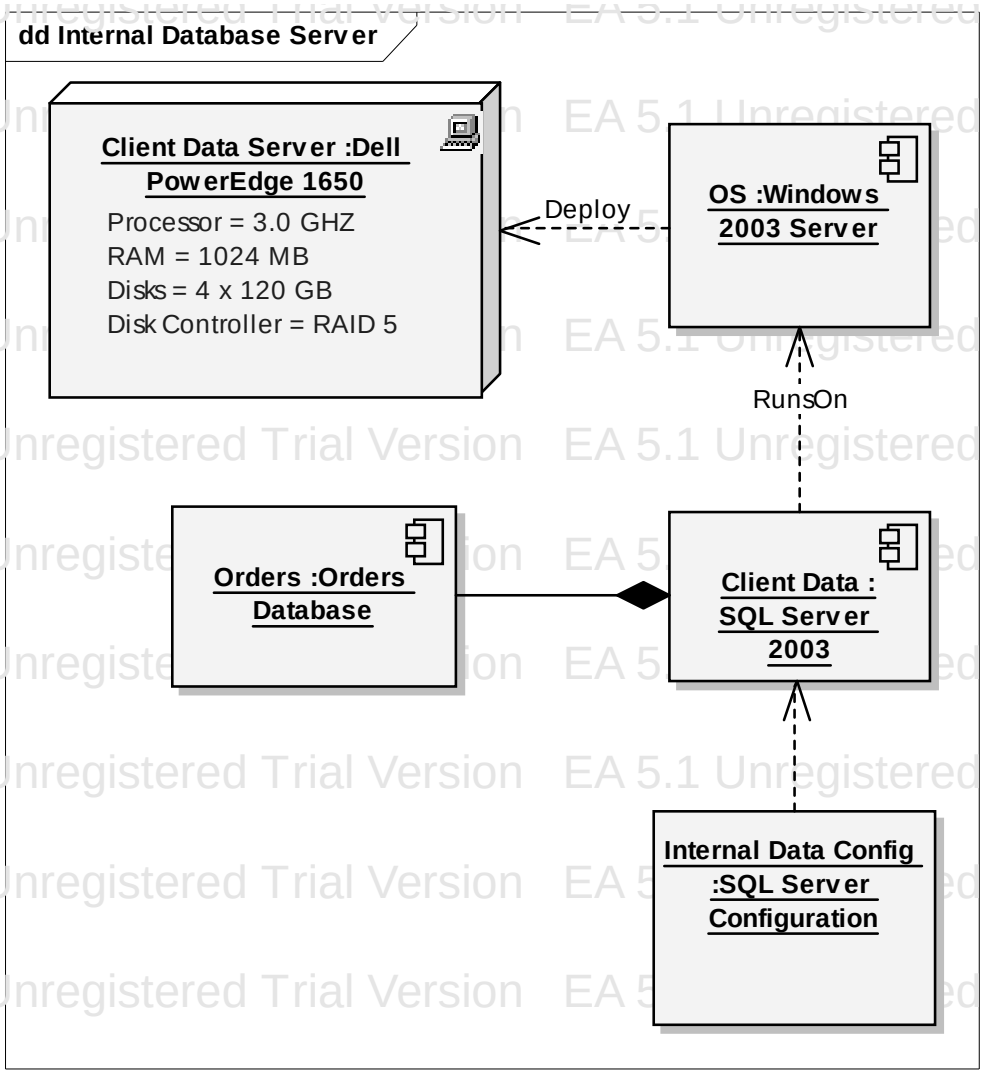
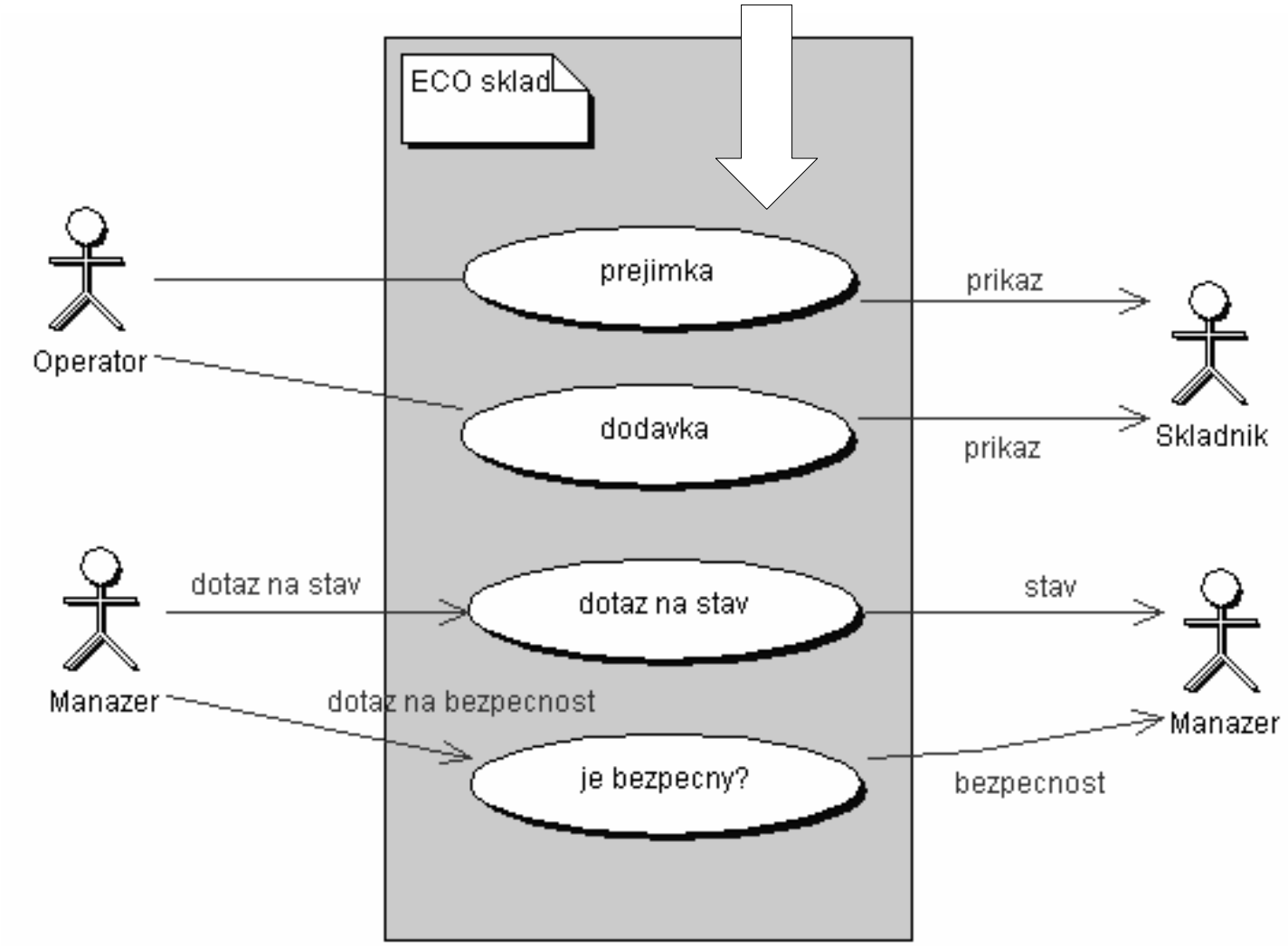


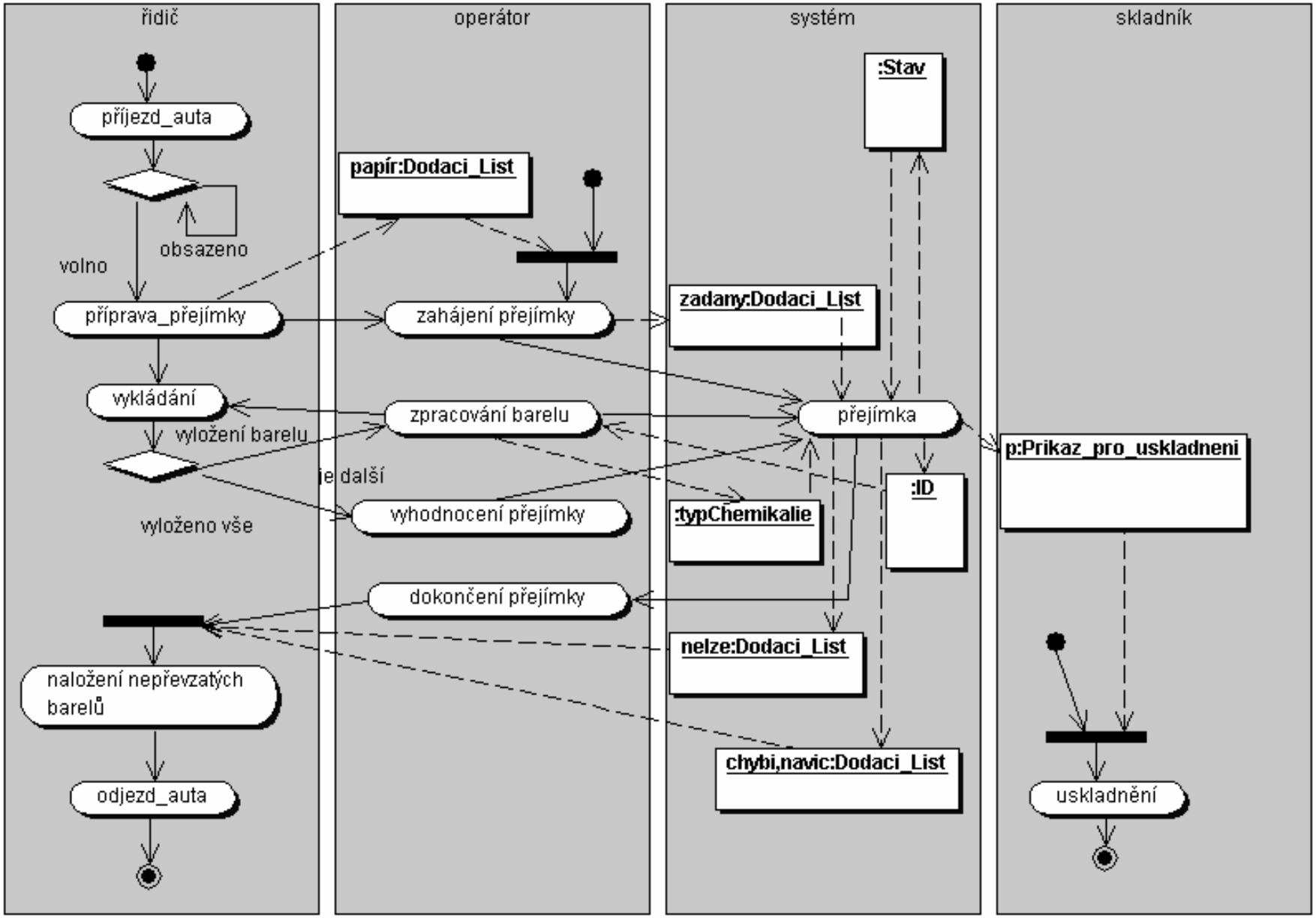
Diagram nasazení komponenty DB



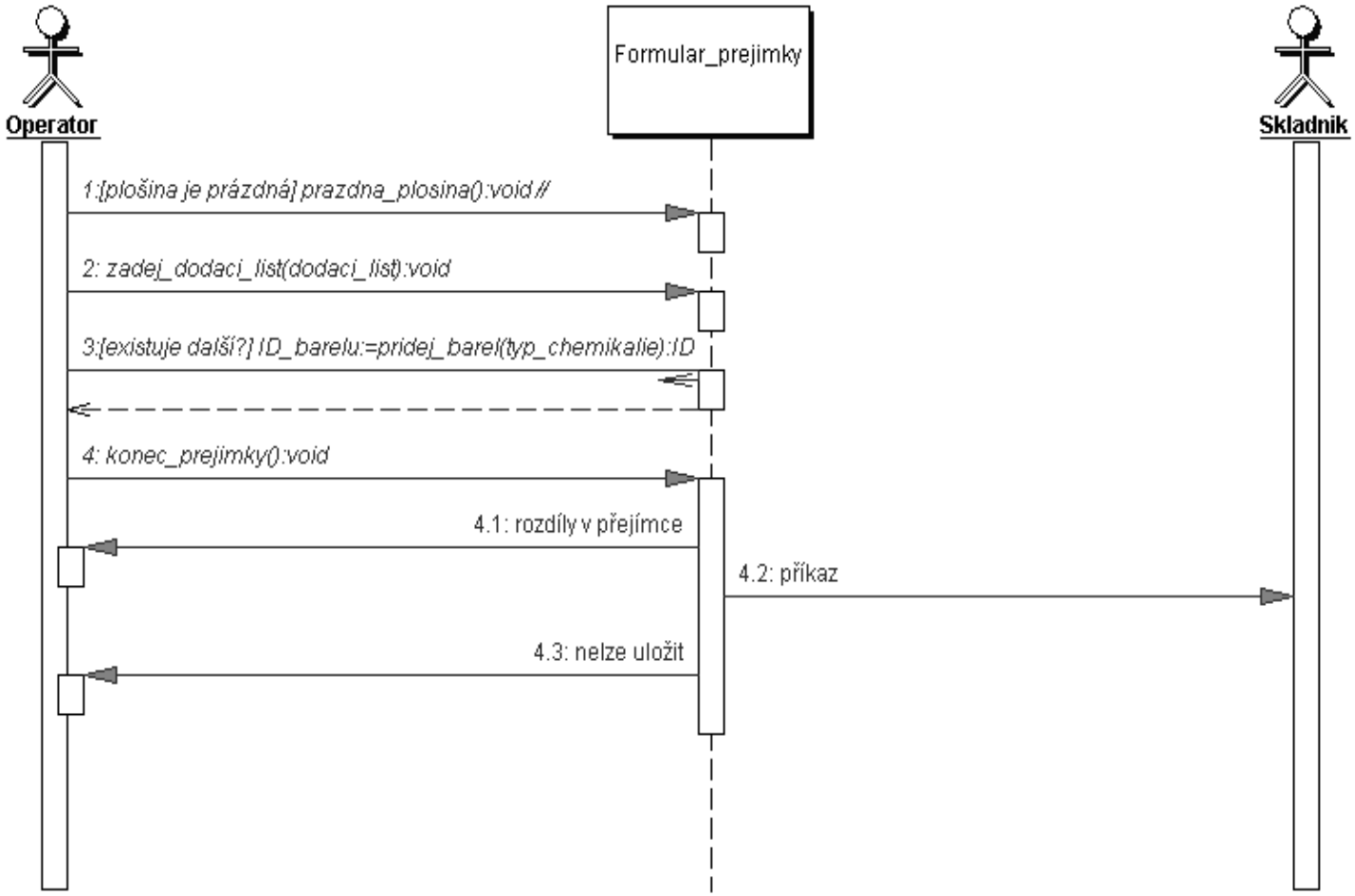
Příklad: Návrh dle Fusion

Př. ECO sklad


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

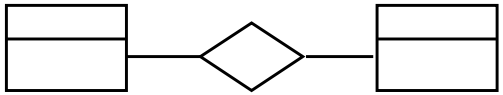

řidič
operátor
systém
skladník
příjezd_auta
obsazeno
volno
příprava_přejímky
vykládání
vyložení barelu
vyloženo vše
naložení nepřevzatých barelů
odjezd_auta
papír:Dodaci_List
zahájení přejímky
zpracování barelu
vyhodnocení přejímky
dokončení přejímky
:Stav
zadany:Dodaci_List
přejímka
:typChemikalie
:ID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

Scénář pro “přejímku” v UML

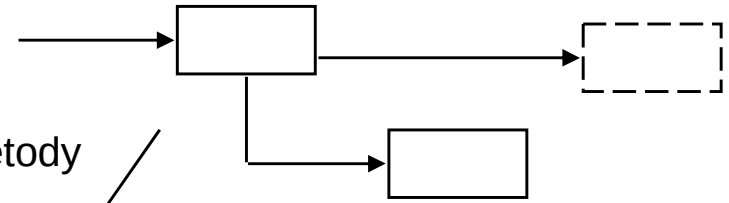


Princip návrhu ve Fusion

Datový model



Model komunikace objektů

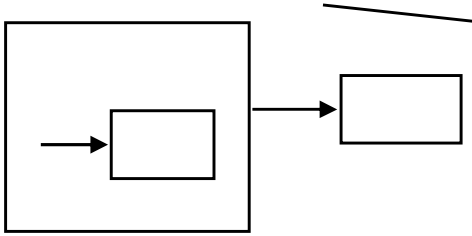


datové atributy

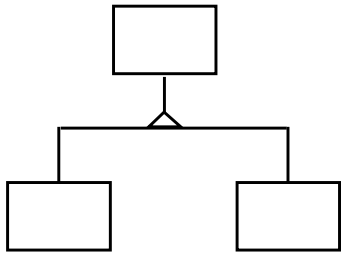
metody

Viditelnost

objektové atributy



isa



Popis tříd

class A **isa** S

  attribute a : int

  attribute b : **shared** B

  …

  method f(par)

  …

endclass

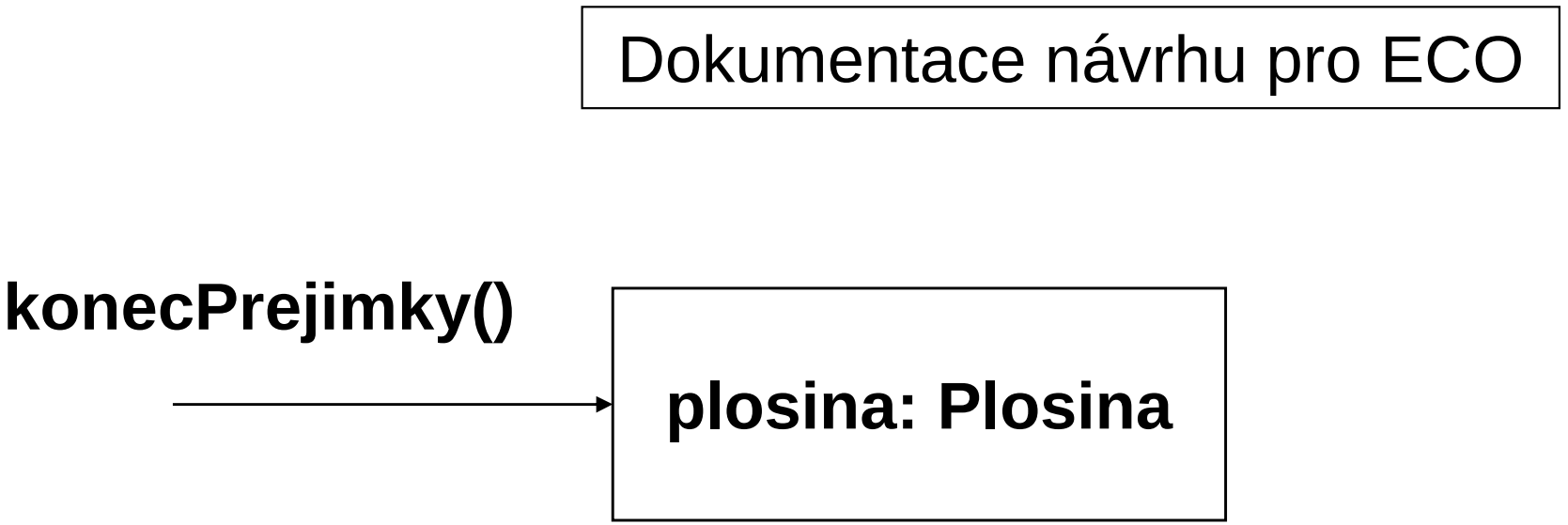
Dědičnost

Datový slovník

Postup návrhu ve Fusion

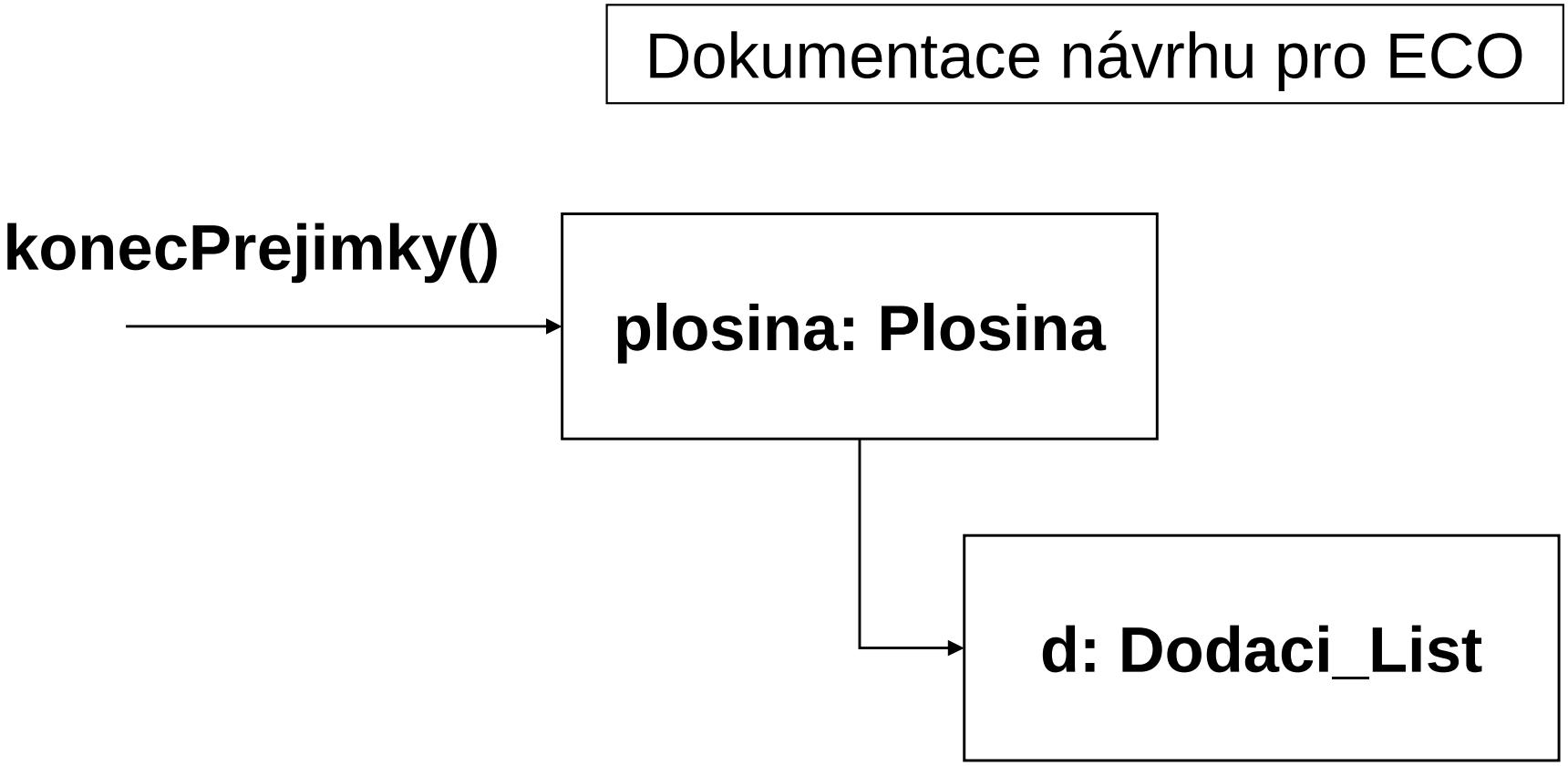
1. **Pro každou operaci funkčního modelu nakresli diagram komunikace (včetně metod vzniklých při návrhu)**
2. Pro každou třídu datového modelu zakresli potřebnou viditelnost podle diagramů komunikace
3. Pro každou třídu datového modelu vytvoř popis třídy
4. Doplň návrh o dědičnost

Diagram komunikace pro “konec přejímky“



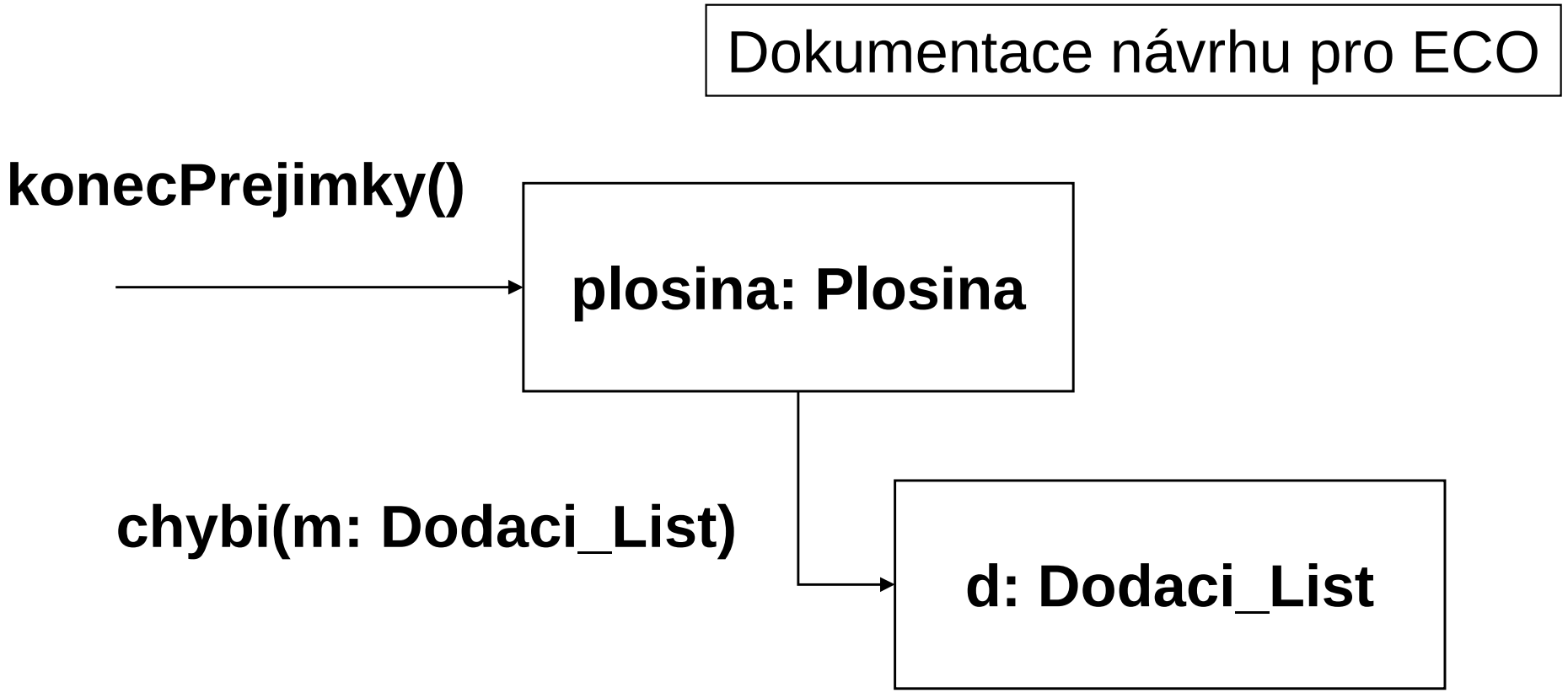
1. Výběr zodpovědného objektu

Diagram komunikace pro “konec přejímky“



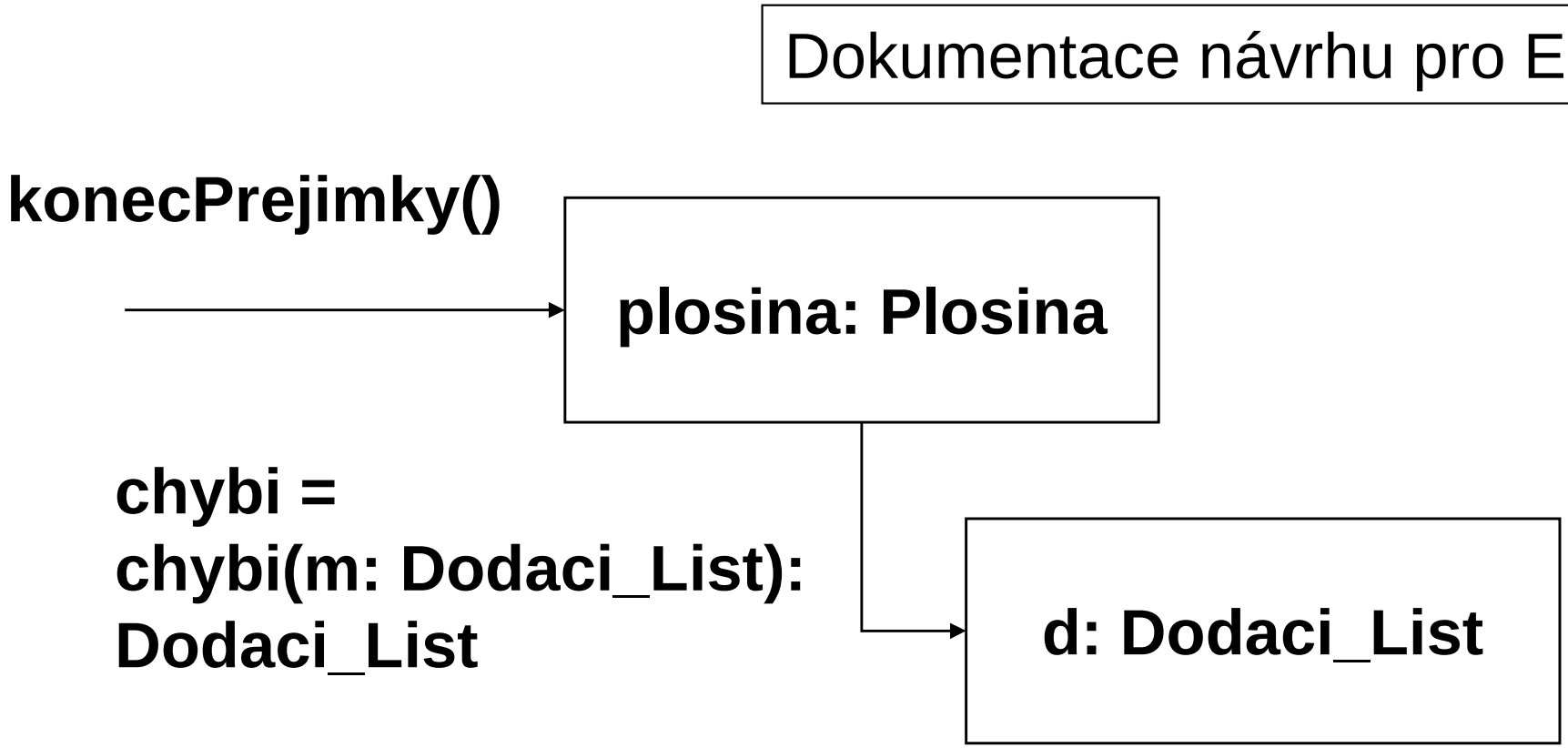
2. Výběr kooperujícího objektu

Diagram komunikace pro “konec přejímky“



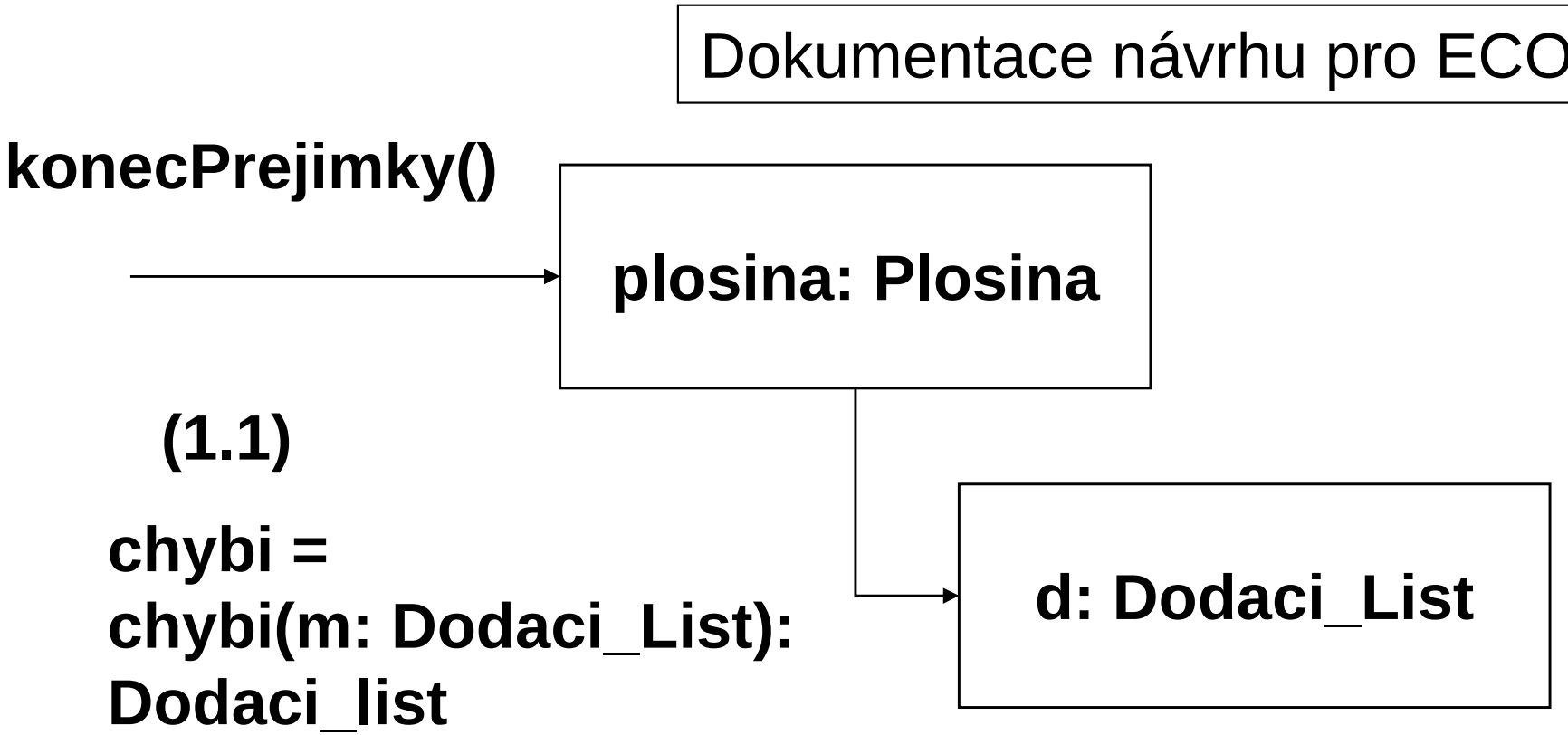
3. Výběr metody kooperujícího objektu

Diagram komunikace pro “konec přejímky“



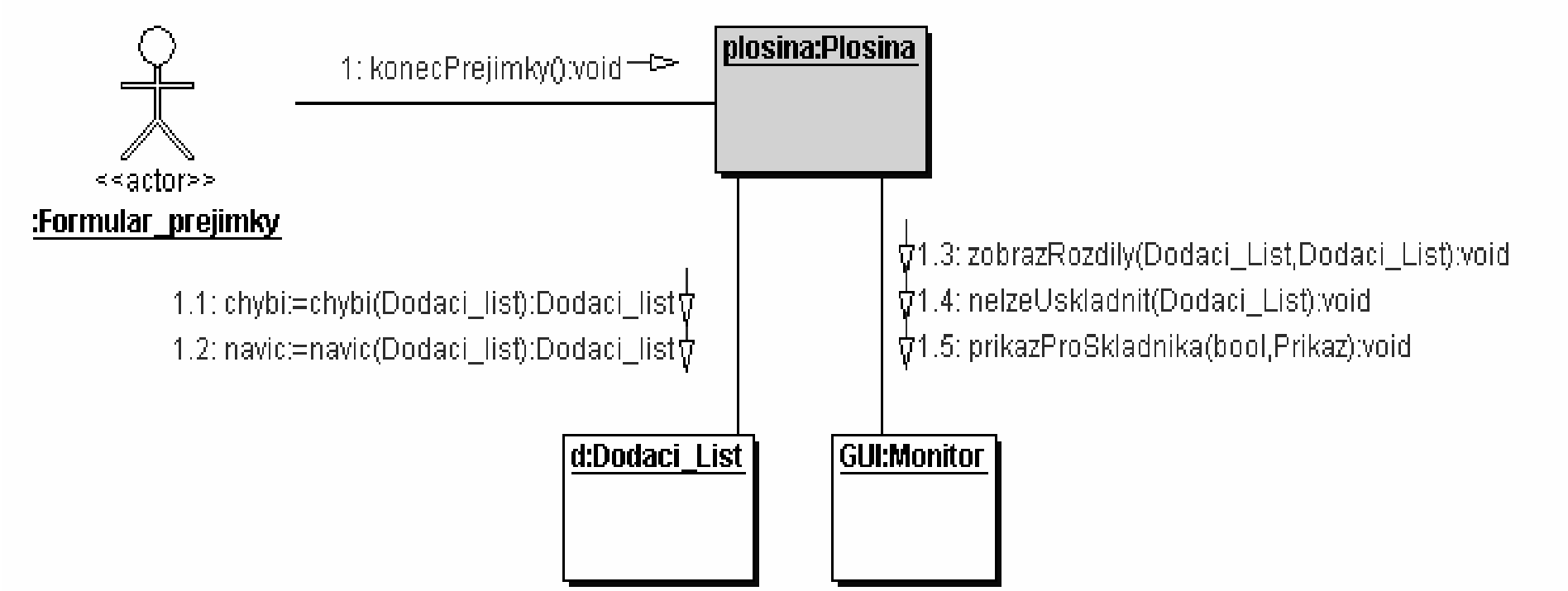
4. Bude něco vracet?

Diagram komunikace pro "konec přejímky"

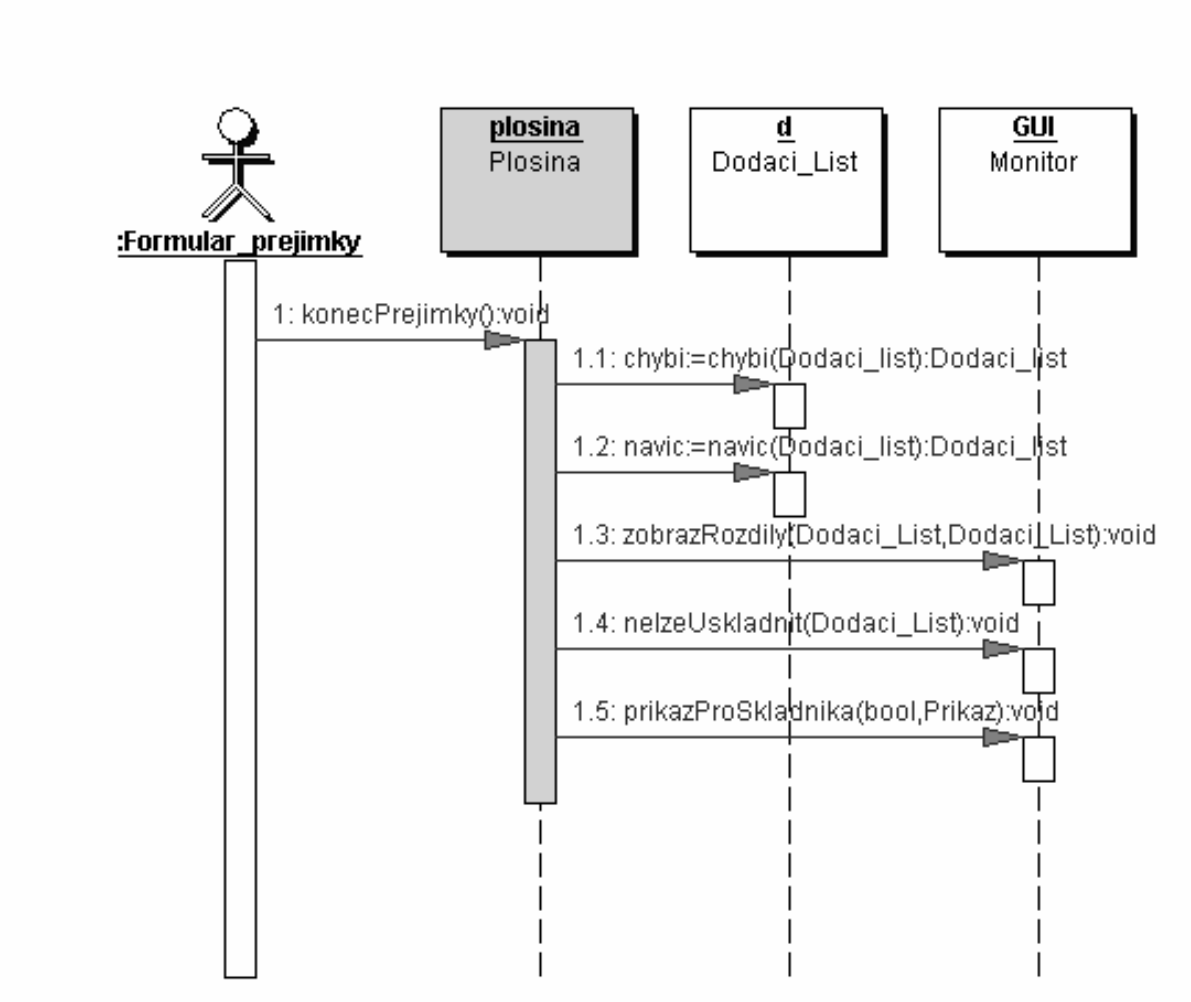


5. Stanovení pořadí volání

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



Popis pro „chybí“

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

Popis pro “chybí”

Dokumentace návrhu pro ECO

Operation: chybi

Description: zjistí, co chybí při přejímce

Reads: supplied m: dodaci_list,
zadany_dodaci_list: dodaci_list

Changes: new chybi: dodaci_list

Sends:

Assumes:

Results:

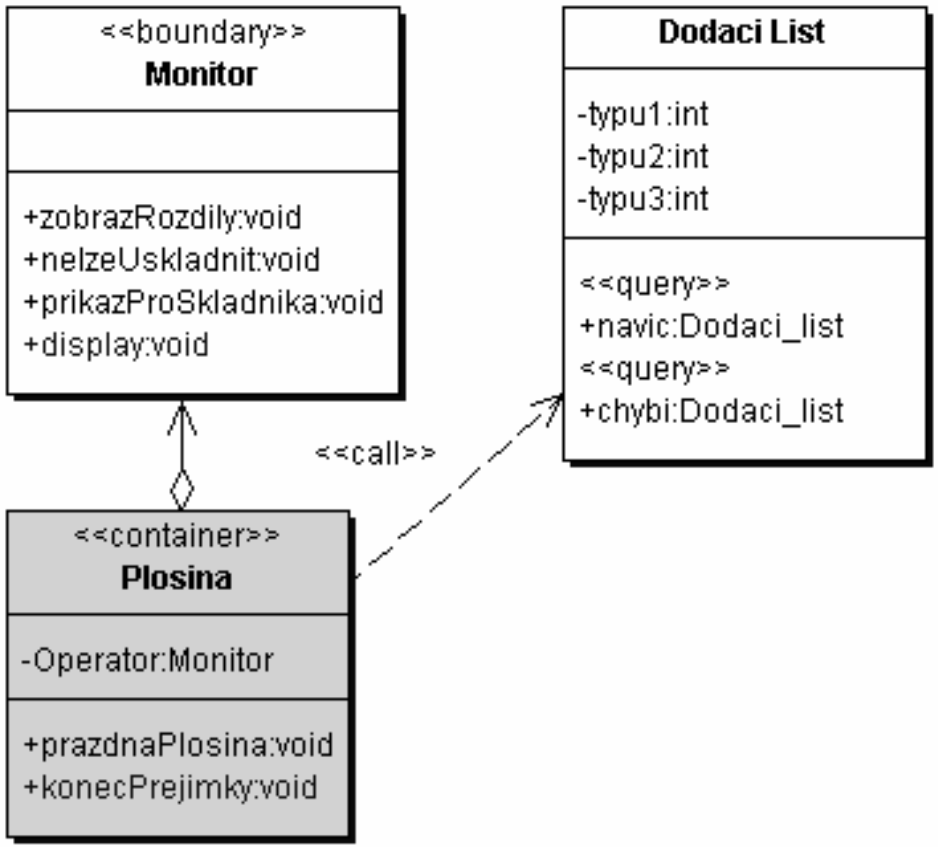
porovná zadaný dodací list m s dodacím listem zadany_dodaci_list

vytvoří objekt chybi: dodací list, obsahující barely které chybí

Postup návrhu ve Fusion

1. Pro každou operaci funkčního modelu nakresli diagram komunikace (včetně metod vzniklých při návrhu)
2. **Pro každou třídu datového modelu zakresli potřebnou viditelnost podle diagramů komunikace**
3. Pro každou třídu datového modelu vytvoř popis třídy
4. Doplň návrh o dědičnost

Viditelnost v UML



Postup návrhu ve Fusion

1. Pro každou operaci funkčního modelu nakresli diagram komunikace (včetně metod vzniklých při návrhu)
2. Pro každou třídu datového modelu zakresli potřebnou viditelnost podle diagramů komunikace
3. **Pro každou třídu datového modelu vytvoř popis třídy**
4. Doplň návrh o dědičnost

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 “Sklad”

Dokumentace návrhu pro ECO

```
class Sklad isa Budova
/*   attribute sousedi:
          exclusive bound col Budova
          – dedi od Budovy */
     attribute kapacita:int
     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
```

Popis třídy “Plosina”

Dokumentace návrhu pro ECO

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

Implementace třídy “Plošina”

Dokumentace implementace ECO

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

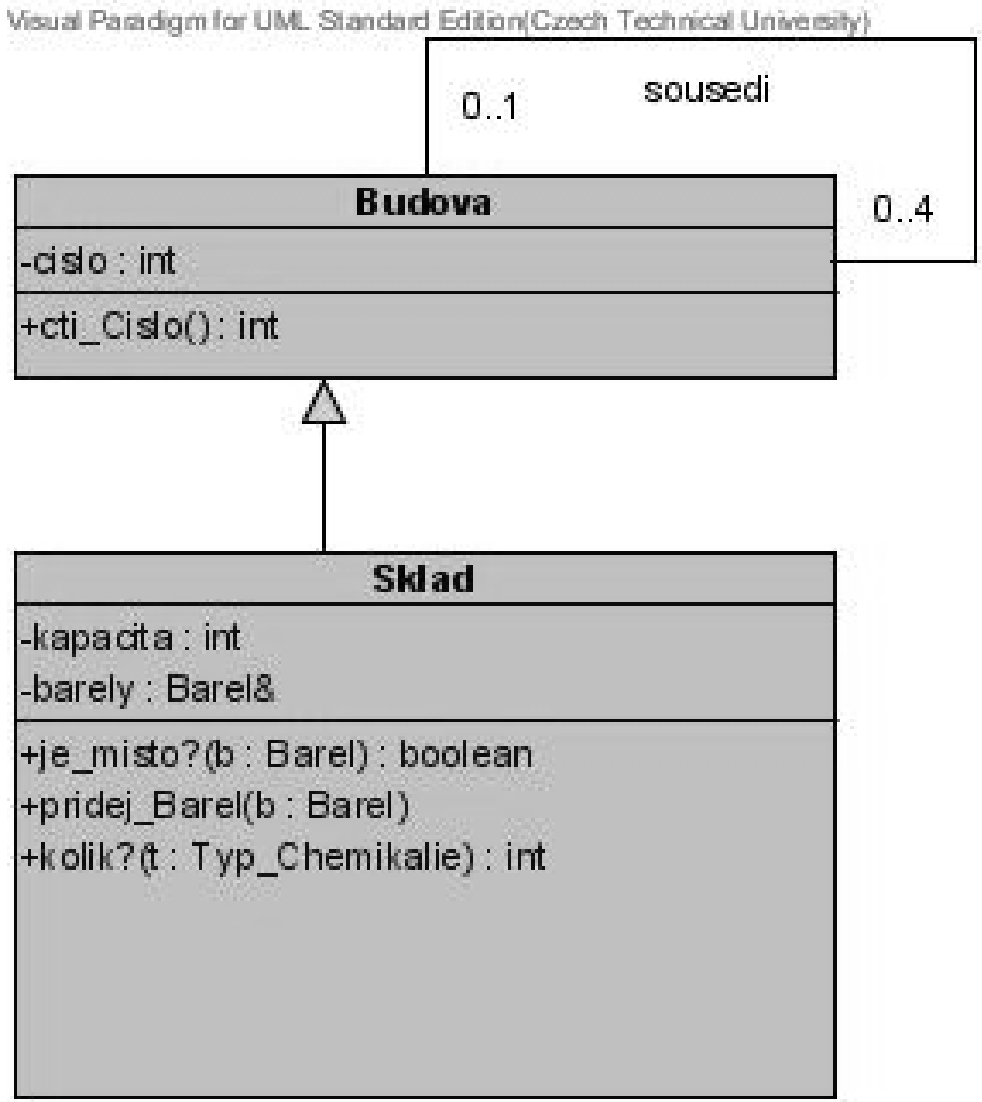
Postup návrhu ve Fusion

1. Pro každou operaci funkčního modelu nakresli diagram komunikace (včetně metod vzniklých při návrhu)
2. Pro každou třídu datového modelu zakresli potřebnou viditelnost podle diagramů komunikace
3. Pro každou třídu datového modelu vytvoř popis třídy
4. **Doplň návrh o dědičnost**

Dva zdroje dědičnosti

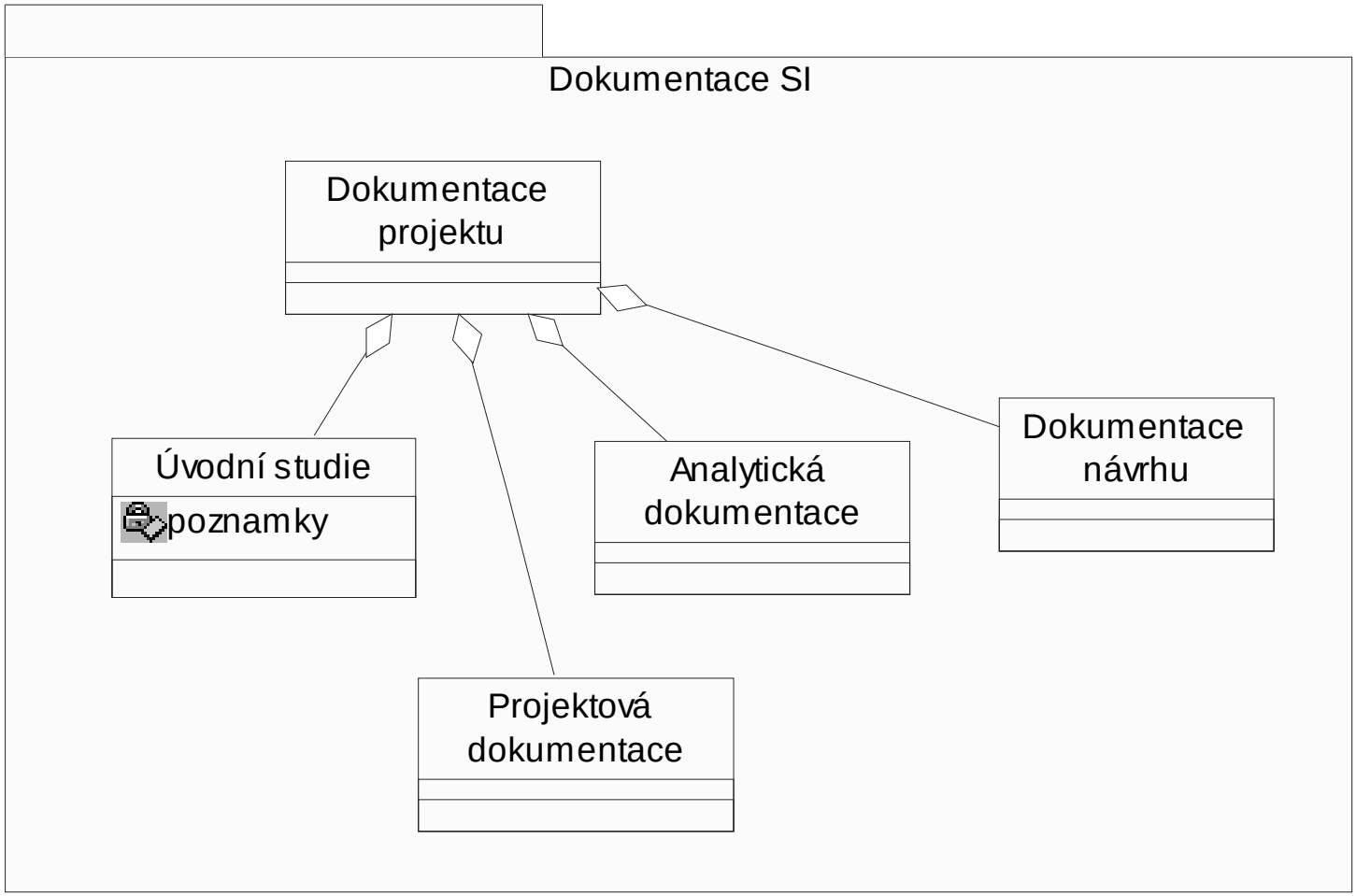
- Společné a speciální vlastnosti dat odhalené při analýze
- Vlastnosti zděděné z použitých knihoven

Popis třídy “Sklad”



Dokumentace
návrhu
pro ECO

Co bude výstupem SIN?



The End

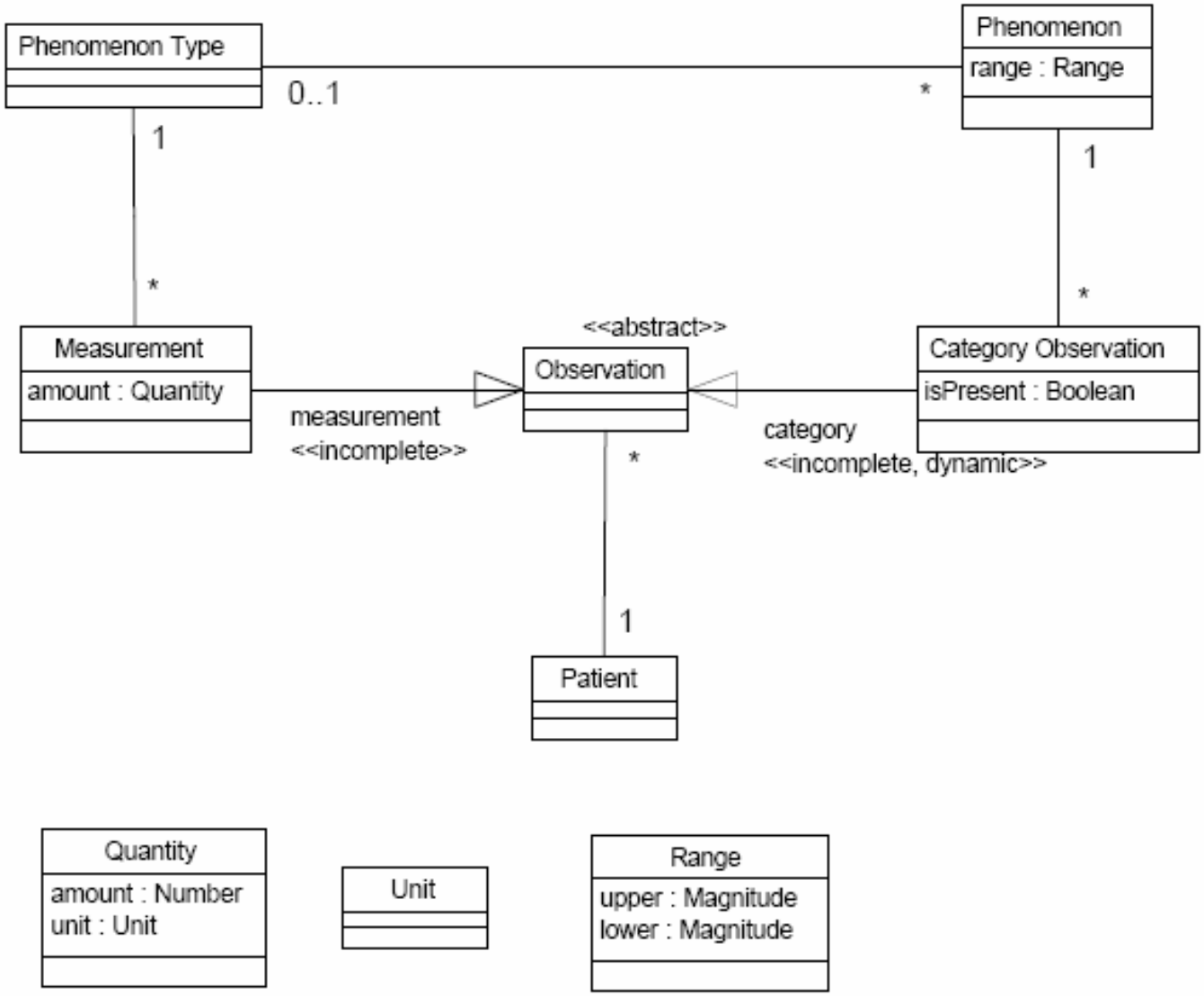
Návrhové vzory (Design Patterns)

JAK se to obvykle dělá

Typy vzorů

- Analytické vzory (konceptuální data banky)
- Architektonické vzory (vrstvená architektura, MVC)
- Návrhové vzory (sekvenční průchod kolekcí – iterátor)
- Programovací vzory (implementace seznamu pomocí pole)

Analytický vzor: Měření a pozorování

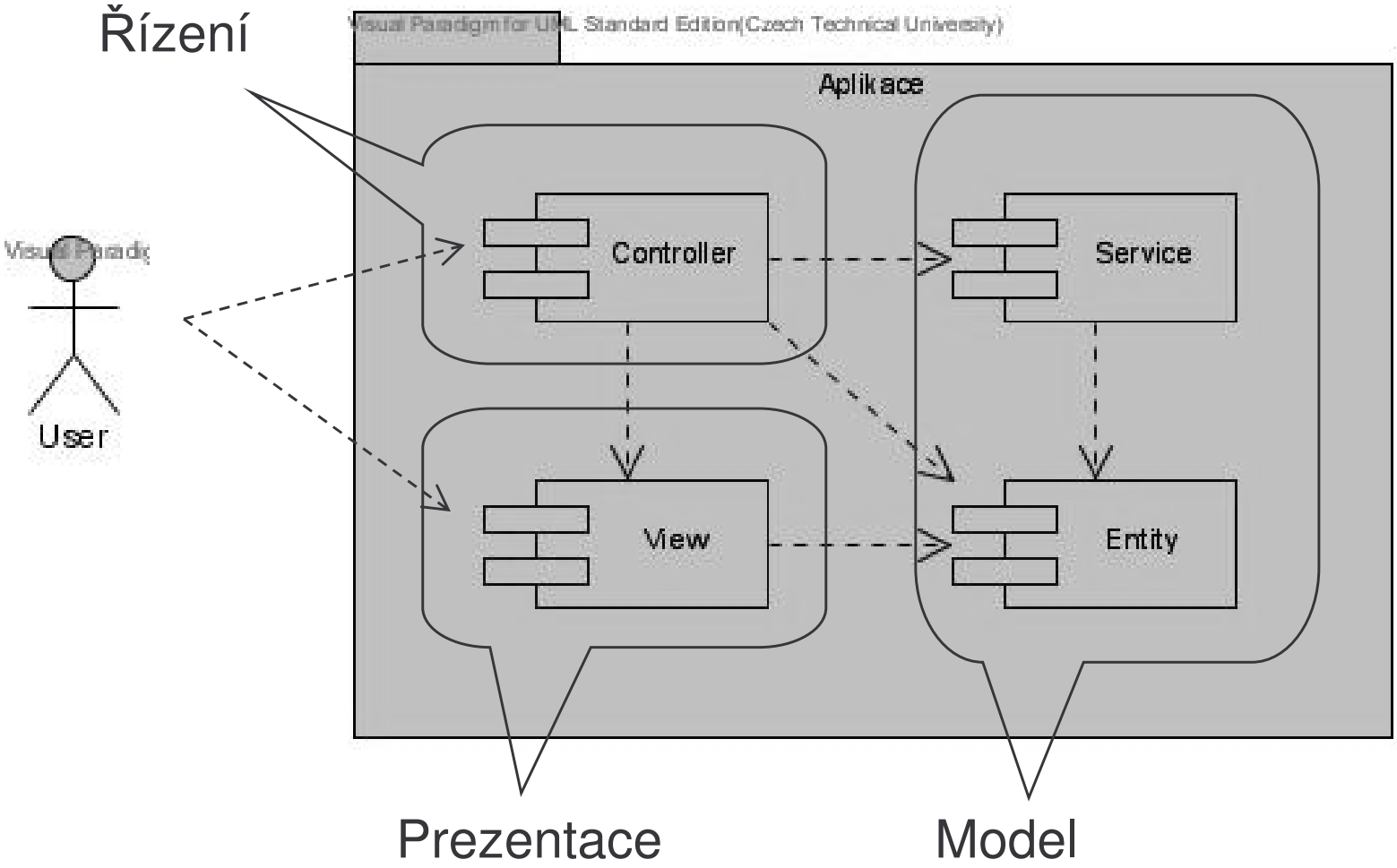


Zdroj: http://www.martinfowler.com/apsupp/appfacades.pdf

Architektonické vzory

- Vzor pro konstrukci, např. snaha o konstrukci dostatečně pružné aplikace
- Vyplatí se oddělit správu dat obhospodařovaných danou aplikací, jejich prezentaci a vlastní řízení aplikace
- Toto rozdělení pak vyjádřit přímo architekturou aplikace

Architektonický vzor MVC



Co jsou návrhové vzory?

- Standardní řešení častých problémů v návrhu softwaru
- Příklad: Sada vzorů GoF = Gang of Four
- Co nejsou návrhové vzory?
  - idiomy = ustálené fragmenty kódu – konkrétní prog. jazyk, nejde o návrh
  - konvence kódu – nejde o návrh
  - algoritmy – řeší výpočet, ne návrh

Co to je návrhový vzor?

- Christopher Alexander: *”Každý vzor popisuje často se vyskytující problém a poté popisuje jádro řešení tohoto problému tak, aby bylo možno toto řešení opakovaně využívat, bez toho, že bychom stejnou věc dělali dvakrát".*
- Přestože se tato definice týká návrhu budov, totéž platí pro návrhové vzory při tvorbě programů. Řešení je zde vyjádřeno pomocí potřebných objektů a způsobu jejich komunikace.

Zdroje příkladů:

- Použity vzory GoF (Gamma, Helm, Johnson, Vlisidess: Design Patterns - The Elements of reusable object-oriented software. 1995)
- Návrhové vzory I., FEL 2005, Tomáš Richta
- Návrhové vzory II., FEL 2005, Petr Šlégr

Obecné části návrhových vzorů

- Jméno vzoru
- Popis problému
- Popis řešení
- Důsledky použití

Jméno vzoru

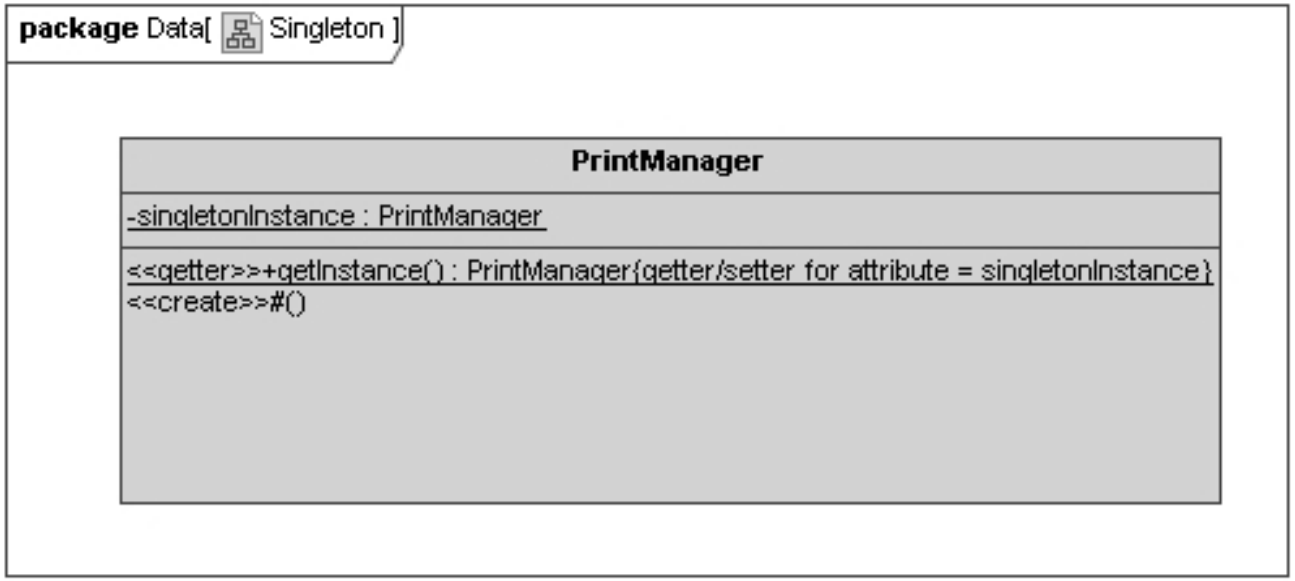
- Termín, přes který se na vzor odvoláváme.
- Aby se v katalogu vzorů dobře a intuitivně hledalo, je volba jména důležitá a obtížná.
- Př.: Proxy (zástupce), Iterator (iterátor)

Popis problému pro vzor

- Vyjadřuje situaci, kde se použití vzoru hodí.
- Může se stanovit příkladem, seznamem podmínek, které musí platit, apod.

Problém

- Je třeba, aby v aplikaci existovala pouze jedna instance nějaké třídy. Jak to zajistit?



Singleton

```
public class PrintManager {
    private static PrintManager instance;

    private PrintManager () { /* ... */  }

    // jina jmena: getDefault(), getPrintManager,...
    public static synchronized PrintManager getInstance () {
        if (instance == null) { // "lazy" inicializace
            instance = new PrintManager ();
        }
        return instance;
    }

    // vlastni instancni metody...
}
```

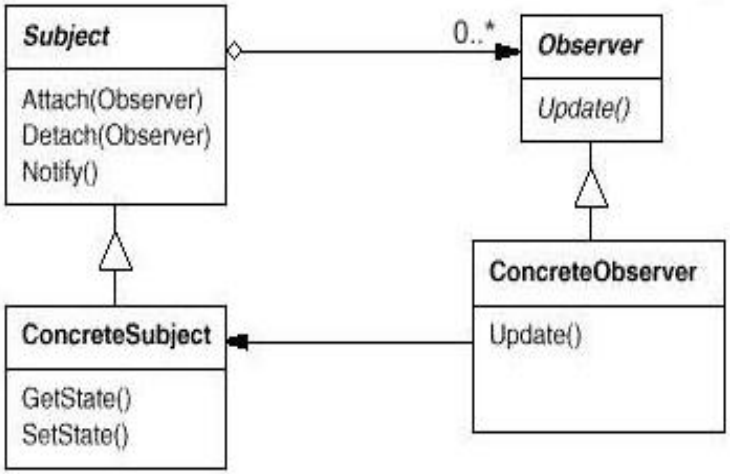
Singleton

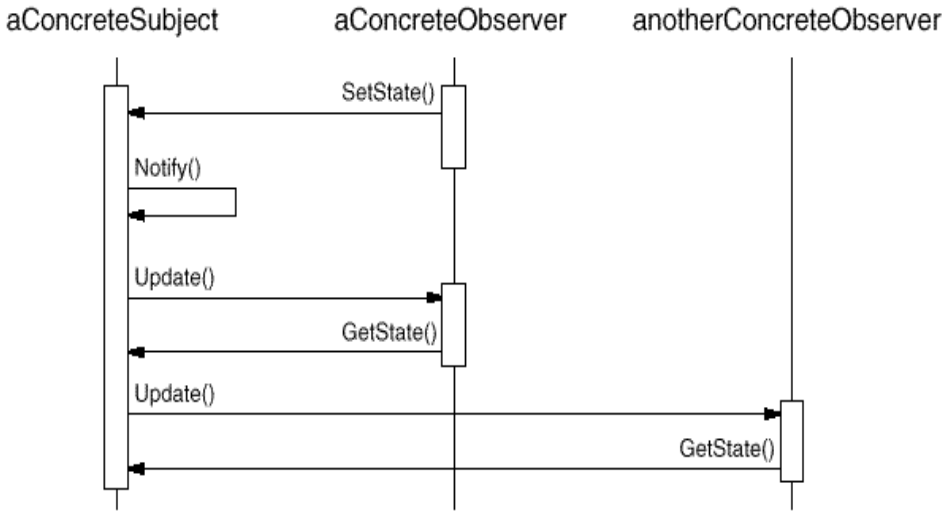
- Vlastnosti:
  - konkrétní typ poskytnuté instance lze určit za běhu
  - poskytovanou instanci lze měnit bez změn klientského kódu
  - snadná změna logiky tvorby - např. povolení více instancí
- Příklad v J2SE: `java.lang.Runtime`

Problém

- Na stavu jednoho objektu závisí řada jiných objektů.
- Při změně stavu objektu potřebují být závislé objekty automaticky vyrozuměny.
- Jak toho docílit?

Observer/Pozorovatel



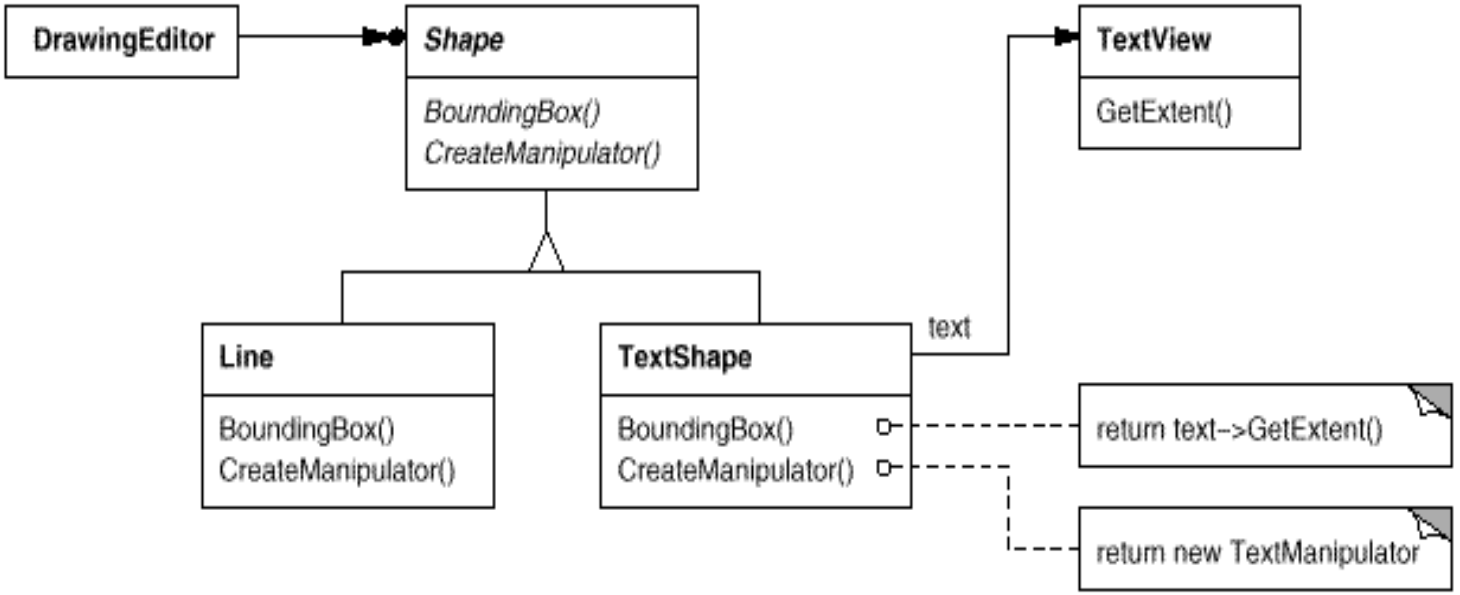


Problém

- Chceme použít nějakou užitečnou třídu, která ale má jiné rozhraní, než očekáváme. Co s tím?

Adapter/Adaptér

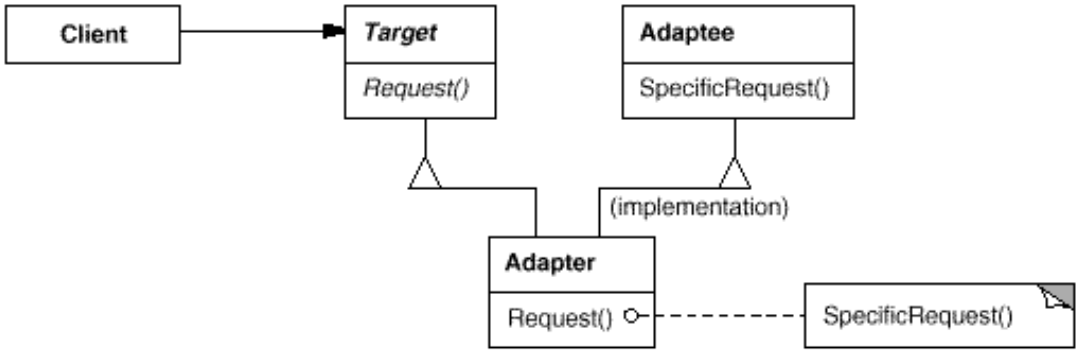
- Také wrapper/obal



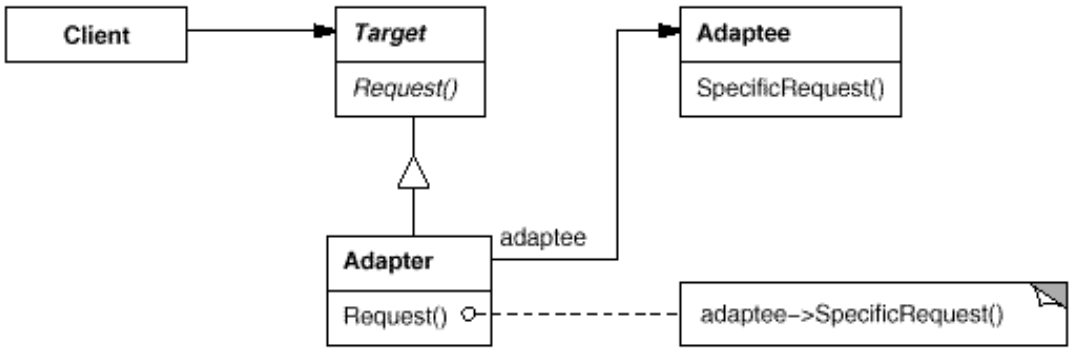
Obrázek převzat z [GoF]

Adapter/Adaptér

- Třídní implementace: vícenásobná dědičnost



- Instanční implementace: kompozice

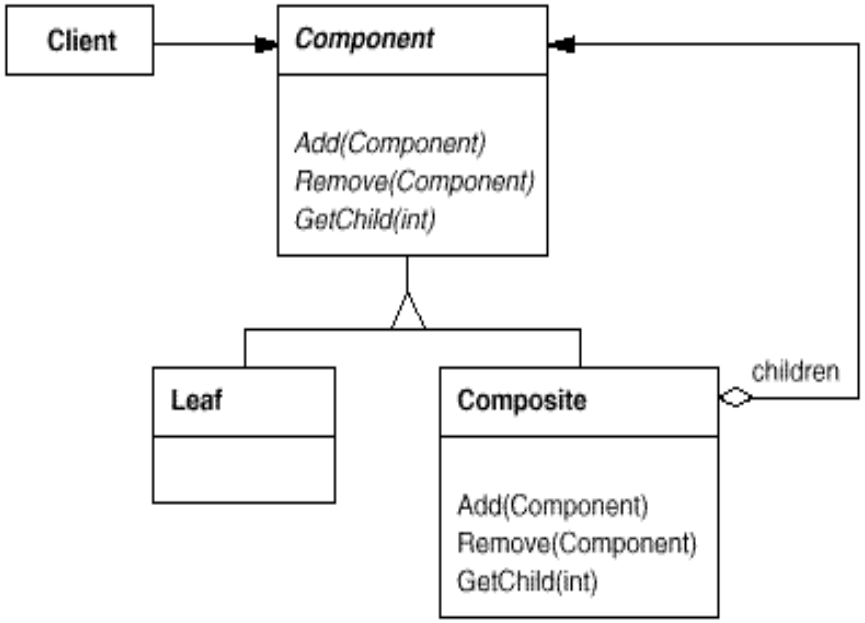


Obrázky převzaty z [GoF]

Problém

- Máme hierarchicky organizovaný celek s jehož částmi chceme manipulovat jednotným způsobem. Jak na to?

Composite/Kompozice



Obrázek převzat z [GoF]

- Použití:
  - grafická uživatelská rozhraní
  - strukturované dokumenty (XML, HTML,...)
  - stromově organizovaná data...

Návrhové vzory: kategorie

- Vzory pro **vytváření** a **manipulaci** s reprezentací informace (např. Abstraktní továrna, Proxy)
- Vzory **strukturální** vyjadřující strukturu implementace (např. Adaptér, Kompozice)
- Vzory pro **chování** (např. Iterátor)

Taxonomie návrhových vzorů

	Tvorba Vznik nových objektů	**Struktura** Skládání tříd/objektů	**Chování** Interakce tříd/objektů
Třída Statické vztahy	Tovární metoda	Adaptér (třídní)	Šablonová metoda
Instance Dynamické vztahy	Jedináček	Adaptér (objektový), Kompozice	Strategie, Pozorovatel, Iterátor

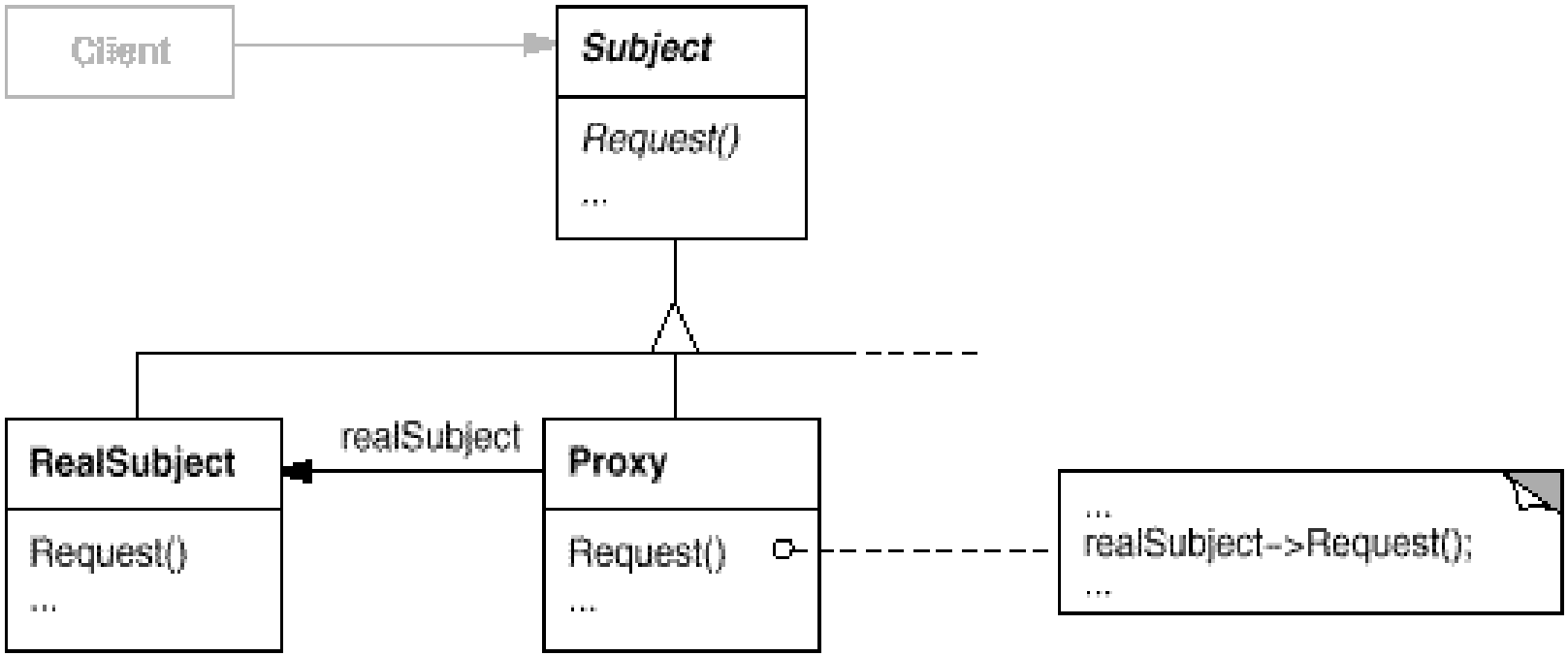
Př: Návrhový vzor "Proxy"

- Deklarace záměru:
  - pokud potřebujeme zástupce objektu, neboť vlastní objekt je někde jinde, je těžko přístupný, má být chráněn, ...
- Motivační příklad:
  - editor dokumentu by měl umět pracovat s obrázky, ale zobrazení obrázku je náročné - často postačí náhradník

Popis řešení pro vzor

- Popisuje elementy použité při řešení a jejich vztahy.
- Nejedná se o konkrétní implementaci, neboť vzor je pouze šablonou pro řešení.

Struktura vzoru Proxy

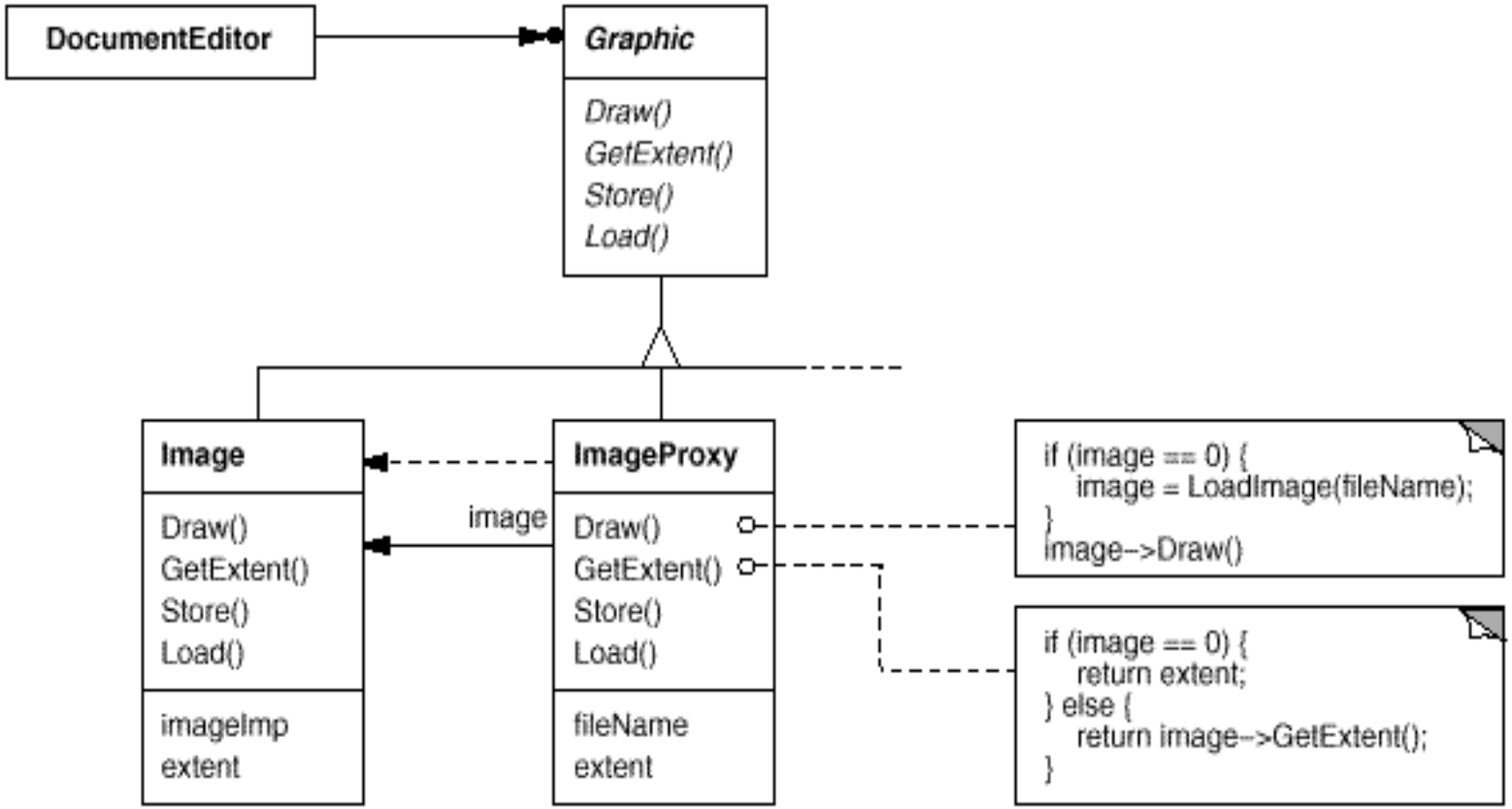


Obrázek převzat z [GoF]

Důsledky použití vzoru

- Rozmanité důsledky, které s sebou použití vzoru přináší - např. časové a prostorové nároky.
- Poslouží dobře i při výběru alternativ.

Příklad použití vzoru Proxy



Obrázek převzat z [GoF]

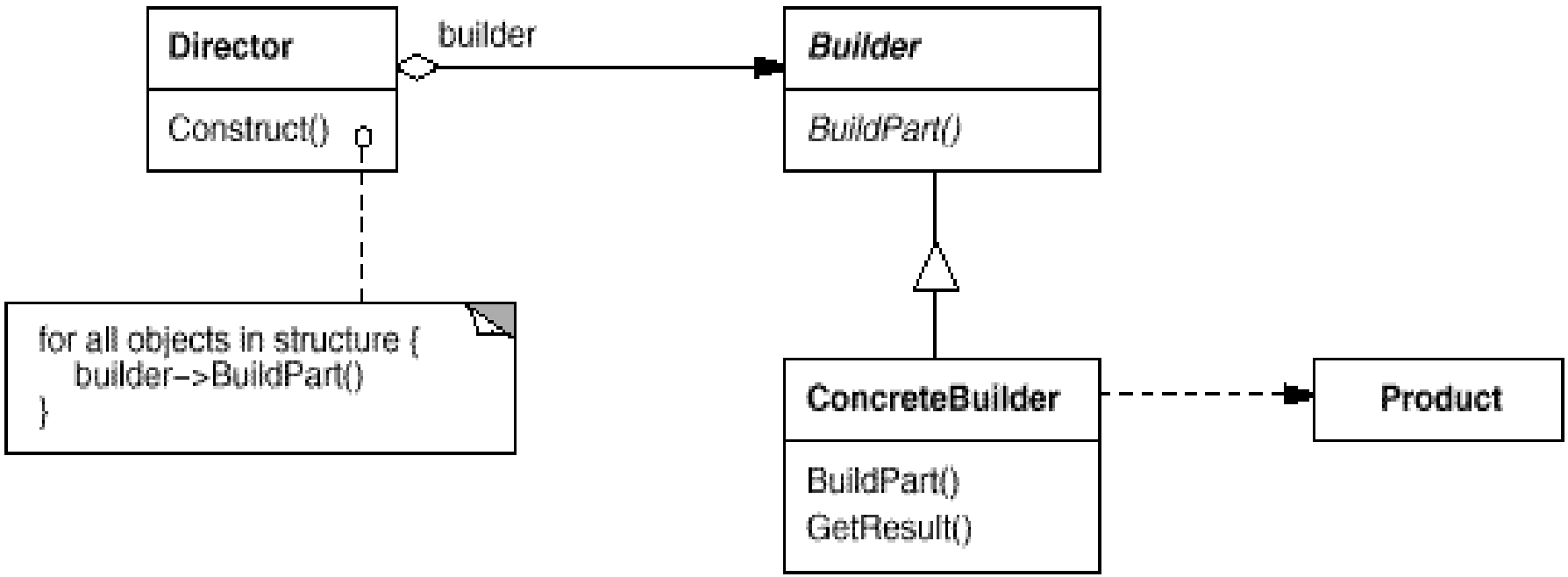
Příbuzné strukturální vzory

- **Proxy** zastupuje objekt a poskytuje stejné rozhranní jako on
- **Adapter** mění rozhranní, které objekt poskytuje
- **Dekorátor** něco přidává

Vzor: Builder

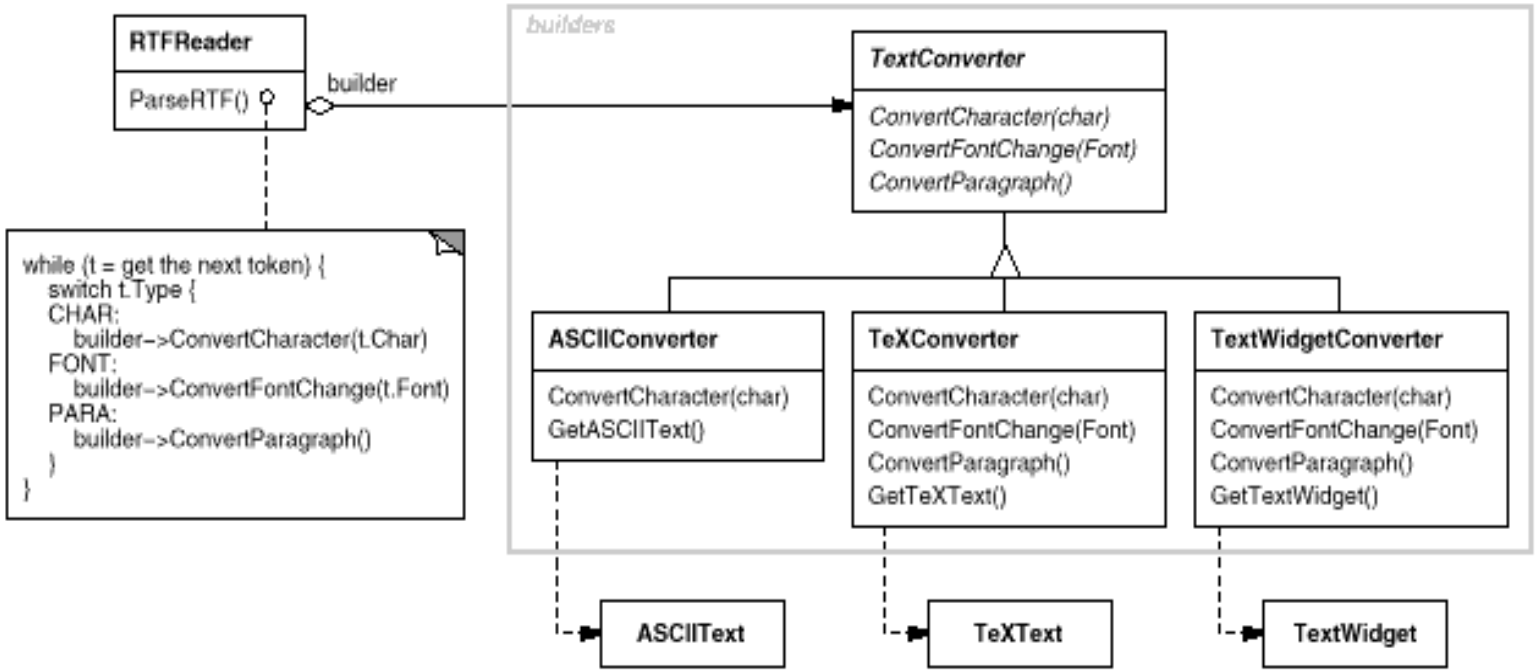
- **Deklarace záměru:**
  - oddělení konstrukce složitého objektu od jeho reprezentace
  - pokud má být algoritmus pro vytváření složitého objektu nezávislý na vytváření částí
  - pokud lze objekt reprezentovat různými způsoby
- **Motivační příklad:**
  - editor dokumentu v RTF by měl umět pracovat s různými reprezentacemi textu (ASCII, TeX, …)

Struktura vzoru Builder



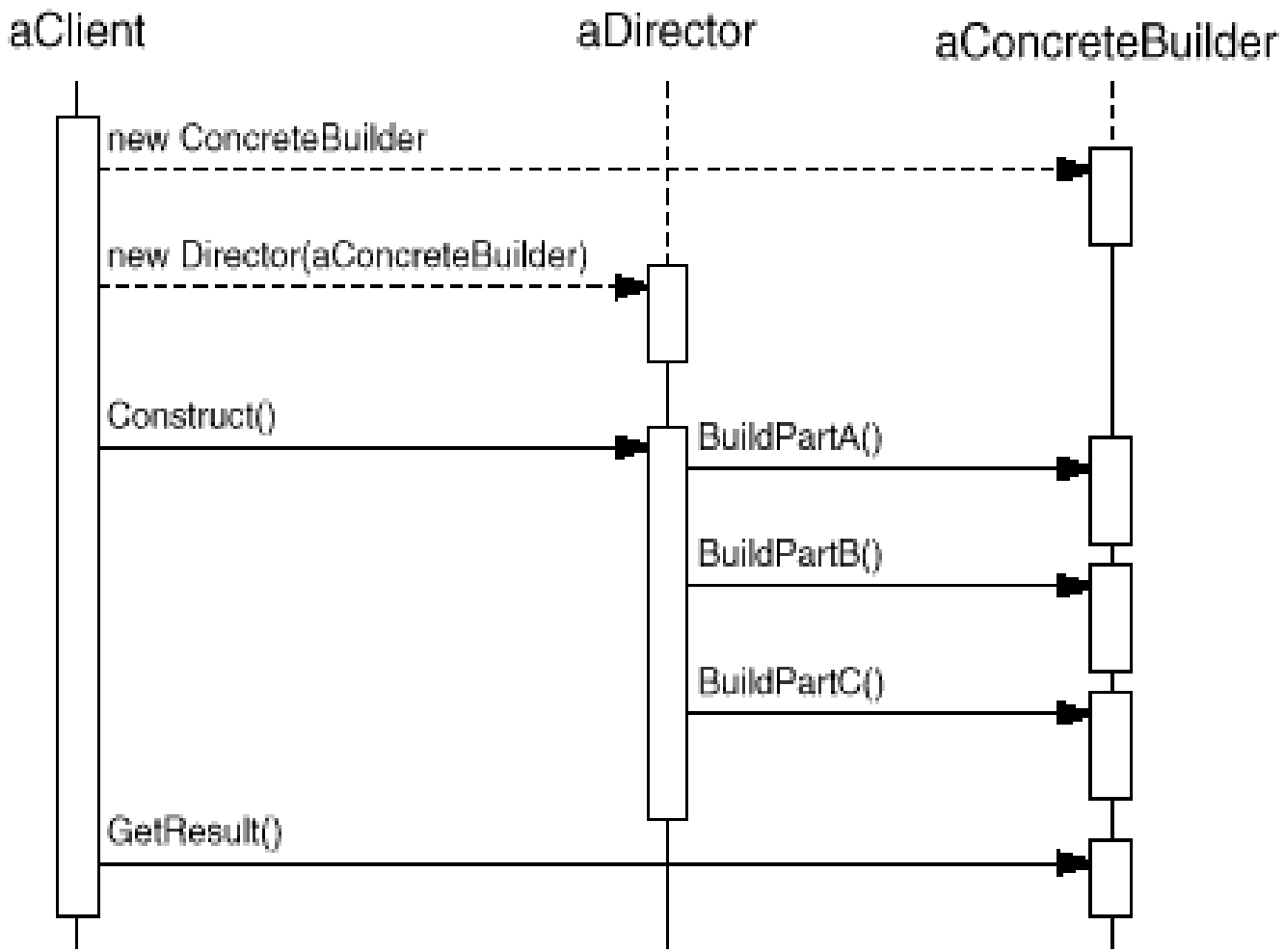
Obrázek převzat z [GoF]

Příklad použití vzoru Builder



Obrázek převzat z [GoF]

Kolaborace při použití Builder

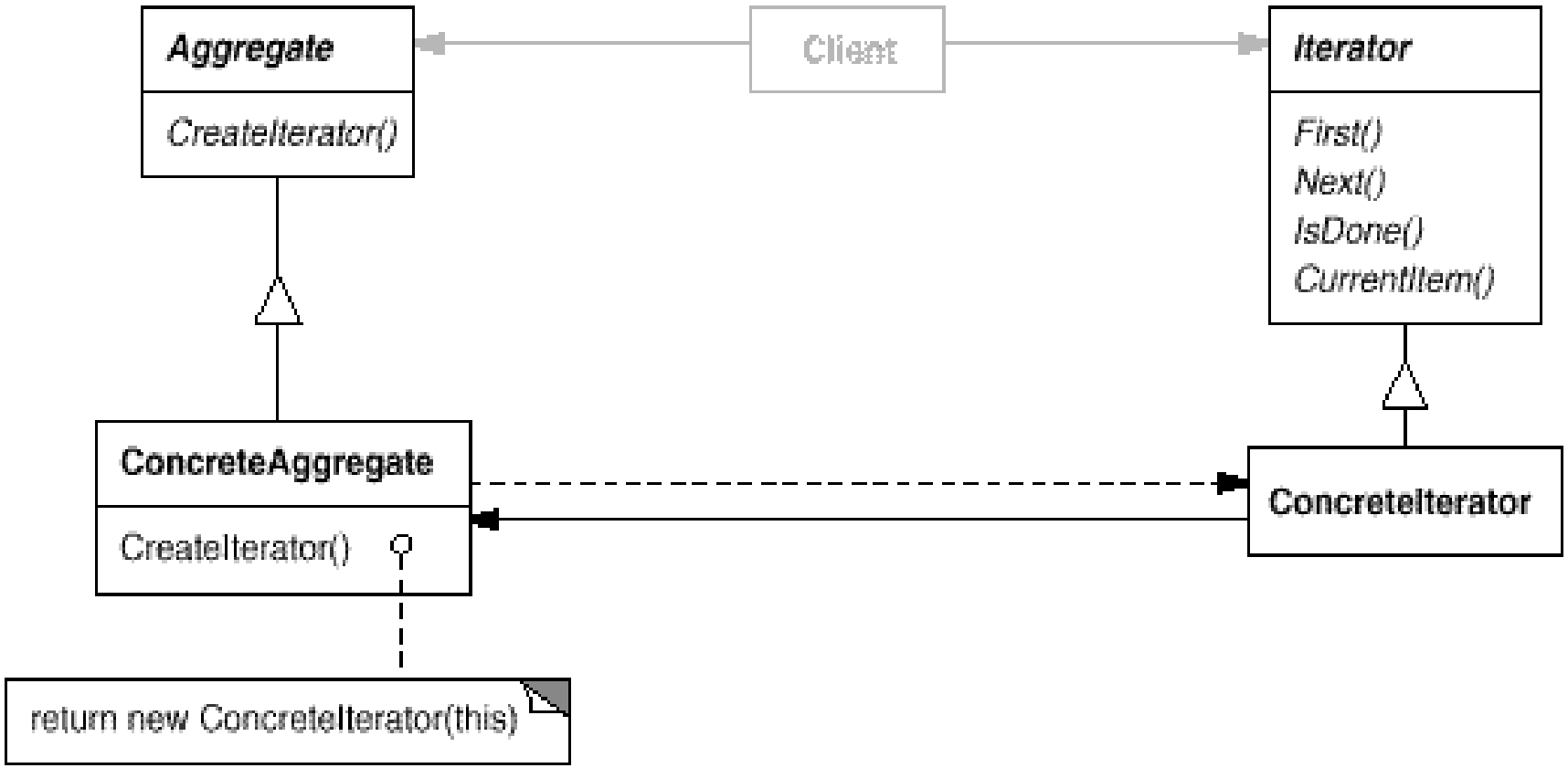


Obrázek převzat z [GoF]

Vzor: Iterator (Cursor)

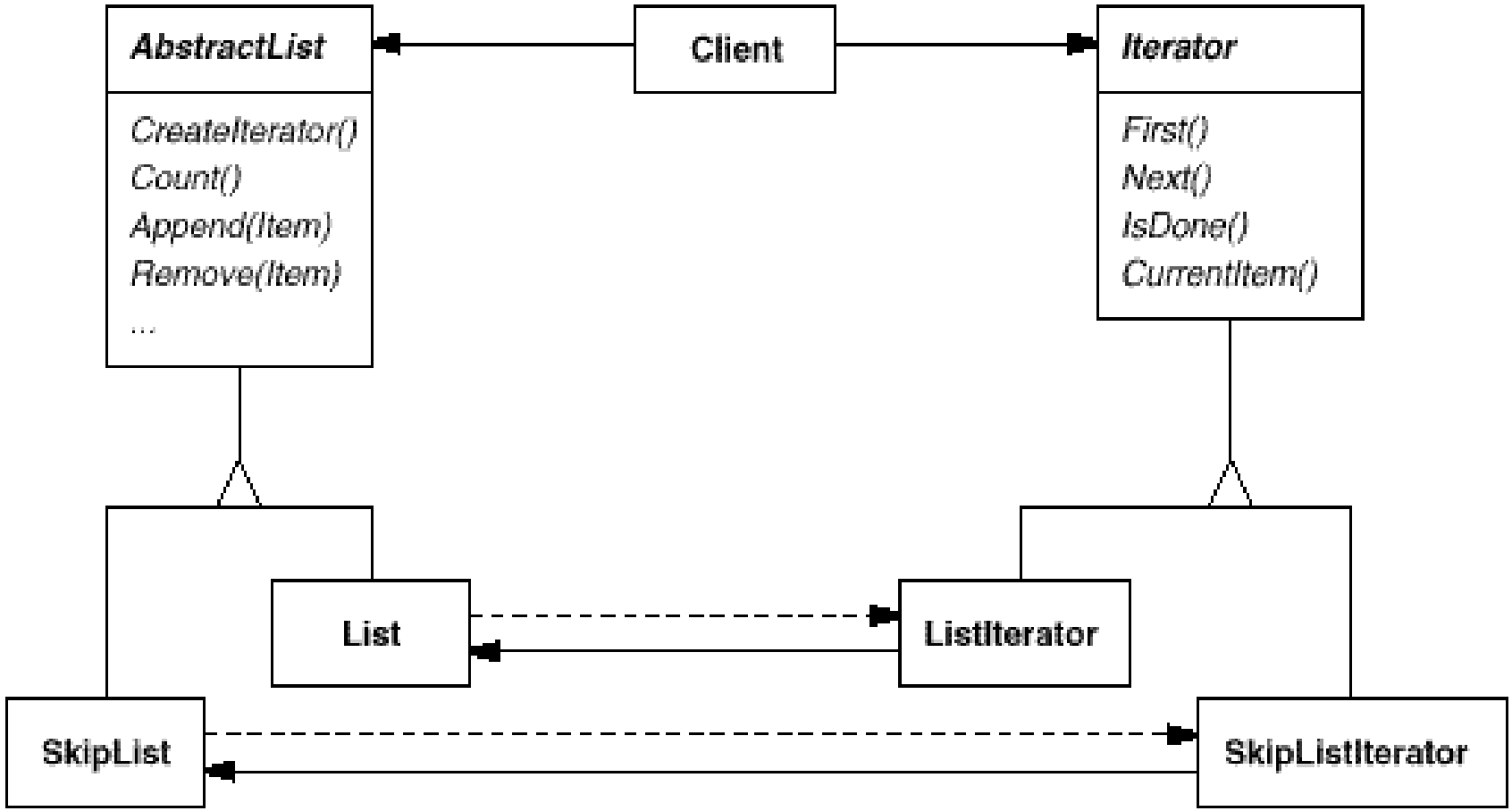
- **Deklarace záměru:**
  - pro sekvenční přístup ke složkám složeného objektu bez ohledu na reprezentaci - pokud má být procházen seznam aniž se zabýváme jeho reprezentací (uniformní přístup pro traverzování agregovaných struktur)
- **Motivační příklad:**
  - sekvenční průchod seznamy

Struktura pro Iterator



Obrázek převzat z [GoF]

Příklad použití Iterator

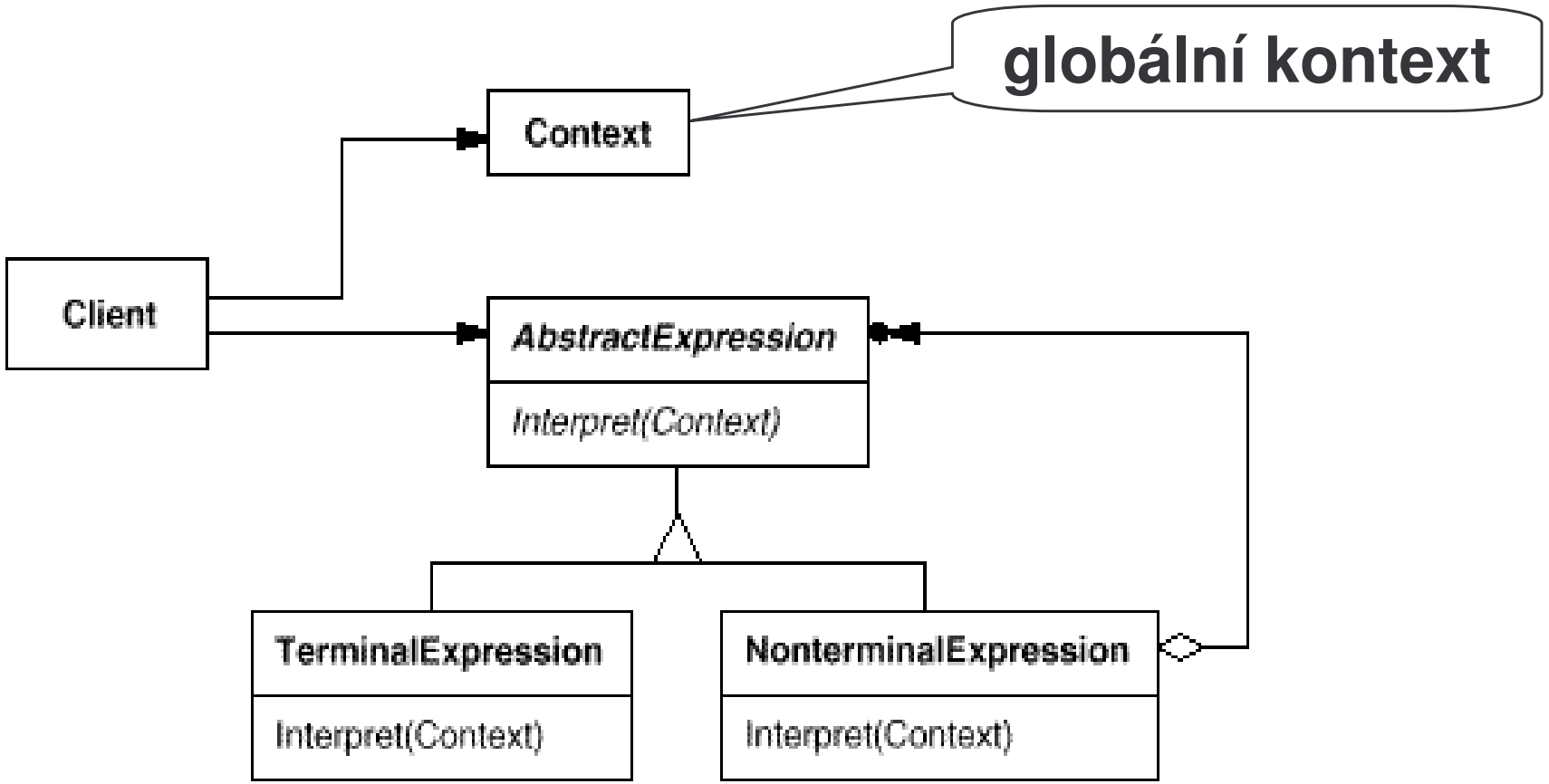


Obrázek převzat z [GoF]

Vzor: Interpret

- Deklarace záměru:
  - máme zadánu gramatiku jazyka, jehož věty chceme interpretovat
- Motivační příklad:
  - hledání vzorku zadaného regulárním výrazem v řetězci (známe gramatiku regulárního výrazu)

Struktura vzoru “Interpret”



Obrázek převzat z [GoF]

Gramatika pro regulární výraz

expression ::= literal | alternation | sequence

| repetition | '(' expression ')'

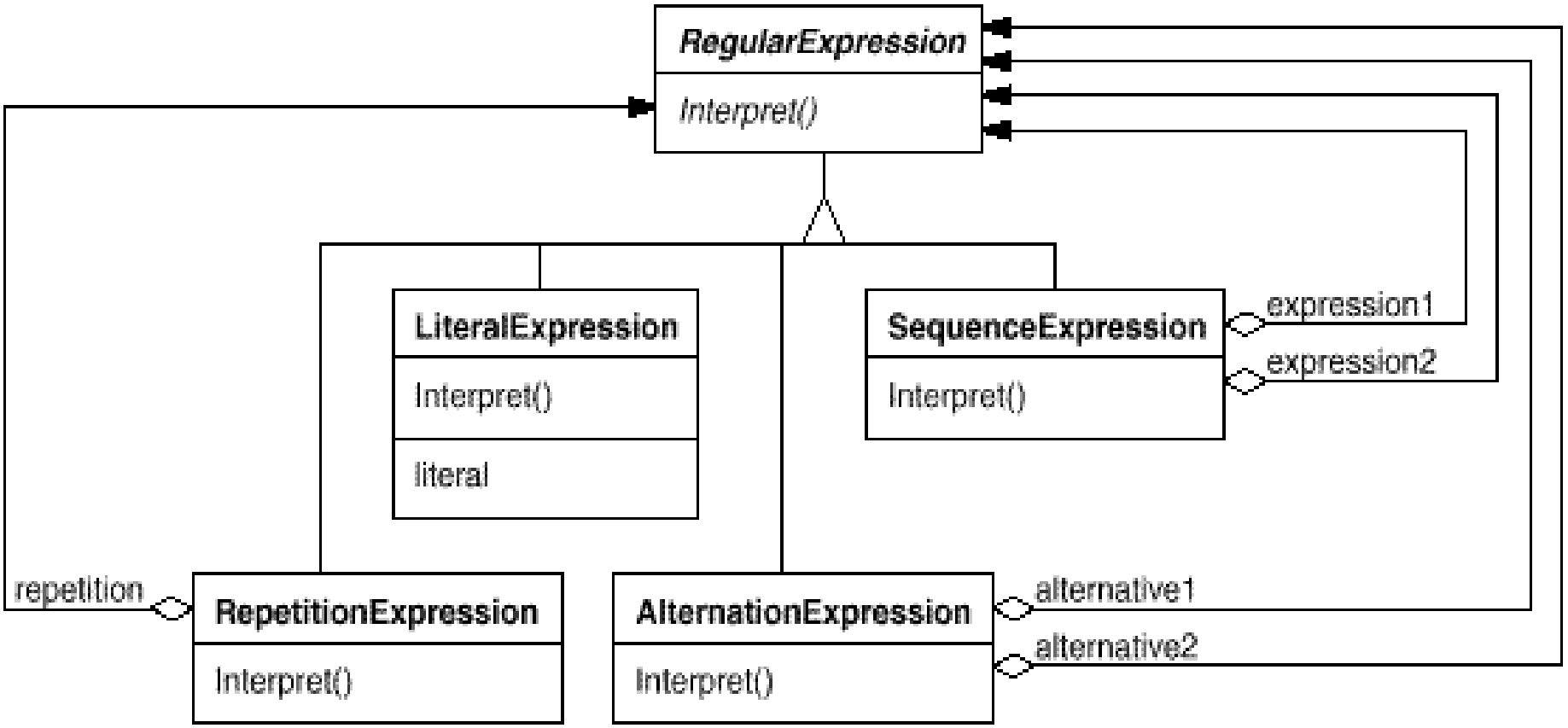
alternation ::= expression '|' expression

sequence ::= expression '&' expression

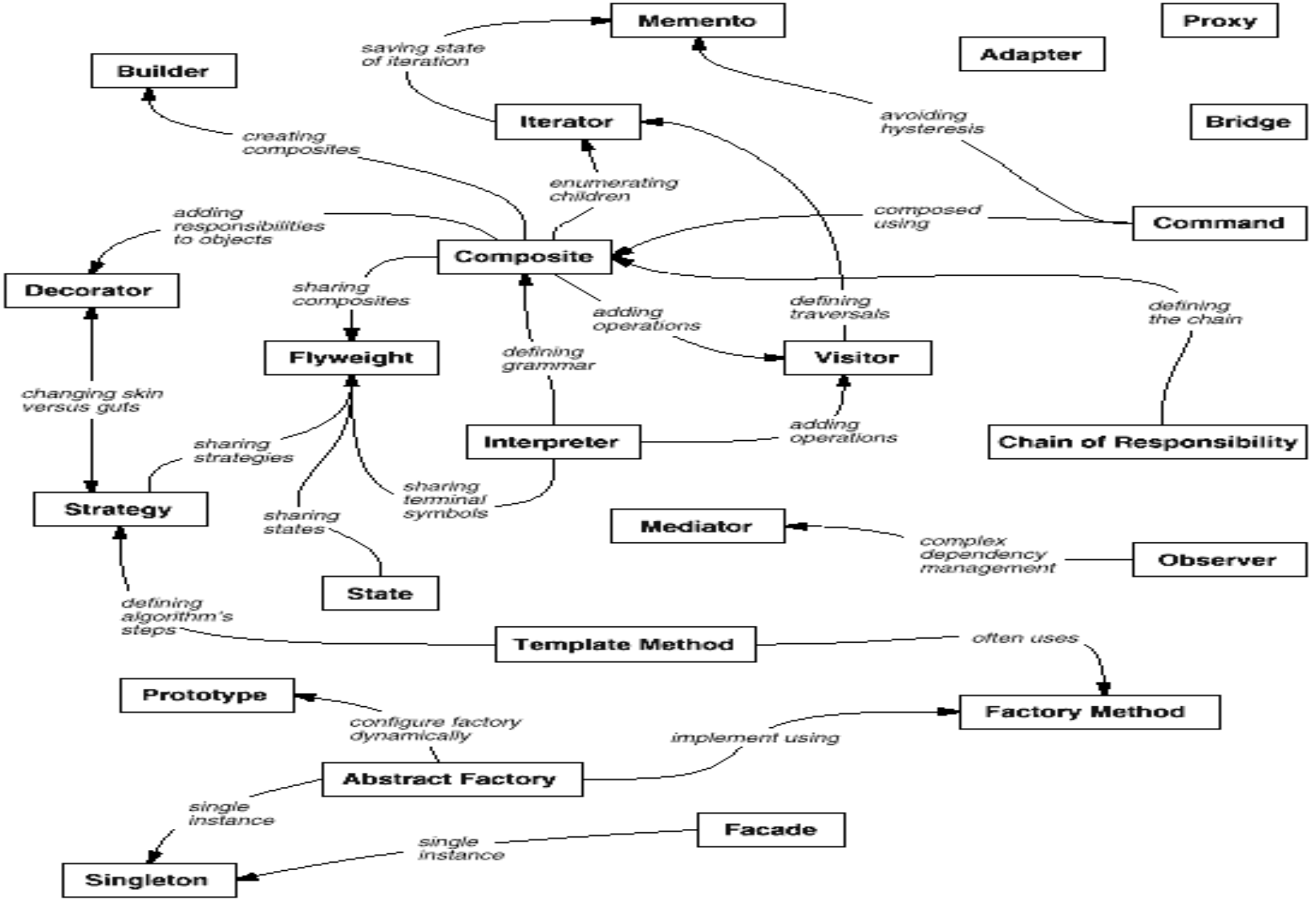
repetition ::= expression '*'

literal ::= 'a' | 'b' | 'c' | ... { 'a' | 'b' | 'c' | ... }*

“Interpret” pro reg. výrazy



Obrázek převzat z [GoF]


Memento
Proxy
saving state of iteration
Builder
Adapter
Iterator
avoiding hysteresis
Bridge
creating composites
enumerating children
adding responsibilities to objects
composed using
Command
Composite
Decorator
sharing composites
adding operations
defining traversals
defining the chain
Flyweight
defining grammar
Visitor
changing skin versus guts
sharing strategies
Interpreter
adding operations
Chain of Responsibility
Strategy
sharing terminal symbols
sharing states
Mediator
complex dependency management
Observer
State
defining algorithm's steps
Template Method
often uses
Prototype
Factory Method
configure factory dynamically
implement using
Abstract Factory
single instance
Facade
single instance
Singleton

The End

Vojtěch Merunka

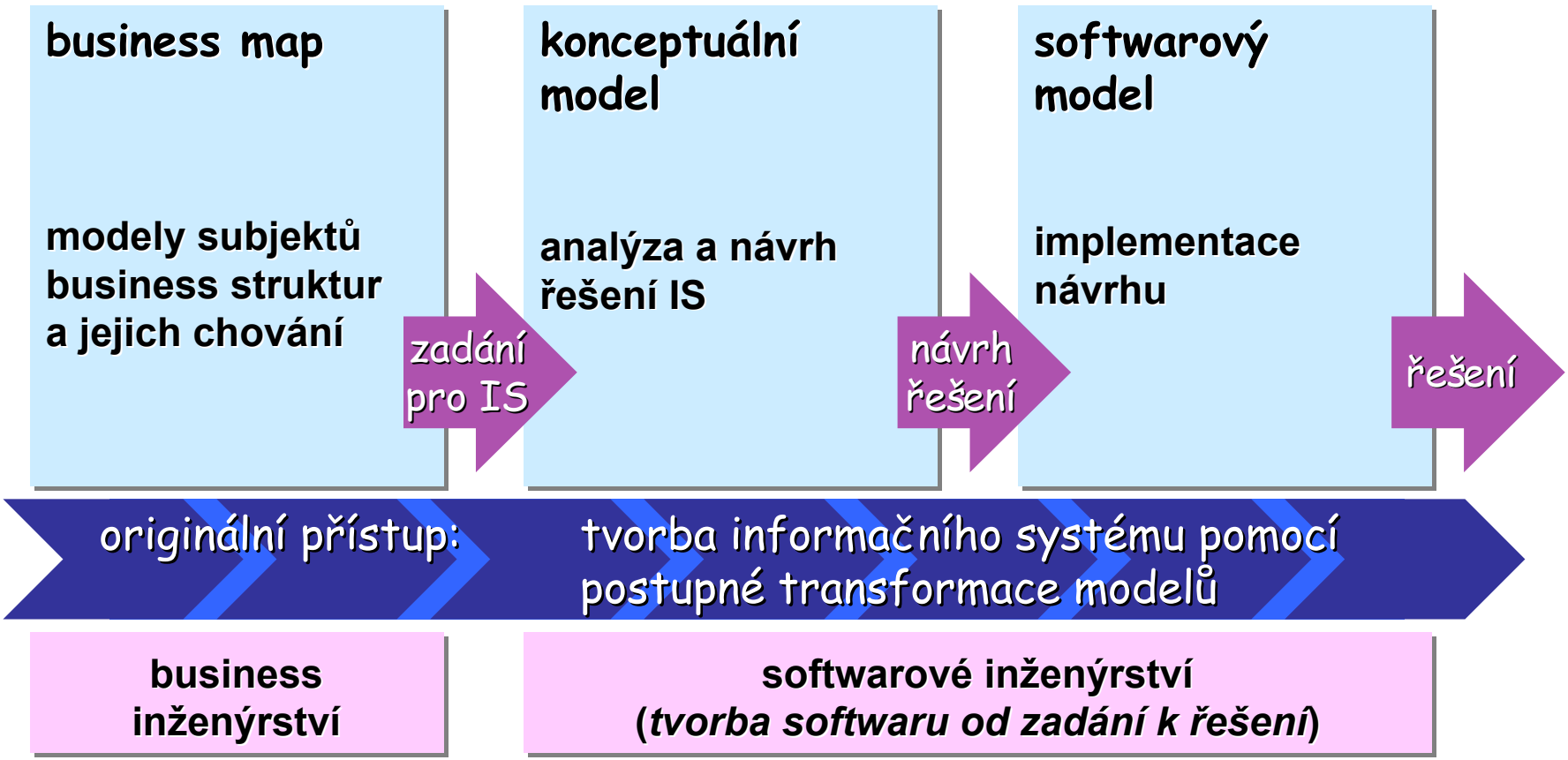
Metoda BORM

BORM – Business and Object Relation Modeling

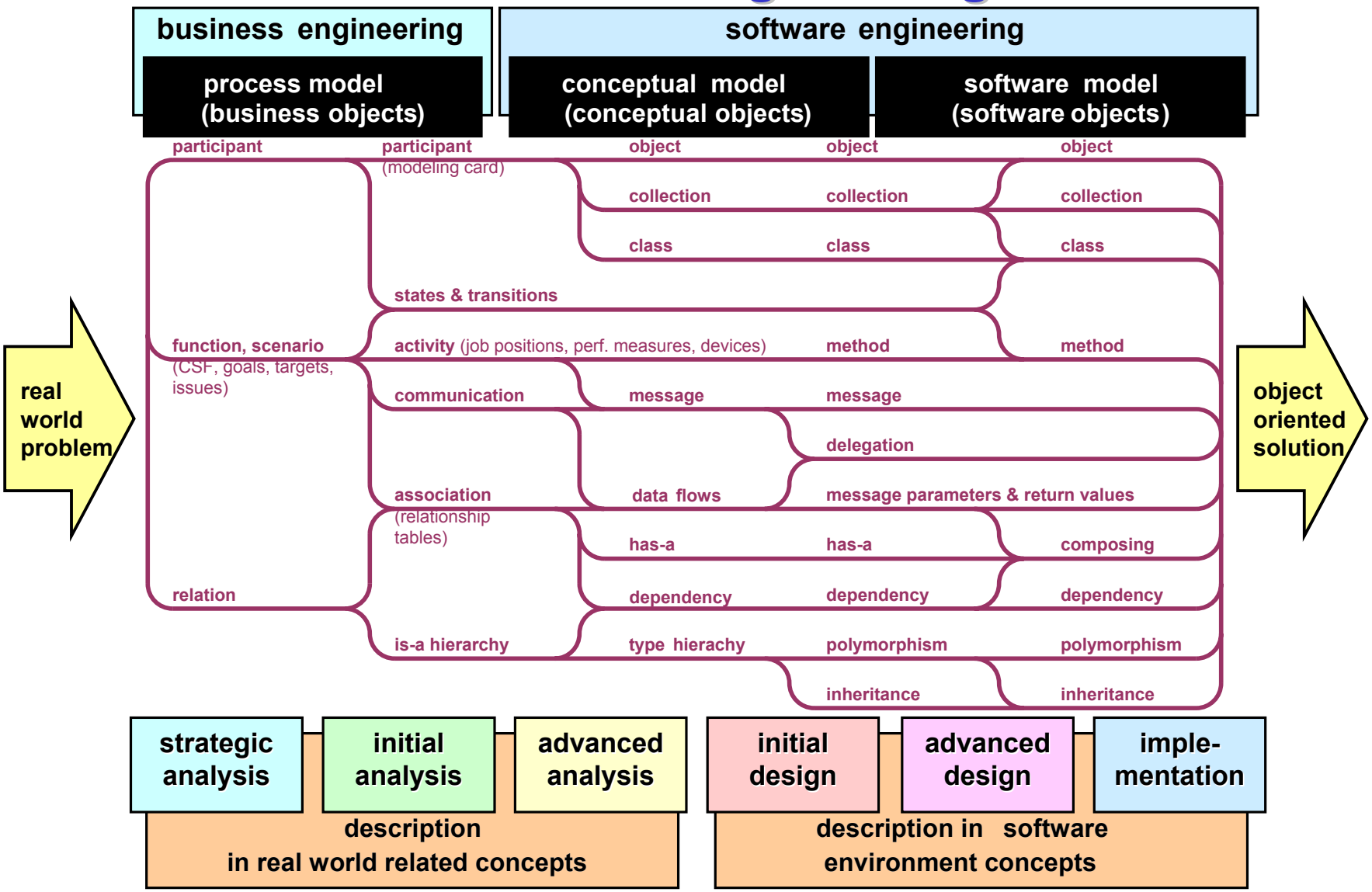
- ✓ Práce na BORMu začaly na počátku 90. let ve výzkumném projektu VAPPIENS Britské rady (Know-How Fund of the British Council).
- ✓ Metoda je od roku 1996 vyvíjena s podporou firmy Deloitte, kde se také používá.
- ✓ Podrobný popis BORMu lze nalézt v knize Carda, Merunka, Polák: Umění systémového návrhu - objektově orientovaná tvorba informačních systémů pomocí původní metody BORM, Grada 2003.

Pro BORM se doposud používal CASE nástroj Metaedit® finské firmy Metacase Ltd nebo MS Visio. Craft.CASE se používá od roku 2005.

BORM - kontext celé metody

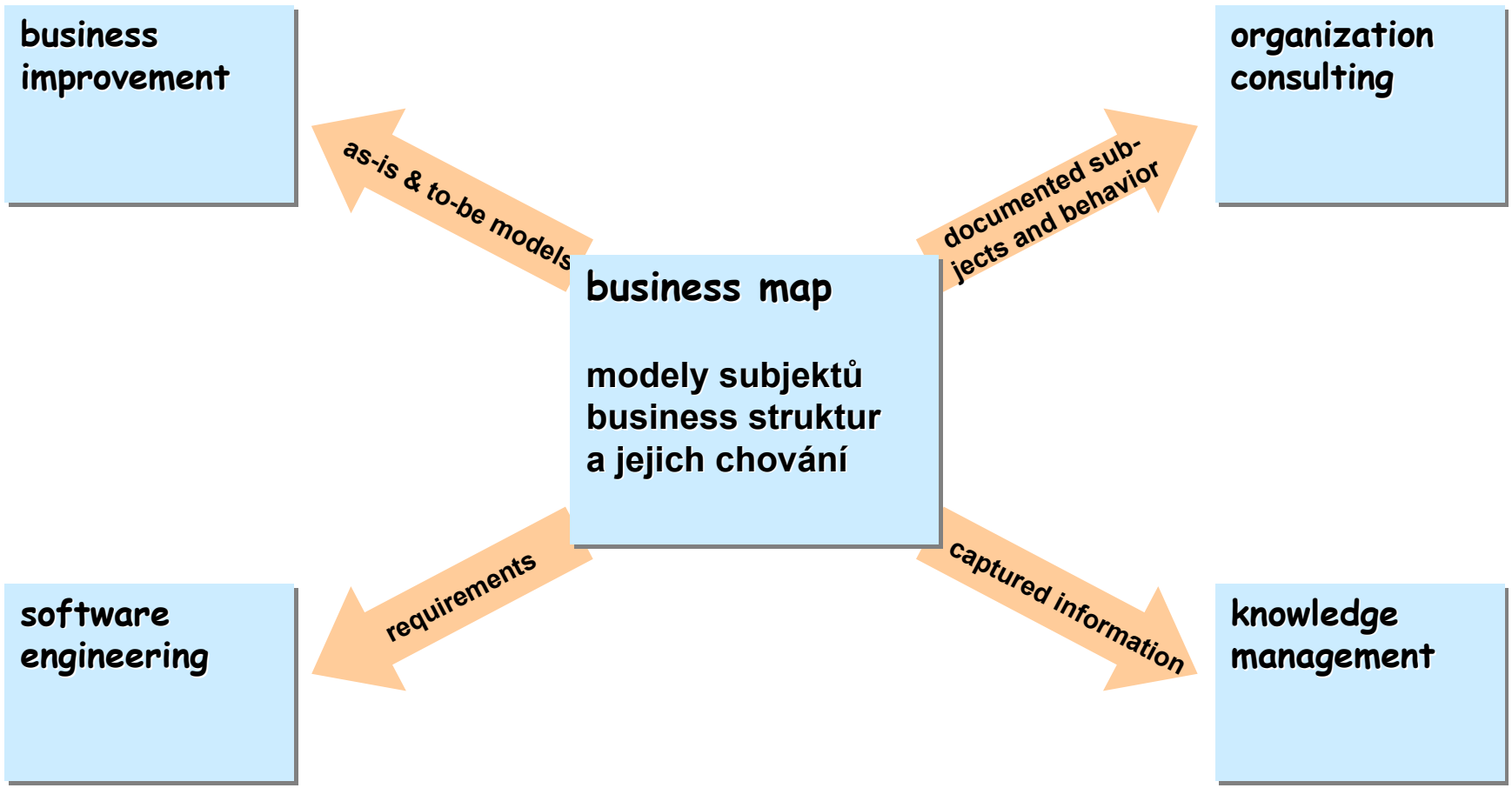


BORM Information Engineering Process

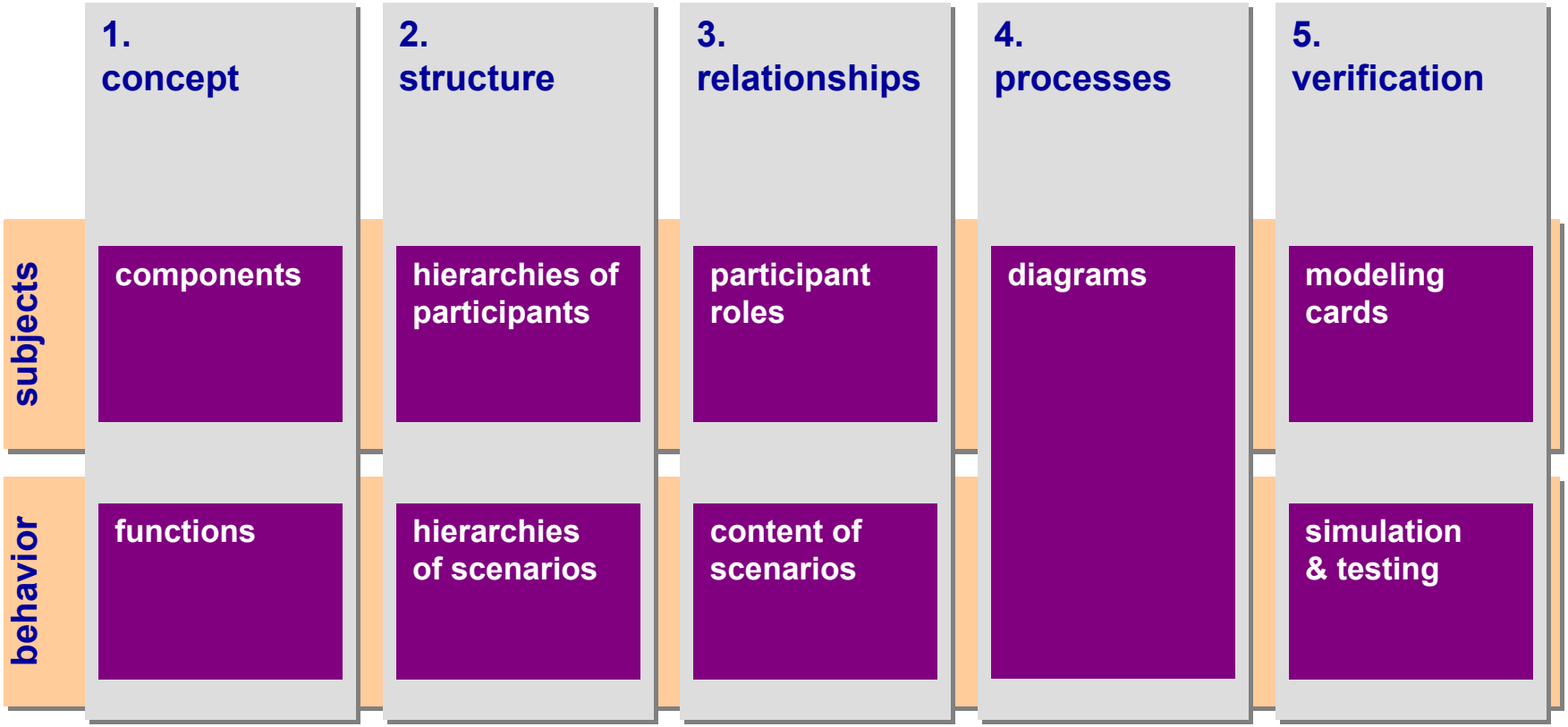


Business Map

„business analysis and design method based on combination of object-oriented approach and process modeling“

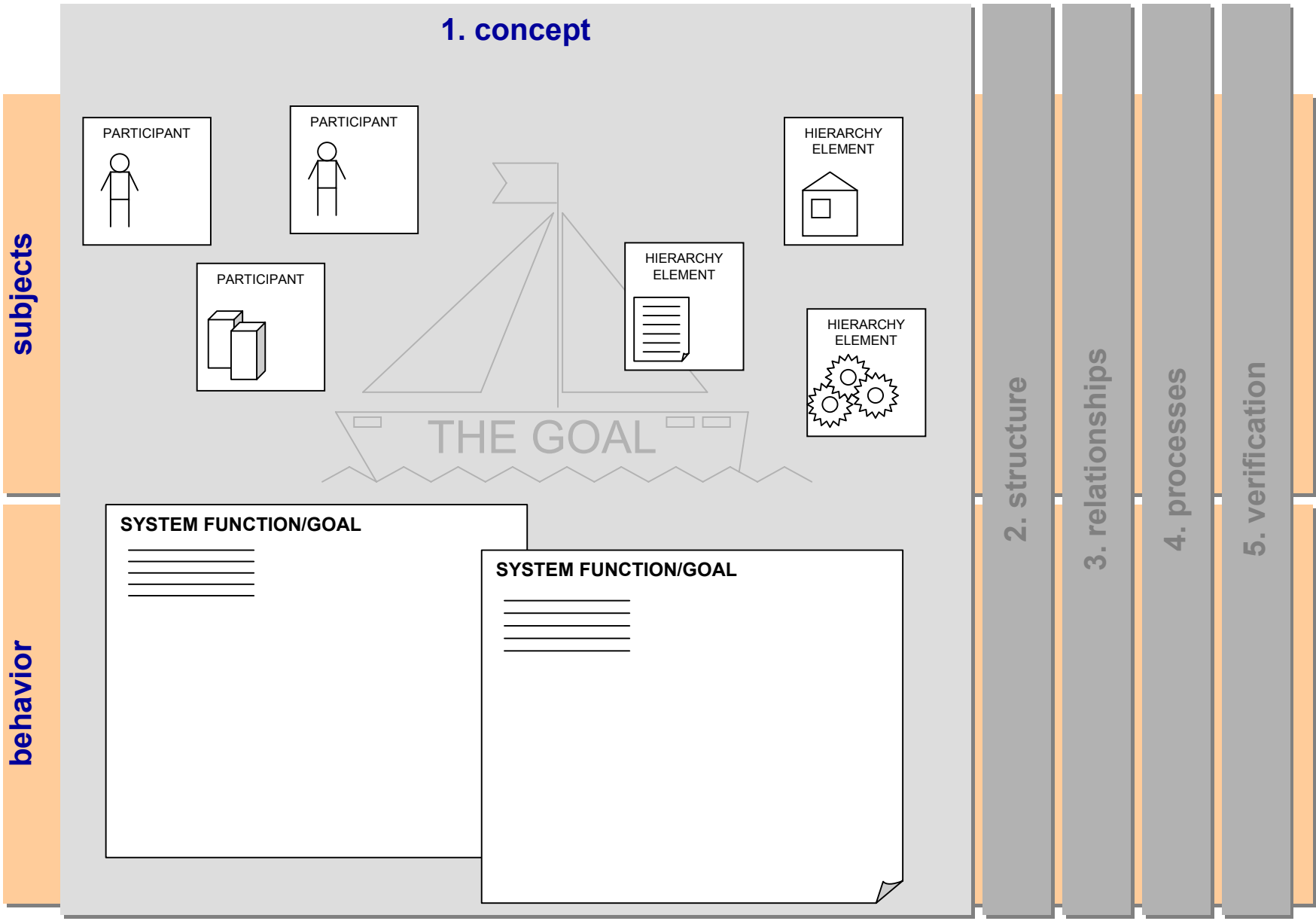


BORM - System Map - framework

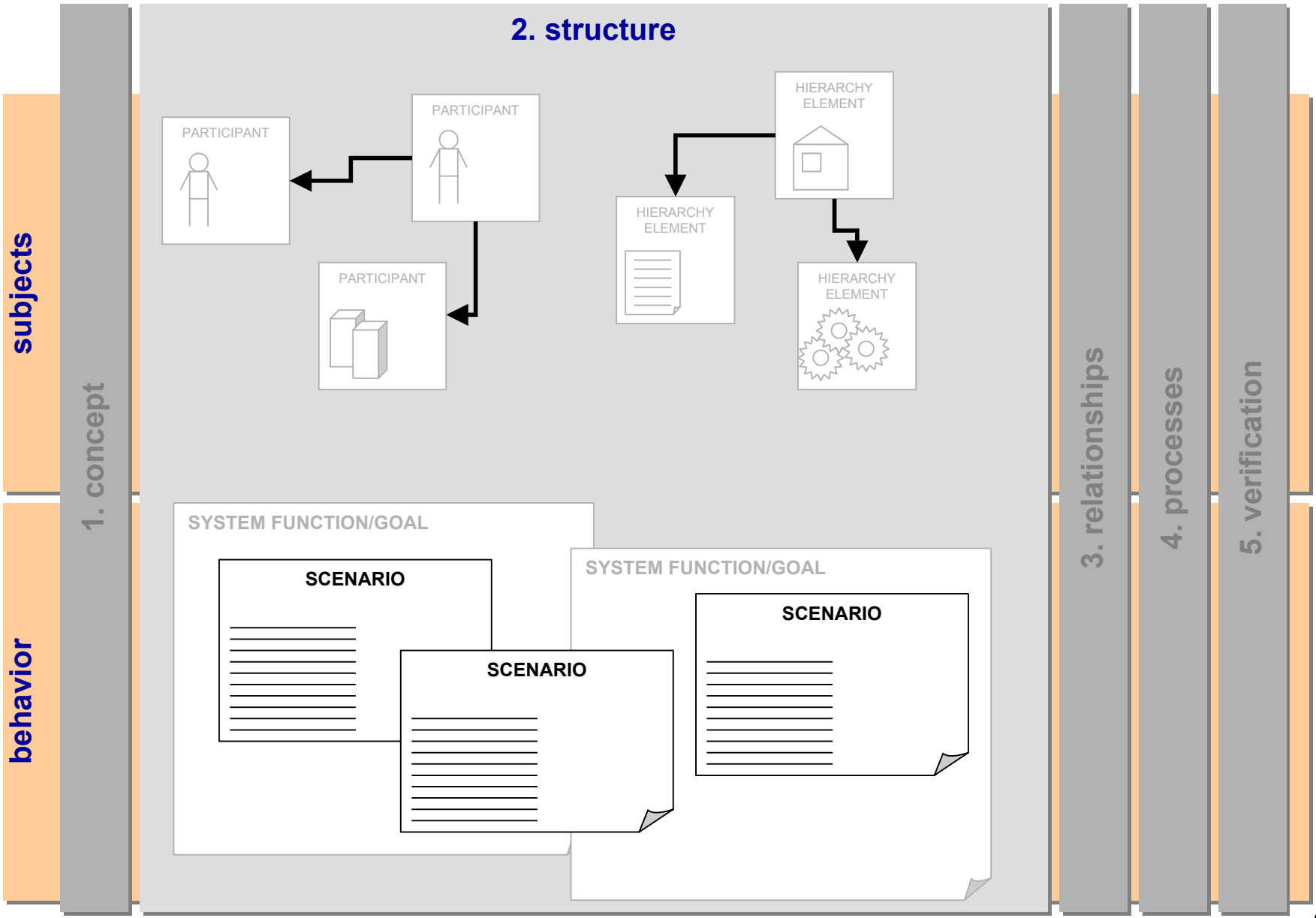


explanation: phase | thread (independent layer) | method

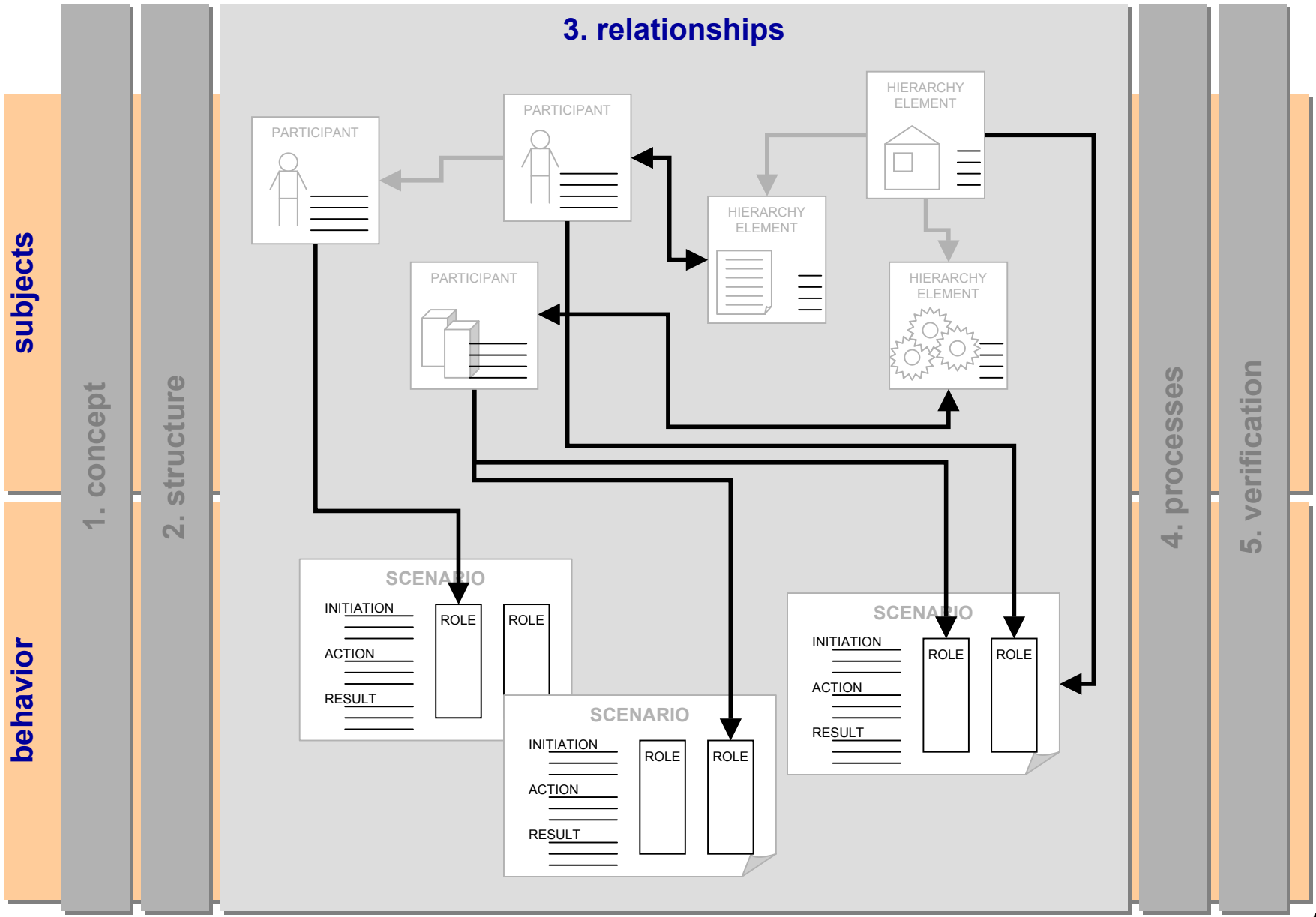
Business Map - What to do/not to do: Concept



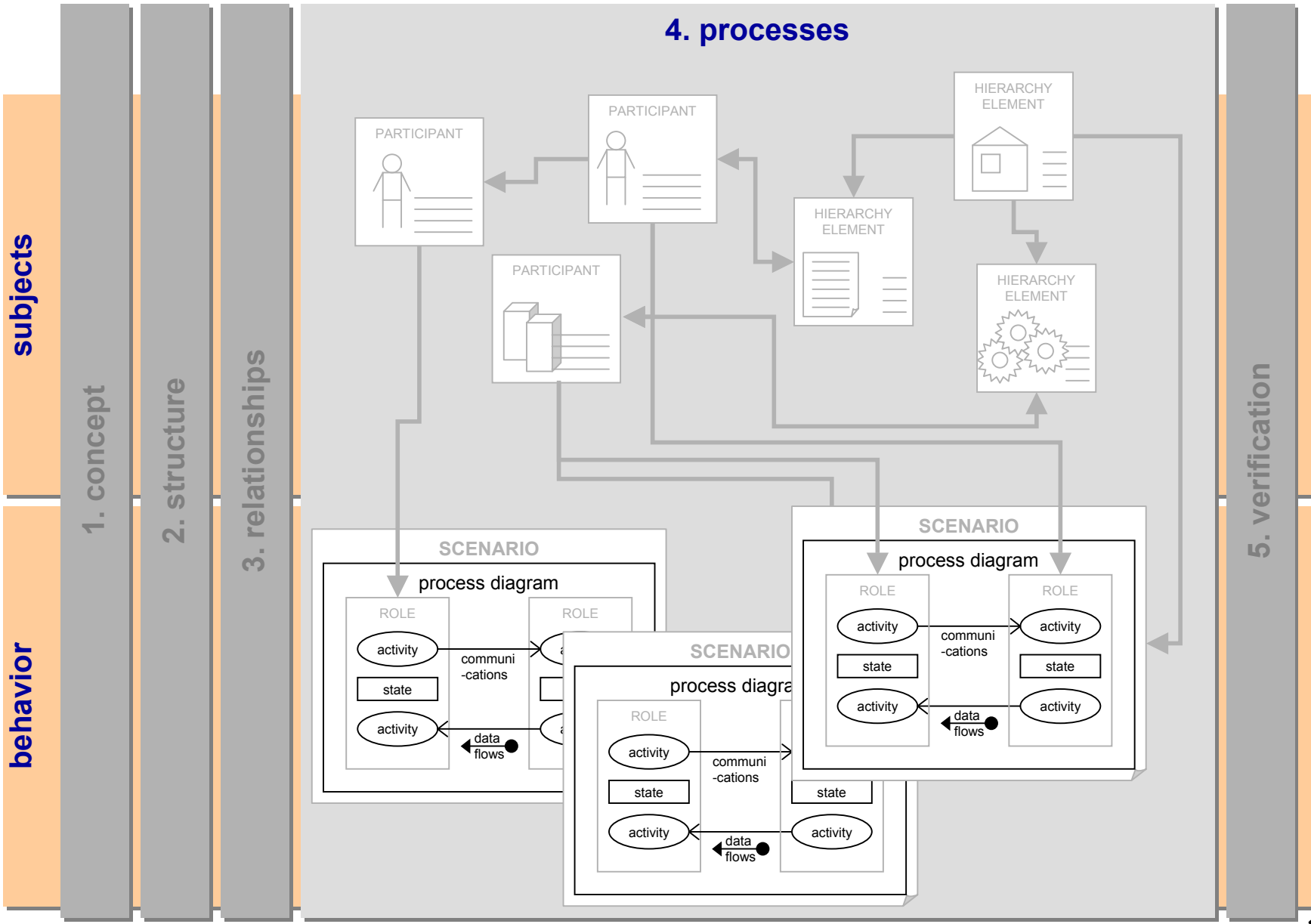
Business Map - How to structure: Structure



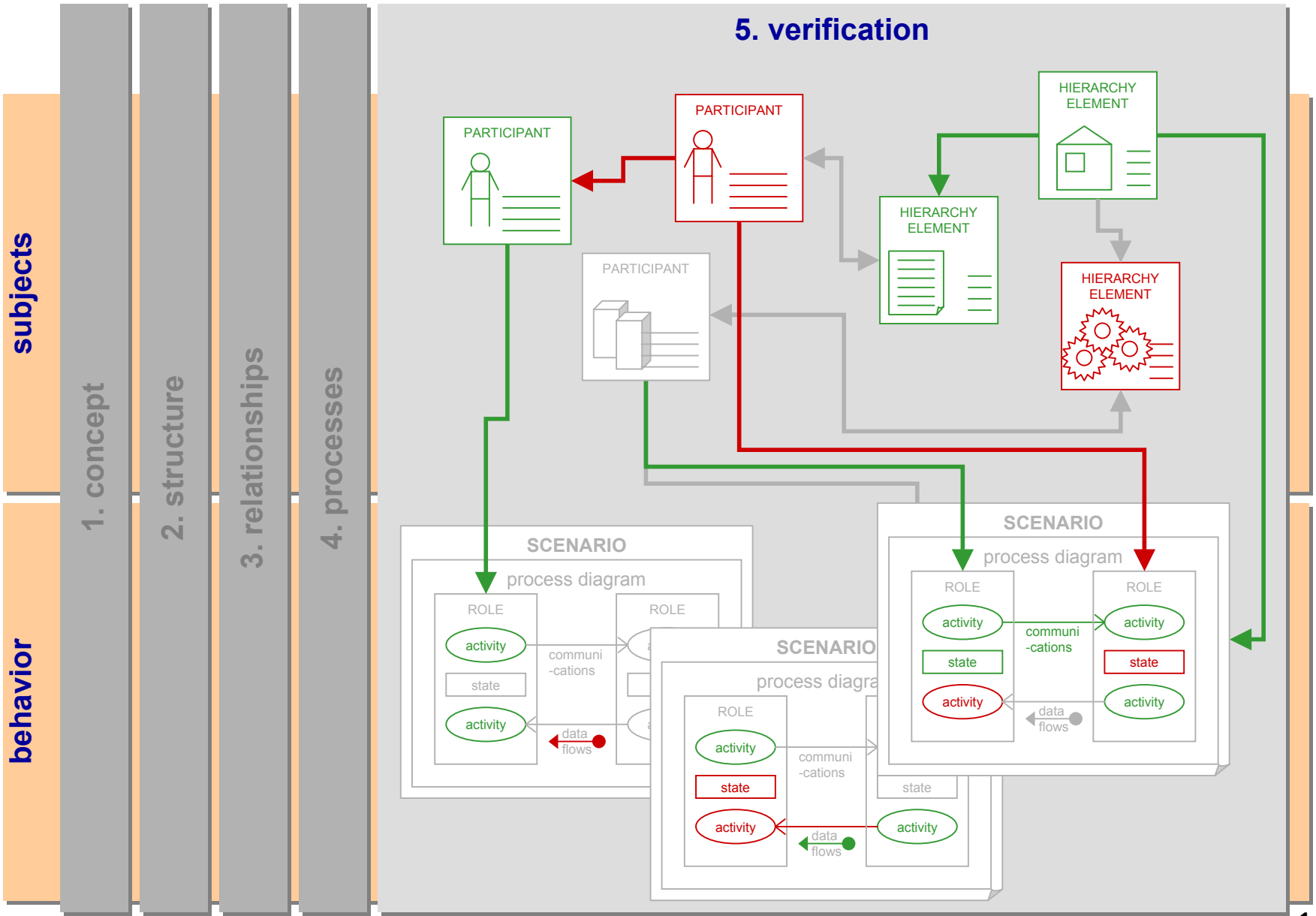
Business Map - Describe detail: Relationships



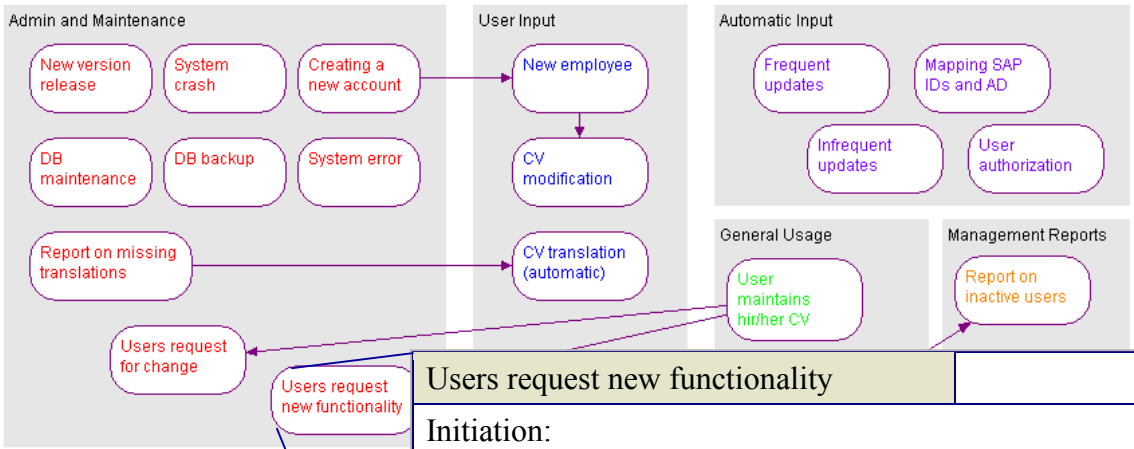
Business Map - visualize the model: Processes



Business Map - simulate and test the model: Verification



Funkce, Scénáře, Participanti a Modelové karty



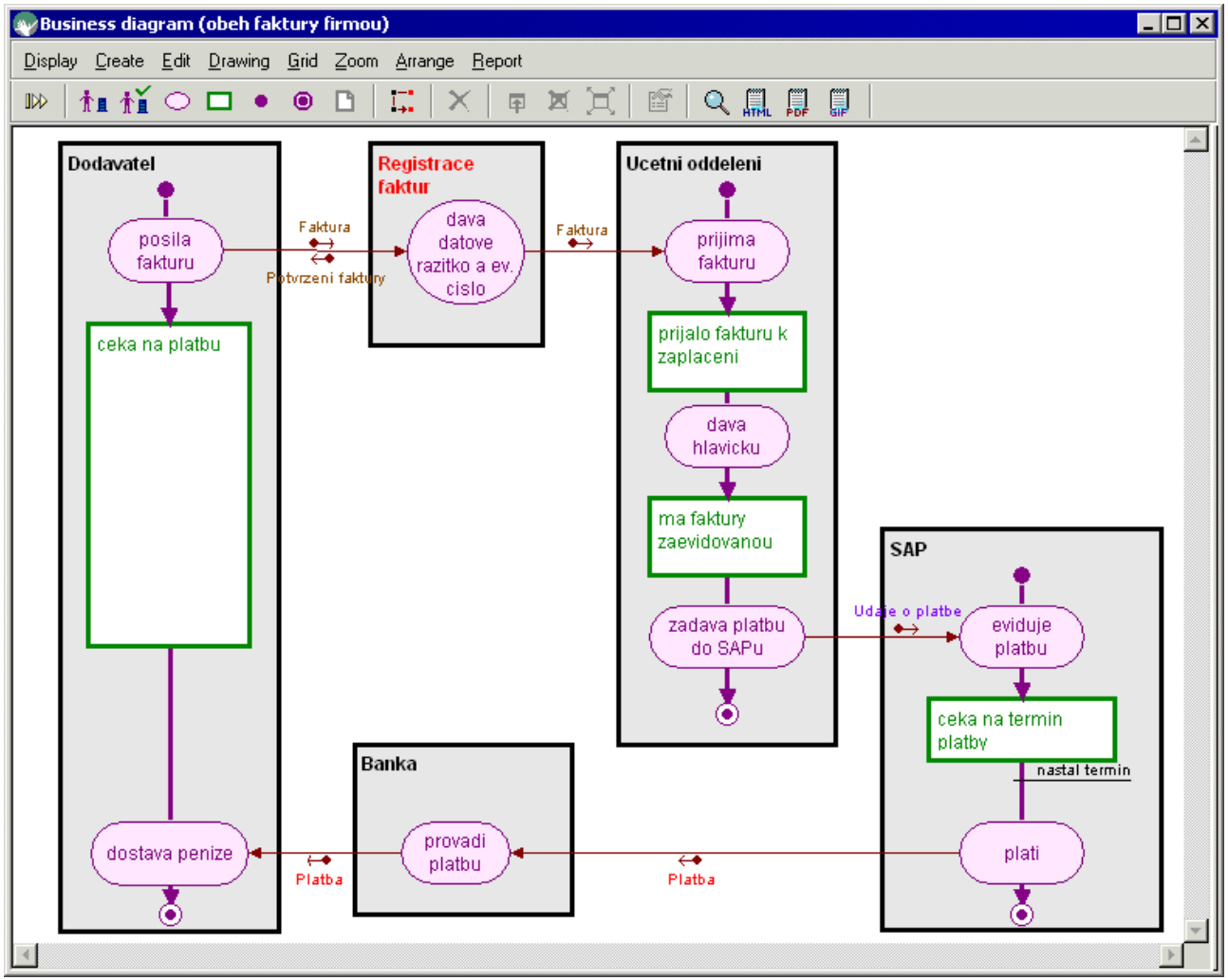
Users request new functionality	Derived from: Admin and Maintenance
Initiation: User requests new functionality.	
Action: IT Support evaluates the requests and accumulates relevant information for future development.	
Result: Developer periodically receives requests for upgrades (accumulated, not one-by-one).	
Developer	
IT Support	

Developer	**CVDB**	**IT Support**	**User**
New version release	x	x	
System error	x	x	
Users request for change		x	x
Users request new functionality		x	x

Procesní diagram

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

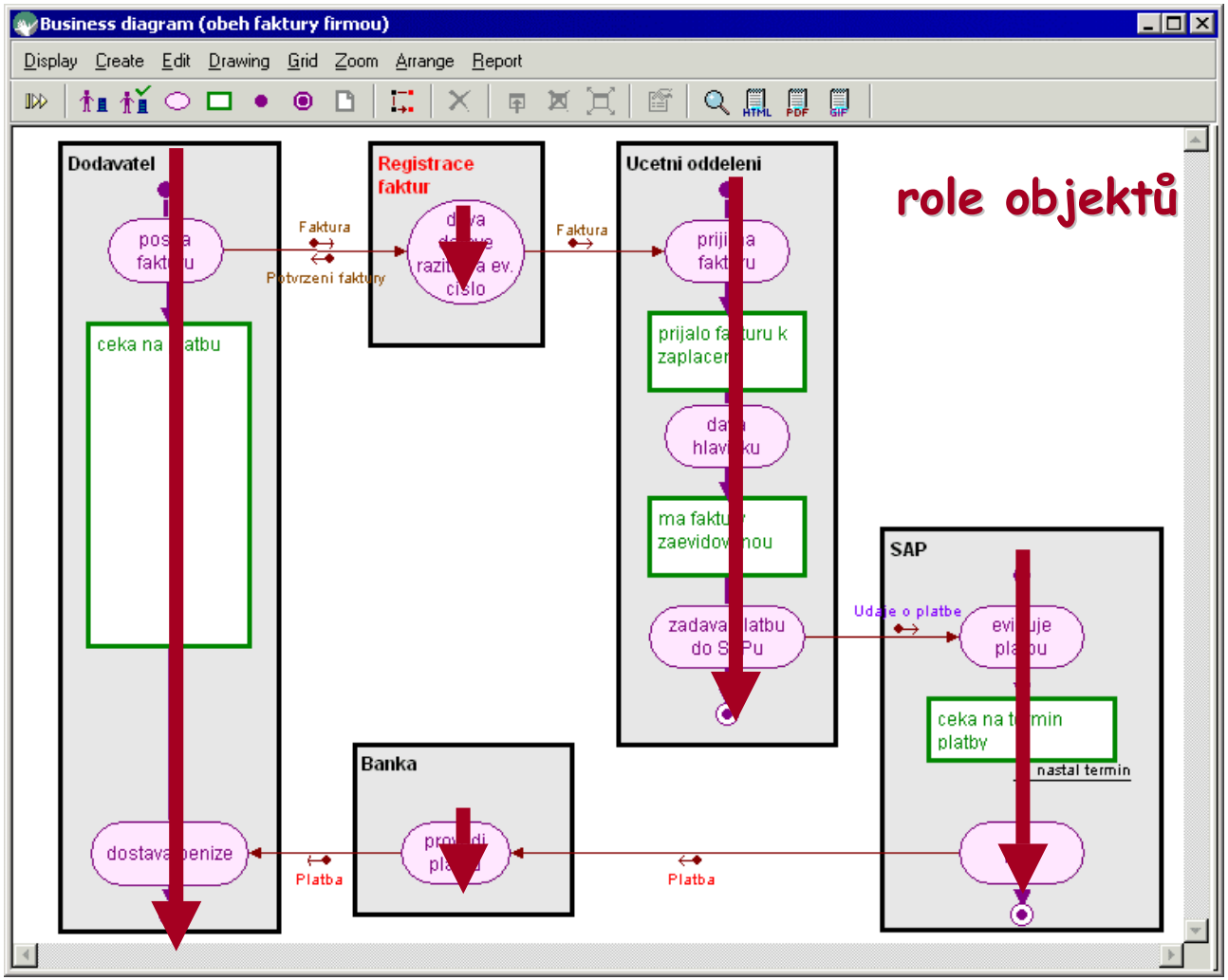
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Procesní diagram

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

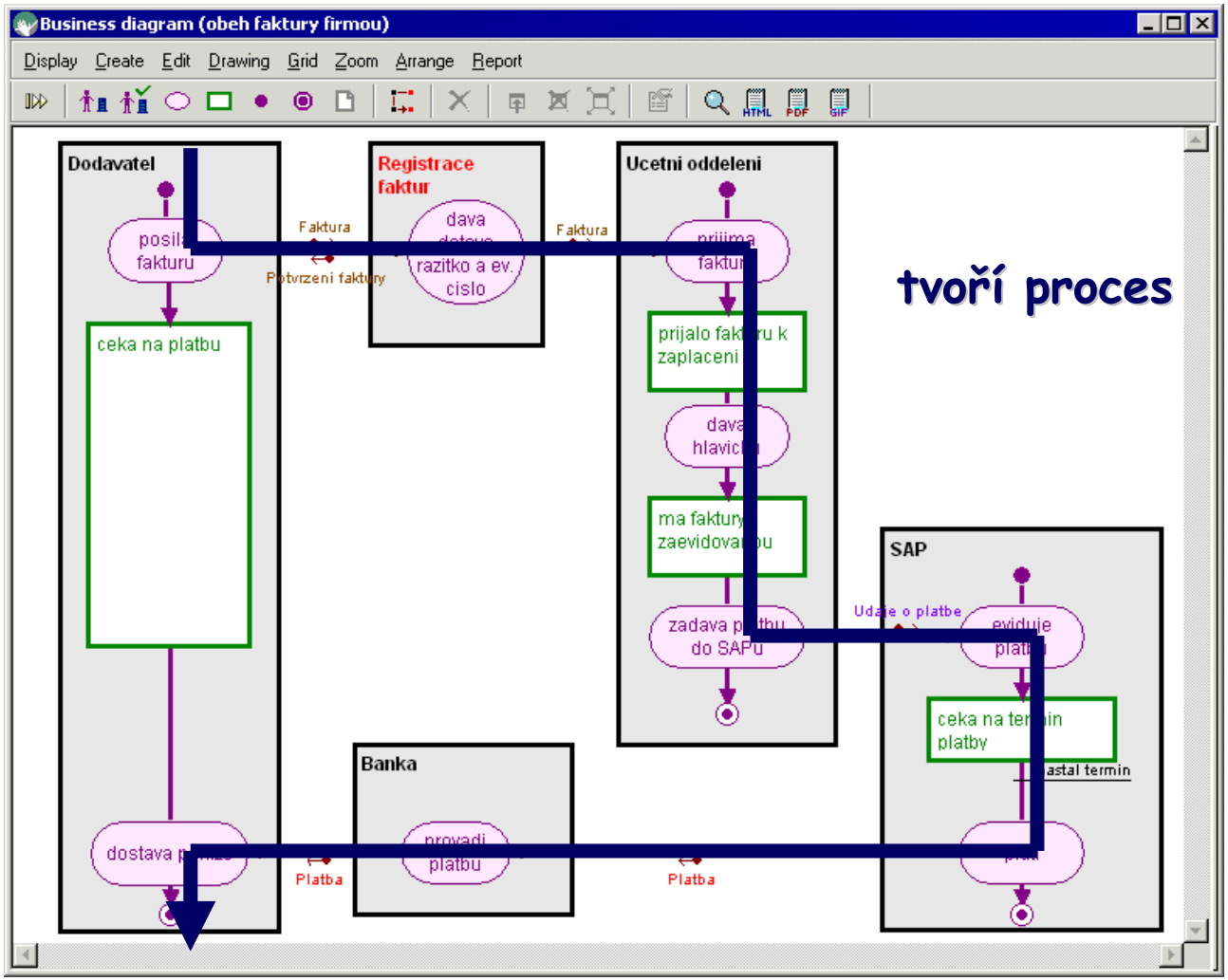
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Procesní diagram

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

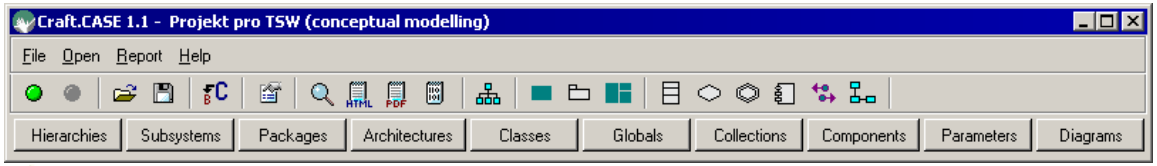
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Antonín Carda,
Vojtěch Merunka,
Vladimír Vlachovský

http://www.craftcase.com

Projekt Craft.CASE



Craft.CASE je původní český modelovací a analytický CASE nástroj podporující metodu BORM®, která je založena na kombinaci objektově orientovaného přístupu a procesního modelování.

Nástroj vzniká ve firmě e-Fractal s.r.o.

Zadání vychází ze dvou potřeb:

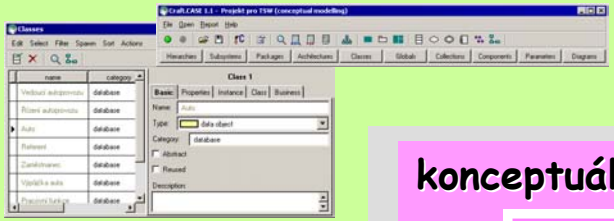
1) Jednoduše ovladatelný a na prostředky počítače nenáročný.
2) Modelovací nástroj přesně šitý na míru metodě BORM, který je částečně konfigurovatelný, dokáže procesy simulovat a generuje výstupní dokumentaci.

Program je vyvíjen v prostředí VisualWorks/Smalltalk a je určen pro použití ve Windows 2000 a XP.

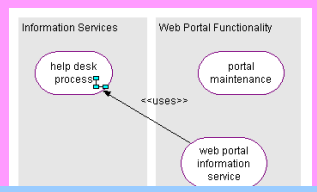
http://www.craftcase.com

Shrnutí - projektování pomocí Craft.CASE

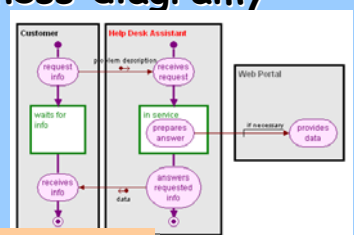
společná databáze



business model



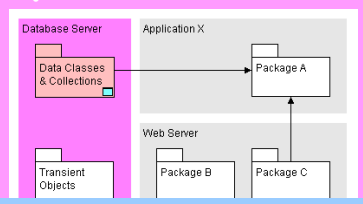
business diagramy



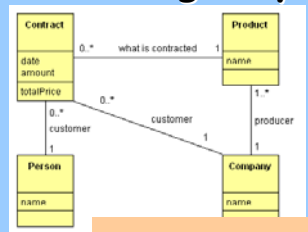
simulátor

transformace

konceptuální model



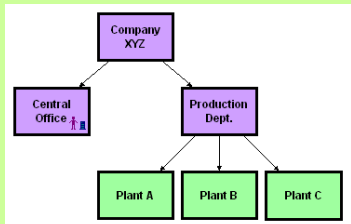
konceptuální diagramy



generátor kódu

vazby

pomocné hierarchie

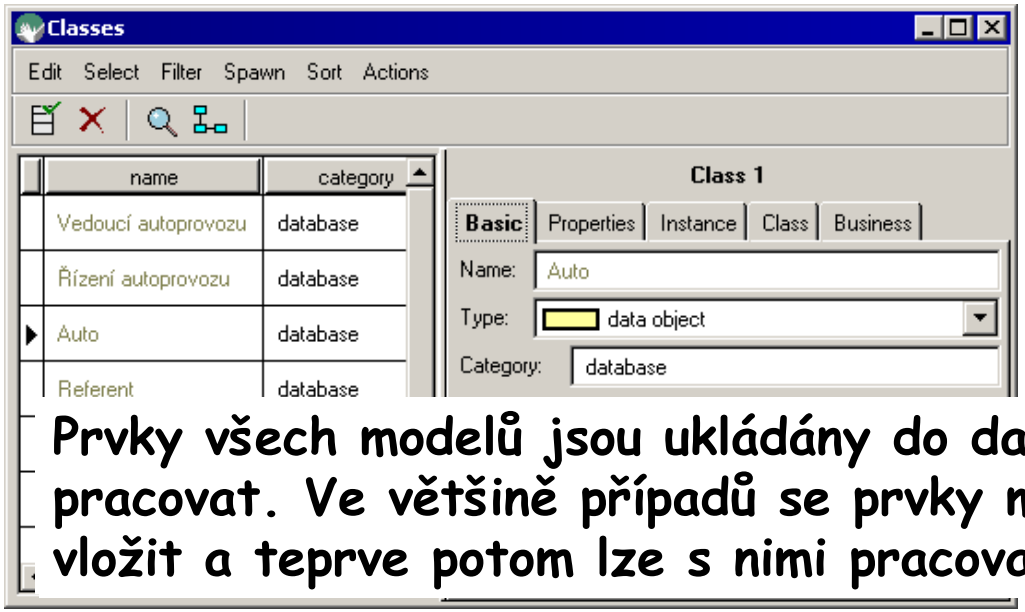


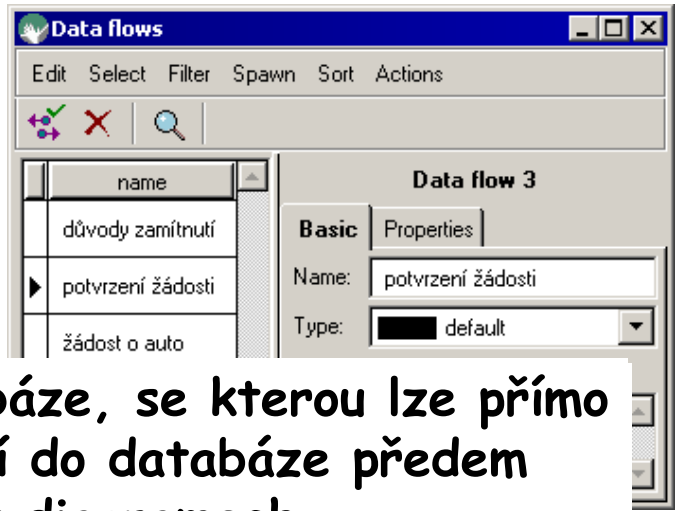
vazby

modelování zadání pro IS a jeho prostředí

analýza a návrh IS

Craft.CASE





Prvky všech modelů jsou ukládány do databáze, se kterou lze přímo pracovat. Ve většině případů se prvky musí do databáze předem vložit a teprve potom lze s nimi pracovat v diagramech.

Výstupní dokumentace je tvořena hypertexty ve formátu HTML a také PDF a obsahuje:

1. seznamy prvků z databáze
2. diagramy
3. simulační záznamy
4. modelové karty
   (= tabulky s křížovými referencemi)

5.2.5. SAP

Collaborators in diagram wit firmou':
start: eviduje platbu
ceka na termin platby: plati

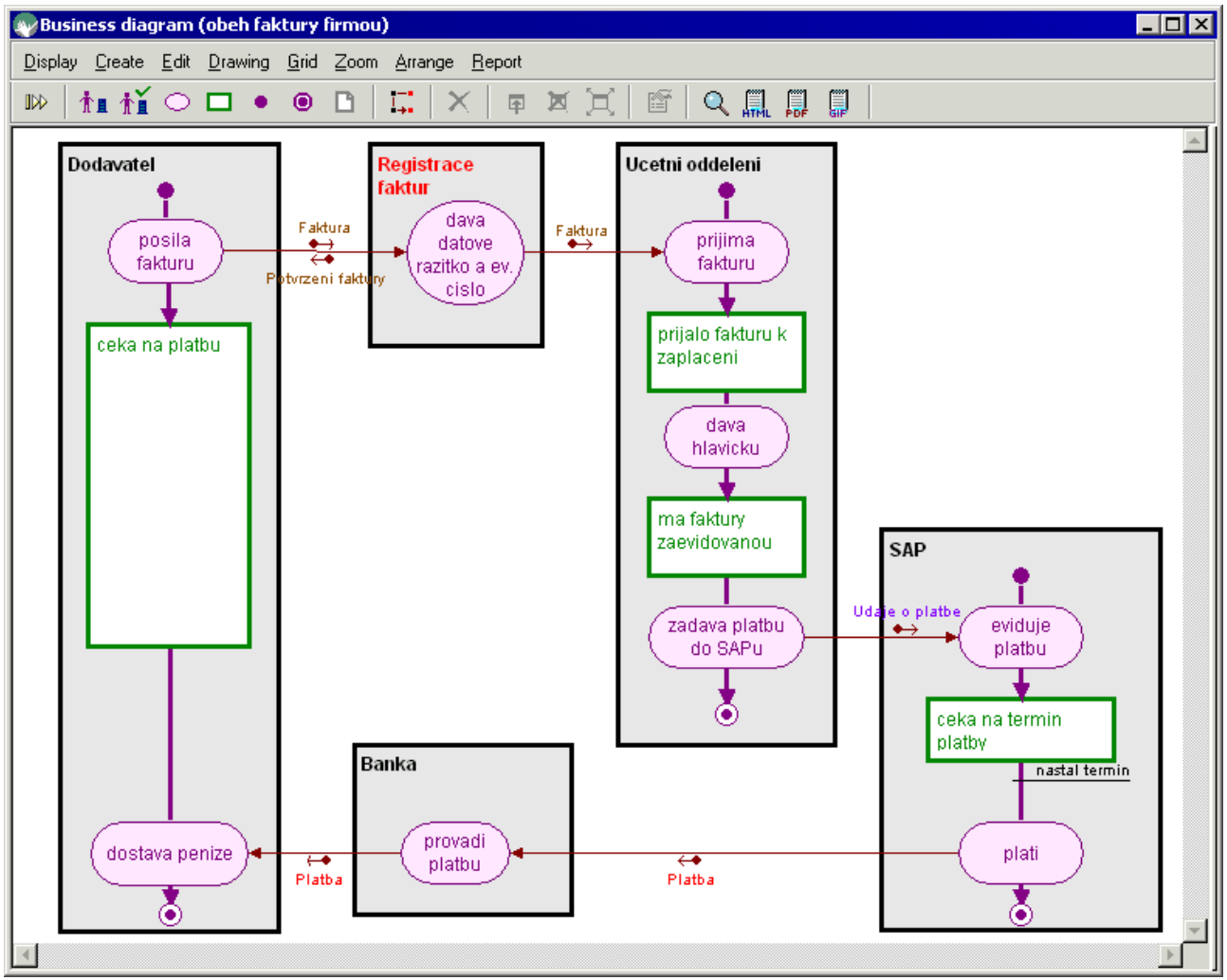
5.2.6. Vedoucí autoprovozu

Collaborators in diagram with id 'výpůjčka auta':	Auto	Referent	Vedoucí
čeká na vrácení auta: přebírá vrácené auto	<<		>>
má žádost o auto: přiděluje auto	>>	>>	
start: přijímá žádost o vydání auta		<<	
má žádost o auto: ruší výběr auta			<<
má žádost o auto: vybírá auto			

Business Map - Procesní diagram - Simulace

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

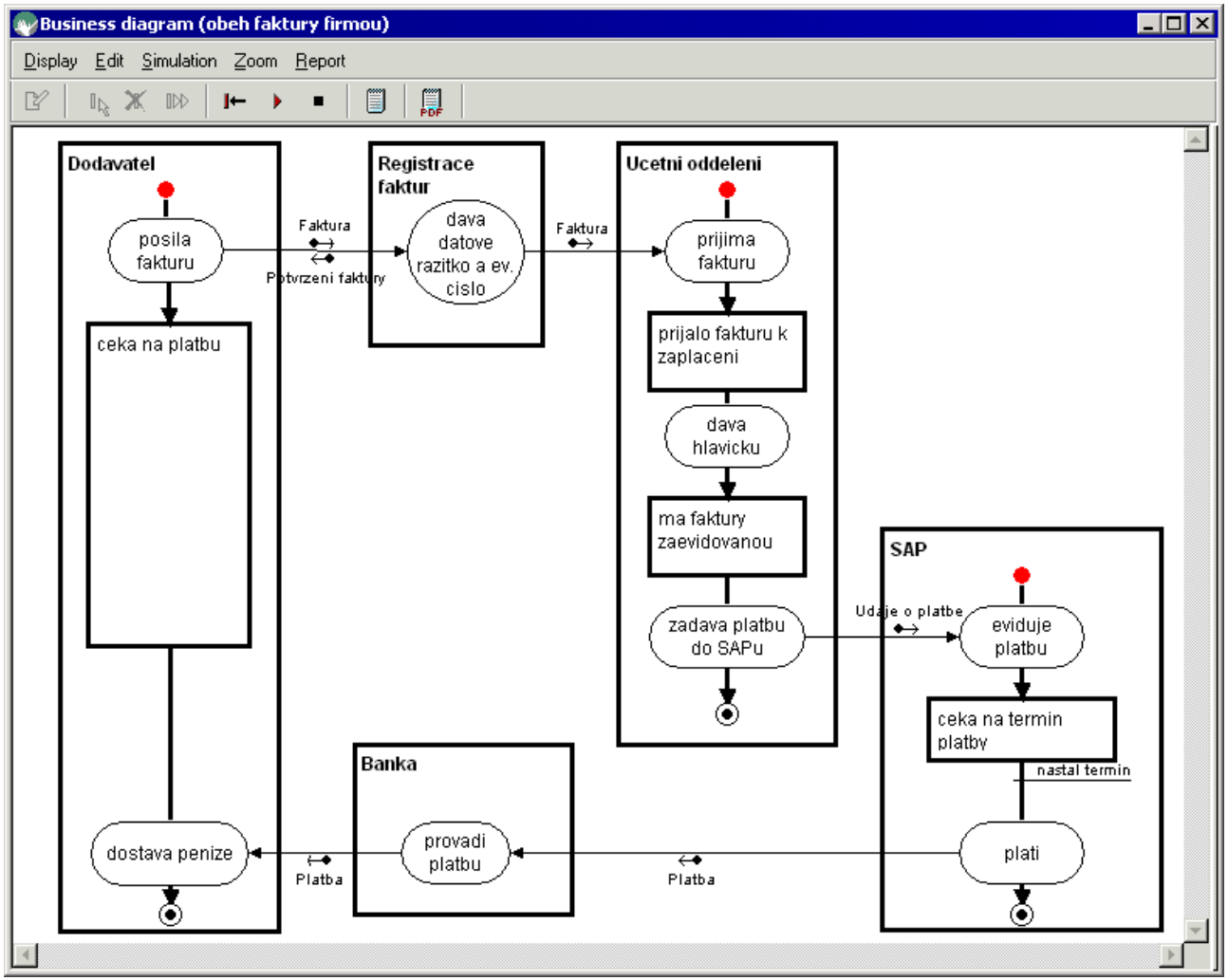
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Business Map - Procesní diagram - Simulace

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

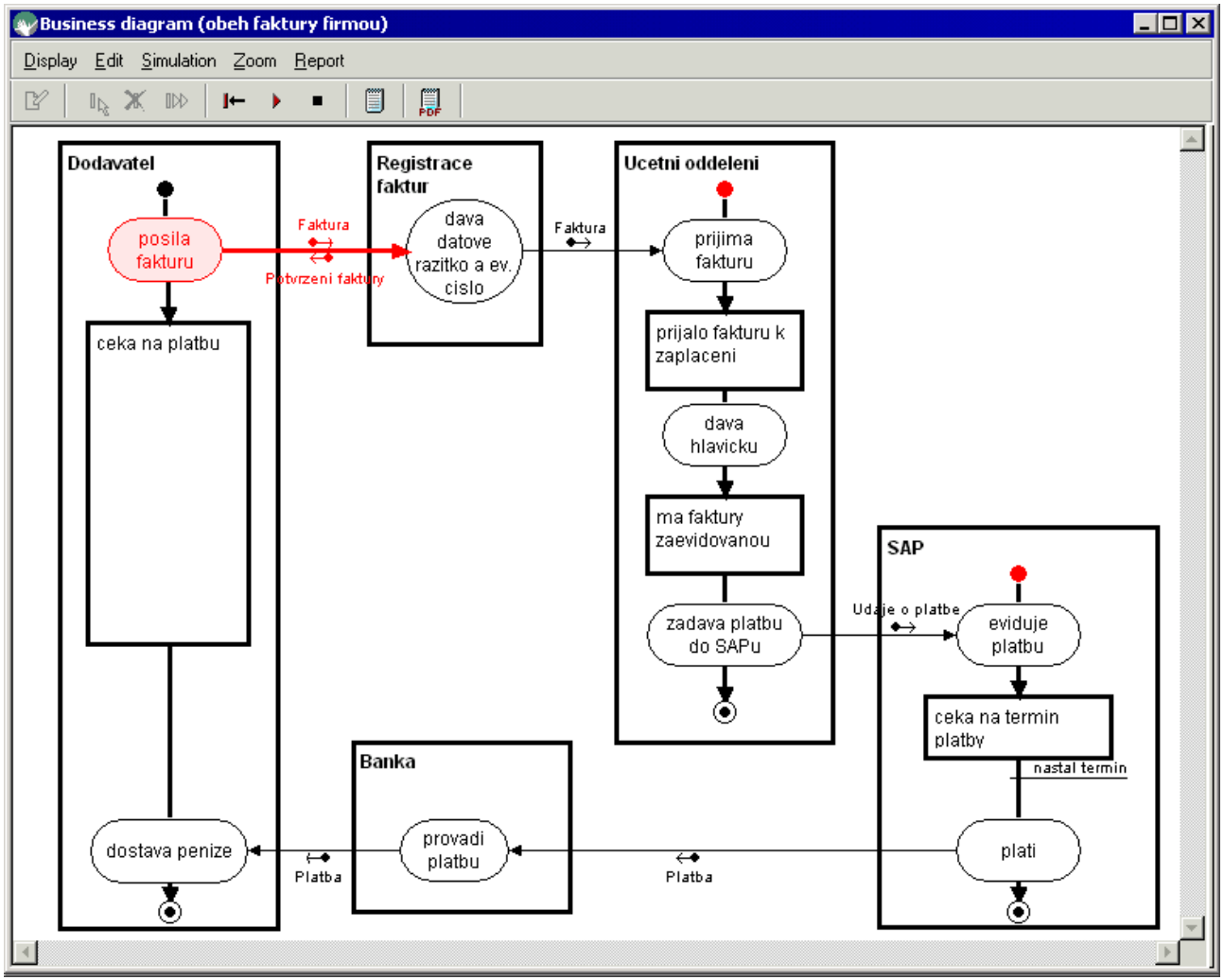
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Business Map - Procesní diagram - Simulace

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

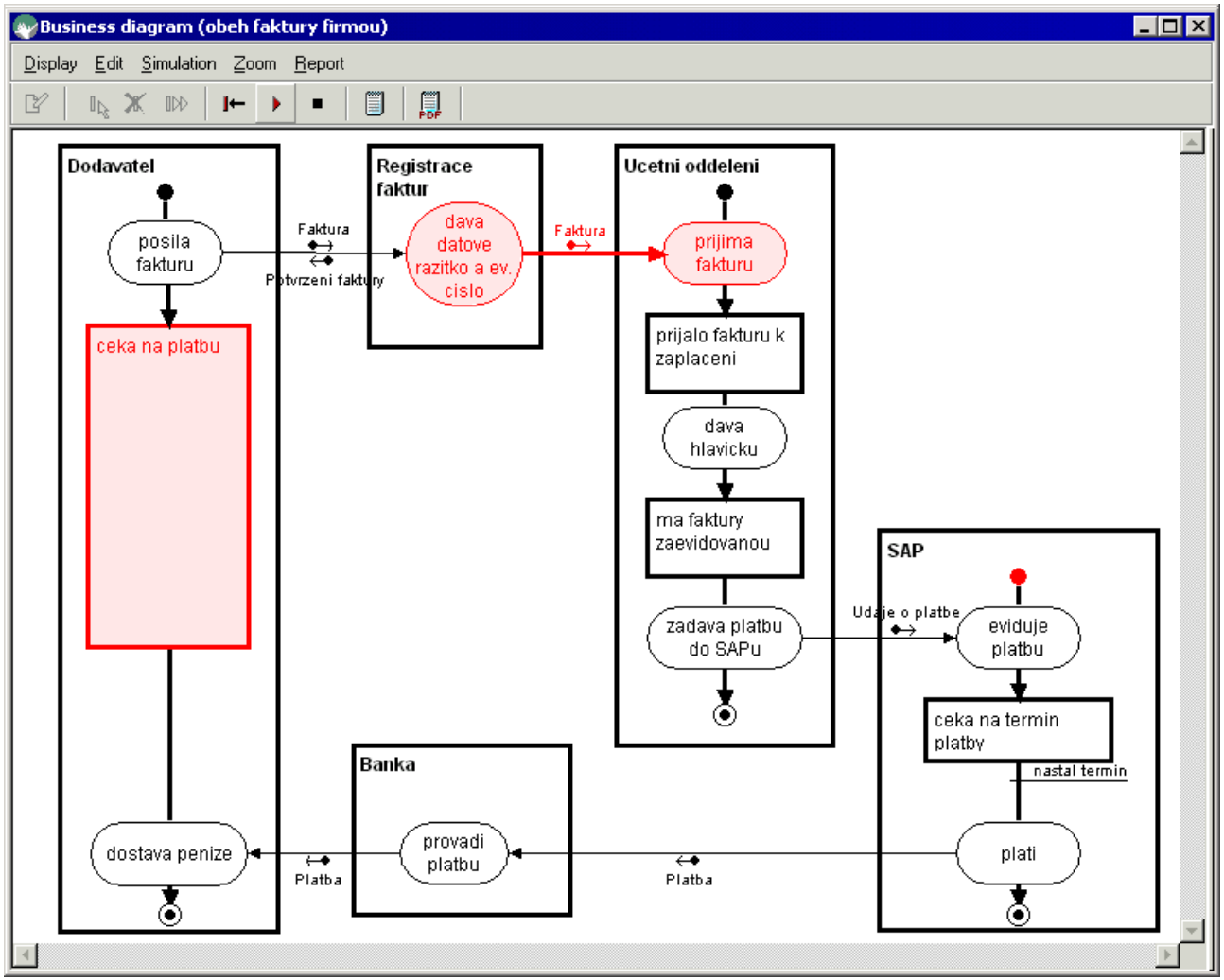
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Business Map - Procesní diagram - Simulace

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

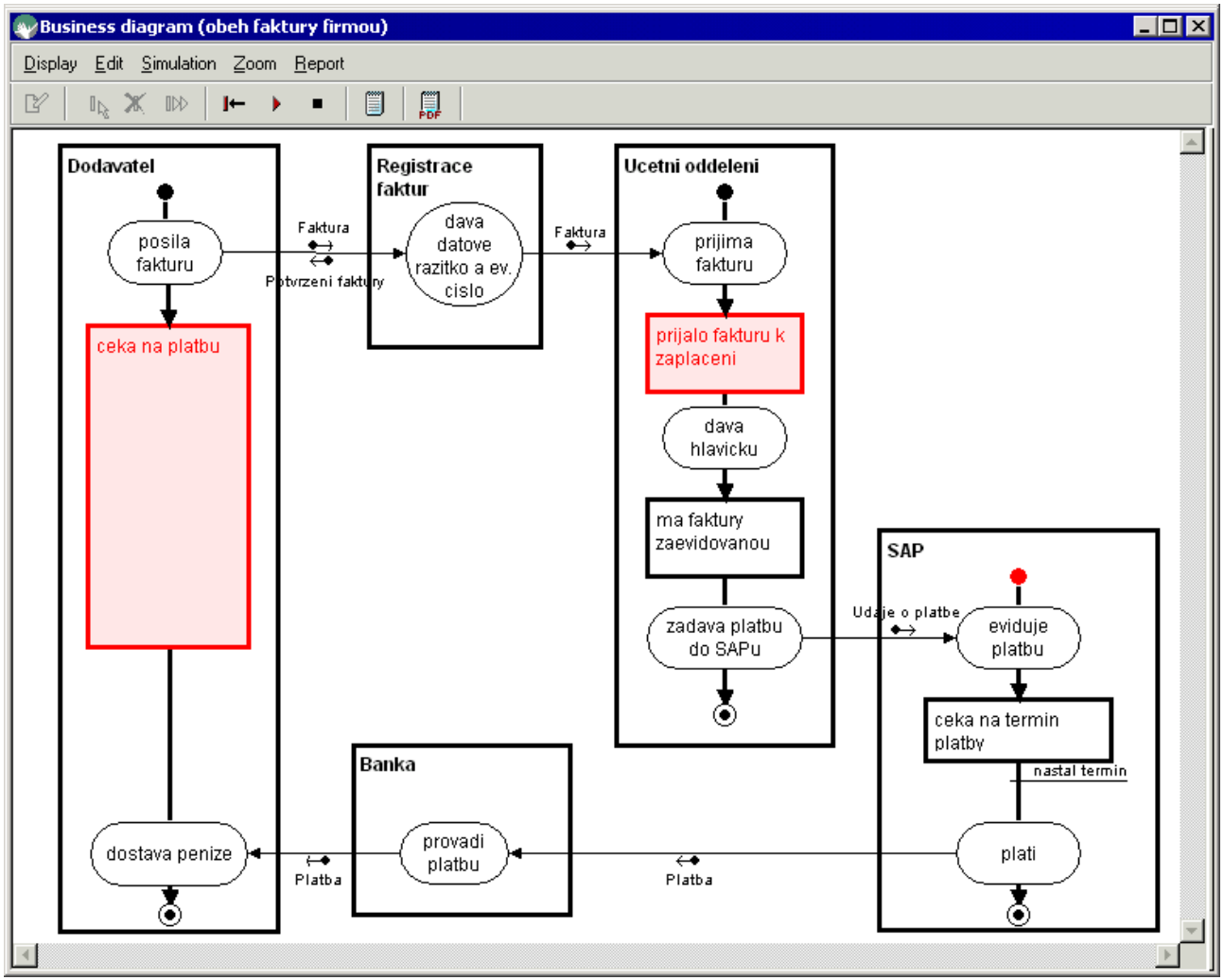
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Business Map - Procesní diagram - Simulace

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

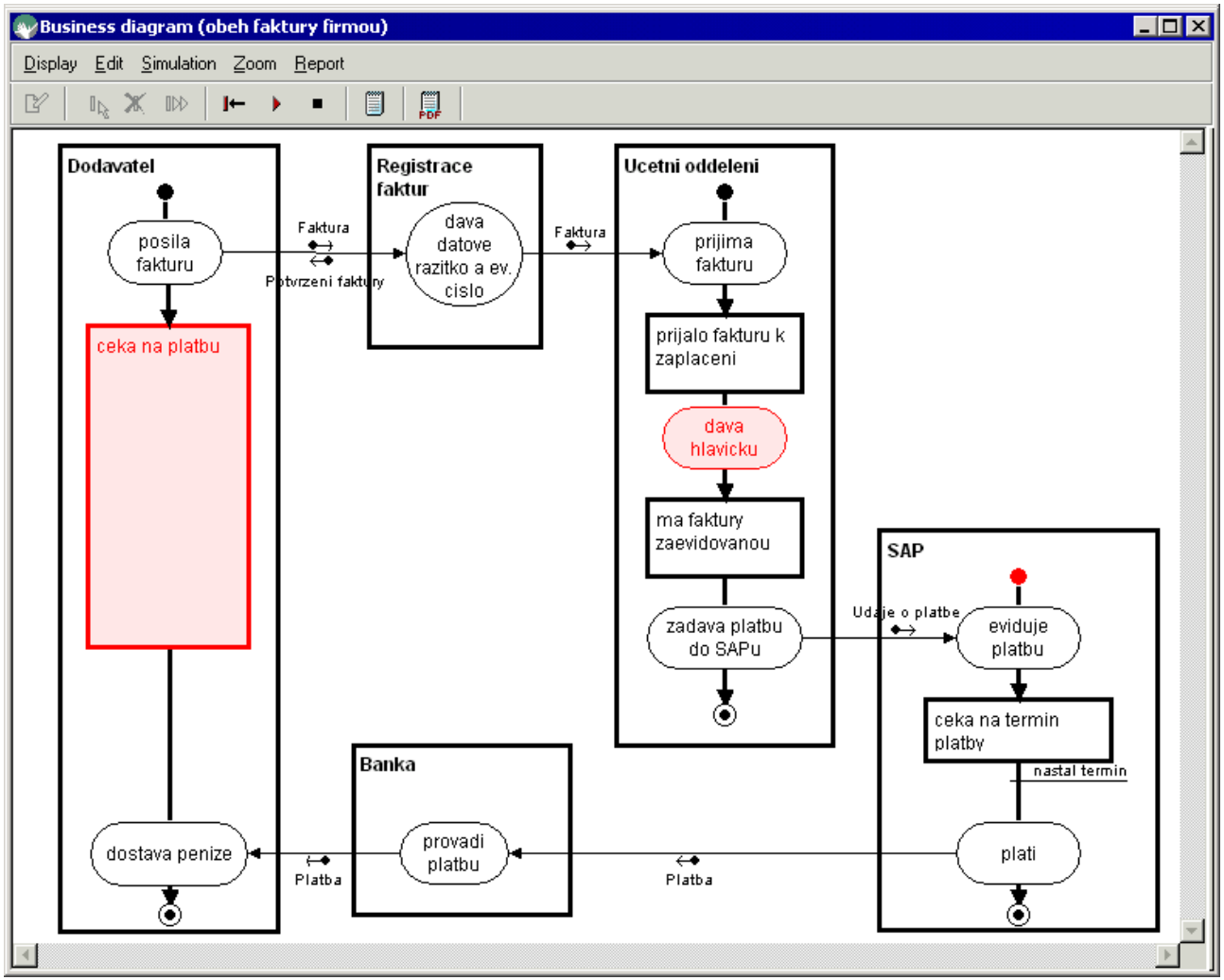
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Business Map - Procesní diagram - Simulace

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

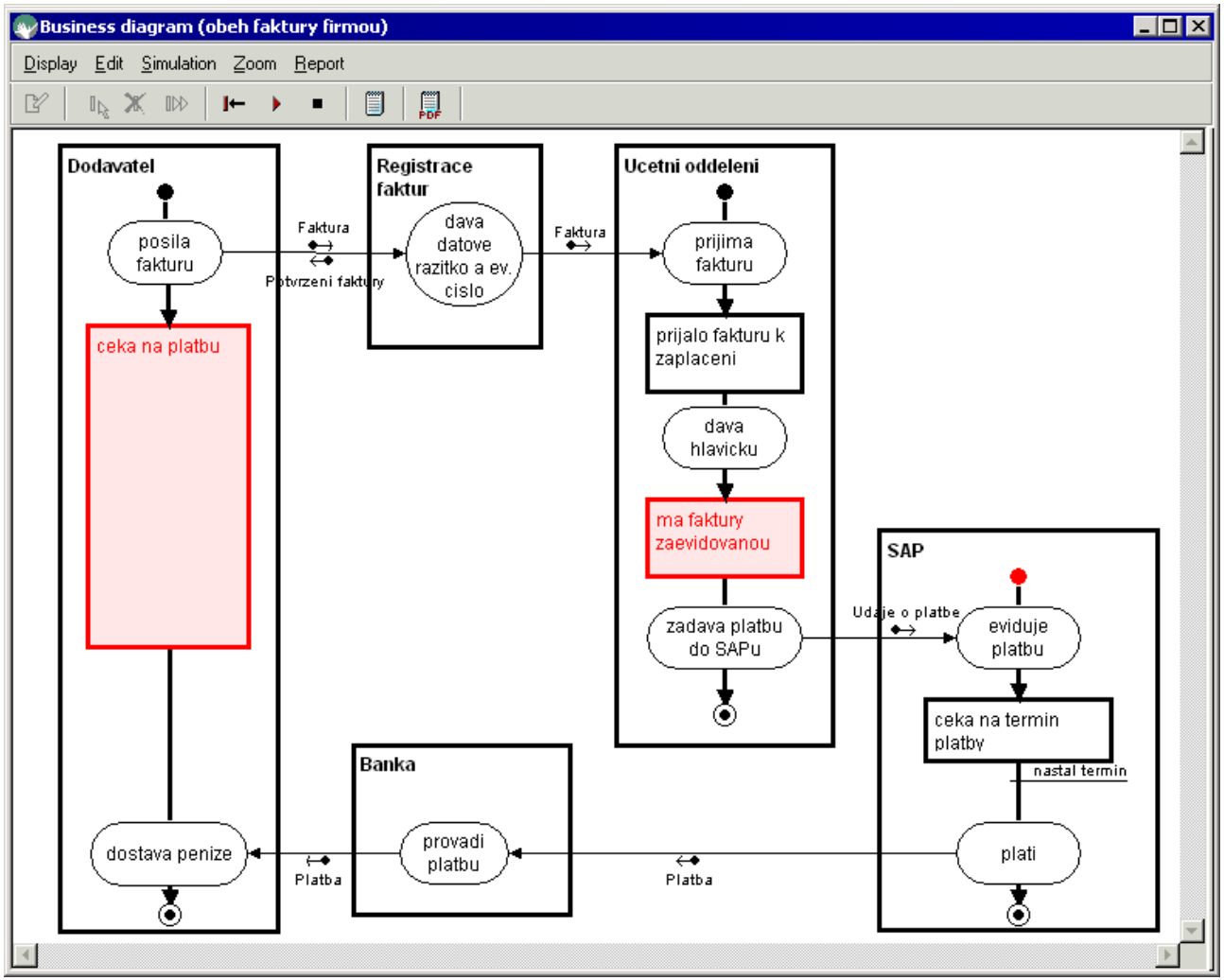
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Business Map - Procesní diagram - Simulace

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

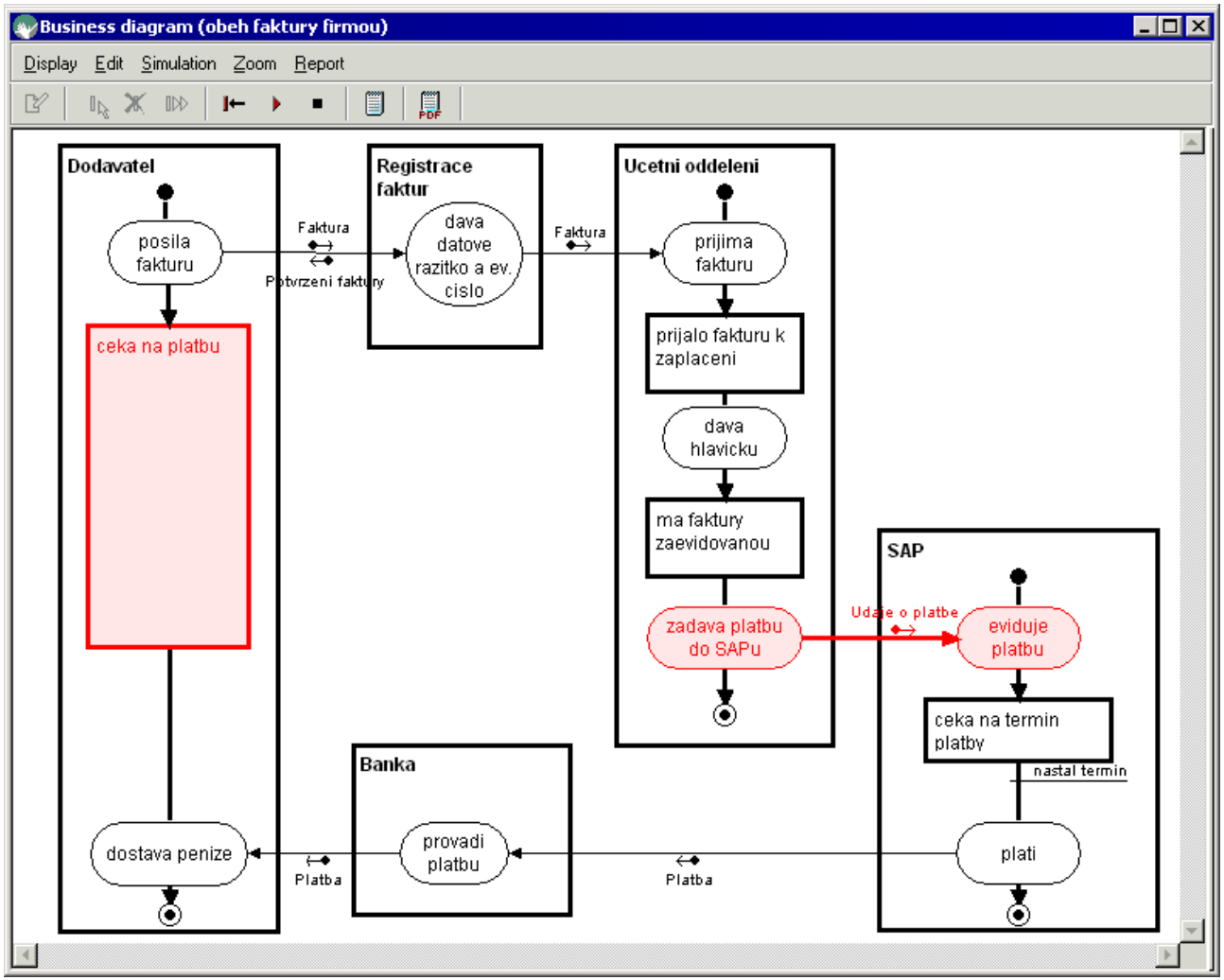
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Business Map - Procesní diagram - Simulace

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

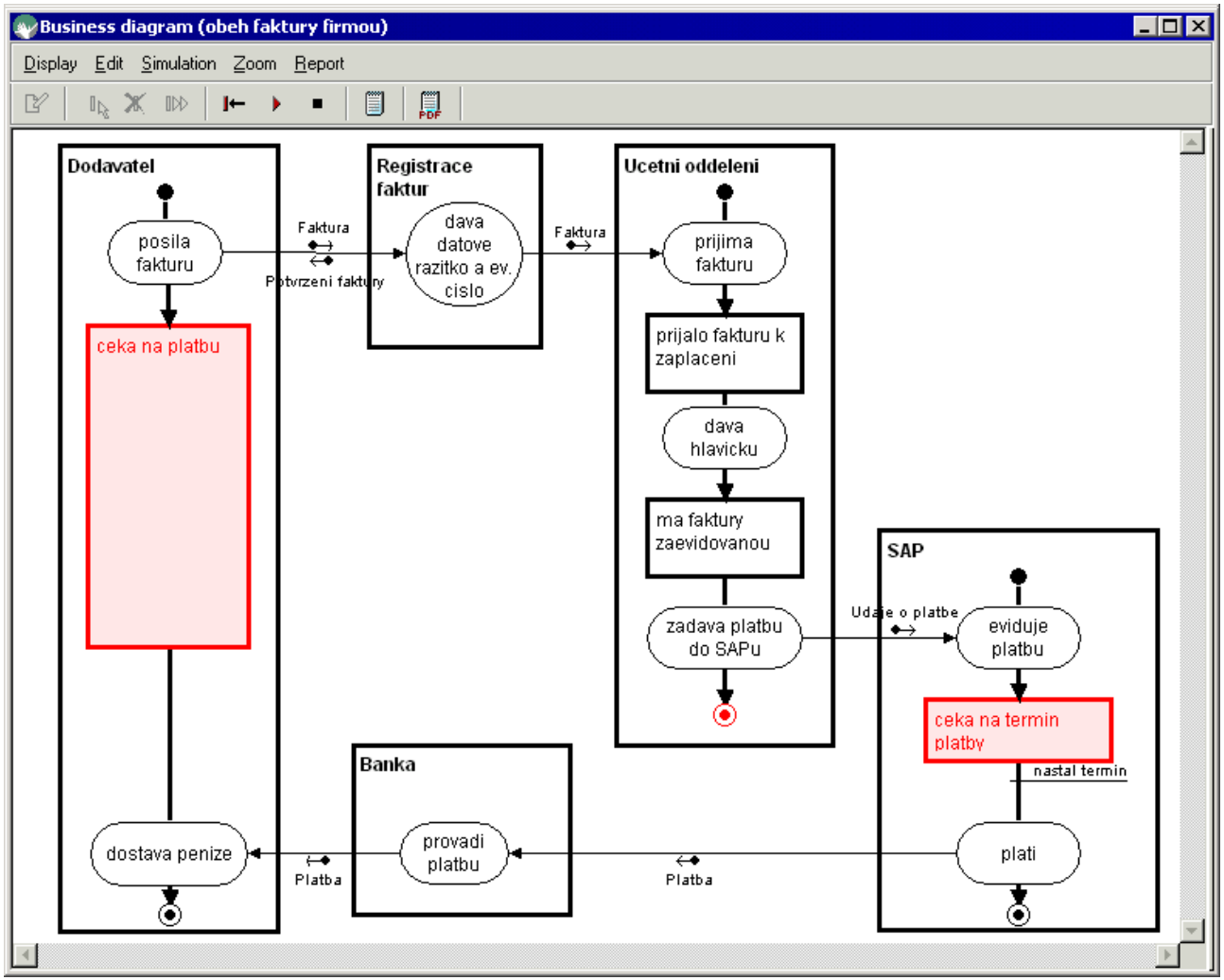
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Business Map - Procesní diagram - Simulace

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

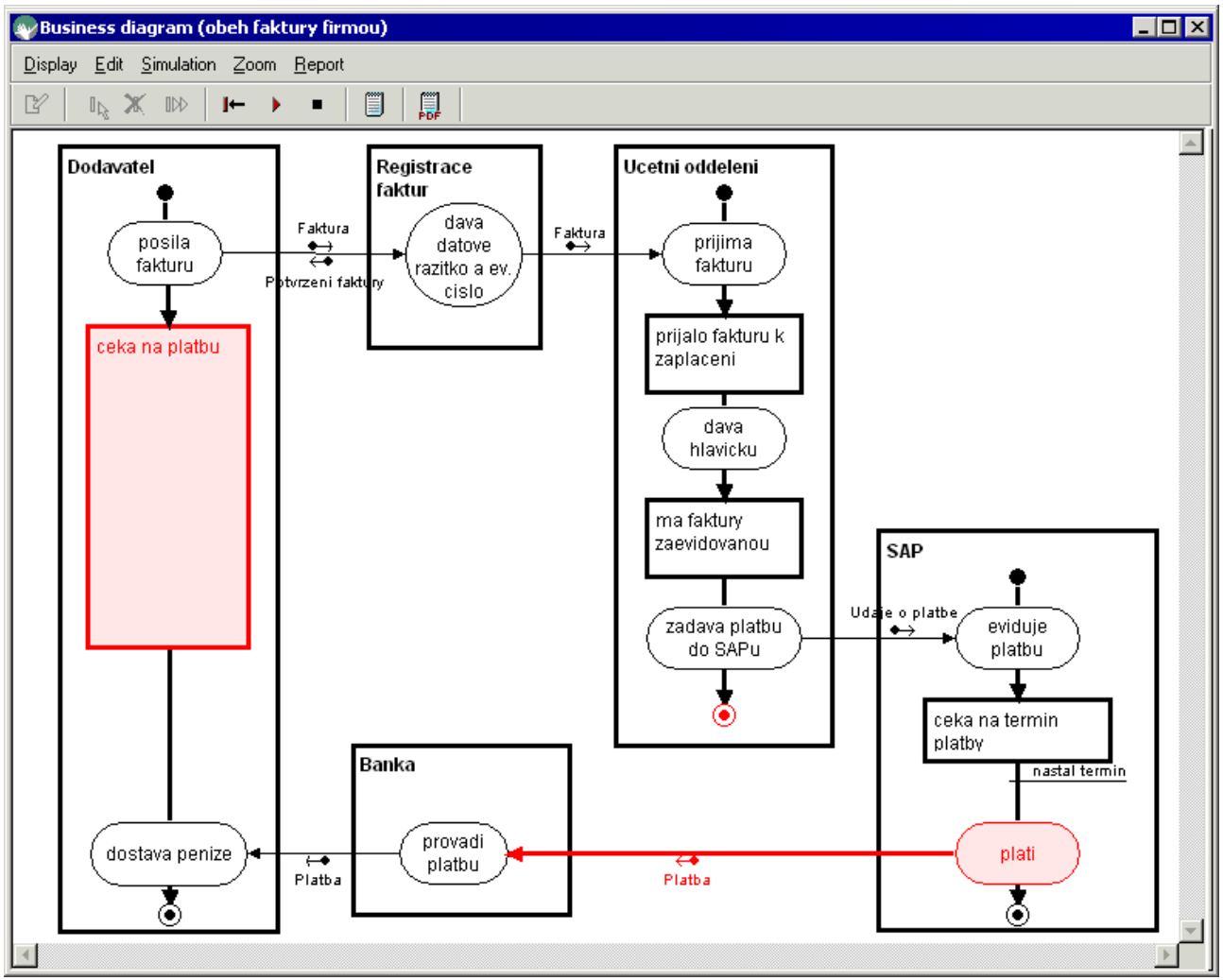
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Business Map - Procesní diagram - Simulace

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

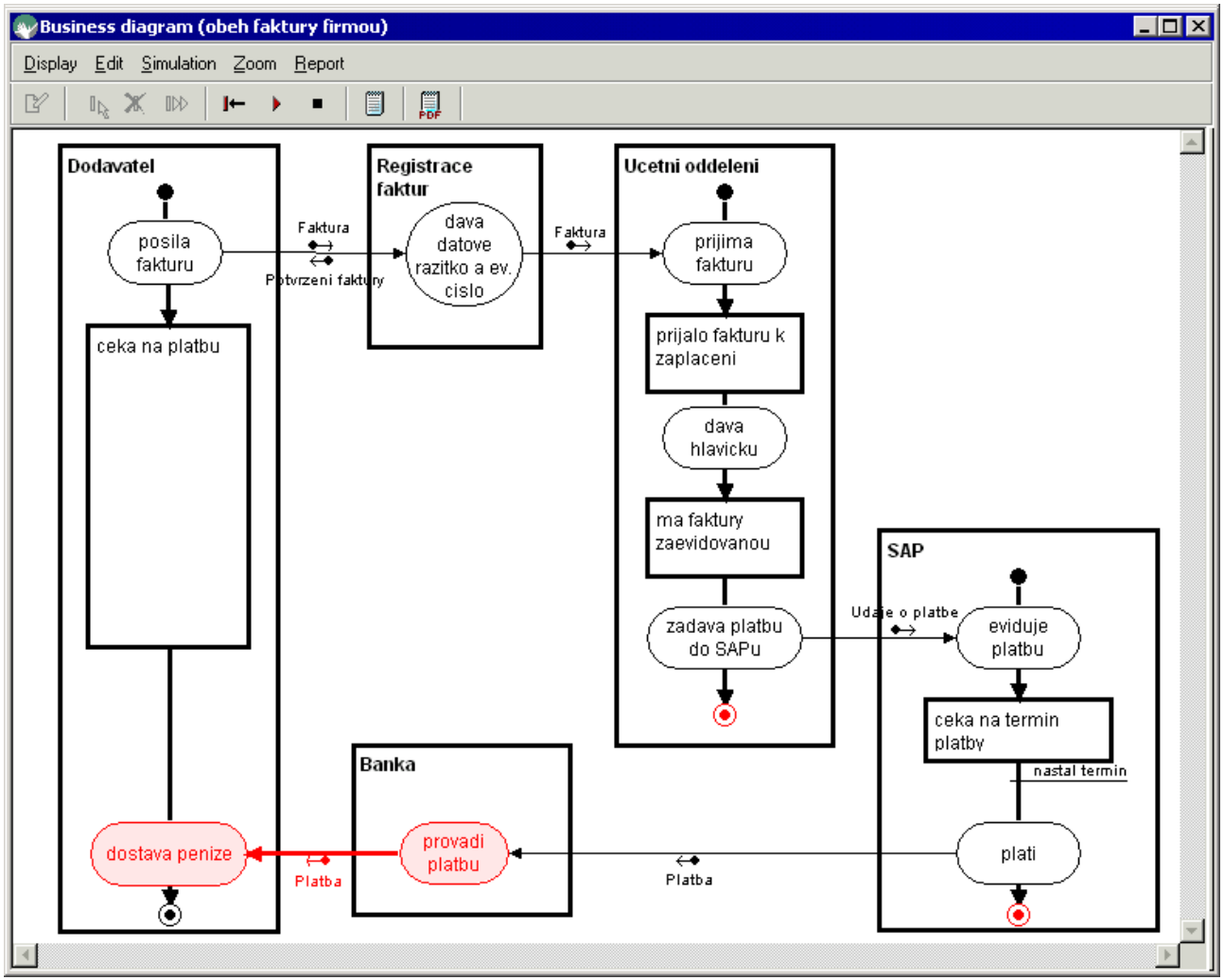
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



Business Map - Procesní diagram - Simulace

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

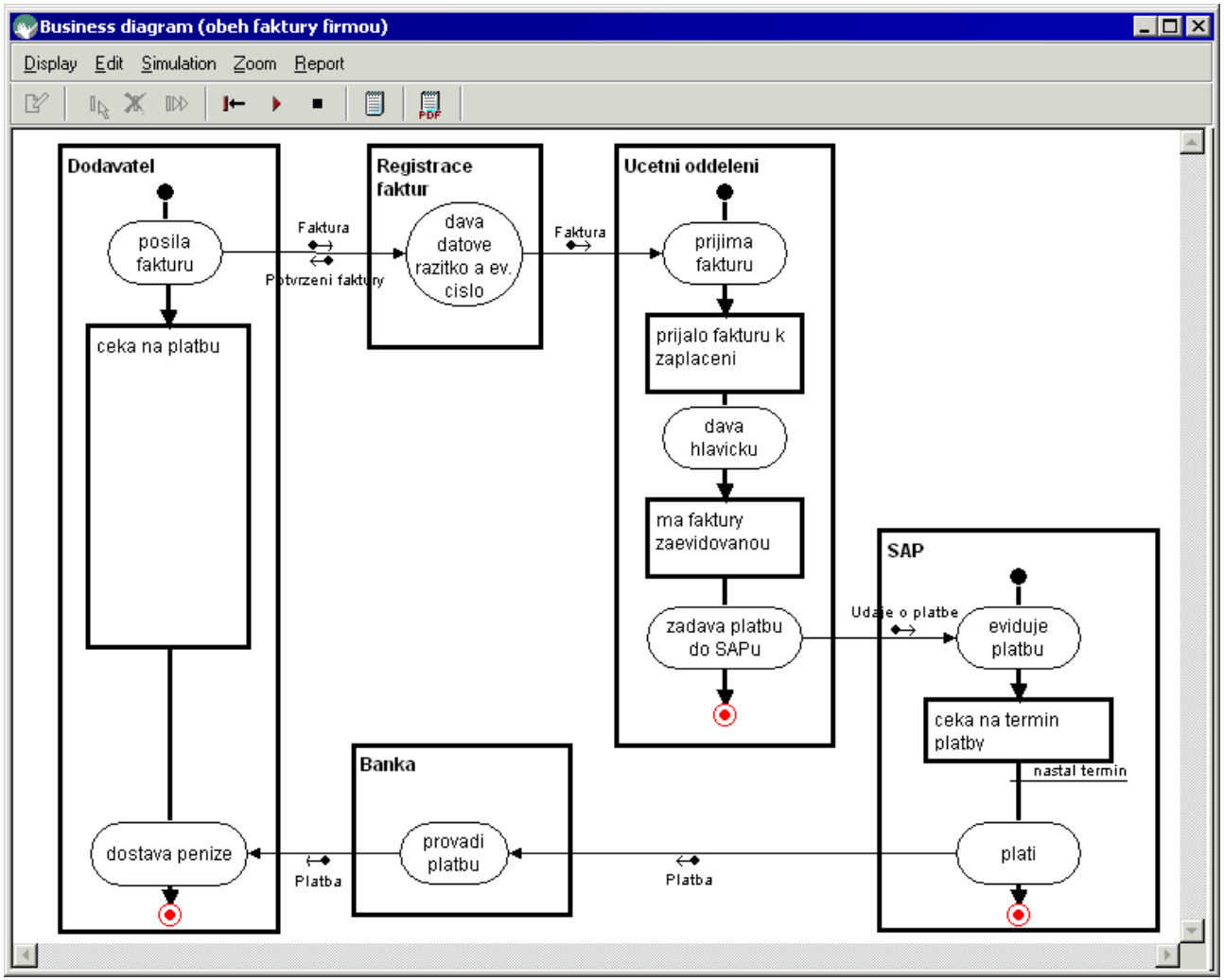
Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



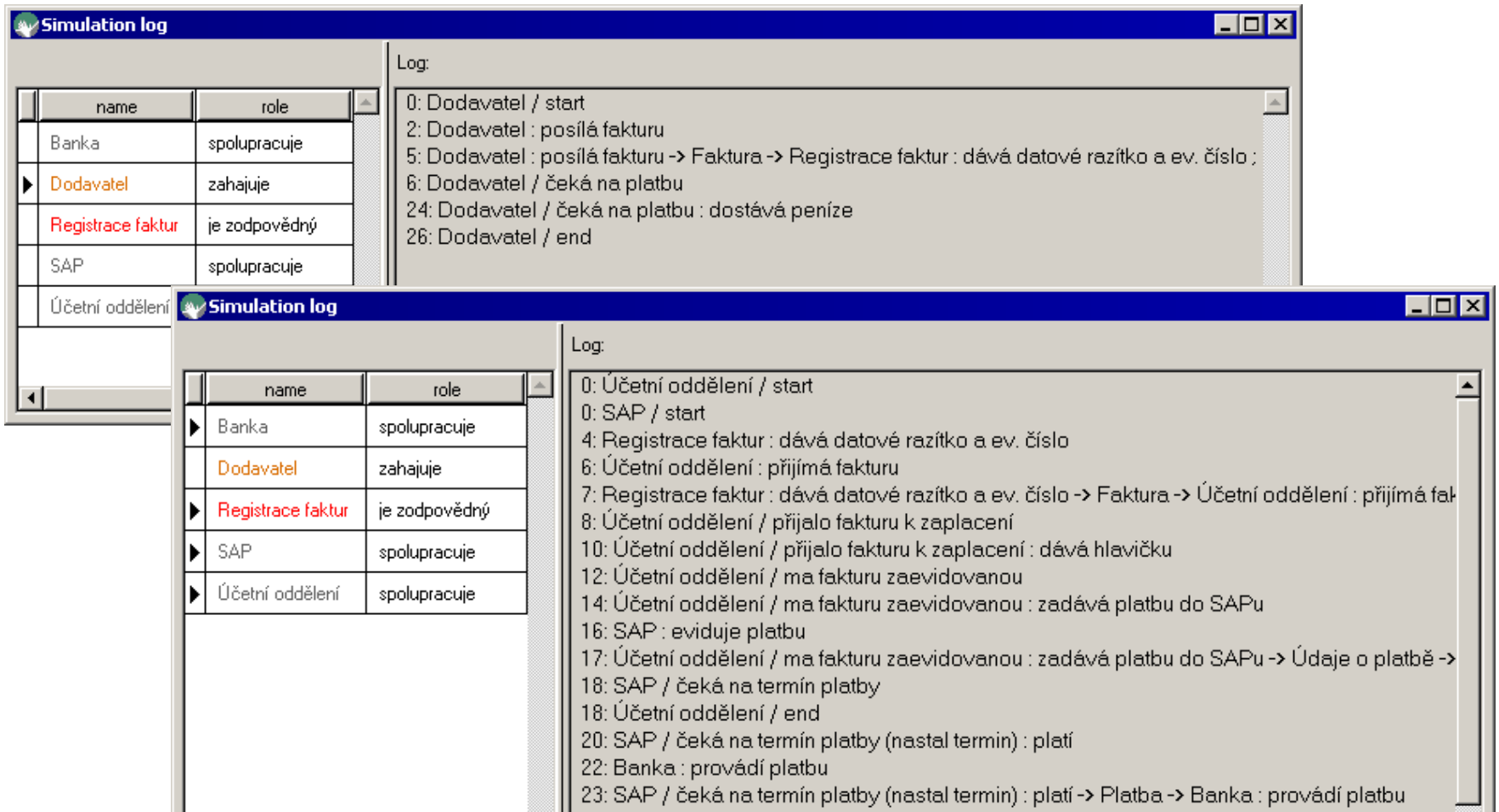
Business Map - Procesní diagram - Simulace

Každý objekt, který v procesu participuje, má svoje v čase proměnlivé chování a komunikuje s druhými objekty.

Výsledný proces je dán sledem událostí vyvolávaných komunikacemi a datovými toky mezi objekty.



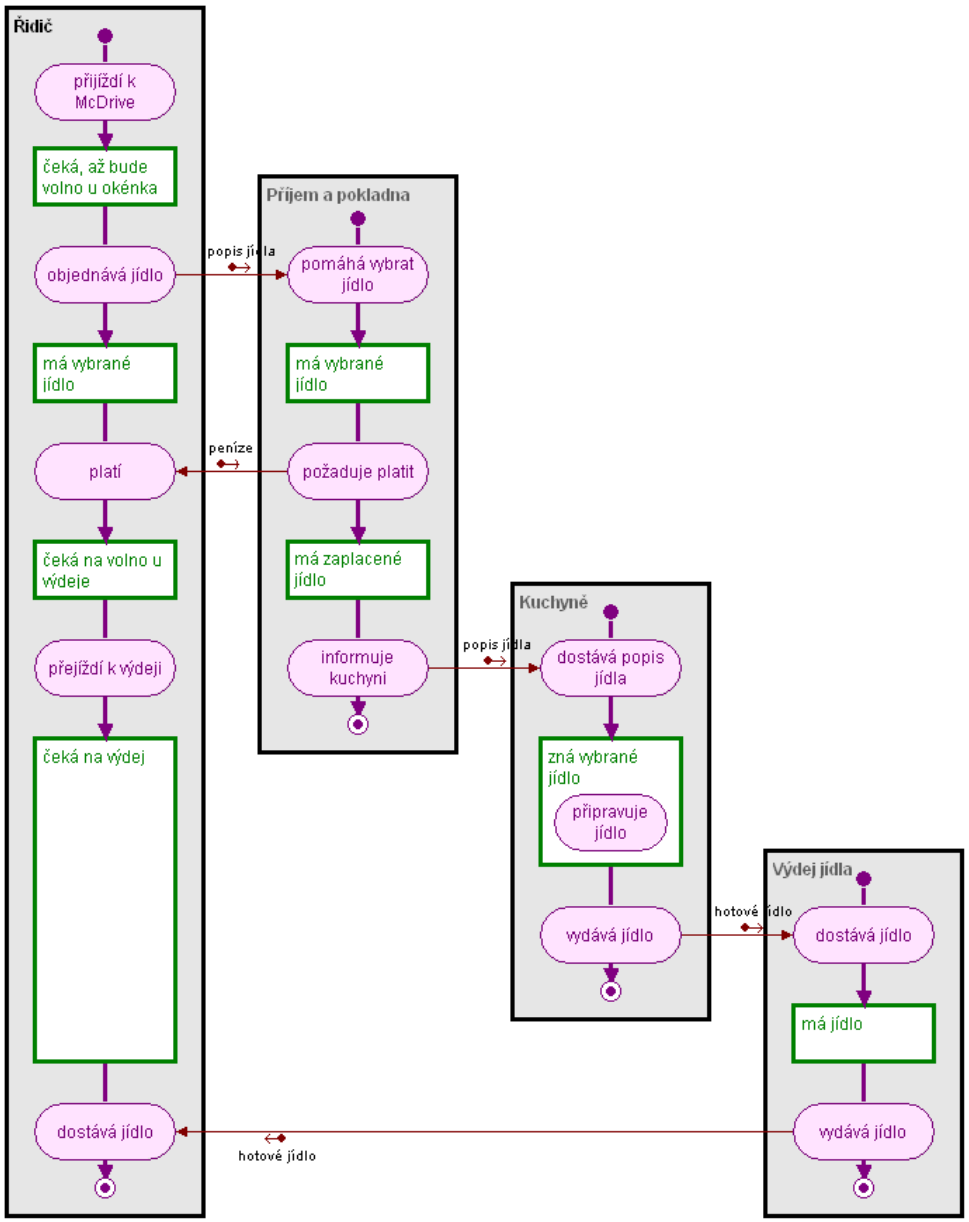
Business Map - Simulační záznam



Díky simulacím lze podrobně analyzovat role jednotlivých objektů nebo skupin objektů v modelovaném procesu.

Další příklady

MCDrive

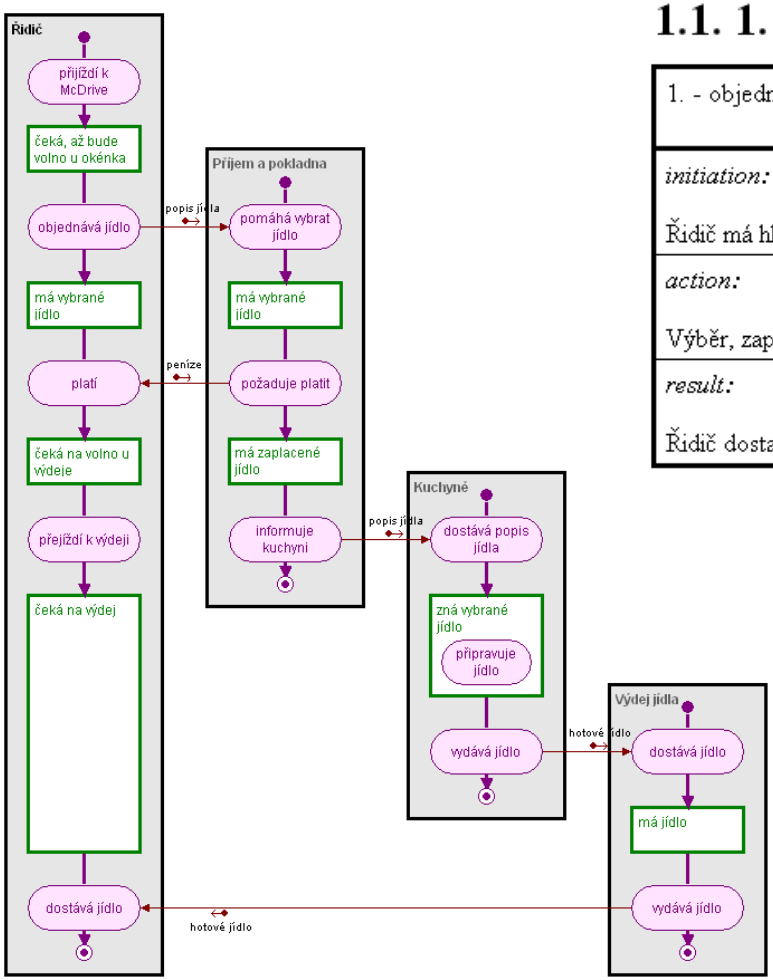

Řidič
přijíždí k McDrive
čeká, až bude volno u okénka
objednává jídlo
má vybrané jídlo
platí
čeká na volno u výdeje
přejíždí k výdeji
čeká na výdej
dostává jídlo
popis jídla
penize
hotové jídlo
Příjem a pokladna
pomáhá vybrat jídlo
má vybrané jídlo
požaduje platit
má zaplacené jídlo
informuje kuchyni
popis jídla
Kuchyně
dostává popis jídla
zná vybrané jídlo
připravuje jídlo
vydává jídlo
hotové jídlo
Výdej jídla
dostává jídlo
má jídlo
vydává jídlo

Scénáře

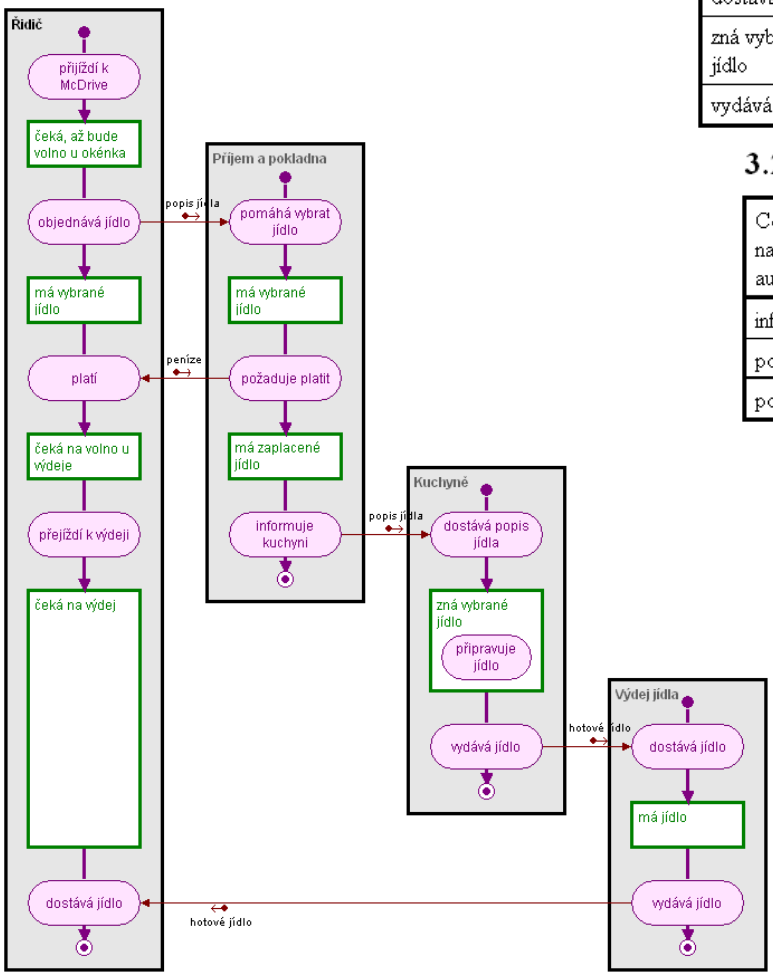
1. Scenarios

1.1. 1. - objednání jídla z auta

1. - objednání jídla z auta	Derived from: zákaznické procesy	
initiation: Řidič má hlad a chce si kopit jídlo v Mc Drive		*roles:* **Kuchyně** cooperates
action: Výběr, zaplacení a vydání jídla do auta		**Příjem a pokladna** cooperates **Řidič** performs
result: Řidič dostal jídlo		**Výdej jídla** cooperates



Modelové karty



3.2.1. Kuchyně

Collaborators in diagram with name '1. Objednání jídla z auta':	Příjem a pokladna	Řidič	Výdej jídla
dostává popis jídla	<<		
zná vybrané jídlo: připravuje jídlo			
vydává jídlo			>>

3.2.2. Příjem a pokladna

Collaborators in diagram with name '1. Objednání jídla z auta':	Kuchyně	Řidič	Výdej jídla
informuje kuchyni	>>		
pomáhá vybrat jídlo		<<	
požaduje platit		>>	

3.2.3. Řidič

Collaborators in diagram with name '1. Objednání jídla z auta':	Kuchyně	Příjem a pokladna	Výdej jídla
dostává jídlo			<<
objednává jídlo		>>	
platí		<<	
přejíždí k výdeji			
přijíždí k McDrive			

3.2.4. Výdej jídla

Collaborators in diagram with name '1. Objednání jídla z auta':	Kuchyně	Příjem a pokladna	Řidič
dostává jídlo	<<		
vydává jídlo			>>

Závěr

http://kii.pef.czu.cz/~merunka/books/DatoveModelovani/

knihy

Datové modelování - Alfa Publishing 2006

Umění systémového návrhu – Objektově orientovaná tvorba informačních systémů pomocí původní metody BORM, Grada, Praha 2003

Management of the Object-Oriented Development Process, Akron Publishing, Virgin Island 2005

Accounting Information Systems, South-Western Publishing, Mason-Ohio 2004

Využití metod umělé inteligence ve vodním hospodářství, nakladatelství Akademie věd ČR, Praha 2004

a dalších cca 5 vysokoškolských skript

časopisy

The BORM methodology: a third-generation fully object-oriented methodology, Knowledge-Based Systems 3(10) 2003, Elsevier Science Publishing, New York.

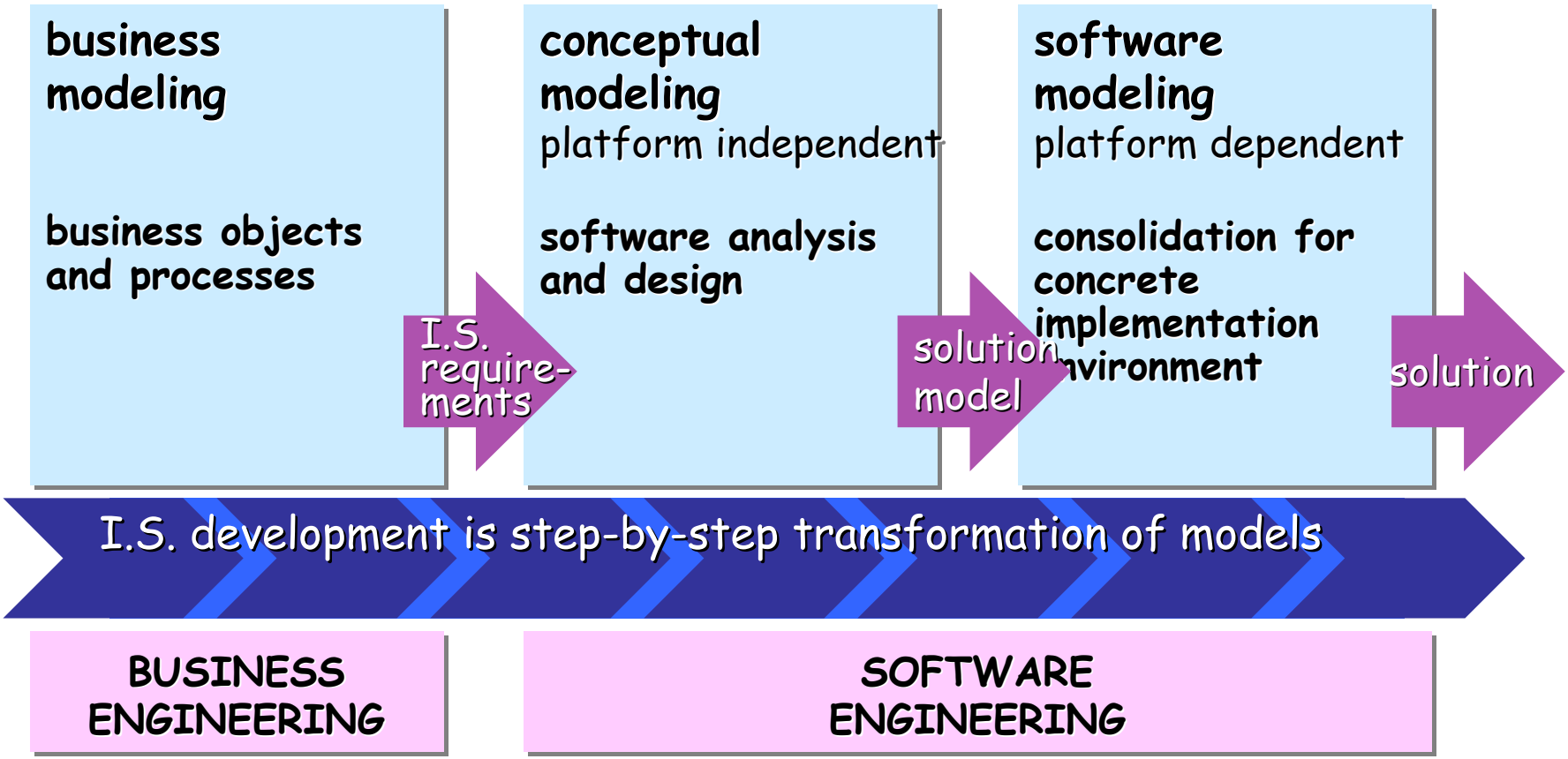
… a další 4 články ve vědeckých časopisech

konference

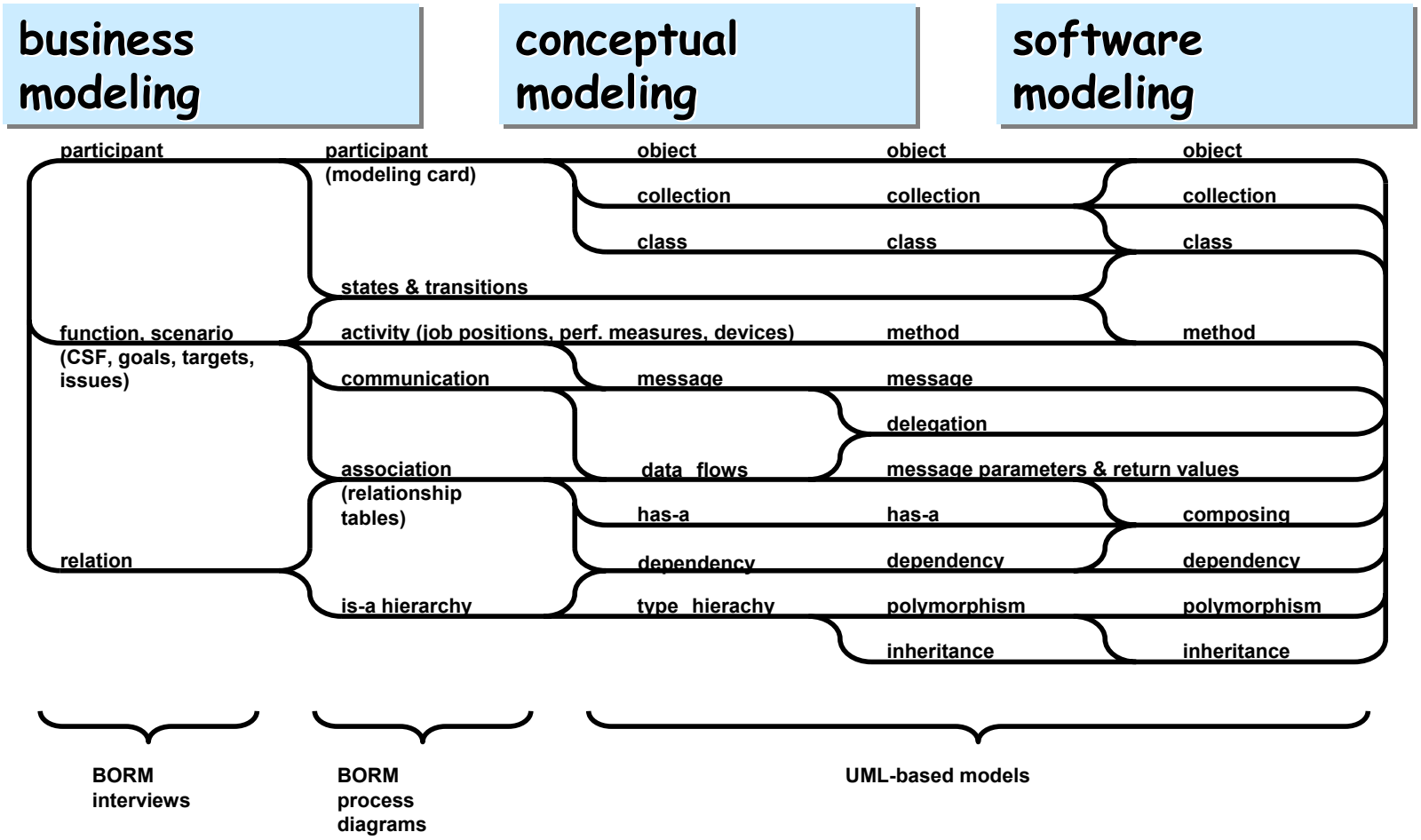
Process Modeling for Object Oriented Analysis using BORM Object Behavioral Analysis, in Proceedings of Fourth International Conference on Requirements Engineering, IEEE Computer Society, Chicago 2000.

… a dalších cca 20 příspěvků na konferencích doma i v zahraničí

BORM Philosophy



BORM Philosophy - Detail



IS-A ≠ subtype-supertype ≠ inheritance

The hierarchy of object in the phase of problem formulation is not totally identical with inheritance or the hierarchy of types.

Nevertheless, UML expresses these three similar, but not identical hierarchies in the same way.

During modeling, it is necessary to look at them as follows:

1. From the designer's perspective – new object designer. This hierarchy is a hierarchy of inheritance because inheritance is a tool for the development of new classes.

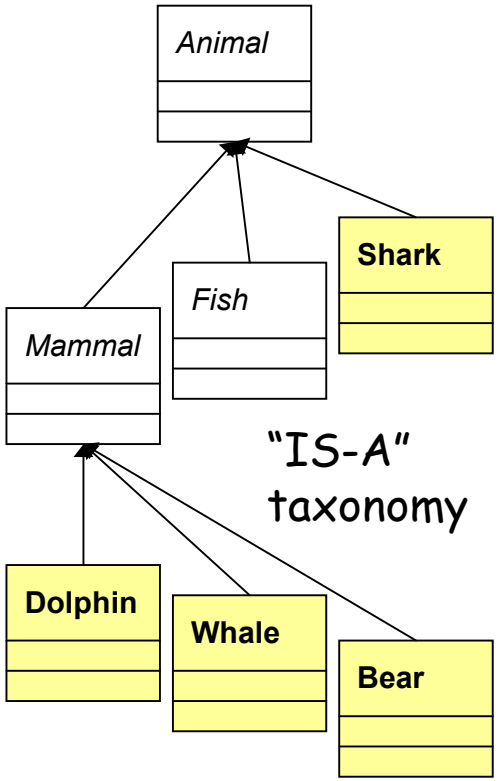
2. From the user's perspective. This perspective can be further divided as follows:

   a) From the perspective of usability – a view of an application programmer who needs to use the objects in his system but does not develop them. This hierarchy is the hierarchy of types.

   b) From the perspective of the user/analyst – the objects of lower-level classes are elements of the same domain as objects of upper-level classes. This hierarchy is called IS-A.

The Example

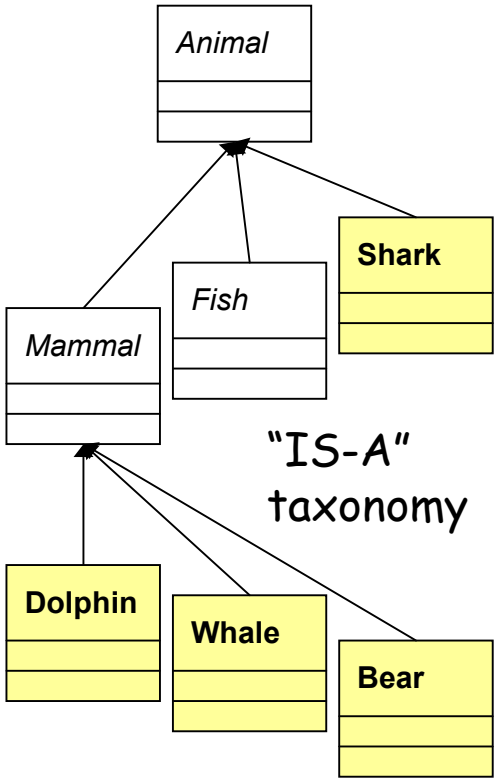
business modeling	conceptual modeling	software modeling



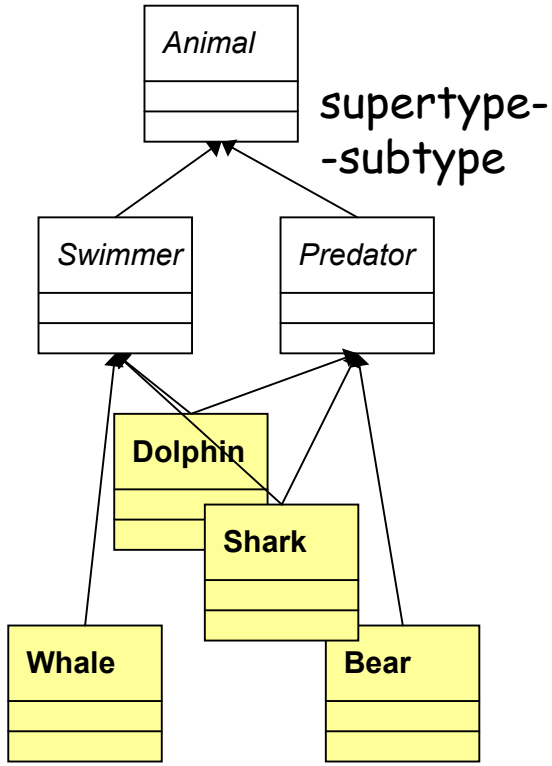
BUSINESS ENGINEERING	SOFTWARE ENGINEERING

The Example

business modeling



conceptual modeling



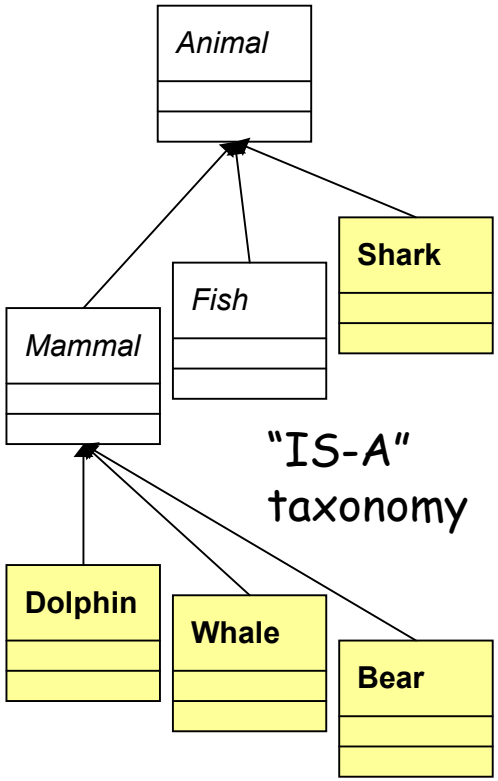
software modeling

BUSINESS ENGINEERING

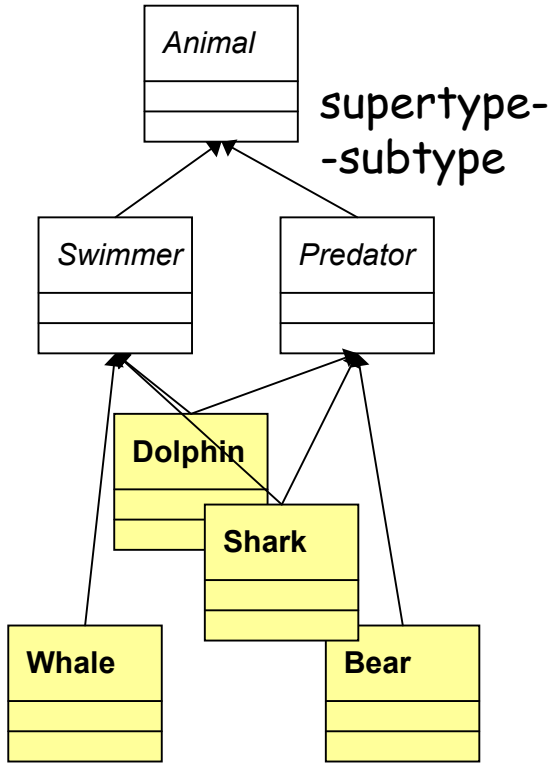
SOFTWARE ENGINEERING

The Example

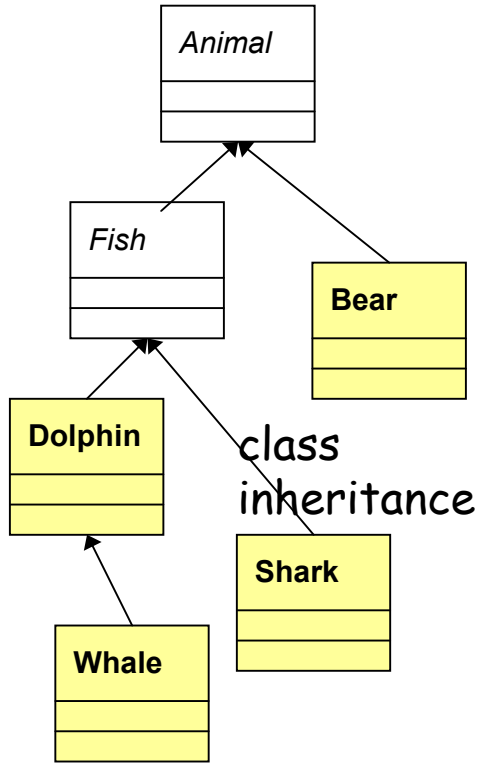
business modeling



conceptual modeling



software modeling

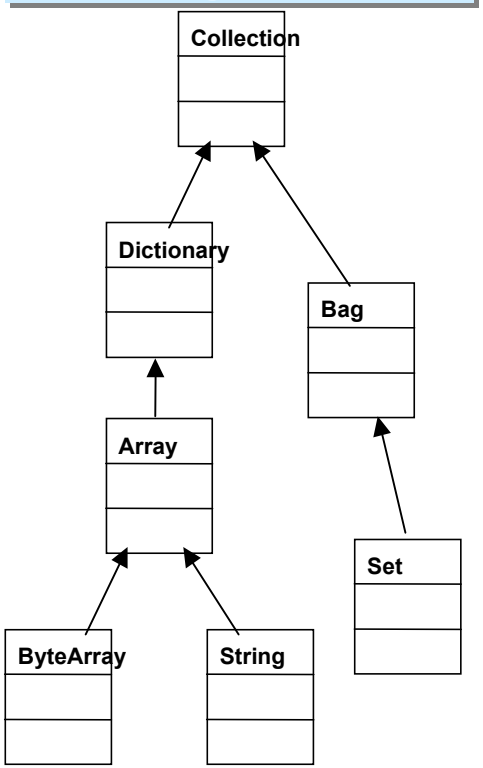


BUSINESS ENGINEERING

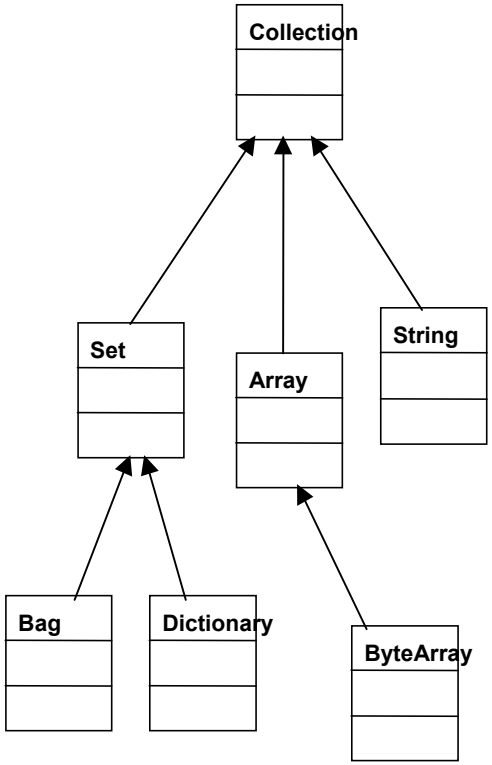
SOFTWARE ENGINEERING

Another Example

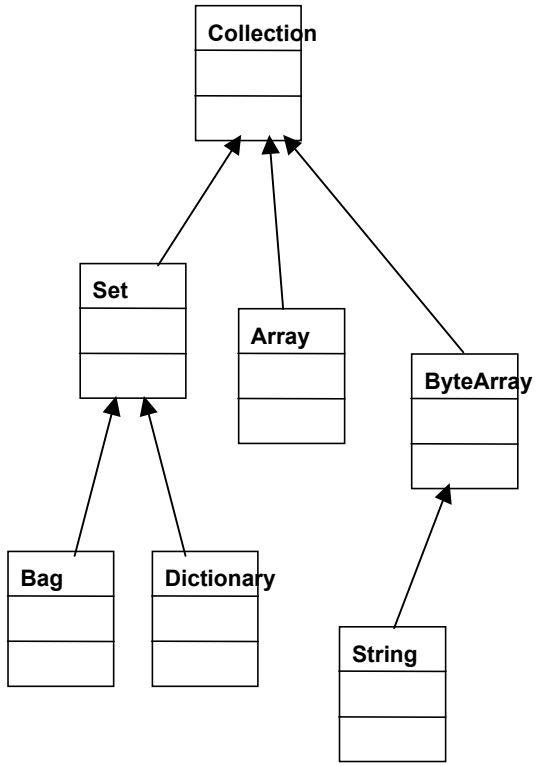
business modeling



conceptual modeling



software modeling



BUSINESS ENGINEERING

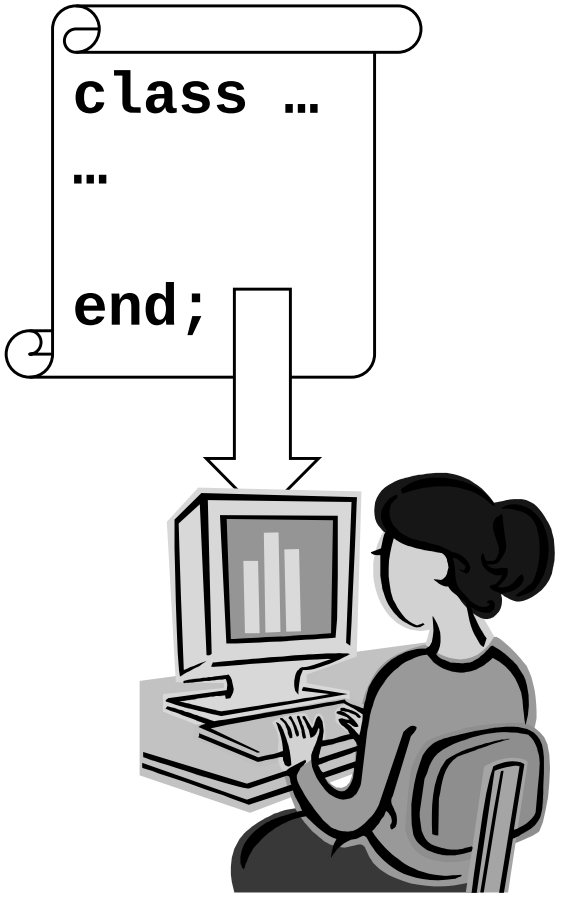
SOFTWARE ENGINEERING

X36SIN
Seminář o metodice projektu „Dlouhán“

Hlavní problém tvorby software

?

Jak to udělat
rychle,
efektivně,
bezchybně,
…



Jak postupovat?

- Jak postupovat by nám měla doporučit nějaká metodika - přehled metodik najdeme např. na „**Google directory**“.
- Dosud jsme probírali metodiky „**BORM**“ a „**Extrémní programování**“.
- Dnes se budeme zabývat metodikou vytvořenou v rámci projektu „**Dlouhán**“.
- Příště budeme probírat metodiky „**UP**“ a „**RUP**“ (17.5.).

Co to je projekt „Dlouhán“?

- Projekt vedený snahou vytvořit smysluplnou metodiku vhodnou pro malé a střední firmy, která by dostatečným způsobem zohledňovala dobré zásady tvorby SW, ale brala v úvahu možnosti menší firmy.
- Název „**Dlouhán**“ je zkratkou za „dlouhý“ celkový název projektu: „Dlouhodobý vzdělávací program řízení vývoje životního cyklu software a zavádění projektového řízení pomocí softwarových nástrojů do malých a středních podniků“:
  - http://www.dlouhan.org/
- Řešitelé:
  - AARON GROUP, s.r.o.: http://www.aarongroup.cz/
  - ILikeThis!, s.r.o.: http://www.ilikethis.cz/
  - ČVUT FEL: http://cs.felk.cvut.cz/webis/research/g20208.html

Představení projektu „Dlouhán“ z pohledu firmy Aaron Group, s.r.o.

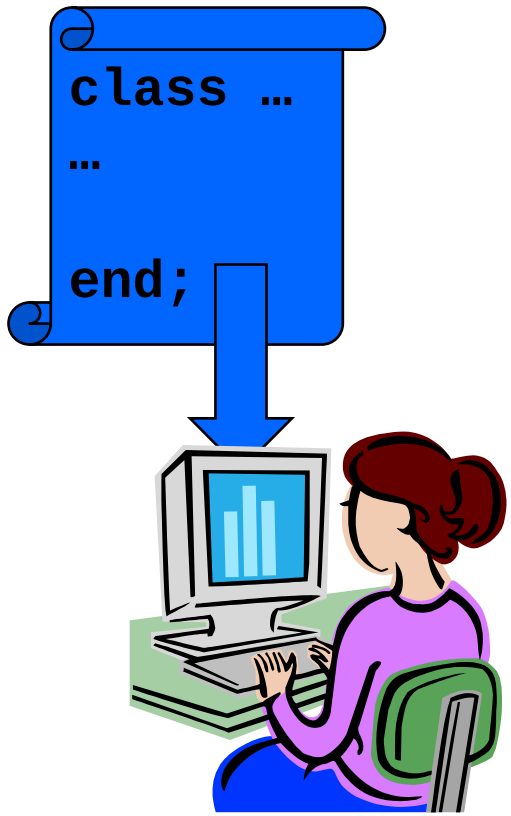
Ondřej Volráb

X36SIN
Seminář o metodice projektu „Dlouhán“

Hlavní problém tvorby software

?

Jak to udělat
rychle,
efektivně,
bezchybně,
…



Jak postupovat?

- Jak postupovat by nám měla doporučit nějaká metodika - přehled metodik najdeme např. na „**Google directory**“.
- Dosud jsme probírali metodiky „**BORM**“ a „**Extrémní programování**“.
- Dnes se budeme zabývat metodikou vytvořenou v rámci projektu „**Dlouhán**“.
- Příště budeme probírat metodiky „**UP**“ a „**RUP**“ (17.5.).

Co to je projekt „Dlouhán“?

- Projekt vedený snahou vytvořit smysluplnou metodiku vhodnou pro malé a střední firmy, která by dostatečným způsobem zohledňovala dobré zásady tvorby SW, ale brala v úvahu možnosti menší firmy.
- Název „**Dlouhán**“ je zkratkou za „dlouhý“ celkový název projektu: „Dlouhodobý vzdělávací program řízení vývoje životního cyklu software a zavádění projektového řízení pomocí softwarových nástrojů do malých a středních podniků“:
  - http://www.dlouhan.org/
- Řešitelé:
  - AARON GROUP, s.r.o.: http://www.aarongroup.cz/
  - ILikeThis!, s.r.o.: http://www.ilikethis.cz/
  - ČVUT FEL: http://cs.felk.cvut.cz/webis/research/g20208.html

Představení projektu „Dlouhán“ z pohledu firmy Aaron Group, s.r.o.

Ondřej Volráb

Jan Vraný

8.3.2006

eXtrémní Programování

Jan Vraný, vranyj@fel.cvut.cz

Co to je XP?

XP je agilní technika vývoje softwaru, která upřednostňuje:

- **teamovou spolupráci a interakci před formálními procesy a nástroji,**
- **fungující software před obsáhlou, komplexní dokumentací,**
- **spolupráci mezi zákazníkem a vývojáři před specifikacemi, zadáními, dohodnutými kontrakty,**
- **rychlou adaptaci na změny v zadání před dodržováním předem stanoveného plánu.**

12 technik používaných v XP

- **Planning game**
- **Small releases**
- **Metaphor**
- **Simple design**
- **Pair programming**
- **Testing**
- **Refactoring**
- **Collective code ownership**
- **Continuous integration**
- **40-hour week**
- **On-site customer**
- **Coding standards**

12 technik používaných v XP

Planning game

Software se vyvíjí v malých iteracích. Vždy se vyvíjí jen to, co zákazník v tu danou chvílí potřebuje.

Iterace se sestává ze:

- **sepsání user stories (zákazník)**
- **rozdělení user stories na úkoly (vývojáři)**
- **casových odhadů na jednotlivé úkoly (vývojáři)**
- **určení priorit (zákazník)**
- **vlastního programování (vývojáři)**
- **akceptace naprogramovaných user stories (zákazník)**

Small Releases

na konci každé iterace (naprogramování user story) se vytvoří nová verze (release). To znamená:

- **zákazník dostává nová verze poměrně často (jednou za 14 dní, jednou týdně, každý den),**
- **existuje okamžitá zpětná vazba, neboť zákazník může software neustále testovat a připomínkovat,**
- **zákazaník má k dispozici funkční část SW velmi rychle a může ho začít používat.**

Metaphor

Metafora je jednoduchá paralela nebo příběh, který popisuje budovaný systém slovy a pojmy, kteří všichni zůčastněni (vývojáři i zákazník) intuitivně chápou.

Slouži především k lepší komunikaci a porozumění mezi členy teamu.

Simple Design

Co přesně má software dělat a jak má vypadat víte v nejlepším případě až když ho potřetí přepisujete celý znova. Scott W. Ambler.

Proto je vhodné udržovat návrh tak jednoduchý, jak jen to jde. Použít to nejjednodušši, co ještě splní akceptační testy.

Neprogramovat pro budoucnost, netvořit frameworky, pokud to neni bezpodmínečně nutné.

Pair Programming

Každá řádka kódu je napsána dvěma programátory u jednoho počítače s jedním monitorem, s jednou klávesnicí.

To zajišťuje:

- **vyšší spolehlivost kódu, neboť programátoři se kontrolují navzájem,**
- **vyšší čitelnost a pochopitelnost kódu, neboť programátoři se kontrolují navzájem,**
- **všeobecné povědomí všech vývojářů o celém kódu**
- **a v neposlední řadě postupné vzájemné učení se vývojářů od sebe.**

Testing

Neustálé testování je jedna z fundamentálních technik XP, bez testování XP nemůže fungovat.

Vývojáři píší jednotkové testy na každou komponentu vyví-jeného systému, dokonce na každou třídu, tyto testy se spouští neustále.

Pokud neprochází všechny jednotkové a akceptační testu user stories, není možné pokračovat ve vývoji.

Refactoring

Refactoring je technika zlepšování stávajícího kódu beze změny dosavadního chování. Refaktoruje kvuli:

- **zjednodušení kódu**
- **optimalizaci kódu**
- **lepší čitelnosti, pochopitelnosti kódu,**
- **budoucí přidání nějaké další funkcionality**

Jistotu, že refaktorováním se nezmění stávající chování nám dávají jednotkové testy.

Collective Code Ownership

V XP teamu, kterýkoliv vývojář je oprávněn kdykoli modifikovat kterýkoliv kus kódu, pokud po modifikaci procházejí testy.

Continuous Integration

Veškerý kód je zpřístupněn všem vývojářům v centrálním repozitáři.

Kdykoli je úkol dokončen (naprogramován) a otestován, je uložen do centrálního repozitáře.

40-hour week

Psychicky nebo intelektuálné unavený vývojář je:

- **nevýkonný,**
- **chybující**
- **velmi naštvaný**

Proto je nutné vývojovému teamu zajistit klid na práci, důstojné pracovní prostředí a hlavně je nepřetěžovat.

On-site customer

On-site customer je člověk od zákazníka, který představuje typického uživatele vyvíjeného software a která je *neustále* k dispozici vývojářům aby:

- **zodpovídal otázky vývojářů během práce**
- **tvořil akceptační testy pro user stories**
- **testoval a podával zpětnou vazbu**

Coding Standards

Je nutné, aby *všichni* programátoři dodržovali stejné "coding standards", tedy:

- **stejné zarovnání kódu,**
- **stejné názvové konvence,**
- **stejně nastavené vývojové prostředí.**

Kdy má cenu použít XP?

- **Když je limitovaný čas nebo nebo financování.**
- **Když zákazník neví přesně co chce nebo není schopen dodat dostatečně přesné zadání.**
- **Když zákazník chce spolupracovat na projektu.**
- **Když se ví, že systém se bude měnit.**
- **Když členové teamu spolu dobře vycházejí a jsou schopni spolupracovat.**

Kdy nebude XP fungovat?

- **Je nutný početný (>15) team vývojářů?**
- **Jak dlouho trvá cyklus uprava kódu - překlad, linkování - spuštění? Více než 5 minut? Vice než hodinu?**
- **Jak douho běží testy? Minuty? Hodiny?**
- **Je nemožné dostat zákazníka na pracoviště?**
- **Nesouhlasí zákazník s XP? Nechápe dobře jeho principy?**

Pokud alespoň na jednu otázku odpovíte ano, XP zřejmě nebude nejlepší volba...

eXtrémní Programování

- **Představuje alternativu ke klasickému vodopádovému modelu vývoje SW.**
- **XP není nekoordinované hackování softwaru zdivočelými programátory - hackery.**
- **Aby XP fungovalo, team musí mít jiste zkušenosti a musí bezpodmínečně dodržovat *všech* 12 technik.**
- **XP není všelék, hodí se jen někdy. Pokud se ale správně nasadí, mívá lepší výsledky za kratší čas.**

Zkušenosti se zaváděním formalizovaných postupů do řízení projektů v malých a středních firmách

Ing. Ondřej Volráb
AARON GROUP
e-mail: volrab@aarongroup.cz

Náplň semináře

- Vymezení pojmů
- Vývoj SW a jeho specifika
- Specifika malých organizací
- Proces zavádění metodiky do organizace
- Přínosy pro organizaci
- Doporučení a diskuse

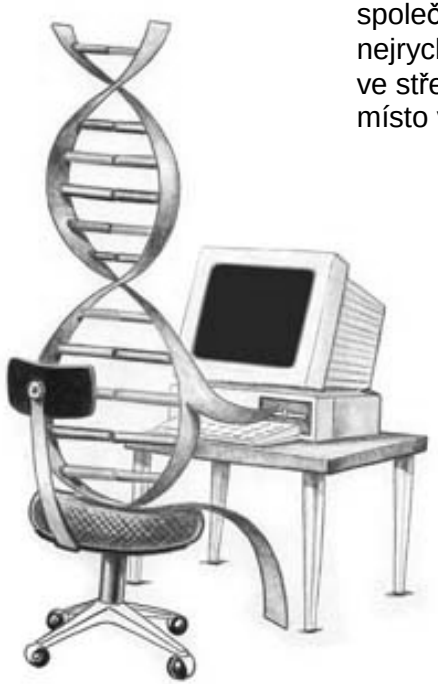
AARON GROUP

10 let

AARON GROUP úspěšně působí na českém trhu od roku 1997. Do obchodního rejstříku byla zapsána v květnu roku 1998 jako společnost s ručením omezeným se základním jměním 600 000 Kč, Dnes má společnost přes 20 stálých zaměstnanců.

Deloitte. Technology Fast 50

... in Central Europe
Po čtvrté v řadě byla AARON GROUP společností Deloitte vyhlášena v žebříčku TOP50 nejrychleji se rozvíjejících technologických společností ve střední Evropě s růstem 551 % v letech 2001-2005 (5. místo v ČR).



motto: **... aby informace pracovaly pro Vás.**

ALTRON

Letiště Praha

ČESKÉ DRÁHY, a.s.

LETUSKA.CZ

ČSA CZECH AIRLINES

| STUDENT | AGENCY |

Náplň semináře

- Vymezení pojmů
  - Řízení projektů
  - Vývoj software
  - Standardizované postupy
- Vývoj SW a jeho specifika
- Specifika malých organizací
- Proces zavádění metodiky do organizace
- Přínosy pro organizaci
- Doporučení a diskuse

Řízení projektů

- Předmětem bylo/je:
  - Řízení projektů ve smyslu vývoje SW a činností bezprostředně souvisejících
- Předmětem nebylo:
  - Řízení dalších typů projektů jako jsou konzultační služby, monitoring provozovaných aplikací, personální řízení, marketing…

Vývoj software

- Má jasně definovaný cíl
  - Business analýza
  - Detailní sběr požadavků
  - Způsob testování výsledku

    ...
- Před zahájením vývoje se co nejpřesněji definuje způsob vývoje, použité metody a nástroje
- Dosažení cíle je garantováno plánem vývoje

Standardizované postupy

- Přesnost plánu závisí na schopnosti odhadnout náročnost práce
- Pro schopnost odhadovat je nezbytné dokázat
  - Změřit velikost výstupů plánované činnosti
  - Definovat, jak bude výstupů dosaženo
- Standardizace postupů je pro plánování nezbytná
  - Umožňuje předem definovat, jak se bude pracovat
    - Předem vím, jak se co dělá
    - Jsem schopen zajistit, že všichni pracovníci práci dělají určitým způsobem (včetně nových pracovníků)
  - Celkové množství práce je možné změřit před jejím vykonáním

Náplň semináře

- Vymezení pojmů

- Vývoj SW a jeho specifika

- Specifika malých organizací
- Proces zavádění metodiky do organizace
- Přínosy pro organizaci
- Doporučení a diskuse

Čím je vývoj SW specifický

- Stále se mění pracovní prostředky
  - V podstatě v každém SW projektu se používají nové technologie (nástroje, postupy, šablony apod.)
- Složitý odhad celkové pracnosti
  - Pro nové technologie neplatí staré odhady
  - Změna pracovních nástrojů se promítá do změny pracovních postupů
  - Náročnost požadavků, které má projekt splnit, často není na první pohled zřejmá
- Změnové řízení v rámci projektu
  - Požadavky na vyvíjený sw se v průběhu projektu často mění
  - Odhad dopadu požadované změny na projekt vyžaduje detailní znalost aktuálního stavu a podrobnou změnovou analýzu

Jak specifika software řešit

- Přijímat nové technologie opatrně
  - Ověřovat je v rámci interních projektů nebo studie proveditelnosti komerčních projektů
- Neustále zlepšovat odhady složitosti
  - Uchovávat historii odhadů
  - Vytvářet převodní tabulku výpočtu složitosti a časové náročnosti při změně technologie
- Detailní změnové řízení
  - Každou požadovanou změnu důkladně ohodnotit

Náplň semináře

- Vymezení pojmů
- Vývoj SW a jeho specifika

- Specifika malých organizací

- Proces zavádění metodiky do organizace
- Přínosy pro organizaci
- Doporučení a diskuse

Specifika malých organizací

- Ad-hoc přístup
  - Malé firmy mají konkurenční výhodu v dynamičnosti, kterou nesmí standardizací postupů ztratit
  - Malý tým zjednodušuje komunikaci a je možná nižší úroveň standardizace
- Menší vnitřní organizace
  - Pracovníci mají větší podíl na řízení a organizační změny jsou proto obtížnější
  - Existují obvykle jen minimální mechanismy interních kontrol – nejsou na ně kapacity ani vůle
- Nedostatek zdrojů
  - Malé organizace mají menší finanční a personální rezervy
  - Malé organizace nejsou zvyklé využívat externích konzultantů
  - Menší přínos ze zlepšení – zákazníci očekávají od menší firmy vysokou míru flexibility a ne pevně dané postupy.

Náplň semináře

- Vymezení pojmů
- Vývoj SW a jeho specifika
- Specifika malých organizací

- Proces zavádění metodiky do organizace
  - Zahájení projektu
  - Zpracování procesních oblastí
  - Vypracování metodiky
  - Nasazení metodiky
    - Školení pracovníků
    - Pilotní projekty
  - Oponentura adaptace metodiky

- Přínosy pro organizaci
- Doporučení a diskuse

Zahájení projektu

- Stanovení cíle – příprava na ISO certifikaci
- Harmonogram
- Zdroje potřebné k úspěchu projektu
- Jak vyhodnotit výsledek

- Definice postupů
  - Existuje řada metodik přímo navržených pro řízení vývoje SW (PMBOK, RUP, ITIL, BORM, agilní metodiky…)
  - Výhody standardních metodik
    - Nezapomene se na žádnou důležitou oblast
    - Využívání standardu je kladným signálem pro zákazníky
  - Nevýhody standardních metodiky
    - Aplikace vede k potlačení dobrých stránek a kompetenčních výhod konkrétní společnosti
    - Velmi často je zavedení pouze formální
    - Jsou obvykle určeny pro velké firmy a velké projekty

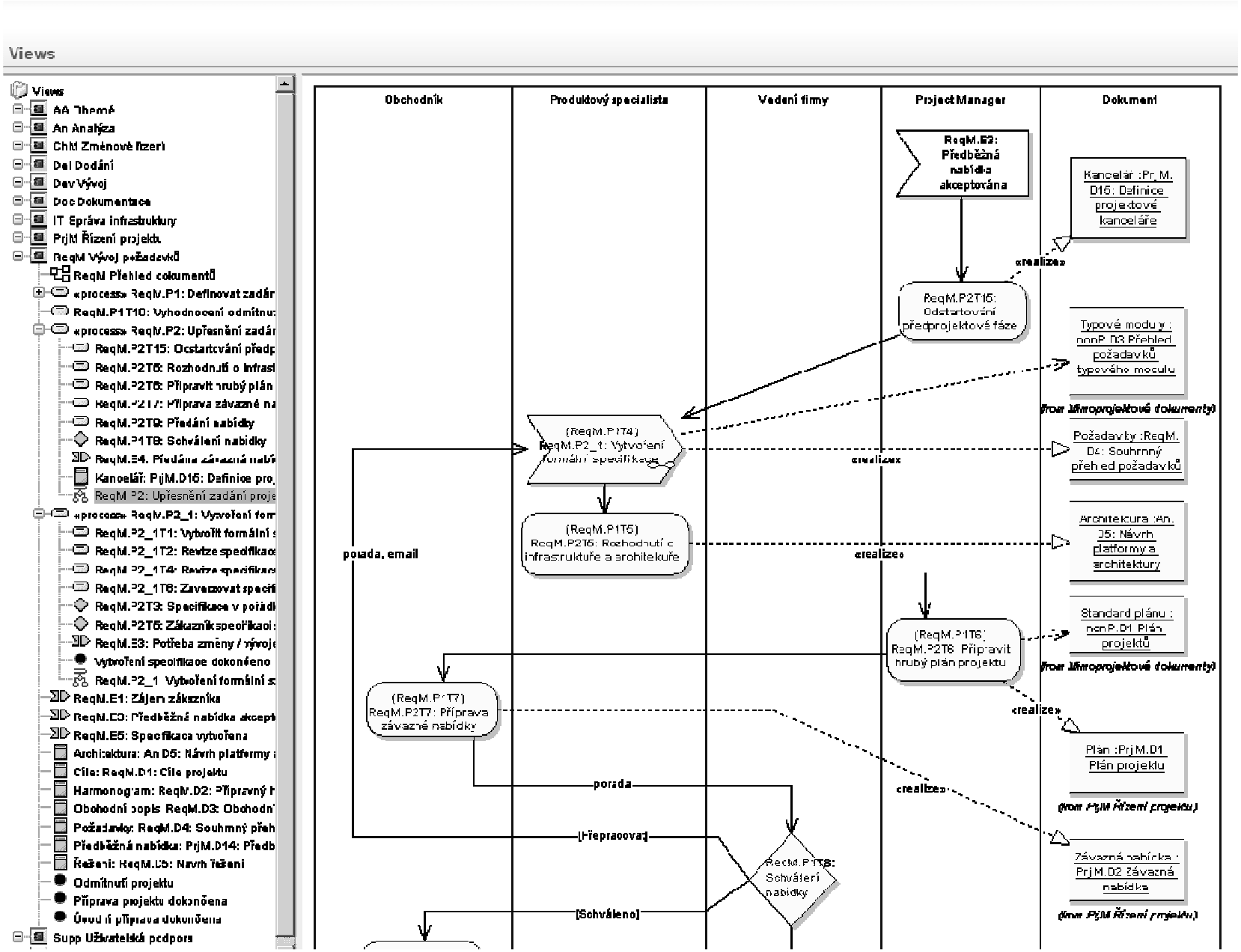
Zpracování procesních oblastí

- Vymezení a pokrytí procesních oblastí:
  - Vývoj požadavků
  - Řízení projektu
  - Analýza
  - Tvorba dokumentace software
  - Změnové řízení
  - Vývoj řešení
  - Testování
  - Předání hotového díla
  - Uživatelská podpora
  - Správa infrastruktury

Vypracování metodiky

- Detailní vymezení každé procesní oblasti – co je jejím obsahem, kde začíná a končí
- Procesní mapy, jejich popis a způsob rozšiřování
- Doporučené standardní metodiky a jejich kombinace
- Doporučené znalosti a způsob jejich získání (vybraná školení)
- Pracovní postupy (style guides) pro jednotlivé činnosti
- Šablony standardních dokumentů

Ukázka procesních map



Oponentura adaptace metodiky

- Každá zpracovaná procesní oblast prošla oponenturou ze strany:
  - Obou firem (AARON GROUP, ILikeThis!)
  - ČVUT
  - Externího konzultanta
- Všechny nasazené procesy jsou průběžně oponovány externím konzultantem

Nasazení metodiky

- Školení pracovníků
  - Systém pravidelného školení rozděleného dle procesních oblastí a participujících rolí
  - Opakování školení je nutnost!
  - Získání zpětné vazby
- Pilotní projekty
  - Ověření životaschopnosti metodiky a její průběžné vylepšování na základě reálných projektů
    - Komerčních (zohlednění vyšší časové náročnosti)
    - Interních (s podporou managementu)
  - Neověřuje se jen metodika samotná, ale rovněž míra přijetí ze strany všech pracovníků

Klíčové faktory úspěchu

- Stanovení důvodu, proč to firma dělá
- Cíl musí být všem ve firmě jasný
  - Každý pracovník sleduje změny ze své vlastní role
  - Nebezpečí, že cíle změn se budou rozcházet
- Standardizace postupů je zásadní zásah do firmy, který může také uškodit!
- Rizika špatné práce
  - Poškození firmy špatně navrženými procesy
  - Nevhodné načasování (nedostatek času)
  - Nedostatečné zdroje

Náplň semináře

- Vymezení pojmů
- Vývoj SW a jeho specifika
- Specifika malých organizací
- Proces zavádění metodiky do organizace

- Přínosy pro organizaci

- Doporučení a diskuse

Přínosy pro malou organizaci

- Zaměření na to, co přinese největší efekt
  - Formalizace postupů plánování projektů
  - Zpřesňování způsobů odhadů náročnosti
  - Zlepšování postupů interních kontrol
- Vymezení kompetencí a odpovědností
  - V rámci projektů
  - V rámci celé organizace
- Standardizace postupů vzhledem k zákazníkovi
  - Proces schvalování zadání a harmonogramu projektu
  - Proces změnového řízení
  - Proces předávání výsledků projektu
- Standardizace komunikace
  - Zápisy z porad a jejich správa
  - Standardizace projektové adresářové struktury

Zabezpečení trvalého zlepšování

- Nic není na poprvé dokonalé
  - Procesy a postupy je třeba trvale zlepšovat
  - Je třeba definovat vlastníka procesů
  - Je nutné interně auditovat, jestli všechny činnosti
    - Slouží svému účelu
    - Jsou efektivní
    - Jsou vykonávány správnými lidmi
- Nic není dokonalé věčně
  - Vývoj firmy = potřeba upravovat její procesy
  - Nutnost definovat osobu odpovědnou za
    - Sledování změn v projektech a aktualizaci pracovních postupů
    - Sledování zkušeností a schopností pracovníků
    - Průběžně školit stávající i nové pracovníky

Náplň semináře

- Vymezení pojmů
- Vývoj SW a jeho specifika
- Specifika malých organizací
- Proces zavádění metodiky do organizace
- Přínosy pro organizaci

- Doporučení a diskuse

Konzultant a podpůrný SW

- Konzultant
  - Ano, pro posouzení současného stavu
  - Ano, pro získání know-how
  - Ne, pro dlouhodobý rozvoj a údržbu metodiky

- Software pro řízení procesů
  - Ano, pokud je dostatečně flexibilní pro dlouhodobou údržbu pracovníky firmy
  - Ne, pokud je vhodný jen pro určitý typ projektů

Diskuse

...děkuji za pozornost