OO sběr požadavků

- vstupem - definice problému
- provedeme doménovou analýzu
- navrhneme základní doménové třídy
- provedeme sběr požadavků, případy užití (use cases)

- předchozí body nemusí být splněny v tomto pořadí, spíše se prolínají a vzájemně doplňují

Příklad zadání – Bankomat (zjednodušený oproti realitě):

Vytvořte software pro síť bankomatů Naší Banky. Bankomaty budou komunikovat s centrálním počítačem banky, který transakce autorizuje a provede změny na účtu. Software centrálního počítače dodá banka. Systém vyžaduje uchovávání záznamů o činnosti a zabezpečení.

Řešení:

Doménová analýza, doménový model

- provedeme doménovou analýzu, cílem maximální porozumění doméně aplikace (seznámíme se s činností bankomatů, jejich zabezpečením apod.)

- navrhneme základní doménové třídy – doménový model, tj. třídy reprezentující objekty relevantní v aplikační doméně (často vytváříme zároveň s případy užití)

- sledujeme podstatná jména v definici problému, věci a místa v aplikační doméně, pro každé vytvoříme předběžnou třídu; slovesa zaznamenáme, aby časem mohla být operacemi
- pouze názvy tříd, asociace mezi nimi, násobnost asociací, výjimečně významné atributy

- doménový model posléze transformujeme (rozvineme) na model analytický (logický) (zároveň s detailním rozborem případů užití, přechod od specifikace k analýze)

  - eliminujeme nepotřebné a chybné předběžné třídy
    - třídy, které nejsou pro aplikaci relevantní, zrušíme
    - pokud předběžná třída popisuje jednotlivý objekt (např. jméno, věk apod.) a nemusí existovat samostatně, prohlásíme jí za atribut
    - pokud předběžná třída popisuje činnost objektu, prohlásíme jí za operaci (např. telefonní spojení může být posloupnost akcí uvnitř telefonní sítě, aktéry jsou telefonní účastníci)

- mezi předběžnými třídami by se neměly objevovat implementační konstrukce (např. seznam, pole, strom, tabulka apod.)

Příklad (k výše uvedenému zadání):

Předběžné třídy vyplývající z definice problému: software, bankomat, centrální počítač, banka, transakce, účet, záznam o činnosti, zabezpečení.

Předběžné třídy vyplývající z aplikační domény: klient, platební karta, stvrzenka, výplata.

Eliminujeme vágní třídy: software, zabezpečení.

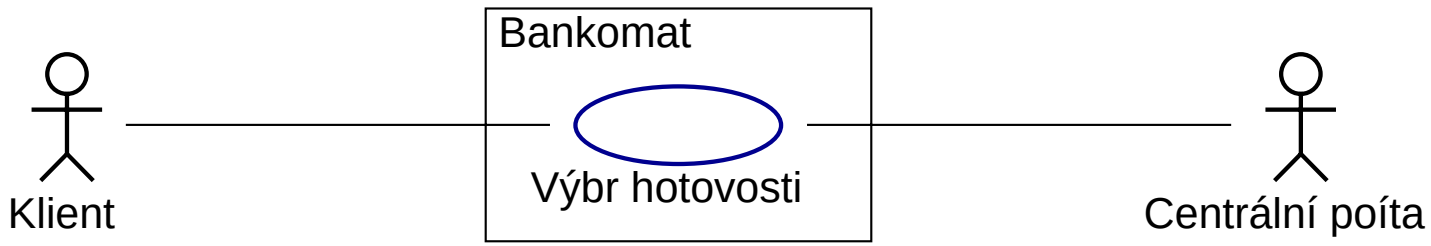
- provedeme sběr požadavků - jak budou potenciální uživatelé využívat systém, případy užití
    - určení aktérů - tj. uživatelů a spolupracujících systémů
        - primární uživatelé, tj. ti, kdo využívají hlavní funkce systému
        - sekundární uživatelé, tj. ti, kdo systém administrují a udržují
        - externí HW nebo SW systémy, se kterými bude navrhovaný systém komunikovat
    - identifikace případů užití, souhrn případů použití by měl pokrývat všechny možné způsoby vyžití systému

- stručný popis účelu případu užití (cca odstavec)

- rozložení případu užití na kroky (-> detailní popisy PU, přechod k analýze)
    - základní posloupnost aktivit, tj. kroky aktéra a odpovědi systému
    - alternativní posloupnosti aktivit

- případy užití, které lze snadno přehlédnout, protože se netýkají primárních funkcí systému:
    - start a ukončení systému
    - administrace systému, např. přidávání nových uživatelů, zálohování dat apod.
    - funkčnost potřebná pro modifikaci chování systému; např. z informačního systému potřebujeme vytvářet nové typy výstupů

Příklad:



Případ použití: *Výběr peněz z bankomatu*.
Aktéři: *Klient, Centrální počítač*
Stručný popis: *Zákazník vloží kartu a požádá o výběr určité částky. Bankomat mu po potvrzení centrálním počítačem požadovanou částku vydá.*

Popis jednotlivých kroků základního scénáře:

1. *Klient vloží kartu. Bankomat kartu přečte a zjistí její sériové číslo.*
2. *Bankomat požádá uživatele o zadání PIN; uživatel zadá "1234".*
3. *Bankomat ověří číslo karty a PIN u centrálního počítače.*
4. *Bankomat požádá o zadání velikosti částky; uživatel zadá 1000 Kč*
5. *Bankomat požádá centrální počítač o provedení transakce; centrální počítač transakci provede a vrátí nový zůstatek účtu.*
6. *Bankomat vydá částku, vytiskne stvrzenku a vrátí kartu.*

Alternativní scénáře:

A1. Uživatel vloží kartu. Bankomat kartu přečte a zjistí její sériové číslo.
A2. Bankomat požádá uživatele o zadání PIN; uživatel zadá "9999".
A3. Bankomat se pokusí ověřit číslo karty a PIN u centrálního počítače; centrální počítač je odmítne.
A4. Bankomat oznámí, že PIN bylo chybné, a vyzve uživatele, aby ho zadal znovu; uživatel zadá "1234", což bankomat úspěšně ověří u centrálního počítače.
A5. Bankomat požádá o zadání velikosti částky; uživatel zadá 1000 Kč.
A6. Bankomat požádá centrální počítač o provedení transakce; centrální počítač transakci odmítne pro nízký zůstatek na účtu.
A7. Bankomat vytiskne stvrzenku a vrátí kartu.

- výsledkem je pouze soubor příkladových scénářů, není úplnou specifikací systému

OO Analýza

- vstup - popis případů užití a doménové třídy
- případy užití zjemníme pomocí dynamických modelů,
    - sekvenčních diagramů (synchronní chování)
    - stavových diagramů a diagramů spolupráce (asynchronní chování)
- vytvoříme diagram analytických tříd

- rozebíráme postupně případy užití
- popíšeme, který objekt je zodpovědný za které chování v případu užití
    - pomocí CRC karet
    - pomocí sekvenčních diagramů

CRC karty (Class/Responsibilities/Collaborators)

- pro třídy vytvoříme kartičky, nahoru napíšeme jméno třídy, vlevo zodpovědnost třídy, vpravo spolupracující třídy
- při průchodu scénářem přidáváme nové odpovědnosti existujícím třídám; pokud neumíme, rozdělíme existující třídu na dvě nebo založíme novou
- nevýhoda - vazby mezi třídami nejsou znázorněny graficky
- pro projekt střední velikosti získáme 30 až 100 tříd

Příklad:

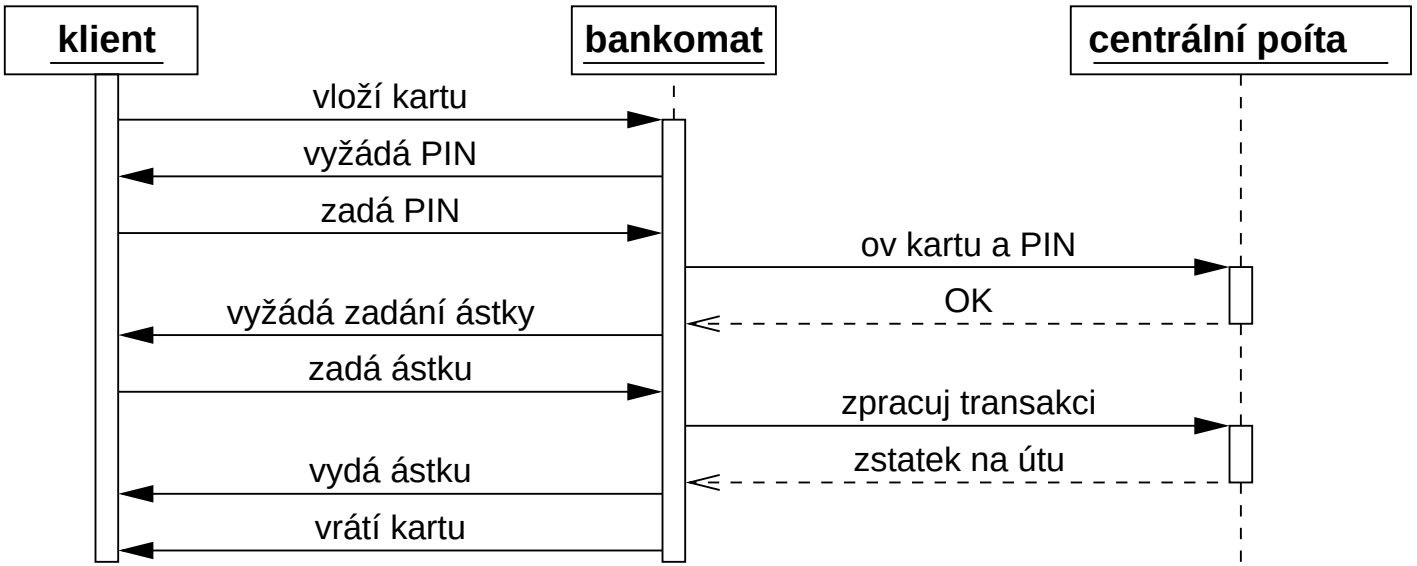
třída: Centrální počítač

odpovědnost:	**spolupracuje s:**
ověří číslo karty a PIN	bankomat
provede transakci	
vrátí nový zůstatek účtu	

Dynamické modely UML

- rozebereme základní sekvenci z případu použití
- alternativní sekvence většinou popisuje reakci na chyby, začleníme později
- opravujeme seznam tříd

Příklad (sekvenční diagram pro případ použití "Výběr peněz z bankomatu")



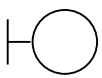
Konstrukce diagramu tříd (analytického modelu)

- na počátku je třída popsána pouze názvem
    - zkontrolujeme, zda není jiným názvem pro existující třídu (ponecháme lepší název), zda není rolí (jméno třídy má popisovat její esenci, nikoli pouze roli v sekvenčním diagramu apod.)
- ke třídě najdeme logické atributy = informace, která se nebude používat samostatně, ale je silně svázána s objektem (např. jméno, věk, adresa budou atributy nějaké Osoby)

- třídy rozdělíme podle následujících stereotypů:
    - hraniční třídy
        - všechno s čím aktéři přímo komunikují, např. formuláře, komunikační protokoly, rozhraní pro tiskárnu,
        - v UML můžeme označovat stereotypem <<boundary>> nebo níže uvedenou značkou (případně oběma způsoby zároveň)

- entitní třídy
  - informace, kterou systém udržuje delší dobu -> entitní objekty (často odpovídají objektům reálného světa, např.: Student, Předmět, BankovníÚčet apod.),
  - entitní objekty samy od sebe neiniciují komunikaci,
  - v UML stereotyp <<entity>>

- řídící třídy
  - koordinují chování v systému.
  - např. třídy obsahující řídící logiku, používající nebo nastavující obsah entitních tříd
  - obvykle se týkají realizace jediného případu použití
  - pokud je chování jednoduché, nemusejí být zapotřebí
  - v UML stereotyp <<control>>


a) hrani ní t ída
boundary class

b) entiní t ída
entity class

c) ídící t ída
control class

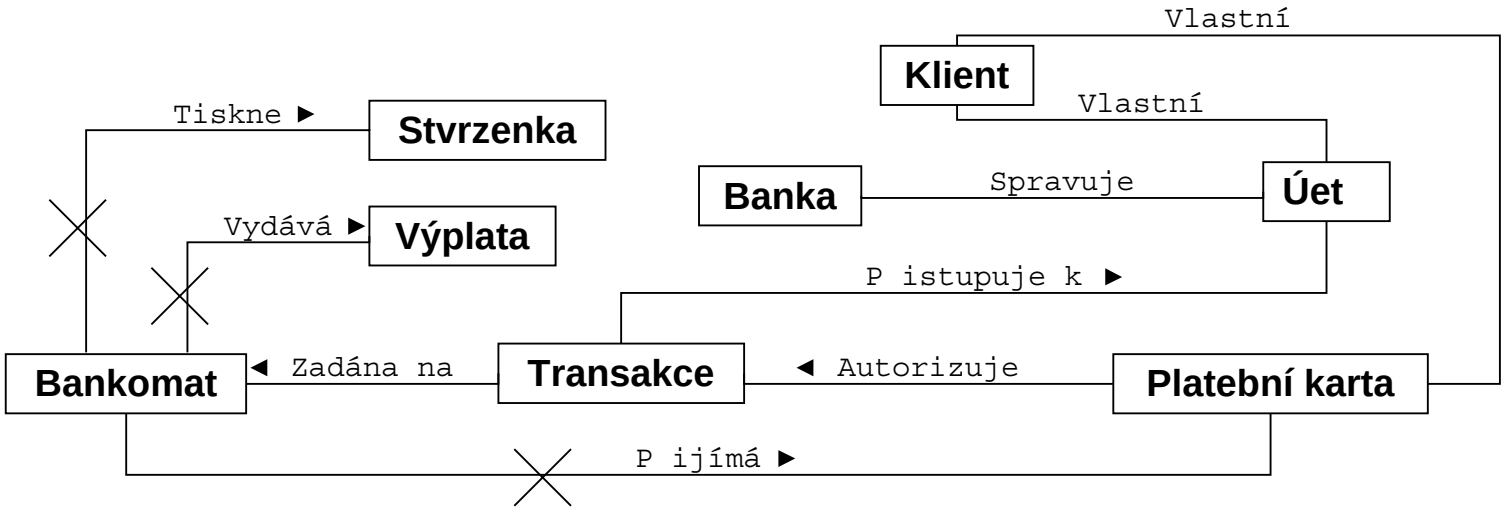
Příklad (Bankomat):

- entitní objekty - Klient a Účet.

- výběr peněz ověřuje centrální počítač -> aktér (mimo hranice našeho systému), komunikace s tímto aktérem probíhá prostřednictvím tzv. ATM sítě, její rozhraní -> hraniční objekt

- hraniční objekty uživatelského rozhraní -> klávesnice, obrazovka, čtečka platebních karet, výdejní automat bankomatu, tiskárna stvrzenek apod. (vytvoření náčrtku, prototypu, který by si uživatel mohl vyzkoušet apod.)

- vytváříme předběžné asociace mezi třídami
    - mohou odpovídat fyzickému umístění nebo vztahu vlastnictví (má spojení s, je součástí, je obsažen v, patří), řízení a komunikaci (řídí, komunikuje s)
    - měly by být pojmenovány účelem asociace
    - nesnažíme se rozlišit asociace a agregace, neuvádíme násobnosti?? ani průchodnost

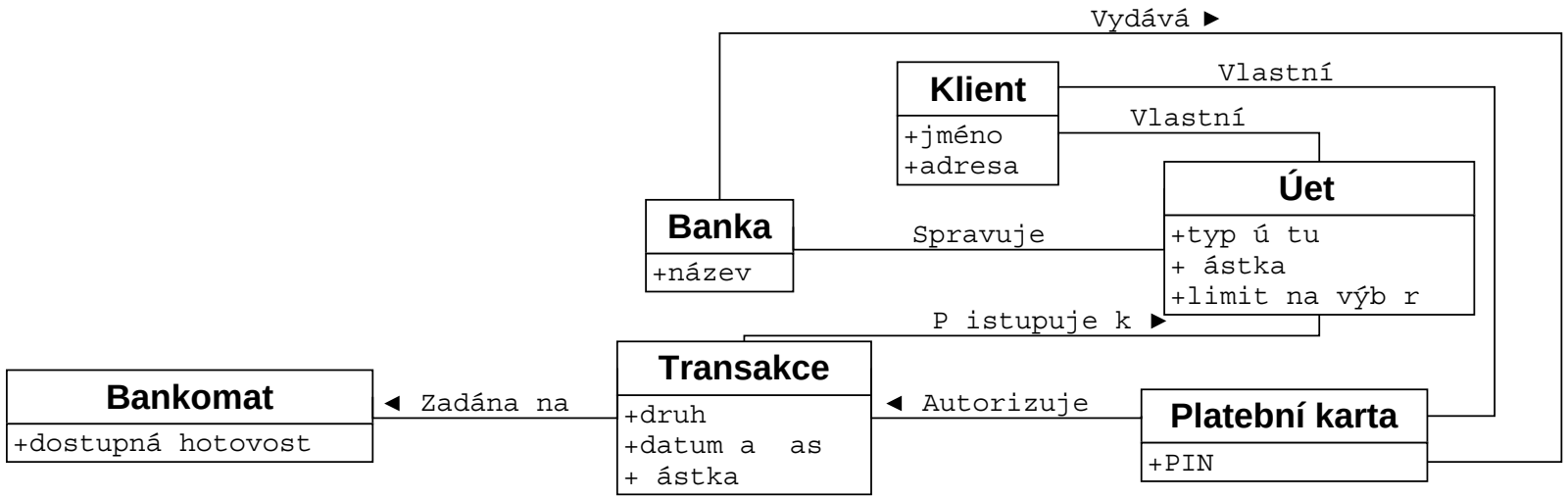


- zrušíme nepotřebné nebo nesprávné asociace

  - zrušíme asociace, které nejsou podstatné pro řešený problém, jsou redundantní nebo které představují popis implementace
  - zrušíme předběžné asociace, představující jednorázové akce; např. bankomat sice přijímá platební karty, ale mezi bankomatem a platební kartou neexistuje trvalý (strukturální) vztah; (v obrázku jsou zrušené asociace škrtnuty)
  - vyhýbáme se asociacím mezi dvěma řídícími objekty a mezi hraničním a řídícím objektem, protože oba typy vztahů trvají krátkou dobu
  - vyhýbáme se ternárním a vyšším asociacím, většinou je lze přestrukturovat na binární asociace nebo případně popsat asociační třídou

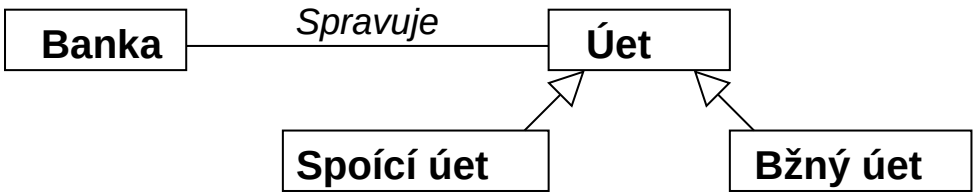
- vytváříme agregace a kompozice
    - agregace - objekt je součástí jiného nebo objekt je podřízený jinému (např. pokud se některé operace nad celkem provedou nad všemi částmi)
    - kompozice, pokud je silné vlastnění (strom součástí, součásti nelze sdílet) a stejná doba života

- hledáme primární atributy objektů a asociací
    - nejdůležitější logické atributy, relevantní pro aplikace - viditelné vlastnosti jednotlivých objektů (jméno, rychlost, barva apod.)


Vydává ►
Klient
+jméno
+adresa
Vlastní
Vlastní
Úet
+typ ú tu
+ ástka
+limit na výb r
Banka
+název
Spravuje
P istupuje k ►
Transakce
+druh
+datum a as
+ ástka
Bankomat
+dostupná hotovost
◄ Zadána na
◄ Autorizuje
Platební karta
+PIN

vytváříme hierarchii dědičnosti, postup dvěma směry

- zdola nahoru, tj. zobecněním - hledáme třídy se společnými vlastnostmi, které vyjmeme do nadtřídy (např. Běžný účet a Spořící účet)
- shora dolů, tj. specializací - existující třídy zjemníme pomocí podtříd
- vytvořená hierarchie nesmí být zbytečně hluboká (zvláště v případě, že cílový jazyk nebude OO)
- pokud chceme využít polymorfismus, vedeme asociaci k rodičovské třídě



- výsledné diagramy specifikují strukturu systému, ale nikoli důvody, které nás k ní vedly; ty bychom měli také zdokumentovat

- cílem analýzy je dostatečně popsat problém a aplikační doménu bez závislosti na konkrétní implementaci (i když v praxi není možné tento cíl vždycky splnit).

- analýza není jednoduchá sekvence akcí s jasným koncem, ale iterativní proces, ke kterému se musíme vracet po většinu doby vývoje systému.

- výrok: ... *analysis is frustrating, full of complex interpersonal relationships, indefinite, and difficult. In a word, it is fascinating*.