Typy tabuliek v MySQL

- MyISAM (používa sa defaultne)
  - nepodporujú transakcie
  - Veľkosť tabuliek obmedzená iba OS (t.j. kľudne radovo GB ...)
  - max 64 kľúčov na tabuľku
  - fyzicky sú tabuľky v súboroch/blokoch realizované ako: statické, dynamické (obsahujú varchar/text/blob), komprimované (Huffman)
    - prehľadávanie v statickej tabuľke je rýchlejšie, CACHE sa nefragmentuje
    - komprimované tabuľky sa dajú iba čítať
- InnoDB
  - podporujú transakcie
  - Podporujú uzamykanie na úrovni riadku
  - podpora cudzích kľúčov
- BerkleyDB, MERGE, HEAP – nepodporujú transakcie

Ako na transakčnosť v MySQL

Problém A: Chceme pripočítať na účet niekomu 500 EUR

1) Riešenie 1 – veľmi problematické

V aplikácii zavoláme :

```
“select zostatok from ucty where c_uctu=12345;”
```

k získanej čiastke (napr. 1000 EUR) v aplikácii pridáme 500 EUR, a vysledok 1500 EUR zapíšeme:

```
“update ucty set zostatok = 1500 where c_uctu = 12345;”
```

2) Riešenie 2 – Atomické – stačí

V aplikácii zavoláme :

```
“update ucty set zostatok = zostatok + 500 where c_uctu = 12345;”
```

Problém B: Chceme previesť z účtu na účet 500 EUR
(ukážeme si 4 riešenia)

1) Riešenie 1 – Atomické časti – veľmi problematické

V aplikácii zavoláme :

```
“update ucty set zostatok = zostatok - 500 where c_uctu = 12345;”
“update ucty set zostatok = zostatok + 500 where c_uctu = 54321;”
```

2) Riešenie 2 – Atomické ako celok– stačí

V aplikácii zavoláme :

```
“update ucty as z, ucty as na
   set z.zostatok = z.zostatok – 500, na.zostatok = na.zostatok+ 500
where z.c_uctu=12345, na.c_uctu=54321;“
```

Ako na transakčnosť v MySQL

- Univerzálne riešenie predchádzajúcich problémov – transakcia
- Problém B, riešenie 3 (explicitné použitie transakcie):

V aplikácii zavoláme :

```
“start transaction;
update ucty set zostatok = zostatok - 500 where c_uctu = 12345;
update ucty set zostatok = zostatok + 500 where c_uctu = 54321;
commit;”
```

- Autocommit
  - V MySQL je defaultne nastavený autocommit
  - autocommit a jeho nastavenia platia pre každé pripojenie zvlášť
  - start transaction/commit tento autocommit dočasne vypína
  - explicitne sa dá zmeniť pomocou „set autocommit=0;“, „set autocommit=1;“
  - pri vypnutom autocommit sa zmeny zapíšu po zadaní „commit“
- Namiesto commit môžeme napísať „rollback;“ eventuálne zmeny sa zahodia.
- Problém B: riešenie 4 (explicitné použitie zámkov):

V aplikácii zavoláme :

```
“lock tables ucty write;
update ucty set zostatok = zostatok - 500 where c_uctu = 12345;
update ucty set zostatok = zostatok + 500 where c_uctu = 54321;
unlock tables;”
```

Ako na transakčnosť v MySQL

- Zámky – všetky potrebné zámky, ktoré sú požadované pre nejaké operácie je treba zadať NARAZ v jednom riadku, napr:

```
“lock tables ucty write, nieco_ine read;”
```

Pozn. opakovanie:

- Write zámok – nikto iný do tabuľky nemôže zapisovať ani čítať z celej tabuľky
- Read zámok – každý môže čítať ale nikto nemôže zapisovať

Izolácia tranzakcií v MySQL

- Ako sme už spomenuli pri InnoDB tabuľkách je defaut stav Repeatable read (MySql default)
- V praxi je treba prejsť na „serializable“ málokedy, lebo
  - MySQL zamyká na úrovni riadkov
  - zamyká aj kľúče, t.j. ak vyhľadávame pomocou kľúča, fantómy by sa nemali objaviť
- MySQL podporuje celú tabuľku:

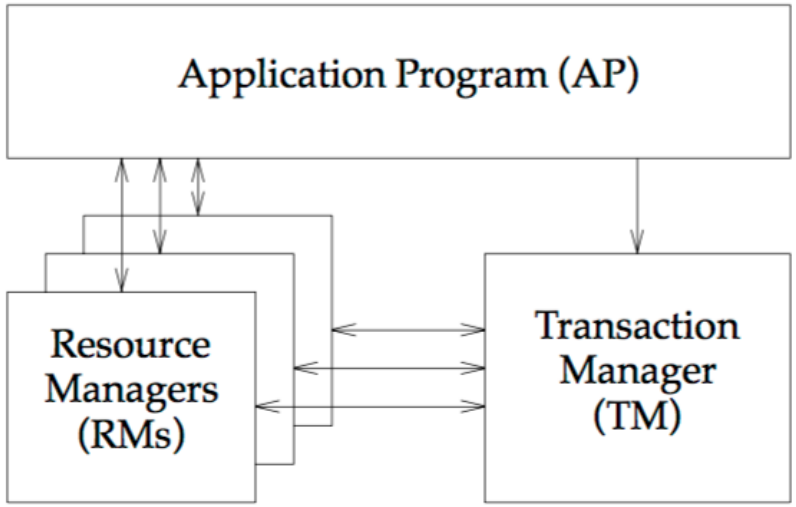
Izolačná úroveň tranzakcií	Anomálie, ktoré sa môžu vyskytovať		
	Špinavé čítanie	Neopakovateľné čítanie	Fantómové čítanie
Read uncommitted	Áno	Áno	Áno
Read commited (MSSql default)	Nie	Áno	Áno
Repeatable read (MySql default)	Nie	Nie	Áno
Serializable	Nie	Nie	Nie

OLAP a OLTP systémy

- SRDB má sa môže deliť z hľadiska využitia aj na SRDB typu
  - **OLTP** (online Transactional Processing)
    - Práca nad údajmi každodenného charakteru
    - Údaje väčšinou v 3 normálnej forme
    - Dôraz na transakčné schopnosti systému
    - Treba rátať s potencionálne veľkým množstvom rollback operácií
  - **OLAP** (online analytical processing)
    - Práca nad údajmi na štatistické a informatívne účely (reporting)
    - Väčšinou dávkové spracovanie údajov (napr každú noc o 3:00)
      - Ak pracujeme s celou tabuľkou, indexy môžu byť príťažou- niekedy je lepšie ich zahodiť
    - Údaje môžu byť naschvál denormalizované z 3NF na 2NF či dokonca 1NF
      - riadená redundancia kvôli zväčšeniu výkonu
      - pracné udržanie referenčnej integrity

Distribuované, Globálne, XA tranzakcie

- znamenajú to isté – t.j. transakcia je medzi viacerými databázami (resp. dátovými úložiskami)
- De facto štandardom je DTP (Distributed Transaction Processing) model XA (eXtended Architecture)
  - Popisuje rozhranie medzi transakčným manažérom (TM) a resource manažermi (RM)
  - databázy sú v roli RM, ak spĺňajú uvedený štandard sú „XA compliant“
    - Oracle, DB2, MySql sú XA compliant
  - Používa tzv. dvojfázové potvrdzovanie (**2 phase commit**)



    - Predpokladá perzistentné úložisko v každom uzle a schopnosť komunikácie ľubovoľných 2 uzlov
    - Používa blokujúce volania
    - Ak zlyhá TM aj nejaký RM naraz, nevie sa zotaviť (toto rieši až 3 fázový commit)
    - FÁZA1:
      - TM posiela „prepare to commit“ všetkým RM, od každého očakáva súhlas/nesúhlas
    - FÁZA2:
      - Ak TM od každého dostal súhlas, posiela „commit“ a očakáva odpoveď
      - Ak TM aspoň od jedného dostal nesúhlas (resp. nastal timeout), posiela „rollback“ tým čo poslali súhlas (resp. všetkým) a očakáva odpoveď
- JTA – Java Transaction API (balíky javax.transaction, java.transaction.xa) – JSR 907
  - Je súčasťou J2EE, existujú aj standalone frameworky ako napr. Atomikos, JBossTS, ...

Paralelné spracovanie údajov na aplikačnej úrovni a perzistencia

- nemusíme sa snažiť o čo najväčšiu izoláciu transakcií v databáze (zbytočne veľká réžia)
  - v aplikácii so zvážením aplikačnej logiky toto často vieme robiť efektívnejšie
- JPA – Java Persitence API (balík javax.persistence)
  - Podľa JSR 220 (JPA 2.0 je JSR 217)
  - Súčasť J2EE
  - implementuje ho napr. ORM (object relational mapping) framework Hibernate
  - mapuje java triedy na databázové tabuľky
    - dosahuje sa to pomocou XML based konfigurácie, alebo pomocou Java Anotácií
    - hibernate si vie udržiavať DB schému
  - Objekty v aplikácii sú vytvárané princípmi OOP v databáze princípmi normalizácie → rôzne požiadavky na reprezentáciu. Tento problém sa volá "object-relational impedance mismatch".
  - Tento problém sa odstraňuje mapovaním – **KTORÁ JAVA TRIEDA SA MÁ ULOŽIŤ V KTOREJ TABUĽKE**.
- Pri paralelnom spracovaní nestačí len nakonfigurovať JPA, môžu byť potrebné aj zmeny v aplikácii
- **Aspekt - Zamkýnanie** – dosť ignorované pri vývoji, potom majú vyvinuté systémy často problém ísť do produkcie – problémy s konzistenciou, výkonové problémy

Zamykanie na úrovni JPA

- Dva základné prístupy
  - **optimistické zamykanie** – predpokladá, že dáta pravdepodobne nebudú modifikované medzitým čo ich prečítam a pokúsim sa aktualizovať.
    - Najbežnejší postup (predovšetkým pri riešení perzistencie WEB aplikácií)
    - Stratégia je, tesne pred zápisom si skontrolovať, či naozaj údaje sú také ako pri čítaní. Keď stratégia zlyhá je vyvolaná OptimisticLockException, táto výnimka môže byť ošetrovaná podobne ako RollbackException,
  - **presimistické zamykanie** – zamykanie pred začiatkom editovania objektu
    - obvykle na úrovni zamknutia databázového riadku napr. v SQL pomocou „`SELECT ... FOR UPDATE`“ kým nie je tranzakcia potvrdená
  - „pštrosie“ zamkýnanie (ostrich locking) – strčíme hlavu do piesku a nezamkýname
    - vhodné, keď robíme prototyp
    - ak predsa len dôjde k problému paralelného prístupu potom
      - pri editovaní toho istého poľa „vyhráva posledný“
      - pri editovaní rôznych polí – závisí výsledok od konkrétnej JPA implementácie
- od JPA 2.0 je podporované už aj pesimistické zamkýnanie

Všeobecný scenár zlého použitia JPA

1) Aplikácia načíta údaje z databázy a ukáže ich užívateľovi napr. prostredníctom WWW prehliadača
2) Užívateľ zmení údaje
3) Aplikácia naštartuje aktualizáciu údajov v DBS
4) DBS začne transakciu na aktualizáciu údajov, zaktualizuje údaje a potvrdí transakciu

Čo sa môže udiať a ako to riešiť:

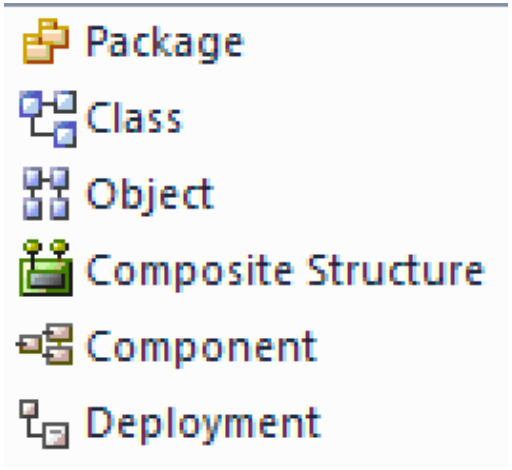
a) Zámok bol použitý iba v kroku 4, teda akékoľvek zmeny v údajoch v DBC počas krokov 1, 2, 3 sa možno prepíšu bez zistenia akéhokoľvek konfliktu
b) Aplikácia musí byť prepísaná aby transakcia začala už v kroku 1, napr. pomocou optimistického zámku
c) Ak medzi krokom 1 a 3 užívateľ išiel na obed, tak pri optimistickom zamknutí sa potencionálny problém (niekto medzitým prepísal hodnoty) ZISTÍ (OptimisticLockException), keď sa užívateľ vráti a spustí krok 3, 4
d) Aplikácia sa môže konflikt pokúsiť riešiť automaticky, ale každopádne je vhodné minimálne informovať užívateľa.
e) Podobne sa situácia rieši v systémoch na kontrolu verzií (VCS)!
f) Typicky krok (1) a (4) sú z pohľadu WEB aplikácie rôzne databázové transakcie, t.j. použiť princípy optimistického zamkýnania sú jediné schodné...

Jazyk UML – unified modelling language

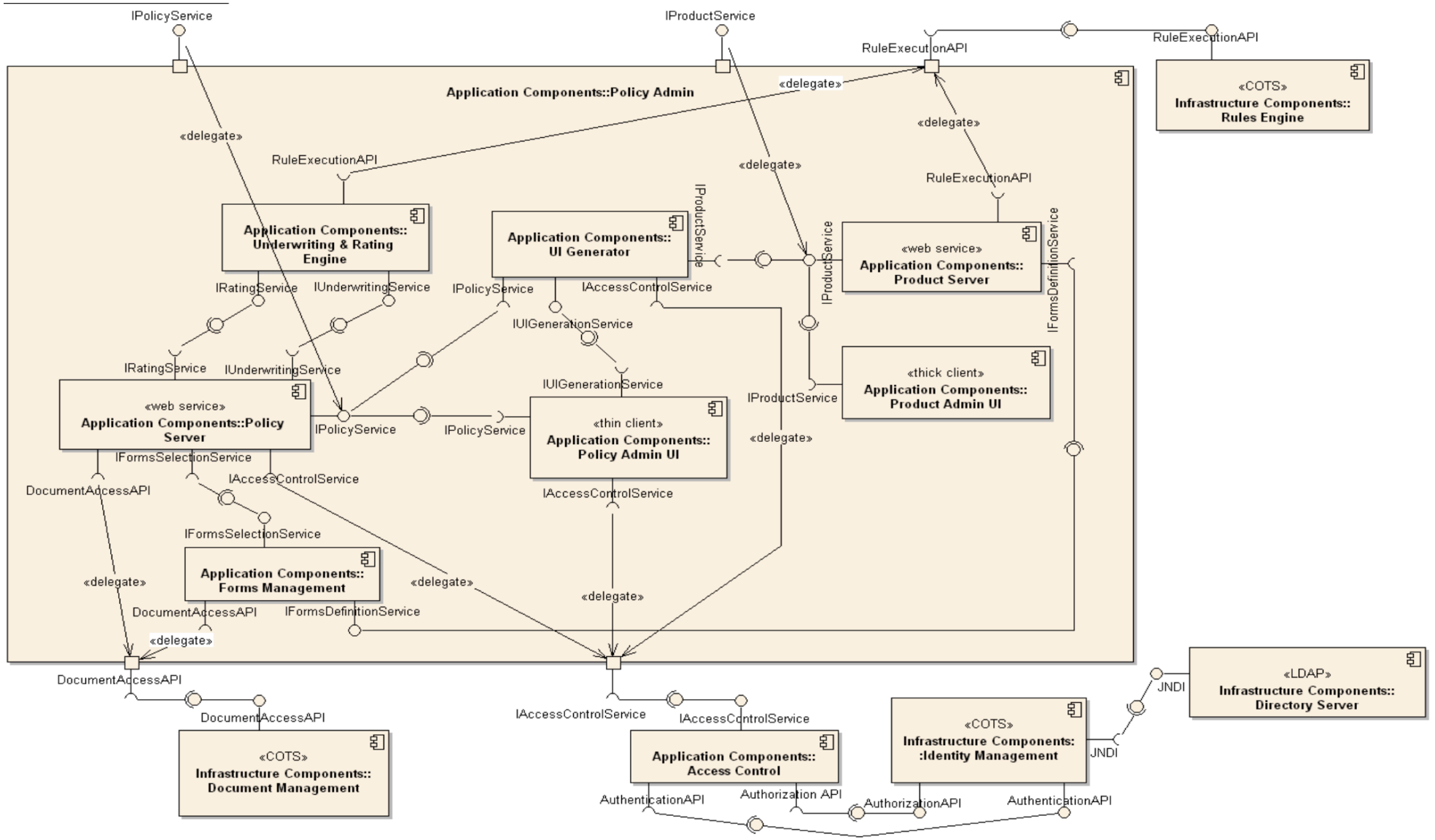
- Všeobecný modelovací jazyk pre SW inžinierstvo
- Od 1997 Je to štandard skupiny Object Management Group (OMG)
- Nie je to metóda tvorby architektúry, to špecifikujú RUP, TOGAF, ...
- UML poskytuje prostriedky ako architektúru vizuálne popísať - špecifikuje veľa druhov diagramov

Štrukturálne UML diagramy

- Diagram tried (class diagram)
  - Vyjadrujú logickú štruktúru systému, aké atribúty majú a ako sa správajú (už sme mali veľa príkladov ...)
- Diagram komponentov
  - Závislosti medzi komponentami a ich organizácia
- Diagram nasadenia
  - Ako a kde je čo nasadené, čo sa kde vykonáva
- Diagram balíkov
  - Organizácia prvkov do balíkov a závislosti medzi nimi
- Diagram objektov
  - inštancie tried a ich vzťahy v nejakom čase
- Kompozitný diagram
  - odráža internú spoluprácu tried, rozhraní a komponentov a tým popisuje funkcionalitu

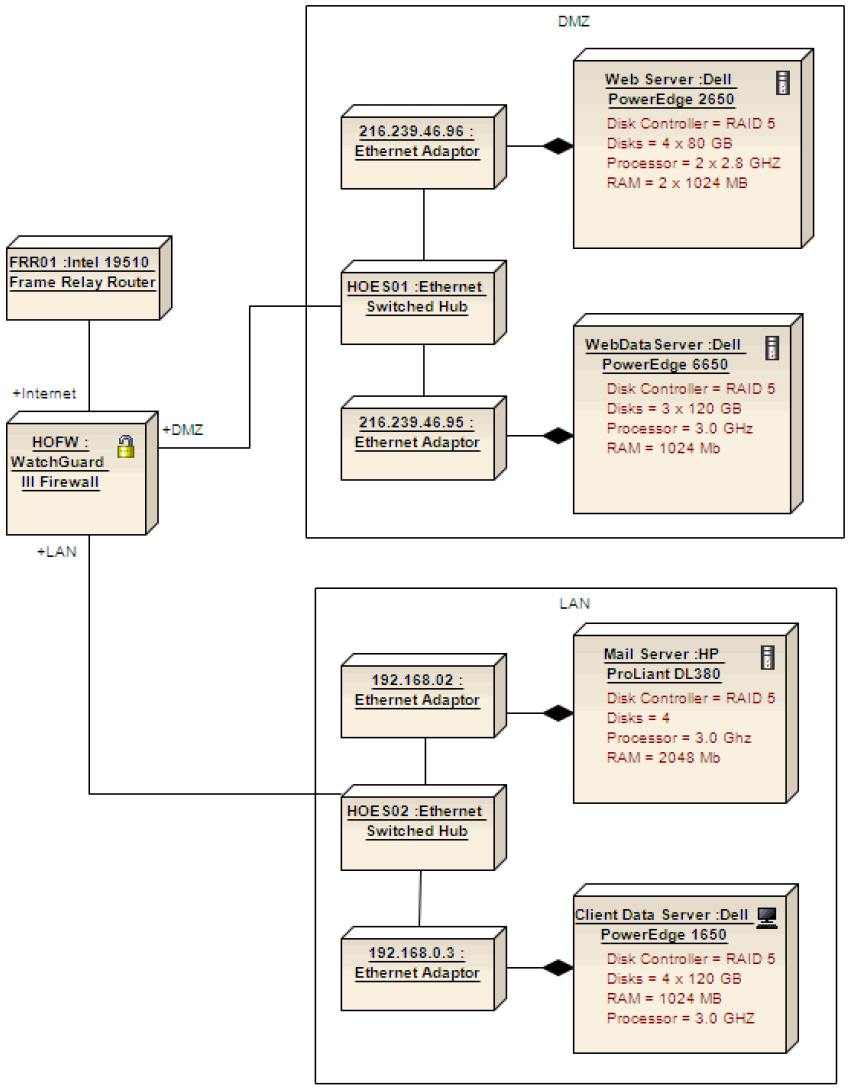


Príklad – diagram komponentov - príklad [AdminComponent]:

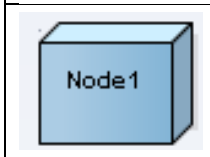


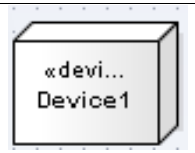
Vysvetlivky	- port	- rozhranie	Komponent	Poskytované Rozhranie	Požadované rozhranie

Príklad - Diagram nasadenia (deployment diagram)



Hardvérové zariadenia sú zobrazené ako uzly. Medzi uzlami sú vzťahy, ktoré odpovedanú fyzickému nasadeniu zariadení.



uzol,


zariadanie,


prostredie, ...

Príklad Diagram balíkov (package diagram)

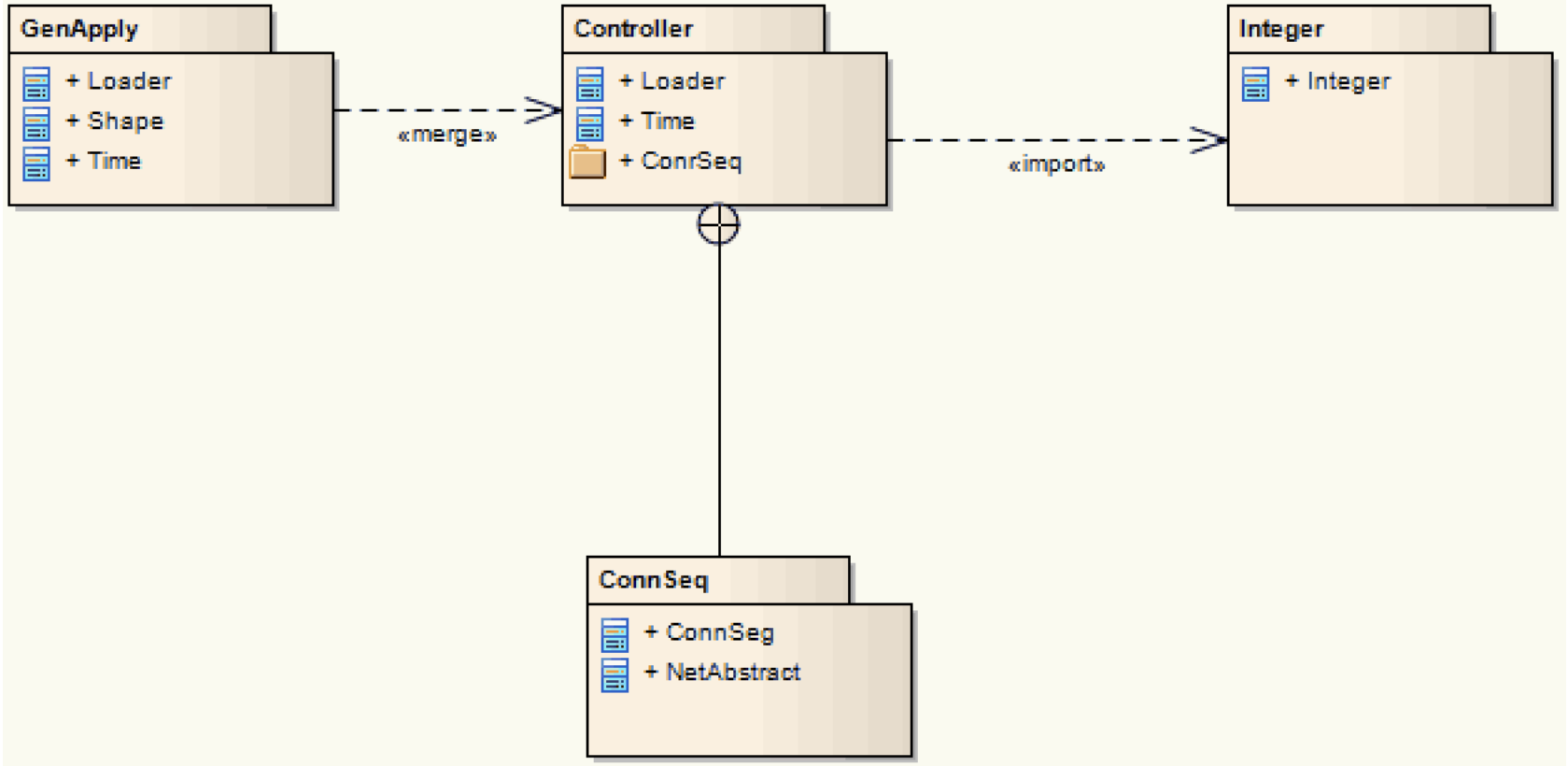
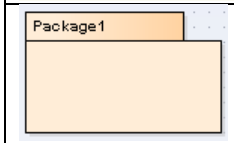


Diagram balíkov znázorňuje organizovanie elementov do balíkov a závislosti medzi balíkmi. Vizualizuje aj priestory mien (namespaces).



balík

Diagram objektov (object diagram)

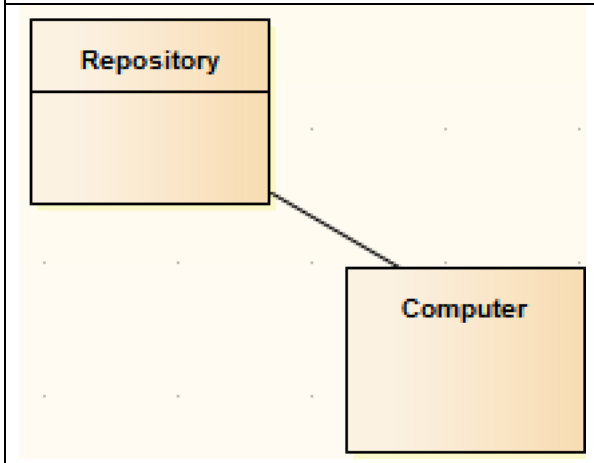
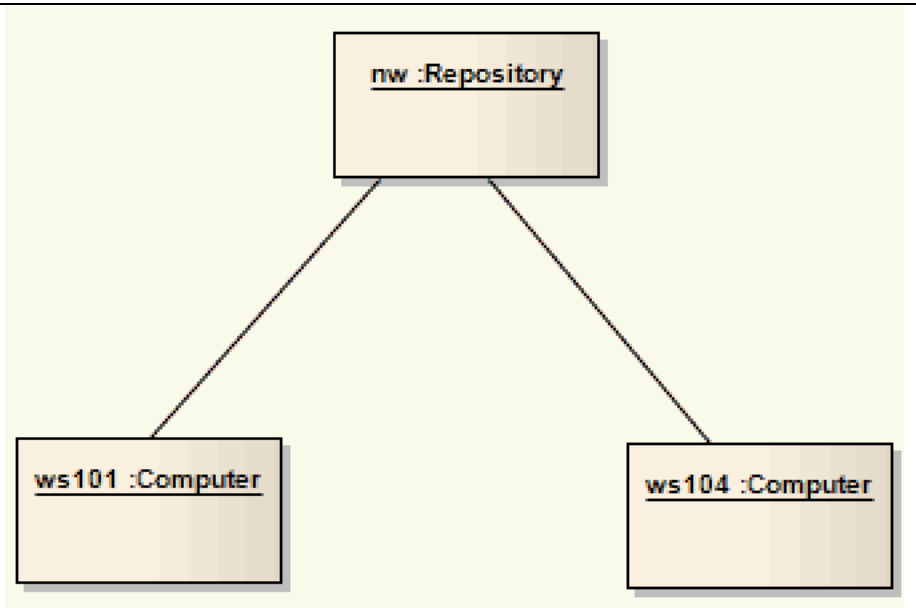
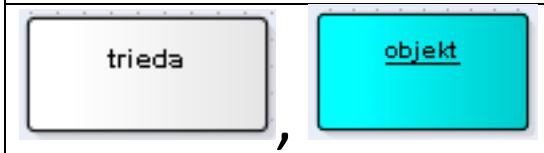
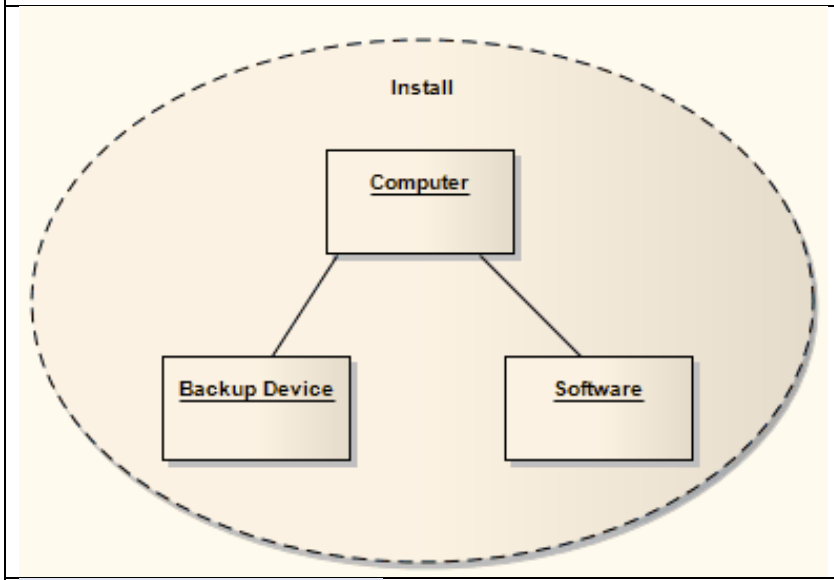
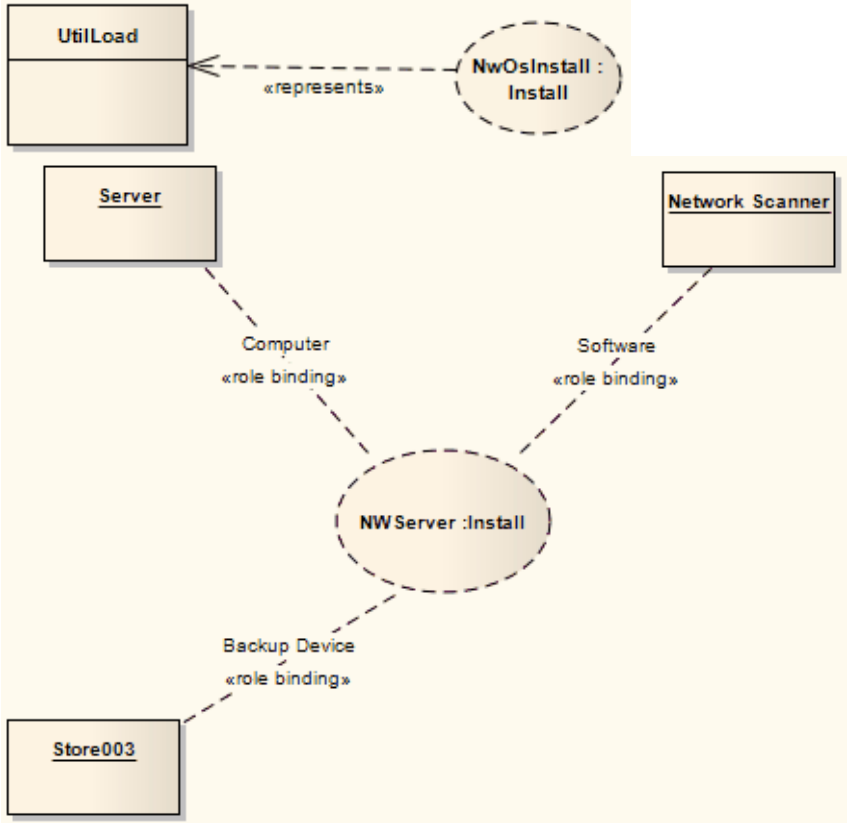
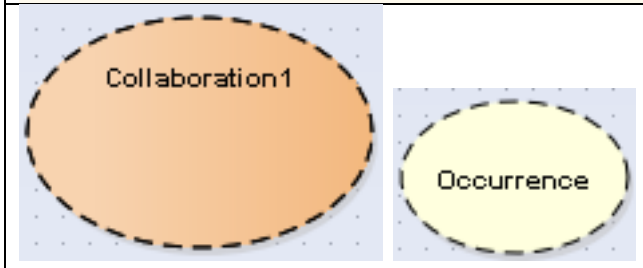
Diagram tried	Diagram objektov
	
	

Diagram objektov

- znázorňuje inštancie tried a ich vzťahy v nejakom čase
- objekty majú názvy, používa sa notácia <názov_objektu> : <názov_triedy>

Kompozitný diagram (composite structure diagram)

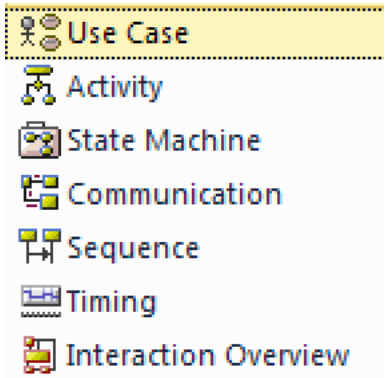
- odráža internú **spoluprácu** tried, rozhraní a komponentov a tým popisuje funkcionalitu

Definovanie spolupráce (collaboration)	Využitie zadefinovanej spolupráce (collaboration use / occurecne)
	
	

Na rozdiel od diagramov tried, ktoré popisujú statické štruktúry a vzťahy

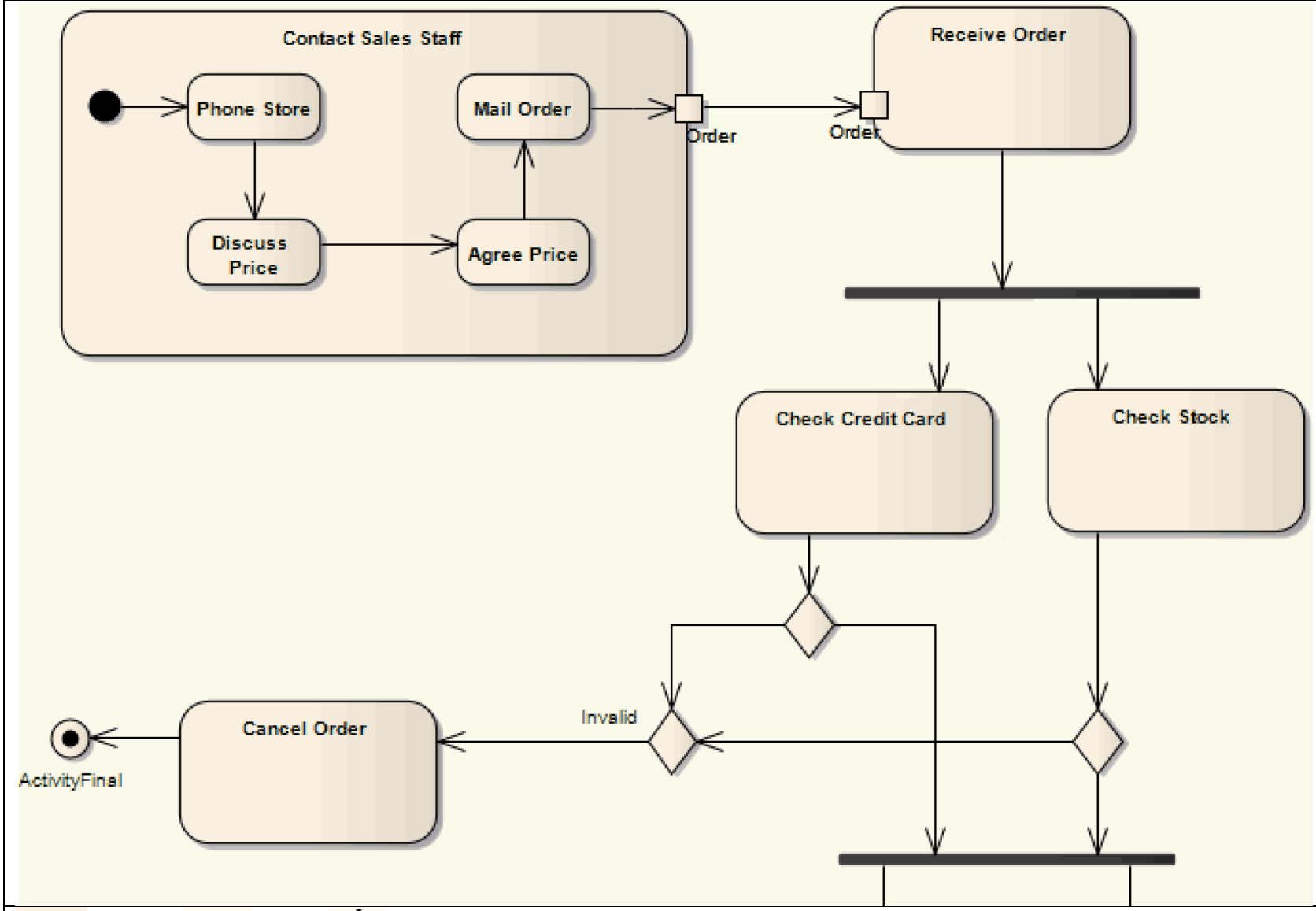
- Kompozitné diagramy modelujú špecifické využitie štruktúry, Run –time vzťahy, vzory použitia, ...
- Bests practice je modelovať **„spolupráce“ aby vytvárali znovu využiteľné vzory**

Behaviorálne UML diagramy



- Diagramy aktivít
  - Modelujú správanie sa systému a spôsob akým je toto správanie previazané s tokom informácií v systéme
- Diagramy použitia (Use case diagram)
  - previazania medzi aktérmi a činnosťami, ktoré pomocou systému môžu vykonávať. Popisujú sa funkčné požiadavky na systém – ako aktéri so systémom pracujú a ako systém reaguje.
- Stavové diagramy
  - Stavové diagramy ilustrujú, ako sa element môže presúvať medzi stavmi, sú vysvetlené spúšťače a podmienky (tento diagram sme využili pri popise stavov tranzakcie)
- Časové diagramy
  - Správanie sa viacerých objektov v nejakom časovom úseku
- Sekvenčné diagramy
  - Popisujú správanie sa pomocou sekvenčných krokov v čase.
- Komunikačné diagramy
  - Popisujú interakcie medzi elementami v čase – vývoj vzťahov medzi objektami
- Diagrami s prehľadom interakcii

Diagram aktivít (activity diagram)



Diagramy aktivít sú používané na modelovanie správania sa systému. Môžu sa takto popisovať procedúry. Aktivity môžu byť súčasťou procesov, workflowov atď...

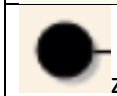
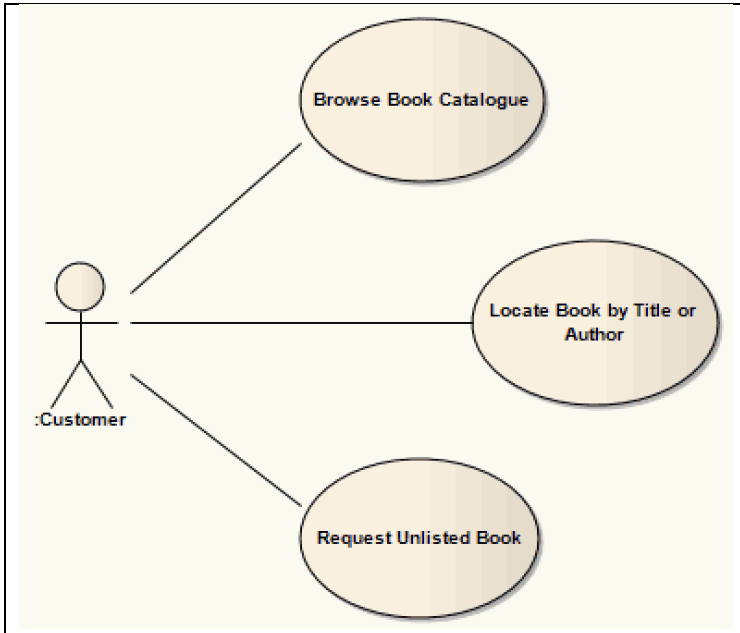
 začiatok,  koniec,  rozhodnutie, fork/spájanie,  očakávanie (vstupné/výstupné parametre),  aktivia

Diagram prípadov použitia (use case diagram)



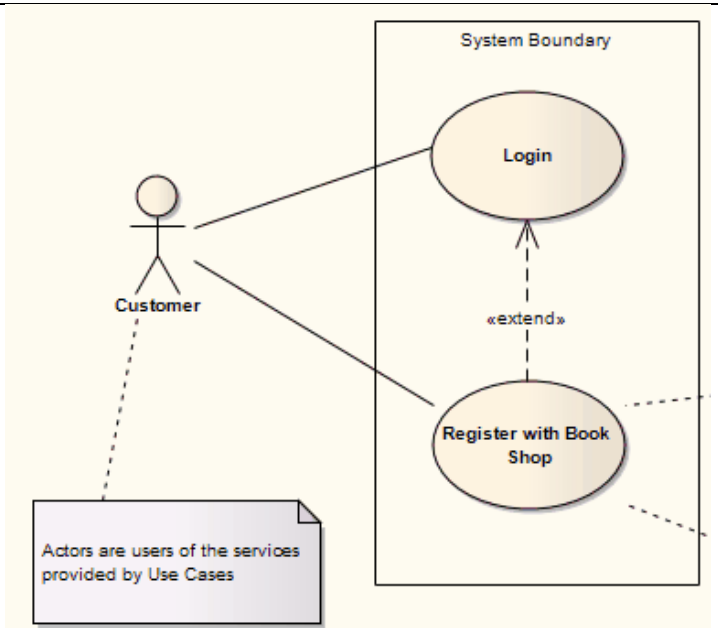
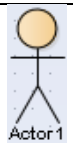
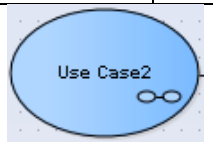


Diagram prípadov použitia mapuje previazania medzi aktérom a činnosťami, ktoré pomocou systému môže vykonávať. Dá sa použiť ako

- popis funkčných požiadaviek na systém
- popis interakcie aktérov so systémom
- popis odpovedí systému

prípad použitia (UC),


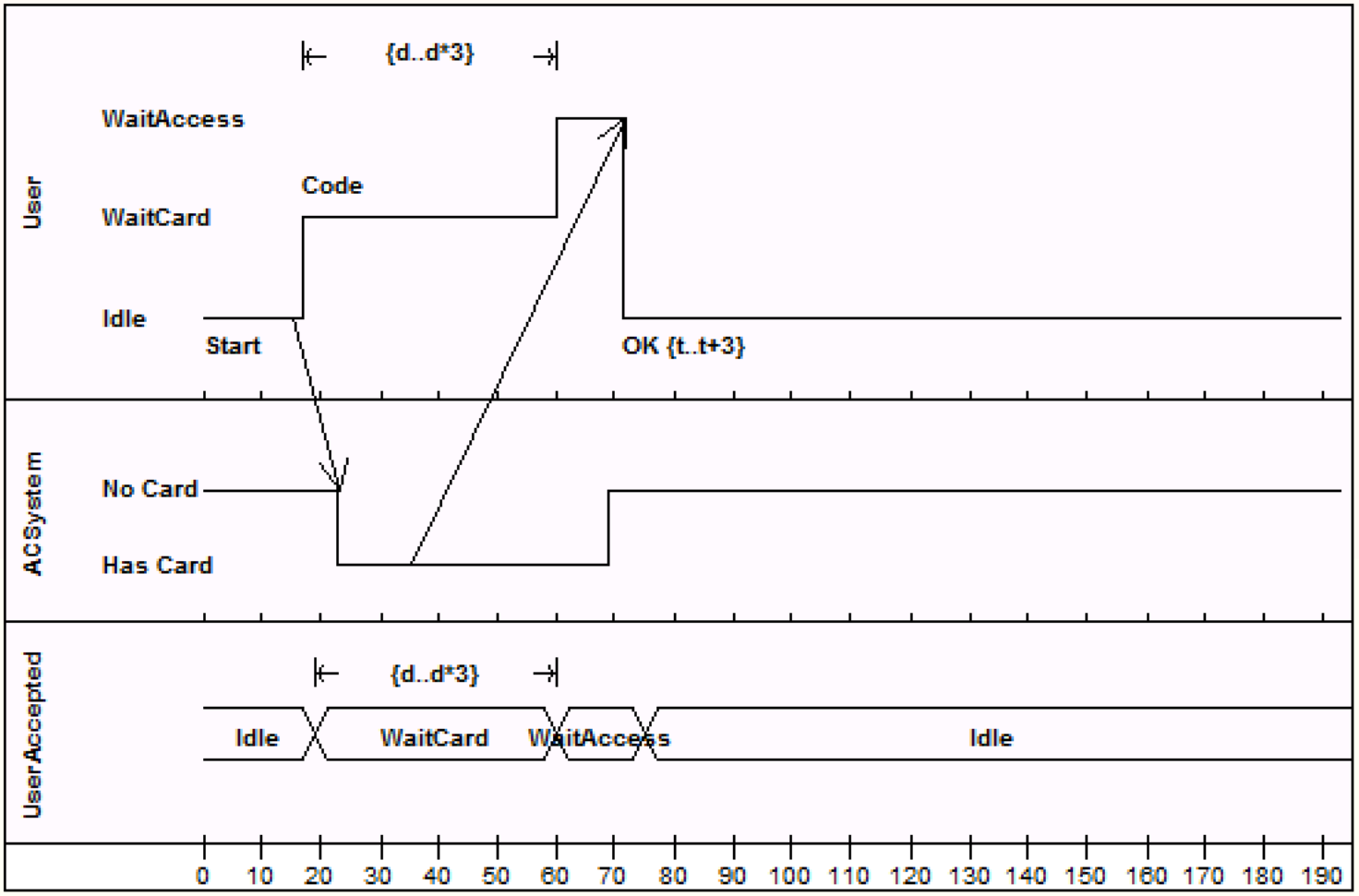
aktér,


poznámka,


UC zo štruktúrou

Pozor: každý prípad použitia (ako kompozitný element – v EA klik pravou myšou -> New child ...) môže obsahovať kombináciu dcérskych diagramov (Sequence, Communication, Activity, State Machine ...), ktoré vo väčšom detaile definujú ako sa to má implementovať. Samotná implementácia prípadov použitia je realizovaná pomocou tried, komponentov, rozhraní, ... v ich vlastných diagramoch.

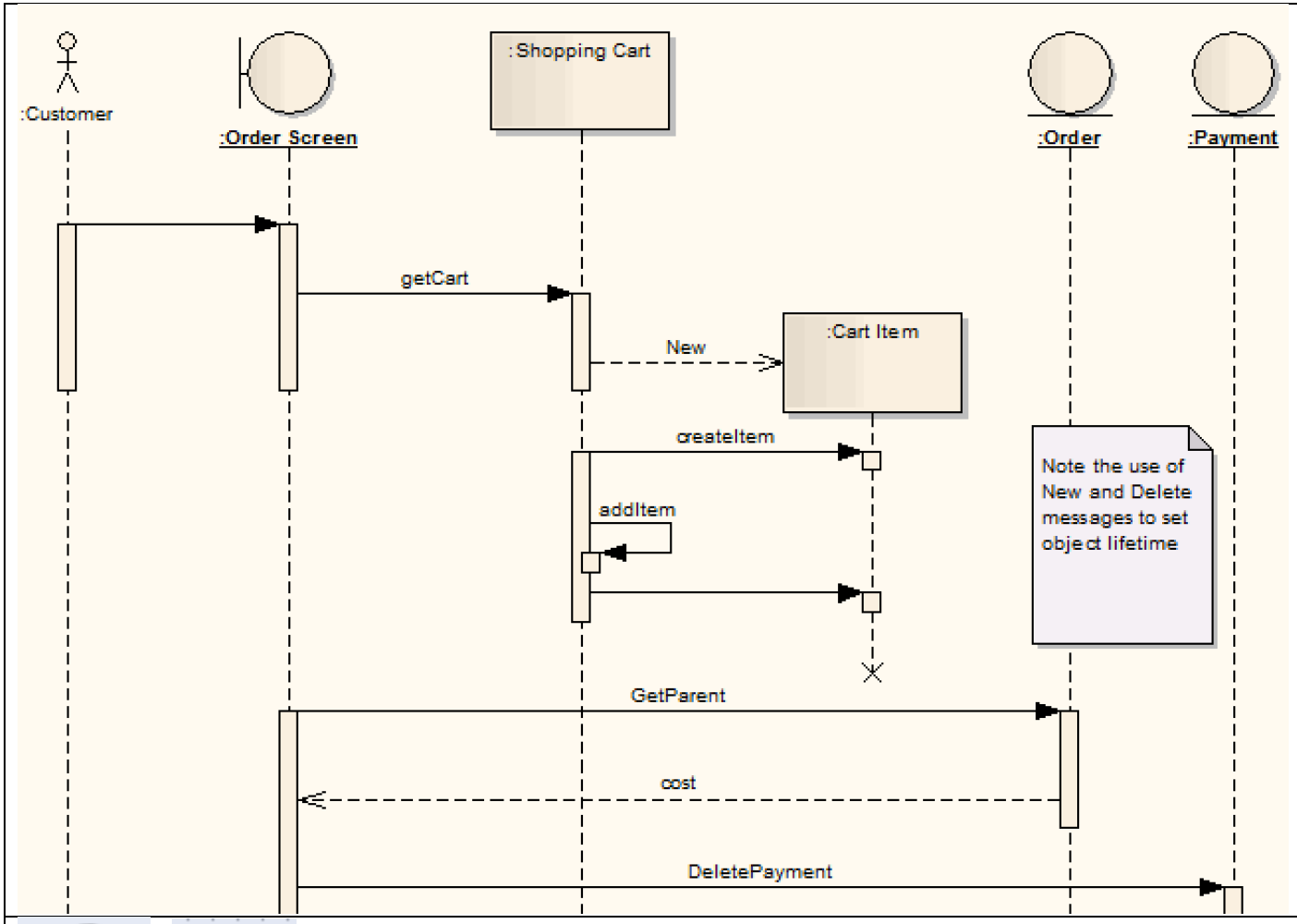
Diagram časovania (timing diagram)



Správanie sa viacerých objektov v nejakom časovom úseku. Využíva sa

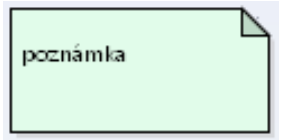
- najmä na: popis HW komponentov, embedded SW komponentov na mikrokontolleroch, ...
- Taktiež pri popise procesov, ktoré sú spúšťané časom.

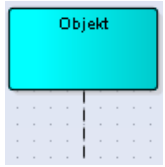
Sekvenčný diagram (sequence diagram)

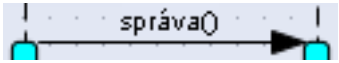


Popisuje správanie sa pomocou sekvenčných krokov v čase.
Použitie: Popis Workflowu, spracovania správ (tok informćie a zodpovednosti), kooperácie elementov.
Dopĺňa prípady použitia – využijú sa aktéri a elementy z prípadov použitia a namodeluje sa sekvencia krokov, ktoré presne vedú k splneniu požadovanej úlohy.

, 

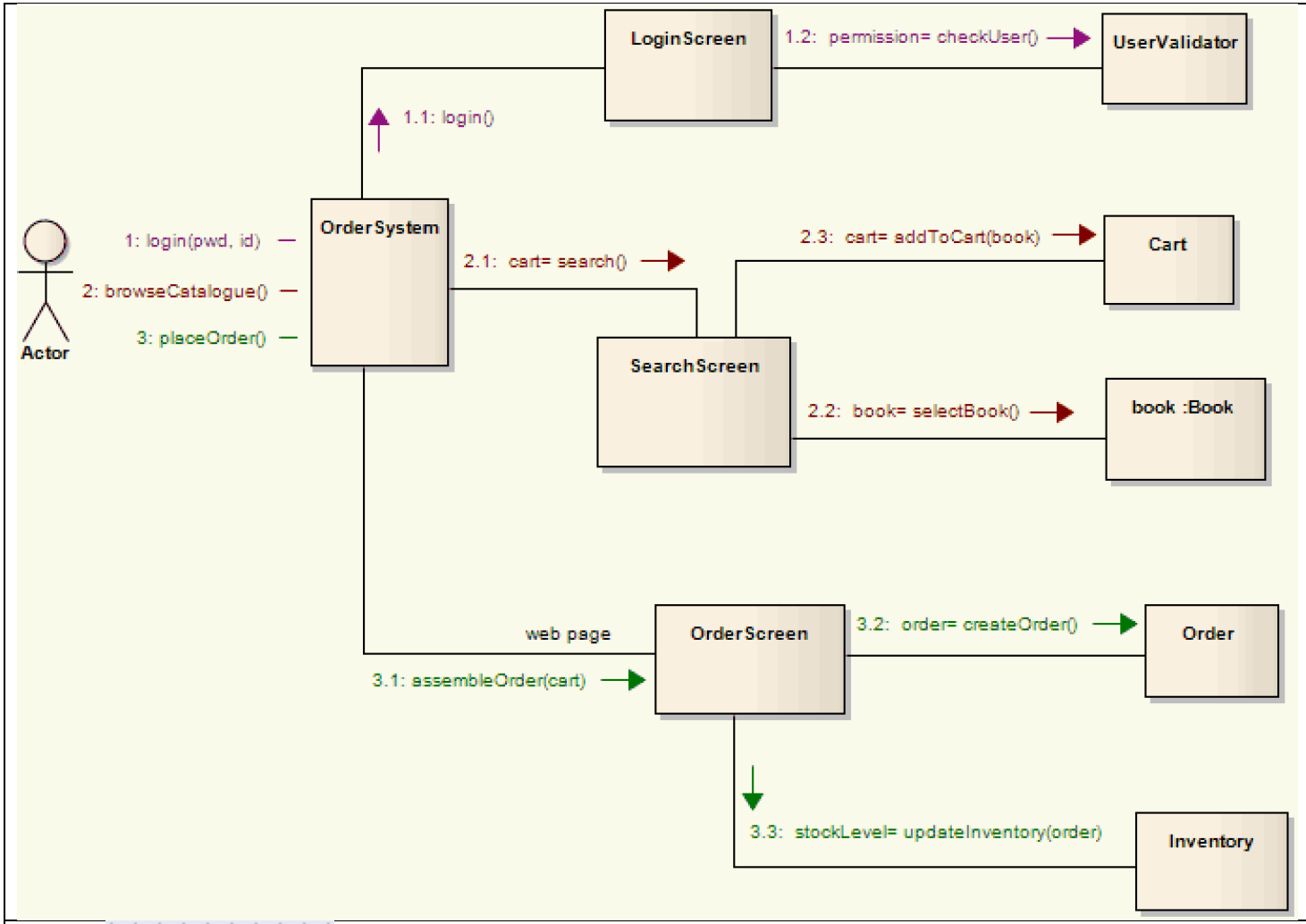
, 

, 

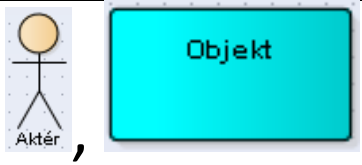
objekt a jeho čiara života, 

správa/volanie

Komunikačný diagram (communication diagram)



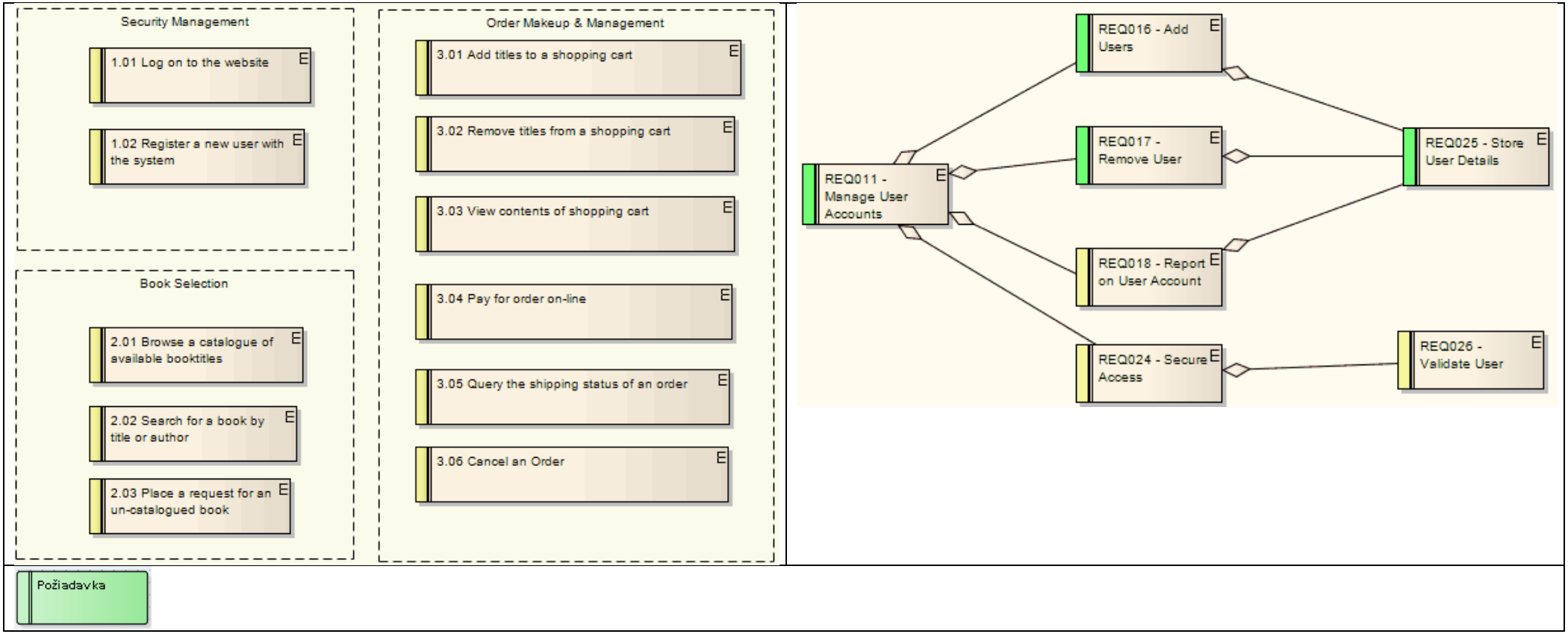
Komunikačný diagram ukazuje interakcia ako sekvenčný diagram. Komunikačný diagram kladie dôraz na vizualizáciu vzťahov a nie na časovú mierku.
Komunikačný diagram potrebuje číselné zoradenie správ a prípadné vnorenia.
Číslovacia schéma môže byť napr.:
1, 1.1, 1.2
2, 2.1, 2.2, 2.3
3, 3.1, 3.2, 3.3



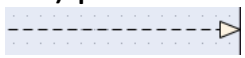
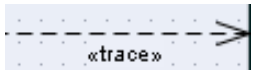
Modelovanie požiadaviek – diagram požiadaviek (requirements)

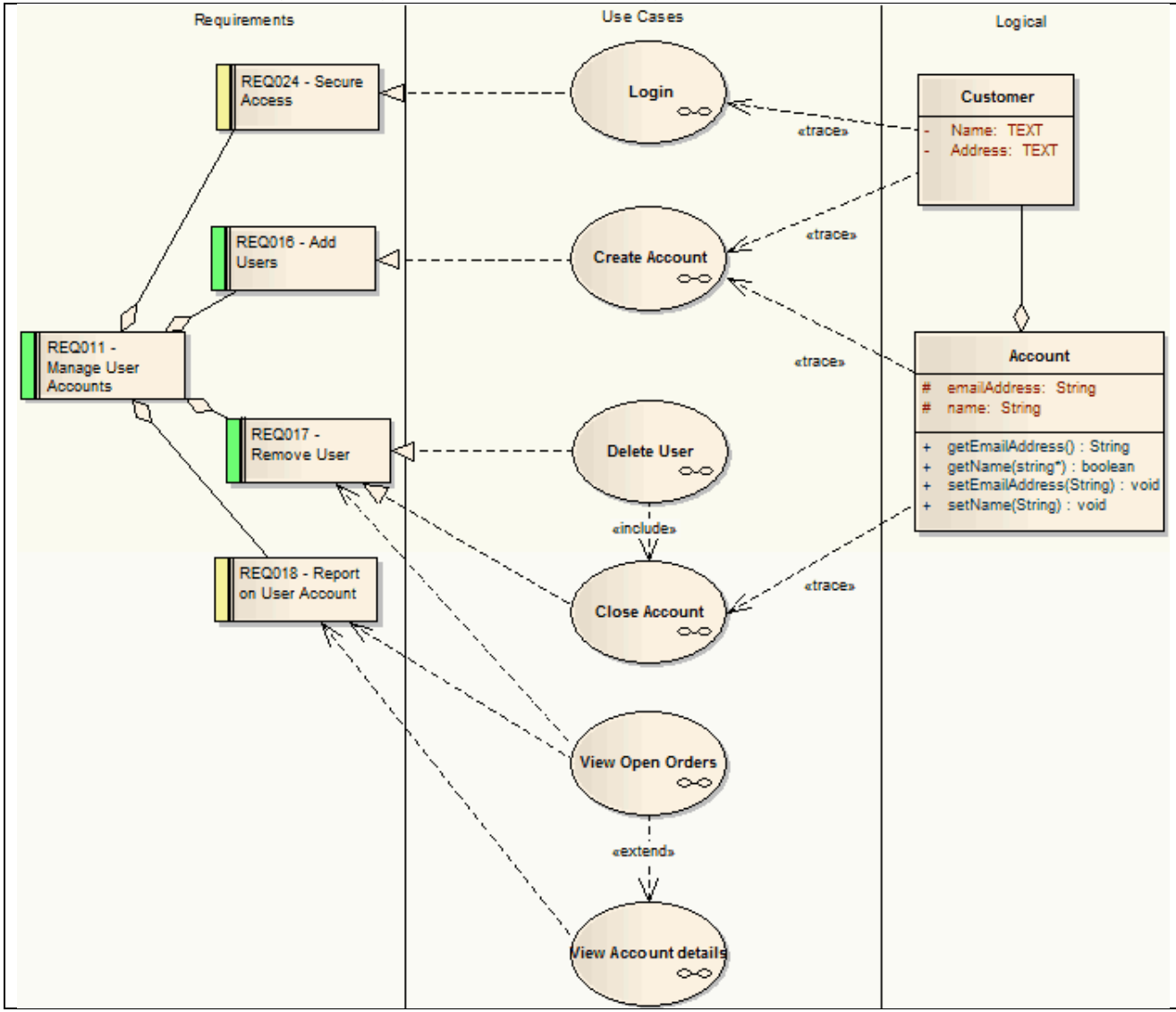
- Patrí medzi Custom diagramy.

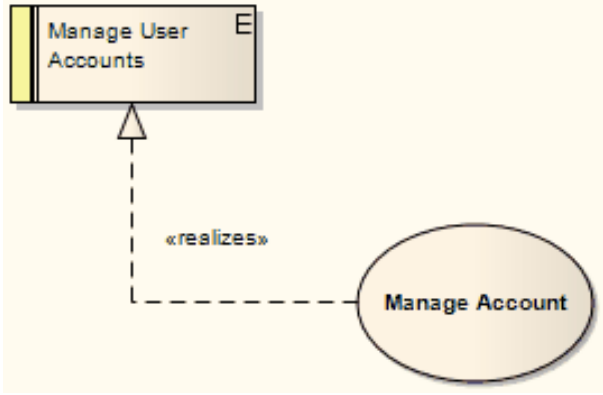
- Každá požiadavka je modelovaná ako „Requirement element“.
- Tieto elementy môžu mať vzťah s inými požiadavkami,
- Tieto elementy môžu mať vzťah s prípadmi použitia a komponentami. Toto sa sleduje v iných, tzv. traceability diagramami.



Traceability diagramy

- Požiadavky sú realizované (implementované) pomocou Prípadov použitia, tried, Interfejsov, Komponentov.
- Označuje sa to UML vzťahom „realizácia“ 
- Ďalšie vzťahy sa dajú zaznamenávať pomocou vzťahu „trace“ 





Doménový model (Domain model)

- Doménový model je koncepčný model, ktorý definuje fyzické a abstraktné objekty ktoré sa v projekte /doméne vyskytujú
- Dokumentuje vzťahy medzi zodpovednosťami koncepčných tried
  - to sú iné triedy ako tie z OOP
  - reprezentujú koncepciu skupiny vecí
- Definuje pojmy, ktoré sa v projekte vyskytujú
- Doménový model ukazuje:
  - fyzické a organizačné jednotky
  - vzťah medzi týmito jednotkami
  - multiplicitu týchto vzťahov
- Modeluje sa pomocou
  - diagramu tried
  - E-R diagramom
- Varianta je **Biznis doménový model**
  - Poskytuje **biznis slovník** – pojmy a fakty na základe ktorých môžu byť definované biznis pravidlá