2. Modelovací prostředky, UML, diagramy UML, jazyk OCL. CASE nástroje. Požadavky a jejich modelování. Trasovatelnost požadavků. (A7B36SIN)

Modelovací prostředky

Úvod

Modelovací jazyk je umělý jazyk, který lze použít k popsání informací, znalostí nebo systémů strukturou definovanou uceleným souborem pravidel.

Modelovací jazyky dělíme na:

- **Graficke (vizuální)** – vyjadření pomocí diagramů, pojmenované symboly vyjadřující nějakou myšlenku a spojnice, které spojují symboly a reprezentují vztahy mezi nimi.
- **Textové** - používají standardizovaná klíčová slova doprovázená parametry, pomocí kterých vytváří výrazy interpretovatelné počítačem

Přiklad grafických modelovacích jazyků

- BPMN – Bussiness Proccess Modeling Notation
- EEML - Extended Enterprise Modeling Language
- Petriho sítě
- Vývojový diagram
- více na http://en.wikipedia.org/wiki/Modeling_language

UML

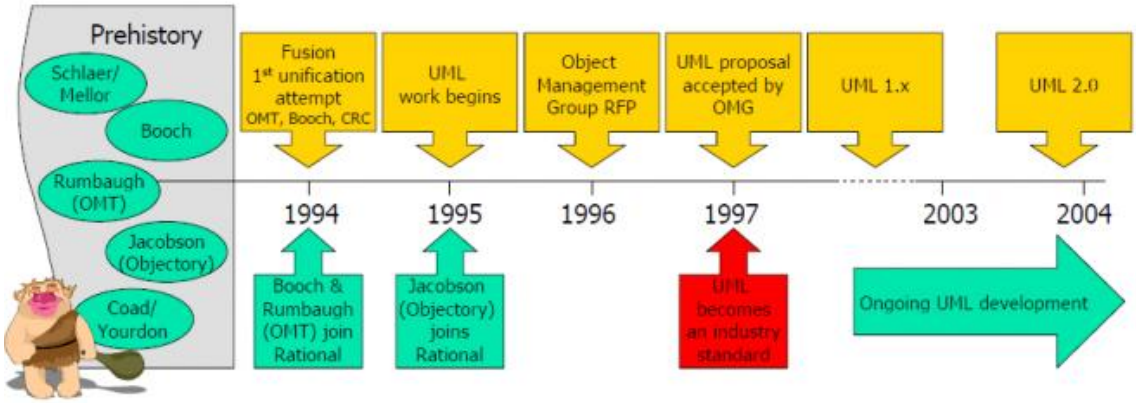
Úvod

UML (Unified modeling language) je grafický modelovací jazyk pro vytváření vizuálních (grafických) modelů. UML je jednotný jazyk (grafický) pro specifikaci, vizualizaci, konstrukci a dokumentaci při OO analýze a návrhu (OOAaD) a pro modelování organizace (business modelling). UML není modelovací metodika – neříká nám nic o tom jak navrhovat SW systémy. Dává nám pouze grafickou syntaxi, pomocí které můžeme vyjadřovat modely. UML je otevřený, standardizovaný, průmyslově užívaný jazyk, snažící se vycházet z principů správného návrhu sw díla (best practices). Byl navržen tak aby byl snadno implementován CASE nástroji.

Původ UML

- do 1994 – nesjednocené metody objektového návrhu, existovalo několik soupeřících grafických modelovacích jazyků a metodik, mezi kterými vynikaly Boochova metoda (autor G. Booch) a Object modeling technique (OMT, autor J. Rumbaugh)
- 1994 – Booch a Rumbaugh přicházejí do Rational Software, aby zde pracovali ma sjednocení jejich metodik – UML, později se k nim připojuje Ivar Jacobson s metodologií OOSE
- 1996 – navržen konsorciem OMG (Object management group)

- 1997 – OMG schvaluje UML jako první otevřený OO modelovací jazyk
- 2005 – schválení vyraznějších změn v revizi UML 2.0



Části UML 2.0 specifikace

- **Superstructure** – definuje zápis a sématiku diagramů a jejich modelovacích elementů
- **Infrastructure** – metamodel stojící v pozadí za UML, specifikovaný pomocí Meta-Object Facility (MOF)
- **Object constrain language (OCL)** - jazyk pro specifikaci vstupních a výstupních podmínek, invariantů v jednotlivých diagramech.
- **Diagram interchange** - popis XML struktur pro výměnu konkrétních modelů mezi jednotlivými modelovacími nástroji.

Základní mechanizmy UML

Specifikace

Každý element může (či měl by ?) být specifikován textem, který popisuje sémantiku tohoto elementu. Tato specifikace upřesňuje, blíže popisuje, udává smysl modelovaného elementu. Popisuje business pravidla elementů (tudíž má největší význam u elementů popisujících problémovou doménu).

Ozdoby (adornments)

Další informace známé o elementu modelu. Každý element může být vyjádřen jednoduchým tvarem, ale je možno k němu přidávat i další informace - ozdoby (např. seznam metod, seznam atributů, název objektu, stereotyp,..).

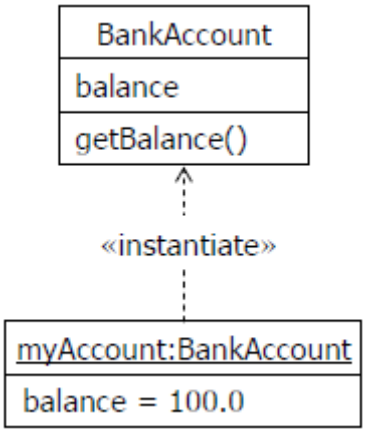
Proč je těchto ozdob u elementu zobrazeno někdy více a někdy méně:

- model vytváříme postupně : zpočátku máme málo informací, které postupně doplňujeme
- tvorbou určitého diagramu sledujeme určité cíle - nechceme v něm zobrazit ty podrobnosti, které jsou v tomto nepodstatné z hlediska toho, co právě chceme zdůraznit (jedno z prvořadých hledisek při tvorbě diagramu je totiž snadná čitelnost)

Podskupiny (common divisions)

Udávají, jak je možno rozdělovat (skupinovat) jednotlivé elementy; první způsob dělení:

- **klasifikátor a instance**: v podstatě se jedná o stejné rozdělení jako v objektově orientovaném programování - klasifikátor odpovídá třídě, zatímco instance odpovídá objektu
- **rozhraní a implementace**: opět je to stejný princip jako v objektově orientovaném programování



Mechanismy rozšiřitelnosti (extensibility mechanisms)

Jazyk UML sám v sobě obsahuje připravené mechanismy umožňující rozšířit jazkyk tak, aby vyhovoval momentálním potřebám. Máme k dispozici tři mechanismy rozšiřitelnosti:

- **omezení (constraints)** : jde o text ve složených závorkách {}. Podmínka či pravidlo v tomto textu musí být vždy splněna
- **stereotypy (stereotypes)** : s jejich pomocí lze z existujího elementu vytvořit nový. Vytvoříme ho tak, že název stereotypu vložíme do dvojitých ostrých závorek : “«novy_stereotyp»“. Stereotyp může mít rovněž přiřazen nový symbol - časté využití bývá např. v diagramu nasazení pro vytvoření symbolů tiskáren, serverů, notebooků apod. Některé stereotypy jsou již součástí jazyka UML a ještě se s nimi v tomto kurzu setkáme.
- **označené hodnoty (tagged values)** : umožňuje přidávat nové vlastnosti k elementům modelu. Zavedeme ji přidáním názvu s připojenou vlastní hodnotou ve složených závorkách, např. {autor=pavus, verze=0.1}

Profily v UML definují množinu stereotypů, značek a omezení, díky nimž lze jazyk UML přizpůsobit konkrétnímu účelu. (např. můžeme mít nějaký profil modelování aplikací pro platformu J2EE, jiný profil pro .NET, apod.)

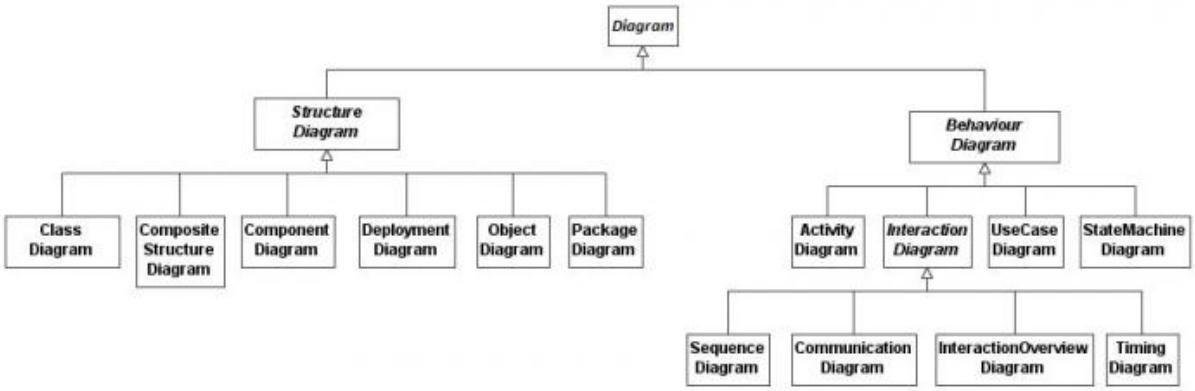
Stavební bloky UML

UML diagramy

UML 2 popisuje 13 oficiálních typů diagramů uvedené v obrázku.

Mezi ně patří

- diagram tříd (class diagram)
- diagram složených struktur (composite structure diagram)
- diagram objektů (object diagram)
- diagram komponent (component diagram)
- diagram nasazení (deployment diagram)
- diagram balíčků (package diagram)
- diagram aktivit (activity diagram)
- diagram případů užití (use case diagram)
- sekvenční diagram (sequence diagram)
- diagram komunikace (communication diagram)
- diagram přehledu interakcí (interaction overview diagram)
- diagram časování (timing diagram)



Vztahy – Relationships

vztah	UML	význam
závislost (dependency)		zdrojový prvek závisí na cílovém a může být ovlivněn změnou jeho stavu
asociace (association)		Popis množiny spojení mezi objekty
agregace (aggregation)		cílový element je součástí zdrojového
kompozice (composition)		silnější forma agregace
generalizace (generalization)		zdrojový prvek je specializace obecnějšího cílového prvku a může jím být zastoupen-nahrazen
realizace (realization)		cílový element realizuje chování, které zdrojový element specifikuje

Diagramy UML

Vhledem k rozsahu této látky uvádím pouze základní informace o některých diagramech. Navíc podrobnější rozpracování diagramů by mělo být součástí následujících státnicových otázek.

Diagram tříd (class diagram)

Diagram tříd popisuje typy objektů v systému a různé druhy statických vztahů, které mezi nimi existují. Diagramy tříd ukazují i vlastnosti (atributy) a operace třídy, a také omezení týkající se způsobu, jakým jsou objekty spojovány.

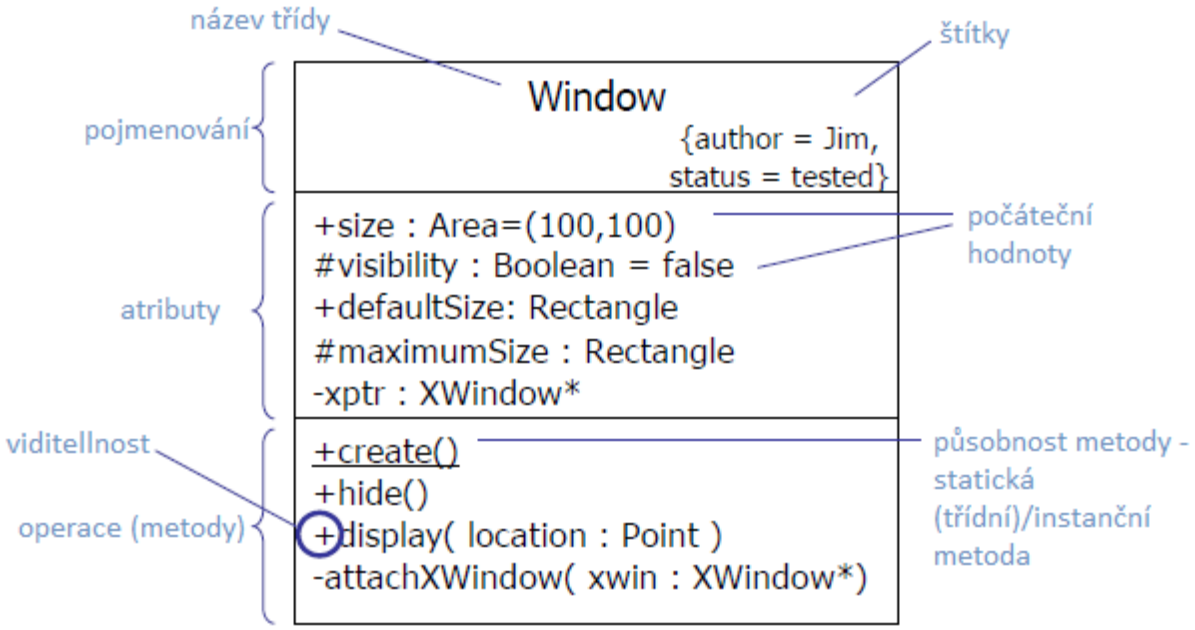
Tři pohledy na systém při použití diagramu tříd:

- **Konceptuální** – koncepty aplikační domény, bez vztahu k implementaci, jazykově nezávislá.
- **Specifikační** – pohled na program, specifikace rozhraní bez specifikace implementace.
- **Implementační** – je vidět implementace - hranice nejsou ostré, UML podporuje všechny tři pohledy

Tipy:

- nesnažit se použít veškerou dostupnou notaci
- přizpůsobit pohled etapě projektu:
    - analýza: konceptuální pohled
    - návrh: specifikační + implementační
- koncentrovat se na klíčové oblasti, raději méně aktuálních diagramů

- nezabřednout brzy do implementačních detailů



Násobnost vlastností

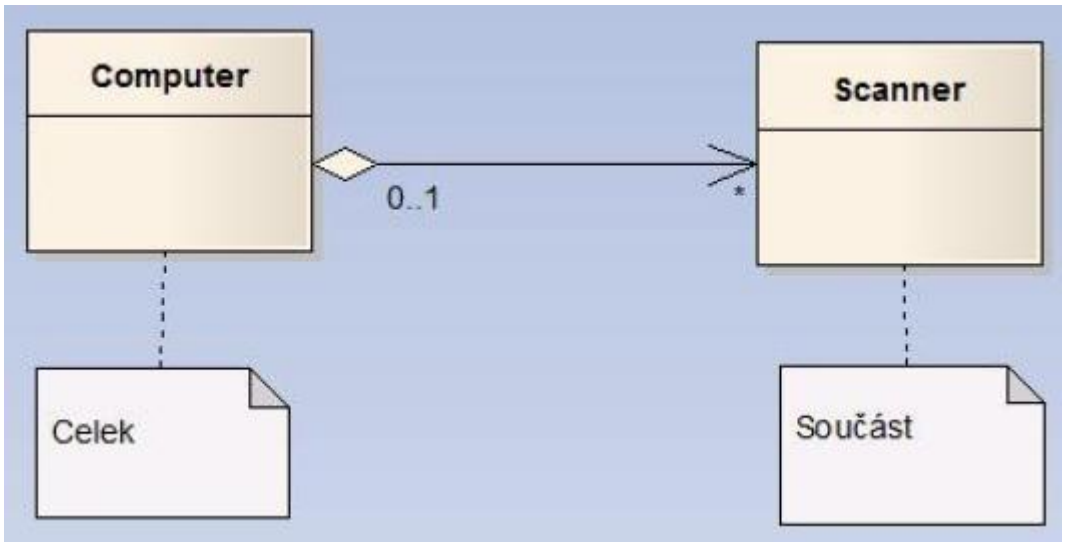
Násobnost (multiplicity) vlastnosti je údaj o tom, kolik objektů může být do vlastnosti umístěno. Nejčastěji se setkáte s těmito násobnostmi:

- 1 (právě jeden)
- 0..1 (0 až jedna)
- * (0 až x, horní hranice není omezena)

Návrhové relace

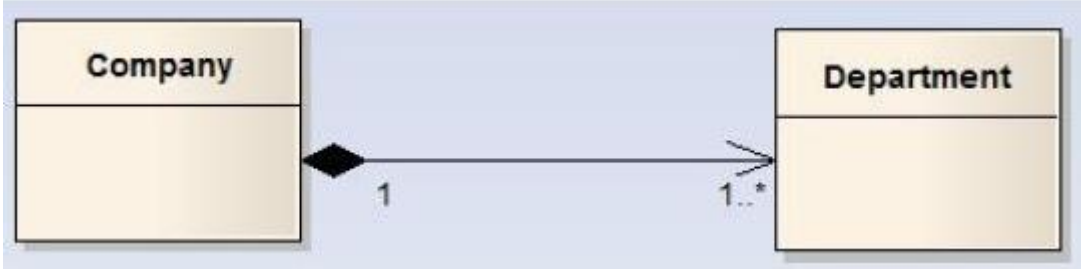
Agregace

- Volná vazba mezi objekty
- Typ celek/součást (celek řídí chod relace, součást poskytuje služby)
- Agregované objekty
    - mohou být sdílené
    - nezanikají se zánikem celku
- Celek někdy existuje nezávisle na součástech, někdy je na nich závislý
- Součást může být sdílena více celky



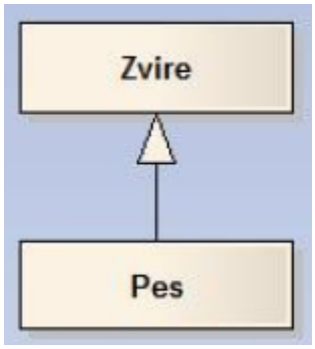
Kompozice

- Silnější forma agregace
- Součásti nemohou existovat mimo celek
- Součást patří právě jednomu celku
- Celek odpovídá za použití svých součástí
- Za předpokladu, že odpovědnost za součásti přejde na jiný objekt, může celek součásti uvolnit.
- Je-li zničen celek, musí zničit své součásti (nebo přenést odpovědnost na jiný objekt)



Generalizace

- Jako zobecnění (generalizace) se označuje relace mezi obecnějším a přesněji definovaným prvkem.
- Na následujícím obrázku je třída Zvíře obecnější než třída Pes.



Dědičnost tříd

- Potomci (podtřídy) přebírají charakteristiku svých předchůdců
- Potomci dědí
    - Atributy
    - Operace
    - Relace
    - Omezení
- Potomci mohou ke zděděnému fondu přidat novou charakteristiku

Realizace

- Fakt, že třída implementuje (realizuje) nějaké rozhraní znázorňujeme čárkovanou čárou, která směřuje od třídy k rozhraní.
- Na následujícím obrázku třída Timestamp implementuje rozhraní TimeComparable.
- Rozhraní se používá k určení společných protokolů pro třídy, mezi nimiž by za normálních okolností neměly být dědičné vazby.

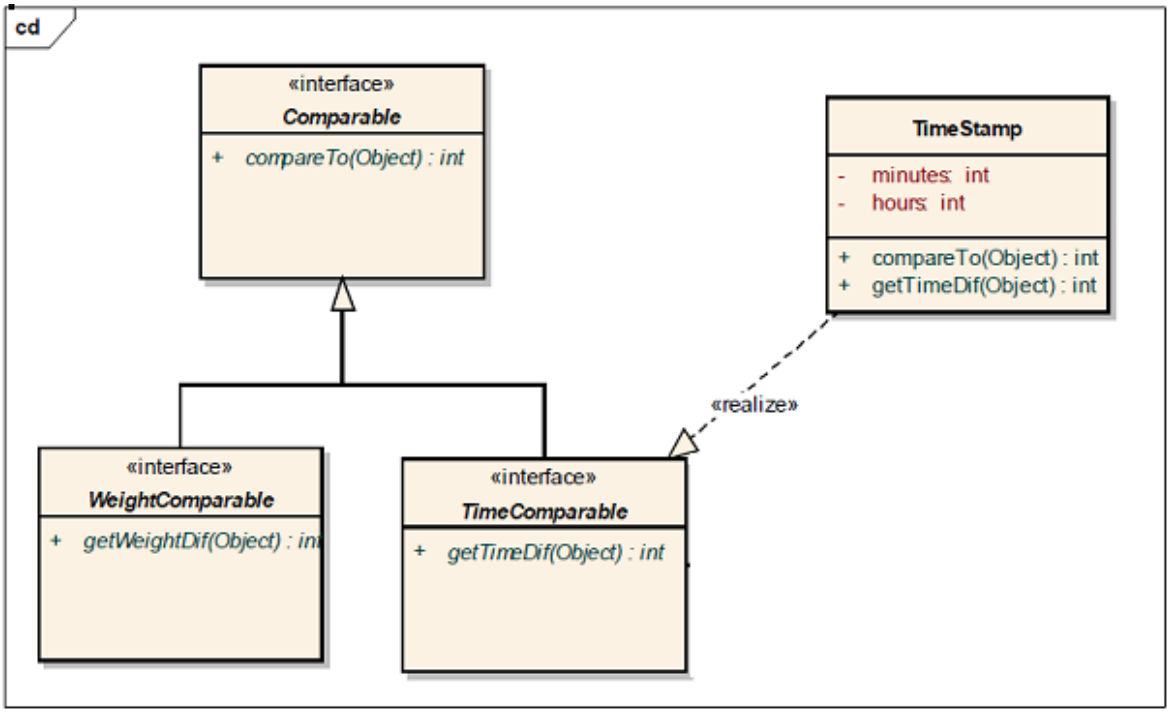
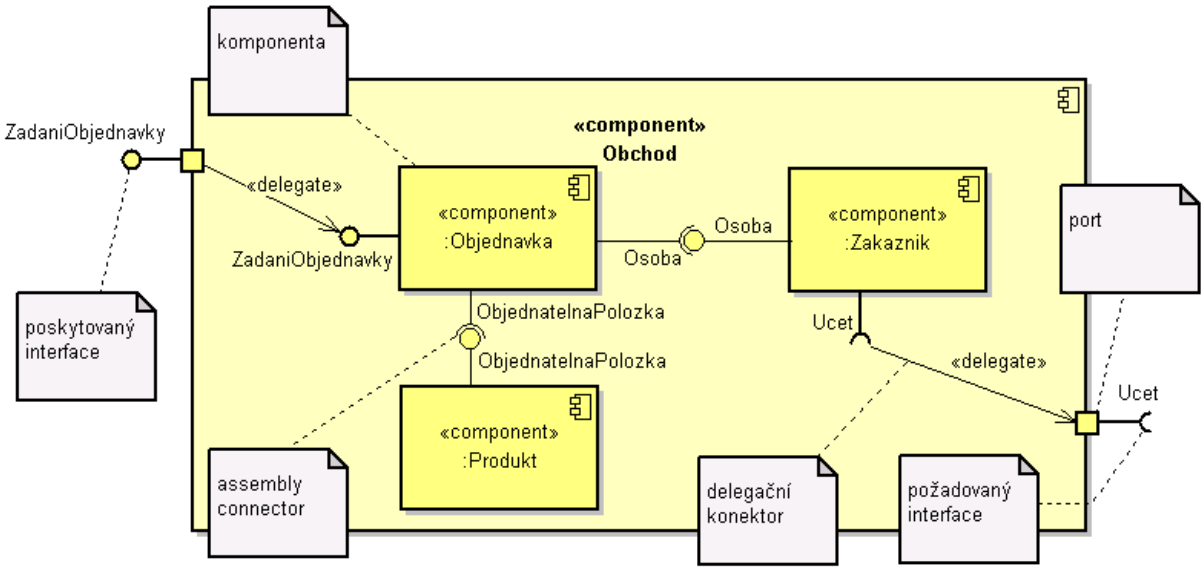


Diagram komponent (Component diagram)

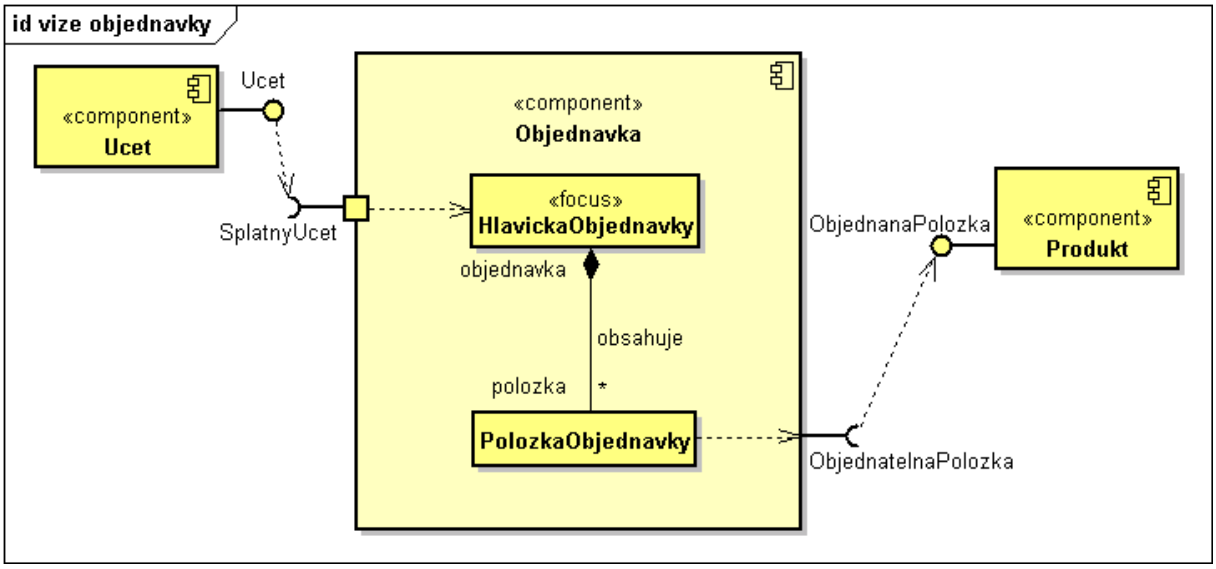
Popisuje jak jsou jednotlivé komponenty (části) systému propojeny. Komponenta je fyzická nahraditelná část systému, která obaluje implementaci a poskytuje realizaci množiny specifikovaných rozhraní.

Diagram komponent znázorňuje komponenty použité v systému, tj. logické komponenty (např. business k., procesní k.) či fyzické komponenty (např. EJB k., CORBA k., .NET k. atd.). Diagram komponent nám může pomoci zejména tam, kde používáme vývoj založený na komponentách a kde struktura systému je založena na komponetách.

Komponenta je modulární část systému, která zapouzdřuje svůj obsah (tj. zapouzdřuje stav a chování i více klasifikátorů) a jejíž projev je nahraditelný (tj. komponenty, poskytující ekvivalentní funkcionalitu založenou na kompatibilitě jejich interfacem mohou být libovolně zaměňovány, a to buď už v čase tvorby designu, nebo až za běhu cílového systému). Komponenta je specializací strukturované třídy. Protože je komponenta specializací třídy, může mít své atributy a operace a může mít asociace a generalizace-specializace (více viz diagram tříd (class diagram)).



Na prvním obrázku je diagram komponent s jednou strukturovanou komponentou **Obchod** a jejími třemi vnitřními komponentami **:Objednavka**, **:Produkt** a **:Zakaznik**.



V druhém obrázku je další diagram komponent, kde vidíme komponentu **Objednavka** a její vnitřní klasifikátory **HlavickaObjednavky** a **PolozkaObjednavky**, které realizují chování této komponenty.

V diagramech můžeme vidět **komponenty**. Komponenta je znázorněna jako symbol klasifikátoru (pravoúhelník), s klíčovým slovem «component». Volitelně má v pravém horním rohu zobrazenu speciální malou ikonu komponenty. (PS.: ve verzi UML 1.x vypadal symbol komponenty právě tak, jako ona speciální ikona, přitom starý symbol komponenty je přípustný i ve verzi UML 2.0).

Druhy komponent:

- **basic component** : pro ni platí vše, co je v této kapitole o komponentě řečeno (pokud není výslovně uvedeno, že se informace týká jen komponenty typu packaging)
- **packaging component** : rozšiřuje komponentu typu basic o možnosti skupinování; takovéto komponenty se týkají aspekty spojené s problematikou namespaces (komponenta může dědit a importovat své členy, jako např. packages, use cases, komponenty, artefakty, třídy, ….)

Rozhraní:

- **poskytovaný** (provided) **interface** je implementován přímo danou komponentou (popř. klasifikátorem, který realizuje chování komponenty), nebo je to vlastně typ poskytovaného portu (zde poskytovaný interface ZadaniObjednavky je připojen na nepojmenovaný port).
- **požadovaný** (required) **interface** je typem požadovaného portu, nebo je určený závislostí usage přímo z dané komponenty (popř. klasifikátoru, který realizuje chování komponenty), (zde požadovný interface Ucet komponenty Obchod (1. diagram) je připojen na nepojmenovaný port, požadovaný interface ObjednatelnaPolozkaUcet komponenty Objednavka (2. diagram) je připojen pomocí závislosti na interní třídu PlozkaObjednavky).
- **port** nám umožňuje organizování interfacesů do skupin : můžeme vytvářet pojmenované sady interfaců
- **delegační konektor** (delegation connector) propojuje externí rozhraní komponenty (tak, jak je specifikováno portem) s vnitřní částí komponenty (tj. s vnitřní komponentou či třídou anebo s portem vnitřní komponenty).

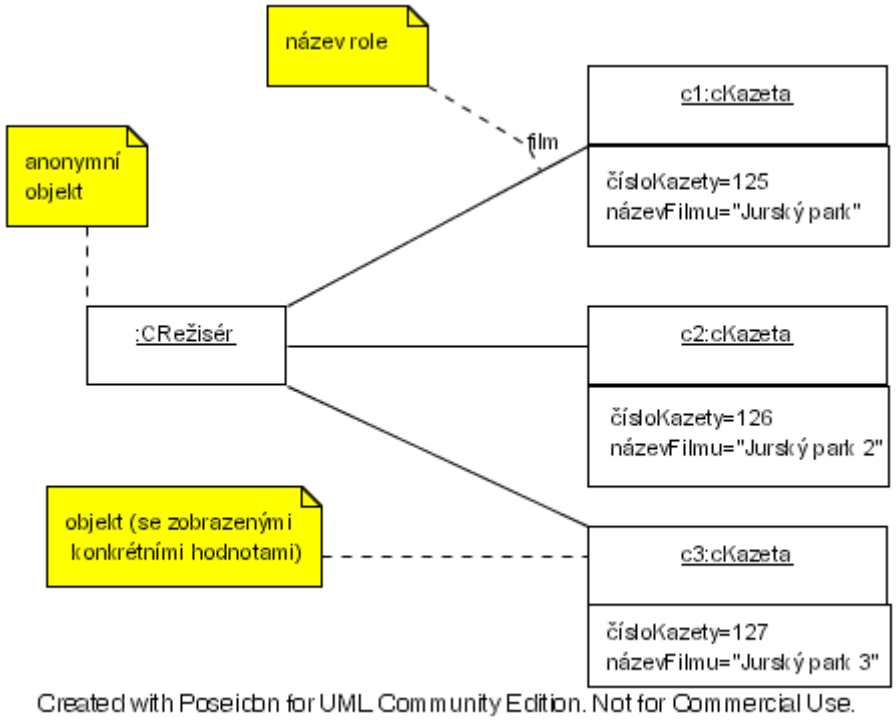
- **montážní konektor** (assembly connector) spojuje požadovaný a poskytovaný interface; strana, která poskytuje interface, musí být schopna poskytnout minimálně všechny služby, které může požadovat strana s požadovaným intefacem

příklady komponent:

- zdrojový soubor
- ActiveX prvek
- Enterprise JavaBean objekt
- Java servlet
- JSP stránka

Diagram objektů (Object diagram)

Objektový diagram zobrazuje objekty a jejich spoje (links) v jednom okamžiku, tj. je to snapshot běžícího systému (či jeho části). Objekty jsou instancemi tříd. Objekty jsou propojeny linky. Objektový diagram vypadá podobně, jako diagram tříd. Název konkrétního objektu se zapisuje před název třídy a odděluje se dvojtečkou.



Na obrázku je příklad objektového diagramu odvozeného z class diagramu naší půjčovny videokazet. :

- objekt je znázorněn podobně jako třída - obdélníkem, ale název objektu je vždy podtržený
- zápis **c1:CKazeta** značí objekt **c1**, který je instancí třídy **CKazeta** (obdobně je tomu s objekty c2 a c3)
- zápis **:CRežisér** značí tzv. **anonymní objekt** : nevíme, jak se objekt jmenuje, ale víme, ze které třídy byl objekt odvozen
- třetí možnost zápisu by byla **c4**, tj. objekt bez vyznačení třídy : nevíme, ze které třídy je objekt odvozen (nebo to autor věděl, ale chtěl znázornit, že objekt c4 mohl vzniknout z různých tříd)
- **spojení** (link) mezi objekty vzniklo jako instance asociace
- objekt **:CRežisér** je znázorněn v minimálním tvaru (jen jeho znázev), objekt **c1:CKazeta** je znázorněn s dalším * oddílem (kompartment) ukazujícím aktuální hodnoty atributů
- u objektu **c1** je vyznačena role - tj. ve vztahu anonymního objektu **:CRežisér** versus **c1:CKazeta** hraje tato kazeta roli filmu (který byl režírován spojeným objektem - režisérem)

Diagram případů užití (use case diagram)

Use case diagram zobrazuje chování systému (nebo jeho části) z hlediska uživatele. Vychází z funkčních požadavků na systém.

postup při modelování:

- najít hranice systému (za kterou funkčnost je systém zodpovědný a za kterou již není)
- najít aktéry
- najít případy užití – specifikace + scénáře

Aktéři

Aktér specifikuje roli, kterou si může vnější entita (osoba, HW zařízení, jiný informační systém) přisvojit. Akteři definují hranice systému Při identifikaci aktérů zjišťujeme:

- Kdo nebo co používá systém?
- V jakých rolích se přitom nachází?
- Kdo instaluje/nasazuje systém?
- Kdo systém udržuje?
- Které jiné systémy tento systém využívají?
- Spouští se nějaká událost v závislosti na čase? (např. zálohování apod.)

Případy užití

- Případ užití je nějaká činnost, kterou aktéři se systémem provádějí. Případy užití jsou vždy iniciovány aktérem.
- Při pojmenovávání UC používáme slovesnou vazbu (např. Zrušit objednávku, v AJ se pro pojmenovávání často užívá CamelCase - např. CancelOrder).

Identifikace případů užití:

- Které funkce budou jednotliví aktéři vyžadovat od systému?
- Ukládá nebo vrací systém nějaké informace? Pokud ano, který aktér tuto akci vyvolává.
- Co se stane pokud systém změní svůj stav (spuštěný/vypnutý)?
- Existují nějaké vnější události, které působí na systém?
- Vytváří systém nějaké zprávy (reporty)?
- Komunikuje systém s nějakým vnějším systémem?

Scénáře případů užití

POZOR! Scénáře případů užití jsou nedílnou součástní modelu. Samotné diagramy (bez scénářů) nemají potřebnou vypovídací hodnotu.

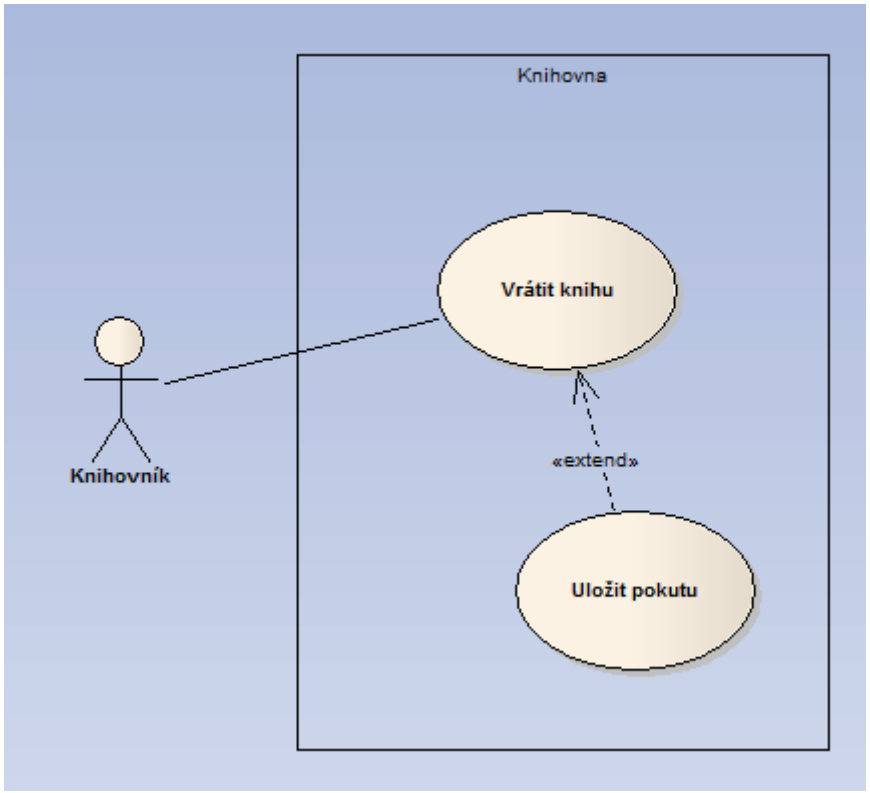
Struktura scénáře:

- název (např. Půjčit knihu)
- identifikátor - jednoznačný (např. UC2.1)
- stručný popis (např. Označí vybranou knihu jako vypůjčenou a zadá příkaz k výdeji)
- seznam primárních aktérů (např. Zákazník)
- seznam sekundárních aktérů (např. Pokladní)
- Vstupní podmínky (např. Kniha je skladem)

- Hlavní scénář, popisuje jednotlivé interakce mezi systémem a aktérem
- Alternativní scénáře
- Výstupní podmínky (např. Kniha byla zapůjčena, byl odeslán příkaz k výdeji)

Use case: PaySalesTax
ID: 1
Brief description: Pay Sales Tax to the Tax Authority at the end of the business quarter.
Primary actors: Time
Secondary actors: TaxAuthority
Preconditions: 1. It is the end of the business quarter.
Main flow: 1. The use case starts when it is the end of the business quarter. 2. The system determines the amount of Sales Tax owed to the Tax Authority. 3. The system sends an electronic payment to the Tax Authority.
Postconditions: 1. The Tax Authority receives the correct amount of Sales Tax.
Alternative flows: None.

Relace include vs. relace extend



include

- vyčleňuje společné kroky několika případů užití do samostatného případu užití

extend

- umožňuje rozšířit případ užití o nové chování

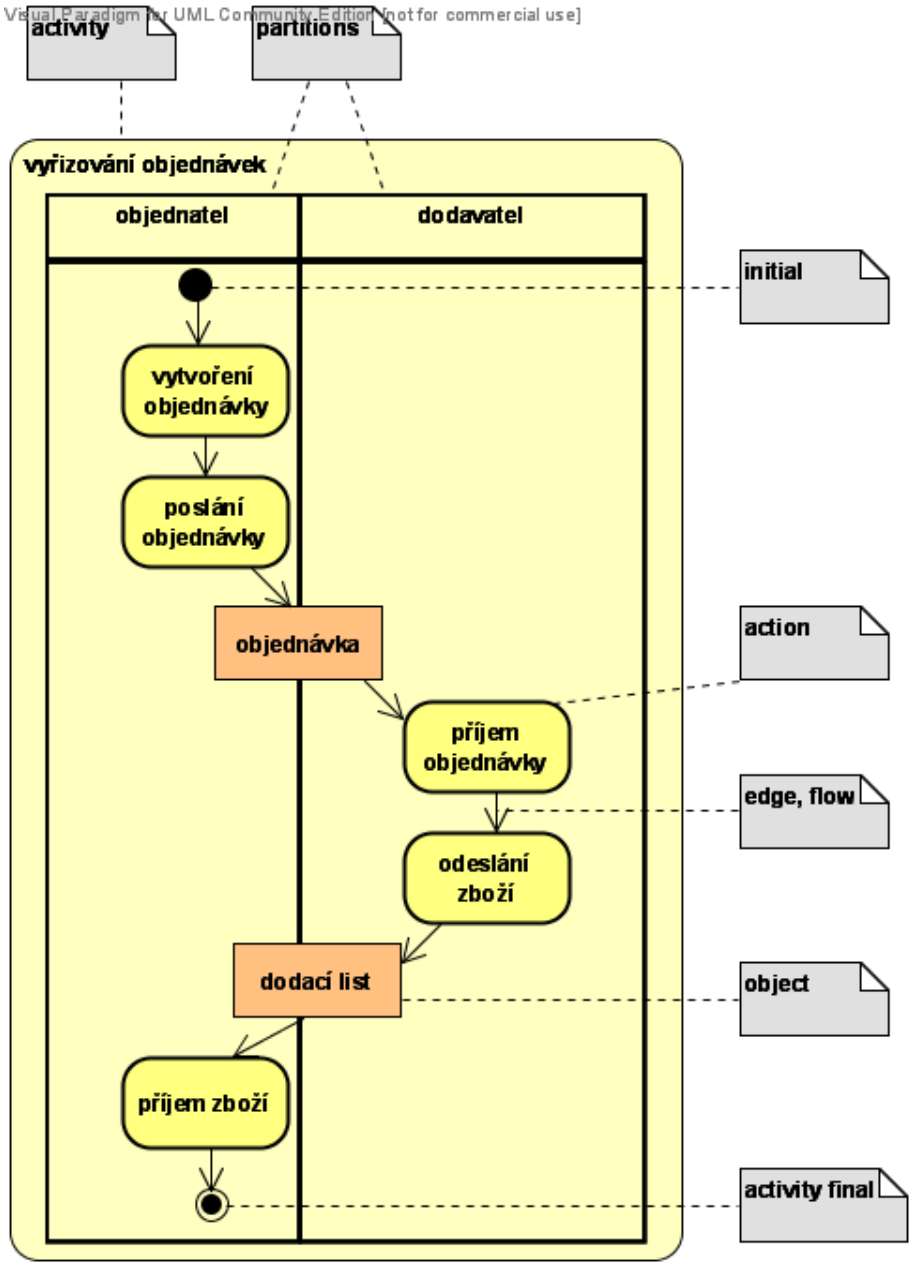
Diagram aktivit (Activity diagram)

Diagram aktivit se používá pro popis dynamických aspektů systému.

Jde o jakýsi flowchart - znázorňuje tok řízení z aktivity do aktivity. Používá se také k modelování obchodních (business) procesů a workflow.

Diagram aktivit se soustřeďuje spíše na proces výpočtu než na objekty účastnící se výpočtu (i když i objekty mohou být znázorněny jako prvek aktivity).

Diagramy stavů a diagramy aktivit jsou si podobné (oba ukazují sekvenci stavů, které nastávají v čase, a ukazují podmínky způsobující přechody mezi stavy). Rozdíl mezi těmito diagramy je však v tom, že d. stavů se soustřeďuje na stavy objektu (tj. objektu provádějícího výpočet či objektu, se kterým je výpočet prováděn), kdežto d. aktivit se zaměřuje na stav samotného výpočtu (stav procesu, algoritmu, ….), kde může být účastno i více objektů, a kde jsou znázorněny řídící a informační toky mezi prvky diagramu.



Na obrázku je příklad jednoduchého diagramu aktivit:

- **aktivita** (activity) **vyřizování objednávek** je ta, která je zde modelována (pro znázornění, jak vlastně toto vyřizování objednávek probíhá)
- **oddíly** (partitions, swimlanes): plavební dráhy - rozdělení diagramů na více částí pro znázornění odpovědností za různé části aktivity; zde jde o rozdělění aktivity na dvě různé firmy
- **akce** (action) : nejprimitivnější, nejnižší prvek výpočtu. Nemůže být dále dekomponován.
- **hrana, tok** (edge, flow) zobrazuje přechod z jedné aktivity do další (v příkladu je přechod vždy automatický, dojde k němu po ukončení předcházející akce)
- **objekt** (object) může vstupovat do aktivit, může být jejich výsledkem
- **počáteční, finální uzel aktivity** (initial, activity final) : zvláštní uzly pro označení počátku a konce aktivity

Aktivity mohou být spouštěny akcemi a nebo jako součásti jiných chování, např. přechody v diagramu stavového stroje, metodou, případem použití, jinou aktivitou… Aktivita je zobrazena v diagramu aktivit.

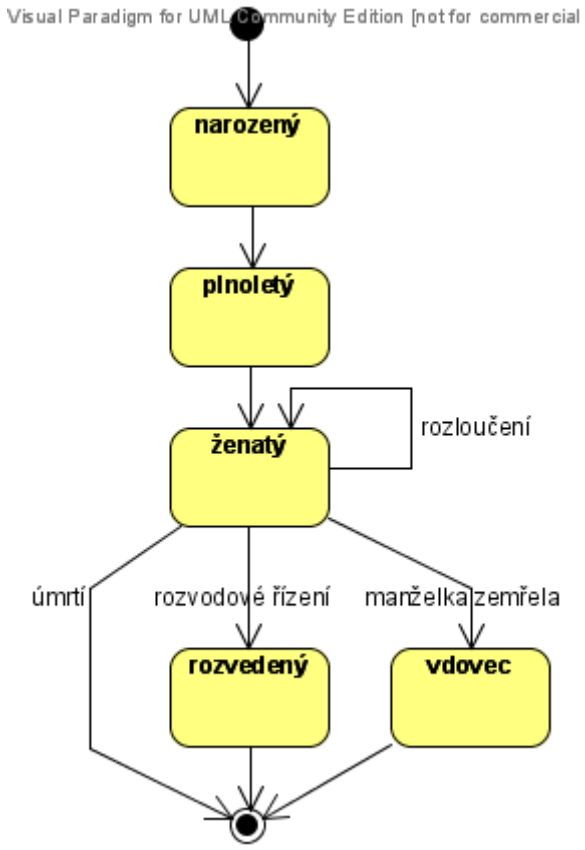
Aktivita je specifikace chování - popisuje sekvenční a souběžné kroky výpočetní procedury. Aktivita je modelována jako graf uzlů aktivity (activity nodes) propojených řídícími a datovými toky (control flow, data flow). PS.: Uzel aktivity je něco naprosto jiného než uzel v deployment modelu! Paradoxně na tuto zmatečnost je poukazováno přímo v UML reference manuálu.

Uzly aktivity jsou:

- vložené aktivity
- akce: modelují účinky na systém
- lokace dat
- řídící konstrukce (control construct)

Stavový diagram (state machine diagram)

Diagram obsahuje stavový stroj (state machine). Stavový stroj vyjadřuje stavy určitého objektu a přechody mezi těmito stavy.



Na obrázku je příklad jednoduchého diagramu stavového stroje:

- stav (state) : situace, kdy modelovaný objekt splňuje nějakou podmínku, provádí nějakou operaci nebo čeká na událost - např. ve stavu narozený splňuje objekt podmínku, že mu není dosud 18 roků a čeká, až dosáhne plnoletosti
- přechod (transition)
- spojení mezi dvěma stavy; objekt přejde z prvního stavu do druhého stavu (za splnění určitých podmínek)
- rozloučení ukazuje přechod do téhož stavu (něco význačného se stane, ale stav se nezmění)
- počáteční, finální stav: zvláštní pseudostavy pro počátek a konec automatu

Stavový stroj je graf **stavů a přechodů** (mezi těmito stavy), který popisuje reakce objektu (přesněji : reakce instance klasifikátoru) na obdržení události. Stavový stroj může být připojený ke klasifikátoru (např. use case, class), nebo ke collaboration či k metodě.

Element, ke kterému je připojen stavový stroj, se nazývá **vlastník (owner) stavového stroje**.

Celý stavový stroj je vlastně **složený stav** (composite state), který je rekurzivně dekomponován na substavy. Nejspodnější (listové) stavy již nemají **substavy**.

Stavový stroj může mít reference na jiný stavový stroj použitím stavového substroje (submachine state). Ve stavovém stroji může být v jeden okamžik aktivních více stavů.

Pro vysvětlování podrobností o stavech musíme vědět, **co je to přechod**:

přechod (transition) je relace ve stavovém stroji mezi dvěma stavy, kde objekt v prvním stavu přejde do druhého stavu tehdy, když nastane specifikovaná událost (event) a jsou splněny specifikované podmínky (guards), přičemž se provede specifikovaný efekt (effect - action nebo activity). Přechod je nepřerušitelný.

Říká se, že přechod je odpálen (transition is fire). Přechod může mít jeden nebo více zdrojových stavů a jeden nebo více cílových stavů.

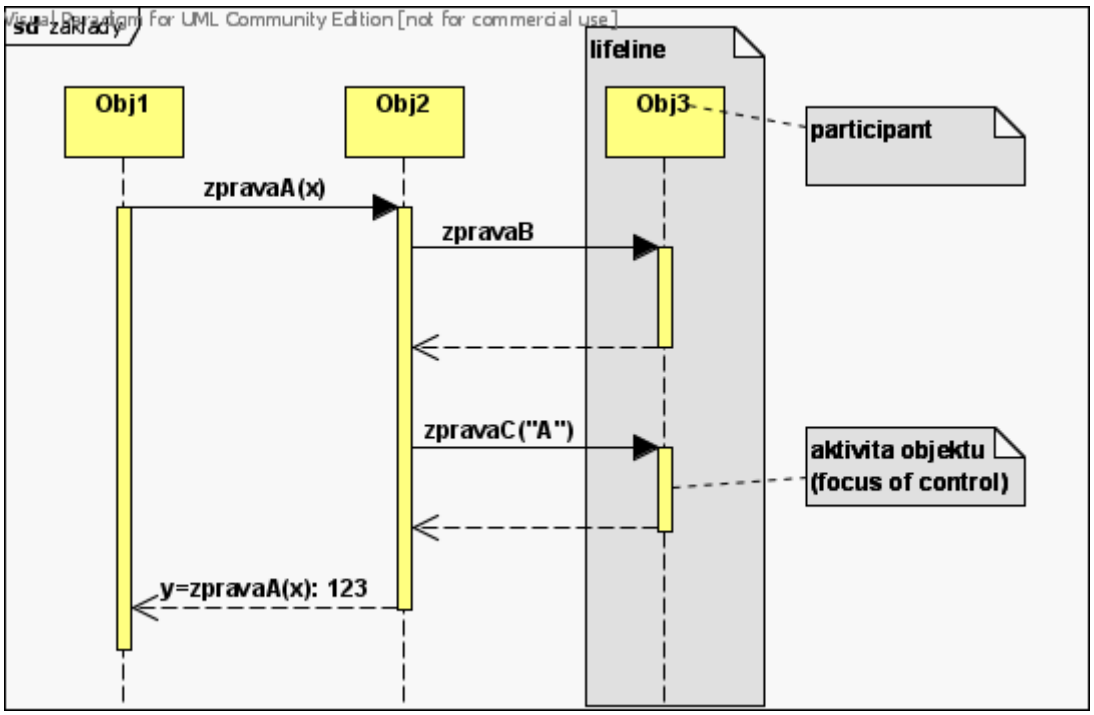
Co je to **stav**

stav (state) : situace, kdy modelovaný objekt splňuje nějakou podmínku, provádí nějakou operaci nebo čeká na událost. - např. ve stavu narozený splňuje objekt podmínku, že mu není dosud 18 roků a čeká, až dosáhne plnoletosti. Stavy jsou obsaženy ve stavovém stroji, který popisuje jak se vyvíjí objekt v čase dle svých reakcí na události.

Sekvenční diagram (Sequence diagram)

Sekvenční diagram typicky zachycuje chování jednoho scénáře. Ukazuje několik vzorových objektů a zpráv, které jsou předávány mezi těmito objekty v rámci daného případu užití.

Příklad takového diagramu lze vidět na následujícím obrázku.



Jedná se o jednoduchý sekvenční diagram. Časová osa je svislá (čas běží shora dolů), na vodorovné ose jsou rozmístěny objekty:

- lifeline (čára života) reprezentuje participanta v interakci:
  - **Participant** je většinou konkrétní objekt.
  - Lifeline také ukazuje, kdy participant žije - v tomto diagramu všechny objekty existovaly už před posláním první zprávy, a dále existují po dokončení sekvenčního diagramu (resp. nevíme, kdy který objekt zanikne nebo nás to v tuto chvíli nezajímá)
  - **execution specification, focus of control** (aktivita objektu) vyjádřená zdvojeným úsekem (obdélníkem) na lifeline : ukazuje periodu, kdy je který objekt aktivní, neboli kdy provádí nějakou činnost, nějak se chová, včetně „podřízeného“ chování dalších objektů (tj. u aktivního objektu vyznačuje celý jeho život, u pasivního objektu časový interval, kdy je prováděna operace objektu, včetně čekání na návrat ze zavolané operace; v programátorském světě je to analogické době, po kterou je určitá hodnota ve „stack“ - v zásobníku) by)
- zpráva:
  - v našem příkladě máme jen jednoduché obyčejné zprávy : u nich se zpráva zobrazuje plnou čárou a plnou šipkou a v našem příkladu je znázorněno např. ohledně zprávy **zpravaA(x) : objekt obj1 posílá** (tedy je to sender) **zprávu zpravaA s jedním argumentem, který je je naplněn hodnotou atributu x, objektu obj2** (to je tedy receiver)

- zobrazení návratové zprávy není povinné (vždy si ji lze „domyslet“), ale může přidat na přehlednosti: zobrazuje se šipkou s čárkovnou čárou
- návratová zpráva může, ale nemusí vracet hodnotu : po poslání zpravaA(x) a po té, co obj2 dokončí svou činnost, tak vrátí zpět návratovou hodnotu 123, kterou si obj1 převezme do svého atributu y

V diagramu je znázorněna tato sekvence:

1. objekt obj1 je hned na počátku aktivován (má focus) a poslal zprávu zpravaA s parametrem x objektu obj2 : obj1 přeruší zpracování (ztratí focus a předá řízení obj2) a čeká, až obj2 odpoví na zpravaA(x)
2. obj2 získává focus a spustí vlastní metodu
3. v jistém bodě zpracování posílá obj2 zprávu zpravaB do obj3 - obj2 započne čekat na odpověď od obj3
4. obj3 získává focus (a spouští vlastní metodu)
5. obj3 dokončí své zpracování, pak vrátí peška (předá řízení zpět) do obj2
6. obj2 pokračuje v práci, a posílá další zrávu do obj3
7. až obj3 dokončí zpracování, tak vrátí řízení zpět do obj2
8. obj2 pokračuje v práci a až skončí, tak vrátí řízení do obj1, přitom ale vrátí návratový návratovou hodnotu 123
9. obj1 si převezme návratovou hodnotu do atributu y

Popis šipek:

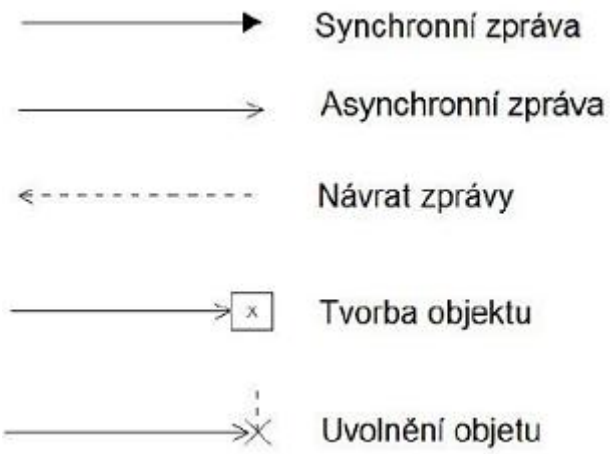


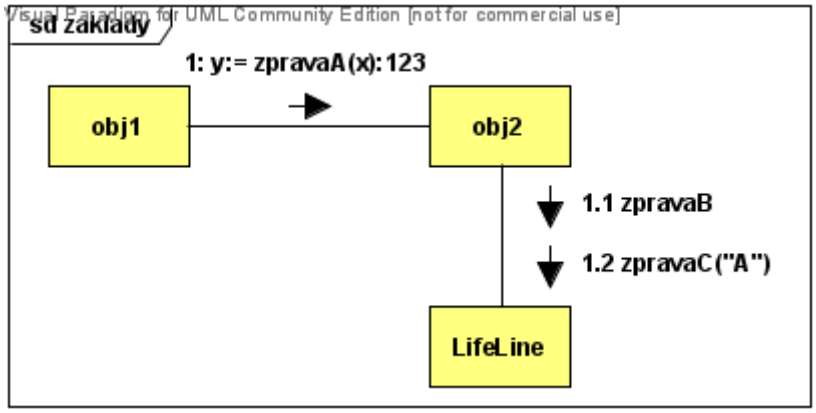
Diagram komunikace (Communication diagram)

Diagram komunikací je i není nový v UML 2.0 : totiž v UML 1.x se jmenoval collaboration diagram, což bylo matoucí (collaboration byly statické struktury, kdežto diagram zahrnoval zprávy).

D. komunikací a sekvenční d. jsou téměř izomorfní - tj. dají se převádět z jednoho tvaru na druhý (často i automatizovaně) (to však platí jen pro jednoduché sekvenční diagramy, které nepoužívají struktorované mechanismy jako interaction use a combined fragment). D. komunikací i sekvenční d. ukazují interakce, ale každý svým vlastním způsobem.

Použití diagramu komunikací bývá vhodnější v těch případech, kde chceme zdůraznit strukturální aspekty spolupráce, tj. ukázat hlavně kdo s kým komunikuje - jsou méně vhodné pro zdůraznění časových souvislostí interakcí.

Objekty si mohou posílat zprávy. Diagram komunikací ukazuje objekty (přesněji : části kompozitní struktury nebo role ve spolupráci(collaboration)) a spojení a zprávy, které si objekty posílají. Čas zde nevystupuje jako zvláštní dimenze, proto musí být sekvence zpráv a souběžnost threadů určena pomocí čísel sekvencí:



Jedná se o diagram komunikací odvozený z našeho jednoduchého sekvenčního diagramu

- uzly v tomto diagramu reprezentují části strukturované třídy nebo role collaborations (dále pro zjednodušení je budeme nazývat jako objekty) a korespondují s lifeline v sekvenčním diagramu.
- komunikační cesty (paths) jsou vyjádřeny čárami (spojkami, connectors) mezi uzly. Čáry můžou být nazvány svým jménem a/nebo jménem výchozí asociace (existuje-li). Můžou být vyjádřeny multiplicity.
- zpráva:
    - zpráva je zobrazena jako malá pojmenovaná šipka umístěná blízko spojnic; v našem příkladě máme jen jednoduché obyčejné zprávy : zde je šipka plná a v příkladu je znázorněno např. ohledně zprávy **zpravaA(x) : objekt obj1 posílá** (tedy je to sender) zprávu zpravaA s jedním argumentem, který je je naplněn hodnotou atributu **x**, objektu **obj2** (to je tedy **receiver**), ten po svém zpracování vrací hodnotu 123, kterou si obj1 převezme do atributu y
    - **sekvenční výraz** (umístěný před názvem zprávy) určuje pořadí, v jakém jsou zprávy posílány : pozor, není to jen prosté pořadové číslo, ale je v něm vyjádřeno i zanoření
    - návratováHodnota není povinná

V diagramu je znázorněna tato sekvence:

1. objekt **obj1** je hned na počátku aktivován (má focus) a poslal zprávu **zpravaA** s parametrem **x** objektu **obj2** : **obj1** přeruší zpracování (ztratí focus a předá řízení obj2) a čeká, až obj2 odpoví na zpravaA
2. **obj2** získává focus a spustí vlastní metodu
3. v jistém bodě zpracování posílá obj2 zprávu **zpravaB** do **obj3** - obj2 započne čekat na odpověď od obj3
4. **obj3** získává focus (a spouští vlastní metodu)
5. **obj3** dokončí své zpracování, pak vrátí peška (předá řízení zpět) do obj2
6. **obj2** pokračuje v práci, a posílá další zrávu do obj3
7. až **obj3** dokončí zpracování, tak vrátí řízení zpět do obj2
8. **obj2** pokračuje v práci a až skončí, tak vrátí řízení do obj1, přitom ale vrátí návratovou hodnotu 123
9. **obj1** si převezme návratovou hodnotu do atributu **y**

Diagram komunikací vlastně obsahuje jen čtyři typy elementů : základní frame, lifeline, komunikační cesty (paths) a zprávy.

Diagram přehledu interakcí (Interaction overview diagram)

Diagram přehledu interakcí definuje interakce pomocí varianty diagramu aktivit. Tato varianta zahrnuje fragmenty sekvenčních diagramů spolu s konstrukty pro řízení toku.

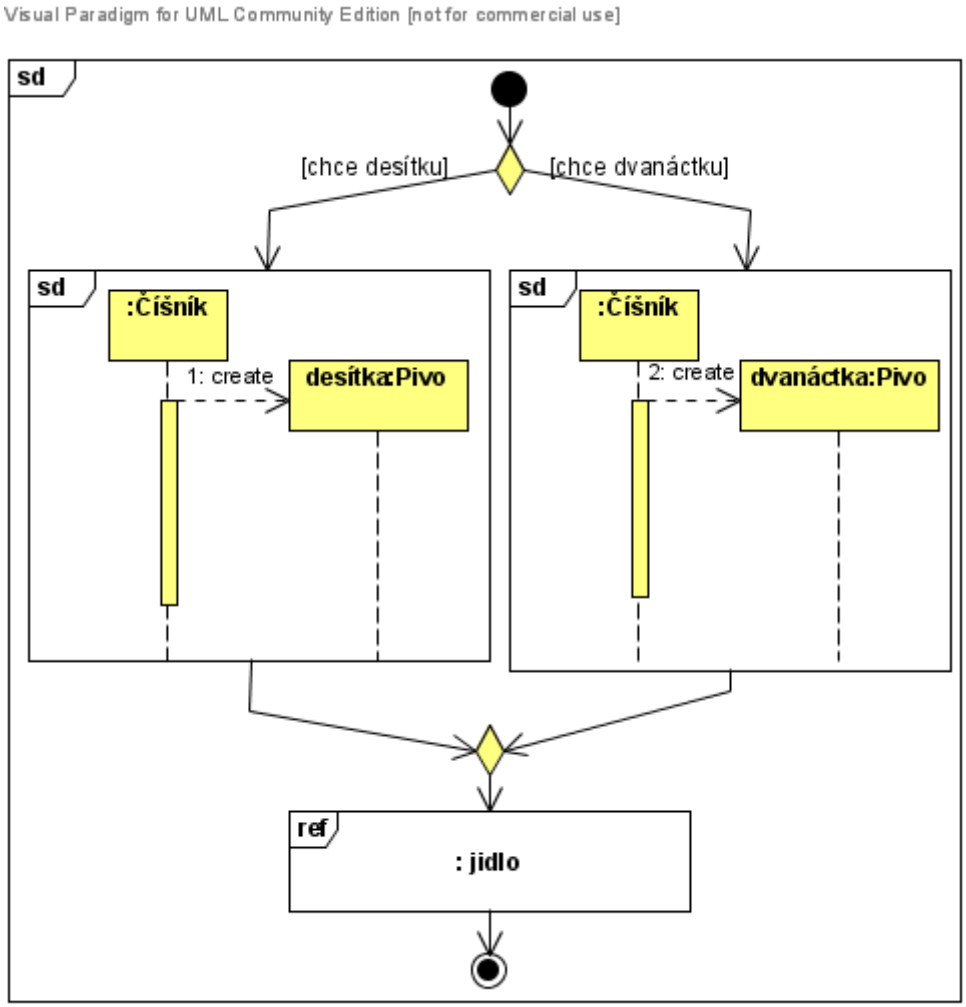


Diagram přehledu interakcí obsahuje jako své uzly interakce (online vložené interakční diagramy) a interaction use (ad interaction use viz sekvenční diagram)

- **frame** (rám): význam a jmenovka jsou stejné, jako v sekvenčním diagramu. Ovšem v refrenčním manuálu k UML se jeho autoři (trio amigos) odchylují od specifikace UML (sami to přiznávají), a používají jmenovku intover namísto sd
- **alternative combined fragment** zde má obdobu v uzlu decision (rozhodnutí) a odpovídajícím uzlu merge (splynutí)
- **paralel combined fragment** zde má obdobu v uzlu fork (rozdvojení) a odpovídajícím uzlu join (spojení)
- větvení a spojování zde musí být (na rozdíl od d. aktivit) explicitně uvedeno

Jazyk OCL

Úvod

- Object Constraint Language
- původně formální jazyk používaný k popsání výrazů na UML modelech
- nyní může být použit s jakýmkoliv Meta-Object Facility (MOF) Object Management Group (OMG) meta-modelem, včetně UML
- výrazy OCL typicky určují neměnné 1) podmínky, omezení systémů, které jsou modelovány, nebo dotazů nad objekty popsanými v modelu, které nemohou být postihnuty v diagramech
- jazyk, který umožňuje doplňovat UML o dodatečné informace
- hraje důležitou roli při analýze

OCL výrazy se dělí obecně do čtyř částí

1. kontext, který definuje situaci, při které je kontext validní
2. vlastnosti, které reprezentují nějaké charakteristiky kontextu (např. pokud je třída kontext, pak vlastností může být atribut)
3. operace (např. aritmetické, množinové)
4. klíčové slova (např. if, then, else, and, or) pro podmínky

Proč OCL?

- UML diagramy jako např. class diagramy samy o sobě neposkytují dostatečné prostředky k určení všech aspektů specifikace. Je potřeba upřesnit další omezení objektů v modelu. Tyto omezení jsou často popsány v přirozeném jazyku. Aby se vyhnulo víceznačným výkladům byly vytvořeny tzv. formální jazyky. Jejich nevýhodou tradičních formálních jazyků je nutnost mít značné matematické znalosti a tak je složité jejich použití pro manažery a systémové inženýry. OCL bylo vyvinuto, aby zaplnilo tuto mezeru. Je to formální jazyk, který zůstává snadný pro čtení i psaní. Vznikl u IBM, jeho předchůdce je Syntropy.
- OCL je čistě specifikační jazyk, nemá vedlejší efekt. Pokud výraz proveden, jednoduše vrací hodnotu, nemůže změnit nic v modelu. To znamená, že se stav systému nezmění provedením OCL výrazu, přestože OCL výraz může být použit pro specifikaci změny stavu (např. v post-conditions)
- OCL není programovací jazyk, není možno napsat programovou logiku (nelze vyvolat procesy, nebo aktivovat operace, které nesouvisí s dotazy) v OCL. OCL výrazy nejsou spustitelné
- OCL je typový jazyk. Každý OCL výraz má typ. Pro well-formed OCL výraz musí odpovídat typovým pravidlům jazyka. Např. najde porovnávat integer a string. Každý klasifikátor popsaný v modelu UML reprezentuje odlišný OCL typ. Navíc OCL obsahuje předdefinované typy.
- Jako specifikační jazyk používaný při analýze nesmí obsahovat implementační záležitosti
- Provedení OCL výrazu je okamžité. To znamená, že se stav objektu v modelu nemůže změnit během provádění výrazu.

Použití OCL

- dotazovací jazyk
- ke specifikaci neměnnosti na třídách a typech v class diagramu
- ke specifikaci neměnnosti typů pro stereotypy
- k popsání pre a post conditions operací a metod
- k specifikaci cílů zpráv a akcí
- ke specifikaci omezení operací
- ke specifikaci odvozujících pravidel atributů jakýkoliv výrazů nad UML modelem

OCL poskytuje operace

- upřesňování omezujících pravidel
- upřesňování provozních pravidel během modelování
- upřesňování těl vyhledávacích dotazů
- důvody použití
- umožnění modelovacím nástrojům vhodné generování kódu
- umožňuje přesnější modelování (výsledkem bude méně nesprávných výkladů modelů)
- umožňuje modelovacím nástrojům „přemýšlet o modelech UML“ (což např zahrnuje ověřevání konzistence modelů)

Nevýhody

- obtížná čitelnost, nepravidelná syntaxe
- není příliš rozšířený, zná ho jen málo lidí
- někdy přesnost, kterou OCL nabízí nemusí být potřeba

Výrazy jazyka OCL

- připojují se k modelovaným prvkům UML
    - kontextu balíčku (nepovinný): definuje jmenný prostor pro výrazy v jazyce OCL
    - kontext výrazu (povinný): definuje kontextovou instanci výrazu
    - jednoho nebo více výrazů
- kontext výrazů lze definovat např. vložením výrazu OCL do poznámky, která se připojí k modelovanému prvku

Příklady použití

- v diagramech interakce se používá k určování
    - kontrolní podmínky
    - selektorů pro čáry života
    - argumentů zpráv
- v diagramech aktivit se používá k určování
    - akčních uzlů volání
    - kontrolních podmínek pro přechody
    - objektových uzlů
    - stavů objektů
- ve stavových diagramech se používá k určování
    - kontrolních podmínek
    - podmínek stavů
    - cílů akcí

    - hodnot atributů

Zdroje

- http://en.wikipedia.org/wiki/Object_Constraint_Language
- http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL
- http://st.inf.tu-dresden.de/oclportal/
- http://lcm.csa.iisc.ernet.in/soft_arch/OCL.htm
- http://www.csci.csusb.edu/dick/samples/ocl.html
- http://cs.ulb.ac.be/public/_media/teaching/infoh302/oclnotes.pdf?id=teaching%3Ainfoh302&cache=cache
- http://www.dcs.fmph.uniba.sk/diplomovky/obhajene/getfile.php/diplomka.pdf?id=96&fid=161&type=application%2Fpdf

CASE nástroje

Úvod

- CASE (z anglického Computer-aided software engineering, počítačem podporované softwarové inženýrství) je použití softwaru pří vývoji (resp. údržbě) počítačových programů, za účelem dosáhnutí vyšší kvality, bezchybnosti, udržovatelnosti apod. Snižují čas vývoje a redukují jeho cenu. Také souvisí s metodami pro vývoj informačních systémů spolu s automatickými nástroji, které mohou být využity během procesu vývoje software. Podpora může probíhat v různých stádiích životního cyklu programu – při sběru požadavků, analýzách, návrhu, programování (s pomocí IDE) atd.
- Nástroje používané pro tento účel se obvykle nazývají CASE nástroje. Některé z nich slouží k automatizaci vývojového procesu - analýza, návrh, programování a dokumentaci.
- Dvě klíčové myšlenky CASE jsou:
    - počítačová podpora ve vývoji software a nebo údržbě software
    - inženýrský přístup k vývoji software a nebo jeho údržbě
- Mnoho nástrojů negeneruje jen kód ale i další analýzy a návrhové techniky
- Některé obvyklé CASE nástroje:
    - nástroj pro generování zdrojového kódu
    - nástroj pro datové modelování
    - UML
    - nástroje pro refaktorování
    - správa konfigurací
    - nástroje pro trasformace

Case tools

Existující CASE prostředí mohou být rozdělena do 4 oblastí:

1. Life-cycle nástroje
    - Vyšší nástroje: Podpora plánování, konstrukce na konceptuální úrovni, ignoruje návrhový aspekt. Zachycuje, analyzuje a organizuje modely systémů, které návrhářům pomáhají stanovit jejich korektní chování: diagramy: ER, Data-Flow, strukturní grafy, rozhodovací stromy atd.
    - Nižší nástroje: zaměřují se na pozadí, podporují fyzický návrh (programování), debugging, integraci softwarových komponent, údržbu, reengineering, reverzní inženýrství atd. Tyto nástroje klasifikují rozsahy a hranice, popisují současný systém, požadavky, podporují prototypování, připravují návrh atd.
2. Integrační nástroje
3. Konstrukční nástroje
4. Znalostní nástroje

Case Workbenches

- integrují více CASE nástrojů do jedné aplikace na podporu specifických operací při vývoji software. Tím dosahují:
    - homogenního a konzistentního prostředí
    - jednoduchý přístup k jednotlivým nástrojům
    - integrace dat
- můžou být rozděleny do 8 tříd:
    1. Business planning and modelling
    2. Analýza a návrh
    3. Vývoj uživatelských rozhraní
    4. Programování
    5. Verifikace a validace
    6. Údržba a reverzní inženýrství
    7. Konfigurační management
    8. Projekt management

Case environments

Spojení jednotlivých nástrojů a workbenches, které podporují softwarové procesy. Jsou klasifikovány podle toho, na jakém základě probíhá integrace:

1. Toolkits: typicky přináší podporu pro programování, konfigurační management a projekt management. Toolkit sám vychází ze základů nástrojů operačních systémů. Výsledné soubory jsou nestrukturované a můžou být v různých formátech, takže přístup z jiných nástrojů může vyžadovat konverzi.
2. Jazykově orientované: prostředí je napsáno v jazyku, pro který bylo vyvinuto, což umožňuje jeho znovupoužití, upravení a rozšíření. Integrace kódu v různých jazycích je častým úkolem pro tyto prostředí. Jejich síla spočívá v dobré prezentaci a kontrole integrace. Např. Smalltalk.

3. Integrované: dosahují integrace prezentace pomocí konzistentního a srozumitelného interfacu. Integrace dat je dosažena pomocí repozitářů. Mají speciální databází obsluhující všechny informace, které jsou pak dostupné z prostředí.
4. Čtvrté generace: první nástroje pro integrované prostředí. Podporují vývoj specifické třídy programů: pro zpracování elektronických dat a business-oriented aplikace. Obecně zahrnují programovací nástroj, jednoduchá konfigurační management, vybavení pro správu dokumentů a občas i generátor kódu v nižších jazycích.
5. Procesně orientované: pracuje s procesním modelem vytvořeným pomocí specializovaných nástrojů, dvě hlavní funkce: realizace process-modelu a tvorbu process-modelu

Aplikace

Zaměřují se na:

- modelování business/reálných procesů a toků dat
- vývoj datových modelů ve formě diagramů
- vývoj popisů procesů a funkcí
- produkci procedur vytvářejících SQL a další úložiště

Výhody

- umožňují lepší pohled na systém
- obsahují nástroje pro komunikaci mezi členy teamu
- nástroje jsou více efektivní pro systémy velkého rozsahu a složitosti
- umožňují dobrý náhled na procesy a logiku
- zvyšují kvalitu a produktivitu software
- snižují čas na opravu chyb a údržbu
- zlepšují čitelnost a udržovatelnost systému

Nevýhody

- neadekvátní standardizace: používání nástrojů od různých firem, které lze těžko propojit pokud nepoužívají stejné standardy, protokoly apod.
- nerealistická očekávání: management nasazuje CASE nástroje k redukování výdajů, ale to zahrnuje vysokou pořizovací cenu, návratnost je pomalá a je potřeba správné nasazení
- rychlá implementace: na CASE nástroje je potřeba si zvyknout, jejich používání vyžaduje trénink a neměly by být nasazeny rovnou na velký, složitý projekt, ale prvně na menší, méně složitý
- slabá kontrola repozitářů: dochází k narušení bezpečnosti, zničení pracovních dokumentů, systémových návrhů, nebo kódu uložených v repozitářích. Kontrola zahrnuje chráněný přístup, verzování a zálohování

Zdroje

- http://en.wikipedia.org/wiki/Computer-aided_software_engineering
- http://cs.wikipedia.org/wiki/CASE
- http://www.npd-solutions.com/case.html
- http://www.theecommercesolution.com/usefull_links/case_tools.php

Požadavky a jejich modelování

- Požadavky identifikují nezbytné atributy, schopnosti, charakteristiky, nebo kvality systému pro to, aby byl prospěšný uživatelům.
- Specifikace požadavků zákazníka je první fází návrhu informačního systému, slouží k poznání potřeb budoucího uživatele systému
- Cílem je vymezit systém z heldiska požadavků na funkcionalitu, technologie, bezpečnost atd.
- Požadavky se využívají pro návrh, jejich zpracování může být rozděleno na:
    - vyvolání požadavků: získání, pochopení, revize, vyjádření účastníků. Vhodné si předem připravit seznam otázek, pečlivě zaznamenat, nechat si podepsat výsledek jednání. Studium podnikových norem zákazníka, studium standardů problémové domény, zapojují se vlastní zkušenosti projektanta
    - analýza požadavků: kontrola konzistence a kompletnosti
    - specifikace: dokumentace požadavků, nutno se zaměřit na tyto faktory: zaměření věcných (podnikových) činností, okolí podniku, legislativu, strategické a podnikatelské cíle, omezení zdrojů (personální, technické …), technické změny, finanční omezení, termínová omezení atd.
    - validace: kontrola, zda jsou specifikované požadavky v pořádku, odpovídají přáním zákazníka
- Požadavky vyjadřují, jaké elementy a funkce systém vyžaduje.

Zásady specifikace požadavků

- Nepodceňovat žádný požadavek
- Rozlišovat mezi požadavkem a způsobem řešení
- V každém požadavku vyjmenovat sice stručně, ale vše, co se v něm požaduje
- Popsat každý požadavek- nedělat selekci
- Nepoužívat technologické (uživatelům nesrozumitelné) pojmy

Typy požadavků

- Business: co je potřeba dodat, nebo splnit, aby byla tvořena nějaká hodnota
- Product: popisuje vlastnosti systému nebo produktu (což může být jedna z několika způsobů, jak splnit business požadavky)
- Process: popisuje aktivity, které má systém vykonávat

Funkční požadavky

- definují co má budoucí systém nabízet uživatelům za služby, jeho chování
- mohou mít podobu výčtu požadovaných funkcí
- nebo cílů, které chce zadavatel prostřednictvím systému dosáhnout
- modelování: use-cases

Nefunkční požadavky

- požadavky na kvalitu (např. rychlost odezvy)
- požadavky na použité technologie
- požadavky na organizaci projektu
- požadavky na formu a obsah dokumentace
- modelování: je potřeba je zaznamenat a neustále je promítat do návrhu řešení
- záznam mívá formu katalogového záznamu
- CASE nástroje umožňují požadavky zaznamenat, dát je do vzájemných vztahů a zobrazit následný způsob jejich realizace

Charakteristiky správných požadavků

- **Soudržnost (koherence)-** požadavek se týká jen jedné věci
- **Kompletnost**- požadavek je popsán na jednom místě a nic v něm nechybí
- **Konzistence**- požadavek není v rozporu s jiným požadavkem a je plně konzistentní s ostatní dokumentací
- **Atomicita**- neobsahuje další spojení, např. požadavek poštovní směrovací číslo musí validovat americké a kanadské poštovní kódy se musí rozepsat do dvou požadavků
- **Trasovatelnost**- všechny schůzky a požadavky musí být řádně zaznamenány
- **Současnost**- požadavky nesmí být zastaralé
- **Uskutečnitelnost**- požadavky musí být implementovány bez omezení projektu
- **Jednoznačnost**- nesmí obsahovat nejasnosti
- **Závaznost**- všechny požadavky jsou zákazníkem definované charakteristiky, bez nichž nemůže být systém správně navrhnut
- **Ověřitelnost**- implementace požadavků může být ověřena jedním z těchto způsobů: inspekce, demonstrace, test nebo analýza

Zdroje

- http://en.wikipedia.org/wiki/Requirement
- http://en.wikipedia.org/wiki/Functional_requirements
- http://en.wikipedia.org/wiki/Non-functional_requirements
- http://www.microtoolsinc.com/Howsrs.php
- http://edu.uhk.cz/~svoboka1/download/OMO1/04_pozadavky_a_typove_ulohy.zip

Zdroje

- http://statnice.stm-wiki.cz/lib/exe/fetch.php?media=wiki:downloads:02_sin_arlowprednasky.zip
- http://statnice.stm-wiki.cz/lib/exe/fetch.php?media=wiki:downloads:02_sin_arlowbook.zip
- http://mpavus.wz.cz/uml/uml-uvod-1.php
- Martin Fowler: Destilované UML