7. Príklady UML modelov

Spracované podľa Thomas A. Pender: UML Weekend Crash Course, Wiley Publishing, 2002. Používa sa UML 1.5, pričom diagramy spolupráce nazývame už podľa UML 2.x ako diagramy komunikácií. V diagramoch komponentov je iná notácia, inak možno všetko použiť aj pre UML 2.x. A aj v prípade inej notácie podstata modelov sa nemení a tá predstavuje najdôležitejší prínos tohto textu.

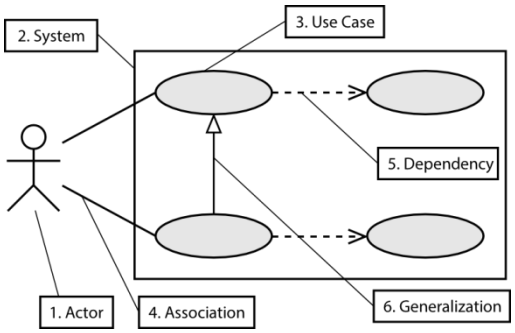
7.1. Diagram prípadov použitia

Diagram prípadov použitia (angl. Use Case diagram) pozostáva z piatich jednoduchých grafických elementov, ktoré reprezentujú systém, účastníkov, prípady použitia, asociácie a závislosti. Diagram prípadov použitia vysvetľuje vzťahy medzi systémom a vonkajším svetom na vysokej úrovni abstrakcie. Diagram prípadov použitia poskytuje iba „povrchový“ pohľad na systém (t.j. systém opisuje ako čiernu skrinku).

Napríklad, pohľad reprezentovaný diagramom prípadov použitia pre bankomat by mohol korešpondovať s hlavnou obrazovkou bankomatu a možnosťami (v menu) dostupnými na danej úrovni. Bankomat poskytuje používateľovi možnosti ako výber, vklad, výpis stavu účtu a platba. Každá z týchto možností môže byť reprezentovaná samostatným prípadom použitia. Zákazník je asociovaný s každým z prípadov použitia, ktoré plánuje použiť.

7.1.1. Elementy diagramu prípadov použitia a notácia

Diagram prípadov použitia pozostáva zo šiestich modelovacích elementov: systémy, účastníci, prípady použitia, asociácie, závislosti a zovšeobecnenia.



UCD 1: Elementy diagramu prípadov použitia.

- **Systém:** Určuje hranice systému vo vzťahu s účastníkmi, ktorí ho používajú a službami, ktoré systém musí poskytovať.
- **Účastník:** Osoba, systém alebo zariadenie, ktoré hrá rolu v jednej alebo viacerých interakciách s modelovaným systémom.
- **Prípad použitia:** Určuje kľúčovú funkcionalitu systému. Bez tejto funkcionality systém nesplní požiadavky používateľa/účastníka. Každý prípad použitia vyjadruje cieľ, ktorý musí systém dosiahnuť.

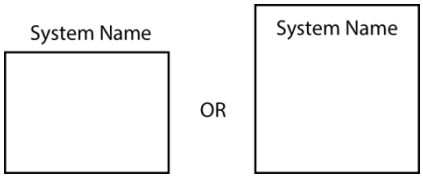
- **Asociácia (väzba):** Identifikuje interakciu medzi účastníkmi a prípadmi použitia. Každá asociácia sa stáva dialógom, ktorý musí byť vysvetlený v opise prípadu použitia. Každý opis (prípadu použitia) zase poskytuje množinu scenárov, ktoré sa používajú ako testovacie prípady (angl. test cases) pri vyhodnocovaní analýzy, návrhu a implementácie prípadu použitia.
- **Závislosť:** Určuje komunikačný vzťah medzi dvoma prípadmi použitia.
- **Zovšeobecnenie:** Definuje vzťah medzi dvoma účastníkmi alebo dvoma prípadmi použitia, kde jeden prípad použitia dedí a pridáva alebo prekonáva vlastnosti druhého prípadu použitia.

Systém

Jednou z prvých úloh v projekte je určenie kontextu a rozsahu navrhovanej aplikácie. Je potrebné zodpovedať otázky ako: Čo všetko je potrebné zahrnúť do systému? Ako súvisí navrhovaný systém s inými systémami vo vašej architektúre? Kto plánuje používať tento systém?

Toto všetko by mohlo byť opísané v rozsiahlom dokumente. Ale ako sa hovorí, jeden obrázok má cenu tisíc slov. Toto tvrdenie pomáha vysvetliť jednoduchosť notácie systému – obdĺžnik s názvom (obrázok). Ikona systému poskytuje kontext, do a okolo ktorého môžeme umiestniť elementy, ktoré ovplyvňujú konštrukciu systému.

Pozn.: Je potrebné povedať, že ikona systému sa používa iba zriedkavo. Má tendenciu byť príliš reštriktívna a do diagramu nepridáva žiadnu zásadnú informáciu. Preto vo väčšine nástrojov uvidíme iba prípady použitia, účastníkov a ich vzťahy.

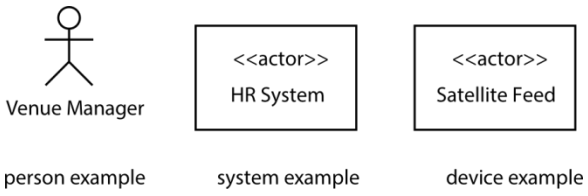


UCD 2: Ikona systému pre diagram prípadov použitia.

Účastníci

Systém má vždy nejakých používateľov. Používatelia v bežnom zmysle predstavujú ľudí, ktorí systém používajú. Ale používateľmi môžu byť aj iné systémy alebo zariadenia, ktoré si medzi sebou vymieňajú informácie.

V diagramoch prípadov použitia sa ľudia, systémy a zariadenia nazývajú *účastníci* (aktéri, hráči, angl. actors). Ikony, pomocou ktorých sa modelujú môžu byť rôzne, ale koncept ostáva rovnaký. Účastník predstavuje *rolu*, ktorú hrá externá entita vo vzťahu k systému. Zdôrazňujeme, že účastník predstavuje *rolu*, nemusí ísť nevyhnutne o konkrétnu osobu alebo špecifický systém. Obrázok zobrazuje príklady niekoľkých ikon účastníkov.



UCD 3: Ikony účastníkov pre diagram prípadov použitia.

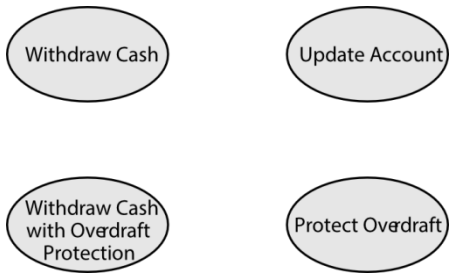
Napríklad, účastník môže byť v roli skladníka, ktorý umiestňuje produkty do inventára. Neskôr v ten istý deň môže tá istá osoba pracovať pri odoberaní produktov z dodávkového auta. Tá istá osoba hrala dve roly. Podobne, viacero ľudí môže pracovať v rovnakej role. Napríklad, skladiská majú viacerých skladníkov.

Používanie rolí nám pomáha sústrediť sa na to, ako bude systém používaný, namiesto sústredenia sa na aktuálnu organizáciu spoločností do názvov pracovných pozícií a zodpovedností. Veci, ktoré ľudia robia musia byť v systéme oddelené od názvov ich aktuálnych pracovných pozícií, aby sme si vedeli poradiť so zmenami, ktoré sú nevyhnutne potrebné v každom systéme.

Ako identifikujeme účastníkov? Treba venovať pozornosť opisom systému a rolám, ktoré ľudia vykonávajú, keď používajú systém. Keď viacero ľudí vykonáva rovnakú funkciu, snažme sa túto spoločnú rolu pomenovať.

Prípady použitia

Prípady použitia (angl. Use Cases) definujú požadované funkcie systému. Každý prípad použitia je pomenovaný slovesnou frázou, ktorá vyjadruje cieľ, ktorý musí systém dosiahnuť, napríklad: vložiť peniaze, vybrať peniaze, prispôsobiť účet (obrázok). Hoci každý prípad použitia implikuje podporný proces, zameriavajte sa na cieľ, nie na proces.



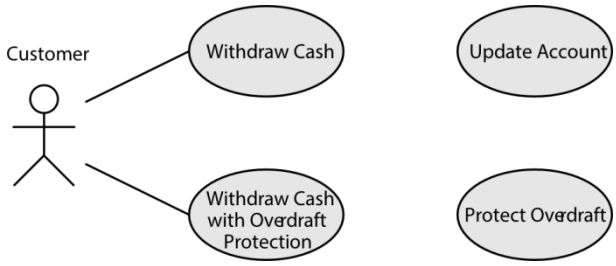
UCD 4: Notácia prípadov použitia pre diagram prípadov použitia.

Definovanie prípadov použitia týmto spôsobom definuje systém ako množinu požiadaviek, nie riešenie. Neopisujeme ako systém musí pracovať. Opisujeme, čo systém musí vedieť robiť. Prípady použitia opisujú iba tie funkcionality, ktoré sú viditeľné a zmysluplné pre účastníkov, ktorí používajú systém. Uvedomenie si tejto skutočnosti vám pomôže vyhnúť sa *funkcionálnej dekompozícii*, rozkladaním procedúr a úloh na menšie a menšie procesy, pokým sa nepopíše celé vnútorné fungovanie systému.

Asociácia

Čiara spájajúca účastníka a prípady použitia reprezentuje asociáciu (angl. association), ako na obrázku. Asociácia reprezentuje fakt, že účastník komunikuje s prípadom použitia. Toto je jediný platný vzťah medzi účastníkom a prípadom použitia. Podľa špecifikácie UML, môžeme pre označenie smeru komunikácie špecifikovať smerovú šípku na oboch koncoch čiary asociácie. Niektoré asociácie sú jednosmerné (napríklad účastník špecifikuje informáciu prípadu použitia). Väčšina asociácií je však obojsmerných (t.j. účastník pristupuje k prípadu použitia a prípad použitia poskytuje účastníkovi funkcionalitu). Pre obojsmerné asociácie môžeme umiestniť šípky na oba konce čiary asociácie, alebo jednoducho obe šípky vynechať. Väčšina používateľov UML zvykne šípky úplne vynechať. Väčšina modelovacích nástrojov poskytuje možnosť zobraziť alebo skryť obojsmerné šípky.

Kľúčové je identifikovať, ku ktorým prípadom použitia potrebuje účastník pristupovať. Tieto prepojenia vytvárajú základ pre rozhrania systému a pre množstvo úsilia, ktoré vynaložíte pri ďalšom modelovaní.



UCD 5: Notácia asociácie pre diagram prípadov použitia.

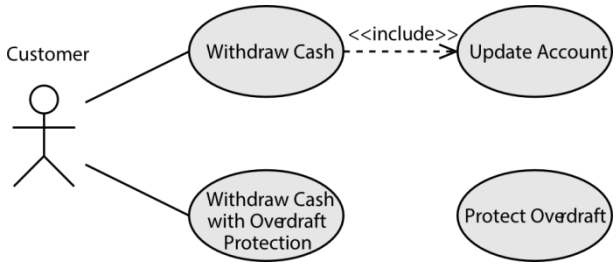
Závislosť <<include>>

Niekedy môže prípad použitia požiadať o pomoc od iného prípadu použitia. Napríklad, prípady použitia s názvami *Vložiť peniaze* a *Vybrať peniaze* v skutočnosti nemusia aktualizovať stav bankového účtu. Môžu delegovať zmeny existujúcemu prípadu použitia Aktualizovať účet tak, aby boli zmeny kontrolované cez jednu službu, ktorá garantuje, že sa všetky zmeny vykonajú korektne.

Keď jeden prípad použitia deleguje inému, závislosť sa nakreslí prerušovanou šípkou z „používajúceho“ prípadu použitia do „používaného“ prípadu použitia a šípka sa označí stereotypom <<include>>, ako na obrázku. Toto vyjadruje, že vykonávanie „používajúceho“ (alebo volajúceho) prípadu použitia zahrnie alebo obsiahne funkcionalitu „používaného“ prípadu použitia.

Delegovanie môže vzniknúť z dvoch dôvodov:

1. Pre vykonanie potrebnej úlohy už existuje prípad použitia.
2. Niekoľko prípadov použitia potrebuje vykonať rovnakú úlohu. Namiesto opakovaného písania rovnakej logiky je spoločná úloha izolovaná do vlastného prípadu použitia a znovupoužitá alebo zahrnutá do každého prípadu použitia, ktorý ju potrebuje.

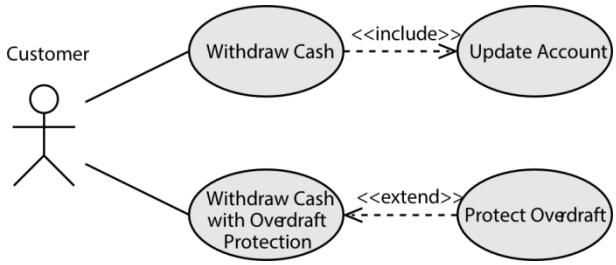


UCD 6: Notácia závislosti <<include>> pre diagram prípadov použitia.

Závislosť <<extend>>

Stereotyp závislosti <<extend>> hovorí, že prípad použitia *môže* potrebovať pomoc od iného prípadu použitia. Pre porovnanie, stereotyp závislosti <<include>> hovorí, že jeden prípad použitia bude *vždy* volať iný prípad použitia. Niekde v logike prípadu použitia, ktorý potrebuje pomoc je *bod rozšírenia* – testovacia podmienka, ktorá určuje, či sa má volanie uskutočniť. Takáto podmienka sa pri stereotype <<include>> nenachádza.

Ďalší rozdiel medzi týmito dvomi stereotypmi je v smere šípky závislosti. Šípka závislosti <<include>> smeruje z hlavného prípadu použitia (ktorý sa práve vykonáva) do toho, od ktorého potrebuje pomoc. Šípka závislosti <<extend>> smeruje z rozširujúceho prípadu použitia (poskytujúceho extra pomoc) do hlavného prípadu použitia, ktorému poskytuje pomoc (obrázok).



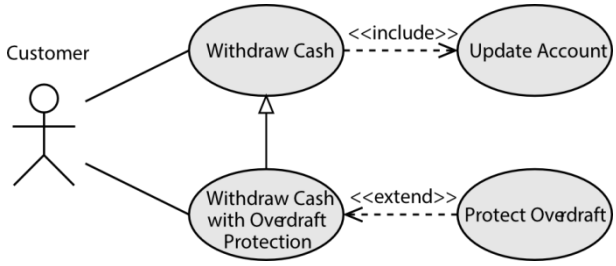
UCD 7: Notácia závislosti <<extend>> pre diagram prípadov použitia.

Zovšeobecnenie

Dedenie je kľúčový koncept v objektovo-orientovanom programovaní a OO analýze a návrhu. Dedenie nám hovorí, že objekt má v čase vytvorenia prístup ku všetkým vlastnostiam inej triedy (okrem svojej vlastnej triedy). Preto vytvorený objekt obsiahne všetky tieto vlastnosti do svojej vlastnej definície. Laicky povedané, hovoríme veci ako: „Ford Explorer je auto.“ Auto je dobre definovaný všeobecný koncept. Keď vytvárame Ford Explorer, namiesto opätovného definovania všetkých vlastností auta jednoducho „zdedíme“ alebo asimilujeme všetky existujúce vlastnosti auta a následne prekonáme a/alebo pridáme nové vlastnosti pre kompletizáciu definície nového objektu Ford Explorer.

Rovnaká myšlienka aplikovaná na účastníkov a prípady použitia sa nazýva *zovšeobecnenie* (generalizácia, angl. generalization) a často je označované ako vzťah „*je*“. Senior Bank Teller *je* Bank Teller s ďalšími autoritami a zodpovednosťami. Prípad použitia „Withdraw Cash with Overdraft Protection“ *je* rozsiahlejšia požiadavka ako prípad použitia „Withdraw Cash“.

Pre modelovanie zovšeobecnenia používa UML plnú čiaru a prázdny trojuholník. Vyzerá trochu ako šípka, ale treba byť opatrný a nezamieňať ich. Trojuholník je vždy na konci pri položke, ktorá bude zdedená. V spomínaných príkladoch by bol trojuholník pri „Bank Teller“ a „Withdraw Cash“, ako na obrázku.



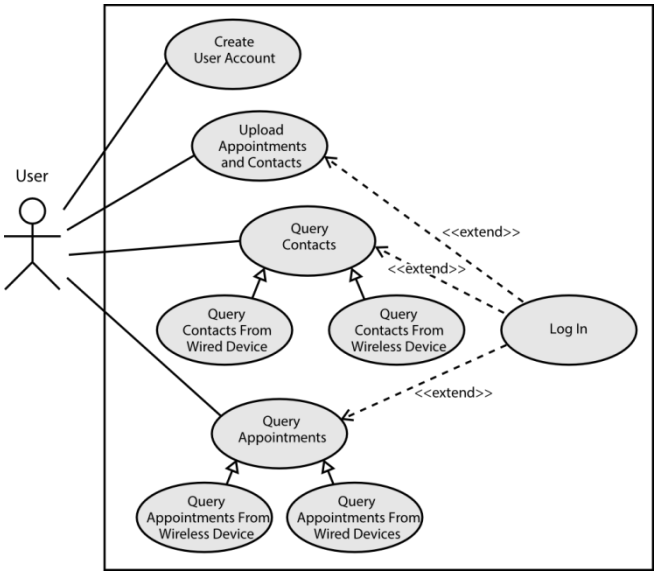
UCD 8: Notácia zovšeobecnenia pre diagram prípadov použitia.

7.1.2.Postup tvorby diagramu prípadov použitia

1. Určí sa kontext cieľového systému.
2. Identifikujú sa účastníci.
3. Identifikujú sa prípady použitia.

4. Definujú sa asociácie medzi účastníkmi a prípadmi použitia.
5. Zhodnotia sa účastníci a prípady použitia pre nájdenie príležitostí na zlepšenie.
6. Zhodnotia sa prípady použitia pre závislosti <<include>>.
7. Zhodnotia sa prípady použitia pre závislosti <<extend>>.
8. Zhodnotia sa účastníci a prípady použitia pre zovšeobecnenie.

7.1.3.Príklady



UCD 9: Diagram prípadov použitia – Friendly Reminder system.

7.2. Diagram tried

Diagram tried (angl. Class diagram) reprezentuje triedy, ich komponenty a spôsob akým triedy objektov navzájom súvisia. Diagram tried obsahuje atribúty, operácie, stereotypy, vlastnosti, asociácie a dedenie.

- **Atribúty** opisujú stav triedy objektov.
- **Operácie** definujú správanie triedy objektov.
- **Stereotypy** pomáhajú pochopiť typ objektu v kontexte iných tried objektov s podobnými rolami v rámci návrhu systému.
- **Vlastnosti** poskytujú spôsob, ako sledovať údržbu a stav definície triedy.
- **Asociácia** je len formálny termín pre typ vzťahu, v ktorom sa môže zúčastňovať daný typ objektu. Asociácie sa môžu vyskytovať v rôznych obmenách, ako napríklad jednoduchá asociácia, zoskupenie a kompozícia, kvalifikovaná a reflexívna asociácia.
- **Dedenie** umožňuje organizovať definície tried pre zjednodušenie a uľahčenie ich implementácie.

Tieto elementy spolu poskytujú bohatú množinu nástrojov pre modelovanie softvéru a biznis problémov.

7.2.1.Elementy definície triedy

Symbol triedy pozostáva z troch *oddelení* (obdĺžnikových plôch), ktoré obsahujú charakteristické informácie potrebné pre opis vlastností jedného typu objektu.

- *Oddelenie s názvom* jednoznačne definuje triedu (typ objektu) v rámci balíka. Ak sú triedy umiestnené v rôznych balíkoch, môžu mať rovnaký názov.
- *Oddelenie atribútov* obsahuje všetky dátové definície.
- *Oddelenie operácií* obsahuje definíciu pre každé správanie podporované daným typom objektu.

7.2.2.Modelovanie atribútu

Atribút (angl. attribute) opisuje kúsok informácie, ktorú objekt vlastní alebo o sebe vie. Aby sme vedeli túto informáciu použiť, musíme jej priradiť názov a následne špecifikovať druh informácie alebo dátový typ. Dátové typy môžu byť elementárne (poskytnuté jazykom) alebo abstraktné (definované vývojárom). Navyše, každý atribút môže mať pravidlá ohraničujúce hodnoty, ktoré mu môžu byť priradené. Východisková hodnota častokrát pomáha zabezpečiť, že atribút bude vždy obsahovať platné, zmysluplné dáta.

Viditeľnosť atribútov

Každá definícia atribútu musí taktiež špecifikovať, ktoré objekty sú oprávnené daný atribút vidieť – t.j. ich *viditeľnosť* (angl. visibility). Viditeľnosť je definovaná nasledovne:

- Verejná (+) viditeľnosť (angl. public visibility) dáva prístup objektom všetkých tried.
- Súkromná (–) viditeľnosť (angl. private visibility) limituje prístup iba v rámci triedy samotnej. Napríklad, iba operácie triedy majú prístup k súkromným atribútom.
- Chránená (#) viditeľnosť (angl. protected visibility) dáva prístup podtriedam. V prípade *zovšeobecnení* (dedenia), podtriedy musia mať prístup k atribútom a operáciám nadtriedy, inak nemôžu byť zdedené.
- Balíková (~) viditeľnosť (angl. package visibility) dáva prístup objektom v tom istom balíku.

Symboly jednotlivých typov viditeľnosti poskytujú pohodlný skrátený zápis a zvyknú sa používať namiesto celých názvov. Nasledovná notácia predstavuje bežný spôsob definovania atribútu:

viditeľnosť / názov atribútu : dátový typ = východisková hodnota {ohraničenia}

Tu uvádzame podrobný prehľad jednotlivých elementov výrazu.

- **Viditeľnosť (+, –, #, ~):** Povinná pred generovaním kódu. Programovací jazyk zvyčajne špecifikuje platné možnosti.
- **Lomka (/):** Indikátor odvodeného atribútu je voliteľný. Odvodené hodnoty môžu byť vypočítané alebo zistené z iných dát a množiny pravidiel alebo vzorcov.
- **Názov atribútu:** Povinný. V rámci triedy musí byť unikátny.
- **Dátový typ:** Povinný. Toto je dôležité. Počas analýzy by mal dátový typ odrážať ako klient vidí dáta. Môžete si to predstaviť ako externý pohľad. Počas návrhu musí dátový typ reprezentovať dátový typ programovacieho jazyka pre prostredie, v ktorom bude trieda naprogramovaná.
- **Operátor priradenia a východisková hodnota:** Voliteľné. Východiskové hodnoty slúžia na dva dôležité účely. Východiskové hodnoty môžu poskytnúť významné vylepšenia v jednoduchosti používania pre klienta. Východiskové hodnoty však predovšetkým chránia integritu systému pred škodami spôsobenými chýbajúcimi alebo neplatnými hodnotami.

- **Ohraničenia:** Ohraničenia (angl. constraints) vyjadrujú všetky pravidlá potrebné pre garantovanie integrity daného kusu informácie. Kedykoľvek sa objekt pokúsi upraviť hodnotu atribútu, musí vyhovieť pravidlám stanoveným v ohraničeniach.
- **Atribút na úrovni triedy (podčiarknutá deklarácia atribútu):** Voliteľný. Značí, že všetky objekty danej triedy sa delia o jednu spoločnú hodnotu atribútu (*statická* hodnota).

7.2.3.Modelovanie operácie

Správanie objektov sa modeluje pomocou operácií. Operácie vyžadujú názov, argumenty a niekedy aj informáciu o návratovej hodnote. Argumenty alebo vstupné parametre sú jednoduché atribúty, takže sa špecifikujú použitím notácie atribútu (názov, dátový typ, ohraničenia a východisková hodnota), hoci je veľmi bežné použiť skrátenú formu, ktorá obsahuje iba názov a dátový typ.

Ak použijeme na argument ohraničenia, ohraničujete vstupné hodnoty, nie hodnoty atribútu, ktorý patrí zdrojovému objektu. Hodnota v zdrojovom objekte bola ohraničená v definícií atribútu v rámci triedy.

Informácia o návratovej hodnote predstavuje dátový typ atribútu. Môžeme špecifikovať aj viditeľnosť operácie (+, –, #, ~). Nasledovná notácia predstavuje bežný spôsob definovania operácie:

> **viditeľnosť názovOperácie (názovarg : dátový typ {ohraničenia}, ...) : dátový typ návratovej hodnoty {ohraničenia}**

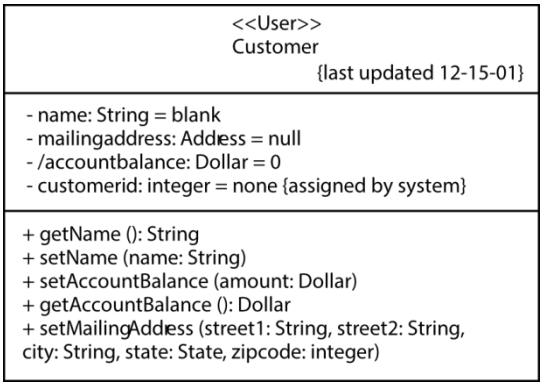
Tu uvádzame podrobný prehľad jednotlivých elementov výrazu.

- **Názov operácie:** Povinný. Nemusí byť unikátny, ale je potrebné, aby kombinácia názvu a parametrov bola v rámci triedy unikátna.
- **Argumenty/parametre:** Je povolené použiť ľubovoľný počet argumentov. Každý argument vyžaduje identifikátor a dátový typ. Ohraničenia môžete použiť na definovanie množiny platných hodnôt, ktoré môžu byť predané ako argument.
- **Dátový typ návratovej hodnoty:** Povinný pre návratovú hodnotu, ale návratové hodnoty sú voliteľné. UML dovoľuje určiť iba typ, nie názov, čo je v súlade s väčšinou programovacích jazykov. Môžeme použiť iba jeden dátový typ návratu.
- **Viditeľnosť (+, –, #, ~):** Povinná pred generovaním kódu. Hodnoty viditeľnosti definuje programovací jazyk.
- **Operácia na úrovni triedy (podčiarknutá deklarácia operácie):** Voliteľné. Značí, že operácia je prístupná na úrovni triedy; vyžaduje referenciu inštancie (objektu).
- **Názov argumentu:** Povinný pre každý parameter, ale parametre sú voliteľné. Môžete použiť ľubovoľný počet argumentov.
- **Dátové typy argumentov:** Povinné pre každý parameter, ale parametre sú voliteľné.
- **Ohraničenia:** Voliteľné. Vo všeobecnosti, ohraničenia vyjadrujú pravidlá, ktoré musia byť vynútené pri vykonaní operácie. V prípade parametrov vyjadrujú kritériá, ktoré musia hodnoty splniť pred tým, ako môžu byť použité v operácii. Môžeme si ich predstaviť ako vstupné podmienky (angl. pre-conditions) na úrovni operácie.

7.2.4.Modelovanie oddelení triedy

Teraz potrebujeme toto všetko vložiť do symbolu triedy. Notácia triedy pozostáva z troch oddelení spomínaných v predchádzajúcom texte. Práve sme videli obsah druhého a tretieho oddelenia pre atribúty a operácie (v tomto poradí). Prvé oddelenie – oddelenie názvu – dáva triede identitu.

Obrázok zobrazuje UML symbol triedy s tromi oddeleniami a všetkými elementmi potrebnými na definovanie triedy pomocou notácie UML.



CLD 1: Kompletná špecifikácia triedy so všetkými tromi oddeleniami.

Oddelenie názvu

Oddelenie názvu (angl. name compartment) zaberá na obrázku vrchnú sekciu obdĺžnika triedy. Oddelenie názvu uchováva názov triedy, voliteľný stereotyp a voliteľné vlastnosti. Názov je umiestnený v strede oddelenia. Stereotyp (<<>>) sa zvykne používať na obmedzenie role triedy v modeli a umiestňuje sa na vrch oddelenia. Medzi bežné príklady stereotypov tried patrí stereotyp <<Factory>>, založený na návrhovom vzore Továreň a stereotyp <<Interface>> pre Java rozhrania alebo pre používateľské rozhrania.

Vlastnosti (angl. properties) používajú notáciu ohraničenia { } a sú umiestnené v pravom spodnom rohu oddelenia. Vlastnosti sú v podstate ohraničenia použité na objasnenie zámeru pri definovaní elementu modelu. Vlastnosti môžeme použiť na dokumentovanie stavu triedy vo vývoji alebo na označenie triedy ako *abstract* a *concrete*.

Oddelenie atribútov

Oddelenie atribútov (angl. attribute compartment) zaberá na obrázku strednú sekciu obdĺžnika triedy. Oddelenie atribútov jednoducho vymenováva špecifikácie atribútov triedy. Poradie atribútov nie je významné.

Oddelenie operácií

Oddelenie operácií (angl. operations compartment) zaberá na obrázku spodnú sekciu obdĺžnika triedy. Operácie sú jednoducho vymenované v oddelení operácií. Na poradí nezáleží. Ak sú viditeľné všetky oddelenia triedy, tak sa oddelenie operácií umiestňuje pod oddelenie názvu a pod oddelenie atribútov.

7.2.5.Asociácie

Asociácie medzi objektmi sú podobné ako asociácie medzi ľuďmi. Aby som s tebou mohol pracovať, musím s tebou komunikovať. K tomu potrebujem mať nejaký spôsob, ako ťa môžem kontaktovať,

napríklad pomocou telefónneho čísla alebo e-mailovej adresy. Ďalej je často nevyhnutné identifikovať prečo sme asociovaní, aby sme objasnili prečo sa zúčastňujeme a prečo nezúčastňujeme v určitých druhoch komunikácie. Napríklad, ak sme asociovaní pretože ty si programátor a ja správca databázy, pravdepodobne nebudeme v rámci svojich pracovných povinností diskutovať o výhodách zamestnancov.

Taktiež by pravdepodobne existovali určité obmedzenia kladené na naše interakcie:

- Chceli by sme obmedziť počet zúčastnených vo vzťahu, aby sme zabezpečili efektívnosť.
- Chceli by sme skontrolovať kvalifikáciu zamestnancov, aby sme zabezpečili, že máme správnych zúčastnených.
- Chceli by sme definovať roly zúčastnených tak, aby každý vedel ako sa má správať.

Všetky tieto požiadavky sú rovnako aplikovateľné aj na objekty. UML poskytuje notácie na určenie všetkých týchto požiadaviek.

Názov asociácie

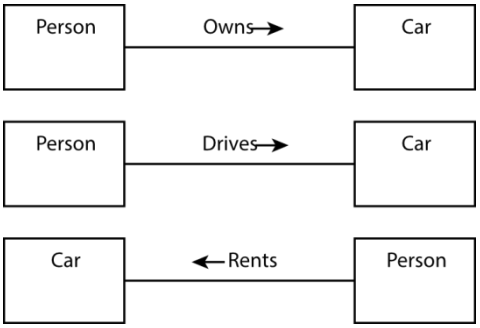
Účel asociácie môžeme vyjadriť *názvom*, slovesom alebo slovesnou frázou, ktorý opisuje ako objekty jedného typu (triedy) súvisia s objektmi iného typu (triedy). Napríklad, osoba vlastní auto, osoba riadi auto, osoba prenajíma auto. Hoci zúčastnení sú vo všetkých asociáciách tí istí, účel každej asociácie je unikátny a ako také implikujú rôzne pravidlá a interakcie.

Pre nakreslenie UML asociácií pre tieto tri príklady musíte vychádzať zo štyroch základných elementov.

- *Zúčastnené triedy*, osoba a auto.
- *Asociácie*, reprezentované čiarou medzi týmito dvomi triedami (veľmi technické, že?).
- *Názov* asociácie reprezentovaný slovesom alebo slovesnou frázou na čiare asociácie.
- *Smer*, v ktorom je potrebné čítať názov (indikovanie smeru je voliteľné).

Prvé dva príklady na obrázku sú modelované presne tak, ako sme ich opísali v texte – Person owns Car a Person drives Car. Ak tieto dve tvrdenia sú pravdivé, potom je pravdivý aj opak – Car is owned by Person a Car is driven by Person. Asociácie môžeme čítať v oboch smeroch, treba meniť význam názvu asociácie z aktívneho do pasívneho tvaru.

Avšak v treťom príklade na obrázku by názov asociácie nedával zmysel, ak by sme ho prečítali typicky zľava doprava – Car rents Person. Toto je prípad, kedy je indikátor smeru (angl. direction indicator) veľmi vhodný, dokonca vyžadovaný, aby asociácia dávala zmysel – Person rents Car.

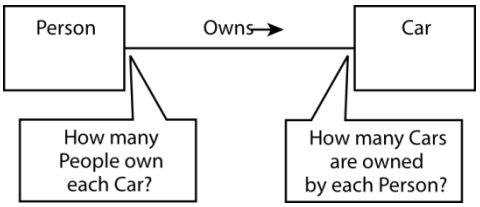


CLD 2: Notácia smeru čítania názvov asociácií.

Násobnosť asociácie

UML umožňuje riešiť niektoré ďalšie dôležité otázky o asociáciách: „Koľko áut môže osoba vlastniť?“ „Koľko áut si môžu prenajať?“ „Koľko ľudí môže riadiť dané auto?“ Asociácie definujú pravidlá, *ako* môžu objekty rôznych tried navzájom súvisieť. Ako teda presne špecifikujeme *koľko* objektov sa môže zúčastniť v danom vzťahu?

Násobnosť (angl. multiplicity) je UML názov pre pravidlo, ktoré definuje počet zúčastnených objektov. Hodnota násobnosti *musí* byť priradená každej triede zúčastnenej v asociácii. Musíme sa pýtať dve otázky, ako uvádza obrázok.



CLD 3: Priraďovanie násobnosti každému koncu asociácie.

Odpoveď na každú otázku sa umiestňuje vedľa triedy, ktorá opisuje objekty, ktorých počet zisťujeme. Odpoveď na „Koľko ľudí...“ umiestnime na koniec asociácie vedľa triedy Person. Odpoveď na „Koľko áut...?“ umiestnime na koniec asociácie vedľa triedy Car.

Násobnosť sa vyjadruje viacerými spôsobmi. Najčastejšie sa používa rozsah definujúci minimálny povolený počet objektov a maximálny povolený počet objektov vo formáte

minimum . . maximum

Minimum a maximum musia byť celočíselné hodnoty. V reálnych situáciách však niekedy nepoznáme horný limit alebo jednoducho žiadny horný limit neexistuje. UML navrhuje použiť hviezdičku na vyjadrenie, že *neexistuje horný limit*. Ak sa hviezdička použije samostatne, znamená to, že minimum je nula *a* neexistuje horný limit, *buď nula alebo viac*.

Taktiež sa môžeme stretnúť so situáciou, kedy nie je vhodné použiť rozsah. Ak by sme museli definovať koľko valcov je v motoroch, ktoré opravujeme, možno by sme povedali „Pracujem iba so 4-, 6- a 8-valcovými motormi.“ Pre tieto situácie UML navrhuje zoznam čísel oddelených čiarkou (napríklad 4,6,8).

Keď je hodnota minima a maxima rovnaká, môžeme zjednodušiť notáciu použitím iba jednej hodnoty. Takže namiesto písania 4..4, môžeme jednoducho napísať 4. Toto je síce pekná skratka, ale

treba si uvedomiť, že najbežnejší prípad, kedy sa táto skratka používa, je v prípade násobnosti 1..1. Nanešťastie, táto skratka povzbudzuje ľudí prehliadnuť alebo ignorovať možnosť, že minimum je nula, t.j. vzťah je voliteľný.

Použijeme príklad s autom a motorom. Každé auto má motor. Ale, čo tak autá na montážnej linke? Počas prvých *n* fáz montáže v aute nie je žiadny motor. V tomto prípade by násobnosť mala byť 0..1, aby umožnila existenciu auta pred inštaláciou motora. Reálne existuje len málo inštancií, kedy by mala byť minimálna násobnosť 1. Podľa praktických skúseností by malo byť minimum nastavené na 0, pokým nemáme pozitívny dôkaz, že jeden objekt nemôže existovať bez ďalšieho objektu.

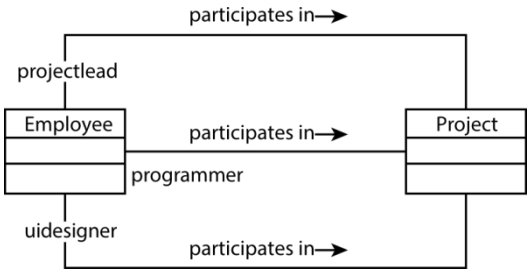
Tu sú zosumarizované možnosti špecifikácie násobnosti nasledované niekoľkými príkladmi.

- Hodnoty oddelené dvomi bodkami (..) znamenajú rozsah. Napríklad, 1..3 znamená medzi 1 a 3 vrátane; 5..10 znamená medzi 5 a 10 vrátane.
- Hodnoty oddelené čiarkami znamenajú zoznam s vymenovanými hodnotami. Napríklad, 4,6,8 znamená, že môžete mať v asociácii 4 objekty, 6 objektov alebo 8 objektov tohto typu.
- Hviezdička (*) použitá samostatne znamená nula alebo viac, bez dolného alebo horného limitu.
- Hviezdička (*) použitá v rozsahu (1..*) znamená bez horného limitu – musíte mať aspoň jeden ale inak môžete mať toľko koľko chcete.

Asociačné roly

Niekedy je trochu náročné určiť názov asociácie. Napríklad, aké anglické slovo by sme mohli použiť pre názov asociácie medzi rodičmi a deťmi? UML poskytuje alternatívu, ktorá môže byť použitá v mieste názvu alebo spolu s ním, aby pomohla čo najviac ujasniť dôvod pre asociáciu. Táto alternatíva sa nazýva *rola*, pretože opisuje *ako sa objekt zúčastňuje* v asociácii.

Napríklad, mnoho zamestnancov prispieva do projektu. Ale z vlastnej skúsenosti viete, že sa zúčastňujú rôznymi spôsobmi. Obrázok zobrazuje ako môžete nakresliť viaceré asociácie a označiť ich na rozlíšenie typov účasti. Každá rola sa umiestňuje na koniec asociácie vedľa typu objektu, ktorý hrá rolu. Môžete ich použiť na jednom, na oboch alebo na žiadnom z koncov každej asociácie.



CLD 4: Názvy rol na asociácii.

Ohraničenia asociácií

Pozrime sa na obrázok, v ktorom nie sú špecifikované žiadne ohraničenia.

CLD 5: Asociácia bez ohraničení.

Je naozaj pravda, že *ľubovoľná* osoba môže riadiť auto? Iba ľudia s platnými vodičskými preukazmi sú oprávnení legálne šoférovať. Túto informáciu môžete pridať do modelu použitím páru zložených zátvoriek {} obsahujúcich text, ktorý opisuje pravidlo, ktoré chcete vynútiť (napríklad, {must have valid driver's license}). Na obrázku je ohraničenie umiestnené na konci asociácie blízko triedy Person – typu objektu, ktorý musí vyhovovať pravidlu pred tým, ako sa môže zúčastniť vzťahu.

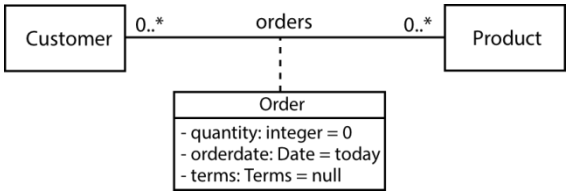
CLD 6: Asociácia s ohraničením na účasť objektu triedy Person.

Ohraničenia sa môžu nachádzať na oboch koncoch, na jednom z koncov alebo na žiadnom z koncov asociácie. Naozaj závisí od konkrétneho problému, ktorý práve opisujete. Veľmi sa nestarajte o umiestnenie. Ohraničenie môžete umiestniť hocikam blízko konca asociácie.

Ale čo ak existuje viac ohraničení? Jednoduché. Iba pridajte viac textu medzi dve zátvorky. *Nevytvárajte ďalšie zátvorky*.

Väzobná trieda

Väzobná trieda (angl. association class) zapuzdruje *informáciu o asociácii*. Na obrázku vidíme, že zákazníci si objednávajú produkty. Ale keď si zákazníci objednajú produkty, zvyčajne potrebujete vedieť viac ako len to, napríklad, kedy si objednali produkty? Koľko si ich objednali? Aké boli podmienky predaja? Všetky odpovede na tieto otázky sú jednoducho dáta. Všetky dáta v objektovo-orientovanom systéme musia byť obsiahnuté (zapuzdrené) v objekte. Pre definovanie každého typu objektu musí existovať trieda. Takže všetky tieto dáta definujte v triede. Potom na zobrazenie, že dáta opisujú asociáciu pripojte novú triedu k asociácii *prerušovanou čiarou*.

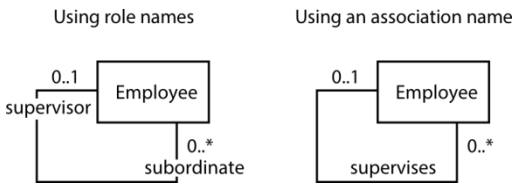


CLD 7: Notácia väzobnej triedy.

Reflexívna asociácia

Reflexívna asociácia (angl. reflexive association) znamená, že objekty rovnakej triedy môžu navzájom súvisieť. Celá notácia asociácie, ktorú sme doteraz uviedli, ostáva úplne rovnaká až na to, že oba konce čiary asociácie ukazujú na tú istú triedu. Odtiaľto pochádza názov reflexívnej asociácie. Čiara asociácie opúšťa triedu a *odráža sa späť na* tú istú triedu.

Obidva príklady na obrázku sú ekvivalentné. Jedený rozdiel je v tom, že jeden používa roly a druhý používa názov asociácie.



CLD 8: Dva spôsoby ako modelovať reflexívnu asociáciu.

Reflexívna asociácia je bežný spôsob vyjadrenia hierarchie. Príklad na obrázku modeluje hierarchickú štruktúru zodpovednosti. Mohol by som použiť rovnakú techniku pre spoločnosti vlastnené inými spoločnosťami.

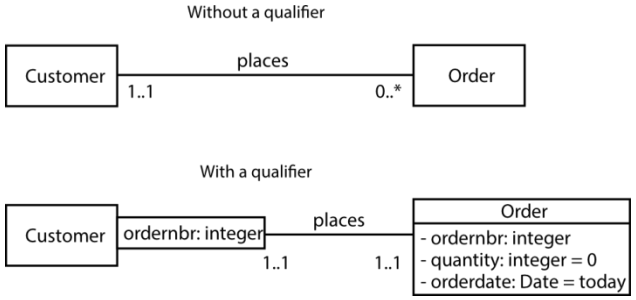
Kvalifikovaná asociácia

Kvalifikované asociácie (angl. qualified associations) poskytujú približne rovnakú funkcionalitu ako indexy, ale notácia je mierne zmenená. Pre indikáciu, že zákazník môže vyhľadať objednávku použitím atribútu *ordernbr*, umiestnime názov atribútu *ordernbr* spolu s dátovým typom do obdĺžnika na koniec asociácie pri triede *Customer*. Asociácia je vytlačená na okraj obdĺžnika, ale zvyšok notácie asociácie ostáva nezmenený.

Umiestnenie kvalifikátora býva občas mätúce. Najlepší spôsob na zapamätanie je rozmýšľať nad tým takto (vzťahuje sa na obrázok):

> **„Zákazník *používa* ordernbr na vyhľadanie objednávky.“**
> alebo
> **„Jeden typ objektu používa kvalifikátor pre prístup k druhému (kvalifikovanému) typu objektu.“**

Kvalifikátor sa umiestňuje vedľa triedy objektov, ktorá *použije* hodnotu na vykonanie vyhľadania. Nie je to úplne intuitívne, ale funguje to.



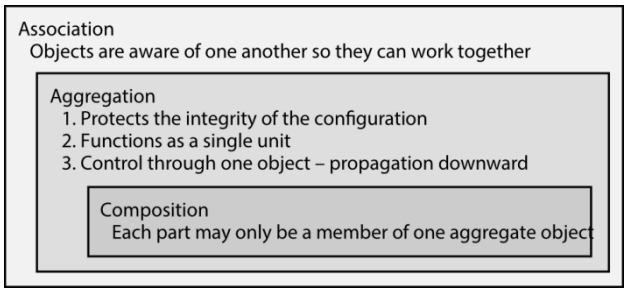
CLD 9: Kvalifikovaná asociácia.

Kvalifikátory redukujú násobnosť rovnakým spôsobom, ako keby sme použili indexy v databáze na skrátenie času potrebného na vyhľadanie špecifického riadku alebo podmnožiny riadkov. Napríklad, všimnime si, ako sa na obrázku zmenila násobnosť konca asociácie pri triede Order z 0..* na 1..1. Táto zmena nastala, pretože kvalifikátor poskytol unikátny kľúč pre objekty triedy Order. Pred tým, ako sa zaviedol kvalifikátor, by navigácia cez asociáciu vyústila do zoznamu všetkých objednávok asociovaných s daným zákazníkom, pretože vzťah so zákazníkom bol jedinou dostupnou referenciou pre výber objednávok.

7.2.6.Modelovanie zoskupenia a kompozície

Obrázok opisuje vzťahy medzi konceptmi asociácie, zoskupenia a kompozície.

- Každé zoskupenie je *typom asociácie*. Takže každé zoskupenie má všetky vlastnosti asociácie plus nejaké vlastné pravidlá.
- Každá kompozícia je *formou zoskupenia*. Takže každá kompozícia má všetky vlastnosti zoskupenia plus nejaké vlastné pravidlá.



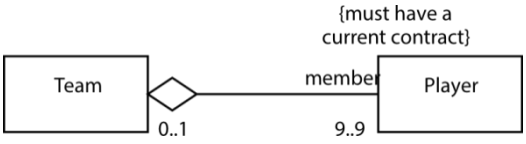
CLD 10: Vzťah medzi asociáciou, zoskupením a kompozíciou.

Elementy zoskupenia

Zoskupenie (agregácia, angl. aggregation) je špeciálny typ asociácie používaný na indikovanie, že zúčastnené objekty nie sú iba nezávislé objekty, ktoré o sebe vedia. Namiesto toho sú *zostavené* alebo *konfigurované* (angl. assembled or configured) dokopy aby vytvorili nový zložitejší objekt. Napríklad, niekoľko rôznych častí zostavených do auta, člna alebo lietadla. Mohli by ste dokonca vytvoriť logické zoskupenie ako tím, kde časti nie sú fyzicky navzájom spojené, ale stále pracujú ako samostatný celok.

Pre modelovanie zoskupenia v diagrame tried:

1. Nakreslite asociáciu (čiaru) medzi triedou, ktorá reprezentuje člena a triedou, ktorá reprezentuje zostavu alebo zoskupenie. Na obrázku by to znamenalo čiaru medzi triedou Team a triedou Player.
2. Nakreslite na koniec asociácie *kosoštvorec*, ktorý je pripojený k zostavujúcej alebo zoskupujúcej triede. Na obrázku je kosoštvorec vedľa triedy Team, ktorá reprezentuje skupinu hráčov.
3. Priraďte vhodné násobnosti na *oba konce* asociácie a pridajte akékoľvek role a/alebo ohraničenia, ktoré by mohli byť potrebné na definovanie pravidiel vzťahu. Obrázok zobrazuje, že hráč môže byť členom najviac jedného tímu, ale hráč nemusí byť v tíme celý čas (0..1). Tím vždy pozostáva z presne 9 hráčov (9..9 alebo len 9). Hráč je považovaný za *člena* (názov roly) tímu. Každý hráč je ohraničený faktom, že musí mať aktuálny kontrakt na to, aby mohol byť členom tímu.



CLD 11: Reprezentácia vzťahu zoskupenia v UML.

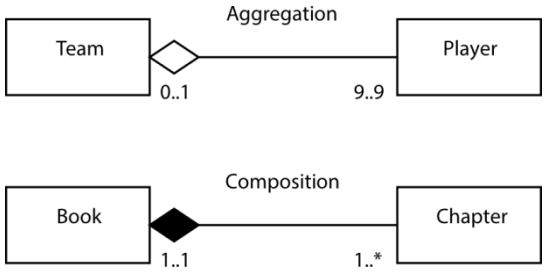
Čo robí zoskupenie jedinečným? Alebo dôležitejšie, prečo je zoskupenie užitočné? Zoskupenie opisuje skupinu objektov spôsobom, ktorý mení spôsob interakcie s nimi. Cieľom tohto konceptu je chrániť integritu konfigurácie objektov dvomi špecifickými spôsobmi.

1. Zoskupenie definuje jeden bod riadenia v objekte, ktorý reprezentuje zostavu. Týmto je zabezpečené, že nezáleží na tom, čo chcú robiť ostatné objekty členom zostavy, riadiaci objekt má posledné slovo a určuje, či akcie povolí. Táto riadiaca úloha môže byť pridelená na rôznych úrovniach v hierarchii zoskupenia. Napríklad, motor môže riadiť jeho časti, ale auto riadi motor.
2. Keď sa stanoví inštrukcia, ktorá môže ovplyvniť celú kolekciu objektov, riadiaci objekt určí, ako členovia zareagujú. V každom prípade sa javí, že zostava funguje ako jeden objekt. Keď stlačím plynový pedál, čím poviem autu, že chcem zrýchliť, zrýchľuje celá zostava auta (s tisíckou súčastí), nie len plynový pedál.

Elementy kompozície

Kompozícia (angl. composition) sa používa pre zoskupenia, kde životnosť časti, závisí od životnosti zoskupenia. Zoskupenie má kontrolu nad vytváraním a deštrukciou časti. Inými slovami, členský objekt nemôže existovať mimo zoskupenia. Túto silnejšiu formu zoskupenia nakreslíme jednoducho vyplnením kosoštvorca zoskupenia (čiernou farbou).

Na obrázku tím používa zoskupenie (nevyplnený kosoštvorec). Hráči sú zoskupení do tímu. Ale ak sa tím rozpadne, hráči žijú ďalej (samozrejme v závislosti od toho, aký dobrý výkon podali). V príklade s knihou sa používa kompozícia, vyplnený kosoštvorec. Kniha pozostáva (je skomponovaná) z kapitol. Kapitoly by však nepokračovali v existencií samostatne mimo knihy. Prestali by existovať spolu s knihou.



CLD 12: Reprezentácia vzťahu kompozícia v UML.

Všimnite si, ako násobnosť poskytuje niektoré záchytné body na rozlíšenie medzi zoskupením a kompozíciou. V príklade s tímom na obrázku každý hráč môže existovať nezávisle od tímu. Príklad s knihou hovorí, že kapitola musí byť asociovaná s práve jednou a len jednou knihou (1..1). Toto mi hovorí, že kapitola nemôže existovať nezávisle od knihy, takže musí ísť o vzťah kompozície.

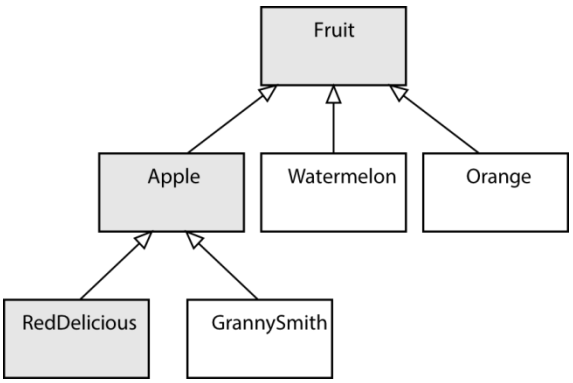
7.2.7.Modelovanie zovšeobecnenia

Zovšeobecnenie je proces organizovania vlastností množiny objektov, ktoré majú rovnaký účel. Ľudia používajú tento proces rutinne na organizovanie veľkého množstva informácií. Prejdite sa po potravinách a nájdete jedlá umiestnené v oddeleniach obchodu v závislosti od ich vlastností. Suché tovary sú umiestnené v jednom oddelení, ovocie a zelenina v druhom a mäso v ďalšom. Všetky tieto položky sú jedlá, avšak predstavujú rôzne druhy jedál alebo typy jedál. Slová ako *druh* alebo *typ* sú

často používané na opísanie vzťahu zovšeobecnenia medzi triedami (napríklad, jablko je druh ovocia, ktoré zase druhom jedla a tak ďalej).

Tento druh vzťahu sa taktiež nazýva *dedenie*. Mnohokrát sú pojmy *zovšeobecnenie* a *dedenie* používané ako synonymá. Ak je jablko druh ovocia, potom *dedí* všetky vlastnosti ovocia. Podobne, jablko je *špecializáciou* ovocia, pretože dedí všetky všeobecné vlastnosti ovocia a pridáva určité unikátne vlastnosti, ktoré sú typické iba pre jablká. Naopak, mohol by som povedať, že koncept ovocia je *zovšeobecnením* faktov, ktoré sú pravdivé pre melóny, jablká, broskyne a všetky typy objektov v skupine.

Zovšeobecnenie *nie* je asociácia. Zopakujem to, aby ste na to nezabudli. Zovšeobecnenie *nie* je asociácia. V skutočnosti asociácia a zovšeobecnenie sú v metamodeli UML považované za dva rôzne elementy modelu. Asociácie definujú pravidlá o tom, ako môžu *objekty* medzi sebou súvisieť. Zovšeobecnenie dáva do spojitosti *triedy*, ktoré obsahujú podmnožinu elementov potrebných na definovanie typu objektu. Inštancovanie všetkých podmnožín elementov z každej triedy v jednej ceste dedenia vyústi do jedného objektu. V ilustrácii s ovocím na obrázku by som pre vytvorenie jablka Red Delicious potreboval skombinovať triedu RedDelicious, triedu Apple a triedu Fruit, aby som dostal všetky atribúty a operácie, ktoré definujú jablko Red Delicious. Zo skombinovanej triedy by som vytvoril (inštancoval) objekt triedy RedDelicious (jablko).



CLD 13: Modelovanie zovšeobecnenia.

Pre vyjadrenie rovnakej veci slovami, by som povedal „‘Red Delicious‘ je typ jablka a jablko je typ ovocia.“ Z tohto dôvodu sa niekedy zovšeobecnenie nazýva vzťah „*je*“ (t.j. každý objekt triedy RedDelicious „je“ objekt triedy jablko a každý objekt triedy jablko „je“ objekt triedy ovocie).

Tento unikátny vzťah medzi triedami vo všeobecnosti vyvoláva zaujímavý problém. Viditeľnosť určuje, ktoré objekty môžu vidieť atribút alebo operáciu. Štandardne sú atribúty nastavované na súkromnú viditeľnosť, aby ich mohli vidieť iba objekty tej istej triedy. Čo by sa ale stalo triede RedDelicious na obrázku, ak by boli atribúty triedy Apple nastavené na súkromnú viditeľnosť? RedDelicious by k nim nemal prístup, takže dedenie sa v podstate obíde. Ďalší typ viditeľnosti, nazývaný *chránená viditeľnosť*, je definovaný, aby riešil práve túto situáciu. Chránená viditeľnosť umožňuje vidieť elementy iba objektom tej istej triedy alebo *podtriedy*.

Elementy zovšeobecnenia

Pretože vzťah zovšeobecnenie (tiež nazývaný *dedenie*) *nie* je formou asociácie, nie je potrebné určovať násobnosti, roly a ohraničenia. Tieto elementy sú jednoducho irelevantné.

Pre nakreslenie vzťahu zovšeobecnenie najprv potrebujeme definovať nadtriedu, podtriedu, abstraktnú triedu, konkrétnu triedu a *diskriminátor*. Nadtrieda (angl. superclass) je trieda, ktorá obsahuje kombináciu atribútov, operácií a asociácií, ktoré sú spoločné pre dva alebo viaceré typy objektov, ktoré majú rovnaký účel. Fruit a Apple sú príklady nadtriedy. Pojem *nadtrieda* odráža koncept nadmnožiny. Nadmnožina alebo nadtrieda v tomto prípade obsahuje vlastnosti, ktoré sú spoločné pre každý objekt v množine.

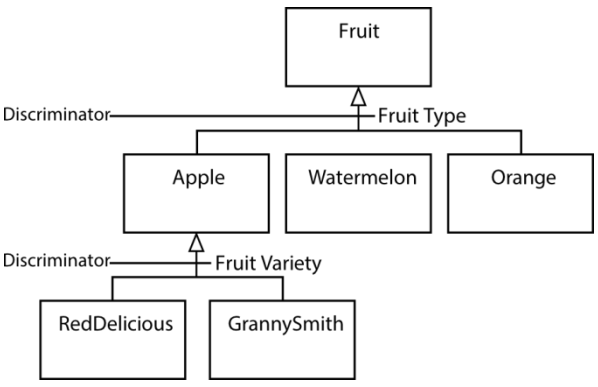
Podtrieda (angl. subclass) je trieda, ktorá obsahuje kombináciu atribútov, operácií a asociácií, ktoré sú jedinečné pre typ objektu, ktorý je čiastočne definovaný nadtriedou. Na obrázku sú triedy Apple, Watermelon, Orange, RedDelicious a GrannySmith všetko príklady podtried. Všimnite si, že trieda môže byť zároveň podtriedou aj nadtriedou.

Pojem *podtrieda* odráža koncept podmnožiny. Podmnožina alebo podtrieda obsahuje jedinečnú množinu vlastností iba pre určité objekty z množiny. Trieda GrannySmith na obrázku by obsahovala iba vlastnosti, ktoré sú jedinečné pre jablká GrannySmith. Objekt triedy GrannySmith by mal zvyšné informácie o vlastnostiach, ktoré má spoločné so všetkými jablkami z nadtriedy Apple a všetkými vlastnosťami, ktoré má spoločné so všetkým ovocím z nadtriedy Fruit. Inými slovami, na generovanie jedného objektu reprezentujúceho jablko GrannySmith sú vlastne potrebné tri triedy.

Abstraktná trieda (angl. abstract class) je trieda, ktorá nemôže vytvárať objekty (nemôže byť inštancovaná). Ľubovoľná nadtrieda, ktorá definuje aspoň jednu operáciu, ktorá nemá metódu sa nazýva abstraktná alebo trieda bez úplnej definície. Iba nadtrieda môže byť abstraktná.

Konkrétna trieda (angl. concrete class) je trieda, ktorá má metódu pre každú operáciu, takže môže vytvárať objekty. Metódy môžu byť definované v triede zdedenej z nadtriedy. Všetky triedy na spodku hierarchie dedenia *musia* byť konkrétne. Ľubovoľná nadtrieda *môže* byť konkrétna.

Diskriminátor (angl. discriminator) je atribút alebo pravidlo, ktoré opisuje ako sa rozhodnem identifikovať množinu podtried určitej nadtriedy. Keby som chcel organizovať informácie o typoch áut, mohol by som diskriminovať na základe rozsahu ceny, výrobcu, objemu motora, typu paliva, použitia alebo akýchkoľvek iných kritérií. Diskriminátor, ktorý si vyberiem závisí od problému, ktorý sa snažím vyriešiť. Na obrázku používam veľmi bežnú praktiku. Identifikujem množinu preddefinovaných typov (t.j. typov ovocia a druhov v rámci typov). Inokedy použijem vlastnosť samotných objektov ako ich veľkosť, cenový rozsah, kapacita alebo vek.



CLD 14: Modelovanie zovšeobecnenia s diskriminátormi.

Kompozícia triedy odhaľuje možné diskriminujúce vlastnosti. Triedy definujú vlastnosti objektov ako sú atribúty, operácie a asociácia. Tieto patria medzi prvé tri diskriminujúce vlastnosti. Ak objekty triedy obsahujú spoločné atribúty, ako vek a adresa, môžu byť v rovnakej podskupine. Avšak, objekty môžu obsahovať rovnaký atribút (ako vek), ale povolené hodnoty pre vek sú v niektorých objektoch iné, ako sú povolené v iných objektoch. Napríklad, každá osoba má priradený vek. Avšak, neplnoletí by (v niektorých štátoch) mali hodnotu veku menšiu ako 21 a dospelí by mali vek väčší ako 20.

Rovnaký koncept sa aplikuje aj na operácie. Objekty môžu mať rovnaké operácie, t.j. rovnaké rozhranie, ako „zrýchľovať“. Ale rôzne objekty môžu implementovať toto rozhranie (veľmi) odlišným spôsobom. Auto v porovnaní s raketou zrýchľuje úplne inak. Dokonca rôzne autá zrýchľujú použitím rôznych kombinácií častí a palív.

Aby som to zhrnul, existuje aspoň päť objektívnych kritérií, ktoré môžem použiť na diskrimináciu medzi objektmi z rámci tej istej triedy (nadtriedy):

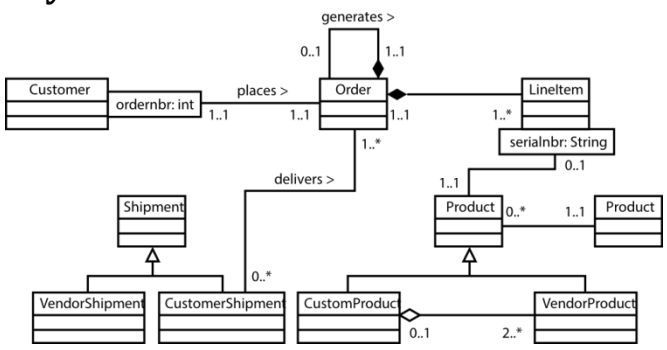
- typ atribútu,
- povolené hodnoty atribútu,
- operácia (rozhranie),
- metóda (implementácia),
- asociácie.

V skutočnosti existujú dva spôsoby ako nakresliť zovšeobecnenie. Obrázok nám ukázal samostatné čiary z každej podtriedy do jej nadriedy. Obrázok zlúčil čiary zo všetkých podtried nadtriedy.
V obidvoch formách nakreslite trojuholník na konci čiary zovšeobecnenia pri nadtriede (ktorý ukazuje na natriedu). (Všimnite si, že kreslíte trojuholník, nie šípku.) Spojte druhý koniec zovšeobecnenia s podtriedou. Na čiaru zovšeobecnenia pridajte diskriminátor vo forme jednoduchého textu.

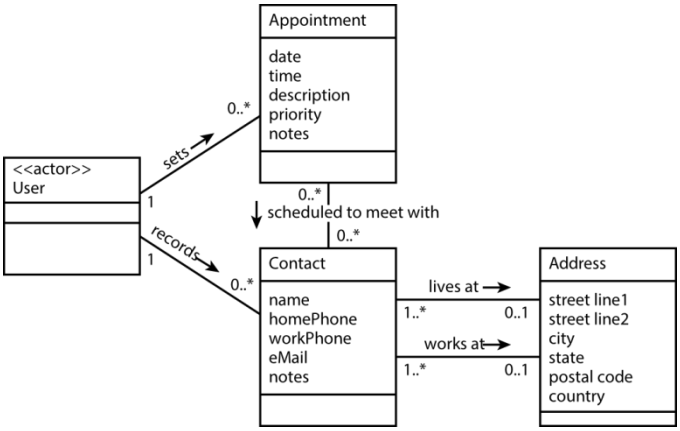
7.2.8.Postup tvorby diagramu tried

1. Identifikujte triedy, pomenujte ich a definujte ich tak, aby ste vedeli, prečo sú časťou modelu.
2. Identifikujte, pomenujte a definujte asociácie medzi dvojicami tried. Tiež si všímajte reflexívne asociácie. Tam kde je to potrebné, priraďte násobnosť a ohraničenia. Keď je pomenovanie asociácie zložité, skúste pomenovať roly.
3. Zhodnoťte každú asociáciu, aby ste určili, či by nemala byť definovaná ako zoskupenie. Ak je to zoskupenie, nemohla by to byť kompozícia?
4. Zhodnoťte každú triedu pre možnú špecializáciu alebo zovšeobecnenie.

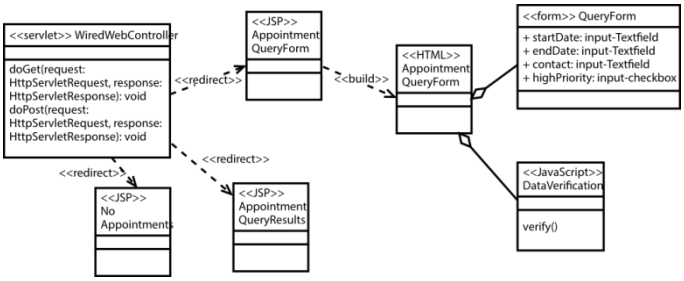
7.2.9.Príklady



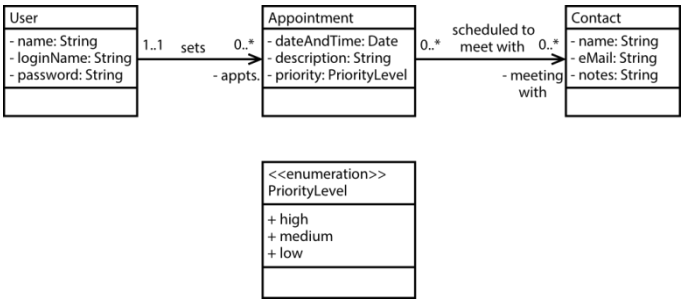
CLD 15: Diagram tried.



CLD 16: Diagram tried – Friendly Reminder appointments.



CLD 17: Diagram tried s Web technológiami.



CLD 18: Diagram tried pre modelovanie XML.

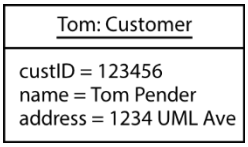
7.3. Diagram objektov

Diagram objektov (angl. Object diagram) modeluje *fakty* o špecifických entitách, zatiaľ čo diagram tried modeluje *pravidlá* pre typy entít. Objekty sú reálne veci, ako ty a ja, táto kniha a stolička na ktorej sedíš. Takže diagram objektov môže, napríklad, reprezentovať *fakt*, že vlastníš túto kópiu knihy *Víkendový rýchlokurz UML* . Naproti tomu by diagram tried opisoval *pravidlá*, že ľudia môžu vlastniť knihy.

7.3.1. Elementy notácie diagramu objektov

Diagram objektov pozostáva iba z dvoch elementov: objekty a prepojenia (angl. links). Už viete, že objekt je reálna entita vytvorená z triedy (z definície typu objektu). Rovnakým spôsobom sa vytvára prepojenie z asociácie (z definície typu vzťahu). Prepojenie reprezentuje vzťah medzi dvoma objektmi. Asociácia definuje typ vzťahu a pravidlá, ktoré ju ovládajú.

Obrázok zobrazuje objekt nazvaný Tom. Podobne ako notácia triedy má aj notácia objektu oddelenie názvu na vrchu obdĺžnika. Názov pozostáva z názvu objektu *a* názvu triedy, ktorej objekt zodpovedá: „Customer“. Toto pomáha rozlíšiť objekt Tom triedy Customer od objektu Tom triedy Employee. Táto notácia vám taktiež umožňuje modelovať príklad alebo testovací prípad, v ktorom sa zúčastňuje mnoho objektov rovnakej triedy (napríklad, jeden zamestnanec dohliada na iného zamestnanca).



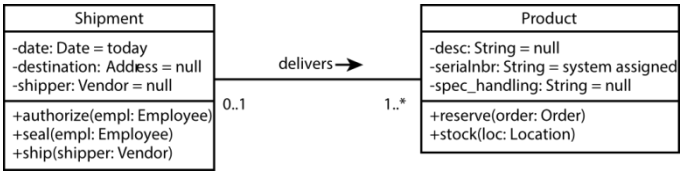
OBD 1: UML notácia objektu.

Pre úplnú definíciu názvu objektu použite formát *názov objektu : názov triedy*. Potom celý tento výraz podčiarknite. Môžete taktiež použiť skrátenú formu, *: názov triedy* bez názvu objektu. Táto forma sa nazýva *anonymný objekt* (angl. anonymous object). Používa sa, keď chcete nakresliť príklad, kde vôbec nezáleží, ktorý špecifický objekt v príklade vystupuje, pretože ľubovoľný objekt tej istej triedy by sa správal úplne rovnako.

Druhé oddelenie objektu obsahuje fakty o atribútoch. Každý atribút je pomenovaný a má priradenú hodnotu. Na obrázku vidíte name = Tom Pender. Objekt je skutočný. Existuje. Tak môže mať priradené hodnoty ku každému atribútu. Dokonca aj prázdna hodnota alebo nulová hodnota je hodnota, ktorá odlišuje stav objektu, od iných možných stavov. Všimnite si, ako sa to líši od oddelenia atribútov triedy. Trieda obsahovala definície atribútov a neobsahovala hodnoty. Opakujem, že trieda je množina *pravidiel*, zatiaľ čo objekt je množina *faktov*. Trieda hovorí, že objekt triedy Employee *môže* mať atribút name, ktorý obsahuje reťazec 20 znakov. Objekt hovorí, že atribút name obsahuje hodnotu „Tom Pender“.

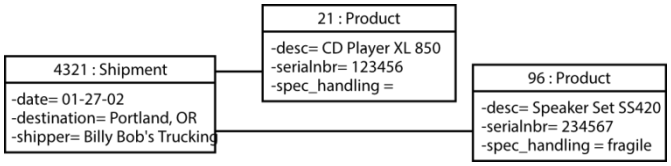
7.3.2.Porovnanie notácií diagramu objektov a diagramu tried

Obrázok obsahuje diagram tried zobrazujúci pravidlá týkajúce sa tried Shipment a Product a vzťahu medzi nimi. Diagram tried definuje atribúty, ktoré musia byť použité na definíciu každého typu objektu a správanie, ktoré musí podporovať každý typ objektu.



OBD 2: UML notácia triedy pre Shipment a Product.

Diagram objektov na obrázku zobrazuje, že objekt 4321 triedy Shipment má dva produkty. Každý atribút týchto troch objektov má priradenú hodnotu. Všimnite si, že atribút spec_handling objektu 21 má priradenú prázdnu hodnotu. Aj toto môže byť platná hodnota. Definícia triedy určuje platné hodnoty, ktoré môžu byť priradené atribútom objektu.



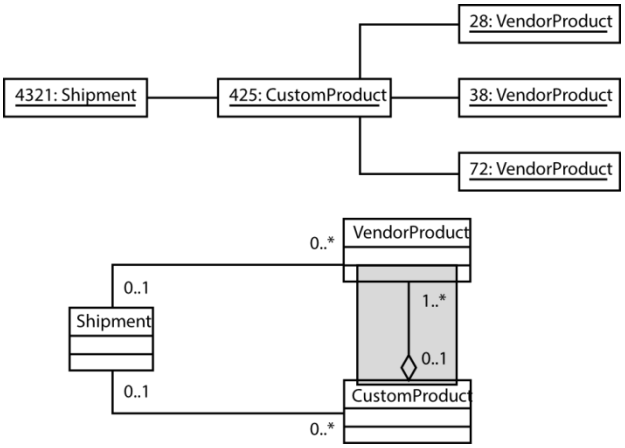
OBD 3: UML notácia objektu pre Shipment a Product.

Všimnite si, čo chýba v notácii objektu, čo bolo vyžadované v každej triede. Tretie oddelenie, obsahujúce operácie, bolo vynechané. Prečo má byť vynechané z objektu, keď trieda určuje pravidlá, ktoré objekty musia spĺňať? Atribúty sa zahŕňajú, pretože každý objekt môže potenciálne vlastniť iné hodnoty atribútov definovaných triedou. Ale trieda definuje operáciu, ktorá *ne*má viaceré interpretácie alebo hodnoty. Každý objekt rovnakej triedy vlastní *rovnaké operácie*. Opätovné modelovanie operácií by do diagramu objektov vnieslo redundanciu bez pridania novej informácie. Preto v diagrame objektov operácie vynechajte.

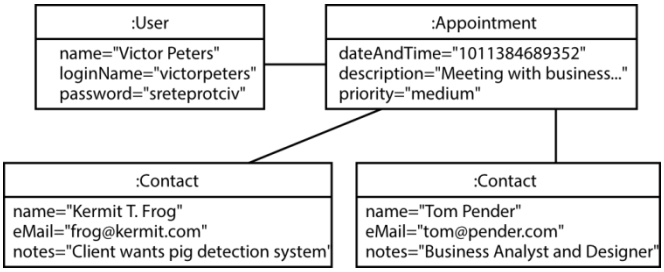
Po príkladoch, ktoré ilustrovali rozdiely medzi diagramom tried a diagramom objektov sa teraz môžete pozrieť na tabuľku, ktorá obsahuje porovnanie ich vlastností. Praktická znalosť vzťahu medzi týmito dvoma typmi diagramov vám pomôže pochopiť ako ich použiť na uľahčenie analýzy a testovanie výsledkov vášho úsilia vynaloženého na analýzu a návrh.

Diagram tried	**Diagram objektov**
Trieda má tri oddelenia: oddelenie názvu, atribútov a operácií.	Objekt má iba dve oddelenia: oddelenie názvu a atribútov.
Oddelenie názvu obsahuje iba názov triedy.	Názov objektu má nasledovný formát: názov_objektu : názov_triedy (napríklad 1234:Order). S touto notáciou sa stretnete aj v iných diagramoch, ktoré modelujú objekty. Niekedy sa názov objektu vynecháva a použije sa iba dvojbodka a názov triedy. Toto sa nazýva anonymný objekt.
Oddelenie atribútov triedy definuje vlastnosti atribútov.	Objekt definuje iba aktuálne hodnoty každého atribútu modelovaného príkladu alebo testu.
Operácie sú v triede vymenované.	Operácie v objekte nie sú zahrnuté, pretože by boli identické v každom objekte rovnakého typu.
Triedy sú spojené *asociáciou* s názvom, násobnosťou, ohraničeniami a rolami. Triedy reprezentujú „klasifikáciu“ objektov, takže je nevyhnutné špecifikovať, koľko sa ich môže zúčastniť v asociácii.	Objekty sú spojené *prepojením*, ktoré má názov a nemá násobnosť. Objekty reprezentujú jednotlivé entity. Všetky prepojenia sú 1:1, takže násobnosť je irelevantná. Prepojenia môžu obsahovať roly.

7.3.3.Príklady



OBD 4: Diagram objektov (hore) pre daný diagram tried (dole).



OBD 5: Diagram objektov.

7.4. Diagram činností

Diagram činností (diagram aktivít, angl. Activity diagram) je časťou funkcionálneho pohľadu, pretože opisuje logické procesy alebo funkcie implementované použitím kódu. Každý proces opisuje sekvenciu úloh a rozhodnutí, ktoré určujú kedy a ako budú jednotlivé úlohy vykonávané. Diagramy činností sa používajú predovšetkým na modelovanie pracovných tokov, prípadov použitia a operácií.

7.4.1.Modelovanie pracovných tokov a prípadov použitia

Keď modelujete prípad použitia, snažíte sa porozumieť cieľom, ktoré musí systém dosiahnuť na to, aby bol úspešný. Diagram činností používajte pre sledovanie používateľa v rámci procedúry a zaznamenávanie rozhodnutí a úloh vykonaných v každom kroku. Procedúra môže zahŕňať viaceré prípady použitia (pracovný tok), samostatný prípad použitia alebo iba časť prípadu použitia.

Diagram činností na úrovni pracovného toku reprezentuje poradie vykonávania množiny prípadov použitia. Pre jeden prípad použitia diagram činností vysvetľuje ako účastník interaguje so systémom aby dosiahol ciele prípadu použitia.

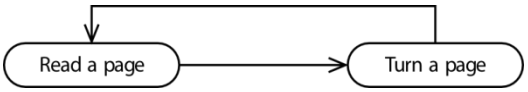
7.4.2.Definovanie metód

Diagramy činností sa môžu taktiež použiť na modelovanie implementácie zložitých metód. Keď definujete implementáciu operácie, potrebujete modelovať sekvenciu dátových manipulácií, cykly a rozhodovaciu logiku. Modelovaním zložitých funkcií môžete predchádzať problémom pri písaní kódu odhalením všetkých požiadaviek explicitne z diagramu. Diagramy činností obsahujú všetky logické konštrukcie, ktoré môžeme nájsť vo väčšine programovacích jazykov.

7.4.3.Notácia diagramu aktivít

Činnosti a prechody

Činnosť (aktivita, angl. activity) je krok v procese, v ktorom sa musí urobiť nejaká práca. Môže to byť výpočet, nájdenie dát, manipulácia s informáciami alebo verifikácia dát. Činnosť je reprezentovaná oblým obdĺžnikom obsahujúcim jednoduchý text. *Diagram činností* predstavuje sériu činností spojených *prechodmi* (angl. transitions) – šípkami prepájajúcimi každú činnosť. Prechod sa typicky vykoná po skončení činnosti. Napríklad sa nachádzame v činnosti „Čítanie strany“. Keď ukončíme túto činnosť, prejdeme do činnosti „Otočenie strany“, atď.

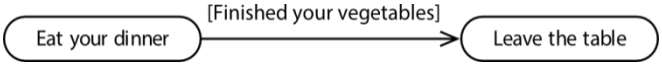


ACD 1: Činnosti a prechody.

Táto notácia začína ukazovať vzájomný prekrýv medzi diagramom činností a stavovým diagramom. V skutočnosti je diagram činností podmnožinou stavového diagramu. Každá činnosť predstavuje *stav akcie*, v ktorej je objekt zaneprázdnený vykonávaním určitej činnosti. Každý prechod predstavuje zmenu stavu, zmenu z jednej činnosti alebo aktívneho stavu do nasledujúceho.

„Strážiaca“ podmienka

V niektorých prípadoch by mali byť prechody použité iba ak nastanú určité udalosti. „*Strážiacu*“ *podmienku* (angl. guard condition) môžeme priradiť prechodu, aby sme obmedzili jeho použitie. Podmienka sa vkladá do hranatých zátvoriek v blízkosti prechodovej šípky. Podmienka musí byť pravdivá, inak nemôžeme použiť asociovaný prechod do ďalšej činnosti. Segment diagramu činností na obrázku hovorí, že nemôžete odísť od stola po zjedení večere, až pokým nezjete všetku zeleninu.

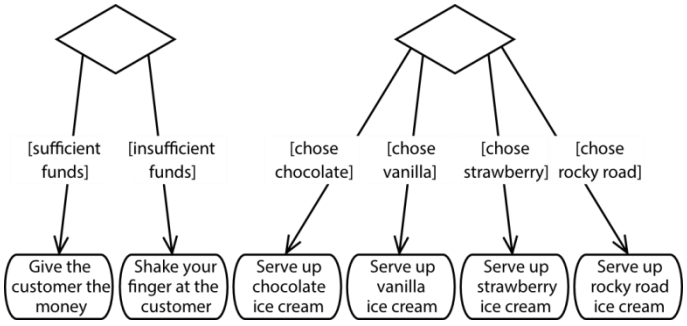


ACD 2: Strážiaca podmienka na prechode.

Rozhodovacie bloky

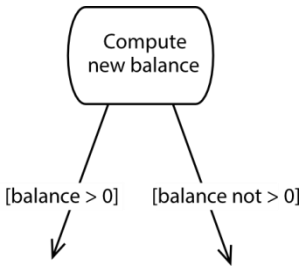
V diagrame činností sa (tak ako aj vo vývojovom diagrame) používa pre rozhodovací blok ikona kosoštvorca. V oboch diagramoch vychádza z rozhodovacieho bloku jedna šípka pre každú z možných hodnôt testovanej podmienky. Rozhodnutie môže byť jednoduchým testovaním pravdivostnej hodnoty true/false (napríklad „Nachádza sa na zákazníckom účte dostatok finančných zdrojov na pokrytie výberu?“). Rozhodnutie môže zahŕňať voľbu z viacerých možností (napríklad „Dáš si čokoládovú, vanilkovú, jahodovú alebo rocky road zmrzlinu?“).

Každá z možností je identifikovaná „strážiacou“ podmienkou. Všetky strážiace podmienky sa musia vzájomne vylučovať, aby existovala vždy iba jedna možná voľba v ľubovoľnom bode rozhodovania (angl. decision point). Strážca sa umiestňuje na prechod, ktorý zobrazuje smer logického toku ak je podmienka splnená. Pri programovaní ste pravdepodobne použili pre riešenie rovnakého typu problému príkaz case.



ACD 3: Rozhodovací blok.

Z dôvodu, že každá voľba v bode rozhodovania je modelovaná strážiacou podmienkou, je možné použiť podmieňovaciu logiku rovnako aj na prechode vychádzajúcom z činnosti. Napríklad činnosť počítajúca nový zostatok na účte prezrádza, či je účet prečerpaný (obrázok). Činnosť poskytuje všetky informácie potrebné pre rozhodovanie. Pre zobrazenie možností vyplývajúcich z činností jednoducho namodelujte prechody vychádzajúce z činnosti, každý s rôznou strážiacou podmienkou.



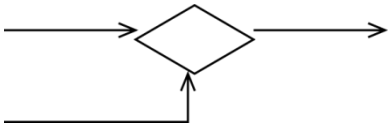
ACD 4: Rozhodnutie v činnosti.

Rozhodovací blok (obrázok) použite v prípade, že v danom kroku nie je zahrnuté žiadne spracovanie. To zvyčajne znamená, že potrebná informácia bola akumulovaná z niekoľkých predchádzajúcich krokov alebo je rozhodnutie explicitnou voľbou účastníka. Rozhodnutie v činnosti (obrázok) použite v prípade, že ukončenie činnosti poskytuje všetky požadované informácie pre podporenie rozhodnutia.

Zlúčenie

Ikona kosoštvorca sa taktiež používa na modelovanie *bodu zlúčenia* (angl. merge point), miesta kde sa dva alternatívne toky spájajú a ďalej pokračujú ako jeden. Dva toky v tomto prípade predstavujú dve vzájomne sa vylučujúce cesty. Napríklad ty a ja kráčame z tvojho domu do obchodu. Ja som sa rozhodol ísť po ľavej strane ulice a ty po pravej. Ale dva bloky pred obchodom potrebujeme obaja odbočiť vpravo a kráčať po pravej strane ulice vedúcej ku vchodovým dverám obchodu. Ani jeden z nás však nemôže ísť naraz oboma cestami. Ale nech si vyberieme ktorúkoľvek cestu, musíme kráčať aspoň dva bloky presne rovnakým spôsobom.

Bod zlúčenia si taktiež môžeme predstaviť ako zariadenie na ušetrenie práce. Alternatívou by bolo modelovať rovnakú sekvenciu krokov pre každú z ciest, ktoré túto sekvenciu krokov obsahujú. Obrázok ilustruje zlúčenie alternatívnych tokov, ktoré ďalej pokračujú ako jeden samostatný tok. Kosoštvorec reprezentuje bod, v ktorom sa toky zbiehajú.



ACD 5: Bod zlúčenia.

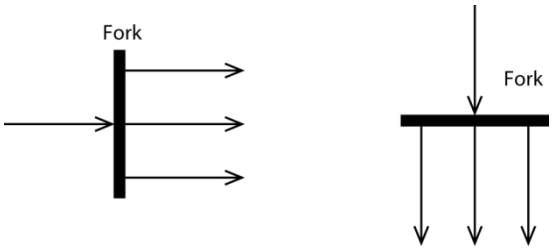
Začiatok a koniec

UML taktiež poskytuje ikony pre začiatok a koniec diagramu činností (obrázok). Plný kruh označuje začiatok toku činnosti. Plný kruh v kružnici označuje koniec. V diagrame činností sa môže nachádzať viac ako jeden koncový bod. Dokonca aj najjednoduchší diagram činností má typicky nejakú rozhodovaciu logiku, ktorá vyústi do alternatívnych tokov, každý s jeho vlastnými unikátnymi výsledkami. Ak chcete, môžete nakresliť všetky šípky tak, aby smerovali do jedného koncového bodu, ale nemusíte to tak robiť. Každý koncový bod znamená to isté: Tu sa zastav.

ACD 6: Notácia začiatku a konca pre diagram činností.

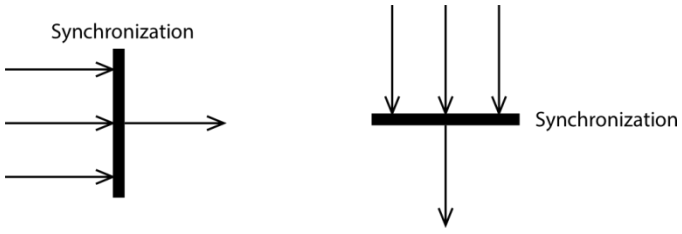
Súbežnosť

Notácia UML pre diagram činností taktiež podporuje súbežnosť. To vám umožňuje modelovať črty jazykov, ktoré boli predstavené od vynájdenia vývojového diagramu, ako Java, C++ a dokonca Smalltalk, na hardvéri umožňujúcom skutočnú súbežnosť. Pre zobrazenie, že jeden proces spúšťa viaceré súbežné vlákna alebo procesy UML používa jednoduchý pruh nazývaný *rozvetvenie* (angl. fork). V príkladoch na obrázku môžete vidieť jeden prechod vchádzajúci do rozvetvenia a viacero prechodov vychádzajúcich z rozvetvenia. Každý vychádzajúci prechod predstavuje nové vlákno alebo proces.



ACD 7: Rozdelenie toku riadenia použitím rozvetvenia: inicializácia viacerých vlákien alebo procesov.

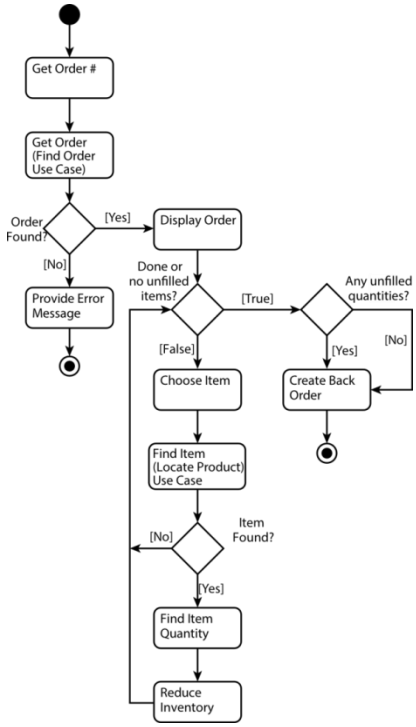
Synchronizácia (angl. synchronization) alebo spájanie súbežných vlákien alebo procesov je zobrazená veľmi podobným spôsobom. Obrázok zobrazuje viaceré prechody vchádzajúce do *synchronizačného pruhu* (angl. synchronization bar) a jeden vychádzajúci zo synchronizačného pruhu. Toto indikuje, že súbežné procesy sa skončili a proces pokračuje ďalej ako samostatné vlákno alebo proces.



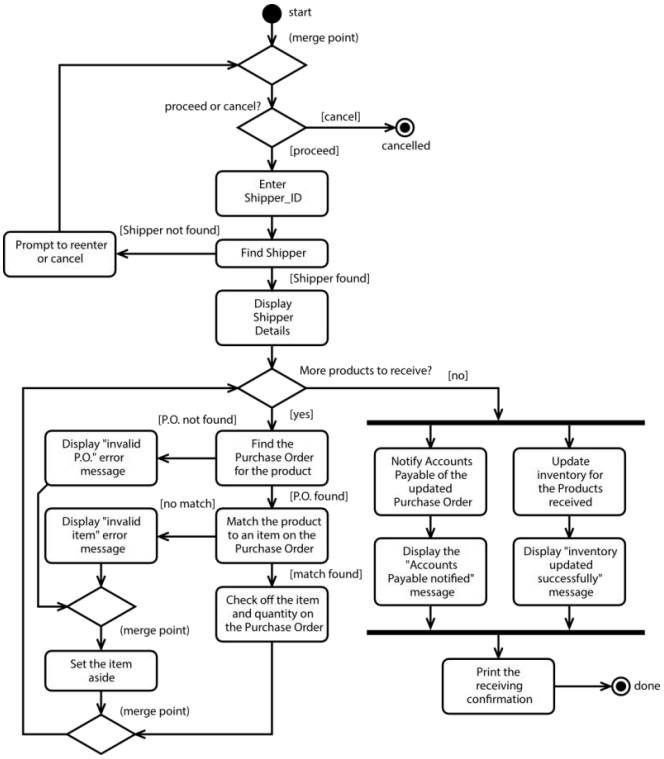
ACD 8: Spájanie tokov riadenia použitím synchronizačného pruhu.

Je potrebné upozorniť, že identifikácia *príležitosti* na súbežnosť nediktuje nevyhnutne *požiadavku* na súbežnosť. Jednoducho modeluje fakt, že sekvenčné spracovanie nie je vyžadované, a ak to implementačné prostredie podporuje, existuje možnosť optimalizovať výkonnosť aplikácie využitím príležitosti na súbežnosť.

7.4.4.Príklady



ACD 9: Diagram činností pre prípad použitia Splniť objednávku.



ACD 10: Diagram činností.

7.5. Sekvenčný diagram

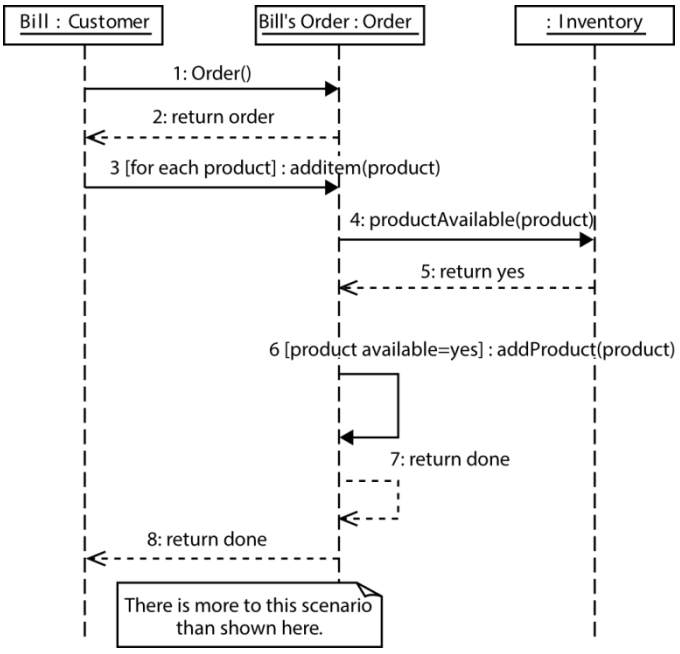
Sekvenčný diagram (angl. Sequence diagram) ilustruje *interakcie medzi objektmi*. Interakcie nám ukazujú ako objekty medzi sebou komunikujú. Vždy keď jeden objekt komunikuje s iným, komunikuje s rozhraním (t.j. vyvolá operáciu). Takže, ak viete modelovať interakcie, viete nájsť rozhrania/operácie, ktoré objekt vyžaduje. Sekvenčný diagram poskytuje prechod z textových opisov správania v scenároch prípadov použitia k operáciám/rozhraniam v diagrame tried.

7.5.1. Mapovanie interakcií do objektov

Všetko v objektovo-orientovanom systéme je vykonávané objektmi. Objekty preberajú zodpovednosť za veci ako manažovanie dát, presúvanie dát v systéme, odpovedanie na požiadavky a ochrana systému. Objekty spolupracujú komunikovaním alebo interagovaním medzi sebou. Obrázok zobrazuje sekvenčný diagram s tromi zúčastnenými objektmi: zákazník Bill, Billova objednávka a inventár. Dokonca aj bez formálnej znalosti notácie pravdepodobne dokážete získať dostatočne dobrú predstavu o čo vlastne ide.

Krok 1 a 2: Bill vytvorí objednávku.
Krok 3: Bill sa pokúša pridať položku do objednávky.
Krok 4 a 5: Pre každú položku sa skontroluje jej dostupnosť v inventári.
Krok 6 a 7: Ak je produkt dostupný, pridá sa do objednávky.
Krok 8: Bill zistí, že všetko fungovalo správne.

Vytvorenie sekvenčného diagramu je jednoduchšie ak ste dokončili aspoň prvý draft modelu prípadov použitia a diagramu tried. Z týchto dvoch zdrojov získate množiny interakcií (scenárov) a množstvo potenciálnych objektov pre prevzatie zodpovednosti za interakcie.



SQD 1: Jednoduchý sekvenčný diagram.

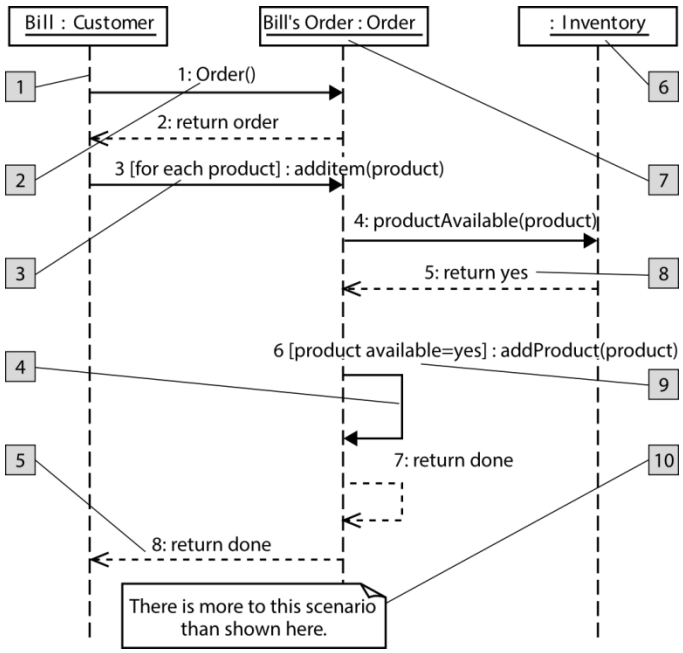
7.5.2. Základná notácia sekvenčného diagramu

Všetky sekvenčné diagramy sú modelované na úrovni objektov (nie na úrovni tried), čo umožňuje vytvárať scenáre, ktoré používajú viac ako jednu inštanciu tej istej triedy a pracovať na úrovni faktov,

testovacích dát a príkladov. Sekvenčný diagram používa tri fundamentálne notačné elementy: objekty, správy/stimuly a čiary života objektov.

V sekvenčnom diagrame *objekty* používajú tú istú notáciu ako v diagrame objektov. Na obrázku môžeme vidieť tri zúčastnené objekty zoradené vo vrchnej časti diagramu. Čiara života objektu (identifikovaná odkazom #1 na obrázku) je prerušovaná vertikálna čiara pod každým z objektov. Čiara života objektu plynie vždy zo začiatku na vrchu ku koncu na spodku. Množstvo reprezentovaného času závisí od scenára alebo ďalšieho správania, ktoré modelujete.

Správa alebo stimul je zvyčajne volanie, signál alebo odpoveď. Správa je reprezentovaná šípkou. Typ šípky vizuálne opisuje typ správy. Štýl plnej čiary a plného hrotu šípky reprezentuje správu, ktorá vyžaduje odpoveď. Prerušované šípky sú odpovede. Správy sú umiestnené horizontálne na časové osi v relatívnej vertikálnej pozícii jedna za druhou, čo reprezentuje poradie, v ktorom sa udiali. Toto usporiadanie umožňuje čítať diagramy od začiatku do konca čítaním správ z hora nadol.



SQD 2: Elementy notácie sekvenčného diagramu.

Čísla odkazov na obrázku označujú tieto položky:

1. Čiara života objektu (angl. object lifeline)
2. Správa/Stimul (angl. message/stimulus)
3. Iterácia (angl. iteration)
4. Odkaz na seba (angl. self-reference)
5. Návrat (angl. return)
6. Anonymný objekt (angl. anonymous object)
7. Názov objektu (angl. object name)
8. Číslo sekvencie (angl. sequence number)

9. Podmienka (angl. condition)
10. Komentár (angl. basic comment)

Čísla sekvencií sú voliteľné, no veľmi užitočné v prípade, že potrebujete diskutovať o diagramoch a robiť zmeny. Každá šípka správy opisuje rozhranie/operáciu objektu, ku ktorému smeruje. Z tohto dôvodu správa obsahuje signatúru operácie (angl. operation signature), t.j. názov, argumenty a prípadne návrat, ako napríklad addItem(product):boolean.

Prerušované šípky návratu s odkazmi #2 a #5 obsahujú iba odpoveď na správu. Niektorí ich vynechávajú. Ale účelom modelovania je odhaľovať informácie, nie vytvárať predpoklady. Zobrazenie návratov môže pomôcť ubezpečiť sa, že to, čo dostanete naspäť je konzistentné s tým, o čo ste žiadali v správe.

Odkaz #3 na obrázku zobrazuje ako môžete indikovať, že operácia by mala byť vykonávaná opakovane. Použite hranaté zátvorky podmienky, do ktorých vložte buď počet opakovaní alebo podmienku, ktorá kontroluje opakovanie, napríklad [for each product].

Rovnaké podmienkové zátvorky môžu byť použité na kontrolu, či bola správa vôbec odoslaná. Odkaz #9 ukazuje na krok 6, ktorý používa test [product available = yes], aby sa uistil, že predchádzajúci krok uspel pred vykonaním operácie v kroku 6.

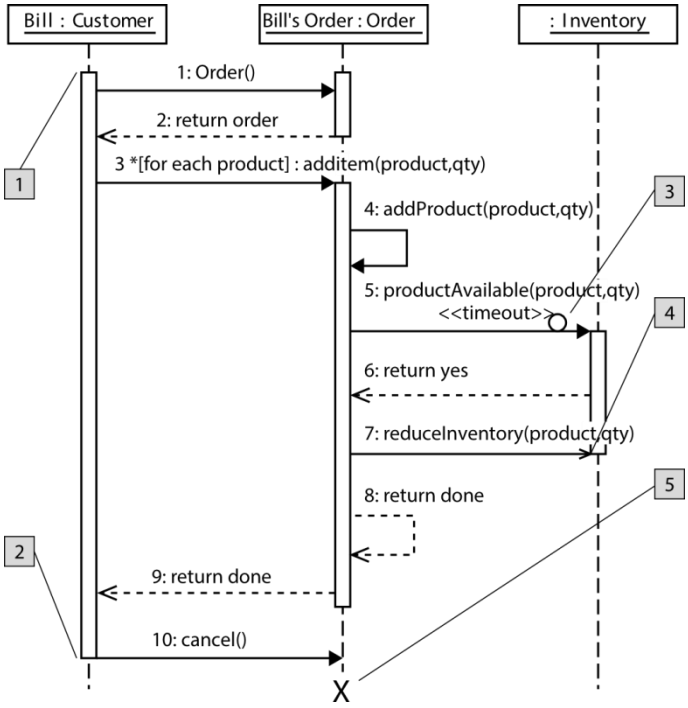
Odkaz #10 zobrazuje ako môžete použiť UML poznámku pre pridanie informácie, ktorá nie je explicitnou súčasťou notácie.

7.5.3.Rozšírená notácia pre sekvenčný diagram

Sekvenčné diagramy môžu byť rozšírené pre ilustrovanie *aktivácie* objektu a *ukončenia* objektu a na prispôsobenie správ.

Obrázok zahŕňa niekoľko zmien oproti pôvodnému sekvenčnému diagramu, aby ilustroval tieto nové črty. Pre zobrazenie, že objekt je aktívny sa používa notácia rozšírenia vertikálnej čiary života objektu na úzky obdĺžnik, ako na obrázku. Úzky obdĺžnik sa nazýva „aktivačný pruh“ (angl. activation bar) alebo „stredobod kontroly“ (angl. focus of control). Odkaz #1 zobrazuje, kedy sa stáva objekt aktívnym (na vrchu obdĺžnika). Všimnite si, že objekt sa stáva aktívnym keď začína vykonávať činnosť. Odkaz #2 zobrazuje, kedy je objekt deaktivovaný alebo kedy ukončuje svoju činnosť a čaká na ďalšiu požiadavku. Použitím tejto notácie môžeme vidieť, že objekt triedy Inventory je aktívny iba pokým odpovedá na požiadavku „productAvailable“ a objekt triedy Order je aktivovaný viac ako jedenkrát: raz pri vytvorení objektu triedy Order a vždy, keď je požiadaný Billom aby vykonal „addItem“.

Pre zobrazenie, že objekt je ukončený, umiestnite X do bodu na čiare života objektu, kedy nastane ukončenie. Toto je zvyčajne odpoveď na správu ako vymazať alebo zrušiť. Viď správu 10: cancel() nasledovanú X, odkaz #5. Absencia X na čiare života objektu znamená, že objekt žije ďalej aj po ukončení tejto sekvencie udalostí.



SQD 3: Rozšírené elementy notácie sekvenčného diagramu.

Na obrázku môžete vidieť tieto notácie:

1. Aktivácia (angl. activation): Začiatok vertikálneho obdĺžnika, aktivačného pruhu.
2. Deaktivácia (angl. deactivation): Koniec vertikálneho obdĺžnika, aktivačného pruhu.
3. Udalosť vypršania času (angl. timeout event): Typicky značený plným hrotom šípky s malým ciferníkom alebo kružnicou na čiare.
4. Asynchrónna udalosť (angl. asynchronous event): Typicky značená s čiarovým hrotom šípky.
5. Ukončenie objektu (angl. object termination) symbolizované X.

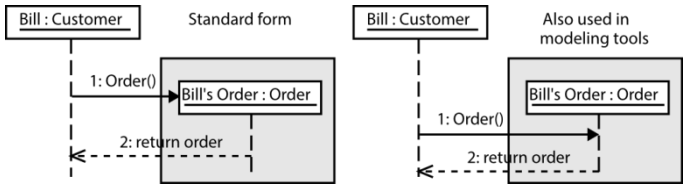
Obrázok taktiež uvádza niekoľko nových typov správ. Odkaz #3 ukazuje na správu s krúžkom na čiare a stereotypom <<timeout>>. Toto sa nazýva *časová udalosť* (angl. timed event). Často sa tam nachádza aj podmienka alebo ohraničenie na správu, ktoré vyjadruje parametre časovania udalosti, napríklad {ak nedostaneme odpoveď z inventára do 2 sekúnd túto položku preskočíme a skontrolujeme neskôr}. Vypršanie časového limitu je príkladom rozšírenia UML. Nie je časťou jadra notácie UML, ale reprezentuje platné použitie.

Odkaz #4 ukazuje na *asynchrónnu udalosť*. Typicky vidíte udalosti, ktoré vyžadujú nejaký druh odpovede ako addItem (vykonalo sa to správne alebo nie?) alebo productAvailable (je na sklade nejaký produkt?). Ale sú prípady kedy udalosť predstavuje jednoducho signál pre iný objekt, aby niečo urobil. Správa 7 na obrázku bola poslaná do inventára na zredukovanie počtu produktov o zadané množstvo. Je na inventári, aby zaradil požiadavku do fronty a postaral sa o ňu. Proces, ktorý zadal objednávku nebude čakať. (Tento príklad je dobrý na ilustráciu, ale pravdepodobne to nie je dobrý návrh.)

Všimnite si rozdiel v šípkach. Asynchrónna správa používa čiarkovaný hrot šípky namiesto plného hrotu šípky použitej pre jednoduchú alebo synchrónnu správu.

Teraz sa pozrime na správy 4 a 8. Správa 4 spúšťa operáciu „addProduct“, ale návrat sa neuskutoční až do správy 8. Všetky správy medzi 4 a 8 sú správy poslané objektom triedy Order počas vykonávania operácie „addProduct“. Toto je ďalší dobrý dôvod, prečo je vhodné zobrazovať návraty. Bez explicitne zobrazených návratov na obrázku by bolo možné interpretovať diagram tak, že systém najprv pridá produkt, teda ešte pred tým, ako vôbec skontroluje, či je produkt dostupný.

Nakoniec, pre modelovanie vytvorenia objektu máte niekoľko možností. Na obrázku vidíte správu 1: Order() smerujúcu k čiare života objektu. Toto je bežná programovacia konvencia pre konštruktor (operácia s rovnakým názvom ako má trieda) – operáciu, ktorá vytvára objekt. Ale sekvenčný diagram používa čiaru života objektu, ktorá by nám mala umožniť reprezentovať vytvorenie objektu vizuálne. Obrázok zobrazuje dve varianty s použitím čiary života objektu.



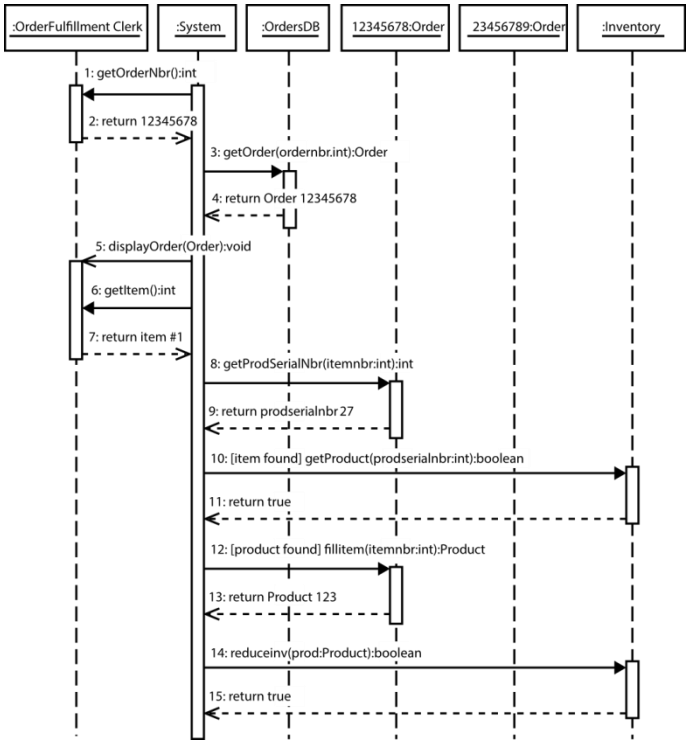
SQD 4: Dva spôsoby ako modelovať vytvorenie objektu.

Príklad vľavo je forma explicitne definovaná UML. Správa na vytvorenie objektu (konštruktor) ukazuje priamo na objekt. To znamená, že ikona objektu musí byť umiestnená niekde nižšie na strane na miesto, kde sa prebieha samotné vytvorenie (namiesto umiestnenia na vrchu diagramu). Používanie tejto techniky implikuje, že objekty na vrchu strany už existovali na začiatku scenára.

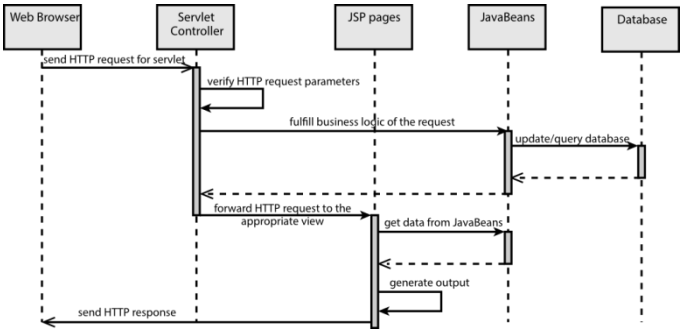
Príklad vpravo je menej používaný variant, kde konštruktor ukazuje na čiaru života objektu hneď pod objektom. Objekt je však aj v tomto prípade umiestnený v bode, ktorý reprezentuje čas jeho vytvorenia (nie na vrchu diagramu).

Cieľom je reprezentovať fakt, že objekt neexistoval pred vyslaním určitej správy. Čiara života objektu doslova začína existovať, keď sa pošle správa na vytvorenie objektu, takže pred (nad) správou na vytvorenie objektu nie je žiadna čiara života objektu.

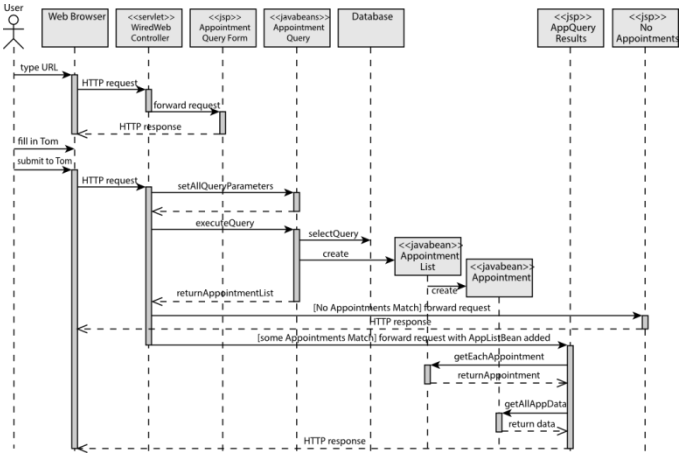
7.5.4.Príklady



SQD 5: Sekvenčný diagram.



SQD 6: Sekvenčný diagram – Model 2 Architecture.

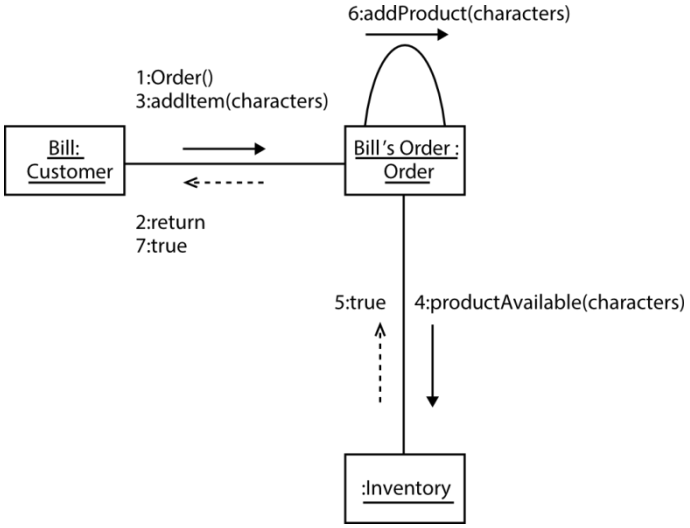


SQD 7: Sekvenčný diagram – Appointment querying implementation.

7.6. Diagram komunikácií

Obrázok zobrazuje rovnakú množinu interakcií modelovaných na obrázku použitím sekvenčného diagramu. Scenár zobrazuje zákazníka Billa ako vytvára objednávku a pridáva do nej položky, pričom sa kontroluje dostupnosť každej z pridaných položiek. Pre prechádzanie scenárom jednoducho nasledujte číslované správy.

Rovnakú vec môžete dosiahnuť oboma diagramami (t.j. môžete modelovať logické kroky v procese ako je scenár prípadu použitia). V nasledujúcich častiach identifikujem podobné a rozdielne črty týchto dvoch diagramov tak, aby sme vedeli vybrať diagram, ktorý najviac pomôže s určitým problémom v projekte.



CMD 1: Diagram komunikácií pre zákazníka zadávajúceho objednávku.

7.6.1. Sekvenčný diagram a diagram komunikácií

Podobné črty diagramov

Sekvenčný diagram a diagram komunikácií modelujú rovnaké dva elementy: správy a objekty. V skutočnosti sú oba diagramy natoľko podobné, že niektoré modelovacie nástroje, ako System Architect a Rational Rose poskytujú možnosť prepínať sa medzi týmito dvoma pohľadmi tam aj späť.

Podobne ako sekvenčný diagram aj diagram komunikácií poskytuje nástroj na vizuálne priradenie zodpovedností objektom za zasielanie a prijímanie správ. Identifikovaním objektu ako príjemcu správy v podstate tomuto objektu priraďujete rozhranie. Je to podobné ako prijímanie telefonických hovorov. Prijať telefonát môžete len, ak máte číslo volajúceho. Číslo je vaše rozhranie. Opis správy sa stáva signatúrou operácie na prijímajúcom objekte. Zasielajúci objekty vyvoláva operáciu.

Rozdielne črty diagramov

Diagram komunikácií kladie prioritu na mapovanie interakcií objektov na prepojenia objektov (t.j. kreslenie zúčastnených objektov vo formáte diagramu objektov a umiestňovanie správ paralelne s prepojeniami objektov). Táto perspektíva umožňuje validovať diagram tried poskytovaním evidencie potreby pre každú asociáciu prostredníctvom zasielania správ. Na druhej strane, sekvenčný diagram prepojenia medzi objektmi vôbec neilustruje.

Toto zvýrazňuje výhodu diagramu komunikácií. Logicky, nemôžete umiestniť správu mimo prepojenia, pretože tam neexistuje žiadne fyzický spôsob, ako preniesť správu. V sekvenčnom diagrame vám nič nebráni nakresliť šípku medzi dvoma objektmi, aj bez korešpondujúceho prepojenia. Týmto by ste ale namodelovali logickú interakciu, ktorá sa fyzicky nemôže uskutočniť.

Môžete sa na to pozerať aj opačne. Kreslenie správy mimo prepojenia odhaľuje *potrebu* nového prepojenia. Uistite sa, že ste naozaj aktualizovali váš diagram tried, inak (ako bolo povedané pred tým) nebudete schopní implementovať správu ilustrovanú v diagrame.

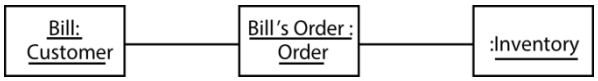
Výhodou sekvenčného diagramu je jeho schopnosť zobrazovať vytváranie a deštrukciu objektov. Novovytvorené objekty môžu byť umiestnené na časovú os v bode, kde sú vytvorené. Veľké X na konci časovej osi zobrazuje, že objekt už viac nie je možné použiť.

Sekvenčné diagramy majú taktiež výhodu zobrazovania aktivácie objektov. Pretože diagram komunikácií neilustruje čas, je nemožné explicitne indikovať, kedy je objekt aktívny alebo neaktívny bez interpretovania typov zasielaných správ.

7.6.2.Notácia diagramu komunikácií

Diagram komunikácií (angl. Communication diagram) používa ako základ diagram objektov. Najprv určte, ktoré objekty budú zúčastnené v scenári. Nakreslite objekty iba s oddelením názvu (nie s oddelením atribútov). Potom nakreslite prepojenia medzi nimi. Pretože ľubovoľná dvojica tried môže mať viac ako jednu asociáciu, potrebujete použiť diagram tried ako pomôcku pre identifikovanie platných typov prepojení, ktoré sa týkajú aktuálnej sekvencie správ.

Obrázok zobrazuje objekty a ich prepojenia. V prípade, že existuje medzi dvomi súvisiacimi triedami iba jeden typ asociácie, môžete názov prepojenia vynechať. Pridajte názvy, ak je možné, že existuje viac ako jeden druh prepojenia medzi dvoma objektmi a potrebujete objasniť, ktorý vzťah podporuje interakciu.



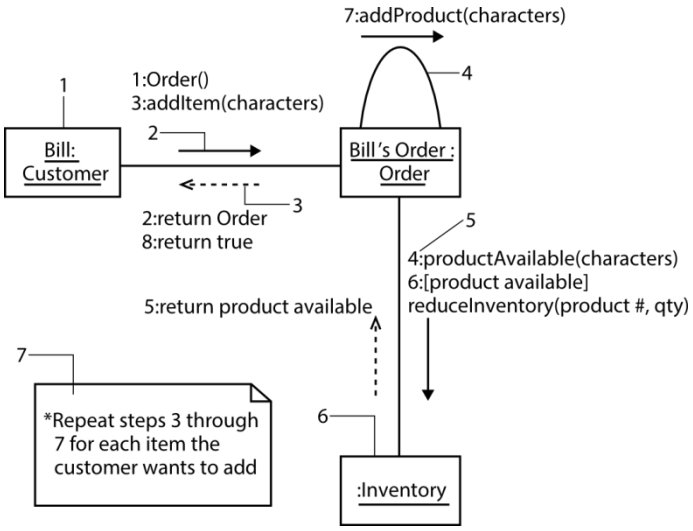
CMD 2: Základná notácia diagramu komunikácií.

Pre každý krok scenára nakreslite šípku správy zo zasielajúceho objektu do prijímajúceho objektu. Šípku správy umiestnite paralelne s prepojením medzi zasielajúcimi a prijímajúcimi objektmi. Umiestnenie viacerých správ na to isté prepojenie je platné a v skutočnosti bežné, pokiaľ majú naozaj rovnaký typ správy (šípky). Očíslujte správy v poradí, v akom sa udejú.

Formát špecifikácie správy je rovnaký ako v sekvenčnom diagrame:

číslo_sekvencie iterácia : [podmienka] operácia alebo návrat

Obrázok modeluje celý scenár vytvárania objednávky. Oproti originálu na obrázku som pridal niekoľko zmien, aby som mohol demonštrovať všetky notácie.

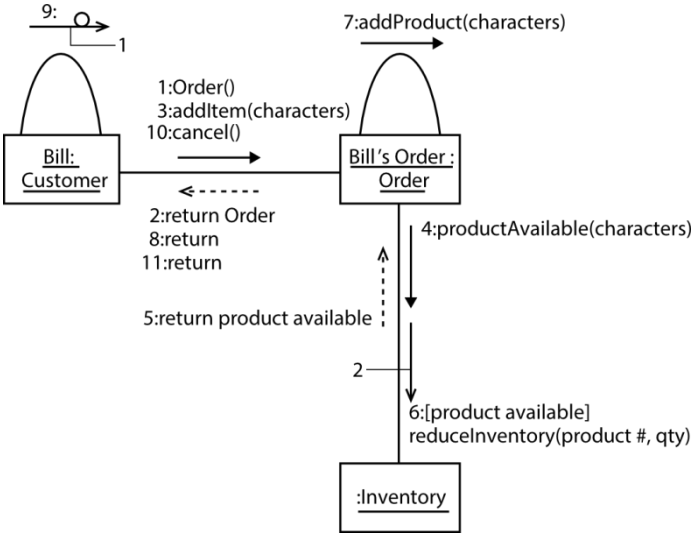


CMD 3: Detailnejšia notácia diagramu komunikácií.

Nasledujúce opisy sa odkazujú na čísla položiek na obrázku, aby ste mohli vidieť notácie použité v kontexte:

1. **Objekt:** Toto je plne kvalifikovaný názov objektu, Bill, triedy Customer. Notácia je zhodná s notáciu v sekvenčnom diagrame.
2. **Synchrónna udalosť alebo volanie procedúry:** Synchrónna udalosť je správa, ktorá vyžaduje odpoveď, takže by ste očakávali korešpondujúcu návratovú správu pozdĺž toho istého prepojenia niekde ďalej v sekvencii. Volania procedúr sú iné známe spôsoby opisu tejto „spýtaj sa a odpovedz“ formy interakcie.
3. **Návrat:** Toto je návratová správa správy 1. Správa 1 povedala triede Order, aby vytvorila nový objekt, Bill's Order. Po ukončení vytvárania objektu, vráti odkaz naspäť žiadateľovi, Billovi.
4. **Odkaz na seba:** Odkaz na seba je jednoducho, keď objekt hovorí sebe samému niečo ako, „Je čas, aby som si zohnal ďalšiu kávu.“ Na obrázku Order hovorí sama sebe, aby použila informáciu o položke z kroku 3, aby mohla pridať ďalší produkt do jej zoznamu položiek.
5. **Číslo sekvencie:** Pretože diagram komunikácií nemá možnosť zobraziť plynutie času, používa sekvenčné čísla, ako (4:), aby ukázala poradie vykonávania správ. Pre schému číslovania neexistujú žiadne štandardy, takže sa nechajte viesť zdravým rozumom a čitateľnosťou. V sekvenčnom diagrame boli sekvenčné čísla voliteľné. V diagrame komunikácií sú vyžadované.
6. **Anonymný objekt:** Odkaz 6 zobrazuje ďalší príklad platnej notácie objektu. Inštanciu nemusíte pomenovať, ak všetko čo chcete vyjadriť je, že ľubovoľný objekt tejto triedy (Inventory) by sa správal týmto spôsobom.
7. **Komentár:** Odkaz 7 zobrazuje *jeden zo spôsobov*, ako môžete odhaliť váš zámer opakovať množinu správ. Rovnako ako v sekvenčných diagramoch môžu byť komentáre veľmi užitočné pri vysvetľovaní vašich zámerov, týkajúcich sa iterácie cez *množinu správ*, pretože notácia iterácie poskytnutá UML funguje iba pre jednu udalosť.

Teraz zmením v diagrame niekoľko položiek tak, aby ste mohli vidieť niektoré nové notácie. Nasledujúce opisy sa odkazujú na číslované položky na obrázku.



CMD 4: Ďalšia notácia diagramu komunikácií.

Všimnite si nasledované položky v diagrame:

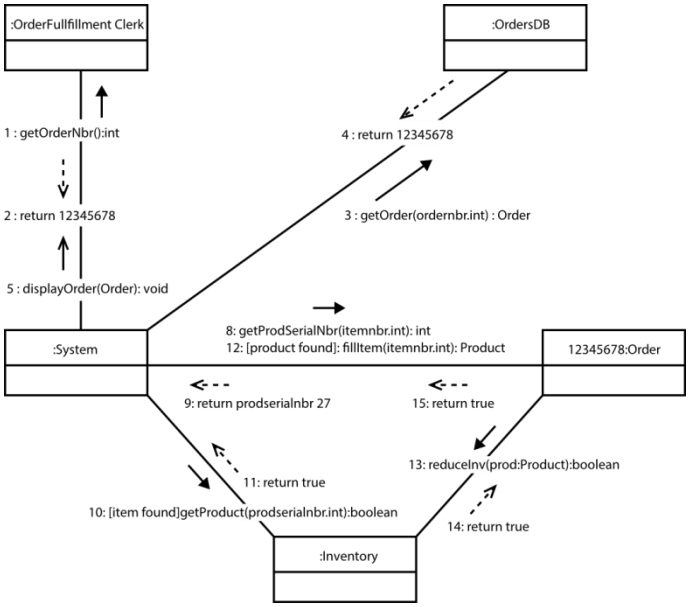
1. **Udalosť vypršania času:** Túto správu som neoznačil predovšetkým preto, lebo vypršanie času by bolo pre tento typ scenára mierne nezvyčajné. Takto môžete vidieť notáciu aj napriek tomu. V skutočnosti ide o bežné rozšírenie UML (t.j. UML to explicitne nedefinuje). Malý krúžok reprezentuje hodiny a niekedy dokonca zobrazuje vo vnútri krúžku ručičky. Vypršanie času by sa mohlo použiť na niečo ako vytáčanie do siete alebo voľba uzla v sieti. Ak nedostanete odpoveď v stanovenom časovom intervale, prestaňte skúšať a pokračujte ďalej. V tomto prípade, ak Order neodpovie v stanovenom časovom limite, objednávka sa zruší v kroku 10.
2. **Asynchrónna správa:** Asynchrónna správa nevyžaduje odpoveď. Krok 6 sa zmenil, aby inventáru jednoducho povedal, „Vzal som si nejaké zásoby. Možnože si chceš aktualizovať svoje záznamy. Ja však na to nebudem čakať.“

Diagram komunikácií modeluje v podstate rovnaké informácie ako sekvenčný diagram, interakcie medzi objektmi. Iná je perspektíva. Diagram komunikácií zobrazuje interakcie súvisiace so štruktúrou objektov a vzťahmi (prepojeniami) medzi nimi. Sekvenčný diagram sa sústreďuje na časové rozvrhnutie. Preto výhodou diagramu komunikácií je, že vám môže pomôcť overiť asociácie medzi triedami alebo dokonca objaviť nové potrebné asociácie.

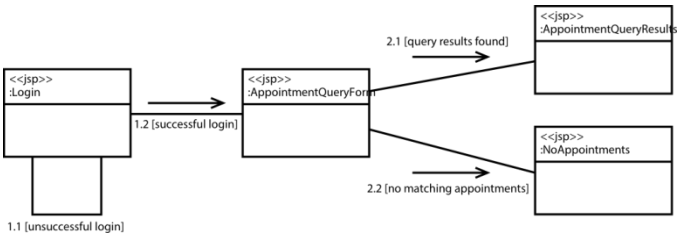
Diagram komunikácií je postavený na diagrame objektov.

1. Do diagramu umiestnite zúčastnené objekty.
2. Nakreslite prepojenia medzi objektmi, pričom ako pomôcku použite diagram tried.
3. Pridajte všetky udalosti. Umiestnite šípku správy paralelne s prepojením medzi dvoma objektmi. Šípku kreslite tak, aby ukazovala smerom od odosielateľa k príjemcovi.
4. Očíslujte správy v poradí ich vykonávania.
5. Opakujte kroky 3 a 4, pokým nenamodelujete celý scenár.

7.6.3.Príklady



CMD 5: Diagram komunikácií.



CMD 6: Diagram komunikácií – Querying appointments page flow.

7.7. Stavový diagram

Stavový diagram (angl. State machine diagram) opisuje život objektu z hľadiska udalostí, ktoré spúšťajú zmeny v *stave* objektu. Identifikuje *externé udalosti* aj *interné udalosti*, ktoré môžu zmeniť stav objektu. Čo to ale znamená? *Stav* (angl. state) objektu je jednoducho jeho aktuálny stav (angl. condition). Tento stav odráža hodnoty atribútov, ktoré opisujú objekt. V systéme existujú reakcie (angl. behaviours), ktoré upravujú hodnoty týchto atribútov.

Stav opisuje objekt, takže typicky sa v opise problému vyskytuje vo forme prídavného mena. Napríklad, účet je otvorený (*otvorený* účet) alebo účet je prečerpaný (*prečerpaný* účet).

Keď aktuálny stav účtu je prečerpaný, účet bude odpovedať iným spôsobom ako keby bol v stave otvorený – šeky budú odmietnuté (namiesto ich vyplatenia), alebo šek pokryje banka a za túto láskavosť vám naúčtuje prehnaný poplatok.

Ďalej porovnajme rozsah stavového a sekvenčného diagramu. Rozsah stavového diagramu je celý život objektu. Rozsah sekvenčného diagramu je jeden scenár. Preto je možné odvodiť stavový diagram z množiny sekvenčných diagramov, ktoré daný objekt používajú.

Stavový diagram modeluje udalosti, ktoré spúšťajú *prechody* (zmeny) z jedného stavu do iného. Každá udalosť môže mať korešpondujúcu *akciu*, ktorá spôsobí zmeny v objekte (t.j. upraví hodnoty

atribútov). Pokým je objekt v stave, môže taktiež vykonávať prácu spojenú s týmto stavom. Takáto práca sa nazýva *činnosť* (aktivita, angl. activity).

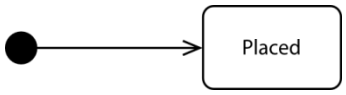
Stavový diagram môže byť taktiež použitý na modelovanie súbežných aktivít vo vnútri stavu vytvorením paralelných *podradených stavov* vo vnútri *nadradeného stavu*. Používaním notácie podradených stavov a nadradených stavov môžete explicitne identifikovať rozdelenie a zlúčenie riadenia súbežnosti.

7.7.1.Základná notácia pre stavový diagram

Základom pre stavový diagram je vzťah medzi stavmi a udalosťami. Nasledujúci príklad ilustruje notáciu stavového diagramu použitím objektu objednávka. Stav je modelovaný ako zaoblený obdĺžnik s názvom stavu vo vnútri, ako na obrázku, podobne ako skrátená forma ikony triedy, keď je viditeľné iba oddelenie s názvom.

SMD 1: Symbol stavu zobrazený iba s oddelením s názvom (minimálna konfigurácia).

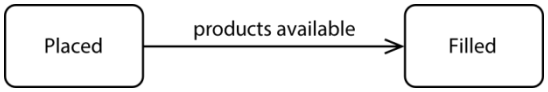
Iniciálny stav objektu má vlastnú unikátnu notáciu, plný kruh so šípkou ukazujúcou na prvý stav. Iniciálny stav indikuje stav, v ktorom je objekt vytvorený alebo skonštruovaný. Obrázok by ste prečítali „Objednáva začína v stave ‚Placed'". Inými slovami, objednávka začína existovať, keď ju používateľ zadá.



SMD 2: Notácia iniciálneho stavu.

Všimnite si, že iniciálny stav je celý výsek diagramu na obrázku. Zahŕňa kruh, šípku a ikonu stavu. V skutočnosti kruh a šípka ukazujú na prvý stav.

Notácia udalosti v stavovom diagrame je šípka s plnou čiarou spájajúca jeden stav s druhým stavom. Šípka je vlastne prechod asociovaný s udalosťou. Smer šípky ukazuje smer zmeny z jedného stavu do druhého. Obrázok zobrazuje udalosť „products available", ktorá spôsobuje prechod (šípka) zo stavu „Placed" do stavu „Filled".



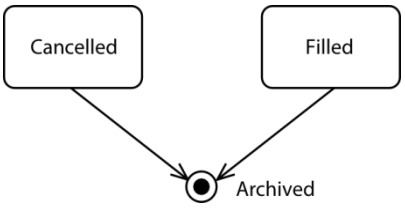
SMD 3: Prechod zo stavu „Placed" do „Filled".

Akcia je asociovaná s udalosťou. Akcia je správanie, ktoré je spustené udalosťou a je to správanie, ktoré mení atribúty, ktoré definujú stav objektu. Pre namodelovanie akcie umiestnite lomku za názov udalosti nasledovanú názvom akcie alebo akcií, ktoré chcete vykonať, ako na obrázku, kde udalosť „products available" spúšťa akciu fillOrder(). Akt splnenia objednávky upravuje jej obsah a redefinuje jej stav.

Akcia je atomická úloha a ako taká nemôže byť členená do menších podúloh, ani nemôže byť prerušená. Neobsahuje žiadne body prerušenia (angl. break points) a navyše jej zastavenie uprostred vykonávania môže zanechať objekt v nedefinovanom stave.

SMD 4: Pár udalosť/akcia.

Objekt môže dosiahnuť *finálny* stav, z ktorého sa už nemôže vrátiť do aktívneho stavu. Inými slovami, nikdy neuvidíte šípku vychádzajúcu z tohto stavu. Bežné použitie je zobrazené na obrázku. Objednávka môže byť archivovaná z obidvoch stavov. Ale po jej archivácii ju už nikdy nemôžete zmeniť. Môžete ju stále vidieť a stále môže existovať, ale už viac nemôžete meniť jej stav. Finálny stav môže taktiež znamenať, že objekt bol v skutočnosti zmazaný.

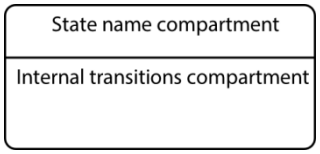


SMD 5: Notácia koncového stavu.

Pretože máme sklony narábať s našimi dátami opatrne, je veľmi ojedinelé, že doslova zmažeme dáta, takže je rovnako ojedinelé vidieť finálny stav. Častokrát, dokonca ak je objekt označený na vymazanie alebo archiváciu, nechávate otvorenú možnosť vrátiť späť operáciu vymazania alebo archivácie na zotavenie z chyby alebo jednoducho, ak ste si to rozmysleli. V takejto situácii by bol stav vymazaný alebo archivovaný normálnym stavov (zaoblený obdĺžnik).

7.7.2. Vnútorné udalosti a činnosti

Ikona stavu môže byť rozšírená. Účelom rozšírenej formy je odhaliť, čo objekt môže robiť, pokým je v danom stave. Notácia jednoducho rozdeľuje ikonu stavu do dvoch oddelení: *oddelenie s názvom* a *oddelenie vnútorných prechodov*, ako je ilustrované na obrázku.

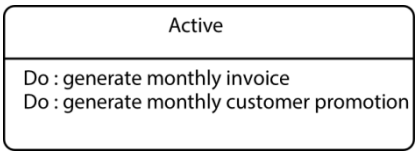


SMD 6: Ikona rozšíreného stavu.

Oddelenie s vnútornými prechodmi obsahuje informácie o akciách a činnostiach špecifických pre daný stav. Už ste videli akcie asociované s udalosťami. Na tomto miesto hovorím o rovnakých akciách, avšak tentokrát dokumentovaných ako vstupné a výstupné akcie vo vnútri stavu.

Činnosti sú procesy vykonávané vo vnútri stavu. Činnosť väčšinou nebýva atomická, teda môže pozostávať zo skupiny úloh. Činnosti môžu byť prerušené pretože neovplyvňujú stav objektu. Porovnajte to s predchádzajúcou definíciou akcie, ktorá hovorila, že nesmiete prerušiť akcie, pretože menia stav objektu. Zastavenie akcie uprostred vykonávania môže objekt zanechať v nedefinovanom stave. Činnosti ale zastaviť môžeme. Nemenia stav objektu.

Napríklad, obrázok modeluje aktívny stav objektu triedy Customer. Pokým je objekt v tomto stave, generuje mesačne faktúry pre nákupnú aktivitu zákazníka a generuje mesačne propagáciu výrobkov šitú na mieru pre zákazníka. Pre modelovanie aktivít vo vnútri stavu použite kľúčové slovo *Do:* nasledované jednou alebo viacerými činnosťami.

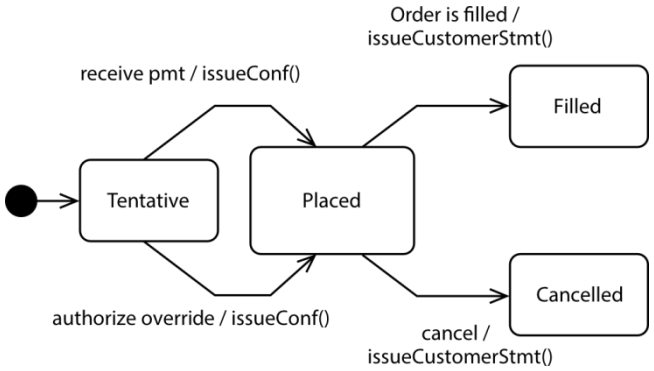


SMD 7: Ikona rozšíreného stavu s činnosťami.

Tieto činnosti budú vykonávané odkedy objekt vstúpi do stavu, až pokým objekt stav neopustí alebo pokým sa činnosť neskončí.

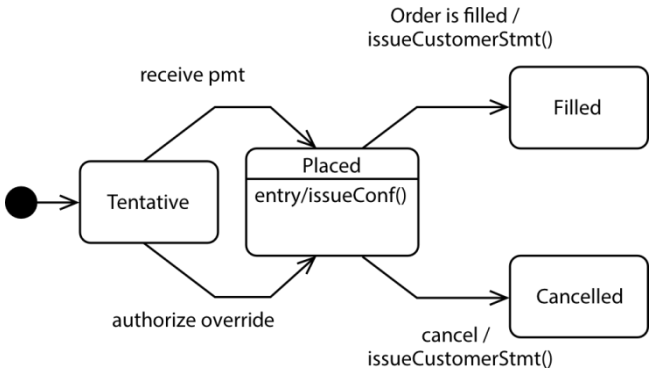
7.7.3.Vstupné a výstupné akcie

Modelovanie prechodov medzi stavmi má častokrát za následok, že viac ako jedna udalosť zmení objekt do rovnakého stavu. Každá z týchto udalostí môže mať korešpondujúcu akciu. Napríklad stavový diagram (obrázok) pre objekt triedy Order hovorí, že objednávka môže prejsť zo stavu Tentative do stavu Placed buď prijatím platby za objednávku alebo obídením autorizácie. Ale obidve udalosti vyžadujú vykonanie rovnakej akcie: vydať potvrdenie objednávky (issueConf()).



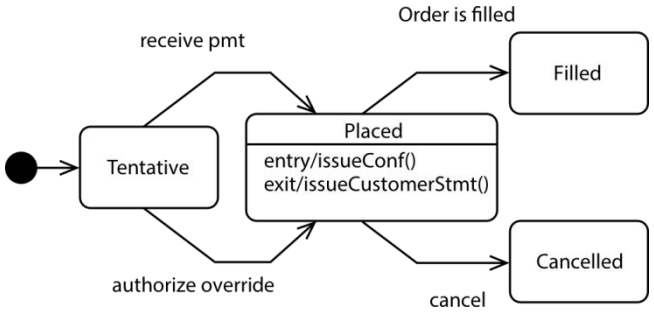
SMD 8: Redundantné akcie vstupujúce a vystupujúce zo stavu Placed objektu triedy Order.

Keď zistíte, že sa určitá akcia sa musí vykonať pri *každej* udalosti, ktorá spôsobí prechod do rovnakého stavu, môžete napísať akciu (akcie) iba jedenkrát ako vstupnú akciu (angl. entry action). Obrázok zobrazuje notáciu vstupnej akcie, entry/akcia(-e). Keď sa redundantná akcia nahradí vstupnou akciou, môžete ju odstrániť z individuálnych šípok udalostí. Toto zjednodušuje diagram a zároveň zachováva rovnaký význam. Diagram by ste prečítali nasledovne: „Vždy keď vstúpiš do tohto stavu, vydaj potvrdenie objednávky.“



SMD 9: Konsolidácia vstupných akcií.

Rovnaké zjednodušenie môžete použiť pre akcie asociované s udalosťami, ktoré opúšťajú stav. Tieto sa nazývajú *výstupné akcie* (angl. exit actions) a modelujú sa rovnakým spôsobom, ako vstupné akcie. Ak sa vrátite späť k obrázku, uvidíte dve udalosti, ktoré opúšťajú stav Placed. Obidve udalosti majú asociovanú rovnakú akciu, issueCustomerStmt(). Obrázok zobrazuje notáciu *exit/akcia(-e)* pridanú do oddelenia s vnútornými prechodmi stavu Placed a odstránenie akcií z šípok udalostí. Aby ste ocenili toto zjednodušenie, porovnajte obrázok s obrázkom.

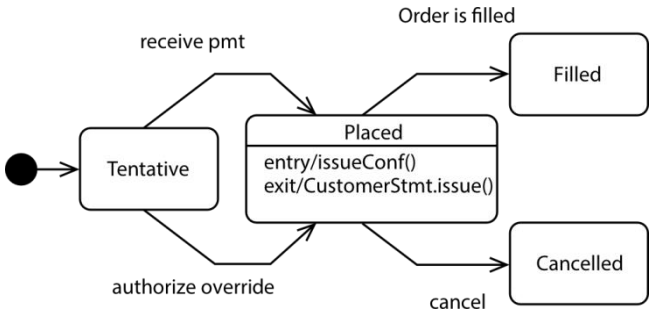


SMD 10: Konsolidácia výstupných akcií.

Notácia vstupných a výstupných akcií poskytuje krásne zjednodušenie stavového diagramu. Pamätajte si, že ich môžete použiť, iba keď sa akcia vykonáva *vždy* keď vstupujete (pre vstupné akcie) do stavu alebo *vždy* keď vystupujete (pre výstupné akcie) zo stavu. Ak existuje čo len jedna výnimka, túto notáciu pre danú akciu nemôžete použiť.

7.7.4.Udalosť zaslanie

Obrázok predstavuje *udalosť zaslanie* (angl. send event). Udalosť zaslanie sa používa, keď objekt v stavovom diagrame potrebuje komunikovať s iným objektom. V stavovom diagrame sa zdroj prichádzajúcich udalostí nezobrazuje, pretože rovnaká udalosť môže prísť z ľubovoľného počtu iných objektov a odpoveď musí byť tá istá. Ale odchádzajúca udalosť musí definovať prijímajúci objekt, t.j. či je to iba jeden objekt alebo ide o vysielanie mnohým objektom. Funguje to rovnakým spôsobom, ako používate váš telefón. Môžete prijímať hovory bez toho, aby ste poznali volajúceho. Ale nemôžete uskutočniť hovor bez čísla, ktoré chcete vytočiť.



SMD 11: Modelovanie akcie vyvolanej na inom objekte.

V príklade na obrázku, pri zrušení objednávky má objekt triedy Order vydať zákazníkovi oznámenie. Ale oznámenie je iný objekt, ktorý sám vybavuje vydávanie. Objednávka mu iba potrebuje povedať, že je čas, aby to urobil. Všetky doteraz namodelované akcie boli akcie toho istého objektu. Pre zobrazenie, že akcia je vyvolaná na inom objekte, jednoducho uveďte názov objektu nasledovaný bodkou umiestnenou pred výrazom akcie. Toto sa často označuje ako *notácia bodky* (angl. dot notation). Pozrite sa na notáciu výstupnej akcie na obrázku, kde sa akcia issue() posiela objektu triedy CustomerStmt.

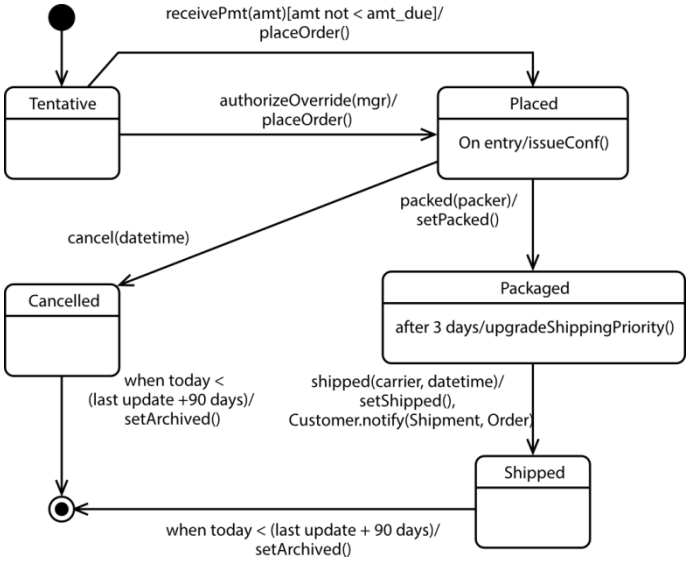
7.7.5.Poradie udalostí

So všetkými týmito udalosťami by sme mohli skončiť pri spletitom zhluku udalostných akcií, vstupných akcii, výstupných akcií a činností. Ako spracovať všetko toto správanie rozumným spôsobom? Keď nastane udalosť, poradie vykonávania sa uskutočňuje takto:

1. Ak sa v aktuálnom stave vykonáva činnosť, preruší sa (kultivovane, pokiaľ je to možné).
2. Vykoná sa exit: akcia(-e).
3. Vykonajú sa akcie asociované s udalosťou, ktorá to celé začala.
4. Vykoná sa entry: akcia(-e) nového stavu.
5. Vykoná sa činnosť alebo činnosti nového stavu.

7.7.6.Modelovanie udalostí

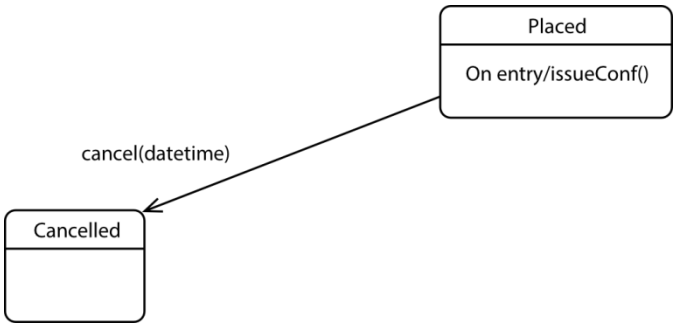
Obrázok ilustruje stavový diagram pre objekt triedy Order. Objednávka je vytvorená v *iniciálnom stave* Tentative. Prechod do stavu Placed by mohli spôsobiť dve udalosti. Zo stavu Placed môže byť objednávka buď zrušená alebo balená alebo pripravená na odoslanie. Po zabalení sa môže odoslať. Potom, čo bola zrušená alebo odoslaná, sa objednávka po 90 dňoch archivuje.



SMD 12: Stavový diagram pre typický objekt triedy Order.

Udalosť volanie

Udalosť volanie (angl. call event) je najbežnejší typ udalosti. V podstate je to volanie operácie na prijímajúcom objekte. Tento typ udalosti predstavuje zlúčenie udalosti a akcie. Samotná udalosť je inštrukcia na vykonanie operácie. Avšak, toto vám nezabráni pridávať ďalšie akcie. Obrázok zobrazuje prechod zo stavu Placed to stavu Cancelled. Prechod je spustený udalosťou cancel(datetime). „cancel(datetime)“ je vlastne signatúra operácie triedy Order.



SMD 13: Udalosť volanie „cancel(datetime)“.

Časová udalosť

Časová udalosť (angl. time event) vyhodnocuje plynutie času ako spúšťač. Z toho vyplýva, že objekt podporuje určitý mechanizmus na monitorovanie plynutia času. Toto môže byť implementované mnohými spôsobmi. Mechanizmus by mohol byť dávkový program (angl. batch program), ktorý v určitých intervaloch obnovuje časový atribút. Udalosť by mohla použiť hlasovací (angl. polling) typ implementácie, kde objekt konštantne kontroluje aktuálny čas.

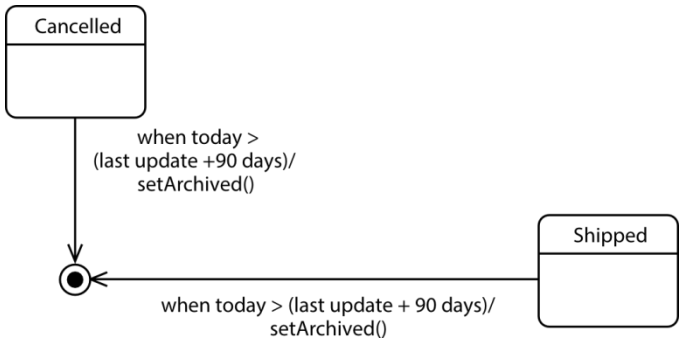
Pre špecifikáciu časového inkrementu pre vyhodnotenie použite kľúčové slovo *after*. Napríklad, pokým je objednávka v stave balená, čaká na odoslanie. Ale ak nebola odoslaná do troch dní, musí sa zvýšiť priorita, aby sme zabezpečili, že odíde načas. Obrázok modeluje internú udalosť v rámci stavu balená nazvanú „after 3 days“. Z toho vyplýva, že v objekte bude kód, ktorý bude sledovať plynutie času aby vedel, kedy inicializovať akciu „upgradeShippingPrioriy()“.

SMD 14: Časová udalosť „after 3 days“.

Udalosť zmeny

Udalosť zmeny (angl. change event) testuje zmeny objektu alebo bod v čase. Použite kľúčové slovo *when* spolu s požadovaným testom. Napríklad, možno by ste chceli vedieť, kedy urobiť výkazy alebo zredukovať faktúry, tak môžete povedať „when 12:00 AM the last day of the month“. Alebo môžete sledovať špecifickú podmienku ako zmena teploty. Potom môžete povedať „when temp > 75 degrees“.

Obrázok reprezentuje udalosti zmeny, ktoré spôsobia archiváciu objednávky. V oboch prípadoch objednávka čaká, dokým sa v priebehu 90 dní nevykoná nad objektom triedy Order žiadna aktivita.



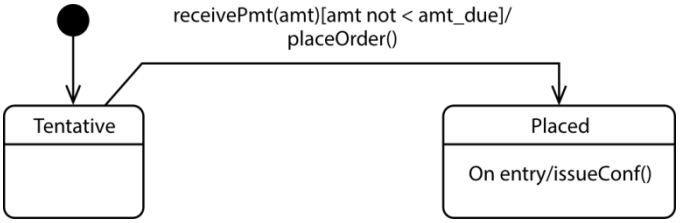
SMD 15: Udalosť zmeny „when today > (last update + 90 days)“.

Pre pripomenutie si všimnite, že táto udalosť je vyhodnocovaná iba pokým je objednávka v stave Cancelled alebo Shipped. Fakt, že udalosť nie je nakreslená z iných stavov znamená, že pokým je objekt v iných stavoch nebude testovať túto podmienku. Zapamätajte si, že to, čo nie je zobrazené v stavovom diagrame vám hovorí takmer toľko, koľko vám hovorí to, čo je zobrazené v diagrame.

Vytváranie podmienených udalostí

Strážiaca podmienka riadi odpoveď na udalosť. Keď nastane udalosť, otestuje sa podmienka. Ak je podmienka pravdivá, vykoná sa korešpondujúci prechod spolu so všetkými asociovanými akciami; inak sa udalosť ignoruje.

Obrázok modeluje jeden z prechodov zo stavu Tentative do stavu Placed. Spúšťacou udalosťou je „receivePmt(amt)“. Ale prijatie platby nespôsobí prechod objednávky do stavu Placed, ak suma platby nie je dostatočná na zaplatenie objednávky. Výsledný efekt udalosti je, že objekt potvrdí udalosť, vyhodnotí efektívnosť udalosti a buď ju príjme alebo odmietne na základe strážiacej podmienky. Ak sa udalosť odmietne, objekt ostáva nezmenený.

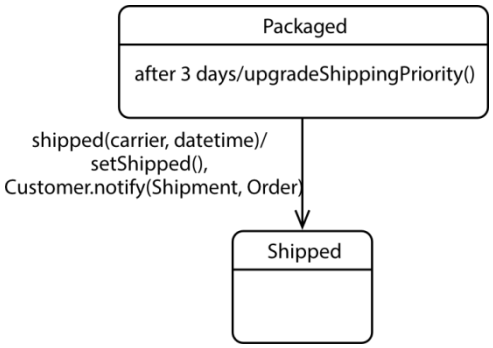


SMD 16: Strážiaca podmienka [amt not < amt_due].

Udalosť zaslanie

Objekty medzi sebou interagujú, aby vykonali prácu. V stavovom diagrame modelujete prečo a kedy by objekty mali vykonať takéto interakcie. Udalosť zaslanie (angl. send event) modeluje fakt, že objekt hovorí inému objektu čo má robiť. Udalosť zaslanie môže byť odpoveďou na udalosť prechodu alebo internú udalosť.

Obrázok modeluje udalosť zaslanie ako časť akcie požadovanej od udalosti prechodu. Keď nastane udalosť shipped(carrier, datetime), spustia sa s prechodom dve akcie: setShipped(), ktorá zmení stav stavu; a zašle sa správa objektu triedy Customer pre vygenerovanie notifikácie skutočnému zákazníkovi, ktorému bola objednávka poslaná.



SMD 17: Udalosť zaslanie Customer.notify(Shipment, Order).

Strážiace podmienky ako udalosti

Strážiaca podmienka sa v skutočnosti môže použiť aj na spustenie udalosti. Z toho jednoducho vyplýva, že objekt sleduje, kedy sa splní podmienka. Napríklad, váš bankový účet monitoruje svoj stav vždy keď sa počítajú vklady a výbery. Ak v ľubovoľnom čase stav účtu klesne pod nulu, účet si nenechá ujsť šancu poslať vám poplatok za prečerpanie účtu.

7.7.7.Nadradené a podradené stavy

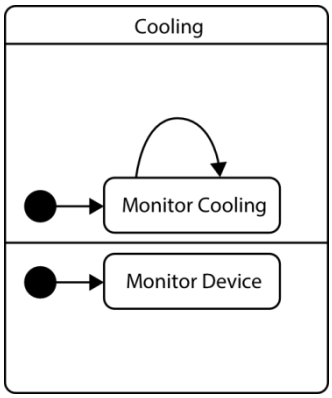
Modelovanie si často vyžaduje pozerať sa na problém z rôznych pohľadov. Vysokoúrovňové pohľady model zjednodušujú. Nízkoúrovňové pohľady sa zameriavajú na detaily problému. Stavový diagram podporuje koncept vnorených stavov, ktoré umožňujú vysoko- aj nízkoúrovňové pohľady na správanie objektu a jeho stavy.

Nadradený stav (angl. superstate) je jednoducho stav, ktorý je rozšírený, aby zobrazil viac detailov. Ikona stavu sa roztiahne a detaily sú reprezentované ako jeden alebo viac stavových diagramov vo vnútri nadradeného stavu. Názov stavu sa umiesti hore. Nadradené stavy reprezentujú vysokoúrovňový pohľad na zložitú situáciu. Umožňujú sa sústrediť na väčší, všeobecnejší problém bez straty detailov. Podradené stavy sa umiestnia dovnútra rozšíreného nadradeného stavu.

Podradený stav (angl. substate) je stav vo vnútri stavu, nižšia úroveň detailu v rámci stavu. Napríklad, auto môže byť v stave idúce dopredu. V rámci *nadradeného stavu idúce dopredu* sú podradené stavy *idúce dopredu na prvom rýchlostnom stupni*, *idúce dopredu na druhom rýchlostnom stupni*, atď. Podradené stavy poskytujú nízkoúrovňový pohľad na element modelu tak, že môžete adresovať špecifické záležitosti individuálne a z hľadiska ich interakcií a vzájomných závislostí. Tento detailnejší pohľad vám taktiež umožňuje upozorniť na súbežné stavy a sústrediť sa na to, ako riadiť rozdelenie a zlúčenie súbežných stavov.

Na ilustráciu týchto konceptov použijem známy príklad – termostat. Ešte viac to zjednoduším a budeme sa dívať iba na jeho zodpovednosti spojené s chladením. Termostat je typicky objekt riadiaceho typu. Jeho prácou je určovať prácu iných objektov, podobne ako aplikácia určuje správanie obrazovky a prístupu do databázy.

Obrázok modeluje nadradený stav Cooling s dvomi súbežnými podradenými stavmi. V tomto príklade sú oba podradené stavy iniciálne stavy. Diagram hovorí, že keď termostat vstúpi do stavu Cooling, rozdelí sa na dva súbežné podradené stavy, t.j. teraz robí dve veci súčasne: monitoruje postup procesu chladenia a monitoruje chladiace zariadenie pre prípad vzniku problémov. Dva podradené stavy začínajú okamžite po vstupe do stavu Cooling.



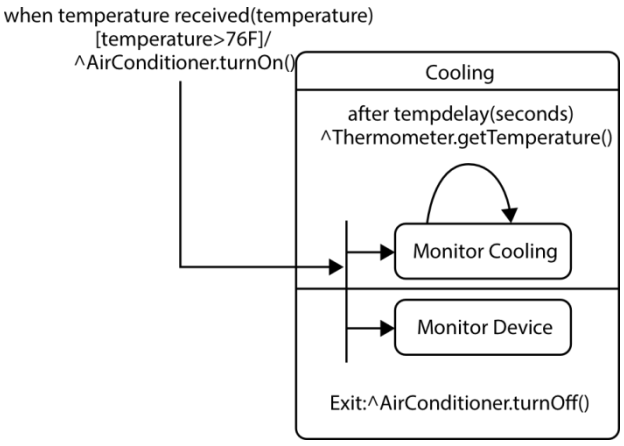
SMD 18: Nadradený stav Cooling s dvomi podradenými stavmi – Monitor Cooling a Monitor Device.

Je tiež možné spustiť podradené stavy prechodovými udalosťami ako volanie alebo časová udalosť. V tomto prípade by sa prechodová udalosť rozšírila až do nadradeného stavu a ukázala by priamo na podradený stav. Toto častokrát zahŕňa rozdelenie riadenia, ktoré pokryjem v ďalšom texte.

Rozdelenie riadenia

Rozdelenie riadenia znamená, že z jedného prechodu chcete pokračovať viacerými úlohami. Toto je presne rovnaký koncept, ktorý ste sa naučili pri diagrame činností. Rozdelenie kontroly sa zobrazuje jedným prechodom rozdeleným do viacerých šípok, ktoré ukazujú na viaceré stavy alebo podradené stavy. Rozdelenie modelujeme rozvetvovacím pruhom (angl. fork bar), ktorý ste používali v diagrame činností a je ilustrovaný na obrázku.

Spojenie riadenia môže byť modelované viacerými prechodovými šípkami ukazujúcimi na synchorinzačný pruh, ako ste videli pri učení diagramu činností. Synchronizácia na obrázku nie je zobrazená.



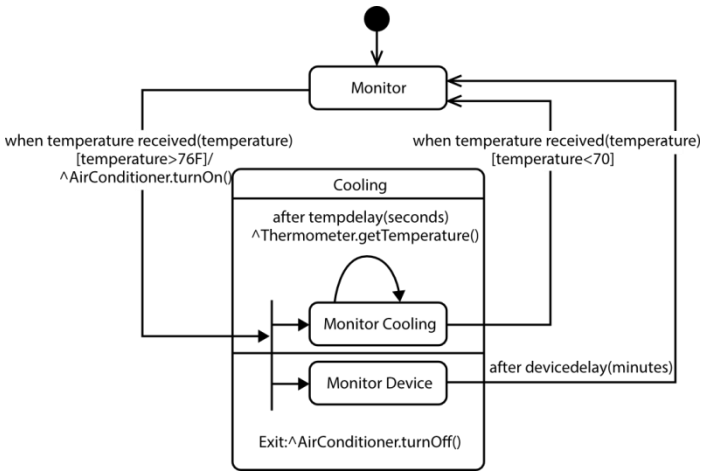
SMD 19: Rozdelenie riadenia.

Súbežnosť

Povolením viacerých stavových diagramov vo vnútri stavu, UML podporuje súbežnosť v rámci stavu. Pre modelovanie súbežnosti jednoducho rozdeľte oddelenie interných prechodov nadradeného stavu do toľkých separátnych oddelení koľko potrebuje (jeden pre každý podradený stavový diagram).
V príklade s termostatom Thermostat vykonáva súčasne dve úlohy: monitoruje chladiace zariadenie a sleduje, či nenastali problémy so zariadením. Takže oddelenie interných prechodov stavu Cooling sa pomocou čiary rozdelí na dve (oddelenia).

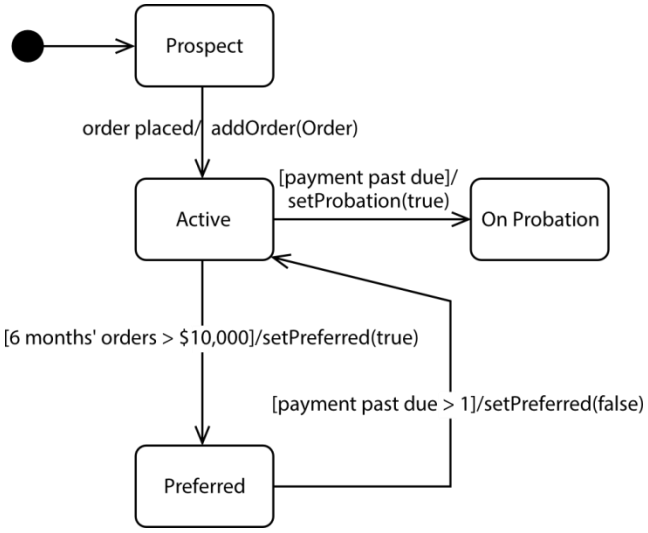
Všimnite si, že v tomto konkrétnom príklade poskytuje každý podradený stav rôzne prechody smerujúce von z nadradeného stavu. Podradený stav Monitor Cooling sleduje udalosť „when temperature received(temperature) [temperature < 70]“. Ak túto udalosť zachytí, spôsobí prechod (termostatu) zo stavu Cooling naspäť na monitorovanie teploty. Tak čo sa stane s druhým stavom? Toto vyrieši definícia podradeného stavu. Podradený stav je stav vo vnútri stavu – v tomto prípade Monitor Device vo vnúri Cooling. Ak Thermostat opustí stav Cooling, podľa definície opustí aj stav Monitor Device. Toto je ilustrované na obrázku.

Rovnako to platí aj v prípade, ak stav Monitor Device zachytí udalosť, na ktorú čakal.

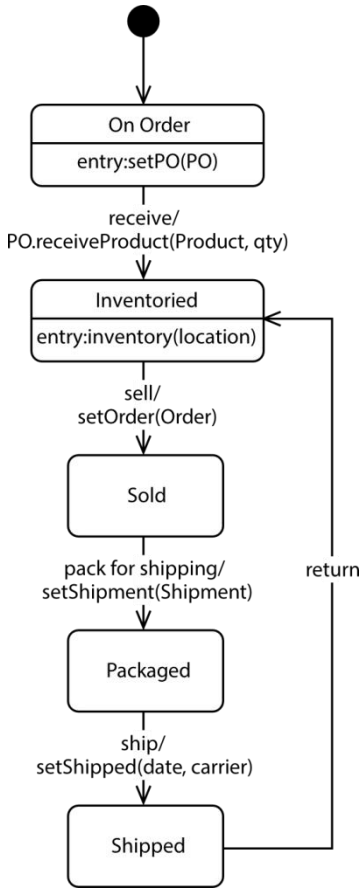


SMD 20: Viaceré prechody vychádzajúce z nadradeného stavu.

7.7.8.Príklady



SMD 21: Stavový diagram.

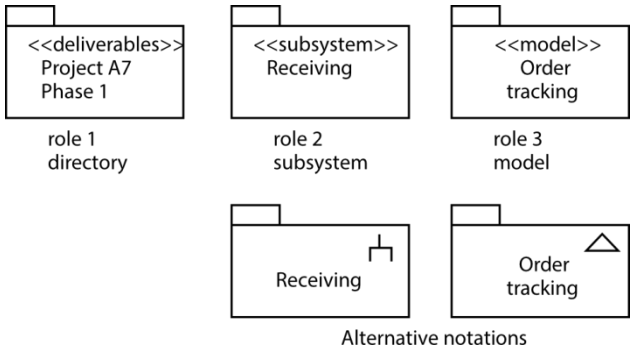


SMD 22: Stavový diagram pre objekt triedy Product.

7.8. Diagram balíkov

Balík sa modeluje v diagrame balíkov (angl. Package diagram) ikonou katalógu, ako majú tri balíky na obrázku. Na obrázku je taktiež ilustrovaný fakt, že balíky môžu byť použité na tri rôzne účely. Môžu byť použité na organizáciu niektorých alebo aj všetkých diagramov, ktoré vytvoríte počas projektu. Diagramy môžete umiestniť do rôznych balíčkov presne tak, ako by ste umiestnili súbory do rôznych adresárov na vašom počítači. Adresáre a balíky pomenujte tak, aby indikovali účel súborov, ktoré obsahujú. Obrázok túto rolu ilustruje balíkom vľavo, balíkom pre projekt A7, fázu 1.

Balíky môžu obsahovať hocijaké logické elementy modelu, ktoré ste sa doteraz naučili, ako diagramy prípadov použitia, sekvenčné diagramy, diagramy tried a dokonca iné balíky. V skutočnosti väčšina modelovacích nástrojov poskytuje navigačný mechanizmus založený na balíkoch, ktorý vyzerá a funguje rovnako, ako adresárová štruktúra. Pretože toto použitie balíkov je tak všeobecné, môžete použiť prakticky ľubovoľný stereotyp, aby ste vysvetlili, ako používate konkrétny balík.



PCD 1: Tri použitia balíkov: adresáre, podsystémy a modely.

Balík môže reprezentovať *podsystém*, ako podsystém Receiving na obrázku. Podsystém je stereotyp definovaný UML, ktorý identifikuje súdržnú podmnožinu celého systému. Napríklad, Inventory Control System môže byť okrem iného organizovaný do podsystému Receiving a podsystému Shipping.

Elementy umiestnené do balíka typu podsystém sú automaticky viditeľné iba v rámci balíka. Avšak viditeľnosť individuálnych elementov modelu v rámci balíka môže byť definovaná ako verejná, súkromná alebo chránená. Každý balík predstavujúci podsystém musí mať aspoň jedno verejné rozhranie (t.j. aspoň jednu triedu s verejným rozhraním).

Tretie použitie balíkov sa nazýva *model*. Model je taktiež stereotyp definovaný UML, ktorý sa podobá na podsystém v tom, že obsahuje súdržnú množinu elementov systému. Rozdiel je v tom, že model sa sústredí na predmet alebo typ správania v rámci systému. Napríklad, informácia o predmete Order Tracking tretieho balíka na obrázku sa s veľkou pravdepodobnosťou objaví vo väčšine podsystémov Inventory Control. Pretože sa model sústredí na jeden predmet, nebude obsahovať žiadne systémové elementy, ktoré nepomáhajú vysvetliť predmet.

7.8.1. Menný priestor

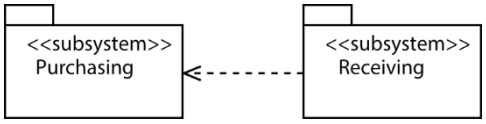
Všetky tieto typy balíkov poskytujú samostatné menné priestory pre elementy modelu, ktoré obsahujú vrátane iných balíkov. Pomenovanie elementov v rámci balíka vyžaduje dve informácie: názov elementu a typ elementu. Napríklad, balík môže obsahovať niečo nazvané Produkt typu Trieda a niečo nazvané Produkt typu Stavový diagram. Názvy elementov rovnakého typu musia byť unikátne

v rámci balíka, ale elementy rôznych typov nemusia mať unikátne názvy. Balík by nemohol obsahovať dve položky nazvané Produkt, ktoré by boli typu trieda.

Elementy modelu v rôznych balíkoch môžu mať rovnaké názvy. Avšak keď sa dva elementy použijú spolu, musia byť kvalifikované názvom balíka, ktorom sa nachádzajú. Plne kvalifikované názvy elementov používajú notáciu *balík :: element*, napríklad *Receiving :: Product* a *Shipping :: Product*.

7.8.2.Notácia balíkov a diagramov balíkov

Ikona balíka vyzerá ako štítkový katalóg. Balíky sa na seba odvolávajú použitím notácie závislosti, prerušovanej šípky. Príklad na obrázku čítajte ako „podsystém Receiving závisí od alebo potrebuje pomoc od podsystému Purchasing“.



PCD 2: Ikona balíka a notácia závislosti.

Stereotypy balíka

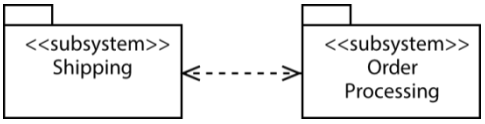
Na obrázkoch obsahuje každá ikona balíka stereotyp ako <<subsystem>> alebo <<deliverables>>. Do balíka môžete vložiť takmer čokoľvek čo chcete, takže opis balíka si často vyžaduje trochu vysvetlenia. Stereotyp vám umožňuje charakterizovať obsah balíka a stále poskytuje špecifické pomenovanie jeho obsahu. Napríklad, balík Receiving je charakterizovaný ako podsystém. Toto nám zabráni interpretovať ho ako adresár obsahujúci dokumenty alebo nejaké iné prostriedky okrem tried podsystému.

Napriek tomu buďte opatrní. Stereotypy nie sú súčasťou názvu balíka, takže vám nepomôžu vytvárať unikátne názvy. Dva balíky na rovnakej úrovni nazvané <<documentation>> Receiving a <<subsystem>> Receiving by boli konfliktné a pravdepodobne by neboli povolené vo väčšine modelovacích nástrojov. Na druhej strane, ak sú samotné balíky obsiahnuté v iných balíkoch, potom sú kvalifikované ich nadradenými balíkmi, čo ich robí unikátnymi. Musíte však skontrolovať, ako sú tieto pravidlá implementované vo vašom modelovacom nástroji.

Balíková závislosť

Obrázok tiež zobrazuje prerušovanú šípku z balíka Receiving do balíka Purchasing. Vzťah závislosti znamená, že aspoň jedna trieda v balíku musí komunikovať s aspoň jednou triedou v inom balíku. Závislosť na obrázku by mohla znamenať, že trieda Receipt v balíku Receiving (Receiving :: Receipt) potrebuje mať možnosť získať detaily triedy PurchaseOrder v balíku Purchasing (Purchasing :: PurchaseOrder), aby mohla potvrdiť prichádzajúce produkty.

Obojsmerná závislosť (indikovaná hrotom šípky na oboch koncoch prerušovanej čiary) je úplne platná. Obrázok zobrazuje príklad , kde podsystém Shipping môže potrebovať aktualizovať objednávku v podsystéme Order Processing. Podsystém Order Processing ale taktiež môže potrebovať kontrolovať stav podsystému Shipment, ktorý obsahuje objednané produkty.



PCD 3: Obojsmerná závislosť.

Pre jednoduchosť, všetky ostatné závislosti ilustrované v tejto kapitole budú iba jednosmerné.

Stereotypy závislosti

Balíková závislosť môže byť označená stereotypom, ktorý opisujte podstatu závislosti. UML definuje dva stereotypy závislosti, <<import>> a <<access>>. Stereotyp <<import>> na obrázku znamená, že balík Receiving k sebe počas behu pridáva triedu Purchasing (v tomto prípade triedu PurchaseOrder), čo umožňuje *interné odkazy* (odkazy v rámci balíka) na triedu bez špecifikovania názvu zdrojového balíka.

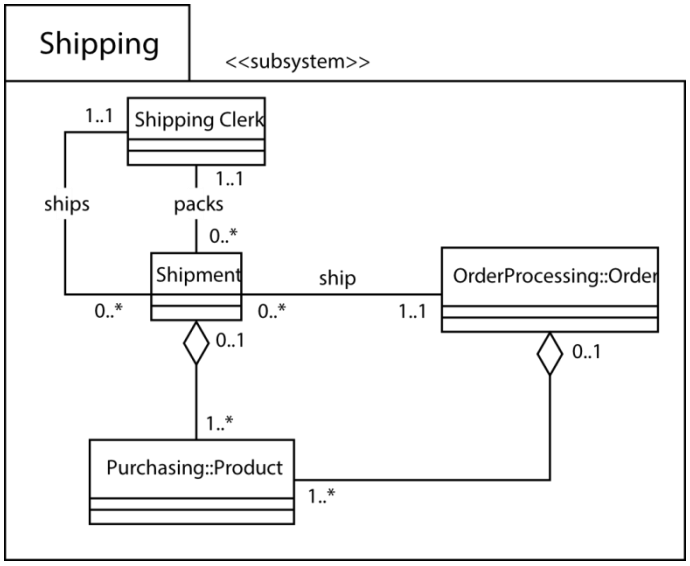
PCD 4: Stereotyp <<import>>.

Stereotyp <<access>> na obrázku hovorí, že podsystém Shipping bude chcieť komunikovať s podsystémom Receiving, ale v skutočnosti počas behu nevytiahne triedy z Receiving do Shipping. Počas behu by ste potom očakávali, že uvidíte nejaký objekt z podsystému Shipping vykonávať volania v rozhraní podsystému Receiving.

PCD 5: Stereotyp <<access>>.

Elementy modelu v balíku

Jedným z najbežnejších použití balíka je uchovávanie vašich diagramov. Vo väčšine modelovacích nástrojov poskytujú balíky mechanizmus vnárania (t.j. balík môže obsahovať iné balíky, ktoré zase môžu obsahovať diagramy). Balíky v tejto schéme predstavujú (v tomto poradí) systémy, podsystémy a diagramy. Schéma môže obsahovať toľko úrovní, koľko daný problém vyžaduje. Obrázok zobrazuje príklad balíka Shipping, ktorý uchováva diagram tried, ktorý podporuje funkcie podsystému Shipping.

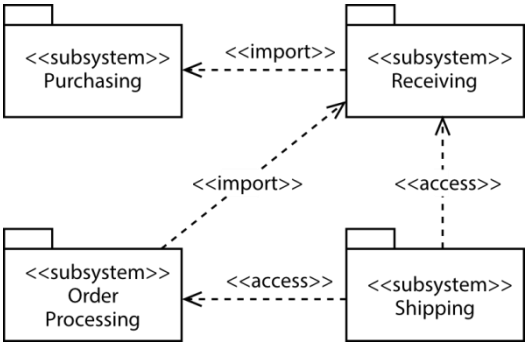


PCD 6: Balík obsahujúci diagram tried.

Bežnejší spôsob takejto reprezentácie v modelovacom nástroji je otvoriť balík a dať diagram do nového okna. Konceptuálne je diagram je umiestnený v balíku ako som to reprezentoval na obrázku. Ale nástroj to nereprezentuje týmto spôsobom.

Obrázok taktiež zobrazuje dva príklady vzťahu import. Triedy Order a Product používajú kvalifikujúcu notáciu *balík :: element*. Táto notácia vám hovorí, že dve triedy pochádzajú z pomenovaného balíka (t.j. trieda Order je importovaná z balíka OrderProcessing a trieda Product je importovaná z balíka Purchasing). Toto jasne identifikuje fakt, že trieda je definovaná v inom balíku, ale odkazuje sa na ňu v tomto balíku.

7.8.3.Príklady



PCD 7: Diagram balíkov.

7.9. Diagram súčiastok

Diagram súčiastok (diagram komponentov, angl. Component diagram) modeluje fyzickú implementáciu softvéru. Diagram rozmiestnenia modeluje fyzickú architektúru hardvéru. Kombinácia týchto diagramov modeluje integráciu a distribúciu vášho aplikačného softvéru cez hardvérovú implementáciu.

Presne tak ako diagramy tried opisujú organizáciu a zámer vášho softvérového návrhu, súčiastky reprezentujú fyzické implementácie vášho softvérového návrhu. Účelom diagramu súčiastok je

definovať softvérové moduly a vzťahy medzi nimi. Každý modul je kus kódu, ktorý je uložený v pamäti na kuse hardvéru. Každá súčiastka musí definovať rozhranie, ktoré umožňuje iným súčiastkam s touto súčiastkou komunikovať. Rozhranie a vnútorná implementácia súčiastky sú zapuzdrené do tried, ktoré vytvárajú súčiastku.

UML rozdeľuje súčiastky do troch rozsiahlych kategórií:

- Súčiastky rozmiestnenia (angl. deployment components), ktoré sú potrebné pre chod systému.
- Súčiastky produktu práce (angl. work product components) zahŕňajú modely, zdrojový kód a dátové súbory použité na vytvorenie súčiastok rozmiestnenia.
- Súčiastky vykonania (angl. execution components) sú súčiastky vytvorené počas behu aplikácie.

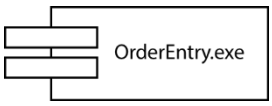
Súčiastky môžu závisieť jedna od druhéj. Napríklad, spustiteľný súbor (.exe) môže vyžadovať prístup k dynamicky linkovanej knižnici (.dll) alebo klientska aplikácia môže závisieť od aplikácie na strane servera, ktorá zase závisí od rozhrania databázy.

Súčiastky môžu závisieť od tried. Napríklad, pre skompilovanie spustiteľného súboru potrebujete kompilátoru dodať zdrojové triedy.

S danými elementmi, súčiastkou, rozhraním súčiastky a závislosťami môžete opísať fyzickú implementáciu vášho systému z hľadiska softvérových modulov a vzťahov medzi nimi.

7.9.1.Notácia pre súčiastky a závislosti súčiastok

Ikonu súčiastky modelujeme ako obdĺžnik s typickým obrázkom súčiastky s dvomi portami. Názov umiestňujeme do stredu ikony, ako na obrázku.



(UML 1.4)

(UML 2.0)

CPD 1: Ikona súčiastky.

Stereotypy súčiastky

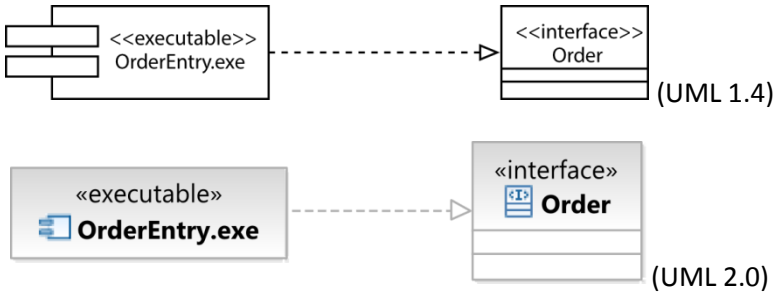
Stereotypy súčiastky poskytujú vizuálne pomôcky k role, ktorú hrá súčiastka v implementácii. Niektoré bežné stereotypy súčiastok zahŕňajú:

- <<executable>>: Súčiastka, ktorá je vykonávaná na procesore (program).
- <<library>>: Množina zdrojov, na ktoré sa odkazuje program počas behu.
- <<table>>: Databázová súčiastka, ku ktorej pristupuje program.
- <<file>>: Typicky reprezentuje dáta alebo zdrojový kód.
- <<document>>: Dokument ako stránka vložená do web stránky.

Tieto stereotypy sa odvolávajú na klasifikátory (angl. classifiers, implementácie tried definovaných v skoršej fáze procesu) a artefakty implementácie klasifikátorov, ako zdrojový kód, binárne súbory a databázy.

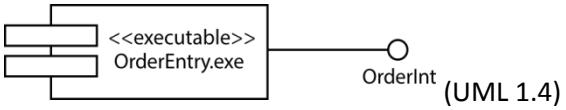
Rozhranie súčiastky

Rozhranie súčiastky môžeme modelovať dvomi spôsobmi. Prvým zo spôsobov je použiť triedu so stereotypom <<interface>> spojenú so súčiastkou pomocou šípky realizácie, ako na obrázku. Šípka realizácie vyzerá ako symbol zovšeobecnenia s prerušovanou čiarou. *Realizovať* rozhranie znamená aplikovať ho na niečo skutočné, ako napríklad program.



CPD 2: Notácia rozhrania použitím triedy a stereotypu.

Druhou bežnejšou technikou je použiť „lízanku“ spojenú so súčiastkou plnou čiarou, ako na obrázku. Ak sa pozriete do príkladov v špecifikácii UML, krúžok na konci lízanky je veľmi malý. Toto je malý odklon od typickej notácie využívanej v modelovacích nástrojoch.

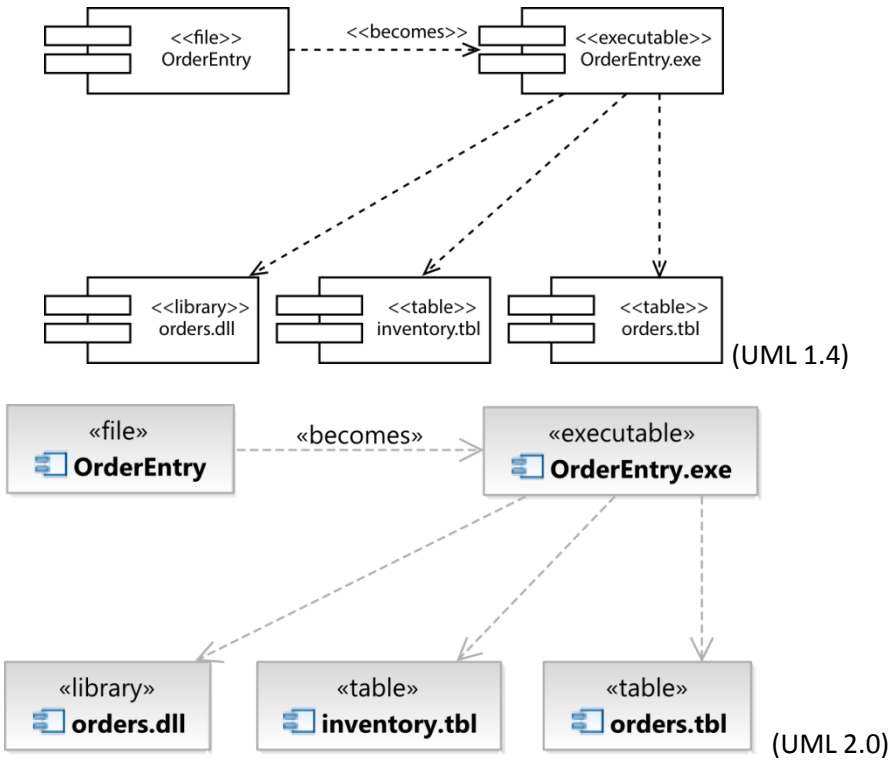


CPD 3: Notácia rozhrania použitím lízanky.

Rozhranie implementované súčiastkou je v skutočnosti implementované triedou vo vnútri súčiastky, takže rozhranie by už malo byť definované vo vašom diagrame tried. Súčiastky môžu taktiež implementovať toľko rozhraní, koľko je potrebné. Počet a exaktné typy rozhraní určujú triedy implementované súčiastkou.

Závislosti súčiastok

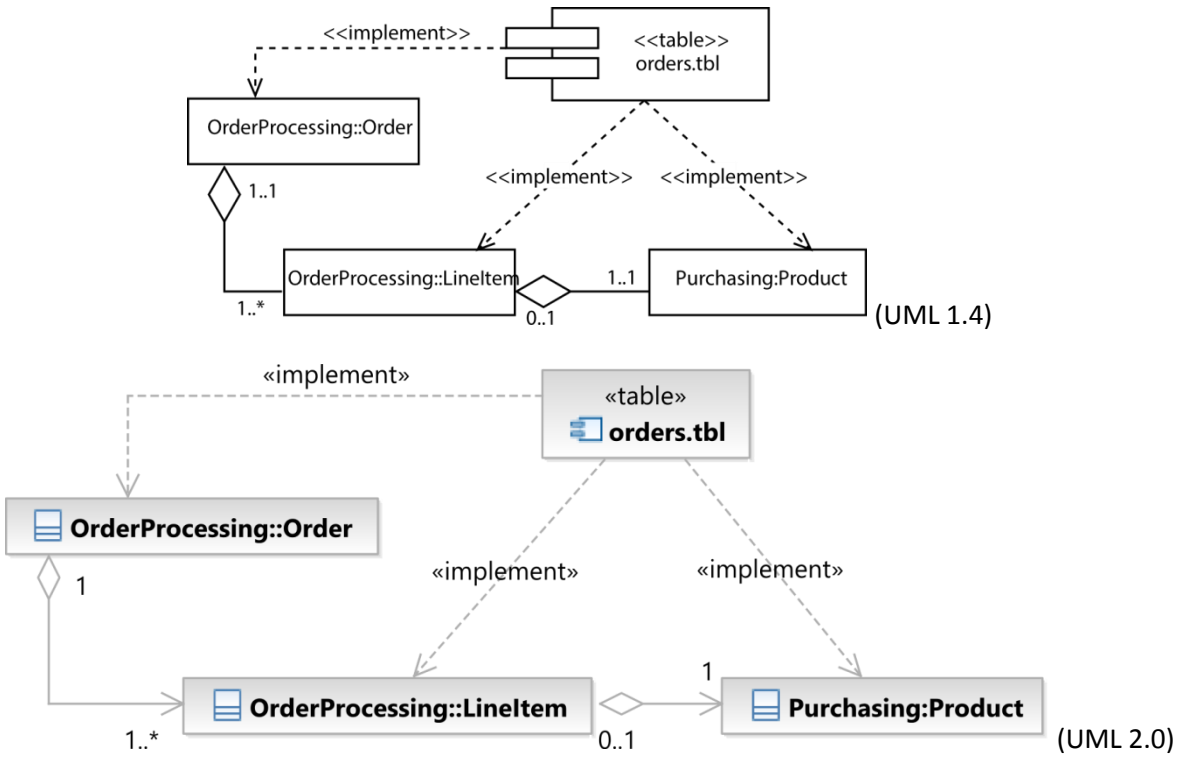
Závislosti medzi súčiastkami sa kreslia prerušovanou šípkou zo závislej súčiastky do súčiastky, od ktorej potrebuje pomoc. Na obrázku , súčiastka OrderEntry závisí od súčiastky OrderEntry.exe. UML stereotyp <<becomes>> znamená, že súbor OrderEntry sa doslova stáva počas behu vykonateľnou súčiastkou OrderEntry. OrderEntry môže byť kód umiestnený na pamäťovom zariadení. Počas behu sa nahrá do pamäte a môže sa dokonca aj skompilovať. Potom počas behu by súčiastka OrderEntry.exe závisela od troch ďalších súčiastok: orders.dll, inventory.dll, a orders.tbl.



CPD 4: Závislosti súčiastok a stereotypy závislostí.

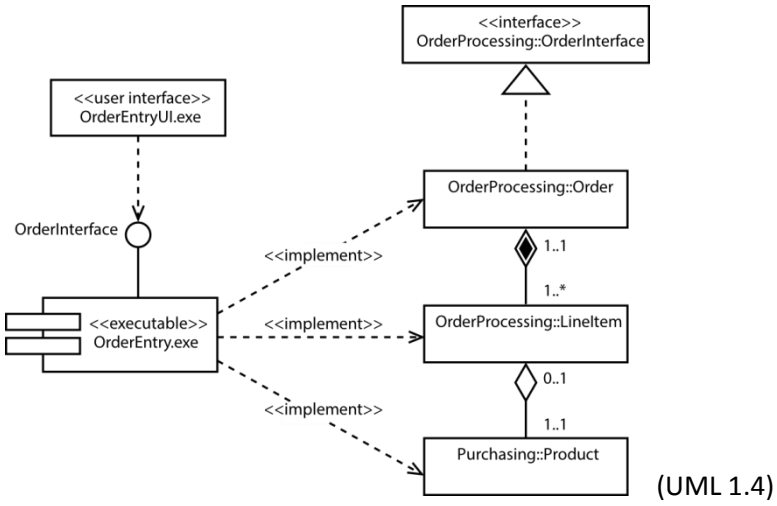
7.9.2.Mapovanie logického návrhu na fyzickú implementáciu

Vytváranie súčiastok z tried so sebou prináša rozhodnutia o tom, ako zostaviť tieto triedy do súdržných celkov. Rozhrania tried v súčiastke vytvárajú rozhranie súčiastky. Obrázok zobrazuje súčiastku databázovej tabuľky, orders.tbl, ktorá implementuje triedy, ktoré definujú objednávku, konkrétne Order, LineItem a Product a ich asociácia.



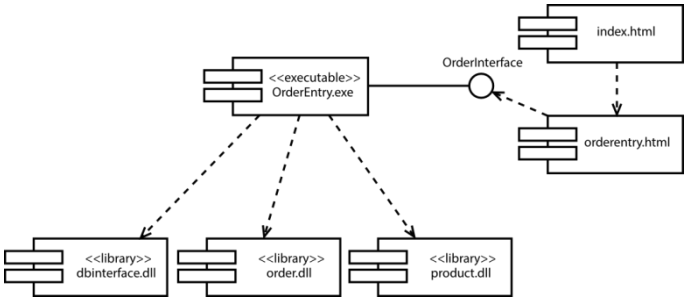
CPD 5: Súčiastka je vytvorená z tried.

V podobnom duchu hlavný program v aplikácii môže implementovať niektoré alebo všetky kľúčové triedy v logickom modeli. Pre vytvorenie programu na obrázku, skompilujete triedy do jedného vykonateľného programu.



CPD 6: OrderEntry.exe je vytvorený z viacerých zdrojových tried.

Pomerne často súčiastka avšak pozostáva z jednej triedy implementovanej ako spustiteľný program, súbor, knižnica, tabuľka alebo dokument. Na obrázku sa OrderEntry.exe odkazuje na množinu knižničných súčiastok namiesto kompilácie tried do jednej súčiastky. Používateľské rozhranie aplikácie je rozdelené do dvoch HTML súčiastok. Výsledkom je modulárnejší návrh.

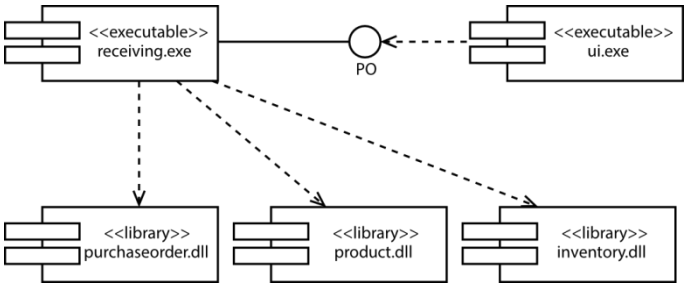


(UML 1.4)

CPD 7: Jedna trieda = jedna súčiastka.

Súčiastky môžu byť organizované do balíkov rovnako, ako aj iné diagramy a elementy modelu. Toto môže byť veľmi užitočné pri manažovaní distribúcie aplikácie. Výsledkom je adresár obsahujúci všetky softvérové elementy potrebné na implementáciu systému alebo podsystému reprezentovaného balíkom.

7.9.3.Príklady



(UML 1.4)

CPD 8: Diagram súčiastok.

7.10. Diagram rozmiestnenia

Diagram rozmiestnenia (angl. Deployment diagram) opisuje fyzické zdroje podobne, ako diagram tried opisuje logické zdroje. Diagram rozmiestenia sa zameriava na uzly, na ktorých bude bežať váš softvér.

Každý *uzol* (angl. node) je fyzický objekt, ktorý reprezentuje zdroj spracovávania. Najčastejšie je to počítač určitého druhu, ale môže to byť aj ľudský zdroj pre manuálne spracovanie. Každý uzol obsahuje (alebo je zodpovedný za) jednu alebo viacero softvérových súčiastok alebo objektov. Softvérové súčiastky na rôznych uzloch môžu komunikovať cez fyzické *asociácie* medzi uzlami.

Účelom diagramu rozmiestnenia je reprezentovať statický pohľad na implementačné prostredie. Úplný opis systému bude pravdepodobne obsahovať niekoľko rôznych diagramov rozmiestnenia, každý zameraný na iný aspekt systémového manažmentu. Napríklad:

- Jeden diagram môže byť zameraný na to, ako sú softvérové súčiastky distribuované. Napríklad, kde je umiestnený zdrojový kód a kam sa odosiela na implementáciu.
- Ďalší diagram môže modelovať, ako sa spustiteľný súbor načíta z jedného uzla na druhý, kde sa v skutočnosti spustí.
- Pre viacvrstvové aplikácie by diagram rozmiestnenia modeloval distribúciu vrstiev aplikácie, ich fyzické prepojenia a ich logické dráhy komunikácie.

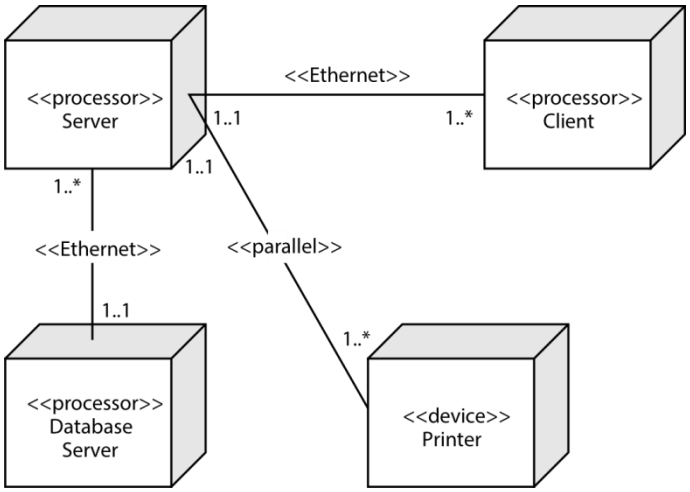
7.10.1. Notácia diagramu rozmiestenia

Schéma pre tieto fyzické diagramy by pre vás mala byť stále jasnejšia (t.j. zdroje a prepojenia). Rovnako ako diagram súčiastok a diagram balíkov, tak aj diagram rozmiestnenia má dva typy elementov: uzly (zdroje) a asociácie (prepojenia).

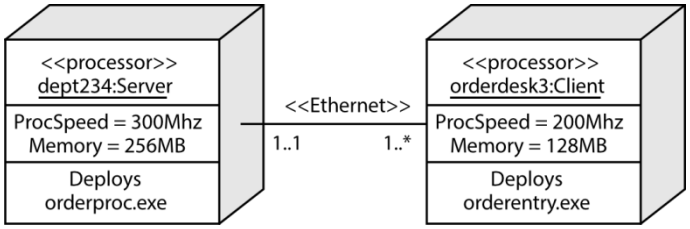
Ikona uzla sa kreslí ako štvorboký hranol (tieňovanie nie je nevyhnutné). Obrázok modeluje štyri typy uzlov: Server, Client, Database Server a Printer. Čiary medzi uzlami sú fyzické *asociácie*, ktoré sú reprezentované plnou čiarou z jedného uzla do druhého. Pre definovanie počtu uzlov použite notáciu násobnosti na oboch koncoch asociácie. Napríklad obrázok hovorí, že každý Server je pripojený na jeden alebo viac uzlov Client, a každý uzol Client je pripojený na práve jeden uzol Server.

Pomenovávanie asociácií uzlov predstavuje zaujímavý problém. Pretože všetky asociácie sú fyzické prepojenia, všetky by mohli dostať rovnaký názov „connects to“. Namiesto toho možno chcete použiť na opis typov prepojení stereotypy. Obrázok hovorí, že uzol Server a uzly Client sú spojené pripojením Ethernet použitím stereotypu <<Ethernet>>.

Uzol je klasifikátor (rovnako ako aj triedy, prípady použitia a súčiastky), takže môže mať atribúty a špecifikovať správanie z hľadiska spustiteľných súborov, ktoré rozmiestňuje. Obrázok zobrazuje pohľad na diagram rozmiestnenia na objektovej úrovni. Diagram na objektovej úrovni modeluje inštancie každého uzla rovnako, ako objektový diagram modeluje reálne entity. Názov vrchného oddelenia identifikuje názov a typ uzla rovnako, ako aj voliteľný stereotyp. Oddelenie atribútov v strede definuje vlastnosti uzla. Oddelenie operácií naspodku definuje súčiastky, ktoré bežia na uzle.



DPD 1: Diagram rozmiestnenia so štyrmi uzlami a tromi asociáciami.



DPD 2: Diagram rozmiestnenia na objektovej úrovni.

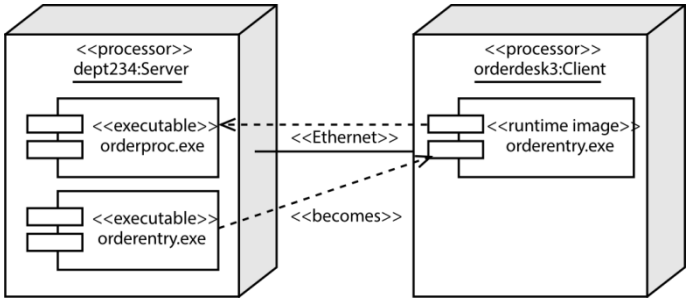
Kreslite diagram rozmiestnenia aj keď každý uzol vo vašej fyzickej architektúre je triedou v diagrame tried. Každý uzol spĺňa špecifický účel. Každý uzol má komunikačné asociácie s inými uzlami, ktoré reprezentujú fyzické prepojenia, ktoré podporujú komunikáciu.

Diagramy rozmiestnenia môžu taktiež fungovať ako sieťové diagramy pre ilustráciu štruktúry vašej siete. Diagram rozmiestnenia na objektovej úrovni môže fungovať ako špecifikácia požiadaviek na každý uzol, ktoré definujú nároky na pamäť, procesor a dátové úložisko.

7.10.2. Mapovanie softvérových súčiastok na architektúru

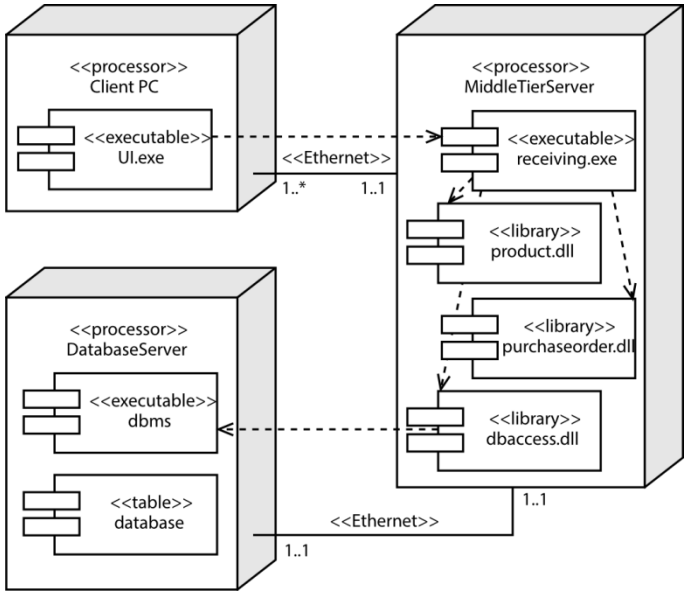
Zaužívanejšou technikou pre modelovanie súčiastok na uzloch je skombinovať notácie súčiastok a uzlov z dvoch fyzických diagramov. Pre zobrazenie obmedzenia modelujte ikony súčiastok dovnútra roztiahnutých uzlov. Pre zobrazenie logickej komunikácie medzi súčiastkami, nakreslite prerušovanú šípku závislosti presne tak, ako v diagrame súčiastok.

Na obrázku je orderentry.exe umiestnený na serveri, ale v čase vykonávania sa načíta na klienta. Túto migráciu v čase vykonávania špecifikuje stereotyp <<becomes>> . Po načítaní spustiteľného súboru, tento závisí od pomoci od orderproc.exe. Všimnite si, že toto by som mohol rovnako jednoducho nakresliť na úrovni tried. Ale tu modelujem fakt, že logický návrh reprezentovaný triedami bol v skutočnosti implementovaný v tejto fyzickej architektúre.

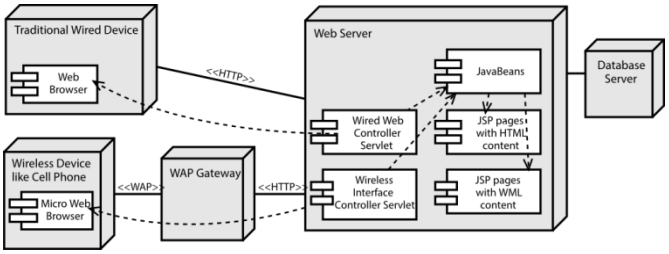


DPD 3: Kombinácia diagramu súčiastok a diagramu rozmiestnenia.

7.10.3. Príklady



DPD 4: Diagram rozmiestnenia.



DPD 5: Diagram rozmiestnenia – Model 2 Architecture.