UPLATNENIE PRINCÍPOV OBJEKTOVÉHO MODELOVANIA PRI BUDOVANÍ INFORMAČNÝCH SYSTÉMOV

Iveta Kremeňová
Roman Súlovec

Žilinská univerzita v Žiline, Fakulta prevádzky a ekonomiky dopravy a spojov, Katedra spojov, krem@fpedas.utc.sk, Sulovec@fpedas.utc.sk

Abstract

During the last decade the importance of object oriented modeling methods in the area of the information system development dramatically increased. There is a lot of new methods but the most known are particularly UML, Booch, Object Modeling Technique, OOSE, Shlaer/Mellor, Business Object Notation, Business Object Relation Modeling, etc. This article deals with the Unified Modeling Language, that could be taken as an integrating and communication tool in the process of information system development. It is a collection of tools used in that process. UML distinguish tools for modeling of the static part of the system and the dynamic one.

1. Vývoj modelovania, objektovo orientované modelovanie

Prvou metódou pri budovaní informačných systémov, ktorá našla širšie uplatnenie, bola metóda štruktúrovanej analýzy a výstavby informačných systémov. Od doby jej vzniku (70. roky) prešla oblasť modelovania informačných systémov veľkými zmenami, a to najmä na začiatku 90. rokov, kedy vznikli desiatky nových, objektovo orientovaných metód. Niektoré z nich boli prínosom, iné sa nikdy nepresadili mimo úzky okruh svojich tvorcov. Väčšina užívateľov mala problém nájsť modelovací jazyk, ktorý by čo najlepšie riešil ich požiadavky. To bol jeden z dôvodov prečo postupom času vznikali snahy nájsť v tejto oblasti akýsi zjednocujúci prostriedok.

1.1 Objektovo orientované modelovanie

Vychádzame z predpokladu, že je možné na akýkoľvek reálny systém pozerať ako na množinu nejakých objektov a väzieb medzi nimi. Cieľom objektového modelovania je potom nájsť, resp. vytypovať príslušné objekty, zadefinovať ich vlastnosti a vzájomné väzby a vymedziť hranice celého systému. Následne vo fáze návrhu systému a vývoja podrobne rozpracúvame jednotlivé objekty a bližšie definujeme vzťahy medzi nimi.

2. Unified Modeling Language (UML)

UML nie je ucelená metodika definujúca aj nevyhnutný proces tvorby informačných systémov (IS), ale iba množina nástrojov, ktoré sa v tomto procese používajú. Až nasadením zodpovedajúcej objektovej metodiky, ktorá definuje kedy, kým a ako jednotlivé nástroje používať, môžeme vyťažiť z UML maximum. UML odlišuje nástroje pre modelovanie statickej a dynamickej stránky systému. Metodika pozná nasledovné typy diagramov:

2.1 Diagram prípadov použitia (Use Case Diagram)

Prvá úloha, t.j. vymedzenie hraníc systému, je v UML podporené dynamickým pohľadom modelovaným prostredníctvom Use Case diagramov. Tento diagram, všeobecne model, zobrazuje základný vzťah systému k jeho okoliu, tzv. aktérom. Aktérmi sú najčastejšie samotní užívatelia systému, všeobecne však nimi môžu byť ľubovoľné externé činitele stojace mimo modelovaný systém.

Môže teda ísť napr. o ďalšie HW a SW prostriedky, iné IS, s ktorými má nový IS spolupracovať, inštitúcie, ktoré so systémom komunikujú, atď. Každý prípad použitia potom predstavuje jeden z možných spôsobov použitia systému, jednu z možných ciest komunikácie aktér – systém. Prípady použitia stimulujú použitie reálneho systému externými aktérmi. V rámci tvorby týchto diagramov modelujeme vzťah systému a jeho okolia, nie vzájomnú interakciu, t.j. spôsob, akými sú jednotlivé prípady použitia zaistené internými funkciami systému.

Najväčšie úskalie pri používaní tohoto prostriedku spočíva v odhadnutí „správnej úrovne abstrakcie“. Občas sa môže stať, že sú prípady použitia príliš všeobecné a nedá sa s nimi bez patričného spodrobnenia efektívne pracovať. Na druhú stranu je nutné vyvarovať sa druhého extrému, ktorý je zneužívanie tohoto nástroja pre funkcionálny rozklad, ktorý nemá s „use case modelovaním“ nič spoločného! História hovorí, že systém postavený na funkciách, ktoré má zaisťovať, skôr než na objektoch, ktoré ho tvoria, je často nestabilný z hľadiska meniacich sa užívateľských požiadaviek. Aj keď toto riziko a problémy s ním spojené objektový prístup neeliminuje bezo zbytku, je predsa len možné nežiadúce efekty vyplývajúce zo zmeny požiadaviek čiastočne potlačiť. V ideálnom prípade sa zmena premietne v mieste, ktorého sa bezprostredne týka, v horšom prípade ovplyvní čiastková zmena mnoho súčastí systému. S rastúcim počtom následných úprav potom logicky rastie i riziko nekonzistencie modelov.

Prax ukazuje, že z hľadiska používania UML je zvládnutie tvorby týchto modelov jedným z najťažších úloh, ktorý naviac vďaka svojej povahe výraznou mierou ovplyvňuje podobu výsledného produktu. Napriek tomu, že je pomocou týchto prostriedkov možné rozdeliť systém na dielčie časti, nie je tento prostriedok určený na komplexné modelovanie architektúry.

V spojení s ďalšími prostriedkami pre dynamické modelovanie je tvorba diagramov Use Case základným prostriedkom pre nájdenie objektov participujúcich v modelovanom systéme. Rozdelenie celého systému na jednotlivé prípady použitia prináša okrem vymedzenia hraníc systému i možnosť spracovávať jednotlivé prípady použitia oddelene, a čiastočne tak realizovať iteratívny prírastkový životný cyklus. Cez tvorbu prípadov použitia sú samozrejme definované i základné užívateľské požiadavky.

2.2 Sekvenčný diagram (Sequence diagram)

Sekvenčné diagramy sa vytvárajú väčšinou priamo z Use Case diagramov. K jednému prípadu použitia môže existovať niekoľko sekvenčných diagramov, ktoré modelujú interakciu objektov v rámci komunikácie aktéra so systémom.

Sekvenčné diagramy sú zamerané výhradne na dynamickú stránku systému. Sú vhodné pre nájdenie jednotlivých objektov a zobrazenie komunikácie medzi nimi. S tvorbou sekvenčného diagramu sa postupne vynárajú jednotlivé objekty (resp. kandidáti na objekty), ktoré sú rovno

zapracovávané do diagramov tried. Je vhodné zdôrazniť, že objekty nájdené v začiatočných fázach modelovania systému sa zriedka kedy v pôvodnej podobe uplatnia tiež ako SW objekty implementované v cieľovom prostredí. Najprv ide skôr o koncepty, pojmy, ktoré sa postupom času, najmä vo fáze designu, vyhodnocujú a dochádza spravidla k čiastočnej redukcii a konsolidácii objektových modelov.

Výsledný systém môže maž nakoniec podobu logickej trojvrstvovej architektúry, kde sú od seba oddelené objekty prezentačnej vrstvy, objekty samotnej problémovej oblasti nájdené v analýza a rozpracované v designe a nakoniec napríklad objekty zabezpečujúce funkcie perzistentnej vrstvy. Ak ignorujeme analýzu problémovej oblasti, môžeme vo výslednom systéme nakoniec onú prostrednú vrstvu predstavujúcu tzv. obchodnú logiku úplne postrádať. Ak sme prozreteľnejší, snažíme sa obchodnú logiku systému navrhovať priamo z analytických modelov, čo nám okrem iného pomôže zabezpečiť lepšiu náväznosť s užívateľskými požiadavkami. Sekvenčný diagram obsahuje dve dimenzie. V horizontálnej rovine sa zobrazujú jednotlivé objekty, zatiaľ čo vertikálna rovina predstavuje tok času. Správy posielané medzi jednotlivými identifikovanými objektmi môžu byť rôzneho druhu. Ak záleží na ich bližšom odlíšení, dajú sa klasifikovať správy asynchrónne, vnorené, správy predstavujúce návratové hodnoty a pod. V prvotných fázach pochopiteľne ešte nemajú jednotlivé správy podobu metód objektov vrátane počtu a typov parametrov; v skutočnosti by tieto informácie zatiaľ nemali v modeli čo robiť. N takéto „low-level“ rozhodnutia prichádza čas až v etapách designu, kedy to má zmysel. V analýze nás zaujíma hlavne všeobecná komunikácie, ktorej jednotlivé detaily rozpracovávame až po ustálení objektového modelu. V prípade nutnosti sú v sekvenčných diagramoch používané cykly a vetvenia, ktoré sú užitočné hlavne v spojení s textovým popisom správ, ktoré sa najčastejšie nachádzajú v ľavej časti diagramu.

S rozpracovávaním sekvenčného diagramu môže pomerne rýchlo vzrastať jeho celková komplexnosť, čo je celkom spoľahlivým príznakom príliš všeobecného prípadu použitia, ku ktorému je sekvenčný diagram vytváraný. Namiesto tohoto prostriedku je niekedy možné zvoliť prostriedok jemu významovo veľmi blízky – diagram spolupráce.

2.3 Diagram spolupráce (Collaboration Diagram)

Ak chceme v jednom diagrame znázorniť tak štruktúru objektov, tak aj ich dynamické chovanie sa, použijeme s výhodou diagramy spolupráce, ktoré sú vedľa sekvenčných diagramov ďalším prostriedkom, ktorého ťažisko tkvie v modelovaní dynamiky. Na rozdiel od sekvenčných diagramov je však znateľne ťažšie vysledovať náväznosť jednotlivých posielaných správ zabezpečujúcich samotnú funkčnosť systému. Zatiaľ čo v sekvenčných diagramoch je táto náväznosť zrejmá z vertikálneho usporiadania celého diagramu, v diagramoch spolupráce je následnosť zobrazená poradovým číslom.

Tvorba diagramu spolupráce by mala byť v súlade s diagramom tried. Pre zachovanie vzájomnej konzistencie modelov je nevyhnutné dodržovať základné pravidlá, ktoré medzi týmito modelmi platia. Komunikácia medzi objektmi musí byť napríklad podmienená existenciou odpovedajúcej väzby medzi ich triedami, o existencii patričných tried nehovoriac. Netreba ani zdôrazňovať, akou nevďačnou prácou je „ručné“ kontrolovanie obdobných pravidiel. Je jasné, že tu môžu opäť pomôcť CASE nástroje. Vcelku seriózne sa dá vyhlásiť, že práve tu existuje hranica medzi skutočnými CASE systémami a obyčajnými kresliacimi programami.

Ak sa sekvenčné diagramy stávajú občas neprehľadnými, potom to u diagramov spolupráce platí dvojnásobne. Na druhú stranu vzhľadom k tomu, že sa poradie zasielania správ určuje poradovým číslom, dá sa lepšie modelovať komplexné chovanie vrátane vetvenia a cyklov.

2.4 Diagram tried (Class Diagram)

Diagram tried patrí bezo sporu k najčastejšie používaným nástrojom UML a tvorí vôbec akýsi základ všetkých prostriedkov objektovej analýzy a designu. Tieto diagramy sú väčšinou vytvárané už vo fáze analýzy často ako pojmové modely, ktorých úlohou je formálnejšie definovať jednotlivé termíny používané v študovanej problémovej oblasti. Takéto modely sú maximálne prínosné a užitočné v okamihu, kedy je nutné spätne vstrebať terminológiu oboru. S postupným približovaním k fáze implementácie sú diagramy tried pomerne zásadne prehodnocované a v ideálnom prípade spracovávané až do podoby grafického modelu ekvivalentného zdrojovému kódu.

Diagramy tried zobrazujú statickú stránku systému, predovšetkým vzťahy medzi triedami. UML explicitne rozlišuje niekoľko druhov tried, rovnako ako rozličné množstvo vzťahov, ktoré jednotlivé triedy navzájom spájajú (asociácie, agregácie, kompozície, ...).

Tvorba diagramov tried patrí medzi kľúčové problémy a celkom vážne možno prehlásiť, že zvládnuť objektový prístup často znamená správne využívať práve tento typ diagramov používaný priebežne naprieč celým životným cyklom tvorby IS. Zvlášť pri tvorbe týchto diagramov sa doporučuje dodržiavať nepísané pravidlo 7 + 2, ktoré hovorí, že v jednom diagrame je vhodné zobraziť 7, maximálne až 9 tried. Pri rešpektovaní tohoto doporučenia sa väčšinou výraznou mierou sprehľadnia výsledné modely, no na druhej strane vyvstáva problém vhodného rozdelenia systému na dielčie oblasti (tzv. packages).

Asi najčastejšími chybami pri tvorbe diagramov tried (a celkovo pri objektovom modelovaní) je zámena pojmov objekt a trieda, čím môže dôjsť k zavedeniu závažných nekonzistencií do vytváraného modelu.

Značne ovplyvnená svojimi predchodcami je notácia diagramov tried v UML vzdialene podobná klasickým ER diagramom obohateným o niektoré objektové črty. Atribúty tried sú zobrazené v strednej časti prvku triedy, metódy potom v časti spodnej. Podľa špecifikácie UML môžu byť atribúty iba základných typov, všetky ostatné by mali byť zobrazené ako vzťahy k patričným triedam. Je tu jasne vidieť polovičatosť, s akou sa UML stavia k objektovému prístupu. V skutočnosti samozrejme neexistuje objektívny dôvod pre odlišovanie jednotlivých atribútov podľa typu, v čisto objektových jazykoch sú aj základné typy reprezentované prostredníctvom patričných tried. To však zďaleka nie je prípad klasického C++, ktorým je UML najviac ovplyvnený. Rozlišovanie spôsobu notácie atribútov podľa typu sa môže zo začiatku zdať veľmi metúce a nelogické. V podobnom duchu je možné určiť pre jednotlivé zložky (atribúty, metódy) tried ich viditeľnosť vzhľadom k ostatným triedam atď.

2.5 Stavový diagram (State Transition Diagram)

Stavový diagram patrí medzi klasické a osvedčené nástroje objektového modelovania. Diagram stavov a prechodov, ako je niekedy tento prostriedok nazývaný, slúži pre modelovanie životného cyklu časti systému svojím rozsahom zodpovedajúcim jednému objektu. Prechod medzi jednotlivými stavmi býva vyvolaný podnetom z vonkajšieho okolia,

najčastejšie vo forme správy zaslanej príslušnému objektu alebo inou externou udalosťou. Validný stav objektu je zjednodušene povedané definovaný prípustnými hodnotami jeho atribútov, prechod medzi stavmi je potom vlastne vyjadrením zmeny hodnôt týchto atribútov.

Ako je asi každému jasné, životný cyklus objektu nejako začína a spravidla aj nejako končí. Nemusí byť už ale celkom samozrejmé, že počiatočný stav by mal byť v diagramu vždy iba jeden, zatiaľ čo „stop stavov", t.j. stavov ukončujúcich životný cyklus sledovaného objektu, môže byť viac, pochopiteľne aspoň jeden.

2.6 Diagramy aktivít (Aktivity Diagram)

Ako určitú obdobu stavových diagramov môžeme chápať i diagramy aktivít, kde jednotlivými stavmi rozumieme aktivity a prechod medzi aktivitami je vyvolaný dokončením aktivity stávajúcej. Diagram aktivít sa spravidla vzťahuje k jednému prípadu použitia, prípadne k jednej metóde objektu. Pomocou diagramu aktivít modelujeme tento krát dynamický tok riadený nie vonkajšími udalosťami ale internými podnetmi. Pri troške dobrej vôle možno tento nástroj používať aj pre modelovanie procesov a pracovných tokov (work – flow), kedy sa nezriedka využíva možnosť modelovať paralelizmus a vetvenie v rámci konkrétneho spracovávaného procesu.

Pomerne elegantným spôsobom je možné priraďovať k jednotlivým aktivitám osoby (resp. aktérov), ktoré sú za uskutočnenie patričnej aktivity zodpovedné. Rovnako dobre ako pre procesné modelovanie je možné používať diagramy aktivít aj pre základnú schematickú tvorbu grafického užívateľského prostredia.

2.7 Diagram komponentov (Component Diagram)

Tento nástroj je využiteľný hlavne vo fáze implementácie a nasadenia SW/IS. Jeho pomocou sa vizualizujú vzťahy medzi jednotlivými SW komponentmi, ktoré môžu byť vo forme zdrojových súborov, hotových spustiteľných častí alebo iba hlavičkových súborov. Medzi jednotlivými komponentmi môže existovať iba jeden druh väzby, a tou je závislosť.

Komponentom v zmysle UML je určitá množina tried, ktoré spolu realizujú požadovanú funkčnosť. Diagramy tried je tak možné „rozsekať" na abstraktnejšie celky, s ktorými môžeme následne pracovať ako so základnými stavebnými kameňmi. CASE nástroje by pochopiteľne mali umožňovať integráciu takto vzniknutých komponentov so štandardami pre distribuované objektové systémy.

2.8 Diagram nasadenia (Deployment Diagram)

Diagram nasadenia je druhým typom diagramov určených pre implementačnú fázu. Jeho úlohou je hlavne zobraziť vzťahy medzi časťami systému tak, ako vypadajú v dobe samotného vykonávania. Diagramy nasadenia zobrazujú rozloženie jednotlivých SW komponentov na HW zdrojoch a ich spoluprácu, rozmiestnenie HW a SW prostriedkov v lokalitách, topológiu používaných sietí, druhy a využitie komunikačných prostriedkov atď.

V príspevoku sme sa snažili aspoň z časti charakterizovať niektoré nástroje a diagramy, ktoré obsahujú systémy pre podporu riadenia výstavby IS - objektovo orientované. Uvedomujeme si, že je to rozsiahla problematika a my sa s ňou máme možnosť zoznámiť i vďaka MetaEditu, ktorý sa pokúšame sprístupniť pre študentov v rámci výučby.

Literatúra

1. Kremeňová, I., Súlovec, R.: Metacase Tools for multidimensional Development of Information System, EUNIS 2001 – Conference, The 7th International Conference of European University Information Systems, Humboldt-University, March 2001
2. Majerčak, J.: Supply chain v oblasti distribučnej logistiky s využitím nástrojov štrukturálnej analýzy, LOGVD 2000, EDIS ŽU Žilina
3. www.rational.com/uml
4. www.uml-zone.com