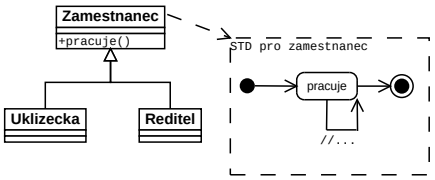
Stavové diagramy a dědičnost

Martin Šlouf, xslom03@vse.cz

esej na předmět it_578

Rád bych se zamyslel nad tím, jaké problémy (výhody?) může způsobit dědičnost ve stavových diagramech třídy a komentoval to na konkrétním příkladě. Rád bych upozornil na to, že metodika OOMT i UML připouští hierarchizaci stavových diagramů. OOMT[1] navíc jak ve smyslu dědičnosti, tak i ve smyslu rozkladu složitých na úloh (stavů) na jednodušší (popsané vlastním diagramem)[2].

*Příklad: Firma má mnoho typů zaměstnanců (uklízečky, ředitele, ...) a všichni zaměstnanci prochází několika společnými stavy (*připraven k práci, nemocen, na dovolené, ...*), navíc každý prochází svými specifickými stavy, danými činností, jež ve firmě vykonává (ředitel* řídí firmu, *uklízečka* uklízí*). Je zřejmé, že podle povahy činnosti může být její zachycení pomocí stavů a přechodů mezi nimi dosti složité. Můžeme si to představit třeba takto:*



Virtuální metoda pracuje() *znamená, že zaměstnanec je právě ve stavu (nadstavu)* pracuje *a dělá cokoli, co právě jeho specifická činnost vyžaduje.*

Metodika OOMT podobné situace řeší podle mého soudu dobře.

1. Umožňuje definici *nadstavů*.
2. Umožnuje rozklad *složitějších* stavů do samostatných stavových diagramů.
3. Vždy máme možnost pro každou odvozenou třídu udělat vlastní stavový diagram, tím se ovšem ochuzujeme o krásu OO návrhu.

Pro abstraktní třídy jsme tedy schopni definovat obecné abstraktní nadstavy (*pracuje*), které mohou být výsledkem nějaké abstraktní události (*začni pracovat*) a mohou vyvolat nějakou abstraktní metodu (*pracuj()*). Není příliš obtížné představit si, že tato abstraktní metoda se řídí pro každou odvozenou třídu zcela vlastními pravidly — volá jiné metody jako akce na různé události (typické pro konkrétní odvozený typ) a tyto metody zajišťují přechod do jiných stavů (opět typické pro konkrétní odvozený typ). V okamžiku navrácení z metody *pracuj()* se konkrétní odvozený objekt opět může začít řídit obecnými pravidly definovanými třídou zaměstnanec.

V této abstrakci vydím ohromný potenciál OO návrhu — definování abstraktních tříd, spolu s jejich abstraktními stavy a událostmi, zajišťující přechod mezi těmito stavy[3]. Síla takovýchto návrhů se podle mého již ukázala – takto typizované třídy (komponenty) se běžně užívají[4]. Před uživatelem tříd zůstávají skryty nedůležité podrobnosti.

Malá implementační poznámka na závěr — za důležité považuji i to, že dané postupy návrhu jsou přímo podporovány OO jazyky a mohou být použity během kódování.

Shrnutí. Pakliže se podaří dobře definovat abstraktní chování třídy (včetně stavů) může to přinést nečekané výhody. Nebezpečí však zůstává skryto v závislostech mezi stavy odvozených tříd (jistě takové závislosti budou); s těmi nemůže obecný model počítat při svém návrhu a proto se s nimi nemusí vždy dobře vypořádat. Ať přínos, či nikoli, metodiky OO návrhu to dokáží řešit, stejně tak i OO jazyky.

---

[1] Viz Drbal, Pavel: Objektově orientované metodiky a technologie 1. díl, Vysoká škola ekonomická, Praha 1997, str. 76

[2] Jestli i UML připouští rozklad složitých stavů na jednodušší, to nevím.

[3] To považuji za klíčové — mezi obecnými stavy musí být zajištěn obecný přechod mezi nimi. Pokud to bude porušeno, zavádí se tím nepříliš přehledné závislosti. Dokážu si ovšem představit situaci, kdy to bude nevyhnutelné (pokud například činnost ve firmě bude vyžadovat interakci mezi ředitelem a uklízečkou)

[4] Dnes již téměř standardizované db rozhraní — například JDBC.