Unifikovaný modelovací jazyk UML [1]

Karel Richta

katedra počítačů, FEL ČVUT v Praze
Karlovo nám. 13, 121 35 Praha 2
e-mail:richta@fel.cvut.cz

Klíčová slova: UML, OCL.

Abstrakt. *Komunikačním prostředkem informační komunity se postupem času stala angličtina. Chcete-li vystavit nějakou informaci tak, aby byla srozumitelná i mimo hranice ČR, použijete angličtinu. Podobně probíhá vývoj i v komunitě softwarových inženýrů. Během vývoje metodik softwarového inženýrství bylo navrženo mnoho různých víceméně formálních jazyků a prezentačních technik, které slouží pro popis softwarových produktů od konceptuálního modelu až po jeho implementaci. Různorodost zápisů přináší radost těm, kteří se neradi váží na určitý jazyk či grafickou techniku, méně radosti však činí těm, kteří musí takové zápisy po někom studovat a zpracovávat. Nové generace metodik softwarového inženýrství se snaží sjednotit užitečné vlastnosti různých metodik a integrovat je do nějaké společné sady. Jedním z nejdále propracovaných přístupů je tzv. unifikovaný modelovací jazyk UML (Unified Modeling Language) – kandidát na jakési "esperanto" moderního softwarového inženýrství.*

1. Úvod

Metodiky softwarového inženýrství představují soubor prezentačních technik (výrazových prostředků) a metodických postupů, které podporují systematický vývoj a údržbu programového díla, včetně plánování a řízení prací na programovém projektu. Metodiky se tedy mohou lišit prezentačními technikami (notací), metodickými postupy (proces) a dostupnými nástroji. Každá metodika přinesla něco nového, obvykle však též novou notaci. Rozmanité novinky jen výjimečně přinesly i novou kvalitu.

To bylo důvodem snahy o unifikaci vyjádření. Vznikají metodiky které se snaží sjednotit výhodné vlastnosti různých notací a metodických postupů. Patří sem např. metodika Fusion (Coleman, 1994), či Unified Method (Rumbaugh/Booch, 1995), která vznikla integrací metodiky OMT (Object Modeling Technique - Rumbaugh) a OOD (Object-Oriented Design - Booch).

Vývoj unifikované metodiky si dala za jeden ze svých hlavních cílů firma Rational [2], kterou založili Rumbaugh a Booch. Později se k nim připojil i Jacobson se svou metodikou OOSE (Object-Oriented Software Engineering) a společně vypracovali prvý návrh UML (1996). v roce 1997 převzalo tento návrh sdružení OMG (Object Management Group [1]) a pod označením UML 1.1 jej začalo doporučovat jako standard (1997). Současná verze z března 2003 má označení UML 1.5, pracuje se na verzi standardu UML 2.0.

UML je zamýšlen jako univerzální standard pro záznam, konstrukci, vizualizaci a dokumentaci artefaktů systémů s převážně softwarovou charakteristikou. Definice UML obsahuje 4 základní části:

- Definici notace UML (syntaxe).
- Metamodel UML (sémantika).

[1] *Vznik tohoto textu byl částečně motivován prací na výzkumném záměru MŠMT číslo MSM 212300014 "Výzkum v oblasti informačních technologií a komunikací" a spoluprací na grantovém projektu GAČR 201/03/0912 "Vyhledávání a indexování XML dokumentů".*

- Jazyk OCL (Object Constraint Language) pro popis dalších vlastností modelu.
- Specifikace převodu do výměnných formátů (CORBA IDL, XML DTD).

1.1 Notace UML

Při analýze požadavků se často jako nástroj pro komunikaci mezi zadavatelem a řešitelem využívá modelování. Vznikají různé modely navrhovaného systému, které mohou být na této úrovni testovány, ověřovány a upravovány. Teprve po dosažení shody mezi analytikem a zadavatelem může být model dekomponován a převeden do cílového implementačního jazyka.

Používané modely mohou být různých typů. Obecně lze model softwarového systému definovat jako souhrn informací, které jsou určitým způsobem strukturovány. Model by měl obsahovat veškeré shromážděné informace o systému.

Diagram je graficky znázorněný pohled na model. Diagram popisuje jistou část modelu pomocí grafických symbolů. Na rozdíl od modelu, jeden diagram málokdy popisuje celý systém – většinou je k popisu systému použito více pohledů, z nichž každý se zaměřuje na jeden aspekt modelu.

Analytické modely se zabývají zejména otázkou "**CO**" by měl systém dělat. Návrhové modely popisují dekompozici systému na programátorsky zvládnutelné části - zabývají se otázkou "**JAK**" by to mělo být uděláno. Implementační modely dokumentují implementaci.

UML se snaží shrnout nejlepší známé pohledy a definuje osm základních druhů diagramů, které budou popsány dále. UML sice nebrání vytváření dalších možných pohledů a diagramů, takový postup je však trochu proti duchu unifikace. Smyslem unifikace naopak je, aby stejné pohledy na model systému bylo možno vyjádřit tak, aby znázornění bylo obecně srozumitelné. Notace UML definuje, jak se zapisují základní pohledy na modelovaný systém.

1.2 Sémantika UML

Zápisy v UML (pohledy, diagramy) sestavené podle pravidel syntaxe musí mít pevně definovaný význam. Tím se zabývá popis sémantiky UML. Je rozčleněn do 4 vrstev, které na různé úrovni abstrakce popisují vlastnosti diagramů UML, včetně možných rozšíření. Na nejnižší úrovni jsou specifikovány vlastnosti primitivních údajů (dat) - typy dat, obory hodnot atributů. o úroveň výše je vyjádřena sémantika uživatelských objektů (tzv. model, nebo též meta-data) - např. co to je záznam o objektu typu "zaměstnanec". Ještě výše stojí popis prvků modelu (metamodel) - např. co to je třída, atribut, vztah, atd. Nejvýše je pak definice vlastností metamodelu (meta-metamodel), např. jak lze korektně vytvářet nové prvky modelu.

1.3 Jazyk OCL

Ne všechny vlastnosti modelu lze vyjádřit pomocí diagramů. Uvažme např. datový model systému, který se zabývá evidencí zaměstnanců. O každém zaměstnanci si potřebujeme pamatovat různé atributy, mimo jiné např. plat zaměstnance. Rovněž si potřebujeme pamatovat hierarchickou strukturu mezi zaměstnanci. Pomocí diagramů vyjádříme snadno požadavky na strukturu dat, těžko zde však zachytíme požadavky jako "nadřízený musí mít vyšší plat než jeho podřízení". UML nabízí (ale nevnucuje) možnost použít pro vyjádření takových omezení specifikační jazyk OCL (Object Constraint Language). OCL poslouží právě pro vyjádření komplikovanějších integritních omezení, popis vlastností operací a může být koneckonců použit i pro vyjádření sémantiky UML.

1.4 Specifikace výměnných formátů

Součástí definice UML je i specifikace výměnných formátů, sloužících např. pro přenos zápisů v UML mezi různými nástroji. Jako příklad lze uvést formát CORBA IDL (Interface Definition Language), či XMI (XML Metadata Interchange).

2. Lexikální elementy UML

Základem definice syntaxe UML je specifikace lexikálních elementů – nejmenších lexikálních jednotek, ze kterých se může dokumentace v UML skládat. Existují 4 základní druhy lexikálních elementů UML:

- **řetězec** (posloupnost znaků, znaková sada není omezena, v některých případech je doporučeno používat ASCII kódu – zajistí to snadný přenos do programového prostředí),
- **ikona** (grafický symbol zastupující element, neobsahující žádné složky),
- **2D-symbol** (grafický symbol zastupující element, který má obsah, případně rozdělený na složky),
- **spojka** (path - posloupnost úseček, které navazují, mohou mít zvýrazněny koncové body a mohou incidovat s hranicí 2D-symbolů - slouží k vyznačení propojení).

Elementy modelu dokumentovaného v UML potřebujeme označovat - potřebujeme jména objektů, tříd, atributů, metod, a pod. Jako jména se v UML používají **identifikátory** – speciální lexikální elementy typu řetězec, které obsahují pouze ASCII písmena, cifry a znaky '_' nebo '-'. Pro zápis identifikátorů, které zastupují víceslovní termín je vhodné zvolit nějakou konvenci, doporučené je používání některého z následujících zápisů:

```
josefNovak, josef_novak, josef-novak
```

Jméno se obvykle vztahuje k nějakému kontextu (scope), pak mluvíme o tzv. **kvalifikovaném jménu**. Např. úplné kvalifikované jméno pro objekt "objednávka", o kterém mluvíme v rámci "účetnictví" bude

```
Ucetnictvi::Objednavka
```

Řetězec přidružený ke grafickému symbolu nazýváme **návěští**. Pokud má identifikátor význam **klíčového slova**, uvádíme jej ve "francouzských" závorkách (**guillemets**):

```
<<keyword>>
```

Výrazy v UML jsou řetězce sestavené z lexikálních elementů podle jistých pravidel. Syntaxe výrazů není striktní, musí být ale zvolen nějaký dobře definovaný jazyk. Tj. výraz představuje formuli, kterou musí být možno v daném jazyce vyhodnotit (má svou **sémantiku**). Definice UML zahrnuje definici jazyka OCL (Object Constraint Language), který je použit pro definici sémantiky UML (metamodelu UML). Je doporučeno používat OCL všude tam, kde je to možné..

3. Diagramy UML

Dokumentace v UML se nemusí skládat pouze z (více, či méně formálních) textů. Ze základních lexikálních elementů lze vytvářet dvourozměrné diagramy – rozmístíme na ploše určitou sadu elementů a případně je propojíme pomocí spojek. Teoreticky lze použít libovolné elementy a spojky, z hlediska unifikace je však výhodné, pokud zvolíme určitou sadu elementů, která je vhodná pro vyjádření některého smysluplného pohledu na modelovaný systém. Touto volbou jsme de facto definovali **typ diagramu**. UML proto zahrnuje definici 8-mi typů diagramů, tj. 8-mi různých pohledů na model systému:

- **diagramy tříd** a **objektů** (class diagrams, object diagrams) popisují statickou strukturu systému, znázorňují datový model systému od konceptuální úrovně až po implementaci,
- **modely jednání** (diagramy případů užití - use case diagrams) dokumentují možné případy použití systému - události, na které musí systém reagovat,
- **scénáře činností** (diagramy posloupností - sequence diagrams) popisují scénář průběhu určité činnosti v systému,
- **diagramy spolupráce** (collaboration diagrams) zachycují komunikaci spolupracujících objektů,

- **stavové diagramy** (statechart diagrams) popisují dynamické chování objektu nebo systému, možné stavy a přechody mezi nimi,
- **diagramy aktivit** (activity diagrams) popisují průběh aktivit procesu či činnosti,
- **diagramy komponent** (component diagrams) popisují rozdělení výsledného systému na funkční celky (komponenty) a definují náplň jednotlivých komponent,
- **diagramy nasazení** (deployment diagrams) popisují umístění funkčních celků (komponent) na výpočetní uzly informačního systému.

Statickou strukturu systému vyjadřují diagramy tříd, diagramy spolupráce, diagramy komponent a diagram nasazení. **Funkční stránku** popisují model jednání, diagramy aktivit, scénáře událostí a diagramy spolupráce. **Dynamickou stránku** dokumentují stavové diagramy, scénáře událostí, diagramy spolupráce a diagramy aktivit.

Použití diagramů v jednotlivých fázích vývoje je dáno konkrétní metodikou, ale orientačně můžeme říci, že v rámci analýzy se využívají diagramy tříd, model jednání, diagramy aktivit, scénáře činností a stavové diagramy. Pro fázi návrhu jsou typické diagramy tříd, diagramy spolupráce, diagramy aktivit, diagramy komponent a diagramy nasazení. Ve fázi implementace se používají diagramy tříd, diagramy komponent a diagramy nasazení.

UML se snaží být univerzální notací pro všestranné použití od obchodního modelování po detailní návrh systémů pro práci v reálném čase. Notace, která by vyčerpávajícím způsobem pokryla tak široké spektrum potřeb by ale byla velmi komplikovaná a složitá. UML proto definuje pouze základní část (UML core) a umožňuje rozšíření notace dle konkrétních potřeb. Rozšíření UML lze dosáhnout použitím jiného jazyka (než OCL) pro výrazy. Standardní sémantiku UML je možno doplnit standardními omezeními, která umožňují změnit interpretaci elementů:

- **příznaky** (tags) – umožňují přidat k prvku diagramu další informace (atributy),
- **omezení** (constraints) – umožňují specifikovat omezení pro elementy (hodnoty atributů),
- **stereotypy** (stereotypes) – umožňují klasifikovat elementy.

Stereotypy umožňují klasifikovat elementy diagramů a tím vyjádřit další sémantiku. Zapisují se jako klíčová slova, ale(mohou se případně zobrazit jako ikony, což se příliš nedoporučuje - porušuje to princip jednotnosti, neboť tyto ikony nejsou standardní. Jako příklad mohou posloužit stereotypy:

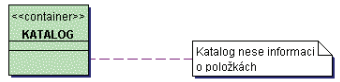
`<<database>>` (označení komponenty zabývající se správou dat)
`<<entity>>` (označení třídy reprezentující data)

Existují standardní stereotypy, např. `<<include>>`, `<<extend>>`, jejichž význam je definován sémantikou UML. Příznaky umožňují přidat k prvku diagramu další informace ve formě dvojice atribut=hodnota:

```
{ autor="Josef Novák", verze=1.2 }
```

Název atributu může být libovolný. Příznaky se zapisují do složených závorek, připojují se k názvu prvku diagramu a typicky je lze použít například ke specifikaci verze, autora, či implementačních poznámek. Omezení umožňují specifikovat požadavky na sémantiku prvků - lze pomocí nich formulovat různá omezení v modelu. Zapisují se jako výraz uzavřený do složených závorek. Příklad:

```
{ počet_zaznamu < 10000 }
```

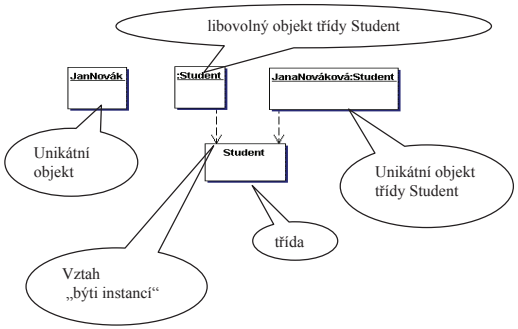


Omezení se může týkat více prvků - pak je spojeno s těmito prvky čárkovanou čárou. Často se kombinuje s poznámkami. Poznámka obsahuje libovolný text. Zobrazuje se jako obdélník s „přehnutým rohem". Může být přidružena k elementu, může obsahovat stereotyp (např. `<<constraint>>` - pak se jedná o specifikaci omezení).

4. Diagramy objektů (Object Diagrams)

Objekt je pojem, abstrakce, nebo věc s dobře definovanými hranicemi a významem. Každý objekt má tři charakteristiky: identitu, stav a chování.

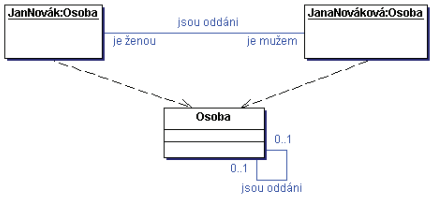
- **Stav** objektu je jedna z možných situací, ve kterých se objekt může nacházet. Stav objektu se může měnit a je definován sadou vlastností - atributy a vztahy.
- **Chování** určuje, jak objekt reaguje na žádosti jiných objektů a vyjadřuje vše, co může objekt dělat. Chování je implementováno sadou operací (metod).
- **Identita** znamená, že každý objekt je jedinečný.



5. Diagramy tříd (Class Diagrams)

Třídy zachycují společné vlastnosti sady objektů - **atributy** a **operace** (metody). Stereotypem lze zavést nové druhy (skupiny) tříd. Nejčastějšími skupinami (stereotypy) tříd jsou:

- **entitní třídy** (stereotyp je <<entity>>),
- **třídy rozhraní** (stereotyp je <<boundary>>) a
- **třídy řídicí** (stereotyp je <<control>>).



Vztahy mezi třídami (asociace) vyznačují možné vazby mezi objekty. Konce vztahů mohou být ohodnoceny rolí - role označují jakou roli objekt ve vztahu hraje. Pro zachycení kardinality a volitelnosti vztahů se používá notace N..M, kde N a M může být číslo nebo *, samotná * znamená totéž jako 0..*.

Pokud je zapotřebí si o vztahu něco pamatovat, požívají se tzv. přidružené třídy (atributy vztahů).

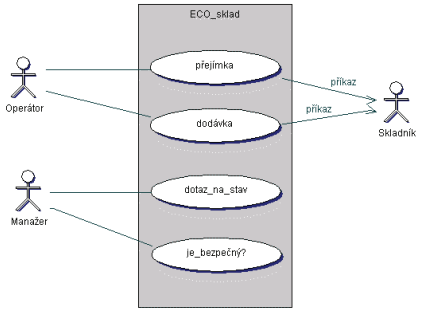
Speciální druhy vztahů představují agregace a generalizace:

- **agregace** (aggregation) – je druh vztahu, kdy jedna třída je součástí jiné třídy - vztah typu celek/část,
- **kompozice** (composition) – je silnější druh agregace - u kompozice je část přímo závislá na svém celku, zaniká se smazáním celku a nemůže být součástí více než jednoho celku,
- **generalizace** (generalization) – druh vztahu, kdy jedna třída je zobecněním vlastností jiné třídy (jiných tříd) - vztah typu nadtyp/podtyp, generalizace/specializace.

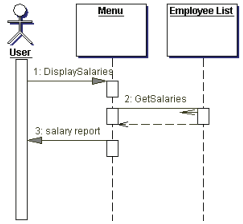
6. Model jednání (Use Case Model)

Model jednání slouží pro základní vymezení hranice mezi systémem a jeho okolím. Komponentami tohoto modelu jsou:

- **aktér** (actor) - uživatelská role nebo spolupracující systém
- **hranice systému** (systém boundary) - vymezení hranice systému
- **případ použití** (use case) - dokumentace události, na kterou musí systém reagovat
- **komunikace** - vazba mezi aktérem a případem použití (aktér komunikuje se systémem na daném případu).



Při vytváření modelu jednání je třeba brát v úvahu, že existují tzv. sekundární aktéři, tj. uživatelské role nebo spolupracující systémy, pro něž není systém přímo určen, ale které jsou pro jeho činnost nutné. Případy, kdy chceme vyznačit směr komunikace, značíme orientovanými šipkami. Mezi případy použití mohou existovat vztahy. Pokud chceme explicitně vyjádřit fakt, že takový vztah existuje, používáme stereotypy, standardně <<include>> - pokud jeden případ zahrnuje případ jiný (např. autentizaci), nebo <<extend>> - pokud nějaký případ rozšiřuje chování jiného (je zde možnost volby).

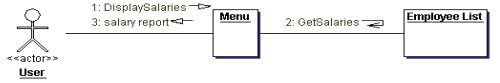


7. Scénáře činností (Sequence Diagrams)

Dokumentují spolupráci participantů na **scénáři** činnosti. Kladou přitom důraz na **časový** aspekt komunikace. Dokumentují **objekty** a **zprávy**, které si objekty posílají při řešení scénáře. Jsou vhodné pro popis scénáře při komunikaci s uživateli.

8. Diagramy spolupráce (Collaboration Diagrams)

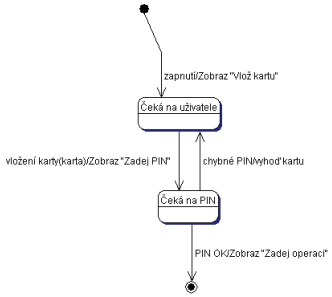
Podobně jako scénáře činností dokumentují diagramy kolaborace **spolupráci** objektů při řešení úlohy. Kladou větší důraz na **komunikační** aspekt (čas je vyjádřen číslováním).



Dokumentují **objekty** a **zprávy**, které si posílají při řešení problému. Jsou vhodné pro popis spolupráce objektů při návrhu komunikace.

9. Stavové diagramy (State Charts Diagrams)

Stavové diagramy slouží k popisu dynamiky systému. Stavový diagram definuje možné **stavy**, možné **přechody** mezi stavy, **události**, které přechody iniciují, **podmínky** přechodů a **akce**, které s přechody souvisí. Stavový diagram lze použít pro popis dynamiky objektu (pokud má rozpoznatelné stavy), pro popis metody (pokud známe algoritmus), či pro popis protokolu (včetně protokolu o styku uživatele se systémem). Přechod může být ohodnocen:




```
událost(parametry)[podmínka]/akce^zpráva
```

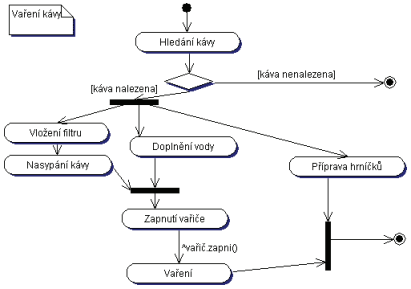
Každý stav může dále obsahovat popis akcí pro události vstup, výstup a opakované provádění:

```
entry/akce
exit/akce
do/akce
```

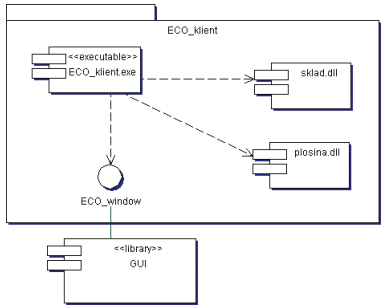
Stavové diagramy mohou být hierarchické. V nejnovějších verzích UML mohou obsahovat tzv. synchronizační značky (viz dále).

10. Diagramy aktivit (Activity Diagrams)

Varianta stavových diagramů, kde kromě stavů používáme **aktivity**. Nachází-li se system v nějakém stavu, je přechod do jiného stavu iniciován nějakou vnější událostí. U aktivity je, na rozdíl od stavu, přechod iniciován ukončením aktivity, přechody jsou vyvolány dokončením akce (jsou synchronní). Používají se pro dokumentaci tříd, metod, nebo případů použití (jako „workflow“). Nahrazují do určité míry v UML neexistující diagramy datových toků. Mohou obsahovat symbol „rozhodnutí“.



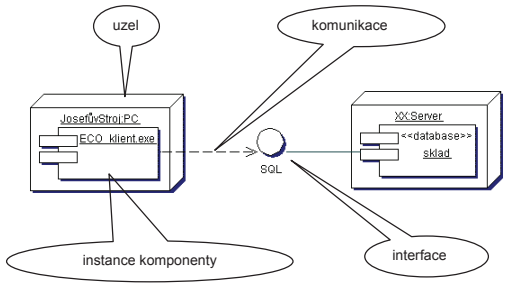
11. Diagramy komponent (Component Diagrams)



Vyjadřují (fyzickou) strukturu **komponent** systému. Popisují typy komponent - instance komponent jsou vyjádřeny v diagramu nasazení. Komponenty mohou být vnořeny do jiných komponent. Při vyjadřování **vztahu** mezi komponentami lze používat „interface“.

12. Diagramy nasazení (Deployment Diagrams)

Diagramy nasazení popisují fyzické rozmístění elementů systému na **uzly** výpočetního systému. Uzly a elementy jsou značeny obdobně jako objekty a třídy (může být uveden pouze typ, nebo konkrétní instance a typ - podtržena). Popisují nutné **vazby** mezi uzly (případně též použitý protokol - „interface“). Obsahují pouze **komponenty** potřebné pro běh aplikace - komponenty potřebné pro překlad a sestavení jsou uvedeny v diagramu komponent.



Shrnutí

Notace UML pomáhá sjednocení a čitelnosti dokumentace systémů. UML se ještě bude vyvíjet.

Poděkování

S potěšením zde mohu poděkovat firmě TogetherSoft Corporation (nyní Borland [4]), jejíž laskavá politika vůči vysokým školám mi umožnila pomocí nástroje Together Control Center připravit obrázky pro tuto publikaci.

Reference

1. OMG, Object Management Group; http://www.omg.org.
2. Rational software: http://www.rational.com/uml
3. Richta,K., Svačina,J.: UML: Teorie a praxe. In: Proc. of DATAKON 2003, pp. 1-26. ISBN 80-210-2958-7, Masarykova Universita, Brno, 2002.
4. Togethersoft: http://www.togethersoft.com
5. World Wide Web Consortium; http://www.w3.org.

Sumary

The paper briefly introduces the reader into the Unified Modeling Language UML. The paper only addresses the idea; it suggests to utilize common notation instead of specific languages. After considerable experience with the diagrams, the reader will optionally accept their usefulness.