POUŽITIE UML NA VÝVOJ ŽELEZNIČNÉHO ZABEZPEČOVACIEHO SYSTÉMU
THE USE OF UML TO DEVELOPMENT OF A RAILWAY INTERLOCKING SYSTEM

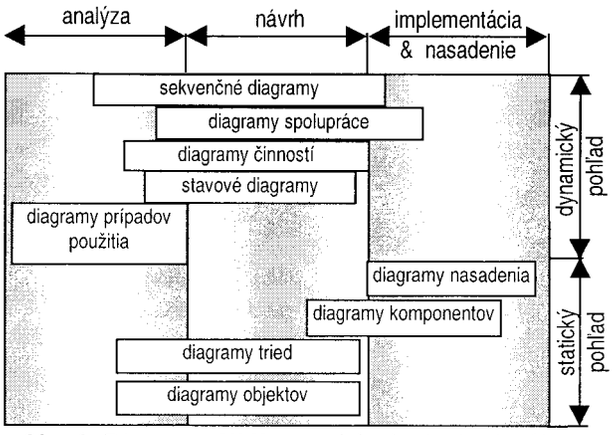
Karol Rástočný, Aleš Janota, Jiří Zahradník
Katedra Informačných a zabezpečovacích systémov, Elektrotechnická fakulta, Žilinská univerzita, Veľký diel, 010 26 Žilina

Abstrakt Článok je venovaný problematike využitia *unifikovaného modelovacieho jazyka* (*UML*) vo vývoji nových železničných zabezpečovacích systémov. Na zjednodušenom príklade riadiaceho obvodu prestavníka výhybky sú demonštrované zásady objektovo-orientovaného prístupu k špecifikácii požiadaviek na funkčnú bezpečnosť. Východiskom prezentovaného príkladu je neformálna špecifikácia vyplývajúca z príslušnej technickej normy, výsledkom je poloformálna špecifikácia v podobe *UML* modelu. V závere sú zhrnuté výhody použitia prezentovaného prístupu a naznačené trendy v súvislosti so zvyšovaním formálnosti *UML* modelov.

Summary The paper deals with problems of using the *Unified Modeling Language* (*UML*) in development of new railway interlocking and signalling systems. A simplified example of the control circuit of a point machine is used to demonstrate an object-oriented approach to specifying the functional safety requirements. An informal specification given by the relevant technical standard is used as a starting point and results in semi-formal specification based on *UML* model. Advantages of the presented approach are discussed and new trends of increasing formality of *UML* models are indicated within conclusions.

1. ÚVOD

Použitie elektronických prvkov a podsystémov využívajúcich softvér mení metódy vývoja železničných zabezpečovacích systémov i zaužívané prístupy a stratégie na dosiahnutie bezpečnosti. Vývoj softvéru prešiel evolúciou od strojového kódu cez symbolický assembler, procedurálne jazyky až po objektovo-orientované jazyky. S nárastom úrovne abstrakcie programovacích jazykov sa však súčasne zvyšovala aj úroveň zložitosti implementovaných funkcií. Nárast zložitosti systémov si vyžaduje, aby sa na opis funkcií systémov používali grafické jazyky a zápisy, z ktorých možno (manuálne alebo automaticky) odvodiť textové zápisy v implementačných jazykoch. Jedným z perspektívnych a výrazne sa presadzujúcich grafických jazykov je objektovo-orientovaný unifikovaný modelovací jazyk (angl. *Unified Modeling Language, UML*), ktorý poskytuje široký rozsah prostriedkov pre grafický a textový opis funkcií systému (nezávisle od realizácie – HW/SW) abstraktným spôsobom.



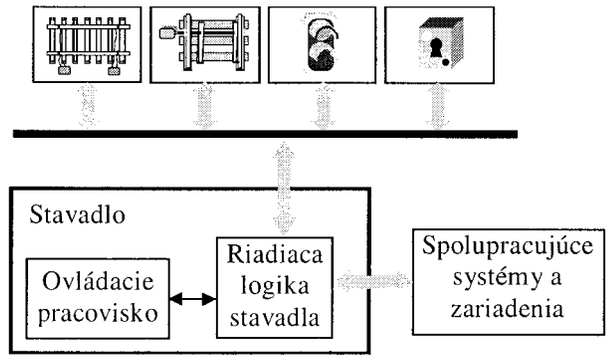
Obr. 1. Diagramy UML vo vzťahu k vývojovým fázam

Fig. 1. UML diagrams in relation to development phases

Štandard *UML* [1,2] ponúka rôzne modelovacie a vizualizačné elementy na zachytenie a modelovanie (špecifikáciu) systémových požiadaviek. Definuje celkom 9 druhov diagramov umožňujúcich opísať vyvíjaný systém z rôznych uhlov pohľadu. Vzťah jednotlivých diagramov k jednotlivým fázam vývojového procesu je naznačený na obr. 1., pri čom hranice zahájenia a ukončenia tvorby jednotlivých diagramov môžu byť predmetom diskusie a treba ich chápať iba orientačne.

2. NEFORMÁLNA ŠPECIFIKÁCIA FUNKČNÝCH POŽIADAVIEK RIADIACEHO OBVODU PRESTAVNÍKA VÝHYBKY

Na prezentáciu objektovo orientovaného prístupu k vývoju systému je použitý zjednodušený príklad riadiaceho obvodu prestavníka výhybky.



Obr. 2. Statická štruktúra systému

Fig. 2. Static structure of the system

Vonkajšie prvky zabezpečovacieho systému (návestidlá, prestavníky výhybiek a výkoľajok, technické prostriedky na zisťovanie voľnosti koľajových úsekov, elektromagnetické zámky, ...) sú cez k nim prislúchajúce obvody výkonového rozhrania a cez zbernicu pripojené na riadiacu logiku stavadla (obr. 2.).

Riadiaci obvod prestavníka výhybky tvorí časť riadiacej logiky stavadla.

Neformálne požiadavky na riadiacu logiku elektricky ovládaného prestavníka výhybky sú špecifikované v technickej norme [3]. Vychádzajúc z príslušných ustanovení normy, riadiaci obvod prestavníka výhybky musí:

a) znemožniť vykonanie príkazu na prestavenie výhybky, ak:
   1. výhybka je uzavretá v jazdnej ceste;
   2. príslušný úsek výhybky nie je voľný a nebol ovplyvnený ovládací prvok pre núdzové prestavenie výhybky;
b) v ktoromkoľvek okamihu prestavovania umožniť zmenu chodu výhybkového prestavníka za predpokladu, že sú rešpektované podmienky podľa bodu a);
c) nedovoliť prerušenie započatého chodu výhybkového prestavníka pri strate informácie o voľnosti výhybkového úseku;
d) po každom násilnom prestavení pohyblivých častí výhybky železničným koľajovým vozidlom (rozreze) znemožniť ďalšie ústredné ovládanie prestavníka až do záznamu jej rozrezu.

Koncová poloha výhybky sa vyhodnotí, ak súčasne:

a) všetky snímače koncovej polohy pohyblivých častí výhybky hlásia požadovanú polohu;
b) nie sú hlásené obidve koncové polohy pohyblivých častí výhybky;
c) sa nezaznamenal rozrez výhybky;
d) sa nedáva príkaz na prestavenie výhybky do opačnej polohy.

Rozrez výhybky sa vyhodnotí, ak súčasne:

a) sa nedáva príkaz na prestavenie výhybky;
b) nie je hlásená koncová poloha výhybky;
c) príslušný výhybkový úsek je obsadený.

3. FORMÁLNA ŠPECIFIKÁCIA FUNKČNÝCH POŽIADAVIEK RIADIACEHO OBVODU PRESTAVNÍKA VÝHYBKY

Uvažovaná špecifikácia funkčných požiadaviek pre daný systém pozostáva z fáz analýzy a návrhu (obr. 1.) a jej výsledkom je nasledujúci súbor diagramov:

- diagramy prípadov použitia (*use case diagrams*);
- diagramy tried a objektov (*class / object diagrams*);
- sekvenčné diagramy (*sequential diagrams*);
- stavové diagramy (*statechart diagrams*).

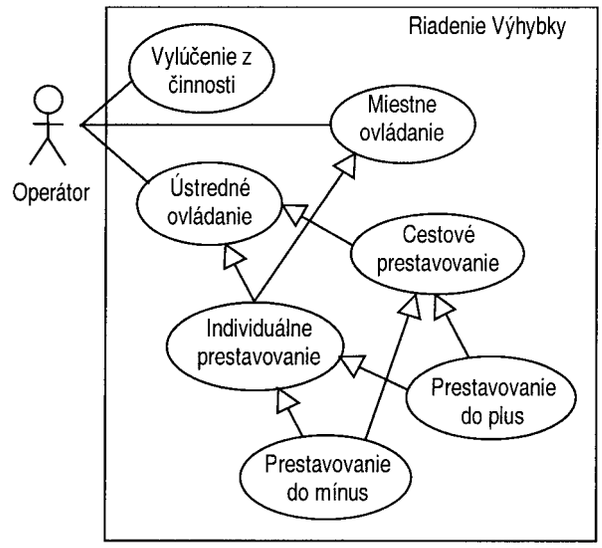
Grafické vyobrazenie diagramov prezentovaných v článku zodpovedá modelu vytvorenému programovým nástrojom *Rhapsody ver. 3.0*. Pre väčšiu zrozumiteľnosť a čitateľnosť boli diagramy prekreslené do prostredia MS Word.

3.1 Diagram prípadov použitia

Na obr. 3. je diagram prípadov použitia riadiaceho obvodu prestavníka výhybky. V diagrame sú definované hranice systému, prípady použitia, aktéri a vzájomné väzby medzi týmito prvkami. Vo funkcii aktéra vystupuje operátor, ktorý môže (uvedené sú len činnosti súvisiace s prestavovaním výhybky):

a) ústredne ovládať výhybku, čo možno realizovať buď nepriamo - príkazom pre postavenie vlakovej cesty (*Cestové prestavovanie*), alebo priamo - individuálnym príkazom na prestavenie konkrétnej výhybky (*Individuálne prestavovanie*);
b) miestne ovládať výhybku (napr. z pomocného stavadla) individuálnym príkazom na prestavenie konkrétnej výhybky (*Individuálne prestavovanie*).

Pri všetkých týchto prípadoch použitia ide o prestavovanie výhybky z jednej koncovej polohy do druhej (*Prestavovanie do mínus*; *Prestavovanie do plus*). Operátor tiež môže vylúčiť výhybku z činnosti alebo ju spätne vrátiť do činnosti (*Vylúčenie z činnosti*). Väzba medzi prípadmi použitia a operátorom je obojsmerná, pretože operátor nielen dáva príkazy na prestavenie výhybky, ale je aj spätne informovaný o jej stave.

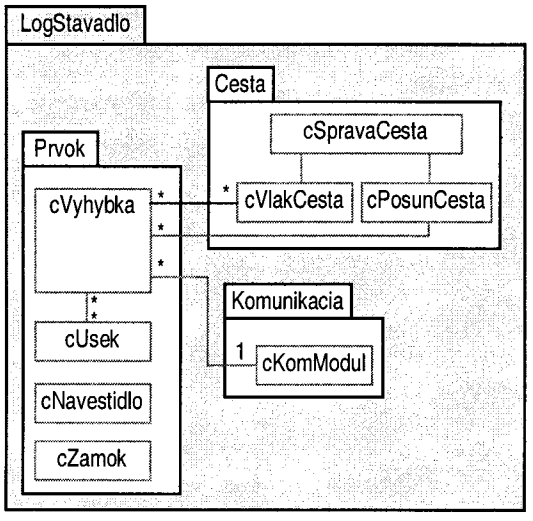


Obr. 3. Diagram prípadov použitia
Fig. 3. Use Case diagram

3.2 Diagram tried

Diagram tried umožňuje opísať statickú štruktúru vyvíjaného systému. Na obr. 4. je zjednodušený diagram tried modelu staničného zabezpečovacieho systému. V diagrame sú z dôvodu prehľadnosti zobrazené len tie triedy a relácie, ktoré sú nutné na opis činností uvedených v tomto príspevku. Package *Cesta* obsahuje triedy *cSpravaCesta*, *cVlakCesta* a *cPosunCesta*. Trieda *cSpravaCesta* generuje jeden objekt *oSpravaCesta*, ktorý realizuje logické väzby medzi jazdnými cestami a zúčastňuje sa na činnosti súvisiacej so zadávaním príkazov na postavenie a rušenie jazdnej cesty. Trieda *cVlakCesta* generuje jeden objekt *oVlakCesta* pre každú možnú vlakovú cestu v stanici. Analogicky, triedou *cPosunCesta* je generovaný počet objektov zodpovedajúci počtu posunových ciest v stanici. Package *Prvok* obsahuje logické závislosti na riadenie a kontrolu jednotlivých prvkov v koľajisku

(návestidlá, prestavníky výhybiek, technické prostriedky na zisťovanie voľnosti koľajových úsekov, elektromagnetické zámky), ktoré sú reprezentované triedami *cVyhybka, cUsek, cNavestidlo* a *cZamok*. Package *Komunikacia* obsahuje triedu *cKomModul* generujúcu objekt *oKomModul, ktorý* umožňuje, okrem iného, prenos informácií medzi objektmi tried z package *Prvok* a vonkajšími zabezpečovacími zariadeniami.



Obr. 4. Diagram tried

Fig. 4. The Class Diagram

Každá trieda je kategóriou alebo skupinou entít, ktoré majú podobné vlastnosti a rovnaké alebo podobné správanie. Správanie entít v danej triede opisujú špecifické operácie (metódy). Vlastnosti entít v danej triede opisujú atribúty. Na obr. 5. je bližšia špecifikácia triedy *cVyhybka*.

LogStavadlo::Prvok::cVyhybka
avPoloha: int avJazdnaDraha: int avOchrannaDraha: int avBočnaOchrana: int avPozadPoloha: int avVyhČíslo: int apCasAutReverz: int
isVyhPouzitie(int pFunkcia, int pPoloha): OM Boolean isVyhPoloha(int pPoloha): OM Boolean isVyhOznacena(): OM Boolean evVyhOznac(int pFunkcia) evVyhCestPrestav(int pPoloha) evVyhPoloha(int pPoloha) evVyhOdznac(int pFunkcia) evVyhVylukaOn() evVyhVylukaOff() evVyhUstrOvladanie() evVyhMiestOvladanie()

Obr. 5. Trieda cVyhybka

Fig. 5. The Class cVyhybka

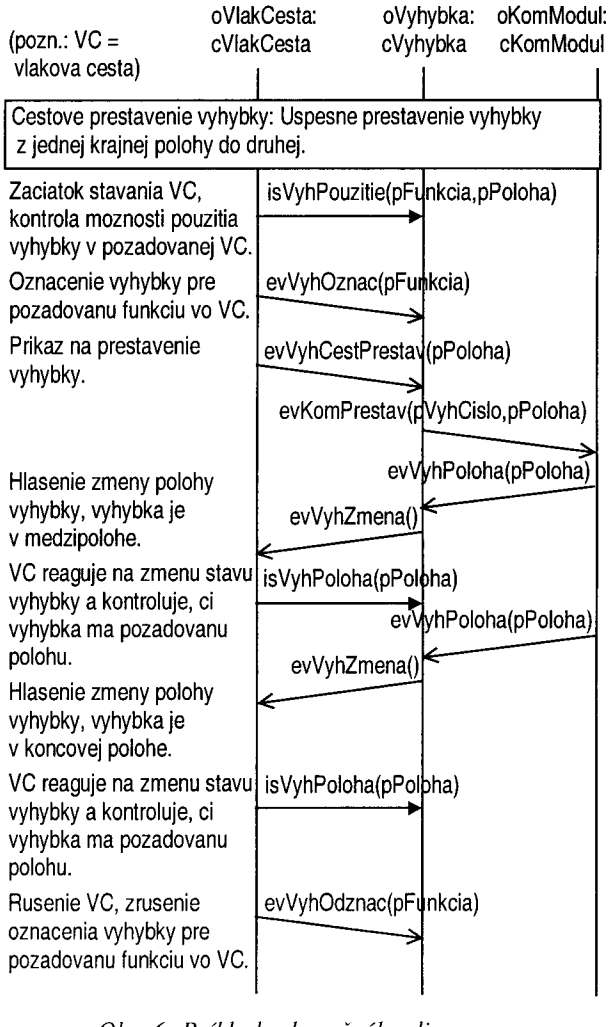
Pri tvorbe názvov atribútov, metód a správ bol použitý nasledovný spôsob značenia:

- *avXXX* ... variabilný atribút, ktorý môže v čase nadobúdať rôzne hodnoty;
- *apXXX* ... projektovateľný atribút, ktorého hodnota je konštantná a vyplýva z projektu konkrétneho systému;
- *gXXX* ... globálny atribút;
- *evXXX* ... udalosť, alebo metóda aktivovaná udalosťou s týmto názvom;
- *isXXX* ... primitívna operácia.

V zátvorkách, ktoré nasledujú za menom operácie alebo udalosti, môžu byť uvedené požadované parametre a ich typy.

3.3 Sekvenčný diagram

Sekvenčné diagramy slúžia na časovo závislý opis spolupráce objektov systému. Treba ich vytvoriť pre všetky možné scenáre týkajúce sa činnosti výhybky.



Obr. 6. Príklad sekvenčného diagramu

Fig. 6. Example of a Sequential Diagram

Na obr. 6. je znázornená časť spolupráce objektov *oVlakCesta, oVyhybka* a *oKomModul* pri cestovom prestavovaní výhybky počas stavania a rušenia

vlakovej cesty. Objekty navzájom spolupracujú prostredníctvom výmeny správ; šikmé čiary reprezentujú asynchrónne správy (udalosti) a vodorovné čiary reprezentujú synchrónne správy (primitívne operácie). V počiatočných fázach stavania vlakovej cesty objekt *oVlakCesta* kontroluje (primitívna operácia *isVyhPouzitie*), či výhybka môže byť použitá vo funkcii, ktorá je určená pri projektovaní vlakovej cesty. Parameter *pFunkcia* obsahuje informáciu o požadovanej funkcii výhybky a parameter *pPoloha* obsahuje informáciu o požadovanej polohe výhybky. Informácia o požadovanej polohe je nutná z toho dôvodu, pretože výhybka môže plniť rôznu funkciu vo viacerých cestách v závislosti od polohy (napr. výhybka môže ležať v jazdnej dráhe jednej vlakovej cesty a pre druhú vlakovú cestu môže tvoriť bočnú ochranu za predpokladu, že požadovaná poloha výhybky v obidvoch vlakových cestách je rovnaká). Ak operácia *isVyhPouzitie* vráti hodnotu *true,* pokračuje sa v stavaní cesty. Objekt *oVlakCesta* vydá povel na označenie výhybky pre požadovanú funkciu (*evVyhOznac*) a na prestavenie výhybky do požadovanej polohy (*evVyhCestPrestav*). Ak operácia *isVyhPouzitie* vráti hodnotu *false,* tak sa stavanie vlakovej cesty zruší. Ak sú splnené podmienky na prestavenie výhybky, objekt *oVyhybka* generuje pre objekt *oKomModul* „preverený" povel na prestavenie výhybky (*evKomPrestav*). Každá zmena polohy výhybky je hlásená objektu *oVyhybka* (*evVyhPoloha*) a následne aj objektu *oVlakCesta* (*evVyhZmena*). Udalosť *evVyhPoloha* obsahuje parameter *pPoloha*, ktorého hodnota závisí od aktuálnej polohy výhybky. Udalosť *evVyhZmena* neobsahuje žiadny parameter. Ak je poloha výhybky pre vlakovú cestu z dôvodu bezpečnosti dôležitá (napr. pred jazdou vlaku), overí si príslušný objekt *oVlakCesta* požadovanú polohu výhybky aktiváciou primitívnej udalosti *isVyhPoloha*. Pri rušení vlakovej cesty vydá objekt *oVlakCesta* povel *evVyhOdznac*, ktorým sa zruší blokovanie výhybky pre funkciu, ktorú výhybka plnila v tejto vlakovej ceste.

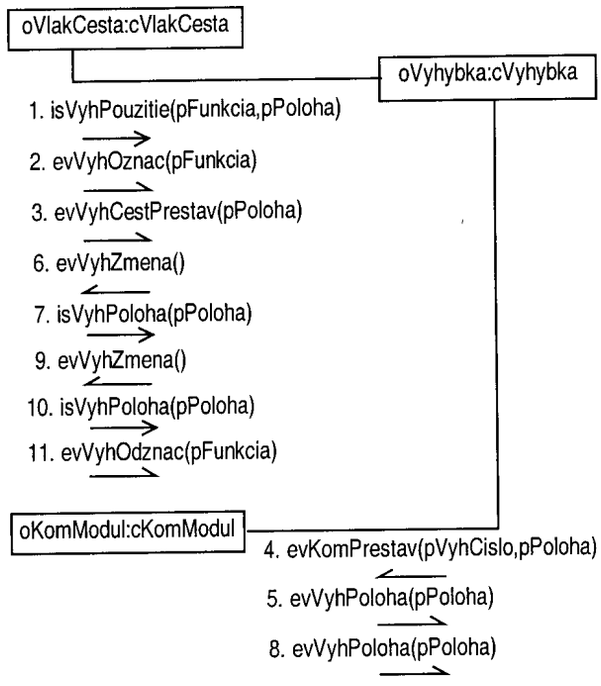
Namiesto sekvenčného diagramu (alebo ako jeho doplnok) možno použiť diagram spolupráce (*collaboration diagram*), ktorý je významovo ekvivalentný. Ak sekvenčný diagram kladie dôraz na poradie jednotlivých udalostí (opisuje, čo sa deje v čase), diagram spolupráce zdôrazňuje kontext a usporiadanie spolupracujúcich objektov (opisuje, čo sa deje v priestore). Sekvenčnému diagramu na obr. 6. by zodpovedal diagramu spolupráce uvedenému na obr. 7.

3.4 Stavový diagram

Ďalším diagramom, ktorý slúži na opis správania sa systému, je stavový diagram. Na obr. 8. a 9. sú stavové diagramy triedy *cVyhybka*.

Stavový diagram na obr. 8. tvoria tri sériové stavy:

- *cVyhybka.Vyhybka.OvladanieVyhybka*;
- *cVyhybka.Vyhybka.PolohaVyhybka*;
- *cVyhybka.Vyhybka.PouzitieVyhybka*.



Obr. 7. Príklad diagramu spolupráce

Fig. 7. Example of a Collaboration diagram

Z hľadiska značenia prechodov zo stavu do stavu je použitá konvencia:

spúšťacia udalosť [podmienka] /
zoznam vyvolaných akcií

Stav *cVyhybka.Vyhybka.OvladanieVyhybka* obsahuje tri substavy, ktoré vyjadrujú, že výhybka môže byť ovládaná miestne (*MiestneOvladanie*), ústredne (*UstredneOvladanie*), alebo môže byť vylúčená z činnosti (*Vyluka*). Udalosti aktivujúce zmenu substavu sú generované objektom *oObsluha*. Prechod z ústredného ovládania na miestne ovládanie je možný len za predpokladu, že výhybka nie je súčasťou žiadnej vlakovej cesty (ak primitívna operácia *isVyhOznacena* vracia hodnotu *true*) .

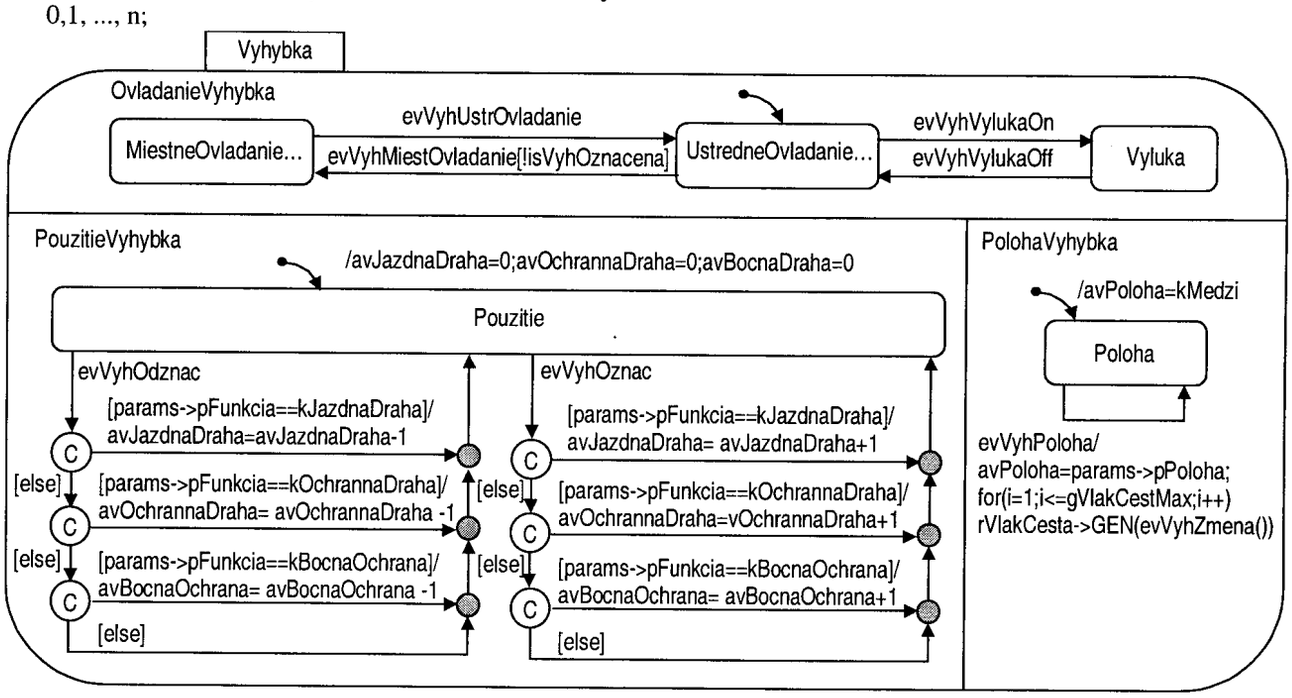
Stav *cVyhybka.Vyhybka.PolohaVyhybka* obsahuje jeden substav (*Poloha*). Tento stav je citlivý na udalosť *evVyhPoloha* generovanú objektom *oKomModul*. Hodnota parametra *pPoloha* tejto udalosti nesie informáciu o stave výhybky, v závislosti od ktorej sa nastaví hodnota atribútu *avPoloha*. Pre všetky objekty *oVlakCesta* sa generuje udalosť *evVyhZmena* informujúca o zmene polohy výhybky. Maximálny počet objektov *oVyhybka* určuje hodnota globálneho atribútu *gVlakCestaMax*.

Stav *cVyhybka.Vyhybka.PouzitieVyhybka* obsahuje jeden substav (*Pouzitie*). V tomto stave dochádza po príjme udalostí *evVyhOznac*, *evVyhOdznac* ku zmene hodnôt atribútov zaznamenávajúcich funkciu (funkcie) výhybky vo vlakovej ceste (vlakových cestách). Výhybka môže plniť vo vlakových cestách tieto funkcie:

- výhybka je súčasťou jazdnej dráhy vlakovej cesty; tejto funkcii výhybky je priradený atribút

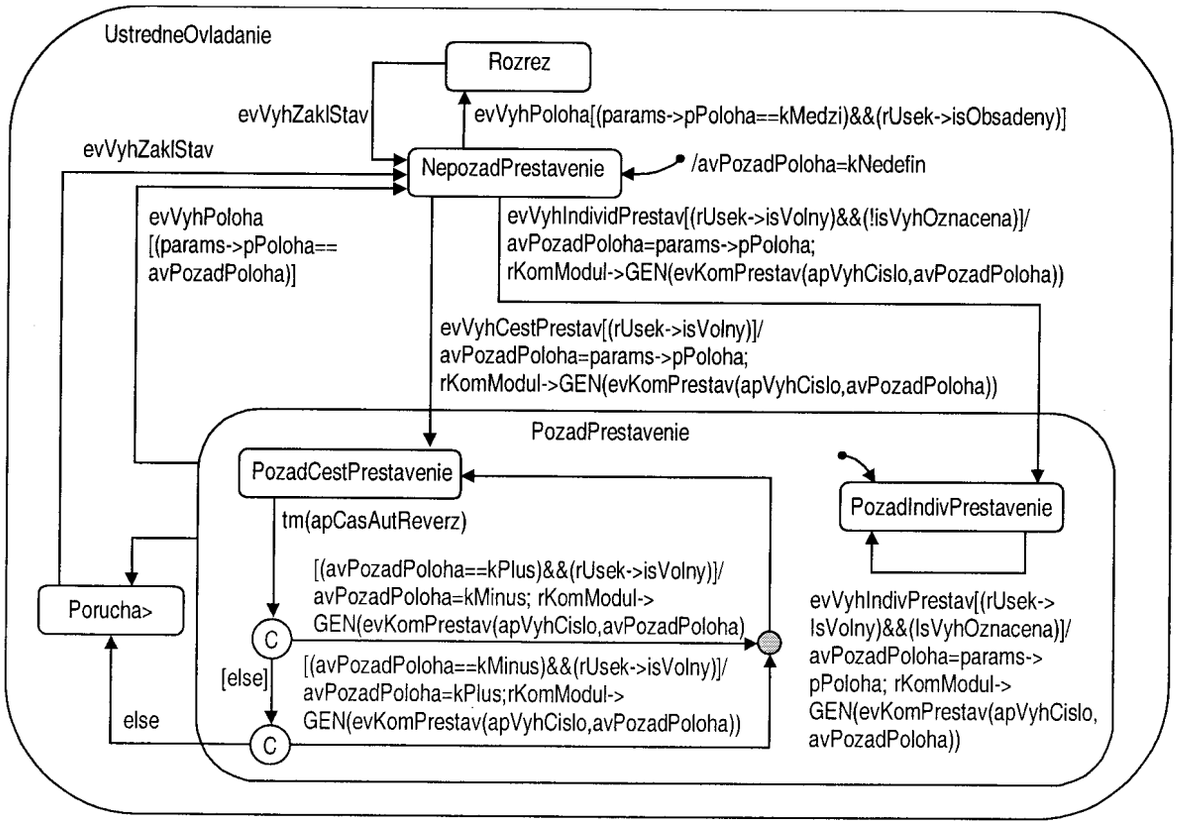
avJazdnaDraha, ktorý môže nadobúdať hodnoty 0,1;

- výhybka je súčasťou ochrannej dráhy vlakovej cesty; tejto funkcii výhybky je priradený atribút *avOchrannaDraha*, ktorý môže nadobúdať hodnoty 0,1, ..., n;
- výhybka tvorí bočnú ochranu vlakovej cesty; tejto funkcii výhybky je priradený atribút *avBocnaOchrana*, ktorý môže nadobúdať hodnoty 0,1, ..., n.



Obr. 8. Stavový diagram triedy Vyhybka

Fig. 8. Statechart of the class Points Switch (Vyhybka)



Obr. 9. Stavový diagram superstavu UstredneOvladanie

Fig. 9. State diagram of the superstate UstredneOvladanie (CentralControl)

Na obr. 9. je stavový diagram superstavu *UstredneOvladanie*, ktorý obsahuje stavy súvisiace s prestavovaním výhybky. Implicitným stavom je stav *NepozadPrestavenie*. Ak je prijatá správa o zmene polohy výhybky (*evVyhPoloha*), pričom výhybka sa nachádza v medzipolohe (*pPoloha==kMedzi*) a výhybkový úsek je obsadený (primitívna operácia *isObsadeny* vracia hodnotu *true*), tak objekt *oVyhybka* prejde do stavu *Rozrez*. Stav *Rozrez* môže opustiť až po prijatí udalosti *evVyhZaklStav*. Stav *NepozadPrestavenie* môže objekt *oVyhybka* opustiť aj po prijatí povelu na prestavenie (*evVyhCestPrestav*, *evVyhIndivPrestav*), ak sú splnené ďalšie podmienky. Napríklad, ak príde povel na prestavenie výhybky (*evVyhCestPrestav*) od objektu *oVlakCesta* a výhybkový úsek je volný (*isVolny*), tak sa vygeneruje povel na prestavenie výhybky (*evKomPrestav*), ktorý postupuje cez komunikačný modul až k výkonovému rozhraniu prestavníka výhybky, a objekt *oVyhybka* prejde do stavu *PozadCestPrestavenie*. Udalosť *evKomPrestav* nesie informáciu o čísle výhybky (*apVyhCislo*) a informáciu o požadovanej polohe (*avPozadPoloha*). V procese prestavovania výhybky môžu nastať tieto scenáre:

- výhybka dosiahne požadovanú koncovú polohu; ak je prijatá udalosť *evVyhPoloha* a aktuálna poloha výhybky sa zhoduje s požadovanou polohou (*pPoloha==avPozadPoloha*), tak objekt *oVyhybka* sa vráti do stavu *NepozadPrestavenie*;
- výhybka v stanovenom čase, ktorý je určený hodnotou atribútu *apCasAutReverz* nedosiahne požadovanú koncovú polohu; vygeneruje sa povel na automatickú reverzáciu výhybky, pričom sa zmení požadovaná koncová poloha; objekt *oVyhybka* zostáva v stave *PozadCestPrestavenie*;
- objekt *oVyhybka* príjme udalosť *evVyhPorucha*; udalosť *evVyhPorucha* je generovaná objektom *oKomModul* vtedy, ak v stanovenom časovom intervale nepríde hlásenie o polohe výhybky od obvodu výkonového rozhrania alebo ak je hlásená porucha; objekt *oVyhybka* prejde do stavu *Porucha*, ktorý môže opustiť až po prijatí udalosti *evVyhZaklStav*;

Obdobnú činnosť možno vysledovať aj pri príkaze na individuálne prestavenie výhybky. V tomto prípade však nie je možná automatická reverzácia.

Funkcie realizované primitívnymi operáciami triedy *cVyhybka* majú nasledovnú implementáciu vo verzii C++:

Kód primitívnej operácie *isVyhPoloha(pPoloha)*

```
if(avPoloha==pPoloha)
   return true;
   else
   return false;
```

Kód primitívnej operácie *isVyhOznacena()*

```
if((avJazdnaDraha==0)&&(avOchrannaDraha==
0)&&(avBocnaOchrana==0))
   return false;
   else
   return true;
```

Kód primitívnej operácie *isVyhPouzitie(pFunkcia, pPoloha)*

```
switch (pFunkcia)
  {
    case kJazdnaDraha:
if((avJazdnaDraha==0)&&(avOchrannaDraha==
0)&&((avBocnaOchrana==0)||(pPoloha==avPol
oha)))
         return true;
         else
         return false;
     case kOchrannaDraha:
if((avJazdnaDraha==0)&&((avBocnaOchrana==
0)||(pPoloha==avPoloha)))
         return true;
         else
         return false;
     case kBocnaOchrana:
if((avJazdnaDraha==0)||(pPoloha==avPoloha
))
         return true;
         else
         return false;
    default: return false;
   }
```

4. ZÁVER

Hlavným cieľom článku bolo poukázať na spôsob, akým možno aplikovať objektovo–orientovaný prístup k vývoju železničného zabezpečovacieho systému. Vychádzajúc z praktických skúseností s *UML*, ktoré autori v posledných rokoch nadobudli v rámci riešenia konkrétnych problémov železničnej praxe, možno zhrnúť skúsenosti s *UML* a vyzdvihnúť jeho prednosti nasledovne:

- v porovnaní s neformálnym prístupom je výsledná špecifikácia funkčných požiadaviek na vyvíjaný systém jednoznačná, úplná a dobre štruktúrovateľná;
- vytvorená *UML* špecifikácia môže slúžiť ako základ pre komunikáciu rôznych vývojových tímov pracujúcich nad spoločným projektom; uvedené tvrdenie sa vzťahuje aj na subjekty aktívne v procese posudzovania a schvaľovania bezpečnosti vyvíjaného systému;
- použitím vhodného softvérového nástroja s možnosťou automatického generovania dokumentov možno unifikovať princípy tvorby dokumentácie;
- v porovnaní s inými formalizmami možno za nespornú výhodu považovať všeobecnú dostupnosť a zrozumiteľnosť štandardu *UML* a minimálne náklady potrebné na jeho osvojenie si;
- nezávislosť vytváranej špecifikácie od konkrétneho programovacieho jazyka (väzba na konkrétny typ sa uplatní najskôr pri automatickom generovaní zdrojového kódu, označovaného ako proces tzv. dopredného inžinierstva);

- možnosť verifikácie funkčnosti vytvoreného modelu prostredníctvom animácie a možnosť predviesť prototyp systému zákazníkovi bezprostredne po ukončení fázy špecifikácie;
- ľahšia údržba a realizácia prípadných neskorších zmien a zásahov do špecifikácie vytvorenej v *UML* v porovnaní s neformálnu špecifikáciou a/alebo špecifikáciou vytvorenou inými formalizmami;
- nezanedbateľná nie je ani permanentná podpora štandardu *UML* zo strany hlavných dodávateľov softvérových nástrojov.

Záverom sa treba dotknúť problematiky správnosti špecifikácií vytvorených pomocou *UML* a možnostiach ich verifikácie. *UML* je v súčasnosti zaradený do skupiny tzv. poloformálnych metód, ktoré na rozdiel od formálnych metód neumožňujú poskytnúť matematický (vypočítaný) dôkaz o schopnosti alebo neschopnosti systému dostať sa do určitého nebezpečného stavu alebo obnoviť bezpečnosť za abnormálnych situácií. Pretvorenie alebo rozšírenie *UML* o potrebný matematický základ je predmetom intenzívneho bádania, ktoré sa uberá rôznymi smermi: osobitná pozornosť sa venuje možnostiam transformácie *UML* modelov do štandardných formálnych modelov [4], vývoju špecializovaných softvérových nástrojov a možnostiam automatického prepisu vybraných *UML* diagramov do niektorých formálnych jazykov, napr. *VDM++* [5], *B* [6], *Z* [7], v snahe formalizovať sémantiku *UML* – napríklad vytvorením tzv. „presného *UML*" (angl. *precise UML, pUML*) [8]. Trendy budúceho vývoja v používaní *UML* na špecifikáciu požiadaviek bezpečnostne kritických systémov sú v týchto súvislostiach zjavné.

Článok vznikol za podpory grantovej agentúry Slovenskej republiky *VEGA*, č. grantu *1/8182/01 „Teoretické podklady pre výpočet akceptovateľného rizika v riadení dopravného procesu, najmä železničného"*.

LITERATÚRA

[1] Object Management Group. *OMG Unified Modeling Language Specification*. Dostupné na: http://www.omg.org

[2] ISO/IEC DIS 19501-1: Information Technology - Unified Modeling Language (UML) - Part 1: Specification. Edition 1, 2000, 1024 s.

[3] TNŽ 342620: Predpisy pre staničné zabezpečovacie zariadenia. 1996.

[4] BRUEL, J.-M. – FRANCE, R. B.: Transforming UML models to formal specifications. In: *UML'98*, vol. 1618 of LNCS. Springer Verlag, 1998.

[5] VERHOEF, M. - Van den BERG, M.: Formal specification of an auctioning system using VDM++ and UML (an industrial usage report). *Proceedings of the VDM Workshop at FM'99*, Toulouse, 1999.

[6] LALEAU, R. – MAMMAR, A.: An overview of a method and its support tool for generating B specifications from UML notations. In: *ASE: 15th IEEE Conference on Automated Software Engineering, Grenoble, France*, IEEE Computer Society Press, September 2000.

[7] DUPUY, S. – LEDRU, Y. – CHABRE-PECCOUD, M.: Overview of Roz: A tool for integrating UML and Z specifications. In: *CAiSE00: 12th International Conference on Advanced Information Systems Engineering, Stockholm, Sweden,* vol. 1789 of LNCS. Springer Verlag, June 2000.

[8] EVANS, A. S. – KENT, S.: Meta-modelling semantics of UML: the pUML approach. In: *2nd International Conference on the Unified Modeling Language*, Colorado, vol. 1723 LNCS, 1999.