2008/33 – 16. 6. 2009

elektrorevue
ISSN 1213 - 1539

MODEL PRO POPIS SDÍLENÝCH PROSTŘEDKŮ A JEJICH ZÁVISLOSTÍ VYCHÁZEJÍCÍ Z OOP PRINCIPŮ

Ing. Martin Vítek, Ing. Ivo Herman, CSc.
Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií
Ústav telekomunikací, Purkyňova 118, 612 00 Brno, Česká republika
Email: vitekm@feec.vutbr.cz, herman@feec.vutbr.cz

Součástí každého moderního objektově orientovaného programovacího (OOP) jazyka jsou techniky, pomocí kterých lze řídit přístup ke sdíleným prostředkům (objektům) při paralelním využívání tohoto prostředku více procesy nebo vlákny. Tyto techniky jsou obvykle založeny na uzamčení objektů a jejich alokaci danému vláknu, což při nevhodném pořadí těchto alokací v prostředích, kde jsou splněny Coffmanovy podmínky, může vést ke stavu uváznutí. Proto je vhodné řešit sdílení prostředků na vyšší úrovni abstrakce. Navrhovaný model popsaný v tomto článku představuje vlastní formální popis prostředků vycházející z principů OOP (aby jej bylo možné snadno a přirozeně implementovat) a ukazuje základní algoritmus zamezující vzniku stavu uváznutí.

1. SDÍLENÍ PROSTŘEDKŮ

Jedním ze základních vlastností počítačových systémů je schopnost efektivně využívat prostředky. V ideálním případě je každému požadavku přiřazen vlastní prostředek, který daný požadavek zpracovává. Jedná se o nejjednodušší a také doporučovaný případ využívání prostředků v systémech, obzvláště pokud se jedná o bezestavové prostředky [1].

Ne vždy lze však této podmínce vyhovět. Jedná se především o případy, kdy

- není možné každému požadavku vytvořit vlastní prostředek z důvodu softwarových nebo hardwarových nároků, který prostředek vyžaduje,
- je potřeba sdílet a udržovat data napříč mezi požadavky pocházející od různých procesů, tj. obvykle se jedná o nějaký centrální objekt.

Přístup ke sdíleným prostředkům může být řešen programově na různých úrovních systému.

1.1. VZÁJEMNÉ VYLOUČENÍ

Nejčastěji používanou technikou řešení přístupu ke sdíleným prostředkům je technika vzájemného vyloučení (*mutual exclusion*). Tato technika obvykle vyžívá speciálních klíčových slov k vymezení tzv. kritického úseku kódu, ve kterém je zaručeno, že tento úsek bude zpracovávat jen jedno vlákno současně.

Využívání kritických sekcí sebou nese řadu úskalí, kterou je potřeba mít na paměti a vyvarovat se jich. O některých z nich, konkrétně pro jazyk JAVA, pojednává článek [2].

Jedním z nejméně příjemným důsledkem nesprávného použití této techniky je vznik stavu uváznutí, tj. stavu, kdy dojde ke křížovému čekání na uvolnění kritické sekce. K uváznutí dojde, pokud jsou splněny následující podmínky [3] (zde se místo pojmu vlákno užívá obecnějšího pojmu proces):

- vzájemné vyloučení (*Mutual exclusion*) - prostředek může být v jednom okamžiku používán pouze jedním procesem.
- drž a čekej (*Hold & wait*) - proces může žádat o přidělení dalších prostředků, i když již má přidělen alespoň jeden prostředek.
- neodnímatelnost (*No preemption*) – uvolnění prostředku je plně v kompetenci procesu.
- čekání do kruhu (*Circular wait*) - v grafu závislosti procesů a prostředků může vzniknout cyklus.

Není-li alespoň jedna ze zmíněných podmínek splněna v žádném okamžiku běhu programu, nedojde ke stavu uváznutí. Možné techniky jsou [3]:

- každý proces si zažádá o všechny potřebné zdroje, a dokud je nedostane, tak bude pozastaveno.
- pokud má proces alokovaný nějaký prostředek, není mu dovoleno alokovat jiný, dokud původní prostředek neuvolní.
- seřazení prostředků a povolení procesu alokovat pouze prostředky, které jsou následnými prostředky prostředků již alokovaných daným procesem.

1.2. INSTRUKCE A API ROZHRANÍ

Další technika pro paralelní přístup k prostředkům, obecně řečeno k datům v paměti, je postavená na speciálních instrukcích nebo API rozhraní realizující přístup k paměti procesu.

Jedná se o techniky MCAS (Multi-word Compare-swap), WSTM (Word-based software transactional memory) a OSTM (Object-based software transactional memory) [4]. Tyto techniky jsou založeny na využívání speciálního aplikačního rozhraní API a v případě MCAS a WSTM pracují až na úrovni paměti. Princip těchto technik využívající atomické instrukce pro práci s paměti popisuje článek [5], kde je popsána implementace LIFO a FIFO zásobníků bez nutnosti využití instrukce pro uzamykání kritické sekce.

Vzhledem k tomu, že tyto techniky pracují až na úrovni paměti, není jejich využití v rámci OOP příliš výhodné, obzvláště v případech moderních OOP prostředích, kde

elektrorevue
ISSN 1213 - 1539

řízení paměti provádí běhové prostředí, jako je platforma .NET nebo JAVA.

1.3. Jazykové rozšíření a podpora

Komfortní technikou pro řešení paralelního přístupu ke sdíleným prostředkům je využití některého z programovacích jazyků, který tuto podporu již v sobě obsahuje jako např. Erlang, Ada, nebo jazyka se syntaktickým rozšířením stávajících jazyků (Split-C, Cilk) [6].

Tyto jazyky umožňují velmi pohodlně řídit a spravovat paralelně pracující kód, ale na druhou stranu nepatří mezi rozšířené techniky a tím pádem nedisponují takovými integračními schopnostmi s ostatními technologiemi jako standardní OOP programovací jazyky.

1.4. Formální verifikace systémů

Jiným přístupem při návrhu a implementaci systémů (obzvláště složitějších, kam multivláknové systémy patří) je jejich ověření prostřednictvím formální analýzy a verifikace, což jsou metody, které v poslední době patří mezi rychle se rozšiřující techniky ověřování správnosti systémů, kam samozřejmě patří i ověření, že se v systému nevyskytuje stav uváznutí [7]. Formální verifikace založená např. na formální specifikaci systému prostřednictvím temporální logiky se používá především tam, kde nelze použít mechanismus prevence nebo detekce a následného zotavení se stavu uváznutí. Jedná se tedy především o ověření korektnosti systému prostřednictvím jeho modelu nebo i systému samotného na základě analýzy, avšak obvykle ještě před jeho zprovozněním v produkčním prostředí.

Protože součástí této práce je návrh modelu, jehož implementace má usnadnit vývoj systému a následně se uplatnit při jeho činnosti, není problematika formální verifikace jakožto nástroje pro ověření správnosti systému v této práci dále studována.

1.5. Programové modely

Programové modely definují postupy a pravidla pro sdílení prostředků. Mezi implementace těchto modelů patří např. Reo, Esterel, SIGNAL nebo Lustre [6].

Programové modely se obvykle používají ve fázi návrhu systému a pro vlastní systém je potřeba pak následně programový model nebo jeho část implementovat, což nemusí být vždy snadné např. z důvodu, že model využívá specifických programových technik.

Tento článek popisuje model pro popis sdílených prostředků a závislostí. Při návrhu modelu bylo snahou, aby tento model vycházel z principů OOP a poskytoval nástroje pro řešení situací při práci se sdílenými prostředky, zejména pak minimalizoval vznik stavu uváznutí.

Součástí návrhu modelu je prezentován jednoduchý algoritmus zaručující, že při korektním definování vazeb nedojde ke stavu uváznutí.

2. Model pro popis sdílených prostředků

Tato kapitola se zabývá matematickým popisem principů vlastního navrhovaného modelu pro popis sdílených prostředků, kdy jsou postupně definovány jednotlivé elementy a vzájemné vztahy mezi nimi. Na základě těchto vazeb se pak provádí alokace sdílených prostředků za účelem minimalizovat pravděpodobnost vzniku stavu uváznutí.

2.1. Prostředek a rozhraní

Rozhraní je předpis, který definuje poskytované služby ve formě metody nebo vlastnosti. Metoda obsahuje název, seznam vstupních parametrů a volitelný výstupní parametr neboli návratovou hodnotu. Vlastnost je v podstatě dvojice metod, z nichž jedna je pro nastavení hodnoty vlastnosti (metoda nemá návratovou hodnotu, ale musí mít alespoň jeden vstupní parametr stejný jako je typ vlastnosti) a druhá pro získání hodnoty vlastnosti (návratová hodnota metody se shoduje s typem vlastnosti). Pokud metody vlastnosti obsahují další parametry, jsou využity jako indexy vlastnosti. Vlastnost může obsahovat i jen jednu z dvojice metod. Pak máme vlastnost jen pro čtení, resp. pro zápis.

Prostředek představuje ucelenou jednotku poskytující svému okolí služby prostřednictvím svých rozhraní. Systém může obsahovat několik typů prostředků, kdy každý prostředek musí mít definované alespoň jedno rozhraní, neboť služeb prostředku lze využívat jen prostřednictvím jeho rozhraní.

Mějme systém $\mathcal{S}$, který obsahuje n různých typů prostředků. Pak definujeme množinu R, jejíž prvky jsou právě jednotlivé typy prostředků:

$$R = \{r_i | 0 < i \leq \mathrm{n}\}. \tag{1}$$

Dále definujeme množinu I, obsahující všechny rozhraní prostředků v systému:

$$I = \{\mathrm{i}_j | 0 < j \leq \mathrm{m}\}, \tag{2}$$

kde m je počet všech rozhraní prostředků v systému.

Každý prostředek může obsahovat jedno nebo více rozhraní. Množinu rozhraní pro jeden r-tý typ prostředku je dána vztahem:

$$I(r) = \{i_l(r) | 0 < l \leq \mathrm{q}, i_l(r) \in I\}, \tag{3}$$

kde q je celkový počet rozhraní prostředku r. Pro množiny I(r) a I tedy platí následující vztahy:

$$I(r) \subset I, \tag{4}$$

$$I = \{i_l(r) | 0 < l \leq \mathrm{q}, 0 < r \leq \mathrm{n}\}. \tag{5}$$

2.2. Závislost rozhraní

Závislost mezi rozhraními definujeme následujícím symbolem:

$$\mathbf{b}_{r,l}^{\varrho,\lambda} \approx \mathrm{i}_l(r) \rightarrow \mathrm{i}_\lambda(\varrho), \tag{6}$$

elektrorevue
ISSN 1213 - 1539

který říká, že rozhraní $i_l(r)$, tj. l-té rozhraní r-tého prostředku, využívá rozhraní $i_\lambda(\rho)$, tj. λ-té rozhraní ρ-tého prostředku. Množinu všech využívaných rozhraní rozhraním $i_l(r)$ definujeme jako:

$$B(i_l(r)) = \left\{\forall \mathrm{i}_\lambda(\varrho) \text{ z } \mathbf{b}_{r,l}^{\delta,\lambda} \middle| r, l = \text{konst.}\right\}. \quad (7)$$

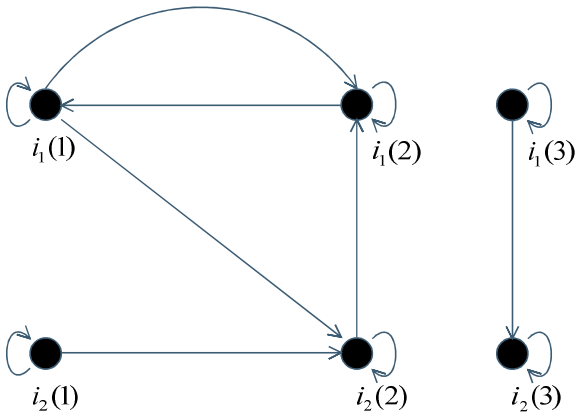
Pokud nás nezajímá od jakého prostředku rozhraní pochází píšeme jen $B(i_l)$.

Pro snadnější práci a názornost je lepší si závislosti rozhraní popsat prostřednictvím grafu závislosti, což je orientovaný graf, který zobrazuje závislosti mezi jednotlivými rozhraními prostředků. Uzly grafu jsou tvořeny rozhraními, orientované hrany grafu odpovídají závislosti mezi dvěma rozhraními. Mezi dvěma rozhraními mohou být dvě různě orientované hrany, což značí situaci, kdy první rozhraní využívá druhé a naopak.

Mějme tedy graf závislosti rozhraní **Q** definovaný předpisem:

$$\mathbf{Q} = (I, B), \quad (8)$$

kde I je množina všech rozhraní (od všech zdrojů) a B je množina všech závislostí mezi rozhraními obsažených v množině I. Ukázka grafu závislosti rozhraní pro 6 rozhraní od 3 prostředků je zobrazena na obr. 1. Odtud jde vidět, že každé rozhraní má definovanou minimálně jednu hranu, která říká, že rozhraní využívá samo sebe.



Obr. č. 1: Graf závislosti rozhraní

2.3. SYNCHRONIZAČNÍ SKUPINA

Prostřednictvím grafů závislostí lze určit skupiny spolu souvisejících vlastností. Skupiny budeme nazývat synchronizační. Synchronizační skupinu definujeme jako množinu spolu souvisejících rozhraní:

$$G = \{i | i \in I\}. \quad (10)$$

Dále definujeme množinu synchronizačních skupin:

$$\Gamma = \{G_k\}, \quad (11)$$

kde G_k je k-tá synchronizační skupina. Algoritmus pro tvorbu synchronizačních skupin, tj. množiny Γ je následující:

1) Z množiny všech rozhraní I vybereme j-té rozhraní i_j.
2) V množině Γ najdeme synchronizační skupinu G_k takovou, že $i_j \in G_k$.
3) Pokud v bodě 2 nebyla nalezena žádná synchronizační skupina, vytvoříme novou synchronizační skupinu G_k takovou, že bude obsahovat všechny závislé rozhraní od i_j, tj. $G_k = \{B(i_k)\}$, $\Gamma = \Gamma \cap \{G_k\}$. Dále pokračujeme bodem 1 pro rozhraní i_{j+1}.
4) Pokud v bodě 2 je nalezena synchronizační skupina G_k, pak pro všechna závislá rozhraní z množiny $B(i_k)$, $\forall i_b \in B(i_k)$ najdeme synchronizační skupinu G_b takovou, že $i_b \in G_b$
   a. Pokud taková množina G_b neexistuje, pak přidáme rozhraní i_b do synchronizační skupiny G_k, tj. $G_k = G_k \cap \{i_b\}$. Dále pokračujeme bodem 1 pro rozhraní i_{j+1}.
   b. Pokud byla nalezena množina G_b, pak vložíme všechny rozhraní z množiny G_b do množiny G_k a množinu G_b zrušíme, tj. $G_k = G_k \cap G_b$, $\Gamma = \Gamma / \{G_b\}$. Dále pokračujeme bodem 1 pro rozhraní i_{j+1}.

Pro naše účely je vhodné množinu Γ přepsat jako dvojice hodnot rozhraní a synchronizační skupina obsahující toto rozhraní, tj. definujeme si následující množinu:

$$W = \left\{(i_j, G_k) \middle| i_j \in I \wedge i_j \in G_k\right\}. \quad (12)$$

Prostřednictvím množiny W snadno zjistíme všechna závislá rozhraní pro dané rozhraní. Zápisem $W(i)$ budeme rozumět synchronizační skupinu G_k z dvojice (i_j, G_k) . Pro graf závislosti rozhraní z obr. 1 by množina W vypadala takto:

W = {
($i_1(1)$, { $i_1(1)$, $i_2(1)$, $i_1(2)$, $i_2(2)$}),
($i_2(1)$, { $i_1(1)$, $i_2(1)$, $i_1(2)$, $i_2(2)$}),
($i_1(2)$, { $i_1(1)$, $i_2(1)$, $i_1(2)$, $i_2(2)$}),
($i_2(2)$, { $i_1(1)$, $i_2(1)$, $i_1(2)$, $i_2(2)$}),
($i_1(3)$, { $i_1(3)$, $i_2(3)$}),
($i_2(3)$, { $i_1(3)$, $i_2(3)$})
}.

Zde vidíme, že synchronizační skupiny sdružují všechny rozhraní, mezi kterými je nějaká vazba, ať přímá, či nepřímá. Jedná se v podstatě o nejjednodušší realizaci synchronizačních skupin, která nebere v potaz orientaci hran a posloupnost závislostí v grafu.

elektrorevue
ISSN 1213 - 1539

2.4. METODY, VLASTNOSTI A UDÁLOSTI ROZHRANÍ

Mějme rozhraní *i*. Zápisem $m_l(i)$ rozumíme *l*-tou metodu rozhraní *i*. Volání metody $m_l(i)$ budeme zapisovat, jako funkci $inv(m_l(i))$. Protože vlastnost v podstatě představuje jiný zápis metody (pokud je vlastnost jen pro čtení, nebo zápis), případně dvojice metod (pokud vlastnost je jak pro čtení, tak i zápis), nebudeme ji zvlášť definovat. Množina všech metod rozhraní je definován jako:

$$M(i) = \{\mathrm{m}_l(i) | 0 < l \leq p\}, \tag{13}$$

kde *p* je celkový počet metod rozhraní *i*.

Rozhraní může obsahovat událost, což je přípojný bod, kam se mohou připojovat zdroje pomocí svých rozhraní. Typ události definuje, jaké rozhraní může být k události připojeno. Toto rozhraní budeme označovat také jako událostní rozhraní. Pokud dojde uvnitř rozhraní k vyvolání událostního rozhraní, zavolají se postupně všechny rozhraní připojené k dané události. Událost definujeme předpisem $e[i'](i)$, který značí událost *e* rozhraní *i*, přičemž k této události mohou být připojeny prostředky mající rozhraní *i´*. Říkáme, že typ události je i´. Pokud nás nezajímá typ rozhraní píšeme jen $e(i)$, čehož využijeme i pro definici množiny všech událostí rozhraní *i*:

$$E(i) = \{e(i)\}. \tag{14}$$

2.5. SUBJEKT

Prostředek představuje určitou definici určitého typu objektu mající definované rozhraní a vazby na ostatní rozhraní, ať už se jedná o závislá rozhraní nebo o rozhraní událostí. Konkrétní výskyt tohoto prostředku budeme nazývat subjektem. Subjekt je tedy instancí prostředku.

Mějme subjekt *s*, který je typu *r*. Tuto skutečnost budeme zapisovat jako $s(r)$. Dále definujme množiny všech subjektů v systému $S=\{s_m\}$. Pro subjekt $s(r)$ dále stanovíme funkci τ pro získání typu subjektu, pro kterou platí:

$$\tau(s(r)) = r. \tag{15}$$

Subjekt $s(r)$ musí implementovat všechna rozhraní, která má prostředek definované, tj. $I(s) = I(r)$. Pokud rozhraní *i* prostředku *r* obsahuje určité událostní rozhraní i_e, tj. $E[i_e](i) \neq \emptyset$, pak subjekt *s* umožňuje připojení libovolné počtu subjektů *s´* implementující i_e, tj. $i_e \in I(s')$. Subjekt *s* pak pro událostní rozhraní i_e obsahuje množinu připojených subjektů. Tuto množinu definujeme následovně:

$$H[i_e](i(s)) = \{s_m | i_e \in I(s_m)\}. \tag{16}$$

Množinu všech připojených subjektů ke všem událostním rozhraním z množiny $E(i)$ budeme zapisovat jako $H(i)$.

2.6. DYNAMICKÁ SYNCHRONIZAČNÍ SKUPINA

V kapitole 2.3 byla definována synchronizační skupina na základě závislosti rozhraní. Tyto synchronizační skupiny budeme nazývat *statickými*, neboť jsou neměnné, jakmile jsou v systému definovány prostředky a závislosti jejich rozhraní.

Z předchozí kapitoly víme, že k událostnímu rozhraní subjektu můžeme připojit další subjekty, přičemž každý subjekt, resp. jeho rozhraní spadá do určité statické synchronizační skupiny. Statická synchronizační skupina určitého rozhraní *i* je tak rozšířena o statickou synchronizační skupinu subjektů připojených k událostním rozhraním definované pro rozhraní *i*. Jelikož připojovat a odpojovat subjekty k událostním rozhraním je umožněno za běhu aplikace, dynamicky tak vznikají u každého rozhraní obsahující událostní rozhraní další související synchronizační skupiny. Tyto skupiny budeme nazývat *dynamické* synchronizační skupiny.

Dynamická synchronizační skupina rozhraní *i* mající množinu událostních rozhraní $E(i)$ je tvořena množinou všech statických synchronizačních skupin subjektů připojených k některé události z $E(i)$ a píšeme:

$$D\big(i(s)\big) =$$

$$\{G_h | \forall i_e \in E(i) \wedge i_e \in G_h \wedge H[i_e](i(s)) \neq \emptyset\}. \tag{17}$$

2.7. GARANCE DOSTUPNOSTI PROSTŘEDKŮ

Prostřednictvím statických a dynamických skupin lze zaručit dostupnost všech vyžadovaných prostředků při požadavku na určité rozhraní libovolného prostředku v systému. Tento požadavek velmi snadno splníme, pokud při požadavku na rozhraní $i(s)$ garantujeme exkluzivní přístup ke všem synchronizačním skupinám, tj. provedeme jejich uzamčení po dobu zpracování požadavku a píšeme:

$$lock(i(s)) = lock(W(i) \cup D(i(s))). \tag{18}$$

Rovnice (18) je jedním z požadavků pro zajištění, aby nedošlo ke stavu uváznutí zmíněných v kap. 1.1. Jedná se o podmínku, kdy každé vlákno si zažádá o všechny potřebné zdroje, a dokud je nedostane, tak bude pozastaveno. Navržený algoritmus zabraňuje vzniku stavu váznutí v systému mezi prostředky mající korektně definované závislosti. Zápis $W(i)$ v rovnici (18) nám říká, že v systému dojde k totálnímu uzamčení všech subjektů implementující, nebo mající jakoukoliv závislost na rozhraní *i*.

Zamykání všech souvisejících subjektů se může zdát jako nevýhodné z důvodu výkonnosti systému, ale přínosem je na druhé straně zamezení stavu uváznutí a spuštění provádění požadavku pouze v případě, že je garantována dostupnost prostředků. Požadavek je proveden buď celý, nebo vůbec. Jedná se o transakční zpracování na základní úrovni, neřešící chyby ve zpracování požadavku vzniklé v důsledku nekorektní činnosti systému, např. výpadek síťové komunikace následně po garantování dostupnosti prostředků. Rozsáhlému zamykání prostředků lze samozřejmě zabránit vhodným návrhem prostředků a jejich rozhraní, např. rozhozením prostředků do jednotlivých regionů.

elektrorevue
ISSN 1213 - 1539

Region představuje oblast v systému, ve kterém jsou právě definovány prostředky a vytvářeny subjekty. Systém $\mathcal{S}$ je tvořenou množinou regionů $\mathcal{R}$, které obsahují definice prostředků r a instance subjektů s, tj. $\mathcal{S} = \{ \mathcal{R}_l \}$, $\mathcal{R} = \{\{r_l\},\{s_m\}\}$.

Region je právě ta oblast systému, ve které dochází k tvorbě statických synchronizačních skupin. Princip zamykání statických synchronizačních skupin je tedy omezen jen na daný region. Toho je docíleno tím, že není dovoleno, aby rozhraní z jednoho regionu bylo definováno jako závislé rozhraní pro rozhraní z druhého regionu. Princip zamykání na rozhraní regionů je tedy stejný jako princip zamykání dvou zcela nezávislých rozhraní.

Mějme definovány dva subjekty s_1 a s_2 nacházející se každý v jiném regionu. Subjekt s_1 bude typu r_1 obsahující rozhraní i_1 a subjekt s_2 bude typu r_2 obsahující rozhraní i_2. Dále řekneme, že někde uvnitř subjektu s_1 se při vykonávaní požadavku rozhraní i_1 využívá rozhraní i_2. Jelikož v definici prostředku r_1 není uvedena (a ani nemůže být z důvodu umístění každého subjektu do jiného regionu) závislost mezi rozhraními i_1 a i_2, bude zamykání rozhraní probíhat postupně, tj. nejprve dojde k zamčení statické synchronizační skupiny rozhraní i_1 a až při požadavku na rozraní i_2 bude zamčena statická synchronizační skupina tohoto rozhraní:

$$lock(i_1) = lock(W(i_1)) \rightarrow lock(W(i_2)). \qquad (19)$$

Rovnice (19) porušuje podmínku vyplývající z rovnice (18), tj. alokaci všech prostředků před vykonáním požadavku. Pokud by bylo definováno, že v rámci rozhraní i_2 se využívá rozhraní i_1, může dojít je stavu uváznutí, neboť pak bude platit:

$$lock(i_1) = lock(W(i_1)) \rightarrow lock(W(i_2)),$$

$$lock(i_2) = lock\big(W(i_2)\big) \rightarrow lock\big(W(i_1)\big).$$

Pro implementaci zamykání synchronizačních skupin je tedy potřeba zajistit, aby při vzniku stavu uváznutí byl tento konflikt vyřešen. Nejjednodušším řešením je časově omezit dobu čekání na alokaci synchronizačních skupin rozhraní.

2.8. Podmíněná alokace

Kromě statické a dynamické synchronizační skupiny definujeme ještě synchronizační skupinu subjektu p, která může být svázána s rozhraním subjektu. Jedná se o synchronizační skupinu svázanou přímo s daným subjektem s a píšeme $p(i(s))$. Pokud subjekt s obsahuje synchronizační skupinu, která je svázána s rozhraním $i(s)$, pak při požadavku na rozhraní $i(s)$ dojde k zamčení statické, dynamických synchronizačních skupin a synchronizační skupiny subjektu, tj.

$$lock(i(s)) = lock(W(i) \cup D(i(s)) \cup p(i(s))). \qquad (20)$$

Prostřednictvím p lze realizovat podmíněnou alokaci subjektu, neboť dostupnost p je plně v režii subjektu a tudíž může být odvozena od stavu subjektu. Tak lze svázat alokaci p s určitou podmínkou.

Kromě synchronizační skupiny subjektu, definujeme u subjektu s výchozí návratovou hodnotu pro každou vlastnost, či metodu rozhraní i prostředku r. Tato hodnota je vrácena v případě, že proces alokace rozhraní nebude úspěšně proveden. Definujeme funkci $def(m_l(i(s)))$, která vrací výchozí návratovou hodnotu metody $m_l(i(s))$. Tato funkce se uplatní zejména při podmíněné alokaci, neboť zde bude docházet k situaci, kdy požadavek na alokaci nebude uspokojen, a tudíž bude vrácena výchozí hodnota.

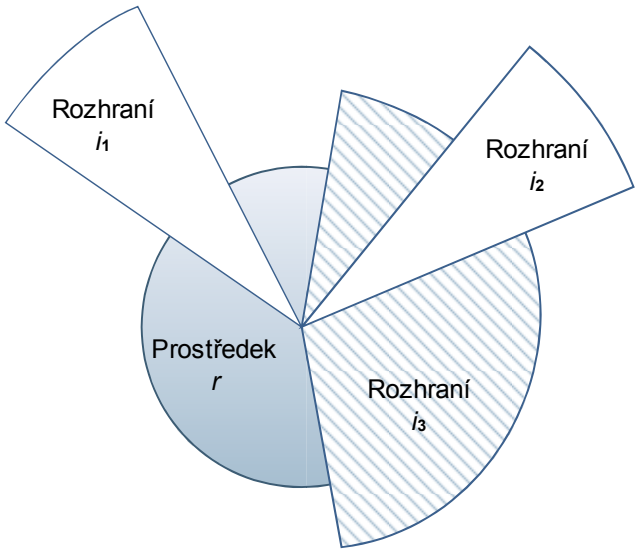
3. Aplikace navrhovaného modelu

Navrhovaný model pro popis prostředků sebou přináší určitá omezení a vlastnosti, které je potřeba mít v úvahu při návrhu systému. V této kapitole nejprve zavedeme vlastní grafickou reprezentaci navrhovaného modelu a pak s pomocí této grafické reprezentace provedeme návrh vybraných praktických situací.

Grafická interpretace modelu byla vytvořena jako nástroj pro názornější a intuitivnější popis systému využívající navrhovaného modelu. Cílem bylo dát modelu obdobnou možnost grafického vyjádření, jako je např. jazyk UML v rámci OOP programování.

3.1. Grafická reprezentace objektů modelu

Prostředek si lze představit jako element mající několik rozhraní, kdy vzájemné funkčnosti rozhraní se mohou překrývat (viz obr. 2). Subjekt je pak konkrétním výskytem prostředku.



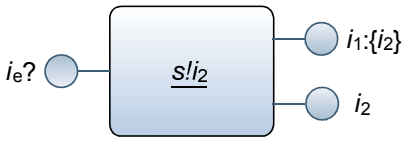
Obr. č. 2: Prostředek mající definované tři rozhraní

Prostředek r implementující rozhraní i_1 budeme zobrazovat podle obr. 3. Na tomtéž obrázku je znázorněna stejná situace, avšak pro subjekt s. Závislé rozhraní i_1 na rozhraní i_2 je zobrazeno na obr. 4. Současně je zde zobrazeno událostní rozhraní i_e a indikace, že subjekt implementuje podmíněnou alokaci pro rozhraní i_2

elektrorevue
ISSN 1213 - 1539

včetně definování podmínky pro splnění dostupnosti synchronizační skupiny subjektu.

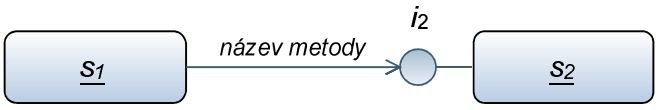
Obr. č. 3: Grafická interpretace prostředku *r* a subjektu mající definované rozhraní *i*



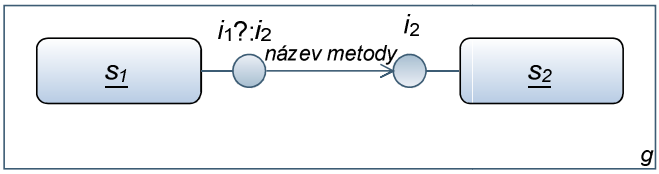
s!i_2: subjekt *s* je va stavu *ready*.

Obr. č. 4: Grafická interpretace závislosti rozhraní, událostního rozhraní a podmíněné alokace

Případ, kdy subjekt s_1 volá metodu, nebo vlastnost rozhraní i_2 subjektu s_2, je zobrazen na obr. 5. Vyvolání události je naznačeno na obr. 6, kde subjekt s_1 vyhazuje událost i_1, na kterou je napojeno rozhraní i_2 subjektu s_2. Na obr. 6 je současně zobrazena hranice regionu *g*. Název volané metody nebo vlastnosti rozhraní je uveden jako text u orientované čáry znázorňující volání. V případě, že název metody/vlastnosti není důležitý, neuvádí se.



Obr. č. 5: Grafická interpretace volání metody nebo vlastnosti rozhraní subjektu



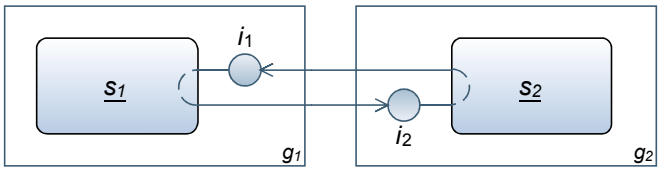
Obr. č. 6: Grafická interpretace vyvolání událostního rozhraní

3.2. PŘÍKLADY VYUŽITÍ MODELU

S využitím grafické reprezentace modelu jsou v této kapitole uvedeny případy ukazující použití navrhovaného modelu na praktických situacích.

3.2.1. STAV UVÁZNUTÍ

Při komunikaci mezi regiony může dojít ke stavu uváznutí. K tomu může dojít, pokud dva objekty chtějí sebou vzájemně komunikovat, přičemž každý z nich se nachází v jiném regionu, ale vzájemně využívají svoje rozhraní. Tento případ zjednodušeně demonstruje obr. 7.



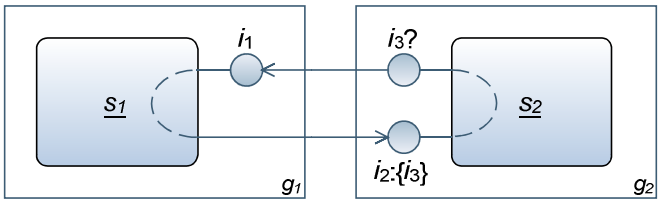
Obr. č. 7: Stav uváznutí mezi subjekty v různých regionech

Pro situaci z obr. 7 je posloupnost alokace rozhraní následující:

$$lock(i_1) = lock(W(i_1)) \rightarrow lock(W(i_2)), \quad (21)$$

$$lock(i_2) = lock(W(i_2)) \rightarrow lock(W(i_1)). \quad (22)$$

Aby nedošlo k uváznutí mezi subjekty různých regionů, je vhodné pro komunikaci mezi regiony využívat událostního rozhraní, tak jak je schematicky zobrazeno na obr. 8.



Obr. č. 8: Komunikace mezi rozhraními s využitím událostního rozhraní

Posloupnost alokací rozhraní je pak tato:

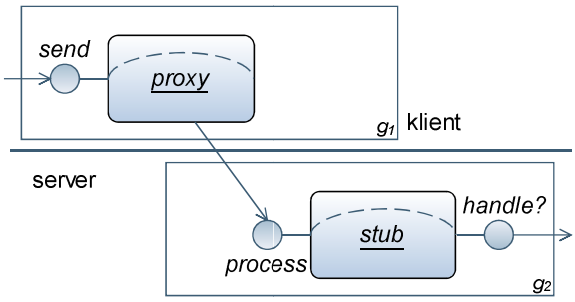
$$lock(i_1) = lock(W(i_1)) \rightarrow lock(i_2), \quad (23)$$

$$lock(i_2) = lock\big(W(i_2) \cup D(i_2)\big) =$$

$$lock\big(W(i_2) \cup W(i_1)\big). \quad (24)$$

3.2.2. KLIENT-SERVER SYNCHRONNÍ KOMUNIKACE

Tato komunikace představuje zasílání požadavku klienta na server, kde jen tento požadavek zpracován a výsledek zpracování je vrácen zpět na klienta ve formě návratové hodnoty volané metody. Princip této komunikace je zobrazen na obr. 9, kde klient obsahuje rozhraní *send* pro zaslání požadavku na server. Proxy objekt klienta volá serverové rozhraní *process*, které volá událost *handle*, na kterou je připojen kód pro zpracování požadavku.



Obr. č. 9: Komunikace klient-server

Posloupnost alokací klient-server komunikace je:

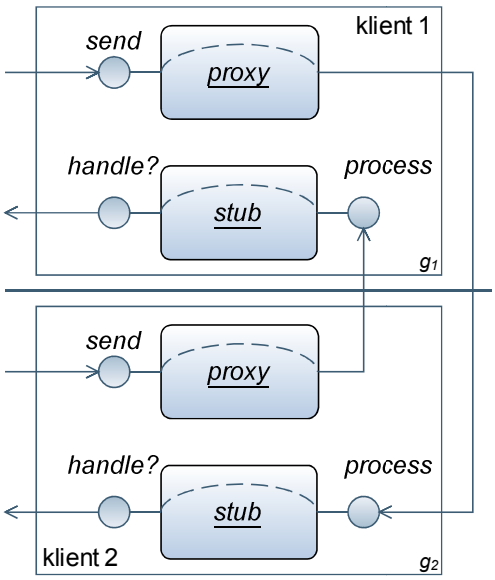
$$lock(send) = lock\big(W(send)\big)$$

$$\rightarrow lock(W(process) \cup D(process)). \quad (25)$$

elektrorevue
ISSN 1213 - 1539

3.2.3. Peer-to-peer synchronní komunikace

Jedná se v podstatě o obousměrnou komunikaci typu klient server z předcházející kapitoly (viz obr. 10).



Obr. č. 10: Peer-to-peer komunikace

Posloupnost alokací je shodná jako u klient-server komunikace, kdy každý klient má svou vlastní alokační posloupnost pro své *send* rozhraní.

3.2.4. Komunikační vrstva s čekáním na potvrzení

Příklad popsaný v této kapitole využívá podmíněné alokace. Jedná se o komunikační vrstvu poskytující rozhraní pro zápis dat (*read*) a událost pro zpracování přečtených dat (*handleRead*) (viz obr. 11).

Zápis přes komunikační vrstvu probíhá ve čtyřech krocích, kdy po volání rozhraní *write* (krok 1) následuje vymazání příznaku potvrzení *ack* (tj. nastavení na *false*) v subjektu *SYNC* (krok 2). Pak se provede zápis dat do spodní vrstvy (krok 3) a volá se rozhraní *wait*, jehož alokace je podmíněna hodnotou příznaku *ack*, které musí mít hodnotu *true*.

Čtení dat je realizováno přes rozhraní *onRead* (krok I), kdy hned dalším krokem je volání rozhraní *set* subjektu *SYNC* (krok II), což způsobí nastavení příznaku *ack* na hodnotu *true*, což značí, že zapsaná zpráva byla potvrzena. To má za následek, že je alokováno rozhraní *wait*, které je provedeno a tím je proveden i poslední krok zápisu. Posledním krokem čtení dat, je opublikování přečtených dat přes událost *handleRead* (krok III).

Pokud nepřijde do určitého časového okamžiku potvrzení na zprávu, je při alokaci rozhraní *watt* vrácena výchozí hodnota, kterou definujeme jako *false*. Z toho nám vyplývá, že návratovou hodnotou provedeného rozhraní *wait*, které bylo úspěšně alokovaného, musí být hodnota *true*, aby na straně zápisu mohlo být rozlišeno, zda zpráva byla potvrzena či nikoli.

4. Implementace modelu

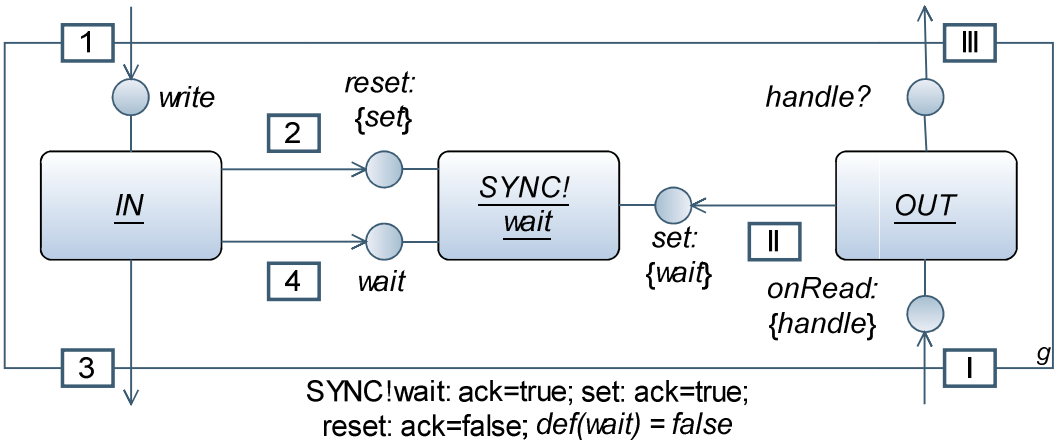
Navrhovaný model byl implementován na platformě .NET Framework. To je také důvodem, proč je model postaven na OOP principech, neboť platforma .NET je striktně objektově orientovaná. Implementace obsahuje cca 50 tříd a využívá aspektově orientovaného přístupu (AOP), neboť základním požadavek implementace modelu bylo, aby jeho použití bylo co nejvíce transparentní. S využitím AOP bylo docíleno, že po prvotní definici prostředků a vzájemných vazeb, se s prostředky resp. s rozhraními prostředků pracuje obdobně jako se standardním objektem platformy .NET, kdy alokační mechanismus je plně skryt.

Na časovém srovnání obdobných implementací postavených na navrhovaném modelu a s využitím standardních synchronizačních přístupů (tj. klíčových slov realizující vzájemné vyloučení) bylo zjištěno, že použití navrhovaného modelu v rámci lokálního přístupu ke sdíleným prostředkům vnáší určité zpoždění (1,1 až 1,6 krát větší při použití událostních rozhraní) způsobené logikou pro alokaci synchronizačních skupin. K razantnímu zpoždění provádění požadavku dojde, pokud dochází k alokaci synchronizačních skupin, které přesahují hranice procesu, kdy vlivem meziprocesové komunikace dochází k několikanásobnému zpoždění. Zde je však potřeba zmínit, že při použití standardních přístupů nebylo implementováno zamykání prostředků za hranice procesů. Zde by byla zapotřebí implementace, která by efektivně využívala použitou metodu meziprocesorové komunikace, což v testech nebylo učiněno, neboť šlo jen o ukázku, že funkčnost implementace modelu není omezena jen na prostředky nacházející se v rámci jednoho procesu.

Implementovaný model obsahuje základní logiku pro zamezení stavu uváznutí, tak jak je definována v modelu, avšak při nesprávném návrhu vazeb a z toho plynoucí využívání prostředků může dojít ke stavu uváznutí (viz kap. 3.2.1). Zde je však chování odlišné, neboť díky tomu, že alokace není prováděna „nízkoúrovňovým“ přístupem, ale prostřednictvím alokačního mechanismu, neznamená stav uváznutí konec činnosti aplikace, ale dojde v rámci aplikace ke vzniku chybového stavu a v případě ošetření této chyby může aplikace pokračovat ve své činnosti.

Vzhledem k tomu, že implementovaný model zatím obsahuje jen základní algoritmus pro alokaci prostředků je v současné době jeho použití víceméně omezeno na alokaci prostředků v rámci jednoho procesu a poskytuje tak alternativní nástroj minimalizující vznik stavu uváznutí místo použití standardních synchronizačních technik založených na vzájemném vyloučení. Pro sofistikovanější algoritmy bude potřeba vhodným způsobem implementovat některý z algoritmů detekce stavu uváznutí v distribuovaných systémech a rozšířit tak navrhovaný model o model komunikační [8].

elektrorevue
ISSN 1213 - 1539



Obr. č. 11: Komunikační vrstva s čekáním na potvrzení

Výhodou implementovaného modelu je, že je postaven nad celou platformou .NET a není svázán s vnitřní strukturou platformy, a tak i pro model platí, že je platformově nezávislý.

5. Závěr

Navrhovaný model poskytuje formální popis prostředků a jejich vzájemných vazeb, kdy model vychází z principů objektově orientovaného přístupu (OOP). Součástí modelu je i návrh jednoduchého alokačního mechanismu prostředků, který při korektním definování prostředků a vazeb zabrání vzniku stavu uváznutí. Pro snadnější aplikace modelu byla vytvořena i grafická reprezentace modelu, prostřednictvím které bylo ukázáno, jak lze řešit některé praktické případy navrhovaným modelem. Tím, že model vychází z principů OOP, lze jej realizovat některým z moderních OOP programovacích jazyků. Důkazem je i vlastní realizace popisovaného modelu na platformě .NET Framework.

Literatura

[1] Barnaby T., Distributed .NET Programming in C#, APress, 2002, 494 stran, ISBN 1590590392.

[2] Sadén B., Coping with Java Threads, IEEE Computer, Volume 37, Number 4, April 2004, s. 20-27, ISSN 018-9162.

[3] Coffman E.G., Elphick M.J, Shoshani A., System Deadlocks, ACM Computing Surveys, ASSOC COMPUTING MACHINERY, Vol. 3, No.2, June 1971, s. 67-78, ISSN 0360-0300

[4] Fraser K., Harris T., Concurrent programming without locks, ACM Transactions on Computer Systems, Volume 25 , Issue 2, May 2007, s. 1-59, ISSN 0734-2071

[5] Fober D., Orlarey Y., Letz S., Lock-Free Techniques for Concurrent Access to Shared Objects [online]. 2003 [cit. 2009-05-24]. Dostupný z WWW: http://www.grame.fr/pub/fober-JIM2002.pdf

[6] Lee A. E., The Problem with Threads, IEEE Computer, May 2006, p.33-42, ISSN 018-9162.

[7] Smrčka, A., Úvod do formální verifikace [online], 2007 ,[cit. 2009-05-06]. Dostupný z WWW: http://www.fit.vutbr.cz/~smrcka/fav/guide

[8] Chandy K. M., Misra J., Haas L. M., Distributed Deadlock Detection, ACM Transactions on Computer Systems, Vol. 1, No. 2, May 1983, s. 144-156,