Totální objektifikací za nový řád ve správě objektů

Ivan Ryant

1 Persistence objektů *(Pod heslem: dejte objektům šanci prožít plný život!)*

1.1 Rozporuplný život objektů v úlohách

1.1.1 Neuspokojivé možnosti životního cyklu objektů

Turbo Vision pro Turbo Pascal charakterizuje životní cyklus objektu trojicí metod „Init – Run – Done". Takový nebo podobný průběh životního cyklu platí obecně. Za prvé: někdo vytvoří objekt. Za druhé: objekt existuje, poskytuje ostatním objektům své služby, nějak se chová. Za třetí: Když objekt zjistí, že jeho další existence ztratila smysl, sám sebe zruší. Otázka je, KDY existence objektu ztratí smysl?

Nástroje pro objektově orientované programování zpravidla omezují život objektů na dobu, po kterou běží úloha (nebo také program, aplikace, proces apod.) Konce úlohy mohou přetrvat jen atributy těch objektů, které si uložíme na disk do tzv. streamu. Co je to stream a proč nás neuspokojí?

Stream vypadá navenek jako obyčejný soubor. Vnitřně však není uspořádán jako posloupnost záznamů jednoho typu (jak bývá zvykem u souborů). Stream je vnitřně uspořádán jako posloupnost datových struktur různých typů. Každá datová struktura ve streamu uchovává konkrétní hodnoty atributů nějakého objektu.

Chyba je v tom, že stream uchovává jenom atributy. Objekt však obsahuje nejen atributy, ale také metody. Atributy zachycují stav objektu v každém okamžiku asi tak, jako políčka na filmu zachycují jednotlivé fáze děje. Metody mohou měnit stav objektu asi tak, jako promítačka posouvá film z jednoho políčka na druhé. Metody objektu umožňují, aby se nějak choval, aby mohl poskytovat své služby ostatním objektům. Teprve metody dávají objektu život. Ve streamech můžeme zachytit jen atributy, tedy stav objektů. Žádný program, který nemá k dispozici příslušné metody, nedokáže oživit objekty uložené ve streamu.

Shrňme nedostatky streamů: Stream neuchovává objekty, pouze zachycuje jejich stav. Objekt neuchovává svůj stav ve streamu sám od sebe, nýbrž ho tam někdo musí výslovně uložit. Stream sám o sobě nestačí k tomu, aby jedna úloha mohla předávat své objekty jiné úloze nebo aby objekty putovaly z počítače na počítač.

Shrňme závislost životního cyklu objektů na životě úlohy: Objekty vznikají až po spuštění úlohy a se skončením úlohy musí zase všechny zaniknout. Objekty vznikají s každým spuštěním úlohy vždy znovu a se skončením úlohy mizí v temnotě věčného zapomnění.

1.1.2 Proč jsou tyto možnosti neuspokojivé?

Vezměme si např. úlohu, která má určitou soustavu nabídek (menu) a určitá dialogová okna. Tyto objekty si musí úloha vytvořit při každém spuštění vždy znovu – a pokaždé stejné objekty. Částečné řešení nabízejí tzv. resourcy (což je zvláštní případ streamů). Jedna zvláštní úloha napřed jednou vytvoří všechny potřebné objekty a jejich stav uloží do resourcu. Druhá úloha pak nemusí tyto objekty vždy znovu pracně nastavovat, jenom nahraje jejich počáteční stav z resourcu. Tento příklad dokládá, že často potřebujeme, aby jedna úloha objekty vytvořila a jiná je pak používala. To je v rozporu se závislostí životního cyklu objektů na době běhu úlohy. Řešení pomocí resourců není zcela obecné a ve své podstatě je polovičaté.

Další příklad: v objektově orientovaném informačním systému banky bude můj účet modelován instancí objektu „Běžný_účet". V šest večer pan bankéř stáhne roletu, vypne počítač a odebere se třeba do hospody. Je snad vypnutí počítače (a ukončení úlohy) důvodem k zániku mého běžného účtu?

Poslední příklad do třetice: představme si, že součástí řídicího systému automatické pračky je instance objektu „Vodoznak". Když zastavím pračku uprostřed praní (tj. ukončím úlohu), měl by objekt „Vodoznak" zaniknout. Znamená to snad, že zanikne i voda v pračce nebo že nemá smysl detekovat její hladinu?

1.1.3 Narážíme na rozpor

mezi životním cyklem objektu a životním cyklem úlohy. Podstatou objektově orientovaného přístupu je modelování abstraktních pojmů a konkrétních předmětů z reálného světa. Podle našich představ reálný svět existuje objektivně, tj. jeho existence nekončí s dokončením nějaké úlohy, s dobběhnutím procesu. Obecně tedy není správné, aby životní cyklus každého objektu závisel na období aktivního života úlohy. Životní cyklus některých objektů sice může být spjat s úlohou, životní cyklus jiných objektů však může být spjat s něčím jiným.

1.2 Analogické problémy a jejich klasická řešení

Už staří Římané... praví okřídlené úsloví. To vážně nevím, jestli už staří Římané bažili po persistenci objektů. Jisté je, že „persistó" je slovo latinské a že znamená „setrvávat" nebo „vytrvávat". Persistentní je tedy takový objekt, který zarputile setrvává v počítači, dokud má jeho existence smysl (a možná i déle). Další jistá věc je, že problém persistence nepřišel až s objektovou orientací, nýbrž je tady už dlouho a je celkem úspěšně vyřešen.

První příklad: Moje deset let stará kalkulačka si pamatuje každý řádek programu a každou hodnotu od okamžiku, kdy je vytvořím, až do okamžiku, kdy je přepíšu nebo vymažu.

Druhý příklad: Kdykoli zapneme tiskárnu EPSON FX 850, tiskárna se tváří, jakoby v ní byl papír nastavený na začátek stránky. To ale nebývá pravda a tiskárna pak jednak tiskne přes hranice stránek a na druhou stranu sice také občas stránkuje, ale úplně nesmyslně. Je to tím, že když tiskárnu vypneme, tak se (pochopitelně) papír v tiskárně nenastaví na začátek stránky, ale tiskárna se chová jako kdyby se nastavil. Nepamatuje si totiž polohu papíru po dobu, kdy je vypnutá. Naproti tomu tiskárna EPSON LQ 1050 si polohu papíru pamatuje i v době, kdy je vypnutá, a potíže se stránkováním tam sice jsou taky, ale úplně jiné.

Jako třetí příklad nám mohou posloužit monitorovací stanice, které sledují znečištění ovzduší. Tyto stanice mají měřit nepřetržitě čtyřiadvacet hodin denně a sedm dnů v týdnu. Pokud jsou napájeny ze sítě, může občas dojít k výpadku proudu. Měření se tím přeruší, měřené procesy však přirozeně pokračují. Podle toho se měřicí počítač musí s výpadkem nějak rozumně vypořádat. Nerozumné by bylo, kdyby po každém výpadku vyžadoval, aby mu lidská obsluha nastavila nějaké počáteční hodnoty (a navíc třeba pokaždé stejné). Sledování a řízení procesů reálného světa bývá naprogramováno jako nekonečný proces, který může být nanejvýš přerušen výpadkem napájení. Pro tyto úlohy se používají speciální jednoúčelové (příp. vestavné) počítače, které umožní speciální úloze její kýžený životní cyklus.

Ostatně nejde o nic nového: před patnácti lety jsem překládal assemblerem na JPR–12. JPR byla řídicí počítač, měla feritovou paměť a uměla ošetřit výpadky napájení. Když jsem po týdnu přišel a zapnul JPR do sítě, rozběhl se assembler přesně tam, kde jsem ho před týdnem přerušil tím nejsurovějším způsobem – vypnutím počítače ze zásuvky.

Na rozdíl od jednoúčelových řídicích počítačů jsou možnosti životního cyklu objektů a úloh v univerzálním počítači ovlivněny typickými rysy práce uživatele (nebo několika uživatelů) s univerzálním počítačem: počítač je určen k řešení mnoha rozlodivných úloh, úlohy se spouštějí buď postupně jedna za druhou nebo běží paralelně (ale v omezeném počtu). Navíc někteří uživatelé jsou zvyklí své počítače vypínat na dobu své nepřítomnosti, takže o nekonečných úlohách nemůže být řeč. Ale i tady mívá uživatel pocit, že práci na úloze vlastně jen přerušuje. Chce pokračovat tam, kde naposledy skončil.

To umožňují například konfigurační soubory. Známým příkladem mohou být vývojová prostředí firmy Borland, kde se po spuštění ocitneme v tom zdrojovém souboru a na tom místě, kde jsme posledně přerušili práci. Nejenže se nám zde nabízí jedno z možných řešení persistence, ale nacházíme tu i neuvědomělý objektový přístup. Můžeme říci, že třída objektů představuje to, co mají všechny instance společné (např. metody). Třída je

reprezentována nainstalovaným programem (tj. obsahem adresáře, kde se nacházejí soubory *.EXE, knihovny, nápověda a všechno, co je neměnné a společné všem rozpracovaným konkrétním úlohám). Všimněme si, že program je nainstalován jen jednou. Naproti tomu každá úloha má svůj adresář a v něm datové soubory a konfigurační soubor – tedy všechno, co je specifické, co zachycuje okamžitý stav úlohy, co vytváří konkrétní instanci objektu.

V univerzálním počítači se úlohy a uživatelé (přes veškerá opatření) navzájem ovlivňují. Úlohy se mohou navzájem hroutit nebo si navzájem přepisovat data v paměti nebo v souborech. Persistence vydává naše úlohy a naše data tomuto nebezpečí všanc. Ochranu veškerých prostředků počítače zajišťuje operační systém, k ochraně dat můžeme použít databáze.

Operační systém chrání soukromé prostředky úlohy proti neoprávněnému přístupu a zajišťuje korektní sdílení sdílených prostředků. Operační systém musí jakékoli nekorektní chování v zásadě vyloučit, protože běží v jednosměrném (někdy dokonce v reálném) čase, který se nedá vrátit zpět.

Ochrana dat v databázích bývá daleko rafinovanější a zahrnuje i možnost vracet se z nekonzistentního stavu do některého předchozího stavu, kdy byla data ještě konzistentní. Databáze se zpravidla dokáže vyrovnat i s havárií počítače. To je spíš nutnost daná realitou než nějaká přednost: databáze jsou často distribuované v sítích, kde pravděpodobnost havárie roste geometrickou řadou s počtem uzlů.

1.3 Jak z toho ven

(Máme i problémy, ale už víme jak 100ho ven, dost možná i několik set... Kdo ví?)

Konečně dospíváme k těmto požadavkům:

a) Životní cyklus objektu nesmí záviset na době života úlohy.

b) Životní cyklus úlohy bychom měli chápat nově. Úloha nekončí tím, že se vrátíme do operačního systému. Tím se jenom přeruší, aby příště mohla pokračovat. Skončí teprve, až ji o to uživatel výslovně požádá. V tom případě by však měla oznámit svůj zánik všem persistentním objektům, které zpracovávala. Ty z nich, které s koncem úlohy ztratí smysl své existence, se zruší také.

c) Výpadek napájení nebo jiný nekorektní zásah do běhu úlohy pouze přeruší její životní cyklus. Úloha se musí umět zotavit.

d) Persistentní objekty musí být odolné (nebo chráněné) proti nekorektním zásahům.

2 Správa objektů

2.1 Už teď vládne v objektech chaos...

Dokud nemáme persistentní objekty, stačí zvládnout dvě věci: knihovnu tříd (jako je např. Turbo Vision nebo Object Windows) a svůj vlastní systém objektů dočasně vznikající a zanikající na dobu jednoho spuštění úlohy. Ale ani to není snadné, tím méně s vývojovými prostředky, které máme k dispozici.

Například: konečně jsem v dokumentaci k Turbo Vision našel popis metody draw, která tak nejspíš vypadala, že bych ji mohl použít pro svůj objekt odvozený od typu TListViewer; teprve po pracném prohledání rodokmenu asi tak do pátého kolena zjišťuji, že jsem se zmýlil, že metodu draw bych rozhodně použít neměl.

Kromě toho: knihovna tříd je v podstatě jakási databáze modulů. Ale program, který ji spravuje nám nabízí asi takové služby, jako LIBRARIAN na Tesle 200 z šedesátých let. Moduly s implementací tříd, definiční soubory v C++ (header files) a uživatelská dokumentace (nápověda) jsou úplně samostané a o jejich konzistenci se nikdo nestará. Přitom by to zvládla i jednoduchá moderní databáze!

A když už konečně splodíme nějaký program, objeví se nové problémy (které ale jen předznamenávají, co nás čeká s persistentními objekty). Instance objektů vytváříme zpravidla dynamicky ve volné paměti, takže objekty vytvořené nezávisle na sobě o sobě navzájem nevědí. Když chtějí dva takové objekty navázat spojení, musí jeden z nich poslat broadcast skrz uživatelské rozhraní – snad se dočká odpovědi.

Přitom databáze dnes dokáží zajistit jak složitou organizaci dat, tak i snadný přístup k nim. A na druhé straně ochranu proti neoprávněným zásahům.

2.2 A což teprve když budou persistentní?

Chceme-li vytvořit několik programů v C++, které budou jeden druhému předávat objekty, musíme použít několik souborů na disku a zajistit jejich konzistentnost. Jde o tyto soubory:

a) stream: obsahuje okamžitý stav objektů, neobsahuje metody

b) modul: obsahuje implementaci, musí se přisestavit ke každému programu

c) header: deklaruje třídu, musí se použít při překladu každého programu

Klasický nástroj, který v klasickém operačním systému spravuje persistentní data, je systém souborů. Systém souborů není opět nic jiného než nemoderní, nebezpečná, strašně špatně udělaná databáze. Má však některé vlastnosti, které současné databáze nemají,

např.: do systému souborů můžeme uložit spustitelný soubor (*.EXE) a co víc, můžeme ho spustit. Kromě toho, systém souborů má prostředky pro snadné vytváření, správu a prohledávání stromových struktur.

2.3 Na půli cesty k řešení: objektově orientované databáze

Už dost dlouho tady nenápadně vytvářím dojem, že zázračným řešením problémů s persistentními (a nejen persistentními) objekty (a nejen objekty) jsou databáze. Asi bych měl předeslat, že je to dojem mylný. Především málokterá databáze dokáže uchovávat objekty i s metodami. Tento požadavek by měly splňovat databáze objektově orientované (bohužel málokterá databáze, která o sobě tvrdí, že je objektově orientovaná, dokáže uchovávat objekty i s metodami).

Výhody databází jsou zřejmé:

a) životní cyklus objektu nemusí záviset na životním cyklu úlohy

b) možnost složité organizace objektů

c) zajištění pohodlného přístupu k jednotlivým instancím

d) možnost šíření objektů v sítích a po telekomunikačních linkách

e) ochrana objektů proti neoprávněným zásahům

f) zajištění objektů proti nekorektnímu zacházení (výpadek napájení)

g) zotavení objektů po havárii

Databáze neřeší všechno:

a) práci objektů v reálném čase

b) problémy spojené s programováním (k tomu slouží OO systémy CASE)

c) rozesílání objektů do jednotlivých počítačů v soustavě jednoúčelových řídicích počítačů

3 OOOS – totálně objektifikované prostředí

Klasický operační systém můžeme chápat jako virtuální počítač, který odclňuje programátora i uživatele od hardwaru, poskytuje jim služby na jejich úrovni abstrakce a komunikuje s nimi v těch pojmech, ve kterých oni uvažují (soubory místo vstupních a výstupních zařízení, virtuální paměť místo fyzické RAM apod.) S rozvojem objektově

orientované technologie by měly vzniknout také objektově orientované operační systémy, které poskytnou programátorům a uživatelům objektově orientovaný virtuální počítač.

Někdo možná namítne, že operační systém, který spravuje operační paměť, nemůže zajistit objektům persistenci (při výpadku paměť všechno zapomene) a naopak databáze nemůže zajistit dost rychlý přístup k objektům, které uchovává na discích. Ve skutečnosti ani jedno není zásadní problém. Virtuální paměť nám už dnes nabízí gigabajty operační paměti a i když se tak zatím netváří, ve skutečnosti vlastně vůbec nezapomíná, protože její obraz je uložen na disku. V podstatě neexistuje důvod, aby databáze nebyly ukládány do operační paměti. Ale ani klasické umístění databáze výslovně na disk nepůsobí zásadní problémy: rychlost disku výrazně vylepší keš. Virtuální paměť a disková keš nám umožňují už úplně abstrahovat od rozlišování, co je RAM, a co je soubor na disku. Úkolem objektově orientovaného operačního systému bude, aby nás odclonil od pojmů jako soubor nebo paměť a komunikoval s námi pomocí jediného univerzálního pojmu OBJEKT.

V objektovém pojetí je všechno objekt. Jde jen o to, aby se každý objekt dal začlenit do složitě organizovaného systému, aby byl snadno přístupný, aby byl odolný, dobře zabezpečený a persistentní. Objekty by měly putovat sítěmi z počítače na počítač, překonávat hranice hardwarových platforem. Některé by měly být použitelné pro nasazení v reálném čase, jiné by měly být schopné obnovit svůj stav takový, jaký byl v určitém okamžiku někdy v minulosti (např. před zhroucením systému, před napadením virem, před revolucí apod.). Součástí objektu by měly být nejen aktuální hodnoty jeho atributů a sada metod, ale také např. aktuální návod k použití.

Tak tedy objektově orientovaný operační systém... Že by zázračný všelék? Samozřejmě: každá totalita přece začíná lákavými sliby.

Autor: Ivan Ryant
Husinecká 2
130 00 Praha 3