

Beztržidní programování v prostředí Squeak Smalltalk

Pavel KŘIVÁNEK

FIT VUT Brno

Božetěchova 2, 612 66 Brno
xkrivall@stud.fit.vutbr.cz

Vedoucí práce: Ing. Vladimír Janoušek, Ph.D.

Abstrakt. Prezentovaný článek popisuje základní principy beztržidní objektové orientace ve formě, v jaké byla realizována v programovacím jazyce Self. Ukazuje, proč je pro řešení praktických problémů výhodnější než běžně používaná hierarchie tříd a proč tento moderní objektový koncept může výrazně změnit vývojové prostředí budoucnosti. Dále představuje dialekt jazyka Self, který byl navržen pro adaptaci myšlenek beztržidního programování do otevřeného smalltalkovského vývojového a aplikačního prostředí Squeak. Popisuje implementaci překladače tohoto nového jazyka využívající speciálně upravený virtuální stroj, který zaručuje velmi efektivní provádění výsledných programů. V závěru jsou diskutovány jeho vlastnosti a náměty na další vývoj.

Klíčová slova: objektově orientované jazyky, Smalltalk, Self, Marvin, beztržidní programování, prototypy, delegace, bytekód, virtuální stroj

1 Úvod

Tato práce se zabývá problematikou beztržidní objektové orientace a její adaptace na otevřené vývojové prostředí Squeak Smalltalk. Nejdříve jsou na vzoru programovacího jazyka Self ukázány její základní principy a vlastnosti včetně přínosu oproti konvenční struktuře programů založené na použití tříd. Dále je prezentován nově navržený programovací jazyk, který vznikl jako dialekt Selfu určený pro práci nad virtuálním strojem Squeaku, a konkrétní implementace překladače tohoto jazyka. Na závěr jsou popsány realizované úpravy virtuálního stroje umožňující hlubokou a efektivní integraci výsledných programů se stávajícím smalltalkovským prostředím.

Kromě druhé kapitoly zabývající se koncepty programovacího jazyka Self je veškerý obsah tohoto článku výsledkem mé vlastní výzkumné činnosti založené na podnětných odborných konzultacích mého školitele a vychází z mé diplomové práce [1].

2 Programovací jazyk Self

Self je objektově orientovaný programovací jazyk s úzkou vazbou na své vývojové prostředí [2]. Svoji syntaxí i sémantikou má velmi blízko k jazyku Smalltalk-80, ze kterého přímo vychází. Byl navržen v roce 1986 Davidem Ungarem a Randallem Smithem na Stanfordské univerzitě. Na začátku devadesátých let dvacátého století jeho vývoj převzala společnost Sun Microsystems Labs [3].

2.1 Objekty

Jazyk Self je čistě objektově orientovaný, což znamená, že vše včetně čísel či metod je v něm reprezentováno jako objekt. Objekty jsou samostatné uzavřené entity, které definují svůj stav a chování. Jsou tvořeny sloty, což jsou pojmenované reference na jiné objekty.

Slotů je několik typů. Prvním jsou datové sloty, které nesou reference na libovolné datové objekty. Dalším druhem slotů jsou sloty s referencemi na metody. Metody jsou objekty, které mají asociovaný vykonatelný kód. Ten se aktivuje v okamžiku, kdy je objektu zaslána zpráva, jejíž selektor se shoduje se jménem referencujícího slotu. Velmi důležité jsou tzv. rodičovské sloty. Ty se formálně shodují s datovými, ale na rozdíl od nich hrají významnou roli při procesu označovaného jako delegace.

2.2 Delegace

Stejně jako ve Smalltalku je jediná přípustná forma komunikace mezi objekty zaslání zprávy. Na ni objekt reaguje tak, že vyhledá metodu odpovídající parametrům zprávy a kód této metody vykoná v kontextu příjemce.

Self se ale od Smalltalku diametrálně liší v tom, jak metody vyhledává. Smalltalk a jiné běžné objektově orientované jazyky vyhledávají tyto metody ve třídách, k nimž příjemce zprávy náleží. Pokud není žádná vhodná metoda nalezena přímo ve třídě příjemce, pokračuje se ve vyhledávání v abstraktnější nadřazené třídě. Třídy slouží především jako struktury, které definují společné chování pro množinu objektů stejného typu.

Přestože je tento mechanismus velice častý, není jediný možný. Objektové paradigma v principu nevyžaduje, aby objekty svoje chování sdílely, tím méně aby se používaly třídy objektů. Stačí, pokud vytvoříme mechanismus, který umožňuje polymorfismus objektů, tedy pokud máme možnost s různými podobnými konkrétními objekty pracovat stejně. Sdílení chování je však velmi přínosné pro snížení paměťových požadavků výsledných programů a jejich lepší udržitelnost.

Self pro sdílení chování mezi více podobnými objekty používá mechanismus nazvaný delegace. Pokud je objektu zaslána zpráva, pokusí se mezi svými sloty najít ten, jehož selektor odpovídá obdržené zprávě. Pokud takový slot najde, provede příslušnou referencovanou metodu nebo přečte resp. zapíše hodnotu do datového slotu. Pokud však žádný vhodný slot nenalezne, deleguje zprávu na objekty, které jsou referencovány jeho rodičovskými sloty. To znamená, že v takto odkazovaném rodičovském objektu pokračuje rekurzivně vyhledávání dále. V okamžiku, kdy je v některém z rodičovských objektů slot nalezen, je kód příslušné metody vykonán v kontextu příjemce zprávy, tedy v objektu, v němž vyhledávání začalo, nikoliv

v objektu, v němž je umístěn nalezený slot. To je hlavní rozdíl mezi delegací a běžným přeposláním zprávy.

Self při delegaci kontroluje případné smyčky tak, aby žádný delegovaný objekt nebyl prohledáván vícekrát. Rovněž kontroluje vznik případných nejednoznačností, které hlásí pomocí vyvolání výjimky. Výjimka je také vygenerována v případě, že žádný vhodný slot nebyl v objektu ani v žádném z jeho rodičů nalezen.

Tento princip je velice jednoduchý, přesto však dokáže plně nahradit třídní hierarchii v míře, jaká je běžná u konvenčních objektově orientovaných jazyků. Třída je zde nahrazena dvojicí objektů. První se označuje jako rys (trait) a definuje sdílené chování, příslušné metody. Druhým nezbytným objektem je tzv. prototyp, což je vzorová instance simulované třídy. Ta deleguje zprávy na rys (referencuje jej rodičovským slotem) a definuje datové sloty. Operace vytvoření instance třídy má v tomto systému obdobu v naklonování prototypu. Pojmenování prototyp se přeneseně používá pro všechny beztřídní objekty.

Rysy mohou samozřejmě delegovat další rysy a tím vytvářet celou ucelenou hierarchii obdobnou běžným knihovnám tříd.

Protože počet rodičovských slotů v objektech není omezen, mají možnost delegovat svoje zprávy na více rodičů a tím vlastně vytvářet obdobu násobné dědičnosti, která se u nejmodernějších jazyků objevuje velice řídko. Často se nahrazuje bezpečnějším konceptem rozhraní. V Selfu se často využívají tzv. mixins, což jsou malé fragmenty rozhraní, které objektům dodávají určité specifické vlastnosti.

Charakteristickým rysem Selfu je jeho schopnost pracovat s dynamickou dědičností. Objekty totiž mohou reference na své rodiče měnit za svého života a tím mnohdy naprosto zásadním způsobem modifikovat svoje vlastnosti. Tento koncept původní Smalltalk nezná.

2.3 Konkrétnost

Smalltalk i Self jsou oba čistě objektově orientované jazyky. Rozdíl mezi nimi spočívá v tom, do jaké míry dokáží reflektovat svoje sémantické akce. Self v tomto ohledu jde až do extrémů. Příkladem toho může být mechanismus, který využívá pro volání zpráv. Pokud je nějakému objektu (příjemci) zaslána zpráva a v něm nebo v jeho rodiči nalezena vhodná metoda, je z objektu metody vytvořena kopie, která se označuje jako aktivační objekt metody. Této kopii jsou následně nastaveny některé sloty tak, aby referencovaly skutečné argumenty zasláné zprávy. Rodičovský slot aktivačního objektu odkazuje na příjemce zprávy. Samotný kód metody je poté vykonáván v kontextu aktivačního objektu.

Díky tomu Self dokáže obsáhnout i konstrukce, jako jsou lokální proměnné, argumenty nebo bloky, pouze pomocí použití základního konceptu objektů tvořených sloty.

Další unikátní vlastností Selfu je jeho absolutní zapouzdření. Nejenže je zaslání zpráv jediný prostředek, jakým mohou objekty komunikovat mezi sebou, v Selfu je to i jediný prostředek, jakým mohou objekty pracovat samy se sebou, jakým mohou modifikovat svůj stav. To tomuto jazyku dodává na čistotě a programátorům to velice usnadňuje případný refactoring hotových programů.

Beztržidní programování v prostředí Squeak Smalltalk

V syntaktické podobě jazyka se Self pokusil odstranit některé problematické vlastnosti Smalltalku. Používá například lehce odlišné konvence pro pojmenování zpráv a jejich priority. Cílem těchto úprav bylo, aby výsledný kód byl na pohled pro programátora jednoznačnější. Nicméně syntaktické změny jsou oproti Smalltalku spíše kosmetického charakteru a hlavní rozdíl spočívá v tom, že Self umožňuje zapisovat literály pro vytváření objektů.

2.4 Přínos

Beztržidní objektové paradigma nachází uplatnění především v komplexních dynamicky se měnících systémech a obzvláště výhodné je při vytváření grafických uživatelských rozhraní.

Samotný Self dokázal něco, co ve vývoji programovacích jazyků bohužel neobjevuje moc často – výrazně vylepšit stávající programovací jazyk jeho zjednodušením. Self doplnil Smalltalk o násobnou a dynamickou dědičnost, rozhraní, lepší práci s jmennými prostory, lepší zapouzdření apod. To vše bez toho, aby mu cokoliv ubral na jeho vyjadřovacích schopnostech.

Self má díky beztržidnímu přístupu podstatně jednodušší základní objektovou strukturu systému. Beztržidní paradigma umožňuje velice snadno vytvářet to, na co se musí běžně používat specializované návrhové vzory.

Nesmírně cenné je, že těchto výhod nebylo dosaženo na úkor efektivity. Pro Self byla navržena technologie kompilace a optimalizace programů za běhu, která vešla ve známost pod označením HotSpot a kterou firma Sun později s úspěchem využila ve virtuálních strojích pro Javu. To je jeden z důvodů velmi dobrého výkonu Selfu. Pokusně v něm bylo vytvořeno prostředí plně emulující Smalltalk, přičemž se uvádí, že tato implementace byla desetkrát rychlejší než v té době dostupná nejrychlejší komerční implementace Smalltalku.

Celý návrh Selfu je přímo předurčen k vizuálnímu programování, kdy programátor bezprostředně pracuje s grafickou reprezentací objektů, vytváří mezi nimi pomocí konektorů vazby a dynamicky celý systém přetváří podle aktuálních potřeb. Zde leží velký příslib pro vývojová prostředí budoucnosti. Self a jeho dialekty jsou jedny z mála obecně použitelných programovacích jazyků, pro které je nezpochybnitelným přínosem přechod do trojrozměrného prostředí virtuální reality. Již současné implementace Selfu pracující s dvourozměrným grafickým uživatelským rozhraním umožňují souběžnou spolupráci více programátorů. O výhodách, jaké by znamenal přechod do virtuálního prostoru, se můžeme jen domýšlet.

Bohužel se Self sám velkého rozšíření nedočkal. Stojí za tím především přístup společnosti Sun, která preferuje komerčně zajímavější Javu. Pro Self neexistuje například žádný virtuální stroj určený pro Windows a i pro Linux existuje pouze neoficiální verze, která neobsahuje důležité optimalizace a je tedy velmi pomalá. Problematické uplatnění Selfu v praxi byl hlavní motiv pro přenesení myšlenek jazyka Self do prostředí Squeak Smalltalk.

3 Marvin

Squeak je otevřená implementace jazyka Smalltalk-80, která byla vyvinuta v druhé polovině devadesátých. Za jeho vznikem stojí lidé jako Alan Kay a Dan Ingalls, kteří

Smalltalk v sedmdesátých letech vymysleli. Z praktických a licenčních důvodů vyšli z implementace Smalltalku, kterou vytvořila společnost Apple v první polovině osmdesátých let.

Daní za možnost vyjít z již hotového řešení byla jeho jistá zastaralost. Nelze sice říct, že by Smalltalk-80 nebyl moderní programovací jazyk. Naopak, všechny dnes používané jazyky hlavního proudu oproti němu minimálně o jednu generaci ztrácí. Pravdou ale je, že nepostihuje všechny nové myšlenky, které se v oblasti objektově orientovaného programování od poloviny osmdesátých let objevily a z nichž nejvýznamnější byla právě myšlenka beztřídní objektové orientace aplikovaná jazykem Self. Z tohoto pohledu působí Squeak spíše konzervativně.

Na rozdíl od Selfu se ale budoucnost Squeaku jeví podstatně příznivěji. Komunita kolem něj neustále roste a v praxi se prosazuje stále častěji. Přiblížit jej Selfu by mu mohlo dát další podnět k vývoji a rozšířit možnosti jeho reálného nasazení.

Bylo již implementováno několik pokusů doplnit do Squeaku podporu prototypů [1]. Všechny ale trpí dvěma základními nedostatky. V první řadě je to jejich malá efektivita. Delegace zde bývá realizována pomocí zpracování výjimek a prototypy jsou realizovány velmi nehospodárně.

Jejich dalším nedostatkem je samotná syntaxe Smalltalku. Ten totiž nebyl navržen pro beztřídní programování a díky vynucenému nadměrnému používání některých pseudoproměnných jsou takto vytvořené programy málo přehledné. Pro tvorbu větších projektů s nimi počítat nelze.

Problematická je také myšlenka vytvořit implementaci Selfu přímo ve Squeaku. Kvůli jiným konvencím pro zaslání zpráv je prakticky nemyslitelná hladká integrace Selfu se Smalltalkem. Také výkon takového řešení by byl velice špatný, protože by bylo nutné vytvořit např. vlastní správu procesů na aplikační úrovni.

Proto jsem přistoupil na nové komplexní řešení, které je založeno na třech základních pilířích. Novém jazyce, jeho kompilátoru a některých modifikacích stávajícího virtuálního stroje.

3.1 Jazyk

Za účelem doplnění beztřídního přístupu do prostředí Squeak jsem vytvořil nový programovací jazyk, jehož cílem bylo skloubit vhodným způsobem vlastnosti jazyků Self a Smalltalk-80 tak, aby byl snadno integrovatelný se současnou smalltalkovskou infrastrukturou Squeaku.

Ve svých podstatných rysech vychází ze Selfu a lze se na něj dívat jako na jeho dialekt. Na druhou stranu ze Smalltalku přebírá konvence pro zaslání zpráv a pro zápis základních literálů.

Ze Smalltalku jsou také přebrána pravidla pro implicitní návratové hodnoty. Self v tomto ohledu unifikoval bloky a metody tak, aby, pokud není určeno jinak, vraceli standardně výsledek svého posledního výrazu. Prázdné metody v něm z principu vytvořit nelze a prázdné bloky vrací objekt beze slotů. Naproti tomu Marvin převzal smalltalkovský přístup, kdy metody (i prázdné) vrací implicitně příjemce zprávy, bloky výsledek posledního výrazu a prázdné bloky nedefinovaný objekt.

Marvin oproti Selfu ztrácí svoji konkrétnost. Na druhou stranu je přechod na něj pro smalltalkovského programátora velice jednoduchý a přirozený. Největší deviza pak spočívá v integraci se smalltalkovským prostředím.

Beztrždní programování v prostředí Squeak Smalltalk

Při návrhu jazyka Marvin byl brán zřetel na to, aby podporoval zápis základních literálů, jako jsou čísla, řetězce či symboly, v takové formě, v jaké se používají ve Smalltalku. I zde se totiž konvence Selfu v řadě případů liší a některé typy literálů (symboly, znaky, pole apod.) Self vůbec nepodporuje. Bez jejich přítomnosti se obejde díky promyšlenější základní objektové knihovně. Marvin je do svého návrhu doplnil kvůli lepší integraci se Squeakem.

Marvin umožňuje literální zápis objektů ve formě, která přímo vychází se Selfu. Zde jsou zásahy do původního Selfu velmi malé. Byly například pro větší obecnost odstraněny anotační a komentářové sloty s popisy vytvářených objektů. Marvin také unifikoval možné zápisy slovních zpráv na jeden, který odpovídá běžným smalltalkovským zvyklostem.

Další syntaktické změny oproti Selfu jsou již spíše drobnosti. Marvin používá jiný zápis datových a rodičovských slotů, který mu umožňuje uchovat si jednoznačnou gramatiku. Oproti Selfu i Smalltalku poskytuje zápis komentářů ukončených koncem řádku.

Marvin má některá omezení, která vyplynula z jeho provázání se Squeakem. Neumožňuje například vytvářet rodičovské sloty u bloků a metod. Tuto konstrukci sice Self umožňuje, ovšem takto vytvořené rodičovské sloty se doplňují do aktivačních objektů metod či bloků a smysluplné využití se pro ně hledá jen těžko. Z praktického hlediska je tedy toto omezení nepodstatné.

Na druhou stranu přináší Marvin oproti Selfu některá vylepšení, která jsou pro praktické nasazení velmi důležitá. V první řadě je to lepší podpora národních znaků v řetězcích a komentářích. Self také nedokáže korektně pracovat s bloky v datových objektech. Tyto bloky jsou totiž konstruovány již v době překladu. V době běhu programu už ale těmto blokům expirovaly domovské kontexty, takže s nimi již nelze nijak pracovat. Marvin práci s nimi umožňuje.

3.2 Překladač

Návrh jazyka Marvin byl uzpůsoben tomu, aby pro něj byl možný překlad využívající přímo virtuální stroj Squeaku bez další interpretační mezivrstvy. To znamená, že překlad zdrojových kódů v jazyce Marvin probíhá přímo do nativního bytekódu Squeaku. Výsledkem jsou tzv. kompilované metody, které se svojí strukturou ani funkcí nijak neliší od běžných squeakovských kompilovaných metod generovaných ze smalltalkovských zdrojových kódů. To je velice důležitá vlastnost pro plnou integraci se stávající infrastrukturou.

Při tvorbě tohoto překladače bylo nutné důkladně promyslet celou řadu implementačních detailů, které ale mohly celkovou použitelnost výsledného řešení výrazným způsobem omezit. Nejdůležitější se ukazuje návaznost na smalltalkovské bloky.

Bloky hrají ve Smalltalku velmi významnou úlohu. Slouží k definování řídicích struktur i například k práci s procesy. Práce s bloky je proto do značné míry optimalizována virtuálním strojem. Je proto velice praktické použít pro bloky v jazyce Marvin přímo smalltalkovské bloky Squeaku.

Přímému použití bloků ovšem brání fakt, že nepoužívají sloty. Lze v nich používat pouze lokální proměnné, které ale Self ani Marvin nazná. Proto byl vytvořen speciální postup, který umožnil emulovat sloty vytvářené v marvinovských blocích a

metodách pomocí dočasných lokálních proměnných, jaké používá Smalltalk. Tato emulace se týká i argumentových slotů metod a bloků. Emulace probíhá na úrovni překladu a generování bytekódu, což znamená, že není při běhu programu nutná žádná další režie. Naopak je přístup k těmto slotům nejrychlejší.

Základní literály jsou vytvářeny jako běžné smalltalkovské literály. Proto se Marvin obejde bez základní objektové knihovny a s výhodou používá přímo zázemí Squeaku.

Tento překladač byl s úspěchem implementován. Napsán byl ve Smalltalku a přímo využívá dále popsaných modifikací virtuálního stroje.

3.3 Virtuální stroj

Pro to, aby programy založené na prototypch mohly ve Squeaku procovat skutečně efektivně, se ukázalo jako velice výhodné stávající virtuální stroj modifikovat tak, aby podporoval konstrukce nezbytné pro beztrždní programování.

Prvním problémem byla fyzická reprezentace samotných prototypů. Zde jsem se snažil o jejich maximální kompaktnost a paměťovou nenáročnost. Proto bylo vhodné vyhnout se například použití asociací v objektech. Dále bylo nutné zajistit, aby prototypy z pohledu Smalltalku nebyly cizorodým prvkem.

Proto jsou prototypy reprezentovány jako běžné smalltalkovské indexované objekty, přičemž pro každý slot je vyhrazeno několik prvků objektu. Například metody jsou v prototypu uloženy jako dva prvky, z nichž první je reference na symbol se jménem metody a druhý prvek je reference na objekt kompilované metody. Prototypy mají pevně definovanou strukturu, což usnadňuje jejich rychlé prohledávání.

Prototypy se od běžných objektů liší především tím, že se na ně uplatňuje jiný režim zasílání zpráv. Virtuální stroj byl upraven tak, aby přímo podporoval delegaci. To znamená, že pokud je prototypu zaslána zpráva, nevyhledává metody ve své třídě, ale začne prohledávat svoje sloty. Pokud nenalezne vhodný slot, pokračuje rekurzivně v hledání v objektech referencovaných rodičovskými sloty. Při tom podobně jako Self kontroluje vznik cyklů. Marvin není tak restriktivní ve zjišťování případných nejednoznačností, jako je tomu v případě Selfu. To může být při neopatrném zacházení nebezpečné, ale na druhou stranu to tomuto jazyku umožňuje podstatně svobodněji zacházet s přetěžováním jmenných prostorů a rychleji vyhledávat sloty.

Až v okamžiku, kdy objekt prozkoumá všechny rodičovské objekty, začne vyhledávání ve smalltalkovské třídě prototypu. Metody prototypové třídy jsou pak jakási obdoba primitivních metod.

Úpravy virtuálního stroje jsou ovšem rozsáhlejší a zahrnují například i implementaci mechanismu přeposílání zpráv rodičovským objektům apod. Přesto byly provedeny velice citlivě a nad modifikovaným virtuálním strojem lze bez jakéhokoliv zpomalení nebo omezení spouštět i běžné obrazy objektové paměti. Navíc lze provedené úpravy využívat i bez přímé návaznosti na kompilátor jazyka Marvin a využívat je přímo ze Smalltalku.

3.4 Integrace se Squeakem

Obrovskou výhodou zvoleného řešení je hluboká integrace se Smalltalkem. Prototypy jsou navrženy tak, aby v nich bylo možné používat přímo smalltalkovské kompilované metody. Toho se s výhodou využívá k tomu, aby byl ve standardních objektech jazyka Marvin implementován mechanismus, který umožňuje bez omezení pracovat se standardními smalltalkovskými objekty a, přestože se jedná o beztrždní jazyk, i se smalltalkovskými třídami. Navíc mohou být smalltalkovské objekty zapojovány na místo rodičovských objektů prototypů. V tomto případě ovšem není technicky možné provádět na smalltalkovské objekty delegaci a pokud se při prohledávání slotů zjistí, že smalltalkovský rodič dané zprávě rozumí, je mu zpráva pouze přeposlána.

Podobně jako Marvin může využívat Smalltalkovské objekty, může i Smalltalk přímo pracovat s prototypy. Je to dáno tím, že oba typy objektů jsou z principu uzavřené entity a lze s nimi komunikovat pouze zasíláním zpráv.

4 Závěr

Návrh jazyka Marvin není tak čistý, jako je tomu u Selfu, ale to je vyváženo jeho plnou integrací se současným nejoblíbenějším smalltalkovským vývojovým prostředím. Jeho velká podobnost se Smalltalkem a poměrně efektivní implementace mu dávají dobrou startovací pozici. K tomu, aby ji plně využil, je v první řadě potřeba vyladit současnou implementaci a vytvořit respektive upravit nezbytné vývojové nástroje. Pak bude jistě přínosem nejen pro celou řadu výzkumných, ale i praktických aplikací vyžadujících dynamickou a flexibilní základnu.

Literatura

1. Křivánek, P.: *Podpora beztrždního programování ve Squeak Smalltalku* [diplomová práce], FEI VUT Brno, 2005.
2. Sun Microsystems, Inc.: *The Self 4.1 Programmer's Reference Manual*, Sun Microsystems, Inc., 1995-2000.
3. Wolezko, M.; Smith, R. B.: *Prototype-Based Application Construction Using SELF 4.0*, Sun Microsystems, Inc., 1994-2005

Annotation:

Classless programming in Squeak Smalltalk environment

This paper describes basic concepts of classless object orientation, as realized in the Self programming language. It shows why it is in many situations better than common class hierarchy and why can this modern object concept change future development environments. Then it introduces a dialect of Self which was designed to bring classless concept into the Squeak Smalltalk. It describes an implementation of this new language which uses special modified virtual machine. It guarantees effective interpretation of resultant programs. Its properties and future development possibilities are summarized at the end.