Univerzita Komenského
Fakulta Matematiky, Fyziky a Informatiky
Ústav Informatiky

Marián Marcinčák

Refaktorovanie jazyka JavaScript a DHTML

Diplomová práca

Školiteľ : RNDr. Marián Vittek, PhD.

Bratislava 2005

Týmto prehlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedenej literatúry, elektronických zdrojov a s odbornou pomocou školiteľa.

Bratislava, apríl 2005 Marián Marcinčák

Touto cestou by som sa chcel poďakovať môjmu školiteľovi, RNDr. Mariánovi Vittekovi, PhD., za odborné konzultácie a cenné pripomienky a rady pri tvorbe tejto práce.

Ďalej by som rád poďakoval svojim rodičom a priateľom, za ich srdečnú podporu počas môjho štúdia.

Obsah

1. Úvod 8

2. Refaktorovanie 11

2.1 Refaktorovanie a jeho koncepty 11

2.1.1 Definícia refaktorovania 11

2.1.2 Motivácia pre použitie refaktorovania 11

2.1.3 Príklady refaktorovania 12

2.1.4 Kde refaktorovať 18

2.2 Katalóg refaktorovaní 19

2.3 Softvérová podpora refaktorovaní 23

3. Jazyk JavaScript a DHTML 26

3.1 Čo je JavaScript 26

3.2 Syntax jazyka JavaScript 28

3.3 Použitie jazyka JavaScript 32

3.4 JavaScript a koncept DHTML 34

3.5 Integrácia JavaScript kódu s HTML, DOM 38

3.6 Objektový model jazyka JavaScript 40

4. Refaktorovanie jazyka JavaScript 46

4.1 Aplikácia princípov refaktorovania na JavaScript 47

4.2 Špecifiká pri refaktorovaní DHTML 57

4.3 Refaktorovanie a prototypový objektový model 71

4.4 Testovanie refaktorovaní JavaScriptu 75

4.5 Pohľad do budúcnosti 75

5. Záver 77

6. Referencie.. 79

7. Slovník pojmov a skratiek 82

7.2 Zoznam skratiek...82

7.3 Slovník pojmov...84

Príloha A : Bad Smells 86

Príloha B : Základné objekty DOM 88

1. Úvod

Oblasti softvérového inžinierstva a vývoja softvéru sa v posledných rokoch dostalo nemalej popularity. Na celom svete je dennodenne vyvíjané množstvo softvérových systémov a aplikácií v rôznych odvetviach a v rôznych škálach od mobilných zariadení až po komplikované viacvrstvové systémy.

Od čias programovania na nízkej úrovni strojového kódu a procedurálneho programovania sa proces vývoja softvérových systémov radikálne zmenil. Koncepty objektovo-orientovaného programovania, modelov vývoja softvéru od analýzy až po testovanie a údržbu, koncept komponentov a modularizácie, integrácia podnikových systémov a štandardizácie formátov a rozhraní výrazne pomohli k zvýšeniu efektivity vývoja softvéru.

Najväčší dôraz pri vývoji softvéru sa kladie na minimalizáciu nákladov, či už časových, materiálnych alebo ľudských. Vytvorením aplikácie či systému sa však životný cyklus tohto softvéru vôbec nekončí. Veľké množstvo vynaložených prostriedkov súvisí s jeho následnou údržbou. Neustále je nutné vykonávať opravy chýb, pridávať novú funkcionalitu alebo upravovať a odstraňovať existujúcu. Všetky tieto zásahy do systému častokrát neboli pri procese návrhu vôbec plánované a preto majú vo väčšine prípadov za následok degradáciu štruktúry pôvodného systému či dokonca zavedenie nových chýb.

Pri údržbe a úpravách softvéru je preto nutné zachovávať štruktúru a prehľadnosť celého systému tak, aby nedochádzalo k jeho degradácií, ktorá by spôsobila zhoršenie funkčnosti a ďalšieho vývoja tohto softvéru. Jedným zo spôsobov, ako udržať štruktúrovanosť a prehľadnosť živého softvéru je refaktorovanie.

Refaktorovanie je proces softvérového vývoja a údržby, pri ktorom dochádza k zmene štruktúry a návrhu existujúceho kódu, bez zmeny vo funkcionalite či správaní programu resp. systému. Aplikovaním refaktorovaní tak nedochádza k zmenám vo vonkajšom správaní softvéru, ale ku zmenám vo vnútornej štruktúre kódu pre účely lepšej údržby. Ako príklad refaktorovaní možno uviesť extrahovanie často používanej ucelenej časti kódu s jasným účelom do samostatnej funkcie.

Pri minimalizácií nákladov pri vývoji softvéru sa tiež využívajú všetky možnosti znovupoužitia už vytvoreného softvéru (proces tzv. *software reuse*). Softvér, ktorý môže byť takýmto spôsobom používaný musí mať jasne špecifikovanú funkcionalitu a rozhranie, ktorým sa dá k nemu pristupovať a ktorým sa jeho funkcionalita dá využívať. Pomocou refaktorovaní je možné upraviť štruktúru systému tak, aby jeho rozhranie bolo jasne definované.

Okrem prínosu v údržbe existujúceho kódu je tak možné využiť proces refaktorovania aj pri vytváraní komponentov a znovupoužití kódu. Refaktorovanie má teda v oblasti softvérového inžinierstva svoje nezanedbateľné miesto.

S príchodom celosvetovej siete Internet prišlo k obrovskému zlomu v spôsobe využívania softvérových systémov na prístup k publikovaným dátam a informáciám. Vzniklo množstvo jazykov, formátov a systémov podporujúcich možnosti komunikácie medzi servermi, poskytujúcimi svoje služby a klientmi, ktorí ich využívajú. Najpoužívanejším spôsobom komunikácie sa stal World Wide Web, využívajúci protokol HTTP (HyperText Transfer Protocol) na prenos údajov, štruktúrovaných do dokumentov vo formáte jazyka HTML (HyperText Markup Language).

Vďaka HTML dokumentom, zobrazovaným v internetovom prehliadači, získali klienti rýchly a ľahký prístup k štruktúrovaným textovým a multimediálnym dátam. Jednoduchá statická štruktúra HTML dokumentov však bola čoskoro doplnená o komplikovanejšie možnosti zobrazenia dát pomocou štýlov (Cascading StyleSheets, CSS) a o dynamické možnosti narábania so štruktúrou dokumentu pomocou jazyka JavaScript.

JavaScript je skriptovací jazyk, využívaný v hostiteľských prostrediach na vykonávanie rôznych výpočtov a manipuláciu s objektami hostiteľského prostredia. V prostredí internetového prehliadača poskytuje možnosti práce s objektami načítaných HTML dokumentov. Môže tak overovať správnosť vstupných polí formulárov, meniť zobrazované údaje dokumentov či zdynamizovať zobrazenie údajov. Spolu s HTML a CSS tvorí JavaScript koncept tzv. Dynamic HTML (alebo DHTML), ktorý poskytuje klientom dáta a informácie dynamickým spôsobom.

Oblasť návrhu webových dokumentov sa stala samostatným prosperujúcim odvetvím IT, ktoré denne produkuje obrovské množstvo HTML stránok, využívajúcich funkcie jazyka JavaScript a DHTML. Na podporu vývoja a návrhu týchto webových stránok využívajú návrhári rôzne vývojové prostredia. Tieto prostredia sa však sústreďujú hlavne na prácu so samotným HTML, prípadne CSS a ponechávajú tvorbu JavaScript kódu na vývojára. Z dôvodu tejto nedostatočnej podpory jazyka JavaScript vo vývojových prostrediach a faktu, že návrhári webových stránok sú zameraní skôr na grafické zobrazenie obsahu ako na programovú štruktúru kódu, dochádza veľmi rýchlo k degradácií JavaScript kódu použitého v HTML stránkach. K zhoršeniu štruktúry naviac prispieva aj veľká dynamika vývoja stránok, ktoré sú modifikované a aktualizované na pravidelnej báze.

Na zachovanie štruktúry JavaScript kódu vo webových dokumentoch sa preto ukazuje použitie refaktorovania ako nanajvýš užitočné. Refaktorovaním zle vnútorne štruktúrovaných dokumentov je možné docieliť stav, v ktorom sú jednotlivé jazyky použité v rámci HTML dokumentu jasne oddelené, štruktúrované a prehľadné. Tieto vnútorné zmeny umožnia návrhárom stránok ľahší vývoj a aktualizáciu bez zavlečenia chýb alebo tiež použitie vybranej funkcionality v iných dokumentoch alebo projektoch. Vývoj

webových stránok sa tak stane rýchlejším, kvalitnejším a v dôsledku aj menej nákladným.

Cieľ diplomovej práce

Hlavným cieľom tejto diplomovej práce je preskúmať možnosti a spôsoby aplikácie techník refaktorovania na jazyk JavaScript a jeho prostredie. Podľa najlepších vedomostí autora je toto jediná práca, ktorá sa zaoberá refaktorovaním jazyka JavaScript.

Nové výsledky, ktoré práca prináša, napĺňajú ďalší z jej cieľov. Týmto cieľom je zadefinovať katalóg refaktorovaní pre JavaScript, ktorý zhŕňa a popisuje najvýznamnejšie refaktorovania v tomto skriptovacom jazyku.

Popri samotných výsledkoch má práca prínos aj ako úvod do oblasti refaktorovania a do prostredia jazyka JavaScript s odkazmi na ďalšiu literatúru.

Organizácia dokumentu

Práca najprv oboznámi čitateľa s všeobecnou oblasťou refaktorovania, jej konceptami, využitím, príkladmi a problémami, bez ohľadu na cieľový jazyk. Časť o refaktorovaní (Kapitola 2) umožní pochopiť dôležitosť refaktorovania v celej oblasti vývoja a údržby softvéru a poskytne prehľad v tejto oblasti.

Následne sa práca venuje skriptovaciemu jazyku JavaScript, oboznamuje s jeho syntaxou, konštrukciami, objektovým modelom, jeho využitím v prostredí webových prehliadačov a integráciou s inými jazykmi HTML dokumentov. Po prečítaní časti venovanej jazyku JavaScript (Kapitola 3) získa čitateľ ucelený pohľad na tento skriptovací jazyk a jeho použitie pri poskytovaní informácií na Internete.

Hlavná časť (Kapitola 4) skúma a analyzuje možnosti refaktorovania jazyka JavaScript a DHTML v prostredí webových dokumentov. Zaoberá sa možnosťami aplikovania všeobecných princípov refaktorovania na JavaScript a novými spôsobmi refaktorovania, ktoré vyplývajú zo špecifických vlastností objektového modelu jazyka a integrácie s ostatnými jazykmi HTML dokumentov. Výsledné refaktorovania sú zhrnuté do katalógu.

Na záver práca skúma možnosti automatizácie navrhnutých refaktorovaní a navrhuje ďalší postup pri výskume refaktorovania v jazyku JavaScript.

2. Refaktorovanie

2.1 Refaktorovanie a jeho koncepty

2.1.1 Definícia refaktorovania

Refaktorovanie je relatívne nová oblasť softvérového inžinierstva a nie je preto jednoznačne definovaná. Existuje veľké množstvo definícii, ktoré vyplývajú z použitia refaktorovania v rôznych oblastiach.

Martin Fowler, vo svojej veľmi známej knihe "Refactoring: Improving the Design of Existing Code" ([FOW99]), definuje *refaktorovanie* ako zmenu vykonanú v internej štruktúre softvéru, ktorá ho urobí lepšie pochopiteľným a jeho modifikáciu lacnejšou, bez pozorovateľnej zmeny v jeho vonkajšom správaní a funkcionalite. *Refaktorovať* znamená reštrukturalizovať softvér aplikovaním sledu refaktorovaní zlepšujúcich štruktúru bez pozorovateľnej zmeny v jeho správaní.

Alternatívne možno definovať refaktorovanie ako proces, pri ktorom sa rôznymi spôsobmi prebuduje objektový návrh systému tak, aby bol flexibilnejší a/alebo ľahko znovupoužiteľný a aby každá časť funkcionality existovala práve na jednom mieste v rámci softvéru ([WIK05], Ralph Johnson, Ron Jeffries). Refaktorovanie sa dá chápať tiež ako prepisovanie kódu z dôvodu jeho „vyčistenia" alebo ako presúvanie funkcionality z jednej časti programu do druhej.

Na základe týchto definícií možno vidieť, rôznosť oblastí a smerov výskumu refaktorovania. Všetky smery však majú spoločný princíp - reštrukturovať systém spôsobom, ktorý nemení jeho funkcionalitu, ale zato produkuje lepšie a prehľadnejšie navrhnutý systém.

2.1.2 Motivácia pre použitie refaktorovania

Či už programátori pracujú na návrhu a vývoji úplne nového softvéru, alebo sa snažia pochopiť a zmeniť neudržiavaný legacy softvér, nasledujúce faktory ich môžu motivovať k úvahám o reštrukturalizácii softvéru ([FOW99], Ken Beck, str. 60). Z hľadiska budúcich úprav sa totiž ťažko modifikujú programy, ktoré :

- sa aj ťažko čítajú,
- majú duplikovanú logiku,

- vyžadujú dodatočné správanie, kvôli ktorému je nutné zmeniť bežiaci kód,
- majú zložitú logiku podmienok.

Pri všetkých týchto programoch je vhodné použiť refaktorovanie. Tento proces nie je síce všeliek pre návrh objektovo-orientovaného kódu, prináša so sebou ale množstvo výhod ([FOW99]) :

- Zlepšuje návrh softvéru.
- Robí softvér ľahšie pochopiteľným.
- Pomáha pri hľadaní chýb v kóde.
- Pomáha programovať rýchlejšie.

Bez použitia refaktorovania pri návrhu kódu v softvérovom systéme nevyhnutne dôjde ku zhoršeniu jeho kvality kvôli konštantným zmenám počas jeho vývoja a údržby. Zlepšenie návrhu štruktúry kódu eliminuje duplikáciu kódu a pomáha pri jeho pochopení.

Všetky výhody refaktorovania sú navzájom poprepájané. Zlepšením návrhu softvéru je ho totiž možné ľahšie pochopiť. Lepším pochopením systému môže vývojár ľahšie nájsť existujúce chyby a identifikovať potencionálne chyby. Vďaka tomu je možné ľahšie implementovať a integrovať nový dizajn a funkcie do už existujúceho dobre navrhnutého systému, čo spôsobí rýchly pokrok pri vývoji.

Refaktorovanie je jednou zo základných súčastí vývojového procesu známeho ako *extrémne programovanie* (Extreme Programming, XP). Extrémne programovanie pristupuje k návrhu a vývoju softvéru inkrementálnym spôsobom – implementovaním po ucelených častiach nasadenia, ktoré vždy začínajú od najjednoduchšieho riešenia. Každá z týchto častí nasadenia pritom požaduje refaktorovanie kódu a jeho testovanie tak, aby sa zachovala vysoká úroveň celkového návrhu systému. Podľa slov jeho tvorcov je extrémne programovanie "ľahký, efektívny, bezpečný, flexibilný, vedecký a zábavný spôsob vývoja softvéru" ([BECK99]).

2.1.3 Príklady refaktorovania

Najlepší spôsob ako pochopiť, čo je to refaktorovanie, je uviesť praktický príklad časti programu so zlým návrhom, ktorý môže byť upravený pomocou pár jednoduchých krokov.

Jednoduchý príklad

Najjednoduchšou ukážkou refaktorovania je *premenovanie metódy* ("Rename Method"). Motiváciou k použitiu tohto refaktorovania býva častokrát nevhodné, príliš všeobecné alebo mätúce pomenovanie metódy objektov. Cieľom je pomenovať metódu tak, aby čo najlepšie zodpovedala jej významu. Z metódy `getinvcdtlmt` triedy `Customer` na výpočet maximálneho limitu na účte zákazníka banky, je možné refaktorovaním premenovania

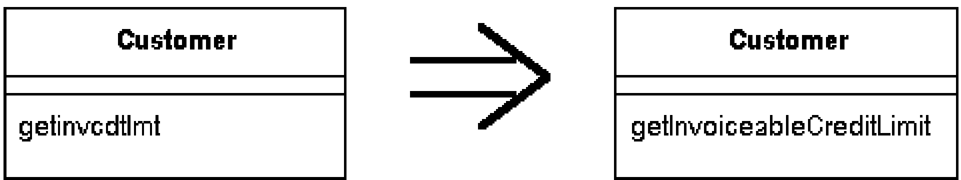
dostať metódu `getInvoiceableCreditLimit` nasledovne (kód je napísaný v jazyku Java) :

```
public class Customer {
      double getinvcdtlmt() {
            ...
      }
}
```

⇓

```
public class Customer {
      double getInvoiceableCreditLimit() {
            ...
      }
}
```

Použitie tohto refaktorovania je možné znázorniť aj pomocou diagramu tried v jazyku UML (Unified Modelling Laguage). UML je unifikovaný rozšírovateľný modelovací jazyk na podporu vývoja softvéru. Popis jazyka UML a jeho modelovacích schopností pomocou diagramov je mimo rozsahu tejto práce. Detailnú špecifikáciu UML možno nájsť na [OMG05]. Aplikáciu refaktorovania z tohto príkladu v jazyku UML možno vidieť na obrázku 1. Trieda ako je napríklad `Customer` je v UML zobrazovaná obdĺžnikom s troma oddelenými časťami. V prvej sa nachádza názov triedy, v druhej jej atribúty a v tretej jej metódy.



Obrázok 1. Refaktorovanie "Rename Method".

Pri refaktorovaní však nestačí len obyčajné textové prepísanie mena metódy. Je nutné vyhľadať všetky zodpovedajúce volania metódy a premenovať ich so zreteľom na úroveň ("scope") použitia metódy. Funkcionalita totiž musí byť zachovaná a to bez vnesenia nových chýb.

Zložitejší príklad

Nasledujúci komplexnejší príklad demonštruje len malú časť kódu a preto dôležitosť jeho použitia v tomto meradle nemusí byť úplne zrejmá. V skutočnosti pri aplikovaní podobných jednoduchých krokov vo veľkom množstve za sebou spôsobí dramatické zlepšenie štruktúry programu. Tento príklad slúži na demonštrovanie niektorých procesov zahrnutých v refaktorovaní.

Príklad, prevzatý z [FOW99] (str. 55), je napísaný v objektovo-orientovanom jazyku Java a je zameraný na refaktorovanie nahradzujúce podmienku polymorfizmom („Replace Conditional with Polymorphism"). Trieda nazvaná „Bird" obsahuje príkaz podmienky v metóde na výpočet rýchlosti letu vtákov na základe ich druhu. Každý druh má rôzne požiadavky ovplyvňujúce rýchlosť jeho letu.

```
class Bird {
  public double getSpeed() {
  ...
    switch (_type) {
      case EUROPEAN :
        return getBaseSpeed();
      case AFRICAN :
        return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
      case NORWEGIAN_BLUE:
        return (_isNailed) ? 0 : getBaseSpeed(_voltage);
    }
    throw new RuntimeException("Unreachable");
  }
  ...
}
```

Horeuvedený kód využíva príkaz `switch`, ktorého použitie je pozostatkom procedurálneho programovania a nie je v súlade s dobrým štýlom objektovo-orientovaného programovania. Odstraňuje síce potrebu použitia množstva príkazov `if`, `then` a `else`, vývojárovi však veľmi nepomáha v pochopení a modifikovaní kódu. Pridanie nového druhu vtákov si totiž žiada dodatočné pridanie vetvy do všetkých výskytov príkazu `switch` s rovnakými podprípadmi. A týchto výskytov je nutne veľa, keďže treba vždy rozpoznávať vtáka podľa druhu. Pri pridávaní sa potom často zabudnú rozšíriť niektoré výskyty `switch` o nový podprípad (napríklad výskyty v kolegovom kóde), čím sa do dobre fungujúceho systému zanesú zbytočné chyby.

Pri veľkom počte druhov sa naviac príkaz `switch` stane príliš veľkým, ťažko pochopiteľným a ťažko modifikovateľným ([FOW99], str. 82).

Jedným z riešení problému s príkazom `switch` je použitie polymorfizmu – základného princípu objektovo-orientovaného programovania. Polymorfizmus „umožňuje programátorovi poslať rovnakú správu objektom z rôznych tried", čo pomáha pri „písaní kódu, ktorý je ľahko modifikovateľný a rozšíriteľný" ([WU01]). V tomto príklade by metóda `getSpeed()` volaná nad ľubovoľným z rôznych druhov vtákov, vracala jeho jedinečnú rýchlosť. Pri implementácii tohto správania sa zmení trieda `Bird` a jej metóda `getSpeed` na abstraktnú.

```
public abstract class Bird {
        ...
        abstract public double getSpeed();
        ...
}
```

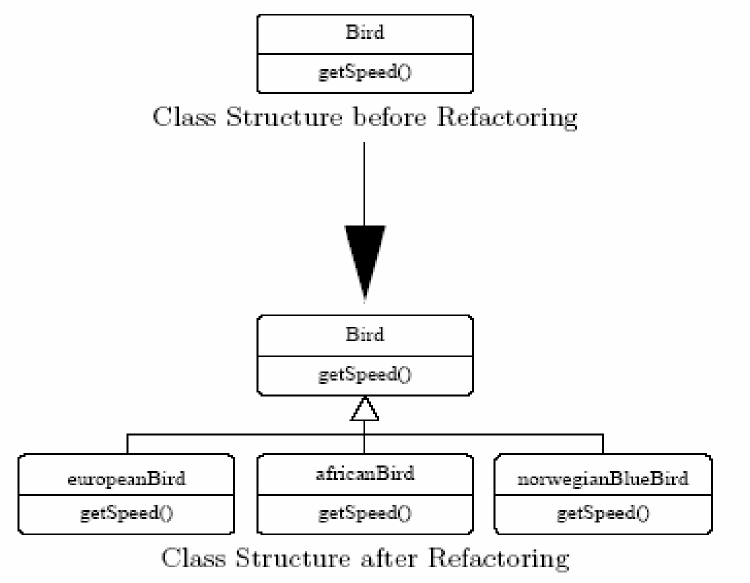
Pre každý druh vtákov bude následne definovaná samostatná trieda, obsahujúca vlastnú implementáciu metódy `getSpeed`, na základe charakteristík rýchlosti druhu.

```
class EuropeanBird extends Bird {
  ...
  public double getSpeed() {
    return getBaseSpeed();
  }
  ...
}
class AfricanBird extends Bird {
  ...
  public double getSpeed() {
    return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
  }
  ...
}
class NorwegianBlueBird extends Bird {
  ...
  public double getSpeed() {
    return (_isNailed) ? 0 : getBaseSpeed(_voltage);
  }
  ...
}
```

Vypísanie rýchlosti všetkých vtákov môže byť potom implementované nasledovne :

```
void printSpeed(Bird[] birds) {
  for (int i=0; i < birds.length; i++) {
    System.out.println("" + birds[i].getSpeed());
  }
}
```

Vyššiu abstraktnejšiu reprezentáciu celkového návrhu horeuvedeného kódu pred a po zmenách možno vidieť na UML modeli obrázku 2. Rozdielom medzi oboma objektovými diagramami je pritom znázornenie rôznych druhov vtákov na druhom diagrame pomocou dedičnosti, čo má nepochybne za následok lepšie pochopenie fungovania systému vývojármi.



Obrázok 2. Refaktorovanie „Replace Conditional with Polymorphism".

Toto riešenie vylepšilo celkový návrh zavedením hierarchickej štruktúry tried objektov. To umožní ľahké a jednoduché pridávanie nových druhov, vytvorením novej triedy oddedenej z hierarchickej štruktúry a zahŕňajúcej nové informácie a vlastnosti systému. Čo je však podstatné, takéto pridanie do súčasného fungujúceho systému nemá žiadny vplyv na jeho celkovú stabilitu, jediné čo stačí urobiť je alokovanie a volanie novovytvorenej triedy. Zlepšenie štruktúry systému má teda za následok stabilnejší, rozšíriteľný a ľahko udržiavateľný systém.

Refaktorovanie a jeho proces

Príklad z predchádzajúcej časti napriek svojej jednoduchosti ukazuje, že refaktorovanie je veľmi komplexný proces. Je nutné sa na celý problém pozrieť zvrchu a premyslieť, kde začať s implementáciou. V prípade, že sa refaktorovania vykonávajú manuálne a nekontrolovaným spôsobom, hrozí nebezpečenstvo zavedenia chýb do fungujúceho systému. To by však v prípade stabilného systému spochybňovalo význam a dôležitosť refaktorovania kódu.

Jedným zo spôsobov, ako sa vyhnúť zavlečeniu chýb do kódu počas procesu refaktorovania, je rozložiť zložitosť zmien na malé, ľahšie zvládnuteľné a kontrolovateľné kroky (, čo je jedným z princípov extrémneho programovania). Po každom z týchto krokov sa odporúča prekompilovať a otestovať systém, čím sa do veľkej miery zabráni zavedeniu chýb. Ak by aj chyby vznikli, je pomerne ľahké ich vystopovať, keďže súvisia s posledným krokom refaktorovania.

Ideu rozdelenia procesu na malé kroky, za účelom pochopenia princípov jeho aplikácie, demonštruje aj ďalší príklad z Fowlerovej knihy ([FOW99]). Je ním postup pri použití refaktorovania „Add Parameter" na pridanie nového parametra metódy v situácií „ak metóda potrebuje viac informácií od volajúceho" :

- Overte, či je metóda implementovaná nadtriedou alebo podtriedami. Ak áno, zmeny z nasledujúcich krokov musia byť vykonané aj pre ne.
- Deklarujte novú metódu s pridaným parametrom. Skopírujte telo starej metódy do novej.
- Prekompilujte.
- Zmeňte telo starej metódy tak, že bude volať len novú metódu.
- Prekompilujte a otestujte.
- Nájdite všetky volania a referencie k starej metóde a zmeňte ich na referencie na novú metódu. Prekompilujte a otestujte po každej zmene.
- Odstráňte starú metódu.
- Prekompilujte a otestujte.

Na horeuvedenom postupe si je možné všimnúť dôležitosť pravidelného testovania vykonaných zmien, čím sa zníži riziko zavedenia nových chýb pri aplikovaní refaktorovania. Na automatickú podporu testov možno využiť špeciálne testovacie nástroje ako napríklad voľne šíriteľné prostredie JUnit pre jazyk Java (Erich Gamma a Kent Beck [JUN03]). Testovanie má význam hlavne pri manuálnom aplikovaní refaktorovaní na programový kód, tak ako to navrhuje Fowler. Pri používaní špecializovaných softvérových nástrojov, ktoré automatizujú použitie refaktorovaní by malo byť riziko zavedenia chýb omnoho menšie.

Kedy refaktorovať

Refaktorovania by mali byť aplikované systematicky počas celého vývoja softvéru systémom „málo a často". Tento spôsob pomáha navrhnúť a vyvinúť kód na vysokom štandarde. Postupne ako sa vývojári stávajú skúsenejšími a istejšími pri aplikovaní refaktorovaní, malo by sa refaktorovanie stať ich denne používanou rutinou.

Počas procesu vývoja sa vyskytnú kľúčové momenty, kedy je vhodné použiť refaktorovanie alebo prípadne inú cestu k lepšej štruktúre kódu. Hlavnými takýmito momentmi sú nasledovné :

- **Pravidlo troch (rule of three)** – Don Roberts ([FOW99], str. 58) tvrdí, že písanie kódu raz je v poriadku. Pri druhom použití rovnakého alebo podobného kódu ešte stále netreba meniť štruktúru kódu. Pri treťom výskyte podobného kódu by však už malo byť použité refaktorovanie na redukciu duplikovaného kódu.
- **Pridávanie funkcionality** – Pridanie nových funkcií je v každom kroku vývoja komplikované a refaktorovanie by pri pridaní malo pomôcť. Použitie refaktorovaní pred pridaním novej funkcionality ju pomôže ľahšie integrovať so zvyškom systému a použitie po pridaní umožňuje vyčistiť návrh a kód rozšíreného systému.
- **Oprava chyby** – Nájdenie, oprava a prevencia chýb v kóde je častokrát veľmi náročná. Je totiž nutné pochopiť fungovanie časti systému, prípadne systému ako celku. Pri lepšom pochopení systému a vďaka jeho lepšej štruktúre vzniká menej chýb a aj hľadanie existujúcich je jednoduchšie a rýchlejšie.
- **Počas revízie kódu (code review)** – Revízia kódu by mala byť vykonávaná pravidelne, hlavne kvôli šíreniu vedomostí o systéme v rámci vývojového tímu. Takto je možné spoločne navrhnúť oblasti kódu na zlepšenie a navrhnúť aj smer ďalšieho vývoja.

Problémy pri procese refaktorovania

Nie vždy je proces refaktorovania tak jednoduchý ako v horeuvedených prípadoch. Niekedy na možnosť aplikovania zmien vplývajú

vonkajšie faktory, ako napríklad externá databáza alebo rozhranie, ktoré je pripojené k systému, avšak nemôže byť žiadnym spôsobom pozmenené. Všetky volania na tieto externé súčasti teda nemôžu byť na rozdiel od lokálneho kódu systému modifikované.

Refaktorovanie pomáha vývojárovi jednoduchšie pochopiť systém pomocou zmeny jeho návrhu na jemu známejšiu štruktúru. V prípade malého systému s jedným vývojárom je všetko v poriadku, pri veľkom tíme s odlišnými konvenciami a zvykmi medzi členmi tímu však prestáva mať refaktorovanie z hľadiska lepšieho pochopenia kódu význam. Každý člen tímu má svoj vlastný štýl a predstavu o návrhu systému – použitie refaktorovania na pochopenie kódu teda nikdy nemôže viesť k stabilnému systému.

V niektorých prípadoch môže byť použitie refaktorovania nákladné a neefektívne, keďže každá snaha o opravu návrhu systému je finančne a časovo príliš náročná. V takomto prípade je jediným spôsobom na dosiahnutie lepšieho dizajnu kompletné prepísanie celého systému od nuly.

2.1.4 Kde refaktorovať

Identifikovať miesta so zlým dizajnom v kóde, ktoré je nutné refaktorovať, je ťažká úloha. Tieto oblasti so zlým návrhom sú známe ako „Bad Smells" (, t.j. „zlé pachy") alebo „Stinks". Ich nájdenie súvisí skôr s „ľudskou intuíciou" ako s exaktnou vedou ([FOW99], Kent Beck, str. 75), pričom skúsenosti vývojárov sú pri ich hľadaní veľmi dôležité.

Na pomoc pri identifikovaní "Bad Smells" načrtli Fowler a Beck ([FOW99], str. 75-88) potencionálne oblasti kódu na refaktorovanie. Tieto oblasti sú zhrnuté v samostatnej prílohe A na konci tejto práce. V tejto časti sú uvedené len tie najdôležitejšie.

Duplikovaný kód

Hlavným dôvodom na refaktorovanie je odstránenie duplicity v kóde. V prípade, že sa na viacerých miestach v systéme nachádza kód rovnakého alebo podobného charakteru, môže dôjsť ku problémom pri úprave systému. Častokrát sa totiž tieto duplicity nachádzajú na rôznych miestach a pri pridávaní či úprave v jednej oblasti sa zabudne na ostatné. To spôsobí zavedenie chýb, ktoré je ťažké vystopovať. Odstránením duplicít sa sústredí kód na jedno miesto, čo umožňuje bezpečnú a samozrejme aj jednoduchšiu úpravu v budúcnosti a naviac zvýši prehľadnosť celého kódu a orientáciu v ňom.

Dlhá metóda

Prítomnosť príliš dlhej metódy v kóde systému predstavuje ďalšiu z oblastí, kedy je vhodné uvažovať o refaktorovaní. Dlhé metódy sú pozostatkom procedurálneho programovania a pri objektovom modelovaní už

ich použitie nie je opodstatnené. Takáto metóda vykonáva príliš veľa funkcií, je ťažké sa v nej orientovať a môže sa ľahko stať zdrojom chýb pri úpravách.

2.2 Katalóg refaktorovaní

Katalóg refaktorovaní slúži na lepšiu organizáciu a vyhľadávanie jednotlivých refaktorovaní. Ku každému refaktorovaniu v katalógu je okrem jeho kategorizácie priložený aj podrobný popis refaktorovania, dôvod jeho použitia, postup jeho použitia, príklady a podmienky za ktorých je možné refaktorovanie použiť.

Martin Fowler vo svojej knihe „Refactoring : Improving the Design of Existing Code" používa katalóg so 72 refaktorovaniami, ktoré sú určené pre celé spektrum profesionálnych programátorov a popisujú celý proces ich manuálneho aplikovania na kód. Tento katalóg nie je uzavretý, slúži hlavne ako základ pre tvorbu nových refaktorovaní. Na stránkach ako [FOWW] a [WIK05] možno nájsť rozšírenia Fowlerovho katalógu o nové refaktorovania.

Zvyšok tejto časti demonštruje niektoré zo základných refaktorovaní z katalógu, každé so stručným popisom a príkladom. Kompletný zoznam refaktorovaní a podrobnejšie informácie o každom z nich možno nájsť v knihe [FOW99].

Refaktorovanie : Extrakcia metódy („Extract Method")

Motivácia : Existuje fragment kódu, ktorý môže byť spolu zoskupený.

Postup : Zmení fragment kódu na metódu, ktorej meno popisuje jej účel.

Príklad : Ukazuje extrakciu označeného kódu do metódy `printDetails`.

```
void printOwing() {
  printBanner();

  //print details
  System.out.println ("name: " + _name);
  System.out.println ("amount " + getOutstanding());
}
```

⇓

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails (double outstanding) {
  System.out.println ("name:  " + _name);
  System.out.println ("amount " + outstanding);
}
```

Refaktorovanie : Vloženie tela metódy („Inline Method")

Motivácia : Telo metódy je rovnako jasné ako jej meno.

Postup : Vloží telo metódy namiesto všetkých jej volaní a odstráni metódu.

Príklad :

Príklad demonštruje dosadenie jednoduchého tela metódy `moreThanFiveLateDeliveries` do jej volania v metóde `getRating`.

```
int getRating() {
  return (moreThanFiveLateDeliveries()) ? 2 : 1;
}
boolean moreThanFiveLateDeliveries() {
  return _numberOfLateDeliveries > 5;
}
```

⇓

```
int getRating() {
  return (_numberOfLateDeliveries > 5) ? 2 : 1;
}
```

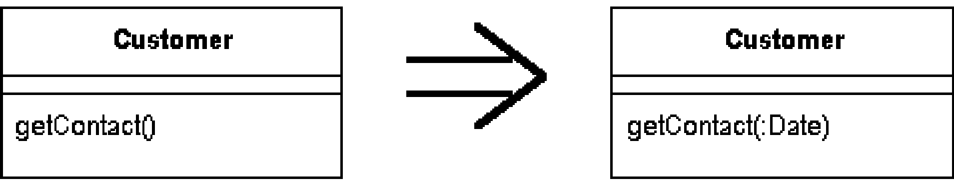
Horeuvedené dve refaktorovania nie sú umiestnené na začiatok náhodne. V prvom rade sa jedná o veľmi užitočné refaktorovania a to najmä v prípade extrakcie metódy. Automatizácia tohto refaktorovania je kvôli svojej zložitosti považovaná za tzv. „refaktorovací Rubikon" ([FOW01]). Okrem užitočnosti sú však tieto refaktorovanie navzájom komplementárne. Takého refaktorovania, ktoré reprezentujú oba smery úpravy a po ich aplikácií vznikne pôvodný kód, sa nazývajú *inverzné refaktorovania*. Každé z nich je plnohodnotným refaktorovaním a má svoj právoplatný význam. V niektorých prípadoch je výhodnejšie použiť jedno a v iných druhé refaktorovanie.

Refaktorovanie : Pridanie parametra metódy („Add Parameter")

Motivácia : Metóda potrebuje viac informácií od svojho volajúceho.

Postup : Pridá parameter, ktorý odovzdáva tieto informácie.

Príklad :



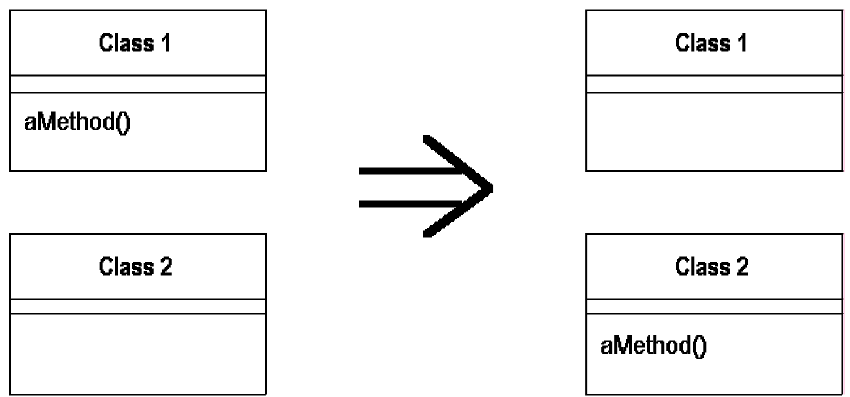
V horeuvedenom UML diagrame je do metódy `getContact` triedy `Customer` pridaný nový parameter typu `Date`, obsahujúci dátum, pre ktorý majú byť kontaktné informácie metódy získané.

Refaktorovanie : Presunutie metódy („Move Method")

Motivácia : Metóda používa alebo je používaná (či bude používať alebo bude používaná) viacerými vlastnosťami inej triedy, ako triedy, v ktorej je definovaná.

Postup : Vytvorí novú metódu s podobným telom v triede, ktorá starú metódu viac používa. Následne zmení starú metódu na odkaz k novej, alebo ju úplne odstráni.

Príklad :



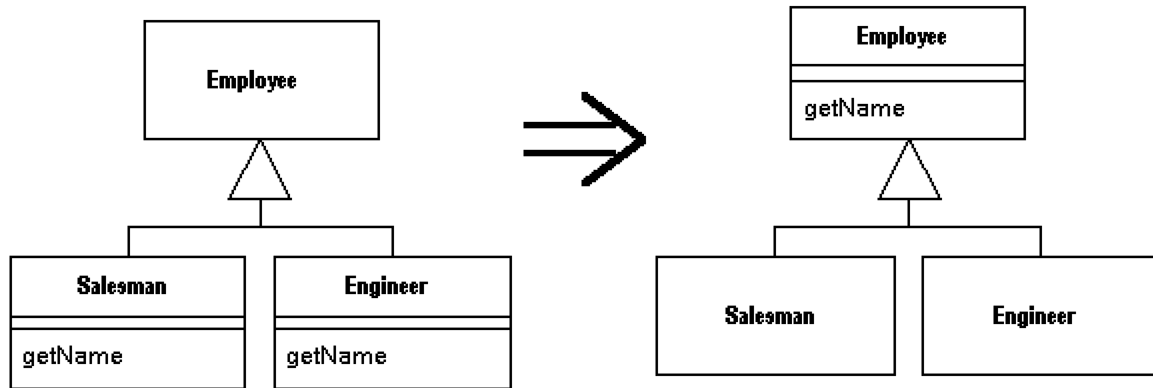
UML diagram príkladu ukazuje premiestnenie metódy `aMethod` z triedy `Class1` do triedy `Class2`, kde je jeho umiestnenie vhodnejšie.

Refaktorovanie : Vyzdvihnutie metódy („Pull Up Method")

Motivácia : V podtriedach existujú metódy s identickými výsledkami.

Postup : Presuň ich do nadtriedy.

Príklad :



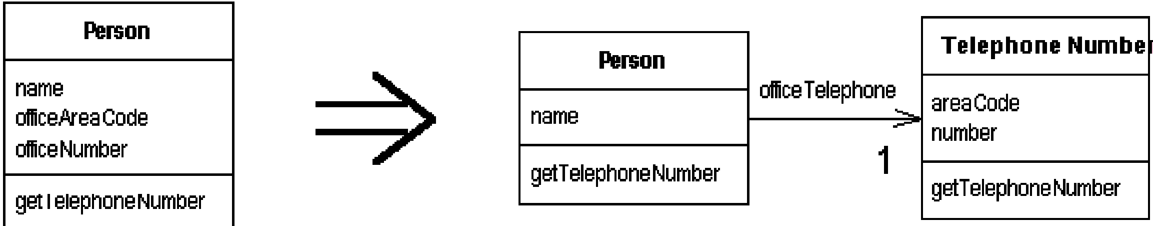
Horeuvedený UML diagram ukazuje presunutie metódy `getName` z podtried `Salesman` a `Engineer` hierarchie dedičnosti do ich nadtriedy `Employee`.

Refaktorovanie : Extrakcia triedy („Extract Class")

Motivácia : Jedna trieda robí prácu, ktorú by mali robiť dve.

Postup : Vytvorí novú triedu a presunie relevantné premenné a metódy zo starej triedy do novej.

Príklad :



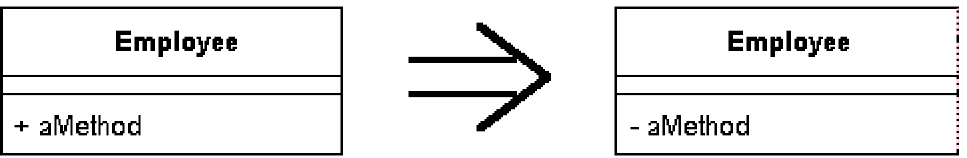
UML diagram v príklade ukazuje vytvorenie novej extrahovanej triedy `TelephoneNumber`, do ktorej sú premiestnené zodpovedajúce atribúty `officeAreaCode` a `officeNumber` a metóda `getTelephoneNumber`. Atribúty sú pritom premenované na `areaCode` resp. `number`. Pôvodnej triede `Person` je po refaktorovaní priradený nový atribút `officeTelephone`, ktorý obsahuje objekt novej triedy `TelephoneNumber` so zodpovedajúcimi hodnotami.

Refaktorovanie : Skrytie metódy („Hide Method")

Motivácia : Metóda nie je používaná žiadnou inou triedou.

Postup : Zmení viditeľnosť metódy na *private*.

Príklad :



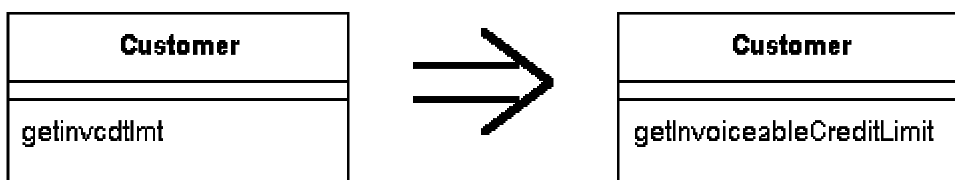
Na UML diagrame sa viditeľnosť atribútu, metódy či objektu znázorňuje ako + pri verejnej viditeľnosti (public) a – pri súkromnej (private).

Refaktorovanie : Premenovanie metódy („Rename Method")

Motivácia : Názov metódy dostatočne nepopisuje jej význam.

Postup : Zmení názov metódy.

Príklad :

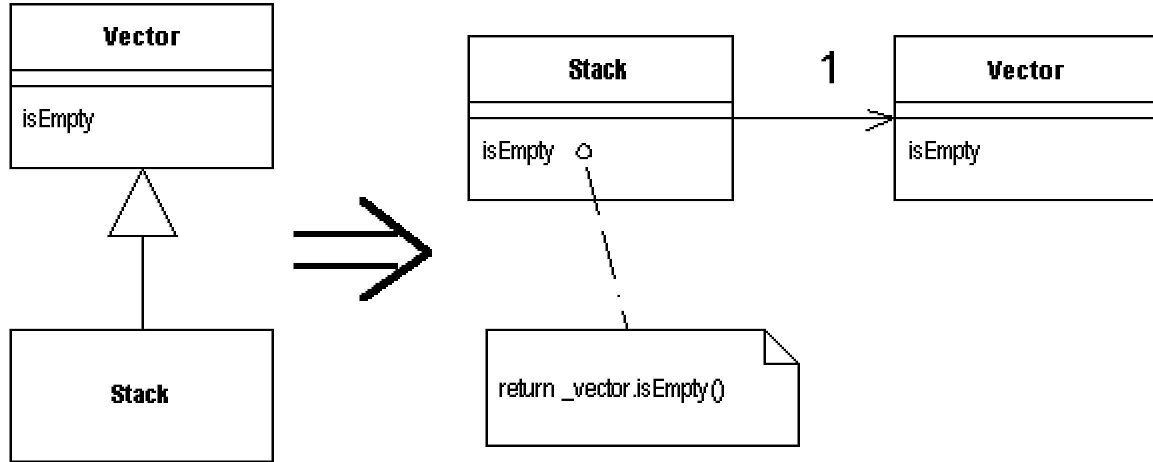


Refaktorovanie : Nahradenie dedičnosti volaním („Replace Inheritance with Delegation")

Motivácia : Podtrieda používa len časť rozhrania nadtriedy alebo nechce dediť dáta.

Postup : Vytvorí metódu nadtriedy. Upraví metódu, aby sa odkazovala na nadtriedu a odstráni dedičnosť.

Príklad :



Horeuvedený príklad znázornený UML diagramom ukazuje v pôvodnom stave dedičnosť medzi triedou `Vector` a `Stack`. Tá je aplikovaním refaktorovania odstránená. Keďže trieda `Stack` pôvodne zdedila metódu `isEmpty` od svojho rodiča, táto metóda je v novom stave implementovaná jednoduchým odkazom na metódu `isEmpty` triedy `Vector`.

2.3 Softvérová podpora refaktorovaní

Charakter refaktorovaní je vo svojej podstate jazykovo špecifický. V počiatkoch vývoja sa refaktorovanie (alebo reštrukturovanie) sústredilo hlavne na štruktúrované a objektovo-orientované jazyky. Medzi hlavné jazyky patrili C++ a Smalltalk, kvôli ich popularite v tom čase. V súčasnosti sa refaktorovanie sústredí na iný objektovo-orientovaný jazyk s odlišnou sémantikou, na jazyk Java.

Pri vývoji softvéru pomáhajú vývojárom do veľkej miery softvérové nástroje známe ako integrované vývojové prostredia (Integrated Development Environments – IDE), ktoré v sebe zahrňujú množstvo techník softvérového inžinierstva. Výhodou týchto prostredí je fakt, že vývojár sa potrebuje zoznámiť len s jedným nástrojom, ktorý môže sprístupňovať rôzne funkcie a reprezentácie kódu. Refaktorovania boli začlenené do týchto prostredí na uľahčenie a pomoc pri celkovom procese vývoja softvéru.

Automatizovaním refaktorovaní sa tiež umožní vývojárom aplikovať refaktorovania ľahko a s menším rizikom zavedenia chýb.

V súčasnosti existuje množstvo rôznych IDE nástrojov, ktoré podporujú refaktorovania na rôznych úrovniach. Medzi najznámejšie patria nasledovné nástroje :

- **IntelliJ IDEA** od JetBrains - IDE prostredie pre vývoj v jazyku Java podporujúce 28 rôznych refaktorovaní v súčasnej verzii 4.5 ([IDEA05])
- **XRefactory** - refaktorovací browser pre Emacs a XEmacs určený pre jazyky C++ a Java (Marián Vittek [XREF05]) v súčasnej verzii 2.0.5. Ako prvý prekonal tzv. „refaktorovací Rubikon" pre Java a C++ ([FOW01]), implementovaním refaktorovania extrakcie metódy („Extract Method").
- **Eclipse** od IBM a iných spoločností – open-source projekt poskytujúci IDE hlavne pre jazyk Java s možnosťou rozširovania o nové funkcie a aj na iné jazyky cez plugins. V súčasnosti vo verzii 3.0.1 poskytuje 26 refaktorovaní ([ECL05]).
- **Together** od Borland – CASE nástroj s podporou použitia refaktorovaní nad UML ([TOG05]).
- **JFactor** od Instantiations – refaktorovací nástroj pre VisualAge a jazyk Java podporujúci 11 refaktorovaní vrátane extrakcie metódy ([JFA05]).
- **JFactory** – voľne šíriteľný nástroj obsahujúci 15 refaktorovaní pre jazyk Java s podporou rôznych IDE prostredí (Chris Seguin a Mike Atkinson, [JREF05]).
- **Refactoring Browser** – refaktorovací browser pre jazyk Smalltalk s podporou refaktorovaní (J. Brant a D. Roberts, [RB05]).
- **JBuilder** od Borland – vývojové IDE prostredie s podporou distribuovaného refaktorovania ([JBU05]).
- **...**

Pri nástrojoch podporujúcich refaktorovania nie je dôležitý len ich počet, ale aj spôsob, akými ich je možné z užívateľského rozhrania vykonávať. Pokročilé nástroje poskytujú možnosť zobraziť súčasný stav pred refaktorovaním a budúci stav po refaktorovaní, ktorý do veľkej miery pomáha pri správnej aplikácií refaktorovania. Keďže refaktorovania vykonávajú zmeny nad kódom, ktoré nemusia byť vhodné, dôležitou je možnosť podpory funkcie „undo" (návratu na pôvodný stav kódu) a prípadne „redo" (opätovné aplikovanie zmien znovu po návrate). Funkcie nad refaktorovaniami by mali čo najviac zjednodušiť a sprehľadniť ich použitie tak, aby vývojár mal stálu kontrolu a prehľad nad ich aplikáciou.

Nástroje na refaktorovanie ponechávajú v súčasnosti identifikáciu „Bad Smells" a výber vhodného refaktorovania, kvôli ich komplikovanosti, na užívateľovi a sústreďujú sa hlavne na refaktorovanie už vybranej časti kódu pomocou zvoleného refaktorovania.

Implementácia refaktorovaní

Vývoj nástroja na automatizáciu procesu refaktorovaní je veľmi zložitý. Ako príklad možno uviesť zdanlivo jednoduché refaktorovanie premenovania metódy ("Rename Method").

V súčasnosti už skoro všetky textové editory poskytujú možnosť nahradenia zadaného textu iným. Operácia, ktorá je korektná v obyčajnom texte však môže pri použití nad programovým kódom spôsobiť jeho znefunkčnenie. Názov premenovávanej metódy totiž môže byť rovnako názvom metódy inej triedy alebo názvom na inej úrovni. Priame premenovanie na úrovni textu by nahradilo úplne všetky výskyty, čo nie je korektné. Je preto nutné, aby refaktorovací nástroj mal všetky potrebné informácie o syntaktických štruktúrach refaktorovaného jazyka a kódu.

Štruktúru refaktorovaného kódu si nástroj ukladá do pamäti vo forme tzv. *abstraktného syntaktického stromu* (abstract syntax tree, AST). Pri aplikovaní refaktorovania sa pracuje práve nad týmto hierarchickým stromom, ktorý je modifikovaný príslušným spôsobom. Po aplikovaní refaktorovania je nutné výslednú podobu AST stromu zobraziť vo forme pôvodného kódu. Na generovanie programového kódu na základe AST slúži *pretty-printer*, ktorý sa postará o finálne zobrazenie a odsadenie programového kódu.

Horeuvedený postup implementácie nástroja automatizujúceho refaktorovanie je načrtnutý veľmi jednoduchým spôsobom. V skutočnosti je nutné z hľadiska optimalizácie uchovávať komplikovaný a rozsiahly model AST stromu v pamäti čo najefektívnejšie, aby bolo možné rýchlym spôsobom vykonávať refaktorovania a následný pretty-printing. Implementovať vyhľadávanie a nahradenie všetkých volaní metódy v zle navrhnutom systéme s miliónmi riadkov, kde sú všetky triedy prepojené, je preto veľmi náročná úloha, ktorá si vyžaduje sofistikované riešenie.

3. Jazyk JavaScript a DHTML

3.1 Čo je JavaScript

JavaScript je platformovo-nezávislý skriptovací jazyk s podporou objektového programovania. Kód jazyka je interpretovaný v prostredí jeho hostiteľskej aplikácie, kde môže manipulovať ako s objektami JavaScriptu, tak aj s objektami zverejnenými týmto hostiteľským prostredím (*host environment*).

História a štandardizácia

JavaScript bol vytvorený Brendanom Eichom z Netscape Communications pôvodne pod menom Mocha a následne LiveScript. Zmena názvu na „JavaScript“ súvisela s priblížením syntaxe k jazyku Java od Sun Microsystems. Prvý krát bol JavaScript štandardizovaný v rokoch 1997-1999 organizáciou ECMA pod menom ECMAScript. JavaScript je teda od verzie 1.5 štandardom ECMA-262 Edition 3 a tiež ISO štandardom. Špecifikáciu štandardu ECMA možno nájsť na stránke [ECMA99].

Interpretovaný jazyk

JavaScript je interpretovaný jazyk. Samotný kód jazyka nie je preto kompilovaný do binárnej formy, ale je priamo interpretovaný pri každom spustení programu. Na interpretáciu príkazov jazyka môžu byť použité rôzne interpretre – najznámejšími sú SpiderMonkey (pôvodný, implementovaný v jazyku C, Brendan Eich) a Rhino (implementovaný v jazyku Java, Norris Boyd).

JavaScript vs. Java

Premenovanie skriptovacieho jazyka na "JavaScript" spôsobilo viac zmätku ako osohu. Častokrát sa totiž JavaScript podľa názvu nesprávne považuje za interpretovanú verziu jazyka Java od Sun Microsystems. V skutočnosti sa však podobnosť jazykov viaže skoro výlučne len na ich syntax. Konštrukcie jazyka JavaScript sú totiž, z dôvodu ľahkej pochopiteľnosti, založené na syntaxe jazykov Java a C. Rozdiely medzi jazykmi, ktorých je však podstatne viac, sú nasledovné :

- **Interpretovaný vs. kompilovaný.** JavaScript je interpretovaný skriptovací jazyk, kdežto Java je plný programovací jazyk, ktorého programy sú kompilované do medzikódu (tzv. bytecode).

- **Objektové modely.** Objektový model JavaScriptu je prototypový. Nerozlišuje medzi typmi objektov a nepozná pojem trieda. Umožňuje dynamické pridávanie atribútov a metód objektom, prípadne zmenu dedičnosti počas behu programu. Objekty sú v JavaScripte konštruované za behu. Java je klasický objektovo-orientovaný jazyk, kde sú objekty rozdelené do tried s pevnou hierarchiou dedičnosti. Objekty v Jave sú konštruované pri kompilácií.
- **Typovanie.** Dátové typy premenných sa v JavaScript nedeklarujú – sú implicitne priradené, čo robí z JavaScriptu jazyk so slabým typovaním. Java je naproti tomu jazyk so silným typovaním, kde všetky premenné musia mať svoj explicitne deklarovaný dátový typ.
- **Viazanie.** JavaScript má dynamické viazanie, pri ktorom sa referencie objektov kontrolujú počas behu programu (v run-time). Java má statické viazanie, kde referencie objektov musia existovať už v čase kompilácie programu.
- **Prostredie.** Pre JavaScript ako skriptovací jazyk je nutná prítomnosť hostiteľského prostredia. Funkčnosť a možnosti programov sú pritom obmedzená na toto prostredie (čo je pre väčšinu prostredí ako napríklad webový prehliadač výhodou). Programy v Jave sú samostatne bežiacim kódom bez obmedzení. Výnimkou sú len Java applety bežiace v rámci webového prehliadača (avšak bez užšieho prepojenia s jeho prostredím).

Objektový model

JavaScript využíva prototypovo-orientovaný objektový model. Na rozdiel od klasického objektového prístupu JavaScript nerozoznáva medzi triedami a ich inštanciami, má jednoducho len objekty. Počiatočné atribúty a metódy objektu definuje jeho prototyp. Objekt môže počas behu meniť a pridávať svoje vlastné atribúty a metódy a dokonca zmeniť svoj prototyp. Dedičnosť v prototypovo-založenom programovaní je dosiahnutá klonovaním existujúcich objektov, ktoré slúžia ako prototyp pre nové objekty. Tento prístup má za cieľ lepšie simulovať fungovanie skutočného sveta. Možnosť pridávania nových atribútov a metód objektom a zmena dedičnosti sú jeho veľkou výhodou. Viac o objektovom modeli JavaScriptu aj s porovnaním s klasickým možno nájsť v samostatnej časti, poprípade detailnejšie v [KITOBJ97]. Objektový model JavaScriptu popisujú aj [RFB01] [YTS01], a [JSK04]. Implementáciou klasického spôsobu objektovo-orientovaného programovania (OOP) v JavaScripte sa zaoberajú [MCK03], [KLD03] a [TSC04].

Hostiteľské prostredie

Ako už plynie so samotného názvu, JavaScript je skriptovací jazyk. Je začlenený do hostiteľského prostredia, ktorým môže byť ľubovoľná aplikácia, schopná pomocou interpretra interpretovať syntax JavaScriptu na základe

jeho špecifikácie ([ECMA99]). Špecifikácia myslí na začlenenie do hostiteľského prostredia a definuje objektový model jazyka ako súčet vstavaných objektov, objektov definovaných užívateľom a čo je dôležité, objektov hostiteľského prostredia. Ponecháva pritom voľnosť v tom, ktoré objekty a akú ich funkcionalitu hosťovské prostredie sprístupní prostredníctvom JavaScriptu.

Napojiteľnosť JavaScriptu na ľubovoľné hosťovské prostredie spĺňajúce špecifikáciu a možnosť manipulácie s objektami tohto prostredia pomocou JavaScriptu sú dôvodom obľúbenosti a rozšírenia tohto jazyka. Hlavnou oblasťou použitia JavaScriptu sa stalo prostredie Internetu, či už na klientskej alebo serverovej strane.

3.2 Syntax jazyka JavaScript

Z hľadiska refaktorovania je syntax a sémantika refaktorovaného jazyka veľmi podstatná. Je teda dôležité oboznámiť sa s možnosťami a špecifickými vlastnosťami syntaxe jazyka JavaScript.

Táto časť sa zameria hlavne na špecifiká skriptovacieho jazyka JavaScript. Plná definícia syntaxe jazyka presahuje rozmer tejto práce. Podrobnú dokumentáciu ku syntaxe JavaScriptu možno nájsť v jeho ECMA špecifikácii [ECMA99], knihe [FLAN98] alebo na stránke [KITJS97].

Príklad kódu JavaScript programu

Nasledujúci program ukazuje základné vlastnosti JavaScriptu – deklaráciu globálnej aj lokálnej premennej, deklaráciu a volanie funkcie s argumentmi a návratovou hodnotou, výrazy, priradenie, cyklus, komentár a výpis správy na obrazovku.

```
var name = "Hello";

function writeHello(text1, text2) {
	var name;
	name = text1 + text2;
	for( var i=0; i<10; i++ ) { // vypíš 10 krát
		alert(name);
	}
	return name;
}

writeHello(name," world!\n");
```

Špecifiká jazyka JavaScript

Dátové typy

JavaScript vnútorne rozoznáva nasledovné typy hodnôt :

- Číselné hodnoty – ako napr. `35` či `2.75`
- Logické hodnoty – `true` a `false`
- Textové reťazce – ako napr. `„auto"`
- `null` – špeciálne kľúčové slovo pre nulovú hodnotu objektov
- `undefined` – hodnota neinicializovanej premennej
- objekty a funkcie

Podľa horeuvedeného príkladu kódu je vidieť slabé typovanie jazyka JavaScript. V samotnom kóde sa pri deklarácií premennej nedefinuje jej dátový typ a neprebieha ani jeho kontrola pred spustením programu. Tento prístup umožňuje priradiť premennej hodnoty ľubovoľného dátového typu za sebou. Nasledujúci kód nevyhlási chybu a vypíše text `"Hello world"` :

```
var premenna = 35;
premenna = "Hello world";
alert(premenna);
```

Aktuálny typ premennej možno zistiť pomocou vstavanej funkcie `typeof`.

```
var premenna = 35;
alert( typeof(premenna) ); // vypíše text „number"
premenna = "Hello world";
alert( typeof(premenna) ); // vypíše text „string"
```

JavaScript vykonáva automatickú konverziu dátových typov. Vo výrazoch obsahujúcich číselné aj textové hodnoty napríklad automaticky konvertuje číselné hodnoty do textových. Programový kód

```
var x = 35;
alert( x + "rokov" );
alert( "rokov : " + x );
```

preto vypíše text `„35 rokov"` resp. `„rokov : 35"`. V prípade, že by však premenná `x` nebola inicializovaná (, t.j. mala by hodnotu `undefined`), pri vykonávaní programu by došlo k chybe. Toto platí pre číselné a textové hodnoty, nie pre prvky polí, kde je vrátená hodnota `false`.

Identifikátory a deklarácia premenných

Podobne ako pri jazyku Java sú mená všetkých identifikátorov a kľúčových slov case-sensitive (,t.j. záleží na veľkosti písma). Syntax identifikátorov premenných, výrazov a operátorov je rovnaká.

Za zmienku stojí možnosť vynechania kľúčového slova `var` pri deklarácií premennej a tiež možnosť vynechania oddeľovača príkazov `";"`.

Nasledujúce tri riadky teda bez chyby rovnako deklarujú premennú a priradia jej počiatočnú hodnotu :

```
var cislo = 5;
var cislo = 5   // bez oddeľovača príkazov
cislo = 5;      // bez kľúčového slova 'var'
```

Výrazy a operátory

Pre výrazy a operátory platia rovnaké syntaktické pravidlá ako v jazyku Java. Je naviac možné použiť operátor `void` nad výrazom, ktorý nevracia hodnotu. Tento operátor je príjemné využiť pri spojení s HTML ako hodnotu atribútu hyperlinku. Napríklad odkaz `<a href="javascript:void(0)">Prázdny odkaz</a>` po kliknutí neotvorí žiadny iný dokument, t.j. neurobí nič.

JavaScript má veľmi dobrú podporu budovania regulárnych výrazov, ktoré je možné využiť na pattern-matching (porovnávanie hodnôt voči regulárnym výrazom). Regulárny výraz `/ab+c/` napríklad povoľuje textové reťazce `abc`, `abbc`, `abbbc` ...

Funkcie

Pri vykonávaní programového kódu hrajú funkcie a procedúry základnú úlohu v každom jazyku. V rámci JavaScriptu sú kľúčovými a tvoria základ prototypového objektového modelu – každý objekt je totiž definovaný funkciou svojho konštruktora.

```
function factorial(x) {
      if ( x <= 1 )
            return 1;
      else
            return x * factorial (x-1);
}
alert( factorial(5) ); // vypíše číslo 120
```

Na deklarovanie funkcie sa používa kľúčové slovo `function`, nasledované menom funkcie a definíciou vstupných parametrov. Špecifickou vlastnosťou jazyka je možnosť využitia premenlivého počtu vstupných parametrov pomocou poľa `arguments` v tele funkcie. Pri volaní funkcie sa toto pole naplní hodnotami toľkých parametrov, koľko ich bolo vo volaní zadaných. Následne je možné vo funkcií pristupovať k jednotlivým elementom tohto poľa a využiť ich pri výpočte. Ako príklad môže poslúžiť funkcia na vypočítanie maximálnej hodnoty ľubovoľného počtu parametrov.

```
function max() {
      var maximum;
      for( var i=0; i< max.arguments.length; i++ ) {
            if (i==0 || max.arguments[i] > maximum )
                  maximum = max.arguments[i];
      }
      return maximum;
}
```

```
alert( max(10,2,13,400,5) ); // vypíše číslo 400
alert( max("hello","world") ); // vypíše „world"
```

Horeuvedený príklad ukazuje možnosť využitia poľa `arguments` vo funkcií, čím sa abstrahuje od počtu vstupných parametrov a dokonca aj od ich dátového typu (funkciu možno použiť nad ľubovoľným typom, ktorý podporuje operátor porovnania „>").

Ak bolo pri volaní funkcie zadaných príliš málo vstupných parametrov, zvyšné parametre sú naplnené hodnotou `undefined`. Táto hodnota je tiež vrátená pri prístupe k prázdnemu nenaplnenému elementu poľa `arguments`.

Zaujímavosťou jazyka je tiež možnosť použitia funkcie ako dátového typu. Týmto spôsobom je možné priradiť funkciu ako element poľa či metódu objektu.

```
function square(x) { return x*x }

a = square; // priradí do premennej „a" funkciu „square"
b = a(5); // priradí do "b" hodnotu 25, t.j. "square(5)"

o = new Object; // vytvorí objekt na základe vstavanej funkcie „Object"
o.sq = square; // priradí funkciu ako hodnotu novej metódy objektu
y = o.sq(16);  // priradí do „y" hodnotu 256

a = new Array(10); // vytvorí objekt poľa o veľkosti 10 prvkov
a[0] = square; // priradí funkciu ako prvok poľa
a[1] = 20;
a[2] = a[0](a[1]); // priradí prvku a[2] hodnotu 400, t.j. „square(20)
```

Cykly a príkazy

Konštrukcie cyklov a syntax príkazov sú skoro úplne podobné jazyku Java.

Jednou zo zaujímavých konštrukcií je cyklus `for...in`, pomocou ktorého možno prechádzať po jednotlivých metódach a atribútoch objektu. Nasledovný príklad demonštruje použitie for...in na výpis hodnôt všetkých metód a atribútov objektu.

```
var text = „";
for ( prop in myObject ) {
      text = "meno: " + prop + "; hodnota: " + myObject[prop] + "\n";
}
alert( text );
```

Ďalšími špecifickými konštrukciami JavaScriptu sú príkaz `with` na nastavenie namespace podľa zadaného objektu, prázdny príkaz „;" a už spomenuté príkazy `var` a `function` na deklaráciu premennej resp. funkcie

```
with(Math) {
      x = sin( i * PI / 20 ); // namiesto "x = sin( i * Math.PI / 20 )"
      y = cos( i* PI / 30 );
}
```

Objekty

Objekty v jazyku JavaScript sú založené na prototypovom objektovom modeli. Kvôli odlišnosti od štandardného modelu objektov a tried je tento model a jeho syntax v JavaScripte podrobnejšie popísaný v nasledujúcej samostatnej kapitole.

3.3 Použitie jazyka JavaScript

Hostiteľské prostredie internetového prehliadača

Užívatelia Internetu sa najčastejšie stretnú s použitím JavaScriptu v hostiteľskom prostredí internetového prehliadača (ako je napríklad Microsoft Internet Explorer, Netscape Navigator, Mozilla či Opera). Internetové prehliadače predstavujú užívateľské rozhranie medzi internetovými servermi poskytujúcimi webové stránky so svojim obsahom a užívateľmi požadujúcimi tieto stránky.

HTML a klient/server komunikácia

Na komunikáciu medzi serverom a klientmi (internetovými prehliadačmi) sa využíva protokol HTTP (HyperText Transfer Protocol). Klient vyšle prostredníctvom protokolu HTTP požiadavku na webový dokument serveru. Server požiadavku spracuje a vráti klientovi odpoveď s príslušným dokumentom, alebo s informáciou o jeho nedostupnosti. Webovým dokumentom pritom môže byť súbor ľubovoľného formátu prezentujúci publikované informácie. Najčastejším formátom na publikovanie webových dokumentov je jazyk HTML (HyperText Markup Language). HTML dokument v sebe zahŕňa samotné informácie, ktoré sú publikované a spôsob akým sú prezentované, t.j. štruktúru a zobrazenie informácií (napr. odseky, farba a typ písma). Pre lepšie oddelenie obsahu od spôsobu jeho zobrazenia je možné využiť CSS (Cascading StyleSheets), ktoré umožňujú definovať štýly zobrazovania jednotlivých elementov a konštrukcií jazyka HTML.

HTML a Javascript

Samotný jazyk HTML poskytuje len statický spôsob prezentovania informácií bez možností manipulovať s HTML elementmi dokumentu. A práve túto úlohu dynamickej manipulácie s HTML dokumentom a prístupu k vlastnostiam jeho elementov plní JavaScript a iné skriptovacie jazyky, v hostiteľskom prostredí internetových prehliadačov. Špecifikácia HTML umožňuje začleniť do dokumentov programový kód skriptovacích jazykov, a teda aj JavaScriptu. Tento kód je interpretovaný hostiteľským prostredím internetového prehliadača pri zobrazovaní HTML stránky a jej obsahu a pri

spúšťaní udalostí nad elementmi stránky (ako napríklad kliknutie na tlačidlo).

Document Object Model

Ako už bolo uvedené, špecifikácia JavaScriptu umožňuje zverejniť hostiteľskému prostrediu jeho objekty a ich funkcie. V prostredí internetových prehliadačov tvorí štruktúra týchto hostiteľských objektov tzv. Document Object Model (DOM). Objektový model DOM je oficiálnym štandardom organizácie World Wide Web Consortium (W3C – [W3CDOM]). Je to platformovo- a jazykovo-nezávislé rozhranie umožňujúce programom a skriptom dynamicky pristupovať a upravovať obsah, štruktúru a štýl HTML dokumentov. Pozostáva z objektov, ktoré reprezentujú jednotlivé elementy HTML dokumentu (ako napríklad formuláre, hyperlinkové odkazy, text, frames či samotný dokument). Pomocou skriptovacích jazykov ako JavaScript je možné pristupovať k DOM a manipulovať s vlastnosťami a funkciami elementov dokumentu (napr. validovať správnosť hodnôt polí formuláru, skrývať a zobrazovať elementy stránky či meniť spôsob zobrazenia elementov).

Iné hostiteľské prostredia

Ďalším hostiteľským prostredím, v ktorom sa využíva JavaScript a jeho vlastnosti je ***hostiteľské prostredie internetového servera***. V tomto prípade nie je jazyk využívaný na manipuláciu s HTML dokumentom, ale na spracovanie požiadaviek od klientov a vygenerovanie príslušných odpovedí. Ako príklad možno uviesť využitie skriptovacieho jazyka JScript, ktorý je implementáciou štandardu ECMAScript z dielne Microsoftu, v technológii Active Server Pages (ASP), slúžiacej na dynamické generovanie webových stránok na strane servera.

Použitie skriptovacieho jazyka JavaScript sa teda neviaže výlučne na dobre známe klientské prostredie internetového prehliadača. Nasledujúca tabuľka zhŕňa najznámejšie hostiteľské prostredia JavaScriptu a jeho implementácií.

Hostiteľské prostredie	Skriptovací jazyk
Internetové prehliadače (MS IE, ...)	JavaScript
Active Server Pages (ASP)	JScript
Macromedia Flash	ActionScript
Scalable Vector Graphics (SVG)	ECMAScript alebo JavaScript
Adobe PDF	Acrobat JavaScript
...	

3.4 JavaScript a koncept DHTML

Hostiteľské prostredie internetových prehliadačov pôvodne slúžilo hlavne na jednoduchú prezentáciu HTML dokumentov, so štandardným zobrazením jednotlivých elementov ako nadpisy, odseky, či formuláre. Dodatočne bolo umožnené začleniť do HTML dokumentov štýly CSS na systematickejšiu tvorbu dizajnu a prezentácie stránky. Stále bol však HTML výlučne statického charakteru. Až po doplnení podpory skriptovacích jazykov ako JavaScript do hostiteľského prostredia je možné využívať programovacie funkcie skriptov na manipuláciu so statickými objektami a elementmi HTML dokumentu. Tým je možné zdynamizovať zobrazovanie a správanie elementov priamo alebo na základe udalostí (napr. stlačenie tlačidla). Koncept tejto dynamizácie HTML dokumentov je známy ako Dynamic HTML (DHTML). Pri DHTML sa teda nejedná o nový programovací jazyk, ale o koncept doplnenia možností dynamickej manipulácie s HTML dokumentmi a integráciu HTML s CSS a skriptovacím jazykom JavaScript.

Jazyk HTML

HyperText Markup Language (HTML), je najčastejšie používaným jazykom na publikovanie dokumentov na Internete. Dôvodom jeho popularity sú hlavne jednoduchosť a štandardizovanosť. Obsah v rámci celého HTML dokumentu je štruktúrovaný do hierarchického stromu tzv. *tagov* nazývaných aj *markup-y* či *HTML elementy*. Príklad :

```
<title>Nadpis</title>
```

Tagy reprezentujú výslednú pozíciu a spôsob zobrazenia jednotlivých častí textu v rámci dokumentu (ako napríklad zobrazenie textu `Nadpis` z predchádzajúceho príkladu v hlavičke okna prehliadača). Každý typ tagu má definovanú sadu svojich *atribútov*, ktoré bližšie definujú jeho správanie a zobrazenie.

```
<img src="obrazok.jpg" />
```

(tag `img` zodpovedajúci obrázku a atribút `src` jeho zdrojového súboru).

Rozsah pôsobnosti jednotlivých tagov HTML dokumentu je určený ich *ukončovacím tagom* (ako napr. `</title>` pre tag `<title>`), prípadne ukončením v rámci tagu samotného : `<img />`). Vnorením vnútorných tagov medzi dvojicu vonkajších tagov sa vytvára stromová hierarchia HTML dokumentu. Príklad :

```
<head>
      <title>Nadpis</title>
</head>
```

Príklad HTML dokumentu :

```
<html>
<head>
<title>Nadpis</title>
</head>
<body bgcolor="white">
      <b>Tučný text</b> a <i>kurzíva</i>.
      <img src="obrazok.jpg" width="30" />
      <ol>Očíslovaný zoznam
            <li>Prvá odrážka</li>
            <li>Druhá odrážka</li>
      </ol>
      <a href="www.w3c.org">Toto je hyperlink na stránku W3C</a>
</body>
</html>
```

Horeuvedený príklad využíva niektoré z tagov, ktoré definuje jazyk HTML na zobrazenie jednoduchej HTML stránky (dokumentu) s textom, obrázkom, zoznamom a hyperlinkom. Názvy tagov a ich ukončovacích tagov sú zvýraznené tučným písmom. Správny HTML dokument musí mať na vrchu hierarchie tagy `<html></html>`.

Každý HTML dokument v sebe okrem obsahu nesie aj informácie o spôsobe jeho zobrazenia. Toto zobrazenie sa však môže líšiť od jedného HTML prehliadača (browsera) k inému. Je na prehliadači, akým spôsobom, pri dodržaní špecifikácie, prezentuje obsah HTML dokumentu. Častokrát sú preto HTML dokumenty, využívajúce špeciálne tagy alebo striktne nedodržujúce špecifikáciu, zobrazované odlišne pod rôznymi prehliadačmi.

Document Type Definition (DTD)

Jazyk HTML je len jedným z veľkého množstva markup jazykov. Všetky tieto jazyky sú vytvorené pomocou metajazyka akými sú XML alebo SGML a využívajú vlastné sady tagov (alebo markup-ov). Na definovanie tejto sady a všetkých ich pravidiel daného markup jazyka slúži tzv. DTD (Document Type Definition; definícia typov dokumentu), ktorý by mal byť uvedený na začiatku každého dokumentu markup jazyka. Napr. pre HTML vyzerá uvedenie definície nasledovne :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Nasledujúci príklad ukazuje definíciu podmnožiny jazyka HTML pomocou DTD.

```
<!ELEMENT html (head,body)>
<!ELEMENT head (title)>
<!ELEMENT body (b*, img*, a*)>
      <!ATTLIST body bgcolor CDATA>
<!ELEMENT b (#PCDATA)>
<!ELEMENT img (#PCDATA)>
      <!ATTLIST img src CDATA>
<!ELEMENT a (#PCDATA)>
      <!ATTLIST a href CDATA #REQUIRED>
```

Tento príklad definície DTD určuje, že každý dokument musí začínať tagom `html`, ktorý bude obsahovať práve jeden vnorený tag `head` a jeden tag `body`. Tag body môže následne obsahovať jeden a viac tagov `b` (tučný text), `img` (obrázok) a `a` (hyperlink), pričom môže mať špecifikovanú hodnotu atribútu `bgcolor` (farba pozadia).

Samotný markup jazyk HTML môže byť definovaný viacerými DTD definíciami. Najpoužívanejšími sú ***Strict DTD***, definujúca obmedzenú množinu tagov a ***Loose DTD***, umožňujúca použitie väčšej sady tagov a atribútov, hlavne na prezentáciu a linkovanie.

Špecifikácia stromovej hierarchie HTML (t.j. možnosti povolených tagov a spôsobov vnárania) je definovaná štandardom SGML (Standard Generalized Markup Language), na ktorom je HTML založený. SGML, predchodca súčasne široko používaného štandardu XML (Extensible Markup Language), je metajazyk, ktorý umožňuje definovať pravidlá štruktúry markup jazykov - množinu tagov, ich atribútov, obsahu tagov a spôsoby vnárania jednotlivých typov tagov.

O štandardizáciu a ďalší vývoj jazyka HTML a príbuzných technológií sa stará World Wide Web Consortium (W3C). Podrobnú špecifikáciu a dokumentáciu k HTML, DTD a iným formátom možno nájsť na oficiálnej stránke [W3CHTML], v knihe [MUKE97] (prípadne neformálnejšie na stránke [SLF]). Súčasnou verziou HTML je verzia 4.01.

Cascading StyleSheets (CSS)

Jazyk HTML je určený hlavne na prenos obsahu dokumentov. Na základe štruktúrovanosti pomocou pevnej sady tagov sú prehliadače schopné zobrazovať hodnoty jednotlivých tagov očakávaným spôsobom (ako napríklad zobrazenie obsahu medzi tagmi `<b>` a `</b>` tučným písmom). Tieto základné možnosti zobrazovania je možné rozšíriť pomocou definovania tzv. Cascading StyleSheets (alebo CSS).

Integrácia s HTML

CSS umožňujú ku každému tagu alebo typu tagov definovať vlastnosti jeho štýlu ako farba, presná súradnicová pozícia v dokumente, formát textu, formát tabuliek či odsadenie. Štýly HTML elementov je možné definovať na troch úrovniach : na úrovni jednotlivých elementov, na úrovni celého dokumentu a na základe samostatného súboru.

Úroveň HTML elementu

Štýly CSS na úrovni HTML elementu sa definujú pomocou atribútu `style`, ktorý je spoločný pre všetky typy tagov. Ako príklad možno uviesť :

```
<p style="color: red">Červený odstavec. </p>
```

Úroveň celého dokumentu

Na úrovni celého dokumentu sa definujú štýly CSS v hlavičke HTML dokumentu dvomi možnými spôsobmi. Prvým je definovanie štýlu pre všetky použitia daného typu tagu rovnako (napr. pre všetky odseky, t.j. tagy `<p>` v dokumente).

```
<html>
  <head>
    <title> CSS definované pre typy tagov </title>
    <style>
      p { color : red; font-style : italic }
      body { background-color : yellow }
    </style>
  </head>
  <body>
    <p> Červený odsek kurzívou. </p>
  </body>
</html>
```

Druhým spôsobom je definovanie štýlu pomocou tried, na základe atribútu `class` prípadne `id` jednotlivých tagov.

```
<html>
  <head>
    <title> CSS definované pre triedy </title>
    <style>
      .podtitul { text-align : center; font-weight : bold }
    </style>
  </head>
  <body>
    <p class="podtitul"> Vycentrovaný tučný odstavec. </p>
  </body>
</html>
```

```
<html>
  <head>
    <title> CSS definované podľa ID </title>
    <style>
      #podtitul { text-align : center; font-weight : bold }
    </style>
  </head>
  <body>
    <p id="podtitul"> Vycentrovaný tučný odstavec. </p>
  </body>
</html>
```

Úroveň samostatného súboru

Poslednou úrovňou je definovanie štýlu CSS v samostatnom súbore. V tomto prípade sa premiestni definícia z hlavičky (v tagoch `<style></style>`) do súboru s príponou `css` a na miesto pôvodnej definície sa umiestni odkaz na príslušný súbor :

```
<link rel="stylesheet" type="text/css" href="styly.css" />.
```

Definovanie štýlov týmto spôsobom poskytuje možnosti použitia rovnakého vzhľadu pre všetky vytvorené HTML stránky jednoduchým spôsobom (pridaním odkazu) a ľahkú a rýchlu distribúciu štýlov medzi projektmi.

Pre každý HTML element je možné použiť celú paletu rôznych CSS štýlov. Treba však poznamenať, že zobrazenie všetkých štýlov nie je úplne štandardizované a je preto vhodné ho otestovať na rôznych typoch internetových prehliadačov. O štandardizáciu a vývoj formátu CSS sa podobne ako pri HTML stará W3C. Viac informácií o možnostiach štýlov možno nájsť na oficiálnej stránke [W3CCSS] a stránke [SLF].

3.5 Integrácia JavaScript kódu s HTML, DOM

Samotná špecifikácia ECMAScriptu a teda aj JavaScriptu ([ECMA99]) myslí na využitie jazyka v rámci hostiteľského prostredia. Definuje spôsob, akým môže hostiteľské prostredie zverejniť svoje objekty a ich funkcionalitu v rámci skriptovacieho jazyka.

Zverejnené hostiteľské objekty v prostredí internetového HTML prehliadača tvoria tzv. Document Object Model (DOM), štandardizovaný organizáciou W3C ([W3CDOM]).

Document Object Model

Document Object Model zverejňuje rôzne objekty HTML prehliadača – objekty práve načítaného HTML dokumentu a informácie o jeho zdroji, históriu dokumentov, informácie o type prehliadača či okno prehliadača.

Medzi hlavné objekty modelu DOM patrí napríklad nasledovné :

- `window` – okno prehliadača a jeho súčasti
- `document` – HTML dokument načítaný v okne prehliadača
- `forms` - pole, ktoré zodpovedá všetkým formulárom v rámci HTML dokumentu
- ...

Podrobný zoznam základných objektov modelu DOM spolu s príkladmi ich použitia je možné nájsť v prílohe B na konci tejto práce. Pre úplné informácie vyhľadajte [W3CDOM], [SLF] alebo [FLAN98].

Prístup k DOM z JavaScriptu

Ku všetkým objektom DOM je možné z JavaScriptu pristupovať rovnakým spôsobom ako k vstavaným a užívateľom definovaným objektom. Možno nastavovať a získavať hodnoty ich atribútov a volať ich funkcie.

Kvôli hierarchickej štruktúre DOM bývajú atribútmi objektov častokrát iné objekty (objekt `forms` je napríklad atribútom objektu `document`). V kóde programov je preto nutné túto hierarchiu dodržať. Výnimkou je najvrchnejší objekt hierarchie – objekt window, ktorý nie je nutné v kóde explicitne uvádzať. Príkaz `alert(„chyba")`; má teda rovnakú sémantiku ako príkaz `window.alert("chyba");`.

Integrácia JavaScriptu a HTML

JavaScript manipuluje s HTML dokumentom a jeho elementmi prostredníctvom objektov DOM modelu. Pomocou objektu `document` a jeho podobjektov je možné pristupovať ku každému elementu a jeho atribútom, pridávať nové elementy či odstraňovať existujúce.

Na prístup k HTML elementom je možné využiť funkcie `getElementById`, `getElementsByName` alebo `getElementsByTagName`, ktoré priamo vyhľadajú požadovaný element. Alternatívne možno prechádzať postupne stromom dokumentu po úrovniach až po cieľový element (napr. výraz `document.forms[0].elements[0].value` získa hodnotu prvého elementu v rámci prvého formulára dokumentu).

Včlenenie JavaScript kódu do HTML

Na začlenenie príkazov skriptovacích jazykov do HTML dokumentov slúži špeciálny HTML element `<script></script>`. Všetok kód napísaný v rámci tohto elementu je prenechaný interpretru príslušného skriptovacieho jazyka a nie je považovaný za kód štruktúrovaný pomocou HTML. Na definovanie použitého skriptovacieho jazyka sa v elemente `script` používajú jeho atribúty `type` alebo `language`. Pri jazyku JavaScript je kód obsiahnutý v nasledovnej dvojici tagov :

```
<script type="text/javascript" language="JavaScript"></script>.
```

Príkazy v elemente `script` sú vykonávané pri načítavaní HTML stránky, hneď po zobrazení HTML elementov pred nimi. Bloky kódu môžu byť pritom umiestnené ako v hlavičke, tak aj tele HTML dokumentu. Pri deklarácií funkcií a premenných sa odporúča ich umiestnenie do hlavičky dokumentu. Tým sa zamedzí prerušovanému zobrazovaniu stránky a naviac docieli, že pri volaniach budú funkcie a premenné už definované.

JavaScript súbor

Podobne ako pri štýloch CSS je možné umiestniť JavaScript kód do samostatného súboru s príponou „js". Výhodou samostatného súboru je jeho ľahká prenositeľnosť a začleniteľnosť do existujúcich stránok, čím sa naplno využijú možnosti reuse existujúceho kódu a oddelenie JavaScript kódu od

kódu HTML a CSS. Na začlenenie súboru do HTML dokumentu stačí použiť element `script` s atribútom `src` – cestou k JS súboru.

```
<script type="text/javascript" src="subor.js">
</script>
```

Udalosti elementov

Umiestnenie blokov príkazov skriptovacieho jazyka medzi elementy HTML je len časťou samotnej integrácie a konceptu DHTML. Každý typ elementu má sadu udalostí, ktoré nad ním môžu byť vyvolané (ako napríklad kliknutie myšou na tlačidlo, prechod myšou nad obrázkom ...) a na ktoré je možné reagovať. A práve skriptovacie jazyky môžu byť využité ako nástroj na reakciu na tieto podnety. Tým sa dosiahne plné využitie možností HTML.

```
<input type="text" value="Textové pole" onClick="this.value='';" />
```

Horeuvedený príklad ukazuje možnosť reakcie na kliknutie myšou nad textovým poľom formuláru, pomocou atribútu `onClick`, v tomto prípade vymazaním obsahu textového poľa po kliknutí. Všetky atribúty HTML elementov, začínajúce predponou „`on`“ zodpovedajú udalostiam nad daným typom elementu. Hodnotou týchto atribútov je príkaz alebo postupnosť príkazov skriptovacieho jazyka, ktorá sa vykoná pri vyvolaní udalosti.

Možnosť reagovať na udalosti nad HTML elementmi poskytuje možnosti interaktívnej komunikácie s užívateľom, čo prispieva k dynamizácií obsahu a prezentácie HTML dokumentov. Ako príklady využitia udalostí možno uviesť validáciu správnosti hodnôt polí formuláru pri stlačení tlačidla na odoslanie formuláru, zmenu zobrazenia pri kliknutí alebo prechode nad elementom či výpočet komplikovaných funkcií na základe hodnôt elementov.

3.6 Objektový model jazyka JavaScript

Objekty v skriptovacom jazyku JavaScript sú rozdelené do troch typov : vstavané objekty jazyka, objekty definované užívateľom a objekty zverejnené hostiteľský prostredím. Tieto typy objektov sa líšia svojim pôvodom, ich charakter a správanie sú však postavené na rovnakých princípoch – na prototypovom objektovom modeli.

Prototypový objektový model

Len objekty

Objektový model JavaScriptu je založený na prototypoch (prototype-based, prototype-oriented object model). Na rozdiel od klasického objektovo-orientovaného prístupu JavaScript nerozoznáva medzi pojmami objekt a trieda. V prototypovom modeli existujú len objekty so svojimi atribútmi (attributes, members, fields) a svojimi metódami (methods).

Nový objekt sa následne vytvára na základe iného, už existujúceho objektu, ktorý týmto spôsobom tvorí prototyp (alebo šablónu) nového objektu.

Dedičnosť

JavaScript a jeho syntax nepozná pojem dedičnosť tak ako je definovaný v klasických objektovo-orientovaných jazykoch ako Java či C++. Dedičnosť je však jednou zo základných vlastností OO jazykov a i keď nie je možné v JavaScripte explicitne deklarovať objekt ako podobjekt iného objektu, dá sa dedičnosť v tomto jazyku ľahko simulovať. Na implementáciu dedičnosti stačí využiť vlastnosti prototypov.

Ako už bolo uvedené, nový objekt sa vytvára na základe iného objektu – svojho prototypu. Vytvorený objekt pritom prevezme všetky vlastnosti (properties – t.j. atribúty a metódy) svojho prototypu, teda dedí všetky jeho funkcie. Nový objekt môže následne vytvoriť vlastnú implementáciu svojich metód (metódou tzv. override), rôznu od pôvodnej implementácie v prototype (napr. zmeniť výpočet v tele jednej z metód).

Nový objekt nededí vlastnosti svojho prototypu priamo, má len informáciu o tom, ktorý objekt je jeho prototypom. Počas vykonávania programu, v ktorom sa odkazuje na hodnotu atribútu alebo metódy objektu, sa potom najprv prezrie, či má objekt podľa definície príslušný atribút resp. metódu. Ak je vlastnosť v objekte definovaná, prevezme sa jej hodnota. Inak sa zisťuje prítomnosť vlastnosti v prototype tohto objektu, následne v prototype prototypu, atď., až kým sa vlastnosť nenájde alebo kým sa nedosiahne objekt bez prototypu. Hierarchický systém prototypov týmto vytvára tzv. prototypovú reťaz (prototype chain).

Zmena dedičnosti a štruktúry objektov

V klasických objektovo-orientovaných jazykoch je štruktúra všetkých objektov pevná. Vlastnosti objektov sú fixne definované na základe tried pri kompilácií a počas behu samotného programu nie je možné ich štruktúru meniť. Prototypový model objektov zmeny štruktúry povoľuje, naviac umožňuje počas behu dokonca meniť prototypy objektov.

Každý objekt je vytváraný a definovaný na základe funkcie ***konštruktora***, ktorý určuje počiatočné vlastnosti objektu, vrátane prototypu. Syntax JavaScriptu poskytuje možnosti pridania nových a modifikácie či odstránenia existujúcich vlastností objektov mimo funkcie konštruktora – v samotnom kóde programu. Štruktúra a vlastnosti objektov sú v prototypovom modeli uchovávané v samostatných blokoch pamäti, ktorý je možné modifikovať.

Polymorfizmus

Dôležitou vlastnosťou objektovo-orientovaných jazykov je polymorfizmus, ktorý umožňuje využitie spoločných vlastností objektov,

definovaných na základe rovnakej triedy alebo nadtriedy. Do funkcie, metódy alebo atribútu využívajúceho tieto spoločné vlastnosti možno priradiť a použiť ľubovoľný objekt, ktorý ich implementuje.

Vďaka slabému typovaniu je možné v JavaScripte naplno využiť vlastnosti polymorfizmu. Pri použití vlastnosti objektu sa pritom berie ohľad len na to, či daný objekt alebo jeho prototyp príslušnú vlastnosť implementuje, alebo nie. Je teda možné použiť ľubovoľné objekty, ktoré túto vlastnosť majú a pritom nemusia mať nič iné spoločné.

Enkapsulácia

Princíp zapúzdrenia, alebo enkapsulácie objektovo-orientovaných jazykov zabezpečuje začlenenie a skrytie funkcionality v rámci objektov. Na podporu enkapsulácie sa využívajú možnosti nastavenia viditeľnosti vlastností objektov. Vlastnosť môže byť takto definovaná ako privátna - prístupná len pre ostatné vlastnosti objektu či verejná – prístupná aj pre všetky iné objekty.

Syntax JavaScriptu neposkytuje priamu podporu nastavenia viditeľnosti objektov a ich vlastností, je ju však možné nasimulovať pre tie vlastnosti objektu, ktoré sú definované konštruktorom, t.j. sú viazané staticky.

Syntax objektov v JavaScripte

Konštruktor

Každý objekt je v JavaScripte definovaný a vytváraný svojou funkciou konštruktora. Táto funkcia sa od ostatných líši jedine použitím kľúčového slova `this`, ktorým sa odkazuje na vytváraný objekt a pomocou ktorého definuje počiatočné vlastnosti objektu.

```
function Manazer( meno ) {
	this.meno = meno;
	this.vypisMeno = function() { alert( this.meno ); };
}
```

Konštruktor objektu `Manazer` z príkladu definuje objekt s atribútom `meno`, ktorý prevezme hodnotu zo vstupného parametra nastaveného pri vytváraní objektu a metódou `vypisMeno`, definovanej pomocou anonymnej funkcie a vypisujúcej hodnotu atribútu `meno` na obrazovku.

Vytváranie objektov

Objekty sú vytvorené použitím kľúčového slova `new` a funkcie konštruktora tohto objektu, s prípadnými parametrami.

```
var peter = new Manazer( "Peter");
```

Prístup k vlastnostiam objektov

Po vytvorení objektu je možné priamo pristupovať k jeho atribútom a metódam a využívať ich.

```
peter.vypisMeno(); // vypíše "Peter"
var meno2 = "Pán " + peter.meno;
```

Zmena štruktúry objektov (dynamické viazanie)

Konštruktor definuje počiatočnú štruktúru atribútov a metód objektu. Tú je však možné vďaka prototypovému modelu modifikovať počas behu programu.

Zmena atribútu a metódy objektu :

```
peter.meno = "Peťo";
peter.vypisMeno = function(titul) { alert( titul + this.meno ); };
```

Pridanie nového atribútu a metódy sa vykoná jednoduchým priradením :

```
peter.vek = 23;
peter.vypisVek = function() { alert( this.vek ) };
// na tomto mieste ma objekt v premennej "peter" 4 vlastnosti :
//    atribúty „meno", „vek" a metódy „vypisMeno" a „vypisVek"
```

Odstránenie existujúceho atribútu alebo metódy možno vykonať pomocou kľúčového slova `delete` :

```
delete peter.vypisVek
// na tomto mieste ma objekt v premennej "peter" 3 vlastnosti :
//    atribúty „meno", „vek" a metódu „vypisMeno"
```

Kľúčové slovo `delete` je možné tiež použiť na odstránenie celého objektu zadaním jeho mena, prípadne na odstránenie elementov polí (keďže aj polia sú implementované pomocou objektov).

Zaujímavosťou JavaScriptu je možnosť aplikácie dynamického viazania na všetky objekty a teda aj na vstavané. Takto možno pridať vlastné metódy a atribúty k objektom ako napríklad `String`.

```
var text = new String( "Ovca" );
text.mojaFunkcia = function () { alert( "beee" ); }
```

Prototyp

Keďže prototyp je vlastnosť objektu, jeho definícia je úplne rovnaká ako definovanie ostatných vlastností objektov. Na definovanie prototypu sa využíva kľúčové slovo `prototype`, ktoré sa použije ako názov vlastnosti objektu. Ako hodnota sa pritom očakáva objekt.

```
function Zamestnanec ( cislo ) { // konštruktor prototypu
      this.cislo = cislo
}
peter.prototype = new Zamestnanec ( 1 );
```

Pri prístupe k vlastnosti, ktorá nie je v definícii objektu sa postupuje po prototypovej reťazi až kým sa vlastnosť nenájde. Ak sa postúpi až k objektu, ktorý nemá prototyp, bez nájdenia vlastnosti, JavaScript vyhlási chybu.

```
alert(peter.cislo);  // vypíše hodnotu 1
peter.prototype = new String("aaa");
alert(peter.cislo);  // vyhlási chybu – "Manazer" ani "String" nemajú
                     // vlastnosť "cislo"
```

Okrem zmeny prototypu za behu programu je možné podobne ako pri iných objektoch meniť vlastnosti objektu prototypu :

```
peter.prototype = new Zamestnanec ( 1 );
peter.prototype.cislo = 2;
```

Typy objektov

Z hľadiska syntaxe nie je rozdiel medzi vstavanými objektami, objektami definovanými užívateľom a hostiteľským prostredím.

Vstavané objekty

Podrobný popis vstavaných objektov možno nájsť v špecifikácii [ECMA99].

Medzi vstavané objekty patria :

Vstavaný objekt	Popis
Array	pole a jeho elementy
Boolean	reprezentuje logickú hodnotu
Date	objekt dátumu a času
Function	definícia funkcie s parametrami a telom
Math	objekt obsahujúci matematické funkcie a konštanty
Number	reprezentuje číselnú hodnotu
Object	najvrchnejší objekt hierarchie, je prototypom všetkých objektov
RegExp	slúži na tvorbu regulárnych výrazov
String	textový reťazec a funkcie nad ním

Užívateľom definované objekty

Medzi kategóriu užívateľom definovaných objektov patria všetky objekty, ktoré sú definované funkciami konštruktora v kóde programov.

Objekty hostiteľského prostredia

Hostiteľské prostredie môže zverejniť svoje objekty v JavaScripte. Z hľadiska vývoja sa správajú podobne ako vstavané objekty. Na rozdiel od vstavaných objektov sa však líšia od jedného hostiteľského prostredia k druhému.

V hostiteľskom prostredí internetového prehliadača sú objekty zverejnené prostredníctvom objektového modelu DOM (Document Object Model).

4. Refaktorovanie jazyka JavaScript

Táto kapitola predstavuje jadro celej práce. Koncentrujú sa v nej všetky výsledky práce autora pri refaktorovaní nad jazykom JavaScript a nad DHTML v hostiteľskom prostredí internetového prehliadača. Keďže sa jedná o prvú prácu venovanú refaktorovaniu jazyka JavaScript, je vytvorený katalóg nových refaktorovaní.

Refaktorovanie v prostredí JavaScriptu nie je vôbec jednoduché. Nestačí len zobrať všeobecné vedomosti o refaktorovaní nad jazykmi ako sú Java, C++ alebo Smalltalk a priamočiaro ich aplikovať na JavaScript. Refaktorovanie totiž úzko súvisí s konkrétnym programovacím jazykom, na ktorom je závislé. Čo sa dá použiť v jednom jazyku nemusí byť možné v inom. Ako príklad možno uviesť refaktorovania v OO jazykoch ako je Java súvisiace s enkapsuláciou (napr. skrytie metódy), ktoré sa v JavaScripte nedajú použiť priamym spôsobom, kvôli chýbajúcim syntaktickým konštrukciám a preto strácajú praktický význam. Na druhú stranu prináša JavaScript, vďaka svojim špecifickým vlastnostiam, úplne nové možnosti refaktorovania. Sem patria refaktorovania súvisiace s prototypovým objektovým modelom jazyka či s integráciou s HTML. Refaktorovanie nad jazykom JavaScript má teda svoje jedinečné vlastnosti, prínosy a aj problémy.

Štruktúra kapitoly je rozdelená na dve časti. Prvá sa zameriava na existujúce refaktorovania objektovo-orientovaných jazykov a analyzuje možnosti ich použitia v prostredí jazyka JavaScript. Skúma význam ich použitia v rámci JavaScriptu a odlišnosti aplikácie v porovnaní s pôvodnými jazykmi týchto refaktorovaní. Týmto prístupom poskytuje štartovný bod pre možnosť rozšírenia existujúcich refaktorovacích nástrojov na jazyk JavaScript bez nutnosti veľkých modifikácií.

Druhá a tretia časť kapitoly berú do úvahy špecifické vlastnosti JavaScriptu, akými sú prototypový objektový model, začlenenie v hostiteľskom prostredí, slabé typovanie či voľnosť syntaxe premenných a príkazov. Tieto vlastnosti sú analyzované a na ich základe sú navrhnuté refaktorovania, ktoré môžu významným spôsobom prispieť k lepšej štruktúre či prehľadnosti kódu.

Možnosti, ktoré sú v tejto kapitole preskúmané a navrhnuté v žiadnom prípade neuzatvárajú oblasť refaktorovania JavaScriptu. Majú slúžiť hlavne ako počiatok výskumu v tejto oblasti, ktorého výsledkom a úspešným koncom by mala byť implementácia softvérového nástroja, podporujúceho čo najväčšiu sadu užitočných refaktorovaní jazyka JavaScript.

4.1 Aplikácia princípov refaktorovania na JavaScript

Táto časť analyzuje možnosti aplikovateľnosti všeobecných refaktorovaní objektovo-orientovaných jazykov na jazyk JavaScript tak, ako boli definované Martinov Fowlerom a zhrnuté do katalógu refaktorovaní ([FOW99] a stránka [FOWW]). Všetky refaktorovania, sú preskúmané s ohľadom na vlastnosti JavaScriptu z hľadiska ich významu a aplikovateľnosti, doplnené o príklad ich použitia v prostredí jazyka JavaScript.

Javascript je objektovo-orientovaný jazyk, ktorého syntax je prevzatá z jazykov C++ a Java, vrátane konštrukcií cyklov a podmienok. Na základe týchto spoločných vlastností s inými OO jazykmi, na ktorých sú už refaktorovania skúmané, je možné veľkú väčšinu princípov refaktorovaní použiť aj na JavaScript a to už priamočiaro alebo len s nutnosťou malej modifikácie. JavaScript má však aj množstvo špecifických vlastností, ktoré neumožňujú použitie niektorých všeobecných refaktorovaní, alebo ktorých použitie na JavaScript nemá význam z praktického hľadiska. Všetky tieto vlastnosti sú identifikované a popísané.

Refaktorovanie a OO jazyky

Refaktorovanie sa dostalo do popredia práve v prostredí objektovo-orientovaných jazykov. Je teda vhodné analyzovať možnosti refaktorovania z hľadiska štruktúr a pojmov objektovo-orientovaného programovania.

Základným prvkom OO jazykov je objekt, ktorý má svoje metódy a atribúty (vlastnosti, properties alebo members). Každý objekt je spoločne s inými objektami rovnakých vlastností kategorizovaný do tried. V rámci JavaScriptu je možné okrem metód a atribútov objektov využívať aj funkcie a premenné, ktoré majú analogický význam, nie sú však viazané na žiadny objekt či triedu. Funkcie a metódy vykonávajú určenú funkcionalitu prostredníctvom postupnosti príkazov, premenné a atribúty uchovávajú hodnoty výrazov.

Refaktorovanie OO jazykov slúži na zlepšenie štruktúry stavebných prvkov daného jazyka. Tieto prvky možno rozdeliť do kategórií podľa ich úrovne nasledovne :

- Triedy a objekty
- Metódy tried/objektov a funkcie
- Atribúty tried/objektov a premenné
- Príkazy jazyka
- Výrazy jazyka

Na základe tohto rozdelenia možno definovať refaktorovania pre jednotlivé úrovne. Každá úroveň má svoje špecifické vlastnosti, ktoré sa

môžu pre jednotlivé jazyky líšiť. Refaktorovania musia pritom tieto špecifiká zohľadniť. Z hľadiska JavaScriptu je možné refaktorovania na niektorých úrovniach aplikovať priamočiaro (ako napríklad funkcie či premenné), pri iných je nutné analyzovať spôsob akým sa dajú použiť (ako napríklad triedy a objekty z dôvodu odlišnej implementácie objektového modelu v JS).

Úroveň metód objektov a funkcií

Z hľadiska funkcionality možno považovať metódy objektov a funkcie za totožné. Jediným rozdielom je príslušnosť metódy ku konkrétnemu objektu/triede na rozdiel od funkcie, ktorá je všeobecná. Túto odlišnosť, spôsobenú enkapsuláciou metódy objektu, je nutné zohľadňovať v príslušných refaktorovaniach. V metódach sa totiž môžu vyskytovať referencie k iným vlastnostiam objektu (pomocou referencie `this`) , ktoré by z funkcie neboli priamo prístupné.

Syntax a sémantika funkcií v JavaScripte je rovnaká ako v iných OO programovacích jazykoch a preto je možné použiť refaktorovania priamočiaro. Pri metódach je už aplikovateľnosť refaktorovaní o čosi zložitejšia. Dôvodom je odlišný spôsob implementácie objektového modelu v JavaScripte, ktorý je založený na prototypoch.

Príkladom rozdielov v aplikácií refaktorovaní na úrovni metód je viditeľnosť. JavaScript explicitne nedefinuje viditeľnosť metód a atribútov objektov a pre vlastnosti, ktoré boli dynamicky pridané k objektu počas behu programu viditeľnosti ani nie je možné nastaviť. Z toho dôvodu je možnosť použitia refaktorovaní manipulujúcich s viditeľnosťou, ako napríklad skrytie metódy, obmedzená.

Možnosť dynamického pridania vlastností k objektom za behu je špecifická pre jazyky s prototypovým objektovým modelom. Je preto vhodné osobitne zvážiť možnosti refaktorovania týchto vlastností. Táto časť sa však venuje len množine všeobecných refaktorovaní OO jazykov a ich aplikovateľnosti na JavaScript a špecifické možnosti refaktorovania JavaScriptu necháva na samostatnú časť.

Refaktorovanie : Premenovanie metódy ("Rename Method")

Kategória : funkcia, metóda objektu

Popis :

Zmena názvu funkcie alebo metódy objektu.

Motivácia :

Ak pomenovanie funkcie/metódy nedostatočne označuje jej význam a použitie, je vhodné premenovaním zvoliť názov, ktorý túto funkciu/metódu lepšie špecifikuje.

Aplikácia na JavaScript :

Voľba nevhodného, alebo príliš všeobecného názvu metódy je častý problém vo všetkých programovacích jazykoch. Pri slabo-typovaných jazykoch ako JavaScript má však táto voľba ešte väčší dopad na zrozumiteľnosť kódu. Pritom práve názov metódy je pri týchto jazykoch možné využiť ako nositeľa informácie o type návratovej hodnoty či type vstupných hodnôt.

Príklad aplikácie v JS :

```
function append() {
	var s = "";
	for( var i=0; i<arguments.length; i++ )
		s += arguments[i] + " ";
	return s;
}
```

⇓

```
function concatenateInputStrings() {
	var s = "";
	for( var i=0; i<arguments.length; i++ )
		s += arguments[i] + " ";
	return s;
}
```

Vďaka premenovaniu metódy na vhodnejší názov `concatenateInputStrings`, (t.j. "zreťazenie vstupných textových reťazcov) vedia vývojári lepšie posúdiť, či je použitie tejto metódy v ich prípade vhodné. Premenovaním v tomto príklade sa pomôže zabrániť použitiu metódy na sčítanie vstupných číselných parametrov. Napríklad pri volaní `append(1,10,5,3)` by miesto očakávanej výslednej hodnoty `20` bol výsledkom reťazec typu `„1 10 5 3"`.

Refaktorovanie : Vloženie tela metódy („Inline Method")

Kategória : funkcia, metóda objektu

Popis :

Dosadenie tela funkcie/metódy namiesto jej volaní s príslušnými hodnotami parametrov a následné odstránenie tejto metódy.

Motivácia :

V prípade, že je telo funkcie/metódy z hľadiska pochopenia a zložitosti na rovnakej úrovni ako jej meno, stráca extrakcia kódu do samostatnej funkcie/metódy význam a možno jej telo priamo dosadiť na miesta, kde je volaná.

Aplikácia na JavaScript :

Použitie vloženia tela metódy má v JavaScripte veľký význam hlavne pri prepojení s jazykom HTML v hostiteľskom prostredí prehliadača. Ak funkcia prestáva plniť svoj význam a nie je často volaná, je vhodnejšie ju začleniť priamo do udalostí HTML elementov (viď. príklad aplikácie). Viac o

použití tohto refaktorovania pri integrácií s HTML možno nájsť v časti o spojení JS-HTML-CSS tejto kapitoly.

Keďže syntax funkcií/metód je v JavaScripte podobný ako pri iných programovacích jazykoch, je možné použiť toto refaktorovanie spôsobom, aký je definovaný vo Fowlerovom katalógu ([FOW99]) . Podobne ako v iných OO jazykoch je pri aplikácií refaktorovania na metódy nutné nahradiť všetky výskyty kľúčového slova `this`, reprezentujúceho objekt metódy, samotnou referenciou objektu.

Príklad aplikácie v JS :

```
function getMessage() {
      alert("Hello."); // vypíše text do dialógu na obrazovke
}
...
<input type="button" value="" onClick="getMessage();">
```

⇓

```
<input type="button" value="" onClick="alert("Hello.");">
```

Kód sa po aplikácií zjednoduší a sprehľadní. Nie je nutné vyhľadávať funkciu, ktorá je volaná a ktorá sa prípadne nachádza v samostatnom JavaScript súbore alebo inom HTML frame. Na druhú stranu však dôjde k pevnejšiemu zviazaniu HTML a JavaScript kódu, čo zamedzí prípadnému použitiu funkcie v iných častiach kódu. V týchto prípadoch je vhodné použiť inverzné refaktorovanie – extrakciu metódy.

Refaktorovanie : Extrakcia metódy ("Extract Method")

Kategória : funkcia, metóda objektu

Popis :

Extrakcia fragmentu zdrojového kódu (prípadne viacerých navzájom podobných fragmentov) do funkcie/metódy a jej pomenovanie podľa jej účelu.

Motivácia :

V prípade, že príslušný fragment vykonáva istú dobre definovateľnú funkcionalitu, prispeje toto refaktorovanie k lepšej štruktúrovanosti kódu. Naviac je možné extrahovať viaceré fragmenty podobného kódu (často s nutnosťou parametrizácie funkcie/metódy), čím sa kód zjednoduší a je možné ho opätovne použiť pri riešení podobnej úlohy.

Aplikácia na JavaScript :

Podobne ako inverzné refaktorovanie „inline method" má extrakcia kódu do metódy v JavaScripte veľký význam. Okrem použitia v rámci kódu JavaScriptu samotného, je možné extrakciu použiť aj pri integrovaní JS kódu s HTML kódom. Viac o použití tohto refaktorovania pri integrácií s HTML možno nájsť v časti o spojení JS-HTML-CSS tejto kapitoly.

Extrakcia kódu do samostatnej metódy je analogicky ako pri „inline" metódy, priamočiaro aplikovateľná na JavaScript (s výnimkou kľúčového slova `this` reprezentujúceho objekt nad ktorým je metóda volaná).

Refaktorovanie : Presunutie metódy ("Move Method")

Kategória : metóda objektu

Popis :

Presunutie metódy z jednej triedy do druhej. V pôvodnej triede možno metódu celkom odstrániť alebo nechať odkaz na presunutú metódu. Dôležité je brať do úvahy nutnosť zmien vo všetkých výskytoch volaní tejto metódy.

Motivácia :

V prípade, že metóda používa, alebo je používaná viacerými vlastnosťami inej triedy ako tej, v ktorej je definovaná, má zmysel túto metódu presunúť.

Aplikácia na JavaScript :

Význam presunutia metódy v JavaScripte je na rovnakej úrovni ako pri ostatných jazykoch. Dosiahne sa čistejšia štruktúra a minimalizuje počet odkazov na iné objekty.

Implementáciu tohto refaktorovania v JS možno dosiahnuť priamočiarym spôsobom. V prípade, že je metóda definovaná v konštruktore objektu, stačí ju premiestniť do konštruktora druhého (vhodnejšieho) objektu.

Ak je metóda priradená ako vlastnosť objektu počas behu programu, je nutné zmeniť v príkaze objekt, ktorému je priradená. V tomto prípade však môže nastať situácia, že v danom momente druhý objekt ešte neexistuje. Musí sa teda zabezpečiť, aby bol nový objekt najprv definovaný a až potom mu bola priradená metóda.

Použitie refaktorovaní manipulujúcimi s dynamickými vlastnosťami objektov si preto vyžaduje opatrnosť, keďže nie je možné ľahko definovať vplyv ich použitia na správnosť kódu.

Príklad aplikácie v JS (metóda je v konštruktore) :

```
function MyObject1() {
      ...
      this.greet() = function () { alert("Hello.") }
      ...
}
function MyObject2() {  ... }
```

⇓

```
function MyObject1() {  ... }
function MyObject2() {
      ...
      this.greet() = function () { alert("Hello.") }
}
```

Príklad aplikácie v JS (metóda priradená za behu) :

```
var myObj1 = new MyObject1()
// ak by na tomto mieste bola metóda priradená a volaná prvým objektom, bolo by nutné pri refaktorovaní presunúť definíciu 'myObj2' nahor, alebo volanie prvým objektom nadol – čo nemusí byť možné
var myObj2 = new MyObject2()
myObj1.greet() = function () { alert("Hello.") }
```

⇓

```
var myObj1 = new MyObject1()
var myObj2 = new MyObject2()
myObj2.greet() = function () { alert("Hello.") }
```

Refaktorovanie : Parametrizácia metódy ("Parametrize Method")

Kategória : funkcia, metóda objektu

Popis :

Vytvorenie funkcie/metódy miesto iných funkcií/metód, ktoré majú podobné použitie a líšia sa len v hodnotách v ich tele. Novovytvorená funkcia/metóda bude mať tieto rôzne hodnoty ako vstupný parameter.

Motivácia :

Nová metóda zovšeobecňuje pôvodné, ktoré vykonávali tú istú funkcionalitu, len s inými hodnotami. Použitie parametra preto zlepší prehľad a štruktúru kódu.

Aplikácia na JavaScript :

Parametrizácia má v JavaScripte okrem použitia na "čistom" kóde jazyka veľký význam pri prepojení s HTML v hostiteľskom prostredí webového prehliadača. Práve z podobných HTML elementov sú častokrát volané podobné JavaScript funkcie, ktoré je možné týmto spôsobom zovšeobecniť.

Použitie tohto refaktorovania je analogické ako pri iných programovacích jazykoch vďaka podobnej implementácií funkcií/metód jazyka JS.

Príklad použitia v JS :

```
function fivePercentRaise() { // metóda objektu
      this.salary *= 1.05;
}
function tenPercentRaise() { // metóda objektu
      this.salary *= 1.10;
}
```

⇓

```
function raise(percentage) { // metóda objektu
      this.salary *= 1.0 + (percentage/100);
}
```

Refaktorovanie : Pridanie parametra metódy („Add parameter")

Kategória : funkcia, metóda

Popis :

Pridanie nového parametra do funkcie alebo do metódy objektu.

Motivácia :

Potrebné v situácií, keď funkcia/metóda potrebuje na vykonanie svojej funkcionality viac informácií od kontextu, ktorý ju volá.

Aplikácia na JavaScript :

Ako bolo uvedené v definícii jazyka JavaScript, počet argumentov funkcií a metód môže byť variabilný vďaka poľu `arguments`, ktoré je pri volaní napĺňané vstupnými hodnotami. To je dôvodom, prečo má toto refaktorovanie pridania parametrov funkcií v JS zanedbateľný význam, na rozdiel od iných programovacích jazykoch, kde je počet parametrov funkcií a metód pevne daný. Napriek tomu je možné toto refaktorovanie využiť v JS na pomenovanie nových vstupných parametrov, čo vedie k lepšiemu pochopeniu parametrov funkcie a ich významu. Z hľadiska programátora je totiž vhodnejšie pracovať napríklad v rámci kódu funkcie `prijmiZamestnanca()` s pomenovaným parametrom `plat` ako s elementom poľa `arguments` s príslušným indexom.

Úroveň atribútov objektu a premenných

Refaktorovanie na úrovni atribútov objektov a premenných má analogické vlastnosti a problémy ako úroveň metód objektu a funkcií.

Refaktorovanie : Presunutie atribútu objektu („Move field“)

Kategória : atribút objektu

Popis :

Presunutie atribútu z jednej triedy do druhej a zmena vo všetkých použitiach tohto atribútu.

Motivácia :

Ako je atribút používaný inou triedou viac ako tou, v ktorej je definovaný, má z hľadiska štruktúry význam ho presunúť.

Aplikácia na JavaScript :

Aplikácia tohto refaktorovania je podobná aplikácii refaktorovania "move method". Použitie na atribúty definované v konštruktore objektu je priamočiare. Pri presunutí atribútu definovaného dynamicky však podobne ako pri presúvaní metódy treba postupovať opatrne. Po nevhodnej zmene totiž môže byť atribút používaný pôvodným objektom pred tým, ako je vôbec priradený novému.

Príklad aplikácie v JS :

Viď. refaktorovanie "Move Method" (s atribútom miesto metódy).

Refaktorovanie : Dosadenie dočasnej premennej ("Inline temp")

Kategória : premenná

Popis :

Dosadenie výrazovej hodnoty dočasnej premennej do všetkých jej použití.

Motivácia :

Dočasná premenná môže stáť častokrát v ceste iným dôležitým refaktorovaniam. Má preto význam uvažovať o jej odstránení a to hlavne v prípadoch, keď je do nej len raz priradený jednoduchý výraz. Dosadením tohto výrazu namiesto premennej sa dosiahne lepšia východisková pozícia pre aplikáciu iných refaktorovaní.

Aplikácia na JavaScript :

Význam odstránenia dočasnej premennej v JavaScript kóde má rovnaký význam ako v iných jazykoch. Keďže sa z hľadiska premenných a priradzovania výrazov do premenných JavaScript správa rovnako ako iné programovacie jazyky, implementácia a aplikácia tohto refaktorovania bude preto analogická.

Príklad aplikácie v JS :

```
var result = 0;
var temp = 0;
for( var i=0; i<items.length; i++ )
      temp += items[i];
result = temp;
```

⇓

```
var result = 0;
for( var i=0; i<items.length; i++ )
      result += items[i];
```

Úroveň tried a objektov

Objektový model JavaScriptu je prototypovo-orientovaný, t.j. implementovaný odlišným spôsobom od väčšiny OO jazykov ([KITOBJ97]). To vplýva na možnosti celkového využitia všeobecných refaktorovaní, modifikujúcich hierarchiu tried a objektov, v JavaScripte.

Syntax JavaScriptu neobsahuje explicitné konštrukcie na podporu vlastností OO jazykov ako dedičnosť, polymorfizmus či enkapsulácia. Dedičnosť však možno simulovať pomocou prototypov, polymorfizmus využitím slabého typovania a enkapsuláciu využitím lokálnych premenných a funkcií v rámci konštruktorov objektov. Podpora týchto vlastností OO programovania je možná vďaka voľnosti prototypového modelu. Ten umožňuje dynamickú manipuláciu so štruktúrou a dokonca aj hierarchiou objektov za behu programu, čo poskytuje vývojárovi úplne flexibilné prostredie na vývoj. Uvoľnením štruktúry a hierarchie objektov sa však na

druhú stranu zväčšuje možnosť degradácie kódu, straty kontroly nad objektovým modelom a možnosť zanesenia chýb.

Použitie refaktorovaní určených pre OO jazyky so statickou štruktúrou a hierarchiou tried je v JavaScripte možné len do určitej miery. Je to možné v prípade, že sú všetky vlastnosti objektov definované v ich konštruktoroch, t.j. sú viazané staticky. Ak sú však niektoré vlastnosti modifikované, odstraňované alebo pridávané dynamicky, použitie refaktorovaní či už manuálnym, alebo automatickým spôsobom sa stáva príliš komplikovaným resp. neriešiteľným.

Nasledovné refaktorovanie ukazuje spôsob, akým sa dá reštrukturovať hierarchia v rámci prototypového objektového modelu jazyka JavaScript.

Refaktorovanie : Vyzdvihnutie metódy („Pull up method“)

Kategória : metóda objektu, trieda / objekt

Popis :

Presunutie metódy do nadtriedy objektu / nadobjektu.

Motivácia :

Ak podtriedy majú metódy s rovnakou funkcionalitou resp. výstupmi, je z hľadiska štruktúry vhodné odstrániť duplicitu presunutím tejto metódy do nadtriedy / nadobjektu.

Aplikácia na JavaScript :

Ak je refaktorovaná metóda definovaná „staticky“ v konštruktore podobjektov, jej vyzdvihnutie do nadobjektu je možné analogicky ako v jazykoch s pevným objektovým modelom. Rozdiel je len v použití prototypu miesto klasickej štruktúry dedičnosti.

Ak je však v niektorom z objektov definovaná mimo konštruktora, priame použitie môže spôsobiť zanesenie chýb (ak sa pred definovaním metódy v kóde využíva fakt, že ešte nie je definovaná).

Príklad aplikácie v JS :

```
function Employee() {
      this.dept = “general”;
}
function Salesman( name ) {
      var privateName = “”;
      function getPrivateName() {
            return privateName;
      }

      privateName = name;
      this.getName = getPrivateName;
      this.dept = “sales”;
      this.quota = 100;
      this.prototype = new Employee; // Employee bude prototyp pre
                                     // objekty Salesman
}
```

```
function Engineer( name ) {
	var privateName = "";
	function getPrivateName() {
		return privateName;
	}

	privateName = name;
	this.getName = getPrivateName;
	this.dept = "engineering";
	this.machine = "";
	this.prototype = new Employee;
}
```

Po refaktorovaní sa presunie definícia metódy `getName` z konštruktorov objektov `Salesman` a `Engineer` do konštruktora objektu `Employee`.

```
function Employee() {
	var privateName = "";
	function getPrivateName() {
		return privateName;
	}

	privateName = name;
	this.getName = getPrivateName;
	this.dept = "general";
}
function Salesman( name ) {
	this.dept = "sales";
	this.quota = 100;
	this.prototype = new Employee;
}
function Engineer( name ) {
	this.dept = "engineering";
	this.machine = "";
	this.prototype = new Employee;
}
```

Zhrnutie

Na základe horeuvedených poznatkov a refaktorovaní je možné povedať, že možnosť aplikácie všeobecných princípov a prístup refaktorovania závisí od úrovne, na ktorej sú refaktorovania použité.

V prípade metód, funkcií, atribútov a premenných nie je medzi JavaScriptom a inými OO jazykmi veľa rozdielov, či už z hľadiska syntaktickej štruktúry, alebo ich sémantiky. Z týchto dôvodov je možné na tejto úrovni aplikovať väčšinu refaktorovaní bez nutnosti veľkých zásahov. Zaujímavá je však otázka významu použitia niektorých refaktorovaní v JavaScripte. Jedným z príkladov sú refaktorovania pridávajúce a odstraňujúce parametre funkcií. Keďže JavaScript je natoľko flexibilný, že umožňuje variabilné množstvo vstupných parametrov, stáva sa použitie týchto refaktorovaní zbytočným.

Pri aplikácií refaktorovania na objekty JavaScriptu je však situácia odlišná. I keď je JavaScript založený na objektovo-orientovaných princípoch, refaktorovania na úrovni objektov je možné využiť len do istého rozsahu. Dôvod je jasný – JavaScript neobsahuje explicitné syntaktické konštrukcie, ktoré definujú viditeľnosť tried a ich vlastností a dedičnosť medzi jednotlivými triedami. Dokonca samotný pojem „trieda" v JavaScripte, ako aj iných prototypovo-založených jazykoch neexistuje.

Prototypový objektový model JavaScriptu umožňuje dynamické zmeny v štruktúre a hierarchií objektov jednoduchým spôsobom aj za behu programu. Táto silná vlastnosť jazyka JavaScript nie je v klasických OO jazykoch s pevnou štruktúrou a hierarchiou tried možná. Výhody tejto vlastnosti sa stávajú problémami pri analýze refaktorovania objektov a prípadnej implementácií softvérového nástroja. V čase návrhu programového kódu totiž nie je možné predvídať, akým spôsobom bude pozmenená štruktúra či dokonca hierarchia objektu počas behu programu. V prípade aplikácie všeobecných refaktorovaní na úrovni objektov v JavaScripte je teda nutné obmedziť záber ich použitia na vlastnosti objektov, ktoré sú definované „staticky" v rámci konštruktora objektu a nie sú dynamicky modifikované za behu programu. V takomto prípade je možné aplikovať všeobecné refaktorovania pomerne priamočiaro.

4.2 Špecifiká pri refaktorovaní DHTML

Pri prieskume refaktorovania v prostredí jazyka JavaScript nepostačuje len analýza použitia existujúcich refaktorovaní, ktoré sú používané v iných jazykoch, na JavaScript. Omnoho dôležitejšie je preskúmať, aké nové možnosti poskytuje refaktorovanie v novom jazyku a jeho prostredí. Pri tomto skúmaní práca vychádza hlavne zo špecifických vlastností jazyka JavaScript, ktoré nie sú prítomné v jazykoch, nad ktorými bolo refaktorovanie vyvinuté (t.j. Java, C++, Smalltalk a iné). Práve tieto vlastnosti umožňujú nový pohľad na refaktorovanie, identifikáciou dosiaľ nepoznaných možností úpravy štruktúry programového kódu.

Jednou z jedinečných vlastností, ktorá odlišuje JavaScript od ostatných OO jazykov, ktoré sú predmetom refaktorovania, je charakter skriptovacieho jazyka. JavaScript je používaný v rámci hostiteľského prostredia (hostiteľskej aplikácie), kde slúži na programovú manipuláciu s objektami tohto prostredia. Táto symbióza prináša okrem svojich podstatných výhod aj problémy. Tie súvisia hlavne s príliš silným spojením skriptovacieho jazyka a hostiteľského prostredia s jeho inými jazykmi. Z hľadiska štruktúry, prehľadnosti a hlavne znovupoužiteľnosti segmentov kódu skriptovacieho jazyka je však vhodné čo najlepšie oddelenie jazykov a presné definovanie rozhrania. Táto časť je venovaná analýze problémov spojených s integráciou v hostiteľskom prostredí a prináša vlastné riešenia vo forme nových refaktorovaní.

JavaScript v hostiteľskom prostredí prehliadača

I keď je JavaScript používaný v rôznych hostiteľských prostrediach, do povedomia sa dostal hlavne v prostredí internetových prehliadačov, kde sa stal obľúbeným a rozšíreným. Obrovské množstvo navrhnutých a vytvorených HTML stránok v sebe obsahuje časti kódu napísané pomocou JavaScriptu, ktorý sa stal de facto štandardom pre skriptovanie HTML. Na internetových serveroch je naviac každú sekundu použitím server-side programovacích jazykov (vytvorených pomocou technológií ako sú JSP, ASP či PHP) automaticky generované obrovské množstvo HTML stránok s integrovaným JavaScript kódom.

Z dôvodov širokého použitia jazyka v HTML stránkach má teda význam zaoberať sa možnosťami refaktorovania, ktoré súvisia s konkrétnym hostiteľským prostredím – s prostredím webových prehliadačov.

Hlavným dôvodom analýzy možností refaktorovania jazyka v hostiteľskom prostredí prehliadačov je práve jeho široké použitie v tomto prostredí. Tvorba webových stránok je na pohľad jednoduchá vďaka množstvu softvérových nástrojov a IDE prostredí a vďaka jednoduchosti HTML jazyka. V skutočnosti je však proces vývoja website aj vďaka možnostiam skriptovacích jazykov podobne zložitým, ako proces vývoja softvéru. Samotnú tvorbu musí predchádzať podrobná analýza a návrh a po vytvorení musí nasledovať dôkladné testovanie. Na tieto aspekty sa však pri vývoji v malom častokrát zabúda. Dôsledkom je rýchla degradácia kvality webových stránok a ich vzájomného prepojenia, ktorá je umocnená ich pravidelnými aktualizáciami, ku ktorým dochádza v omnoho väčšej miere ako pri klasických softvérových produktoch.

Proces vývoja webových stránok

Pri tvorbe HTML stránok sa obyčajne postupuje nasledovným spôsobom. Na začiatku sa zvolí obsah, ktorý bude klientom zverejnený, t.j. samotné informácie na publikovanie. Následne je tento obsah štruktúrovaný – rozdelený do zodpovedajúcich HTML stránok a v rámci týchto stránok do štruktúry odsekov a častí. Potom sú vytvorené stránky navzájom pospájané odkazmi na zodpovedajúcich miestach textu, čím sa z množiny sekvenčných textových stránok vytvorí hypertextový dokument s možnosťou navigácie na základe obsahu. Po voľbe obsahu, štruktúry a pospájaní jednotlivých stránok je navrhnutá prezentácia HTML stránok, t.j. spôsob, akým bude obsah zobrazený. Pri návrhu prezentácie je definovaný spôsob a štýl zobrazenia jednotlivých elementov stránky, najčastejšie pomocou CSS.

Úlohu JavaScriptu v procese návrhu HTML stránok nie je možné zaradiť priamo do jedného z krokov procesu. Prostredníctvom tohto jazyka je totiž možné manipulovať s obsahom, štruktúrou a aj s prezentáciou stránky a dokonca generovať samotný HTML kód (funkciou `document.write()`). V praxi sa JavaScript používa hlavne pri návrhu prezentácie HTML stránok, kedy slúži na zdynamizovanie ich elementov a spolu s manipuláciou štýlu elementov

pomocou CSS tvorí koncept tzv. Dynamic HTML (DHTML). JavaScript však v HTML dokáže omnoho viac a preto je jeho obmedzenie len na oblasť prezentácie príliš povrchné. Možnosti, akými sú napríklad validácia polí formuláru, komplikované výpočty či komunikácia so servermi posielaním požiadaviek a získavaním odpovedí, poskytujú silné nástroje na obohatenie funkcií klienta než je len samotná prezentácia. V procese návrhu stránok je teda vhodnejšie zaradiť tvorbu JavaScript funkcionality do samostatného kroku, ktorý do istej miery dopĺňa návrh štruktúry a prezentačnej logiky stránok. Po vývoji JavaScript kódu by malo nasledovať jeho dôkladné otestovanie, v rámci kódu samotného a vzhľadom na zvyšok HTML dokumentov, v ktorých je kód integrovaný.

Pri procese návrhu HTML stránok dochádza častokrát k výraznej degradácií ich štruktúry. Napriek tomu, že bol website vytvorený systematicky a kvalitne, žiadané zmeny neberú ohľad na pôvodnú štruktúru a návrh celého systému. Vývojár pri údržbe napríklad začne vkladať JavaScript kód na nevhodné miesto, prípadne odstráni zdanlivo nepoužívanú funkciu z JavaScript súboru. Keďže tieto zmeny sa vykonávajú jednoducho a často, sú zamerané na úzku oblasť bez ohľadu na celkový kontext a vedú priamo zavedeniu chýb či k degradácií štruktúry.

Táto degradácia môže prebiehať na úrovni celého systému stránok (, kde dôjde k porušeniu štruktúry a prepojenia stránok vo website) alebo na úrovni stránky samotnej (kedy dochádza k zhoršeniu štruktúry v rámci stránky, čo môže byť spôsobené nevhodnou modifikáciou vo vnútri stránky alebo iného súboru s ktorým má stránka vzťah).

Degradácia DHTML na úrovni systému stránok

Pri *degradácii na úrovni systému stránok* dochádza k porušeniu štruktúry vzťahov medzi jednotlivými stránkami – obsah nie je prirodzene štruktúrovaný (, t.j. je nevhodne rozložený na viacerých stránkach resp. spojený v jeden veľký text) alebo dochádza k nevhodnému prepojeniu nesúvisiacich stránok či k chýbajúceho prepojeniu tam, kde je to vhodné. V HTML stránkach je obsah fyzicky zviazaný s HTML štruktúrou, CSS a aj s JavaScript kódom. Z tohto dôvodu má význam, podobne ako pri samotnom obsahu, uvažovať nad vhodným štruktúrovaním JavaScript kódu v súvislosti s HTML stránkami. Pri presune časti obsahu a jeho príslušných HTML elementov je nutné premiestniť aj JavaScript kód. Ak sa totiž poruší obsahová štruktúra, obyčajne sa poruší aj štruktúra JavaScript funkcionality.

Z uvedeného vyplýva, že refaktorovanie JavaScript kódu na úrovni systému vzájomne prepojených stránok má z hľadiska štruktúry stránok nezanedbateľný význam. Ako príklad refaktorovaní aplikovateľných na túto úroveň možno spomenúť presunutie JavaScript kódu z jednej HTML stránky do druhej (kde je používaný) alebo jeho extrakcia do samostatného súboru (s príponou „js") a jeho použitie odkazom vo viacerých stránkach, využívajúcich jeho funkcionalitu. Príklady týchto refaktorovaní sú podrobnejšie popísané v ďalšom texte.

Pri aplikácií refaktorovaní na tejto úrovni je nutnou podmienkou znalosť globálnej štruktúry a vzťahov medzi jednotlivými stránkami a informácie o použití JavaScript funkcií a objektov v stránkach. Lepšiu pochopiteľnosť vzťahov siete webových stránok by mohla podporiť funkcia pohľadu na celú sieť prepojení v rámci príslušných IDE prostredí a to aj s možnosťou zobrazenia vzťahov z hľadiska JavaScriptu. Vývojové prostredie by napríklad mohlo zobrazovať vzťah medzi JavaScript súborom a webovými stránkami, ktoré využívajú jeho funkcie. Tým by sa zjednodušilo vyhľadávanie volaných funkcií a zlepšila prehľadnosť, čo výrazne vplýva na zníženie počtu chýb.

Degradácia DHTML na úrovni stránky

K *degradácii na úrovni stránky* dochádza pri modifikácií HTML elementov, CSS a JavaScript kódu v rámci stránky samotnej – napríklad presunutím elementu do iného frame-u, opakovaním rovnakého elementu viackrát (miesto použitia JavaScriptu na generovanie elementu v cykle), použitím nevhodných konštrukcií JavaScriptu a jeho zlým prepojením s HTML, pridaním zbytočného nezobrazovaného elementu alebo nadbytočného atribútu elementov či definovaním funkcie, ktorá nie je používaná. Jedná sa o zlú štruktúrovanosť na úrovni syntaxe jazykov HTML, CSS a JavaScript, prípadne o nevhodné či chýbajúce prepojenie medzi týmito jazykmi.

Potreba štruktúrovanosti na predchádzajúcej vyššej úrovni – úrovni systému stránok, súvisí hlavne s rozložením samotného obsahu a funkcií do siete navzájom prepojených stránok tak, aby bol systém ľahko pochopiteľný a priechodný *z hľadiska používateľa*.

Na úrovni samotnej stránky ide o štruktúrovanosť z hľadiska konštrukcii použitých jazykov, ktorá vedie hlavne k čistejšiemu a efektívnejšiemu kódu stránky, čo pomáha pri ďalšom vývoji a použití stránky *z hľadiska vývojára*. Ako príklad refaktorovaní na úrovni stránky možno uviesť refaktorovania na úrovni funkcií a premenných JavaScriptu (odstránenie nepotrebnej funkcie, extrakcia a dosadenie funkcie, premenovanie funkcie a premennej ...) ale aj extrakciu HTML, či CSS kódu do JavaScript funkcie generujúcej kód stránky alebo extrahovanie JavaScript kódu z udalostí HTML elementu (ako onClick či onLoad) do samostatnej funkcie. Tieto refaktorovania sú podrobne popísané v ďalšom texte tejto časti. Možnosti na úrovni stránky súvisia so syntaxou jazykov HTML, JavaScript, CSS a spôsobom ich prepojenia pomocou objektového modelu dokumentu - DOM.

Cieľ refaktorovaní v prostredí prehliadača

Hostiteľské prostredie HTML prehliadačov slúži hlavne na zobrazenie požadovaných webových stránok zo serverov. Tieto stránky obsahujú informácie vo viacerých jazykoch – HTML, CSS či JavaScript. Všetky z týchto jazykov dávajú stránkam vlastnosti iného charakteru : HTML poskytuje

štruktúru a obsah, CSS definuje zobrazenie a JavaScript dynamickosť. Ich prítomnosť na stránkach je preto dôležitá a použitie refaktorovaní by ju nemalo žiadnym spôsobom obmedzovať.

Cieľom refaktorovaní v tomto prostredí má byť hlavne zlepšenie štruktúry a prehľadnosti kódu, čo vedie k jednoduchšej údržbe a pochopiteľnosti. Refaktorovaný kód je jasný, jednoduchý a bez redundancií (t.j. bez opakovania kódu), čím sa zmenšuje pravdepodobnosť zavedenia chýb pri údržbe systému. Vďaka jasnej štruktúre a rozhraniu je možné jednotlivé časti kódu ľahko použiť v iných moduloch a komponentoch (software reuse).

Pri dôraze na tieto ciele sa ukazuje ako najvhodnejšie jasné a presne definované oddelenie kódu jazykov, ktoré sú použité v rámci HTML stránky. Každý z jazykov slúži na iné účely a mal by byť preto od ostatných jasne oddelený a parametrizovaný, s explicitne definovaným rozhraním. Toto oddelenie a zovšeobecnenie kódu jazyka umožňuje okrem zlepšenia štruktúry použiť generalizovaný kód aj v iných častiach systému stránok, prípadne pri riešení iného problému. Taktiež stačí pri úprave stránky modifikovať jedine samostatne oddelenú časť, keďže redundancia kódu bola zovšeobecnením odstránená.

Oddelenie kódu jednotlivých jazykov v rámci HTML stránky vyplýva zo spôsobu a možností prepojenia týchto jazykov na stránke. Jadro HTML stránok tvoria elementy jazyka HTML so samotným obsahom. Na toto jadro je možné napojiť štýly CSS a funkcionalitu skriptovacích jazykov ako JavaScript, prostredníctvom rozhrania medzi jednotlivými jazykmi stránky - špeciálnych HTML elementov, prípadne ich atribútov či udalostí. Spôsob prepojenia jazykov HTML dokumentu je podrobne popísaný v kapitole o jazyku JavaScript.

Integrácia HTML, či už s JavaScriptom alebo s CSS prebieha na troch úrovniach :

- v definícii HTML elementu (kód v udalostiach resp. v atribúte *style*),
- v kóde hlavičky alebo tela stránky (priame začlenenie v stránke),
- v hlavičke stránky referenciou na samostatný súbor s kódom.

Na najnižšej ***úrovni definície HTML elementu*** je z hľadiska JavaScriptu vhodné, aby kód udalostí (teda rozhranie prepojenia) bol čo najjednoduchší. Extrakciou tohto kódu do samostatnej JavaScript funkcie a nahradením kódu v udalosti volaním tejto funkcie s príslušnými parametrami možno dosiahnuť túto požiadavku. Dôležitá je hlavne parametrizácia funkcií, ktorá by mala brať ohľad na ďalšie zovšeobecnenie funkcie (ako napríklad odstránenie závislosti od elementu, v ktorom je používaná, pridaním elementu ako vstupného parametra, atď.).

Na ***úroveň priamo začleneného kódu*** v stránke je vhodné umiestniť JavaScript kód špecifický len pre túto stránku. Príkazy a definície funkcií všeobecného charakteru, ktoré sú používané aj v iných stránkach, je lepšie premiestniť do samostatného JavaScript súboru.

JavaScript kód ***úrovne samostatného súboru*** je ideálny pri znovupoužití kódu v iných stránkach a projektoch. Jeho účel je preto hlavne pri kóde, ktorý je často používaný a na dostatočne všeobecnej úrovni. Nevýhodou umiestenia kódu do tejto úrovne je zhoršenie prehľadnosti kódu v niektorých prípadoch. Ak je totiž v HTML stránke začlenených viacero súborov, pri odkazoch na funkcie a premenné z týchto súborov nie je hneď jasné, z ktorého súboru sú začlenené (súbory sú totiž do stránky začleňované priamo obsahom). Pri vyriešení tohto problému môžu pomôcť ďalšie refaktorovania, napríklad premenovaním funkcií, objektov či premenných na vhodnejšie mená, identifikujúce súbor, kde sú definované. Dobrým riešením je aj rozšírenie IDE prostredí o zobrazenie týchto vzťahov medzi HTML dokumentom a JavaScript súbormi, ktoré používa.

K lepšej štruktúre JavaScript kódu v rámci HTML stránok prispieva aj využitie objektového modelu jazyka. Vytvorením vlastných objektov v jazyku JavaScript je možné popísať a riešiť problém objektovo-orientovaným prístupom - ako vzťahy medzi objektami so svojimi vlastnosťami a funkciami. Pomocou objektov možno zoskupiť funkcie a premenné, ktoré navzájom súvisia a zapúzdriť ich do jedného celku. Namiesto množstva samostatných funkcií a premenných v stránke sa potom používa len malé množstvo vytvorených objektov s ich stavovými informáciami a funkciami, každý so špecifickým určením.

Katalóg refaktorovaní DHTML

Refaktorovania na úrovni systému stránok

Vzťahy medzi stránkami v rámci systému stránok sú definované hyperlinkovými odkazmi z jednej stránky na druhú. Úpravy na úrovni vzťahov súvisia so štruktúrovaním obsiahnutých informácií medzi jednotlivé stránky a pridávaním resp. odstraňovaním vzťahov medzi nimi. Tieto úpravy sa týkajú HTML stránok a ich elementov, ktoré môžu využívať funkcionalitu jazyka JavaScript. Pri presune obsahu vo forme HTML elementov je častokrát nutné premiestniť aj JavaScript kód s nimi spojený, prípadne ho extrahovať do samostatného súboru pre použitie vo viacerých stránkach.

Refaktorovanie 1 : Extrakcia JavaScript kódu stránky do samostatného súboru

Motivácia : Pre účely budúceho presunutia alebo použitia kódu v iných HTML stránkach je vhodné JavaScript kód extrahovať do súboru, ktorý sa dá ľahko distribuovať a vkladať do iných stránok.

Vstup : Extrahovaný kód, názov nového JavaScript súboru (s príponou „js“).

Podmienky : Názov nového súboru nesmie byť v konflikte s už existujúcimi súbormi.

Príklad použitia :

```
———————— stranka.htm ——————————
...
<SCRIPT language="javascript">
      var sName = "myName";
      function test() {
            alert(sName);
      }
</SCRIPT>
...
```

⇓

```
———————— stranka.htm ——————————
...
<SCRIPT language="javascript" src="funkcie.js" />
...
```

```
———————— funkcie.js ———————————
var sName = "myName";
function test() {
      alert(sName);
}
```

Poznámky :

- Samotnú extrakciu môže predchádzať zovšeobecnenie presúvaného kódu (funkcií) a iné úpravy na úrovni stránky.
- Inverzným k tomuto refaktorovaniu je refaktorovanie č. 2 („Včlenenie JavaScript kódu samostatného súboru do stránky").

Refaktorovanie 2 : Včlenenie JavaScript kódu samostatného súboru do stránky

Motivácia : Ak JavaScript kód umiestnený v samostatnom súbore je používaný len v jednej stránke (prípadne malom počte stránok) a je špecifický pre túto stránku a preto sa nebude používať v iných, je vhodné včleniť kód súboru do samotnej HTML stránky. Vďaka tejto úprave možno pri použití funkcií, objektov a premenných pôvodného súboru ľahšie vystopovať ich definície (hlavne v prípade, že sú používané viaceré začlenené JavaScript súbory).

Vstup : JavaScript súbor, ktorý bude včlenený a HTML stránka (prípadne stránky), ktorá ho používa a kam bude vložený.

Podmienky : JavaScript kód zo súboru musí byť včlenený do všetkých stránok, ktoré používajú funkcionalitu pôvodného súboru, t.j. nemôže existovať HTML stránka, ktorá volá funkciu, objekt, alebo premennú súboru a nie je na vstupe refaktorovania.

Príklad použitia : viď príklad k refaktorovaniu č. 1 opačným smerom

Poznámky :

- Inverzným k tomuto je refaktorovaniu je refaktorovanie č. 1 („Extrakcia JavaScript kódu stránky do samostatného súboru").

Refaktorovanie 3 : Presunutie JavaScript kódu z jednej stránky do druhej

Motivácia : Pri presunutí HTML obsahu, na ktorý je JavaScript kód viazaný je nutné presunúť aj tento kód. Refaktorovanie je možné použiť aj v prípade, že sa v zdrojovej stránke kód nepoužíva a z hľadiska štruktúry (či budúceho použitia) je vhodnejšie jeho umiestnenie v cieľovej stránke.

Vstup : Presúvaný JavaScript kód na zdrojovej stránke, zdrojová stránka, cieľová stránka, cieľová pozícia presunutia na cieľovej stránke

Podmienky : Presúvaný kód nemôže byť používaný v zdrojovej HTML stránke a jej frame-och.

Príklad použitia :

```
——————— stranka1.htm ————————————
...
<SCRIPT language="javascript">
      var sName = "myName";
</SCRIPT>
...
—————————————————————————————————
——————— stranka2.htm ————————————
...
—————————————————————————————————
```

⇓

```
——————— stranka1.htm ————————————
...
—————————————————————————————————
——————— stranka2.htm ————————————
...
<SCRIPT language="javascript">
      var sName = "myName";
</SCRIPT>
...
—————————————————————————————————
```

Poznámky :

- Miesto priameho presunutia je možné taktiež extrahovať presúvaný kód do samostatného súboru (refaktorovanie 1) a premiestniť či skopírovať len referenciu na súbor zo zdrojovej stránky do cieľovej.

Refaktorovania nad frame-ami stránok

HTML stránky môžu byť štruktúrované do tzv. frame-ov, rozdeľujúcich stránku na viacero okien. Frame, je časť okna patriaca (rodičovskej) HTML stránke, ktorá môže obsahovať ľubovoľnú HTML stránku. Rodičovská stránka je teda rozložená do viacerých častí, kde v každej sa zobrazuje nejaká podstránka. Štruktúru frame-ov možno z hľadiska úrovní zaradiť niekde

medzi úroveň systému stránok a úroveň stránok. Podstatnou vlastnosťou stránky obsahujúcej frame-y je možnosť dynamickej zmeny obsahu jednotlivých frame-ov (načítaním inej stránky) a možnosť prístupu k funkciám a prvkov medzi frame-ami prostredníctvom jazyka JavaScript. Týmto spôsobom môže jeden frame využívať vlastnosti iného frame-u tej istej rodičovskej stránky, prípadne vlastnosti samotnej rodičovskej stránky. Koncept frame-ov poskytuje ďalšie možnosti refaktorovania. Jednou z nich je presúvanie kódu z jedného frame do druhého.

Refaktorovanie 4 : Presunutie JavaScript kódu z jedného frame stránky do druhého

Motivácia : Ak je umiestnenie kódu prirodzenejšie a vhodnejšie v cieľovom frame (napr. viac používané, alebo obsahovo bližšie), je lepšie ho premiestniť.

Vstup : Presúvaný JavaScript kód na zdrojovom frame, zdrojový frame, cieľový frame, cieľová pozícia presunutia na cieľovom frame.

Podmienky : Presúvaný kód nemôže byť používaný v zdrojovom frame.

Príklad použitia :

```
————————— stranka.htm ———————————
<FRAMESET ROWS="200,*" ... >
      <FRAME src="frame1.htm" name="prvyFrame" ... />
      <FRAME src="frame2.htm" name="druhyFrame" ... />
</FRAMESET>
—————————————————————————————————
————————— frame1.htm ————————————
...
<SCRIPT language="javascript">
      function test() {
            alert("10");
      }
      test(); // použitie
</SCRIPT>
...
—————————————————————————————————
————————— frame2.htm ————————————
...
<SCRIPT language="javascript">
      parent.prvyFrame.test();
      ...
      parent.prvyFrame.test();
      ...
      parent.prvyFrame.test();
</SCRIPT>
...
—————————————————————————————————
```

⇓

```
————————— stranka.htm ———————————
<FRAMESET ROWS="200,*" ... >
      <FRAME src="frame1.htm" name="prvyFrame" ... />
      <FRAME src="frame2.htm" name="druhyFrame" ... />
```

```
</FRAMESET>
```

```
———————— frame1.htm ————————
...
<SCRIPT language="javascript">
      parent.druhyFrame.test();
</SCRIPT>
...
```

```
———————— frame2.htm ————————
...
<SCRIPT language="javascript">
      function test() {
            alert("10");
      }
      test();
      ...
      test();
      ...
      test();
</SCRIPT>
...
```

Refaktorovania na úrovni stránky

Po úvodnom rozdelení obsahu do systému webových stránok je proces vývoja sústredený hlavne sa štruktúrovanie, zobrazenie a zdynamizovanie obsahu jednotlivých stránok. Väčšina úprav pri údržbe a aktualizácii je teda vykonávaná na úrovni HTML stránok, čo vedie k rýchlejšej degenerácií štruktúry na tejto úrovni ako na úrovni celého systému.

Refaktorovania na úrovni stránky možno rozdeliť do štyroch kategórií :

- premiestňovanie JavaScript kódu po HTML stránke,
- štrukturálne pretvarovanie JavaScript kódu,
- pretvarovanie medzi jazykmi stránky
- a iné refaktorovania súvisiace s vlastnosťami jazyka JavaScript.

Premiestňovanie JavaScript kódu po HTML stránke poskytuje možnosti umiestnenia kódu tam, kde je to najvhodnejšie z hľadiska štruktúry, prehľadnosti a budúceho vývoja stránky. Kód jazyka JavaScript je možné umiestniť do štyroch častí HTML stránky :

- samostatný Javascript súbor (s príponou „js")
- hlavička stránky (kód medzi HTML elementmi `<head>` a `</head>`)
- telo stránky (kód medzi HTML elementmi `<body>` a `</body>`)
- udalosti elementov (kód v atribútoch udalostí elementov – napr. `onClick` ...)

Premiestnenie (extrakcia) kódu do samostatného súboru prebieha na úrovni systému stránok a je popísané v časti, ktorá sa venuje refaktorovaniam tejto úrovne. Táto časť sa bude venovať presúvaniu JavaScript kódu v rámci stránky.

Refaktorovanie 5 : Presunutie JavaScript kódu v hlavičke a tele HTML Stránky.

Motivácia :

1. Premiestnením možno zoskupiť súvisiaci kód, ktorý sa dá neskôr extrahovať či použiť v iných stránkach. Pri správnom použití refaktorovania sa zvyšuje prehľadnosť a štruktúrovanosť kódu.
2. Definície funkcií sa odporúča umiestniť do hlavičky a nie do tela HTML stránky. Umiestnenie má totiž v tomto prípade vplyv na spôsob vykresľovania stránok. Definície v hlavičke stránky sa načítavajú pred zobrazovaním stránky, pričom definície v tele sa načítajú v momente, keď sa zobrazia HTML elementy pred JavaScript kódom týchto definícií. Použitím tohto refaktorovania je teda možné premiestniť definície do hlavičky HTML.

Vstup : Presúvaný JavaScript kód, cieľová pozícia na stránke.

Podmienky : Pri presunutí kódu v hlavičke a tele stránky je dôležité brať do úvahy vzťah presúvaného kódu so súvisiacim HTML a JavaScript kódom. Ak je napríklad kód, v ktorom sa volá funkcia, presunutý pred samotnú definíciu funkcie, dôjde pri zobrazovaní stránky k chybe (t.j. k zmene funkcionality), čo nie je žiadané. Podobne dôjde k chybe, ak je presúvaný kód z tela spojený s niektorým HTML elementom (t.j. využíva alebo nastavuje jeho atribúty) a je refaktorovaním presunutý pred tento element. Ďalším chybným použitím je ak sa definícia funkcie presunie až za jej volanie. Refaktorovanie je teda možné použiť len v prípadoch, ak po vykonaní zostanú definície funkcií pred ich volaniami a použitím.

Príklad použitia :

```
<HTML>
      <HEAD>
      ...
      </HEAD>
      <BODY>
      ...
            <SCRIPT language="JavaScript">
            function test() {
                  alert("10");
            }
            </SCRIPT>
      ...
      </BODY>
</HTML>
```

⇓

```
<HTML>
      <HEAD>
```

```
    ...
        <SCRIPT language="JavaScript">
        function test() {
            alert("10");
        }
        </SCRIPT>
    </HEAD>
    <BODY>
    ...
    </BODY>
</HTML>
```

Poznámky :

- Pri presúvaní kódu len v rámci hlavičky stačí overovať, či sú funkcie a premenné definované pred ich volaním a použitím. Ak je ale kód presúvaný aj v tele, je nutné testovať aj HTML elementy, ktoré sú používané v presúvanom kóde.
- Presunutie kódu z hlavičky do tela stránky väčšinou nemá praktický význam.
- Toto refaktorovanie možno podľa potreby rozdeliť na 4 refaktorovania : presúvanie len v hlavičke, len v tele, z tela do hlavičky, z hlavičky do tela.

Refaktorovanie 6 : Extrakcia JavaScript kódu z udalosti HTML elementu do funkcie

Motivácia :

1. Kód udalostí HTML elementov je častokrát rovnaký pre udalosti rovnakého elementu (napr. správanie HTML elementu pri `onClick` a `onKeyPressed`), prípadne pre udalosti rôznych elementov. Príkladom môže byť overovanie správnosti číselných textových polí. V týchto prípadoch je užitočné odstrániť redundanciu opakujúceho sa kódu v udalostiach a definovať kód len v jednej funkcií, ktorá môže byť ľahko použitá vo všetkých udalostiach – súčasných aj budúcich.
2. JavaScript kód v udalostiach by mal byť čo najjednoduchší. Vtedy je kód jasne oddelený od samotného HTML elementu, čo zabezpečí lepšiu štruktúru stránky podľa jazykov. Rozdelenie potom umožňuje jednoduchšie manipulovať s jednotlivými jazykmi, ako keby boli všetky navzájom pomiešané.

Vstup : Udalosť HTML elementu, názov novej funkcie.

Podmienky : Názov novej funkcie nemôže kolidovať s existujúcimi funkciami.

Príklad použitia :

```
<INPUT id="vek" onClick="if (this.value='') alert('Zadajte vek.'); "
value="" />
```

```
<SCRIPT language="JavaScript">
      function OveritVek(element) {
            if (element.value="")
                  alert("Zadajte vek.");
      }
</SCRIPT>
<INPUT id="vek" onClick="overVek(this)"  value="" />
```

Poznámky :

- Ak JavaScript kód udalosti obsahuje vlastnosti, ktoré sa viažu na príslušný HTML element, je nutné extrahovanú funkciu parametrizovať týmto elementom alebo používanou vlastnosťou/vlastnosťami (viď príklad, ktorý obsahuje parameter elementu samotného).
- Inverzným refaktorovaním je včlenenie definície funkcie do volania v rámci kódu udalosti. Toto refaktorovanie však nemá veľký praktický význam.

Pri ***štrukturálnom pretvarovaní stránky*** dochádza k zmenám použitých konštrukcií jazyka. Konštrukcie JavaScriptu možno rozdeliť na tri skupiny : samostatne umiestnený kód (globálny, obsahujúci postupnosť príkazov), funkcie a objekty. Refaktorovania tejto oblasti zodpovedajú možnostiam transformácií z jednej skupiny do druhej. V skutočnosti sa väčšinou jedná o všeobecné refaktorovania, ktorých aplikovateľnosť už bola analyzovaná v samostatnej časti tejto kapitoly (časť 4.1).

Štrukturálne pretvarovanie zodpovedá nasledovným refaktorovaniam :

- globálny kód ⇒ funkcia : extrakcia kódu do funkcie
- funkcia ⇒ globálny kód : vloženie (inline) obsahu funkcie
- globálny kód (len premenná) ⇒ objekt : priradenie premennej ako vlastnosť objektu
- funkcia ⇒ objekt : priradenie funkcie ako metódu objektu
- objekt ⇒ funkcie a premenné : odstránenie objektu a zachovanie jeho metód ako funkcií a vlastností ako premenných (nemá veľký význam)

Pretvarovanie medzi jazykmi stránky predstavuje zmenu jazyka kódu z HTML na JavaScript prípadne naopak. Ako už bolo spomenuté, JavaScript poskytuje možnosť generovania HTML kódu prostredníctvom funkcie `document.write()`. Túto vlastnosť je možné využiť pri HTML kóde rovnakého charakteru, ktorý sa viackrát opakuje. Častokrát sa stáva, že na strane servera sú programovo generované stránky, ktoré vracajú zle štruktúrovaný a opakujúci sa HTML kód (napríklad 10 prázdnych riadkov tabuľky ...). V týchto prípadoch je vhodné použiť nasledujúce refaktorovanie.

Refaktorovanie 7 : Nahradenie postupnosti rovnakého (podobného) HTML kódu JavaScript cyklom, ktorý ho generuje

Motivácia : Zlepšenie štruktúry a skrátenie kódu odstránením za sebou sa opakujúcich HTML elementov.

Vstup : HTML kód, ktorý bude nahradený.

Podmienky : Refaktorovanie je vhodné použiť, ak sa nahradzuje väčší počet rovnakého HTML kódu (aspoň 3 opakujúce sa časti). Nová premenná cyklu musí byť odlišná od existujúcich premenných.

Príklad použitia :

```
<TR><TD> </TD></TR>
<TR><TD> </TD></TR>
<TR><TD> </TD></TR>
<TR><TD> </TD></TR>
<TR><TD> </TD></TR>
```

⇓

```
<SCRIPT language="JavaScript">
      for( var i=0;i<5;i++ )
            document.write("<TR><TD> </TD></TR>")
</SCRIPT>
```

Poznámky :

- Nahradenie HTML elementov príkazmi JavaScript jazyka môže v niektorých prípadoch zhoršiť oddelenie jednotlivých jazykov stránok (v tomto prípade HTML a JavaScript), čo zle vplýva na štruktúru a prehľadnosť kódu. JavaScript totiž nemusí byť vhodné využívať na vykresľovanie HTML elementov. Pri priamočiarom použití, kedy je jasne vidieť účel JavaScript kódu pri vykresľovaní, však refaktorovanie spĺňa svoj účel.
- Obyčajne sa HTML elementy úplne nezhodujú a preto môže byť nutná ich „unifikácia" pred použitím refaktorovania. Ak sa však veľmi nelíšia, je možné toto refaktorovanie rozšíriť parametrizáciou – pridaním parametra funkcie do ktorého vstupujú rozdielne hodnoty. To si však žiada rozšírenie refaktorovania na extrakciu funkcie.
- Inverzné refaktorovanie nahradzuje JavaScript kód priamo HTML kódom, ktorý pôvodne generoval. Je užitočné hlavne pri oddelení štruktúry jazyka HTML od JavaScriptu, čo je častokrát žiadané. Väčšinou ho však nie je jednoduché použiť, keďže HTML kód, ktorý sa z JavaScriptu generuje, je známy až počas behu programu.

Medzi ***iné refaktorovania súvisiace s vlastnosťami jazyka JavaScript*** patria úpravy ako doplnenie deklarácií premenných, či doplnenie oddeľovačov príkazov. Jedná sa o refaktorovania, ktoré vychádzajú z voľnosti jazyka JavaScript pri implementácií vykonávania a zobrazovania kódu.

Refaktorovanie 8 : Doplnenie deklarácií premenných

Motivácia : JavaScript poskytuje voľnosť pri deklarovaní premenných. Ak nie je premenná deklarovaná prostredníctvom kľúčového slova `var`, deklaruje sa automaticky pri prvom použití s hodnotou `undefined` (ak nie je do nej predtým priradená hodnota). Táto vlastnosť nemusí byť vždy vhodná a častokrát môže zhoršiť prehľadnosť a správnosť kódu. Pri chybe pri písaní názvu premennej sa systém navonok správa korektne, pri používaní premennej však môže prejsť do neočakávaných či chybových stavov. Explicitnou deklaráciou premenných a prípadným priradením počiatočnej hodnoty sa zlepší prehľadnosť kódu a zabráni tejto situácii.

Vstup : Kód, ktorý bude prezretý a doplnený o chýbajúce deklarácie.

Príklad použitia :

```
function test() {
      i = 10;
      alert(i);
}
```

⇓

```
function test() {
      var i = 10;
      alert(i);
}
```

Refaktorovanie 9 : Doplnenie oddeľovačov príkazov

Motivácia : Podobne ako pri deklarácií premenných sa pri interpretovaní JavaScript kódu ponecháva voľnosť v zadávaní oddeľovačov príkazov. Na zlepšenie prehľadnosti je vhodné explicitne doplniť oddeľovače príkazov, čím bude jasné, kde príkazy začínajú a končia. Tým sa zamedzí možnej viacznačnosti a prípadnému zavedeniu chýb do kódu.

Vstup : Kód, ktorý bude prezretý a doplnený o chýbajúce oddeľovače.

Príklad použitia :

```
var i = 10
alert(i)
```

⇓

```
var i = 10;
alert(i);
```

4.3 Refaktorovanie a prototypový objektový model

Prototypový objektový model JavaScript poskytuje veľa výhod pri manipulácií s objektami a modelovaní riešeného problému. Objekty berie ako dynamické entity, ktorých vlastnosti sa môžu meniť či pribúdať počas existencie života objektu. Tento prístup je veľmi odlišný od klasických

objektovo-orientovaných jazykov, kde je štruktúra a hierarchia objektov pevne daná pri ich vytváraní, na základe šablóny (triedy, ktorej patria).

Flexibilita objektov a ich vlastností je nespornou výhodou prototypovo-založených jazykov. Pri úvahách o refaktorovaní však spôsobuje nemalé problémy. Väčšina refaktorovaní klasických OO jazykov vychádza z pevnej hierarchie a vnútornej štruktúry tried a ich inštancií. To umožňuje použiť refaktorovania ako presunutie metódy, vyzdvihnutie metódy či extrakcia nadtriedy a uvažovať nad implementáciou softvérového nástroja na ich automatizáciu. Dynamika vlastností objektov v prototypovo modeli však spôsobuje, že v čase návrhu kódu nie je možné predpovedať štruktúru týchto vlastností objektov. Na ľubovoľnom mieste v kóde môže byť za istých podmienok pridaná nová vlastnosť alebo modifikovaná či odstránená existujúca.

Manuálna aplikácia refaktorovaní objektového modelu, ktoré predpokladajú jeho statickú štruktúru, teda so sebou nesie príliš veľké riziko zanesenia chýb. Je však možná, ak vývojár vie, že dynamické priradzovanie a modifikácia vlastností objekt nie je využívaná. Dosiahnuť tento stav by mu mohlo pomôcť nasledovné refaktorovanie.

Refaktorovanie 10 : Zmena dynamickej vlastnosti objektu na statickú

Motivácia : Vlastnosti, ktoré sú objektom dynamicky priradené zhoršujú orientáciu v kóde. V konkrétnom momente nemusí byť jasné, či vlastnosť patri objektu alebo už bola odstránená. To môže vyvolať ľahké zanesenie chýb.

Vstup : Dynamicky priradená vlastnosť objektu (mimo konštruktora).

Podmienky :

Pri použití tohto refaktorovanie je potrebná veľká opatrnosť. Možno ho použiť v prípade, ak je do vlastnosti priradená hodnota resp. funkcia, ktorá je statického charakteru, t.j. môže byť presunutá do konštruktora objektu. Ak je totiž hodnota závislá na predchádzajúcom kóde, presunutie vlastnosti by spôsobilo porušenie tejto väzby.

Refaktorovanie možno priamočiaro použiť len v prípade, že dynamická vlastnosť je v kóde pridávaná prototypu objektu (`myObject.prototype.age = 10;`), pričom tento prototyp je anonymný (t.j. nie je to objekt definovaný konštruktorom). Ak by totiž pôvodne bola priamo pridaná ku konkrétnemu objektu, presunutím do jeho konštruktora by sa pridala aj ostatným objektom vytvoreným pomocou tohto konštruktora – a to by nemuselo byť želateľné.

Príklad použitia :

```
function myObject(name) {
  this.name = name;
  this.getName = function() { return "Name : " + this.name  }
}
myObject.prototype.age = 23;
```

⇓

```
function myObject(name) {
  this.name = name;
  this.getName = function() { return "Name : " + this.name  }
  this.age = 23;
}
```

Poznámky :

- Inverzným refaktorovaním možno docieliť presunutie vlastnosti z konštruktora do kódu programu. Z praktického hľadiska však toto refaktorovanie väčšinou len zhorší štruktúru kódu a orientáciu v ňom.
- Podobným spôsobom je možné uvažovať o refaktorovaní, ktoré modifikuje a odstraňuje vlastnosť objektu. Pri odstraňovaní vlastnosti objektu v kóde, pomocou kľúčového slova `delete`, sa refaktorovaním odstráni vlastnosť z konštruktora. V tomto prípade je však nutné, aby vlastnosť nebola používaná medzi konštruktorom a príkazom `delete`.

<u>Refaktorovanie 11 : Nahradenie prototypu definíciou vlastností</u>

Motivácia : Ak prototyp objektu prestáva plniť svoj účel a príliš umelo definuje vlastnosti objektov, je vhodné ho nahradiť priamym definovaní vlastností v konštruktore objektu. Tým sa dosiahne aj zjednodušenie prototype chain, čo vedie k jasnejšej a jednoduchšej hierarchickej štruktúre.

Vstup : Konštruktor objektu a prototyp tohto objektu, ktorý bude dosadený do konštruktora.

Podmienky : Je nutné brať ohľad na celý prototype chain nad prototypom objektu. Na zjednodušenie je vhodné použiť refaktorovanie len nad prototypom bez rodičovského prototypu. Druhou alternatívou je okrem doplnenia vlastností, doplniť aj rodičovský prototyp ako prototyp objektu.

Príklad použitia :

```
function protObject() {
  this.name = „Meno";
}

function myObject(age) {
  this.age = age
  this.prototype = new protObject();

}
```

⇓

```
function myObject(age) {
  this.age = age
  this.name = „Meno";

}
```

Poznámky :

- Inverzné refaktorovanie extrahuje vlastnosti objektu z konštruktora do samostatného objektu, ktorý bude prototypom. Motivácia tohto inverzného refaktorovania je podobná všeobecnému refaktorovaniu „Extract superclass".

Automatizácia refaktorovaní prototypového modelu

Pri úvahách o automatizácií refaktorovaní prototypového objektového modelu táto práca dospela k nasledovným dôležitým tvrdeniam :

- Problém, či vlastnosť z konštruktora objektu bude/nebude patriť objektu aj v konkrétnom stave počas behu programu, je **nerozhodnuteľný**. (Možno dokázať redukciou na problém zastavenia : Ak by sa vedelo, či na konci programu patrí vlastnosť ku objektu, vedelo by sa aj to, že program zastaví.)
- Problém, či objekt bude mať prototyp, definovaný v konštruktore, aj v konkrétnom stave počas behu programu, je **nerozhodnuteľný**. (Keďže prototyp je tiež vlastnosť objektu, možno dokázať redukciou na predchádzajúci problém.)

Na základe týchto tvrdení nie je možné vytvoriť automatizovaný nástroj na podporu refaktorovaní prototypového objektového modelu, ktorý umožňuje dynamickú modifikáciu množiny vlastností objektu a hierarchie modelu.

Prvý problém naviac hovorí, že nie je možné povedať, či bude daná vlastnosť odstránená a teda či je výlučne „statického" charakteru, alebo sa s ňou manipuluje počas behu programu. Obmedzenie refaktorovaní len na „statické" vlastnosti definované v konštruktore teda tiež nie je možné automatizovať. Je to možné len v prípade, ak je si vývojár úplne istý, že v programe nie sú vlastnosti dynamicky modifikované a odstraňované.

Guru

Pri refaktorovaní prototypovo-založených jazykov je z dôvodu úplnosti vhodné spomenúť už existujúci softvérový nástroj Guru pre jazyk Self. Prvoradou snahou tohto nástroja je reštrukturalizácia hierarchie objektov (tzv. instance hierarchy) systému alebo jeho časti pri zachovaní správania objektov ([GUR95]). Vlastnosti objektov sú uložené v pamäti v tzv. slotoch a dedia sa prostredníctvom hierarchie objektov. Guru sa snaží minimalizovať počet použitých slotov pri zachovaní vlastností všetkých objektov.

Nevýhodou tohto nástroja, ktorý je však stále ešte vo vývoji, je radikálna zmena štruktúry hierarchie, ktorá je síce optimálna a štruktúrovaná, ale dosť odlišná od pôvodnej. V rámci hierarchie sa totiž vytvoria úplné nové objekty, čo môže zhoršiť orientáciu v systémoch, ktoré sú do istej miery už známe. Naviac, ako sa dá očakávať, Guru nepodporuje

dynamické manipuláciu s vlastnosťami počas behu programu (reflexive code). Spôsob implementácie, pri ktorej je celá hierarchia odstránená a znovu vybudovaná, však môže pomôcť v snahe o vytvorenie softvérovej podpory refaktorovaní v JavaScripte, založenom tiež na prototypoch.

4.4 Testovanie refaktorovaní JavaScriptu

Proces refaktorovania je úzko spojený s testovaním. Pri každom refaktorovaní, či už manuálnom alebo automatickom, dochádza ku zmenám v programovom kóde, ktoré pri nevhodnom použití môžu zaniesť do systému neželateľné chyby.

Pri refaktorovaní jazyka JavaScript je preto testovanie tiež veľmi dôležité. Pri každej zmene je vhodné otestovať modifikovaný kód. To umožní rýchlo identifikovať potencionálne problémy už v ich počiatkoch. Ak totiž test po použití refaktorovania neprejde, je zrejmé, že refaktorovanie nebolo aplikované správne. Na pomoc pri testovaní JavaScriptu je možné použiť vlastné testovacie rutiny alebo využiť nástroj JsUnit. Tento testovací framework je postavený na nástroji JUnit, ktorý slúži na testovanie jednotiek kód v jazyku Java. Viac o JsUnit je možné nájsť na [JSU05].

Z hľadiska automatických a manuálnych refaktorovaní je dôležitá kontrola vývojára nad vykonávanými zmenami. Túto kontrolu možno do veľkej miery dosiahnuť vývojom v špecifických IDE vývojových prostrediach, ktoré umožňujú zobrazovať a simulovať kód JavaScriptu a DHTML. Ak tieto prostredia naviac obsahujú aj automatické nástroje na refaktorovanie a nástroje na testovanie ako JsUnit, riziko nesprávneho použitia refaktorovaní a zanesenia chýb je minimálne. Automatizácia refaktorovaní a súvisiacich oblastí v procese vývoja softvéru a webových stránok je preto veľmi žiadaná.

4.5 Pohľad do budúcnosti

Oblasť refaktorovania v jazykoch ako sú Java či C++ dosiahla zaslúžený úspech. Boli preskúmané možnosti refaktorovania, ktoré sú ešte stále zjemňované a rozširované a čo je dôležité, boli vytvorené softvérové nástroje na podporu refaktorovaní automatickým spôsobom. Tieto funkcie refaktorovaní boli začlenené do IDE prostredí, čo umožňuje lepšiu spoluprácu s nástrojmi na vývoj a testovanie.

Táto práca analyzovala možnosti refaktorovania v prostredí skriptovacieho jazyka JavaScript s prototypovým objektovým modelom a načrtla možnosti, oblasti a problémy refaktorovania tohto jazyka. V budúcnosti by mali byť tieto refaktorovania detailnejšie preskúmané a rozširované o nové užitočné transformácie programového kódu. Výsledky by mali byť zhrnuté do katalógu refaktorovaní s podrobným popisom a podmienkami použitia transformácii.

Vrcholom práce na refaktorovaní JavaScriptu by malo byť vytvorenie softvérového nástroja a jeho začlenenie do IDE vývojových prostredí, a to hlavne v oblasti vývoja webových stránok. Automatický nástroj by mal byť podporený framework-om na testovanie vykonaných zmien. Ako užitočné sa ukazuje tiež rozšírenie IDE prostredí o možnosti vizualizácie prepojení JavaScriptu s objektami a jazykmi hostiteľského prostredia, čo pomáha pri pochopení vzťahov a štruktúry výsledného website.

Výsledkom by malo byť vývojové prostredie, ktorý komplexne podporuje vývoj softvéru a websites v jazyku JavaScript, od analýzy a návrhu až po testovanie refaktorovaní. Umožňuje rýchlu orientáciu v systéme a s využitím refaktorovaní poskytuje ľahký spôsob modifikácie programu pri zachovaní kvalitnej štruktúry a návrhu kódu.

5. Záver

Refaktorovanie je procesom vývoja a údržby softvérových systémov, pri ktorom dochádza k zmene štruktúry programového kódu, za zachovania jeho pôvodného správania a funkcionality.

Táto práca ako zatiaľ jediná skúmala možnosti a význam aplikácie poznatkov z oblasti refaktorovania objektovo-orientovaných jazykov nad skriptovacím jazykom JavaScript, používaným hlavne v prostredí internetových prehliadačov a webových serverov.

Refaktorovanie je úzko závislé od jazyka nad ktorým je aplikované a do veľkej miery súvisí s jeho syntaktickými pravidlami. Vďaka podobnosti syntaxe JavaScriptu s Javou a C++ bolo zistené, že refaktorovania, ktoré sa týkajú premenných, atribútov, funkcií a metód možno priamočiaro alebo s menšími modifikáciami použiť aj na refaktorovanie JavaScript kódu. Na úrovni objektov a ich hierarchie však už použitie všeobecných refaktorovaní naráža na problémy. Ako bolo v práci zistené, tie súvisia s odlišnosťou objektového modelu jazyka JavaScript. Prototypový objektový model umožňuje dynamické pridávanie a modifikáciu vlastností objektov za behu programu, čo spôsobuje problémy pri refaktorovaní a sťažuje jeho prípadnú automatizáciu. V prípade, že sú však vlastnosti priradzované objektom len „statickým" spôsobom v rámci konštruktorov, je možné refaktorovania aplikovať a to aj napriek chýbajúcim syntaktickým konštrukciám dedičnosti a viditeľnosti.

Pri skúmaní spôsobov refaktorovania JavaScriptu, integrovaného v rámci DHTML a webových dokumentov, bolo identifikované množstvo úprav, ktoré významným spôsobom zlepšujú štruktúru kódu celého systému webových stránok a HTML stránok samotných. Z hľadiska dobrej štruktúry webových dokumentov bola zistená dôležitosť separácie jednotlivých jazykov HTML stránok do častí s presne definovaným rozhraním. To umožňuje ľahšiu modifikáciu a znovupoužitie častí kódu a poskytuje väčší prehľad o štruktúre stránky a jej častí. Okrem výskumu nových refaktorovaní v rámci DHTML boli analyzované možnosti refaktorovania súvisiace s prototypovým objektovým modelom. Hranice použitia refaktorovaní objektového modelu boli stanovené na statické vlastnosti objektov. Na manipuláciu s dynamickými vlastnosťami a hierarchiou prototype chain a ich úpravu boli navrhnuté špeciálne refaktorovania.

Práca tiež ukázala dôležitosť automatickej podpory refaktorovaní JavaScriptu a následného testovania vykonaných zmien. Pri výskume automatizácie však bolo dokázaná nerozhodnuteľnosť problémov súvisiacich s dynamickými vlastnosťami prototypového objektového modelu, čo potvrdzuje zložitosť pri implementácií automatickej podpory refaktorovaní.

Na záver práce bol načrtnutý vývoj refaktorovania JavaScriptu do budúcnosti, ktorý predpokladá detailnejšie preskúmanie a rozšírenie vytvoreného katalógu refaktorovaní tohto jazyka. Tento katalóg by mal byť podkladom pre finálnu tvorbu automatického softvérové nástroja na podporu refaktorovaní, ktorý by mal byť začlenený priamo do vývojových prostredí.

6. Referencie

Táto príloha obsahuje zoznam použitej literatúry vrátane odkazov na webové stránky. Pre ľahšiu orientáciu sú zdroje štruktúrované a abecedne zoradené do troch častí podľa ich zodpovedajúcej oblasti :

- JavaScript
- HTML, CSS
- Refaktorovanie

JavaScript

[ECMA99] Standard ECMA-262, ECMAScript Language Specification (3rd edition, December 1999) :
http://www.ecma-international.org/publications/standards/Ecma-262.htm

[JSK04] Creating custom objects in JavaScript
http://javascriptkit.com/javatutors/object.shtml
The prototype object of JavaScript
http://www.wsabstract.com/javatutors/proto.shtml

[JSU05] Hieatt, Edward : *JsUnit Testing framework.* Webstránka projektu.
http://www.edwardh.com/jsunit/

[FLAN98] Flannagan, David: *JavaScript: The Definitive Guide*, Second Edition; O'Reilly, ISBN: 1-56592-392-8, Júl 1998, 790 str.

[KLD03] Lindsey, Kevin: Implementácia dedičnosti v JavaScripte, 2003
http://www.kevlindev.com/tutorials/javascript/inheritance/

[KITJS97] Kitchen, Andrew: *JavaScript Guide*
http://www.cs.rit.edu/~atk/JavaScript/manuals/jsguide/

[KITOBJ97] Kitchen, Andrew: *Object Hierarchy and Inheritance in JavaScript*, 1997
http://www.cs.rit.edu/~atk/JavaScript/manuals/jsobj/index.htm

[MCK03] Koss, Mike: *Object-Oriented Programming with JavaScript*, 2003
http://www.mckoss.com/jscript/object.htm

[RFB01] Frishberg, Ryan: *JavaScript Object-Oriented Programming* (časť 1 a 2), 2001
http://www.sitepoint.com/article/oriented-programming-1/2

http://www.sitepoint.com/article/oriented-programming-2

[TSC04] Scarfe, Tim: *JavaScript and Object Oriented Programming*, 2004

http://wsabstract.com/javatutors/oopjs.shtml

[W3CDOM] World Wide Web Consortium – Document Object Model

http://www.w3c.org/DOM/

[YTS01] Shiran, Yehuda; Shiran Tomer: *Object-Oriented Programming with JavaScript*, 2001

http://www.webreference.com/js/column79/

HTML, CSS

[MUKE97] Musciano, Chuck a Kennedy, Bill : *HTML: The Definitive Guide*, Second Edition; O'Reilly, ISBN: 1-56592-235-2, Máj 1997, 552 str.

[SLF] SELFHTML – webstránka (nemecky, anglicky, francúzsky)

http://www.selfhtml.org/

[W3CCSS] World Wide Web Consortium – Cascading StyleSheets

http://www.w3c.org/Style/CSS/

[W3CHTML] World Wide Web Consortium – HTML

http://www.w3c.org/MarkUp/

Refaktorovanie

[BECK99] Beck, Kent : *Extreme Programming Explained : Embrace Change*. Addison-Wesley, 1999

[ECL05] Webová stránka open-source IDE projektu Eclipse.

http://www.eclipse.org

[FOW99] Fowler, Martin; Beck, Kent; Brant, John; Opdyke, William; Roberts, Don : *Refactoring : Improving the Design of Existing Code*, Addison-Wesley, Jún 1999.

[FOW01] Fowler, Martin : *Crossing Refactoring's Rubicon.* Február 2001.

http://www.martinfowler.com/articles/refactoringRubicon.html

[FOWW] Stránka Martina Fowlera venovaná refaktorovaniu – Refactoring Home Page.

http://www.refactoring.com

[GRI91] Griswold, W.G. : *Program Restructuring as an Aid to Software Maintenance*. PhD. thesis, University of Washington, 1991.

[GUR95] Moore, Ivan : *Guru- A Tool for Automatic Restructuring of Self Inheritance Hierarchies.* Prentice-Hall, 1995.

[IDEA05] Webová stránka JetBrains a ich Java IDE prostredia IntelliJ IDEA.

http://www.jetbrains.com

[JBU05] Webová stránka vývojového IDE prostredia Borland JBuilder.

http://www.borland.com/jbuilder/

[JFA05] Webová stránka produktu JFactor od Instantiations.

http://www.instantiations.com/jFactor/

[JREF05] Seguin, Chris; Atkinson, Mike. Webová stránka Java projektu JRefactory pre IDE prostredia.

http://jrefactory.sourceforge.net/

[JUN03] Beck, Kent; Gamma Erich : *JUnit. A simple framework to write repeatable tests.* http://www.junit.org

[MUN05] Munro, M. J. : *Refactoring. Technical Report.* Department of Computer and Information Sciences, University of Strathclyde, 2005.

[OMG05] Object Management Group. *Unified Modeling Language*. Resource Page. http://www.uml.org/

[OPD92] Opdyke, William F. : *Refactoring Object-Oriented Frameworks.* PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[OPD97] Opdyke, William F. : *Object-Oriented Refactoring, Legacy Constraints and Reuse.* Proceedings of the 8th Workshop on Institutionalising Software Reuse (WISR), Ohio State University, Marec 1997.

[RB05] Brant, J. ; Roberts, D. Webová stránka projektu Refactoring Browser.

http://st-www.cs.uiuc.edu/users/brant/Refactory/RefactoringBrowser.html

[TOG05] Webová stránka CASE softvéru Together od Borlandu.

http://www.borland.com/together/

[WIK05] Wiki Wiki Web. *Wiki pages about Refactoring*, 2005.

http://c2.com/cgi/wiki?WikiPagesAboutRefactoring

[WU01] Wu, C.T. : *An Introduction to O-O Programming with Java.* McGraw-Hill, 2nd edition, 2001.

[XREF05] Vittek, Marián. Webová stránka projektu XRefactory – refaktorovacieho browsera pre Emacs a XEmacs.

http://www.xref-tech.com

7. Slovník pojmov a skratiek

Táto príloha obsahuje abecedne zoradený zoznam skratiek a pojmov, ktoré sa objavujú v rámci práce spolu s popisom ich významu.

7.2 Zoznam skratiek

Skratka	Význam skratky
ASP	Active Server Pages. Jeden z jazykov používaných pri server-side programming z dielne spoločnosti Microsoft. Používaný v rámci webového servera Microsoft IIS (Internet Information Services).
AST	Abstract syntax tree. Abstraktný syntaktický strom predstavuje všeobecnú štruktúru príkazov programového kódu, organizovanú do stromu. Tvorí hierarchickú reprezentáciu programového kódu. Je vstupom aj výstupom v procese refaktorovania.
CSS	Cascading StyleSheets. Štýly rozširujúce možnosti zobrazenia tagov v rámci HTML dokumentu. Umožňujú podrobne definovať farby, pozície, formát či zarovnanie tagov.
DHTML	Dynamic HTML. Koncept dynamického HTML dokumentu predstavujúci integráciu jazyka HTML, štýlov CSS a skriptovacieho jazyka ako JavaScript. Umožňuje dynamickú manipuláciu s elementmi HTML dokumentu, ich hodnotami, pozíciou či zobrazením.
DOM	Document Object Model. Objektový model HTML dokumentu, ktorý je zverejnený hostiteľským prostredím pre skriptovacie jazyky v DHTML.
DTD	Document Type Definition. Definícia XML jazyka, ktorá určuje povolené sady tagov s ich atribútmi a pravidlami vnárania.
HTML	HyperText Markup Language. Jazyk hypertextových webových dokumentov slúžiaci na prenos obsahu a formátu pomocou protokolu HTTP. Dokument v jazyku HTML je štruktúrovaný do hierarchie vnorených tagov. HTML je vytvorený v metajazyku SGML, ktorý je podobný XML.

HTTP	HyperText Transfer Protocol. Protokol na prenos hypertextových dokumentov v sieti Internet alebo Intranet.
IDE	Integrated Development Environment. Integrované vývojové prostredie obsahuje nástroje podporujúce rôzne časti procesu vývoja softvéru. Obsahuje napríklad nasledovné nástroje : editor programového kódu, nástroj na spúšťanie kódu, ladenie, testovanie a modelovanie či nástroj na refaktorovanie.
JS	JavaScript (alebo JScript). Skriptovací jazyk, ktorý je interpretovaný hostiteľským prostredím a slúži hlavne na manipuláciu s jeho objektami.
JSP	Java Server Pages. Jeden z jazykov používaných pri server-side programming z dielne Sun Microsystems. Predstavuje nadstavbu nad Java Servlets vo forme bližšej ku HTML pomocou JSP tagov.
OO	Objektovo-orientovaný. Založený na objektoch, ktoré majú svoje vlastnosti, t.j. atribúty a metódy vykonávajúce istú činnosť.
OOP	Objektovo-orientované programovanie. Rieši problém jeho modelovaním prostredníctvom objektov a ich vzájomných vzťahov.
PHP	Hypertext Preprocessor. Jeden z jazykov používaných pri server-side programming.
SGML	Standard Generalized Markup Language. Metajazyk, ktorý slúži na definovanie jazykov s tagovou štruktúrou. Jazyk HTML je definovaný pomocou SGML. SGML je predchodcom XML.
XML	Extensible Markup Language. Metajazyk, slúžiaci na definovanie jazykov s tagovou štruktúrou, ktoré poskytujú možnosti výmeny informácií. XML definuje jednoduchým spôsobom štandardy na výmenu údajov. Štruktúra jazykov XML je definovaná v rámci DTD.
XP	eXtrémne programovanie (Extreme programming). Proces vývoja softvéru, ktorý kladie dôraz na jednoduchosť vývoja a aktivity ako testovanie, refaktorovanie či programovanie v dvojici.
W3C	World Wide Web Consortium. Organizácia, ktorá má na starosti štandardizáciu a vývoj jazykov a formátov používaných v sieti Internet, ako napríklad HTML, XML, CSS či DOM.

7.3 Slovník pojmov

Pojem	Význam pojmu
C++	Objektovo-orientovaný programovací jazyk. Následník jazyka C.
cookies	Pomenované hodnoty, ukladané pri prehliadaní webových stránok na strane klienta. Slúžia na uchovávanie preferencií klienta a uloženie stavu pri prechode medzi stránkami.
dedičnosť	Vlastnosť objektovo-orientovaných jazykov, umožňujúca objektom podtried prevziať vlastnosti definované v nadtriede.
HTML tag	Element jazyka HTML, ktorý predstavuje prvky HTML dokumentu, ako napríklad text, hyperlinkový odkaz alebo pole formuláru. Každý typ tag má sadu atribútov, ktorých hodnoty určujú formátovanie a zobrazovanie tagu. Každý tag pozostáva zo začiatočného tagu (napr. `<body>`) a ukončovacieho tagu (`</body>`). Medzi týmito tagmi sa môže nachádzať hodnota alebo iný HTML tag. Skrátený zápis oboch tagov je napr. `<img />`.
enkapsulácia	Zapúzdrenie. Vlastnosť objektovo-orientovaných jazykov, ktorá umožňuje zoskupiť vlastnosti podobného charakteru pod objekt, ku ktorému sa viažu. Umožňuje tiež skryť tieto vnútorné vlastnosti objektov vonkajšiemu svetu alebo ich prípadne zverejniť prostredníctvom pevne definovaného a kontrolovaného rozhrania.
frame	Časť zloženého HTML dokumentu, do ktorej sa načítava webová stránka.
frameset	Definuje rozloženie frame-ov v rámci zloženého HTML dokumentu. Do každého frame-u je možné načítať iné stránky a s pomocou JavaScriptu je ich možné aj dynamicky meniť.
Java	Objektovo-orientovaný programovací jazyk z dielne spoločnosti Sun Microsystems. Je to platformovo-nezávislý jazyk kompilovaný do tzv. bytecode. Syntaxou podobný jazyku C++.
konštruktor	Funkcia jazyka JavaScript, ktorá slúži na vytváranie nových objektov. V rámci konštruktora sa definujú počiatočné atribúty a metódy objektu. Nový objekt sa vytvára volaním funkcie konštruktora pomocou kľúčového slova `new`.
legacy softvér	Softvér predchádzajúcich generácií, ktorý je stále nasadený a funkčný v prostredí podnikových systémov. Obyčajne napísaný v zastaralých programovacích jazykoch a so zlou štruktúrou, ktorá je spôsobená množstvom úprav pri

	údržbe. Tieto systémy je väčšinou vhodnejšie nahradiť alebo prepísať namiesto refaktorovania.
open-source softvér	Softvér s voľne šíriteľným zdrojovým kódom, ktorý je možné študovať, modifikovať a prispôsobovať vlastným potrebám.
override	Predefinovanie implementácie vlastností objektu v podobjekte (podtriede).
polymorfizmus	Vlastnosť objektovo-orientovaných jazykov, umožňujúca použiť ľubovoľný objekt tam, kde sa očakáva objekt jeho nadtriedy.
prototyp	„Rodičovský" objekt v objektovej hierarchii prototypovo-založených jazykov, ktorý slúži ako šablóna pre daný objekt. Objekt nepriamo dedí všetky vlastnosti svojho prototypu.
prototype chain	Reťaz, ktorá vznikne hierarchiou na základe vzťahu objekt a jeho prototyp. Ak sa pri odkazovaní nenájde vlastnosť priamo v objekte, prehľadáva sa postupne celá prototype chain až kým sa vlastnosť nenájde alebo sa dosiahne prototyp, ktorý už nemá „rodičovský" prototyp.
server-side programming	Programovanie na strane servera. Predstavuje spôsob programovania, kedy sa na HTTP požiadavky vykonáva kód v príslušnom jazyku, ktorý automaticky generuje HTTP odpoveď, väčšinou vo forme HTML dokumentu.
Smalltalk	Objektový jazyk. Na rozdiel od objektovo-orientovaných jazykov je v Smalltalk-u úplne všetko vytvorené a reprezentované ako objekt. Uplatnenie hlavne v akademickej oblasti.
software reuse	Znovupoužitie softvéru, prípadne jeho častí - komponentov. Odstraňuje nutnosť riešenia problému, ktorý už bol vyriešený a tým zrýchľuje vývoj softvéru.
ukončovací tag	Viď. „HTML tag"
vlastnosti objektu	(angl. properties, members), atribúty a metódy objektu
website	Systém navzájom prepojených webových stránok.

Príloha A : Bad Smells

Táto príloha obsahuje zoznam tzv. „Bad Smells" – oblastí kódu so zlou štruktúrou, nad ktorými je vhodné aplikovať refaktorovania. Zoznam je kompletný tak, ako ho načrtli Fowler a Beck ([FOW99], str. 75-88).

Potencionálna oblasť ("Bad Smell")	Popis
Duplikovaný kód	hlavný dôvod na refaktorovanie
Dlhá metóda	pozostatok procedurálneho programovania
Veľká trieda	jediná trieda toho vykonáva príliš veľa
Veľa parametrov metódy	nie je už potrebné pri práci s objektami
Rozmanité zmeny ("Divergent Changes")	jedna trieda je často menená rôznymi spôsobmi z rôznych dôvodov
"Shotgun Surgery"	zmeny majú vplyv na veľké množstvo tried a metód
Závisť funkcií ("Feature Envy")	príliš veľký záujem o dáta iného objektu
Zhluky dát	dáta, ktoré sú všade používané spoločné sú dobrým dôvodom na vytvorenie vlastnej triedy
Posadnutosť primitívnymi dátovými typmi	vhodné použiť triedy okrem/namiesto primitívnych dátových typov
Príkazy "switch"	OO programovanie má iné spôsoby, ako vykonávať akcie na základe typu/hodnoty
Paralelné hierarchie inštancií	ak pridanie podtriedy jednej triedy si vyžaduje pridanie podtriedy inej; častokrát nevhodné.
"Lenivá" trieda	trieda, ktorá nezarába na seba dosť (nerobí veľa), by mala byť odstránená
Špekulatívna generalizácia	nie je vhodné investovať príliš veľa do flexibility pre budúcnosť
Dočasná premenná	komplikuje štruktúru svojou prítomnosťou a pritom nie je používaná často
Postupnosti správ ("Message Chains")	ak klient chce objekt od iného objektu, atď.; vhodné priamo spojiť

Prostredník (“Middle Man”)	mal by byť odstránený, ak jediné čo robí, je sprostredkovanie
Nevhodná intímnosť	ak sú iným triedam zbytočne zverejnené skromné premenné a metódy; vhodné skryť
Podobné triedy s rôznymi rozhraniami	úpravami je vhodné zlúčiť tieto triedy a ich funkcie
Nekompletná trieda v knižnici	častokrát musí byť rozšírená o požadovanú funkcionalitu
Dátová trieda	mala by byť doplnená o dodatočné funkcie na prácu s jej dátami, ktoré zvýšia jej dôležitosť
Odmietnuté dedičstvo	ak podtriedy využívajú len málo z toho, čo zdedia od rodičov
Komentáre	komentár je vhodné miesto na povedanie, PREČO bolo niečo urobené a nie ČO sa urobilo

Príloha B : Základné objekty DOM

Nasledujúca tabuľka obsahuje zoznam základných objektov modelu DOM spolu s príkladmi ich použitia. Pre úplné informácie o objektovom modeli dokumentu DOM vyhľadajte [W3CDOM], [SLF] alebo [FLAN98].

Objekt DOM	Popis
window	Predstavuje okno prehliadača a jeho súčasti (ako sú menu, plocha a jednotlivé nástroje). Medzi niektoré z hlavných funkcií patria : otvorenie nového okna, zmena veľkosti okna, výpis správ na obrazovku či posun po okne. Najvrchnejší objekt hierarchie modelu.
	Príklady použitia v JavaScripte

	Otvorenie nového okna a prepnutie doň :
	`noveOkno = window.open( "inaStranka.htm", "Druhé okno", "width=300, height=200, scrollbars );`
	`noveOkno.focus();`
	
	Výpis správy na obrazovku :
	`window.alert("chyba!");`

frames	Umožňuje prístup k frameset-u a jeho frame-om (do ktorých sa načítavajú stránky) v rámci okna prehliadača.
	Príklady použitia v JavaScripte
	Načítanie novej stránky do prvého frame-u : `parent.frames[0].location.href = "druhaStranka.htm";`
document	Predstavuje HTML dokument načítaný v okne prehliadača. Umožňuje prístup k vlastnostiam tohto dokumentu (ako napr. URL adresa, či cookies) a k jeho jednotlivým HTML elementom. Umožňuje meniť vlastnosti existujúcich elementov a dokonca vytvárať nové.
	Príklady použitia v JavaScripte

Vytvorenie dokumentu s nastavením znakovej sady :

```
document.charset = "iso-8859-2"
document.open();
document.write("<b>Tučný text</b>");
document.close();
```

Vytvorenie nového elementu s obsahom a jeho priradenie pod iný element :

```
<div id="plocha"></div>
<script type="text/javascript">
<!--
  var novyElem = document.createElement("H1");
  var text = document.createTextNode("Nadpis");
  novyElem.appendChild(text);
  var plochaDiv = document.getElementById("plocha");
  plochaDiv.appendChild(novyElem);
//-->
</script>
```

node

Zodpovedá HTML elementu (uzlu). Poskytuje možnosti prístupu a manipulácie s atribútmi elementu a ich hodnotami a funkcie prechodu na nasledujúce HTML elementy.

Príklady použitia v JavaScripte

Vypísanie hodnoty poduzla :

```
<h1 id="nadpis">Nadpis s <i>kurzívou</i></h1>
<script type="text/javascript">
<!--
  var uzol = document.getElementById("nadpis");
  var hodnotaPodUzla = uzol.firstChild.nodeValue;
  document.write(hodnotaPodUzla); // vypíše „Nadpis s"
//-->
</script>
```

style

Objekt zodpovedajúci CSS štýlom jednotlivých elementov. Umožňuje nastavovanie atribútov zobrazenia CSS ako farba, pozícia, či štýl textu.

Príklady použitia v JavaScripte

Nastavenie farieb, pozadia a okraja elementu :

```
<p id="odsek">Toto bude odsek so štýlom</p>
<script type="text/javascript">
<!--
  var uzol = document.getElementById("odsek");
  uzol.style.color = „#FF0000"; // červená farba
  uzol.style.fontSize = „36pt";  // veľké písmo
  uzol.style.border = „solid red 10px"; // okraj
  uzol.style.backgroundColor = „#FFFFFF"; // biela
//-->
</script>
```

<table>
<tr><td rowspan="3">forms</td><td>Pole, ktoré zodpovedá všetkým formulárom v rámci HTML dokumentu. Podobným spôsobom sú zverejnené elementy formulárov (objekt <code>elements</code>), obrázky (<code>images</code>), aplety (<code>applets</code>) či hyperlinky (<code>anchors</code>).</td></tr>
<tr><td>Príklady použitia v JavaScripte</td></tr>
<tr><td>Nastavenie akcie (cieľovej stránky) pre prvý formulár a jeho odoslanie :
<code>document.forms[0].action = "spracovanie.jsp";</code>
<code>document.forms[0].submit();</code></td></tr>
<tr><td rowspan="3">history</td><td>Zodpovedá zoznamu navštívených stránok. Umožňuje presun dozadu a dopredu po týchto stránkach.</td></tr>
<tr><td>Príklady použitia v JavaScripte</td></tr>
<tr><td>Presun o tri stránky dozadu a následne o jednu dopredu :
<code>window.history.go(-3);</code>
<code>window.history.forward();</code></td></tr>
<tr><td rowspan="3">location</td><td>Predstavuje informácie o adrese z ktorej bol načítaný HTML dokument. Poskytuje všetky informácie z poľa prehliadača, do ktorého sa zadáva URL adresa. Umožňuje nastaviť novú adresu, čím sa načíta nový dokument</td></tr>
<tr><td>Príklady použitia v JavaScripte</td></tr>
<tr><td>Načítanie nového HTML dokumentu :
<code>window.location.href = "inaStranka.htm";</code>

Získanie informácie o počítači, z ktorého je dokument :
<code>alert(window.location.hostname);</code></td></tr>
<tr><td rowspan="3">navigator</td><td>Zhŕňa informácie o prehliadači (hostiteľskom prostredí) ako napríklad meno, platforma, jazyk, verzia či podpora cookies.</td></tr>
<tr><td>Príklady použitia v JavaScripte</td></tr>
<tr><td>Výpis typu prehliadača a jeho verzie :
<code>alert(navigator.appName + " " + navigator.appVersion);</code></td></tr>
</table>