VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

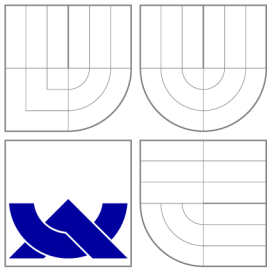
ANIMACE ALGORITMŮ V PROSTŘEDÍ FLASH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

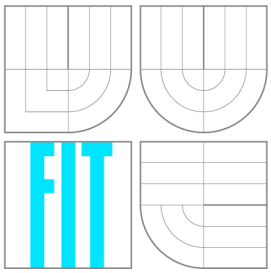
AUTOR PRÁCE
AUTHOR

DOMINIK MALČÍK

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ANIMACE ALGORITMŮ V PROSTŘEDÍ FLASH

ALGORITHM ANIMATION IN FLASH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DOMINIK MALČÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2009

Abstrakt

Hlavním cílem této práce je prodiskutovat a demonstrovat možnosti animace algortimů v rámci platformy Adobe Flash. Pro uvedení do problematiky je vypracován přehled technologií RIA včetně stručného zhodnocení celé oblasti. Jedna kapitola je věnována vizualizaci jako takové, aby poté mohla vzniknout vlastní implementace vizualizace pro zadanou platformu. Nechybí ani zhodnocení práce a motivace k dalšímu vývoji.

Abstract

The main goal of this bachelor's thesis is to discuss and demonstrate possibilities of an algorithm animation within Adobe Flash platform. The second chapter contains a RIA technologies survey including a short review of the whole area. Another chapter is devoted to visualization in general consequently followed by my own implementation. An evaluation of the entire work and motivation for further development can not be certainly missed out.

Klíčová slova

animace, vizualizace, animace algoritmů, vizualizace algoritmů, Flash, RIA

Keywords

animation, visualization, algorithm animation, algorithm visualization, Flash, RIA

Citace

Dominik Malčík: Animace algoritmů v prostředí Flash, bakalářská práce, Brno, FIT VUT v Brně, 2009

Animace algoritmů v prostředí Flash

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. RNDr. Pavla Smrže, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Dominik Malčík
19. května 2009

Poděkování

Především bych chtěl vyjádřit dík vedoucímu mé bakalářské práce, jímž byl pan doc. RNDr. Pavel Smrž, Ph.D, za rady, podněty a poskytnuté konzultace. Poděkování patří také autorům publikací, z nichž jsem čerpal.

Obsah

Kapitola 1

Úvod

V době počítačového věku, kdy řešení čím dál většího počtu úloh připadá právě počítačům, jsme zaplaveni množstvím algoritmů. Ať už původních, z dob, kdy vlastnit počítač bylo údělem pouze několika skutečně velkých organizací, nebo moderních, optimalizovaných pro co možná nejefektivnější využití zdrojů. A právě takové moderní, optimalizované, rychlé a úsporné algoritmy bývají často výsledkem vědecké práce celých týmů odborníků. Jejich stručná dokumentace se často nevměstná ani do desítky stránek odborného textu plného termínů a výrazů, s nimiž se v mnohých případech setkáváme poprvé. K pochopení takových algoritmů je třeba vyměřit čas lépe ve dnech než v hodinách.

Naštěstí nemyslí moderní společnost pouze na vývoj postupů řešení rozličných úloh, stejným tempem se také rozvíjí možnosti prezentace dosažených výsledků a konečně i, což je pro nás z hlediska této práce nejzajímavější, samotných postupů. Možnostem animací takových postupů se chci věnovat v následujících kapitolách.

Ještě bych si dovolil Vás v rámci úvodního slova vnést stručně do pojednání jednotlivých částí této práce.

Kapitola 2 se zabývá oborem tzv. bohatých internetových aplikací. Jelikož tato sféra zažívá skutečný rozmach, dalo by se popisem všech možných způsobů vývoje takových aplikací zaplnit mnoho a mnoho listů, což ale není žádoucí, proto jsou zde popsány pouze nejznámější dostupné technologie a jejich klíčové vlastnosti.

Dále nasledující kapitola 3 přináší pohled do světa vizualizace, abychom získali alespoň základní povědomí o možnostech vizuální reprezentace informací. Možná si to už ani neuvědomujeme, ale jsme denně obklopeni různými formami vizualizace, proto se pokusím o rozdělení celé rozsáhlé oblasti na dílčí části a jejich následné přiblížení.

Nejzajímavější části vlastní implementace a také další publikace hodné postřehy z vývoje pro platformu Flash můžete prostudovat v kapitole 4. Snažil jsem se zacílit na specifická řešení záležitostí kolem vizualizace a naopak vypustit nudné, rutinní pasáže.

Před závěrečným shrnutím (kapitola 6) ještě zhodnotím výsledek celé práce (kapitola 5) a pokusím se nastínit možný směr vývoje.

Kapitola 2

RIA - Rich Internet Applications

2.1 Co je RIA

Pojem RIA *(Rich internet applications)* vznikl v 90. letech minulého století, kdy byla zaváděna nová rozšíření umožňující tvorbu vyspělých webových aplikací. RIA je tedy označení pro vyspělé webové aplikace vyznačující se značnou podobností s desktopovými programy. Použité informační zdroje: [5], [15], [22].

2.2 Historie vzniku

Jak a proč vůbec došlo k vzniku tohoto typu aplikací? Z historického hlediska se internet vyvinul z malé sítě pro vojenské a vědecké účely až v celosvětové médium, ke kterému má přístup téměř každý obyvatel této planety pocházející z moderní společnosti. Zpočátku stačilo lidem komunikovat prostřednictvím zasílání jednoduchých textových zpráv. Později byl na světlo světa přiveden web, který nebyl původně nikterak propracovaný, ale postupně se z něj stal více a více sofistikovaný nástroj schopný úhledně prezentovat v podstatě jakékoliv informace. Bohužel statický protokol HTTP omezoval možnosti interakce s uživatelem a přirozeně muselo dříve nebo později dojít k vývoji nových nástrojů, které stírají tato nepohodlná omezení a umožňují nám v prostředí webu vytvářet do jisté míry plnohodnotné aplikace.

2.3 Základní vlastnosti

Vzhled podobný tradičním desktopovým aplikacím umožňuje sladit grafické uživatelské rozhraní s tím, na co už je uživatel zvyklý, a tak zlepšit přívětivost aplikace směrem k uživateli, kdy není nutné, aby vždy nejprve zkoumal ovládání a až poté přemýšlel nad tím, čeho chtěl původně docílit.

Offline provoz dovoluje při splnění jistých podmínek využívat aplikaci i v případě dočasné nedostupnosti internetového připojení. Je-li aplikace schopna ukládat potřebná data a svůj stav na klientské stanici a se serverem komunikuje výhradně v případě potřeby, můžeme být i přes přerušení konektivity stále schopni provádět úlohy, které by v případě standardní webové aplikace nebylo možné vykonat.

Interakce s uživatelem je na vyšší úrovni než v klasických webových aplikacích, protože je možne reagovat v podstatě okamžitě na podněty uživatele adekvátním krokem.

Není třeba při jakékoliv sebemenší akci obnovovat celý obsah stránky a teprve v tomto momentu vyhodnocovat uživatelem provedené úkony. Díky této vlastnosti lze rovněž ušetřit množství přenášených dat mezi serverem a klientem, a tím také zrychlit celou komunikaci a odlehčit i tak dosti zatíženým linkám a serverům.

Nutnost instalace a následná údržba platformy mohou být považovány za drobnou vadu na kráse, i když ne u všech technologií, jenž nás ale zatíží tak minimálním způsobem, jelikož velké množství updatů probíhá naprosto automaticky, že v podstatě ani nelze hovořit o stinné stránce. Díky této malé povinnosti vznikají na druhé straně nepoměrně větší výhody – budou zmíněny později.

Nezávislost na prohlížeči a do jisté míry i na operačním systému, je bezesporu jednou z klíčových vlastností, která vyváží nutnost instalovat prostředí dané platformy před prvním spuštěním. Aplikace pak zpravidla vypadají stejně ať je prohlížíte v Safari na Mac OS, nebo například ve Firefoxu či Opeře na Windows, což se o klasickém webu ani vzdáleně říci nedá, každý slušný vývojář klasických internetových stránek nebo aplikací je tak nucen věnovat se problému jejich rozdílné interpretace nejen v různých operačních systémech, ba dokonce i v rámci jednoho operačního systému v různých prohlížečích.

Přístup ke zdrojům operačního systému není sice ještě zcela dokonalý jako v případě desktopových aplikací, ale jednoduše řečeno se dá téměř vždy dosáhnout nějakým postupem kýženého výsledku. Například pracovat se soubory na lokálním disku klientské stanice.

Výkon je u vhodně implementovaných RIA aplikací zpravidla vyšší z důvodu omezení kontaktu se serverem. Část operací se provádí pouze lokálně a na server se tak odesílá mnohem méně požadavků a informací. Tím se zvýší výkon nejenom samotné aplikace, ale také serveru, jelikož výpočetně náročné operace se zčásti odehrávají v režii klienta.

Rozšíření funkčnosti prohlížeče ve smyslu doplnění schopnosti, kterou prohlížeč nativně nedisponuje. Jedním příkladem za všechny může být přehrávač videa na fenomenálním webu YouTube.com, v současnosti využíván v různě obměněných verzích i na mnoha jiných webech.

Komplexnost frameworků umožňujících vývoj RIA aplikací nám umožňuje do jediné aplikace zahrnout v podstatě cokoliv a tím vytvořit naprosto kompletní, nezávislou aplikaci, což dříve nebylo možné. Bylo třeba kombinovat několik různých přístupů a mezi nimi laborovat s posílanými daty. Tím vznikal prostor pro zavlečení chyby do celého systému programů a také se mnohonásobně zvyšovala bezpečnostní rizika.

Přístupnost dat pro vyhledávače byla až donedávna kamenem úrazu velké části RIA platforem pro neschopnost vyhledávačů indexovat data zapouzdřená ve specifických formátech odlišných od standardních html, txt, xml, apod. dokumentů, čímž vznikal značný handicap pro provozovatele internetových stránek s takovými aplikacemi. V současné době je optimalizace pro vyhledávače dosti vyhledávanou službou a vzhledem k přesycení internetu miliardami informací je dobrá přístupnost webu pro vyhledávače základem úspěchu. Možná i díky tomu jsme se dočkali zdokonalení v oblasti indexace RIA formátů ať už ze strany vyhledávačů, nebo i samotných RIA platforem. Dnes se tedy situace obrací a RIA už nemusí být v tomto směru žádným problémem.

2.4 Seznámení s dostupnými technologiemi

Na poli RIA můžeme nalézt mnoho různých technologií, některé jsou si podobné, jiné se naopak snaží vybočit z řady. V podstatě každý z velkých hráčů chce mít své želízko v ohni. Tím vzniká určitý chaos a povinnost mít pro většinu platforem nainstalovaný nějaký doplněk či virtualmachine, což nemusí být pro uživatele úplně nejšťastnější. V následujících podkapitolách Vám přiblížím jednotlivé technologie, abych je mohl závěrem zhodnotit. Podotýkám, že se zde budu věnovat technologiím nyní dostupným a značně používaným. Nevylučuji tedy, ba naopak jsem si jist, že exitují jiné, méně proslavené technologie, jelikož není nemožné si framework za určitým účelem vyrobit vlastními silami. Takové platformy pro nás ovšem postrádají význam a není možné se všem v této práci věnovat.

2.4.1 Microsoft Silverlight, Mono Moonlight

Microsoft Silverlight [24] je plug-in vytvořený pro nejpoužívanější webové prohlížeče na platformách Windows, Mac OS, Windows Mobile (od verze 6) a Symbian (Series 60) založený na Windows Presentation Foundation (dále jen WPF). WPF umožňuje při tvorbě aplikací využívat moderní efekty, pracovat s vektorovou grafikou a s podporovanými typy multimédií jako např. MP3, WMA, WMV. Díky zabudovaným kodekům pro video ve vysokém rozlišení je WPF přímo předurčena pro práci s multimédii. Silverlight nám umožňuje velkou část WPF použít i v prostředí webu. Jednotlivé vývojové stupně Vám představím v následujících odstavcích. Ještě předtím bych ale chtěl upozornit na to, že v současnosti používanou verzí je Silverlight 2.0, jelikož verze 3.0 je stále ve vývoji a nyní máme k dispozici pouze betaverzi.

Nemohu opomenout také projekt Mono Moonlight [16] zaštiťovaný společností Novell za oficiální podpory Microsoftu. Ten se snaží vyplnit mezeru na trhu a přinést podporu pro Silverlight aplikace i do operačního systému Linux a dalších systémů založených na Unix/X11.

Silverlight 1.0. Je vůbec první verzí, která přináší základní funkčnost převzatou z WPF. Tato verze umožňuje kromě naprosto běžného zachytávání komunikace z vstupních zařízení a vykreslování bitmapové grafiky také práci s již zmíněnými multimediálními formáty a omezenou podporu pro DOM[1]. Podporovanými datovými formáty jsou XML a JSON. Pro programování aplikační logiky je užíván Javascript, reprezentaci uživatelského rozhraní a rozložení jeho prvků zajistí jazyk XAML[2] *(Extensible Application Markup Language)*.

Silverlight 2.0. Původně označován jako Silverlight 1.1 rozšiřuje zásadním způsobem možnosti psaní aplikační logiky. Je možné použít jazyky C#, VB.NET, Javascript, nebo IronRuby, to díky implementaci stejného Common Language Runtime[3] jako je v .NET Framework 3.0. Spolu se Silverlight je distribuována základní knihovna tříd *(BCL - Basic Class Library)*, která je logicky podmnožinou knihovny tříd z .NET Framework. Dalším krokem kupředu je nativní podpora ovládacích prvků. Pro představu mohu uvést např. `Button`, `ListBox`, `ProgressBar`, `Slider`, aj. Ještě bych zmínil nově přidanou omezenou podporu pro práci se souborovým systémem klientské stanice.

[1] http://www.w3.org/DOM/
[2] http://msdn.microsoft.com/en-us/library/ms752059.aspx
[3] http://msdn.microsoft.com/en-us/library/ddk909ch(VS.71).aspx

Silverlight 3.0 beta. Oficiálně vypuštěn v měsíci březnu tohoto roku, vylepšuje verzi předešlou rozšířením již zavedených vlastností a funkcí. Mezi hlavní vylepšení patří podpora multimediálních formátů H.264/AAC, offline aplikací a také SEO optimalizace.

Mono Moonlight. Implementace s otevřeným kódem umožňující spuštění Silverlight aplikací i na operačních systémech Linux a jiných na bázi Unix/X11. Současně s vypuštěním betaverze posledního Microsoft Silverlight byla uvedena i alfa verze Moonlight 2.0. Označení verzí Moonlight odpovídá značení Silverlight, tudíž Moonlight 1.0 by měl být plně kompatibilní se Silverlight 1.0 a analogicky verze 2.0, jejíž finální uvolnění je plánováno na září roku 2009, by měla implementovat funkce ze Silverlight 2.0. Můžeme si tedy povšimnout, že ve vývoji Moonlight došlo vzhledem k nynějšímu vypuštění již 3. verze Silverlight ke značnému zpoždění, a tudíž je komunita uživatelů systému Linux Microsoftem do jisté míry diskriminována.

2.4.2 OpenLaszlo

Zajímavou architekturu (viz obrázek 2.1) mají na svědomí tvůrci open-source platformy OpenLaszlo[4] [14], [11]. SDK OpenLaszlo se skládá z kompilátoru, Java servletu a JavaScriptové knihovny *(LFC – Laszlo Foundation Class)*.

Kompilátor –napsaný v Javě– se stará o překlad zdrojových textů z LZX jazyka do binárních souborů spustitelných v cílovém prostředí. Nyní podporovanými výstupními formáty jsou Flash (SWF) a DHTML.

Server je Java servlet zajišťující podporu pro SOAP[5], XML-RPC[6] a JavaRPC[7] požadavky. Dále je schopen např. překódovat různé formáty tak, aby byly spustitelné prostřednictvím Flash Playeru.

Knihovna LFC zahrnuje komponenty uživatelského rozhraní, data binding a síťové služby. Podporuje také tvorbu animací a layoutu.

Programování probíhá pomocí LZX programovacího jazyka, ten je v podstatě kombinací XML a JavaScriptu. S podobnými jazyky se můžeme setkat i u konkurence, např. v Microsoftu je využíván XAML jazyk nebo v Adobe používají MXML. Všechny zmíněné příklady –LZX, XAML, MXML– jsou deklarativní jazyky[8] založené na XML.

2.4.3 UltraLightClient

Švýcarská společnost Canoo přišla na trh s knihovnou postavenou na Java platformě zvanou UltraLightClient (ULC)[6], která mě bohužel hned v počátku poněkud odradila svou licenční politikou, i když autoři naopak ceny licencí vyzdvihují jako jeden z kladů. Abychom tomu správně rozuměli, licenci si musí zakoupit pouze vývojáři, pro koncové uživatele je ULC zcela zdarma. Zmíněná knihovna je obrazně řečeno mostem mezi standardním SWING[9]

[4] Zajímavý rozhovor s jedním z tvůrců OpenLaszlo platformy http://news.zdnet.com/2100-9593_22-139903.html

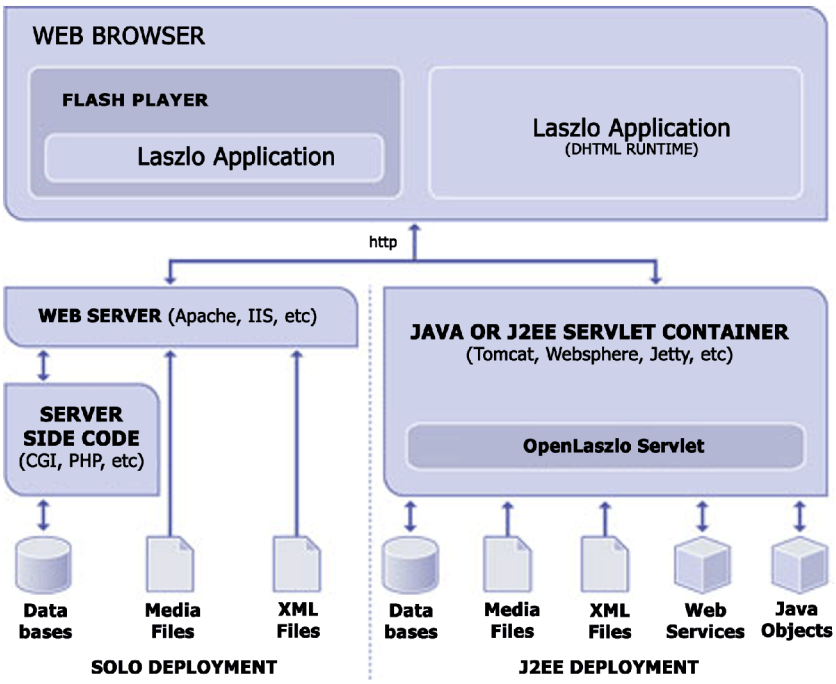
[5] http://www.w3.org/TR/soap/

[6] http://www.xmlrpc.com/

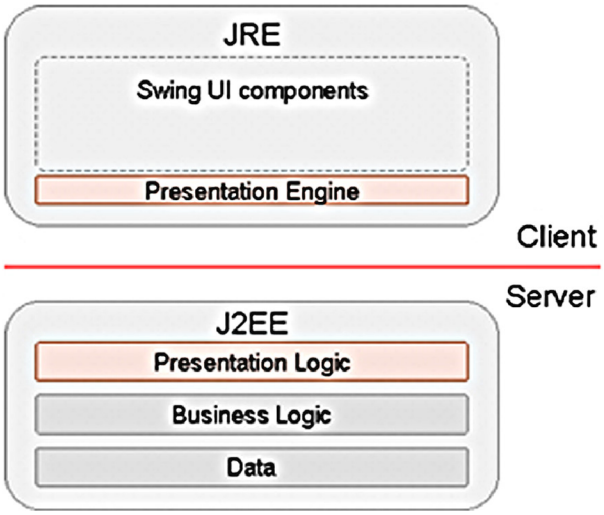
[7] http://www.openlaszlo.org/lps/docs/reference/javarpc.html

[8] http://logicaltypes.blogspot.com/2008/09/what-is-declarative-programming.html

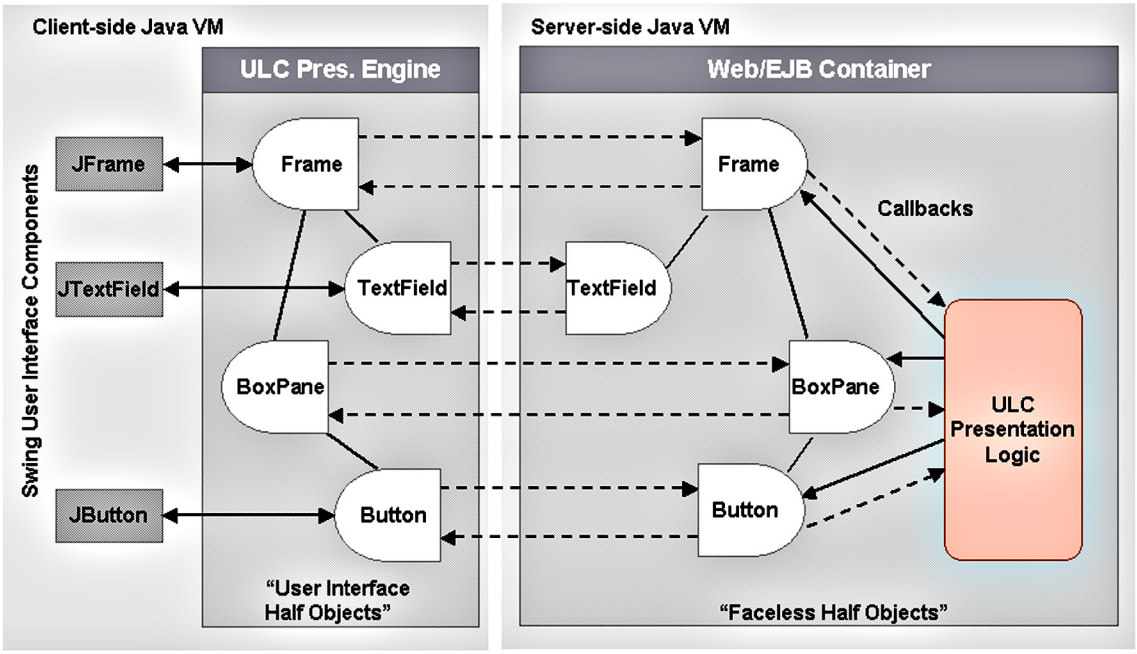
[9] http://java.sun.com/docs/books/tutorial/uiswing/



Obrázek 2.1: Architektura OpenLaszlo [14].



Obrázek 2.2: Architektura UltraLightClient poprvé [6].



Obrázek 2.3: Architektura UltraLightClient podruhé [6].

toolkitem, který umožňuje tvorbu GUI v desktopových Java aplikacích, a webovou architekturou. Zbytek architektury je bezplatně dostupný i pro vývojáře, takže je možné použít k produkci RIA Vaše oblíbené IDE (integrované vývojové prostředí).

Díky vhodnému rozdělení hlavního konceptu architektury (obrázek 2.2) na serverovou a klientskou část se přenáší jen velmi malé množství dat, protože objemné grafické prvky jsou uloženy v klientské části a ze serveru se přenáší téměř výhradně vlastní data aplikace - vysvětlení dobře dokresluje obrázek 2.3. Obzvlášť bych si dovolil vyzvednout myšlenku tzv. půlobjektů, jejichž znázornění si můžete povšimnout právě na zmíněném obrázku 2.3. Aplikační logika je naopak celá zapouzdřena v části serverové. Toto důmyslné rozdělení zaručuje relativně dobrou bezpečnost platformy.

Nasazení

Client-side. Prezentační část může být provozována jako applet nebo v rámci webového prohlížeče, je rovněž možné ji konfigurací přizpůsobit klientské stanici.

Server-side. Aplikace bývají typicky produkovány pro server. Jde o hlavní vývojový proud ULC.

Stand-alone. Výhodou je možnost použití aplikace jako samostatný celek. V tomto případě probíhá napodobení komunikace typu klient-server, tím je zachována kompatibilita kódu stand-alone aplikací s klient-server architekturou.

Ke spuštění aplikací je nutné mít nainstalované prostředí Java Runtime Environment[10] (JRE). Výsledné produkty a jejich vzhled jsou tedy spjaty s daným prostředím, a tudíž je možné dosáhnout stejného vzhledu a chování napříč operačními systémy i prohlížeči.

[10] http://java.sun.com/javase/6/docs/technotes/guides/index.html#jre-jdk

Myšlenka to ovšem není nijak revoluční vzhledem k tomu, že hlavní konkurenti používají podobné principy.

2.4.4 JavaFX

Dalším multiplatformním počinem na bázi Javy je –s příznačným názvem– JavaFX [19]. Stejně jako konkurenční nástroje nám nabídne podporu pro široké spektrum užití. Tudíž bych se zde věnoval spíše vlastnostem, které by mohly upoutat Vaši pozornost.

Mezi stěžejní lze zařadit snahu autorů o nezávislost na platformě *(tzv. write once and run anywhere)*. Základním předpokladem je nainstalovaná podpora prostředí Javy. O zbytek už není třeba se starat, protože JavaFX aplikace by měly být snadno nasaditelné jako desktop aplikace, aplikace spouštěné v prohlížečích nebo mobilních zařízeních. Intenzivně se pracuje také na dokončení podpory set-top boxů, herních konzolí, Blu-ray přehrávačů, aj. Této vlastnosti je dosaženo díky tzv. Common profile konceptu. Aplikace je vyvinuta na úrovni společného profilu pro všechny platformy a každá platforma pak doplní knihovny potřebné pro vlastní chod. Komplexnost přenositelnosti je završena možností jednoduše pomocí drag-and-drop přetáhnout aplikaci z prohlížeče na pracovní plochu a tím zapříčinit změnu typu produktu z webového na desktopový.

Možnost využít už dříve napsané knihovny v Javě snad nikoho nepřekvapí a je jen jakýmsi nutným doplněním komfortu a využitím dalšího možného benefitu, který plyne ze spojení napříč Java prostředím.

Na závěr bych ještě zmínil, abychom neopomněli tuto skutečnost, že vlastní zdrojový kód není viditelný pro koncové uživatele, narozdíl třeba od AJAXu, což přispívá k lepší bezpečnosti celé platformy.

2.4.5 AJAX

AJAX *(Asynchronous JavaScript and XML)* nemusím jistě nijak přehnaně představovat, jelikož se jedná o jednu z nejznámějších technologií na poli RIA. Podpora pro AJAX je implicitně zahrnuta snad ve všech moderních prohlížečích, leč ne vždy bývá stejný kód všude 100% funkční.

Programování aplikační logiky má na svědomí nejčastěji JavaScript, zatímco nosičem dat bývá zpravidla XML. AJAX striktně nevyžaduje JavaScript pro tvorbu skriptů, jako alternativa může posloužit třeba VBScript, stejně tak pro reprezentaci dat není XML jedinou cestou, vystačíme si například i s prostým textem nebo HTML. Výměna dat se serverem probíhá asynchronně, jak už vyplývá z názvu, prostřednictvím `XMLHttpRequest`.

Hlavní výhody AJAXu jsou zřejmé – nativní podpora ve velké části současných prohlížečů, a pak hlavně možnost aktualizovat jen určité části stránky bez nutnosti jejího kompletního reloadu. Z výhod ale plynou i nevýhody [23]. Tím, že nedochází k znovunačítání obsahu stránky, dochází ke zmatení prohlížečů a jejich integrovaných navigačních prvků, které s tímto konceptem neumějí správně pracovat. Může tak dojít k disfunkcím například u tlačítek zpět, vpřed, apod. Díky jednoduchosti celého konceptu nám vyvstává další neduh, protože zdrojové texty, často potem vykoupené, jsou na internetu vystavené jako ve výloze, a tudíž může naše revoluční myšlenky kdokoliv, kdo alespoň trochu zná tuto technologii, bezproblémů zcizit nebo okopírovat.

Existuje mnoho různých frameworků postavených právě na technologii AJAX, pro za-

jímavost jsem vybral několik povedených zástupců: Bindows[11], Pyjamas[12], Qooxdoo[13].

2.4.6 Google Web Toolkit

Google Web Toolkit (GWT)[10] je framework umožňující pohodlně tvořit v programovacím jazyce Java se všemi jeho výhodami a ve výsledku se vyhnout nutnosti instalace JRE pomocí kompilace projektu do JavaScriptu a HTML.

Základní kameny GWT [17]

- emulátor platformy Java
- kompilátor z Javy do JavaScriptu
- knihovny Java a GWT API
- tzv. „hosted“ prohlížeč

Google se snaží ve svém frameworku odstranit neduhy typické pro vývoj aplikací v JavaScriptu. Tím, že přesunul vývoj do moderního, plnohodnotného programovacího jazyka, přinesl uživatelům stejný komfort, jakého se dostává i ostatním Java vývojářům. Mohou bez problémů pracovat ve svém oblíbeném IDE, připojovat knihovny, které si již dříve vytvořili, a užívat moderní postupy. Další silnou stránkou GWT je propracovaný debugging, což je vzhledem k možnostem ladění JavaScriptových aplikací dost podstatné vylepšení. V neposlední řadě bylo také zapracováno na správě historie v internetových prohlížečích, aby tím byl eliminován známý neduh AJAXu – viz kapitola 2.4.5.

Vaše aplikace mohou pracovat buď ve *web* módu, nebo v *hosted* módu.

Web mód. Výsledná AJAX aplikace je interpretována ve webovém prohlížeči.

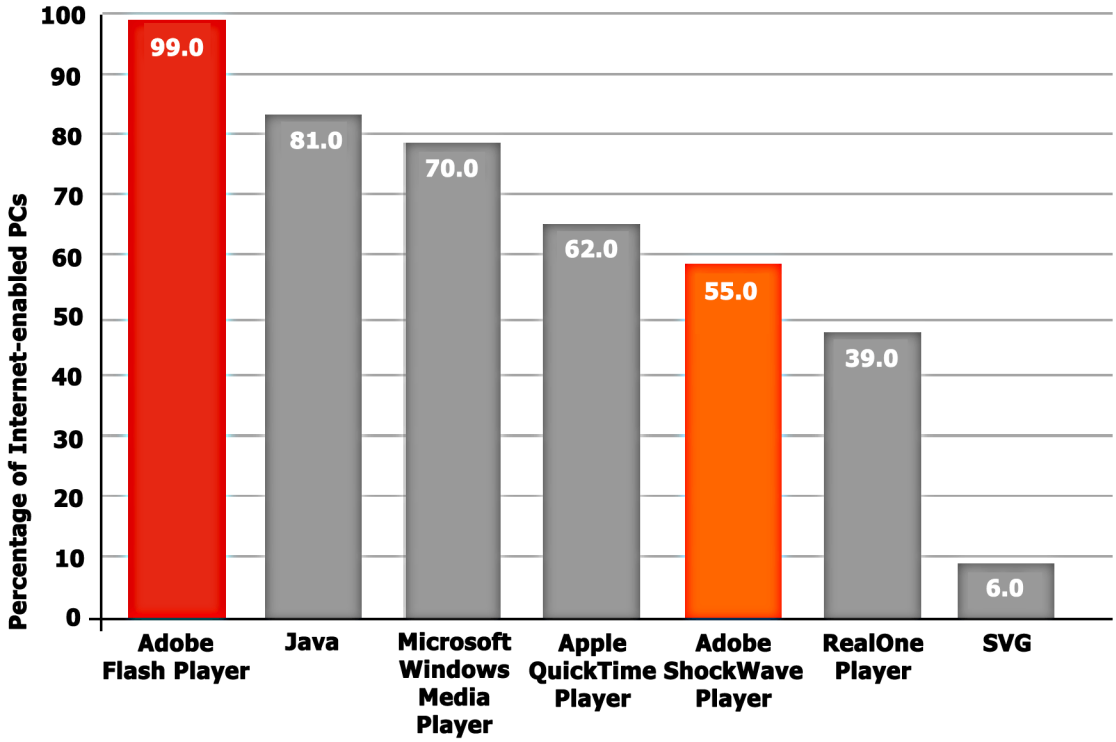
Hosted mód. Hosted prohlížeč umožňuje prohlížet a hlavně ladit Java aplikace ještě během vývoje.

Podobnými technologiemi jsou Pyjamas, kde je programovacím jazykem Python nebo RubyJS, kde, jak jistě z názvu tušíte, je základem Ruby. Obě tyto technologie si vzaly GWT za vzor, výsledkem je tudíž také AJAX.

2.4.7 RIA od Adobe

Vůbec nejrozsáhlejší základnu v oblasti RIA vytvořila společnost Adobe, o které mohu bez obav říci, že to myslí skutečně vážně. Takové portfolio aplikací nemá žádná jiná společnost a navíc pro spuštění Vaší aplikace, ať už z toho či onoho SDK, bude většinou stačit webový prohlížeč a Adobe Flash Player, který je zároveň nejrozšířenějším doplňkem v počítačových systémech připojených k internetu. Dle stránek Adobe [2] je jím vybaveno až 99% stanic s internetovým připojením (obrázek 2.4). Dalším runtime prvkem, který se v poslední době prosadil, je Adobe AIR [1], což je platforma umožňující migraci webových aplikací na desktop. Adobe rozhodně nehraje pouze na vlastním písečku a do AIR integrovala i podporu AJAXu a HTML. Jak Flash Player, tak AIR jsou dostupné pro běžně využívané

[11] http://www.bindows.net/
[12] http://code.google.com/p/pyjamas/
[13] http://qooxdoo.org/



Obrázek 2.4: Vybavenost počítačů doplňkem Adobe Flash Player [2].

operační systémy Windows, Mac OS, Linux. Co do rozšířenosti má tato platforma skutečně výborný základ a není třeba se obávat, že by aplikace pro Flash Player/AIR byly globálně nepoužitelné.

Přemýšlel jsem nad tím, jak by se dala práce této společnosti v oblasti RIA stručně a výstižně zhodnotit, ale to snad ani není možné, proto Vám doporučím navštívit domovskou stránku Adobe[14] a prostudovat možnosti, jaké Vám nabízí jejich softwarová řešení. Připadá mi, že pro každý typ aplikace mají v Adobe jiný styl vývoje a k němu vlastní softwarové řešení, které má zajistit v dané oblasti co nejpohodlnější a nejrychlejší vývoj.

Nyní bych –vzhledem k zaměření této práce– nastínil rozdíl ve způsobu práce v Adobe Flash[15] a Flex[16], jejichž finálním výstupem je naprosto stejný formát SWF a v podstatě také nabízí i shodné možnosti, ale naprosto odlišný přístup. Jako první byl vytvořen software Flash, který s sebou přinesl celou Flash platformu pro webové prohlížeče reprezentovanou Flash Playerem. Později vznikl framework Flex pro vývoj aplikací na bázi Flash platformy, ale s odlišným přístupem a zaměřením na jinou skupinu aplikací. Tyto dva přístupy přiblížím v následujících podkapitolách.

Ještě bych si dovolil připomenout jednu maličkost. Vývoj aplikací pro Flash platformu už dnes není výsadou pouze produktů společnosti Adobe, ale existují i různé kompilátory (viz 2.4.2), které jsou schopné zdrojový kód v jiném jazyce pohodlně převést do SWF formátu a tím zaručit jeho dobrou nasaditelnost.

[14] http://www.adobe.com
[15] Flash: http://www.adobe.com/products/flash/
[16] Flex: http://www.adobe.com/products/flex/

Flash

Adobe Flash je už od prvopočátku, kdy byl vydán, koncipován primárně jako vektorový grafický editor s výbornými schopnostmi v oblasti animací, přehrávání videa a možností skriptování pomocí skriptovacího jazyka ActionScript. Práce v něm je intuitivní a vcelku pohodlná, ale hodí se spíše pro designéry a aplikace založené především na animacích a scénách s případným doplněním logiky pomocí jednoduchých skriptů. Bylo by možné ve Flashi vyvinout i značně složité aplikace, ale podmiňovací způsob minulé věty snad dostatečně naznačuje, že tento způsob by nebyl zdaleka tím nejšťastnějším. Stejně tak jako ve Photoshopu je možné načíst vektorový pdf dokument, umí i Flash pracovat s bitmapovou grafikou, ačkoliv k tomu není primárně určen.

Flex

Vzhledem k Flashi je Flex nástrojem s přístupem z opačné strany. Především se snaží podporovat vývojáře přehledným prostředím (v podobě komerčního Flex Builderu nebo jiného kompatibilního IDE) a deklarativním jazykem MXML na bázi XML pro popis vlastností a rozložení prvků grafického uživatelského rozhraní. Aplikační logiku reprezentuje nyní již 3. verze dobře propracovaného ActionScriptu. Flex na mě zapůsobil také tím, že vizuální efekty lze zapisovat přímo pomocí MXML jazyka a pokud víte jak, jde to vskutku velmi snadno.

2.5 Shrnutí RIA technologií

Všechny technologie nám patrně mají co nabídnout. Základní myšlenka a hlavní rysy jsou sice obdobné, ale vždy můžeme najít drobné rozdíly, které nás buď upoutají, nebo naopak odradí. Nezatracoval bych ani jednu z technologií, ale spíše bych chtěl vyzvednout řešení společnosti Adobe pro svou rozšířenost a vysokou úroveň (což je bohužel vykoupeno zpoplatněním většiny jejich profesionálních nástrojů).

Zaujaly mě v podstatě všechny přístupy a vždy mě při studiu každého z nich napadaly různé příklady, jak by se konkrétně tento dal využít v dané věci lépe než ty ostatní. Musím ale konstatovat, že co do komplexnosti, promyšlenosti, propracovanosti a rozšířenosti jsou produkty společnosti Adobe o krok napřed. Microsoft se sice vehementně snaží se svým Silverlightem prorazit na špici, ale domnívám se, že to nebude úplně snadné, i když do vývoje investuje nemalé prostředky. Přece jenom je Flash více multiplatformní jak z hlediska operačních systémů, tak z hlediska nasaditelnosti (díky AIR). Budoucnost mají jistě i počiny založené na Javě pro svou kompatibilitu s dříve napsanými knihovnami v tomto populárním jazyce, pohodlnost, vyspělost a snahu na poli multiplatformního nasazení.

Jednoduše shrnuto všechny technologie nám poskytují dobrý základ pro psaní RIA a je možné napříč nimi docílit vcelku stejných, kvalitních výsledků. Otázkou zůstává, kterou si zvolíte pro vývoj vlastních RIA Vy?

Kapitola 3

Vizualizace

3.1 Vymezení pojmu

Samotný pojem vizualizace může působit značně rozporuplně a jelikož je mu věnováno obecně málo pozornosti a spíše se o něm rozšiřují určitá dogmata, věnujme mu společně několik málo řádků. Existuje jen velmi málo kvalitních definic, které se snaží najít skutečnou podstatu. Často se setkáváme s definicí vizualizace jako technikou v 3D grafice, což může být sice pravda, ale hloubavému čtenáři toto skromné vysvětlení nemůže postačovat. Naopak maximalistické tvrzení, že vše, co vidíme, může být vizualizací, nelze také považovat za zcela správné. Pojďme se proto pokusit tento pojem správně definovat.

3.1.1 Definice

Jak už jsem zmínil v minulém odstavci, definic existuje skutečně mnoho, ale zdaleka ne všechny pokrývají jádro věci. Vybral jsem tedy jednu krátkou a relativně výstižnou.

„Vizualizace je grafická reprezentace dat nebo konceptů, která je buď vnitřním obrazem mysli, nebo vnějším výtvorem pro podporu rozhodování.“ – viz [3].

Mohl bych zde úvést citace několika podobných –možná i kvalitnějších– definic, ale všechny by nám měly shodně naznačit, že **vizualizace se neodehrává pouze na papíře nebo obrazovce, ale také v mysli pozorovatele, kde jsou přijímané informace interpretovány. Účelem vizualizace je usnadnit pozorovateli vytvoření mentálního obrazu.**

Nyní už by mělo být patrné, že zde hovoříme o rozsáhlé oblasti, která nezaštiťuje pouze přetransformování jisté informace do vizální podoby, ale také zpětné přetvoření vizuálního vjemu pozorovatele na vnitřní mentální obraz.

3.2 Trocha historie

Z historického hlediska byly první primitivní formy vizualizace –slovo vizualizace zde uvažujme prosím s rezervou– používány již v pravěku, ať už šlo o umělecké jeskynní malby nebo například o mapu okolí vyrytou do kosti. Vždy nesly určitou informaci, která měla být prostřednictvím vizuálního vjemu předána inteligentnímu pozorovateli. O tom, zda by taková zobrazení splnila svůj účel i v dnešní době, bychom mohli polemizovat, ale základy oboru byly tímto, ač nevědomě, položeny. Je tedy zřejmé, že už pravěcí lídé přikládali vizuálnímu předání/uchování informace značnou váhu.

Od těch dob však uplynula mnohá tisíciletí, a jako každý jiný obor se i tento přirozeně rozvíjí napříč dynastiemi spolu s evolucí společnosti a techniky, a i přes tento dlouhý vývoj je rozmach vizualizace výsadou moderního věku. Masivní rozvoj nastává až s příchodem počítačů s výkonnými grafickými adaptéry, kde bylo možné produkovat kvalitní grafické výstupy – z historického hlediska může být výkonným grafickým adaptérem například i EGA[1] kompatibilní (ale i starší). Představit si dnes kvalitní výstup z takových zařízení je pro mě ale téměř nemožné.

Dnes už najdeme celou řadu vědců a odborníků, kteří této oblasti zasvětili svůj profesní život. Díky jejich práci v posledních několika desetiletích je vizualizace rozčleněna na podobory a v každém z těchto se vyskytují další, ještě více specializovaní lidé. To je momentálně dosažený výsledek vývoje vizualizace. Nebuďme bláhoví a přiznejme si, že za několik málo let už takový stav nemusí platit, práce totiž neustále pokračují a s každým novým odborným článkem se může leccos změnit.

3.3 Předpoklady kvalitní vizualizace

Tato 3 relativně přísná pravidla [13], do jisté míry ovlivněna informační vizualizací, nemusí nutně platit pro všechny druhy vizualizací, jelikož se setkáváme i s vizualizacemi na bázi umění, kde bývá význam dost často skrytý. V takových případech, a jim podobných, nemusí být uvedené předpoklady zcela naplněny. Považuji za vhodné alespoň nastínit určitá pravidla, aby pak nezavládla v chápání celé této oblasti naprostá anarchie.

Základem jsou data. Vizualizace by měla být založena na datech určených k vizualizaci. Data by měla být původně abstraktní a neviditelná, až vizualizace je transformuje na viditelná. Vizualizace není přepracování obrazových dat na obrazová data.

Vytváří obraz jako primární nosič předávané informace. Nemůžeme nazvat vizualizací něco, kde je obrazová složka pouze částí informačního celku a má jen určitý podíl na předání kompletního sdělení. Obraz musí být schopen jasným způsobem vyjádřit vše podstatné, i když možnost doplnění jinými médii se také nevylučuje.

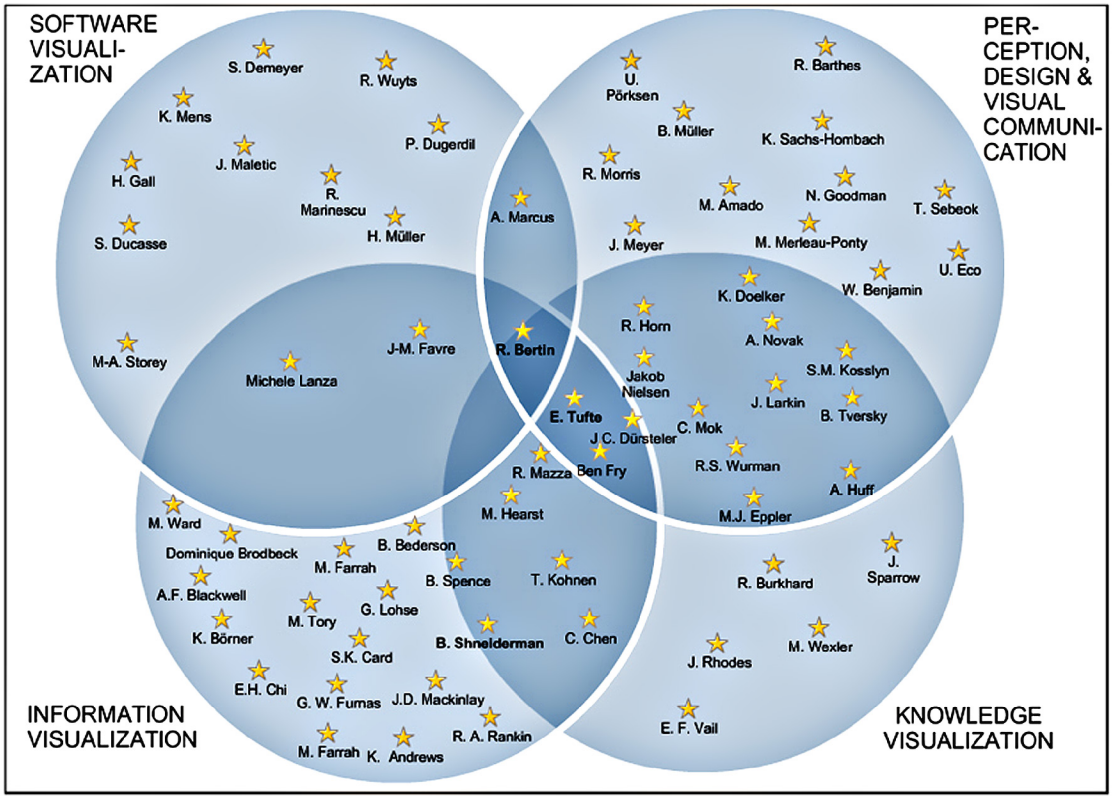
Výsledek musí být čitelný a rozpoznatelný. Výsledkem vizualizace by měl být obraz umožňující nám něco se o konkrétních datech dozvědět. Ledabylá transformace dat snadno pozbyde důležité informace a stává se z ní nic neříkající obraz, který už vlastně ani není vizualizací.

3.4 Obory

Zorientovat se v množství oborů, ve kterých se dnes vizualizace aktivně využívá, není úkolem jednoduchým, jelikož hranice mezi nimi nejsou striktně vymezeny. K ilustraci této sitace dobře poslouží obrázek 3.1, kde můžete vidět rozdělení vizualizace na několik vzájemně se překrývajících oblastí. Nejdůležitější z nich v následujících podkapitolách přiblížím. Existuje samozřejmě i další dělení na dílčí podoblasti, v rámci této práce je ovšem možné věnovat se podrobněji pouze těm hlavním.

Dvě nejdůležitější oblasti, informační vizualizace a vědecká vizualizace, se od sebe vzájemně liší tím, že informační vizualizace se zabývá zobrazením abstraktních dat, kdežto vědecká vizualizace se věnuje převážně datům fyzickým [8].

[1] http://en.wikipedia.org/wiki/Enhanced_Graphics_Adapter



Obrázek 3.1: Příslušnost autorů k jednotlivým oborům a jejich prolínání [20].

Použité informační zdroje v kapitloe 3: [7], [9].

3.4.1 Informační vizualizace

Smyslem celé této rozměrné oblasti, která značně ovlivňuje ostatní obory, je dát pozorovateli možnost zhlédnout v jediném obraze velké množství původně abstraktních informací ve srozumitelné podobě. Bez kvalitního vizuálního podání takového množství dat by bylo velmi obtížné najít důležité vlastnosti a jejich souvislosti v celé množině. Z hlediska délky vizualizačního procesu je možné považovat informační vizualizaci za náročnější než kupříkladu vědeckou, jelikož prvně jmenovaná se musí vypořádat se špatnou počáteční srozumitelností dat, která obecně nesou méně důležitých atributů než jiná data (např. vědecká) a v důsledku toho se s nimi i obtížněji pracuje.

3.4.2 Vědecká vizualizace

Vědecká vizualizace je rovněž rozsáhlá větev primárně orientovaná na zobrazení experimentálně získaných dat za účelem jejich prozkoumání a další analýzy. Hlavními oblastmi, kde je tato disciplína hojně využívána, jsou fyzika, chemie, medicína, matematika, letectví, atp. S vědeckými vizualizacemi máme možnost se setkávat celkem běžně, stačí si například vzpomenout na videa, která nám pouštěli pedagogové na základní škole ve fyzice nebo v chemii. Už tehdy jsme měli možnost nahlédnout do světa vědecké vizualizace, když se na obrazovce otáčel 3-dimenzionální model molekuly vody. [4]

V dnešní době není nic neobvyklého, když se předmětem zkoumání stane elektromagnetické vlnění, infračervené záření nebo enormě vzdálené těleso v kosmu. Výzkum v těchto oblastech by bez vizualizace získaných informací snad ani nemohl existovat.

3.4.3 Softwarová vizualizace

Softwarová vizualizace se snaží usnadnit nám chápání běhu programů nebo algoritmů tím, že do grafické formy převádí vnitřní strukturu a chování dané aplikace[2]. Jelikož vizualizovaná data bývají zpravidla abstraktní, je softwarová vizualizace často považována za podmnožinu informační vizualizace.

O softwarovou vizualizaci se běžně opírá výuka algoritmů nebo procesů probíhajících uvnitř aplikací, při čemž podpora výuky mnohdy neslouží jen studentům, ale je hojně využívána i na poli profesionálního programování. Ladění programů za použití moderních debuggerů se bez tohoto typu vizualizace rovněž neobejde. Tím jsme postihli nejčastější užití, ovšem kdybychom uvážili například celý vývojový cyklus software, našli bychom jistě i jiné možnosti užití. [8]

3.4.4 Další obory

Výčet některých souvisejících oborů.

- Produktová vizualizace
- Vizualizace vědomostí
- Vzdělávací vizualizace
- Vizualizace proudění
- Interaktivní vizualizace
- Vizualizace hudby
- Geovizualizace
- Ilustrace
- Vizuální komunikace
- Vizuální analýza

3.5 Přístupy

Pragmatický přístup

Jsme-li schopni z obrazu bez problémů rozpoznat účel vizualizace a zjistit alespoň nějaké údaje nebo souvislosti, můžeme říci, že jde o vizualizaci pragmatickou. Takový přístup nám má umožnit důkladně prozkoumat a pochopit data/informace. Výhodou pragmatického přístupu je obecnost vizualizačních metod, které můžeme využívat na různé množiny dat. Samozřejmě existují i specifické oblasti, kde není možné použít některou z obecných metod. Stále ale platí, že velká množina dat může být zpracována pomocí generalizovaných postupů. [12]

[2] Velmi zajímavá forma softwarové vizualizace: http://www.cc.gatech.edu/gvu/softviz/3dcv/3dcv.html

Umělecký přístup

Cílem uměleckého přístupu není vyjádřit obraz v takové podobě, aby byl pro pozorovatele co nejlépe čitelný. Často je nutné číst tzv. mezi řádky, abychom odhalili podstatu, jelikož základní myšlenka nebývá obvykle okamžitě zřetelná. Není rovněž důležité předat pozorovateli vyčerpávající přehled údajů. Umělecký směr se spíše orientuje na podstatu celého problému a tu se snaží vyjádřit zajímavým způsobem, současně je ale důležité, aby dílo bylo srozumitelné i přes svůj umělecký ráz. Příkladem může být projekt *The Dumpster*[3]. [12]

3.6 Techniky

Množství způsobů, jak vyjádřit efektivně různé typy datových/informačních množin, neustále narůstá. Tradičními způsoby jsou například tabulky, grafy, diagramy, matice, mapy, stromy, apod. Se zlepšující se dostupností levných a současně výkonných počítačových systémů, ruku v ruce s rozvojem počítačové grafiky, přibývají další, vizuálně realističtější a propracovanější metody.

„It is currently a challenging task for designers to find out the strategies and tools available to visualize a particular type of information.“ – viz [7]. Překlad uvedené fráze nám říká následovné. *„Jde vždy o výzvu pro designéry, aby našli strategie a nástroje dostupné pro vizualizaci jistého typu informací.“* Toto tvrzení musí být nutně pravdivé už jenom z důvodu, že svět je neustále zaplavován novými poznatky, vynálezy, produkty, aj. a každá taková novota si žádá jiný přístup, tedy i jiné metody vizualizace souvisejících informací/dat.

Příklady metod si můžete prohlédnout na obrázcích 3.2 a 3.3. Dnes můžeme na internetu navštívit několik webů[4] zabývajících se shromažďováním různých, ať už typických či originálních, druhů vizualizací. Vřele doporučuji navštívit alespoň odkazy uvedené pod čarou pro získání přehledu možných výstupů používaných technik.

3.7 Hodnocení kvality a dosažených výsledků

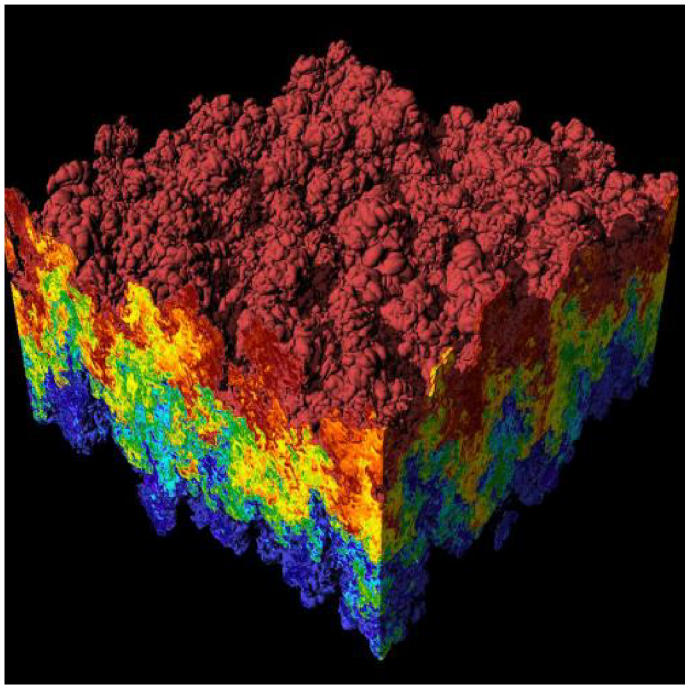
Hodnocení kvality vizualizace může být mnohem větší výzvou, než si dokážeme sami připustit [7]. Ostatně jsem se o tom přesvědčil na vlastní kůži – viz kapitola 5. Nepředbíhejme ale a pojďme si objasnit, proč tomu tak je.

Zaprvé neexistuje dostatek všeobecných kritérií, podle kterých bychom mohli jasně určit jakost celého našeho snažení. Dále musíme vzít v úvahu fakt, že o úspěchu či neúspěchu vizualizace budou rozhodovat lidé. Jak tedy předem zajistíme, aby naše záměry byly interpretovány všemi pozorovateli správně? Tím se dostáváme k jádru věci, rozmanitost lidského chápání, či rozdílná úroveň vědomostí pozorovatelů nám v této oblasti mohou činit skutečné problémy. Proto je třeba při studiu vizualizace počítat i s dalšími příbuznými obory (např. vizuální komunikace) a také s tím, že ne vždy bude naše dílo pochopeno masami lidí. Někdy může být úspěchem i správná interpretace několika málo specialisty daného oboru.

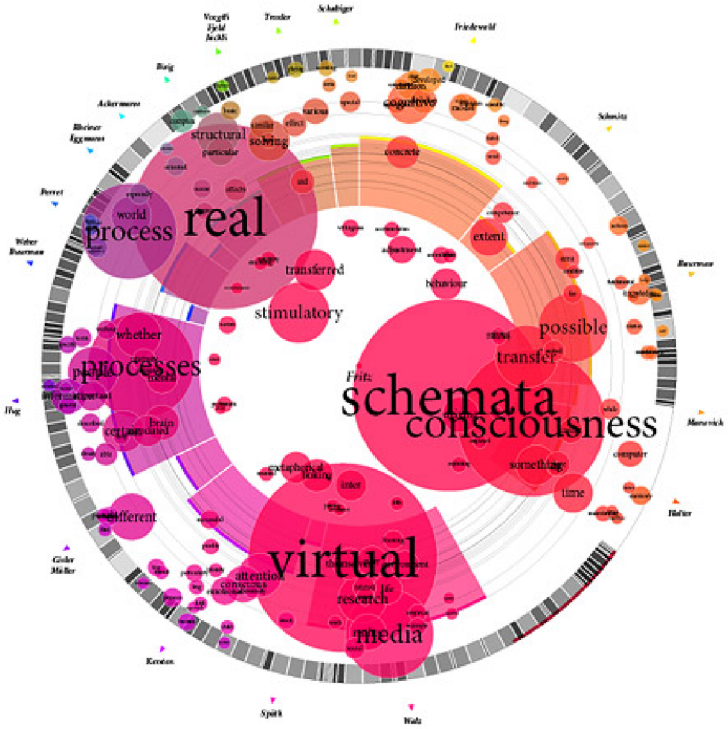
Je tedy více než vhodné ještě před zahájením práce jasně vědět, **co** chci vyjádřit a **komu** chci toto sdělení předat. Ani tak ale nemusí být úspěch zcela zaručen. Psychika, myšlenkové pochody a chápání člověka vyžadují skutečně odborný přístup. Proto může být celá tato záležitost vskutku komplikovaná.

[3] http://artport.whitney.org/commissions/thedumpster/

[4] http://www.visualcomplexity.com/vc/, http://manyeyes.alphaworks.ibm.com/manyeyes/



Obrázek 3.2: Ukázka moderní vědecké 3D vizualizace: Smíchání 2 tekutin [21].



Obrázek 3.3: Ukázka moderní vizualizace: Podobnosti textů v esejích v rámci knihy [18].

Kapitola 4

Implementace

4.1 Vývojové prostředí

K vlastní implementaci byl po zvážení použit Open Source Flex SDK verze 4, který produkuje spustitelné soubory pro platformu Flash, resp. AIR. Tento vývojový KIT je dostupný zdarma ke stažení na stránkách firmy Adobe[1]. Vývoj v SDK je možný prostřednictvím libovolného textového editoru, ovšem ne každý je pro práci s MXML, resp. ActionScriptem vhodně vybaven, proto bych doporučil použití nástrojů k tomu určených[2]. Je také možné vyzkoušet trial verzi komerčního Flex Builderu, což bych do začátku radil všem úplným nováčkům. Můžete se díky němu snadno zorientovat ve stylu programování Flex aplikací a získané dovednosti pak využít k rychlejší práci ve volně dostupných řešeních.

Použití programovacích jazyků je tedy jasně dáno platformou, v tomto případě tedy MXML, ActionScript 3. generace a stylovací CSS[3], který v podání Adobe Flexu není dokonalým ztotožněním známého CSS, s nímž se setkáváme např. při tvorbě webových stránek.

MXML se používá především k umístění a rozložení prvků grafického uživatelského rozhraní (dále jen GUI), i když není žádný problém psát v něm i aplikační logiku, různé efekty, či přímo stylovat prvky. Ovšem je třeba si na styl psaní v tomto jazyce nejprve zvyknout, obzvláště u aplikační logiky. Já jsem se snažil MXML využít především za účelem vytvoření a uspořádání prvků GUI, i když jsem ojediněle použil i další z možností.

ActionScript (dále jen AS) je původem skriptovací jazyk, jak vypovídá i název, ovšem jeho poslední generace je schopna konkurovat moderním, objektově orientovaným jazykům, a je možné jeho prostřednictvím psát i skutečně rozměrné aplikace. V podstatě celá aplikační logika je v mé aplikaci napsána právě v AS.

CSS ve Flexu může působit poněkud neohrabaným dojmem, což musím bohužel potvrdit, a někteří vývojáři jej dokonce odmítají nazývat CSS, protože se, jednoduše řečeno, občas chová jinak, než bychom předpokládali. Přesto se mi podařilo využít jej k ostylování komponent GUI.

[1] http://opensource.adobe.com/wiki/display/flexsdk/Download+Flex+4

[2] NetBeans + FlexBean plug-in - http://www.netbeans.org/, http://sourceforge.net/projects/flexbean; FlashDevelop (podpora pro MXML ve vývoji) - http://www.flashdevelop.org/community/

[3] http://www.w3.org/Style/CSS/

4.1.1 Paradigmata

Volba přístupu k vývoji celé aplikace je pravděpodobně jedním z prvních kroků, který je nutno učinit hned na počátku. Jelikož jsem v té době byl v prostředí Flexu absolutním nováčkem, s AS měl opravdu jen velmi letmé zkušenosti a nedokázal jsem si ani představit rozsah celé aplikace, zvolil jsem procedurální paradigma – i vzhledem k tomu, že původním účelem AS bylo jednoduché skriptování. Podpora objektově orientovaného programování byla implementována až s vývojem jazyka.

Nyní, ohlédnu-li se zpět a zhodnotím-li celou práci, bylo by, v případě pokračování rozvoje celého projektu, pravděpodobně lepší zvolit objektové paradigma z důvodu lepší udržovatelnosti, rozšiřitelnosti a celkové přehlednosti.

4.1.2 Dokumentace

Pro vygenerování dokumentace byl použit konzolový nástroj ASDoc[4] (dále jen ASD), který je od 2. generace Flexu přímo součástí vývojového kitu. Dokumentaci jsem se zde rozhodl zmínit z důvodu, že pro její vygenerování musely být všechny modifikátory přístupu `private` změněny na `public`. ASD totiž odmítá generovat do dokumentace cokoliv, co je uvedeno klíčovým slovem `private`. Neexistuje ani žádný přepínač, jako například v podobném nástroji JavaDoc[5] parametr `-private`, který by umožnil nucené generování privátních částí. Proto můžete v dokumentaci vidět modifikátor přístupu `public` i tam, kde se v aplikaci skutečně nevyskytuje.

Ještě jednu drobnost bych si dovolil zmínit. Bylo také nutné, abych před spuštěním ASD zakomentoval odkaz na externí CSS soubor. Nachází se v něm totiž odkazy na png soubory s ikonami. A právě s odkazy na externí soubory uvedené v CSS se ASD rovněž nemá příliš v lásce a odhalit příčinu pádu ASD v takovém případě může být vskutku mravenčí práce. Více se o tomto tématu, v případě zájmu, můžete dočíst například zde[6].

Dokumentaci naleznete na přiloženém CD disku - viz příloha A.

4.1.3 Zhodnocení prostředí

I přes počáteční potíže –kterým se asi nelze vyhnout při práci s jakýmkoli novým nástrojem– bych Flex hodnotil kladně. Po krátkém čase už byla práce plynulá a pohodlná. Za několik málo let existence tohoto nástroje existuje na internetu již relativně dobrá základna stránek zveřejňujících různé tipy, triky a tutoriály. Je tedy možné se poměrně snadno naučit základní dovednosti a na těch následně stavět při vývoji vlastních implementací.

Jedinou věcí, která mi skutečně čas od času činila problémy, byla jiná interpretace stylovacích CSS, než bych –vzhledem k tomu, že znám styly z tvorby webů– očekával. Odvážil bych se také říci, že konkrétně v této oblasti Flex platformy nejsem nijak zvlášť zdatný, což je škoda, protože díky dobré znalosti stylování je možno vytvořit oku lahodící uživatelská rozhraní – přece jenom je vzhled to první, co při spuštění programu bývá podrobeno hodnocení.

[4] http://livedocs.adobe.com/flex/3/html/help.html?content=asdoc_1.html
[5] http://java.sun.com/j2se/javadoc/
[6] http://viconflex.blogspot.com/2008/04/invalid-embed-directive-in-stylesheet.html

4.2 Struktura programu

Program se skládá z jednoho hlavního balíku, který obsahuje v podstatě celou aplikační logiku včetně komponent potřebných k reprezentaci prvků animace, a také k uchování historie. Pro potřeby grafického uživatelského rozhraní byla použita knihovna SuperPanel[7]. Za účelem načtení souboru s vlastním zadáním byly využity nové techniky[8] dostupné až od 10. verze Flash Playeru.

Cílem této kapitoly není popsat kompletně celý program, nýbrž zvolit zajímavé části řešení a těm se věnovat. Kompletní strukturu aplikace Vám přiblíží –za tímto účelem vytvořená– dokumentace (podotýkám znovu, že musely být změněny modifikátory přístupu viz. 4.1.2).

4.3 Spuštění aplikace

Pro spuštění aplikace je třeba mít v prohlížeči nainstalovaný FlashPlayer alespoň 10. verze. Aplikace je ošetřena tak, aby na starších verzích nešla spustit a vyžadovala po uživateli aktualizaci na požadovanou úroveň. Starší verze totiž neumožňovaly použití některých technik, jako je např. načítání externích souborů, a navíc obsahovaly chyby, které se projevovaly při zobrazování prvků v panelu *Vizualizace*.

4.4 Zvolený demonstrační algoritmus

Po dohodě s vedoucím této práce, panem docentem Smržem, jsme zvolili k demonstraci algoritmus inkrementální konstrukce minimálních acyklických konečných automatů (*Incremental Construction of Minimal Acyclic Finite State Automata*)[9] pro abecedně setříděná vstupní data, jehož autory jsou Jan Daciuk[10], Stoyan Mihov[11], Bruce Watson[12] a Richard Watson. Pro účely testování a demonstrace byly předdefinovány jednoduché příklady, můžete ovšem také zadat příklady vlastní.

4.4.1 Vlastní zpracování algoritmu

Pro snadnou orientaci byly hlavní funkce vykonávající algoritmus pojmenovány stejně jako je uvedeno v dokumentaci k algoritmu – `common_prefix()`, `replace_or_register()`. Jejich provedení muselo být ovšem podřízeno účelu celé aplikace a programovacímu jazyku. Hlavní smyčka algoritmu, kterou je možné vidět v aplikaci v panelu *Zdrojový kód*, musela být vzhledem k nutnosti krokování jednotlivých řádků rozdělena na dílčí části, což způsobilo, že jako taková už se v rámci programu vyskytuje pouze ve zmíněném panelu.

4.5 Implementace animace algoritmu

V této kapitole budou diskutovány hlavní prvky animace algoritmu (dále uváděno i jako vizualizace), případně i jejich technická provedení, jsou-li nějakým způsobem zajímavá.

[7] http://brandonmeyer.net/blog/?p=6
[8] http://www.mikechambers.com/blog/2008/08/20/reading-and-writing-local-files-in-flash-player-10/
[9] http://www.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/daciuk98.ps.gz
[10] http://www.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/
[11] http://www.lml.bas.bg/~stoyan/
[12] http://bruce-watson.com/Bruce_Watson/Bruce_Watson.html

Pro potřeby animace algoritmu bylo použito hned několik prvků. Prvním z nich je zobrazení prováděného pseudokódu v panelu *Zdrojový kód* a jeho následné zvýrazňování v jednotlivých krocích. Dále jsou zobrazovány stavy proměnných, které jsou také dle právě zpracovávaného úseku algoritmu zvýrazněny. Posledním a pravděpodobně také nejnázornějším prvkem je panel *Vizualizace*, kde jsou postupně znázorňovány znaky a vazby mezi nimi. Můžeme zde také vidět dopady optimalizace provedené funkcí `replace_or_register()`.

4.5.1 Zdrojový kód

Vizuální reprezentace zdrojového kódu je provedena pomocí seznamu (`List`), který umožňuje relativně snadné zvýraznění jediného řádku. Při požadavku na zvýraznění více řádků je ale třeba myslet na to, že nelze jen jednoduše jedinou číselnou hodnotou určit pozici zvýraznění, nýbrž je nezbytné užít jinou metodu a té dodat v poli identické objekty k těm, které mají být zvýrazněny.

Původně byla celá tato záležitost řešena prostřednictvím textové oblasti, kde bylo zvýraznění prováděného pseudokódu realizováno stejně, jako když myší označíme text, což se ale později ukázalo jako zbytečně komplikované a neelegantní řešení. Tímto bych chtěl poukázat na vývoj, jímž aplikace postupně procházela.

4.5.2 Proměnné

Přehled o aktuálním stavu proměnných snad ani nemusím nijak složitě rozebírat. Jde o zobrazení dané hodnoty ve správný čas v příslušném textovém poli. Textová pole jsou v patřičných krocích zvýrazňována, navíc je pole *LastState* barevně odlišeno, stejně tak je barevně odlišen i stav LastState v poli *Vizualizace*.

V případě, že zůstanete v kroku, v němž je pole *LastState* zvýrazněno, déle než půl sekundy, začne toto problikávat. Stejně tak bude problikávat i odpovídající stav v panelu *Vizualizace*. Tímto je zdůrazněna důležitost stavu *LastState* a současně je demonstrováno, jakým způsobem je možné upozornit na důležitou část programu.

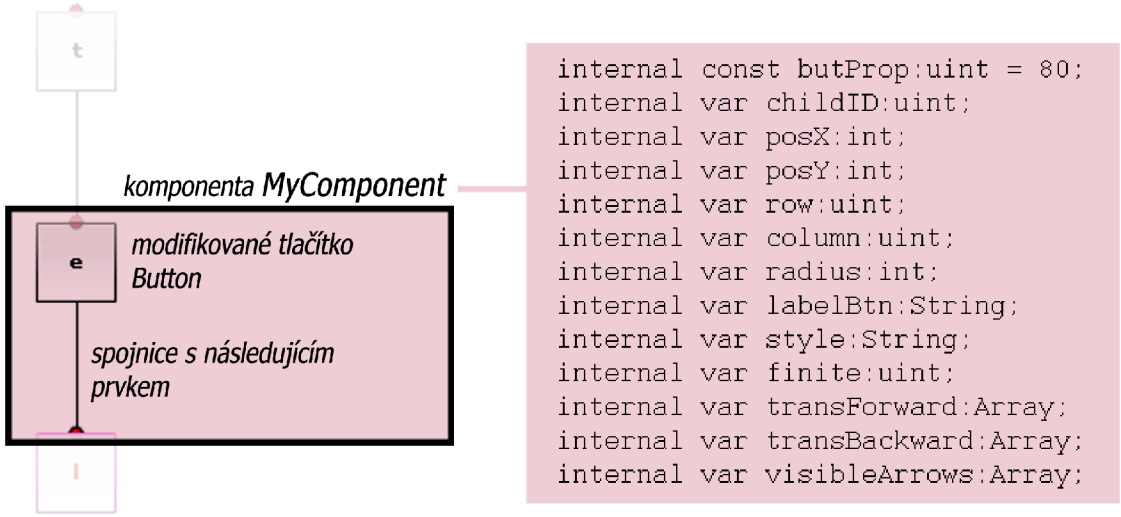
Jinou ukázkou, jak usnadnit uživateli orientaci, může být možnost označit prvek z panelu *Register* a tím barevně odlišit i jeho grafickou reprezentaci.

Z hlediska implementace se dá relativně zajímavě textových polí využívat i jako proměnných pro samotný algoritmus. Stačí jenom zakázat editovatelnost pro uživatele, tím je pole a jeho obsah výhradně v režii programu, a tudíž je možné využívat přímo jeho hodnoty jako globální proměnné.

4.5.3 Grafická reprezentace znaků - panel *Vizualizace*

Data každého stavu (znaku) jsou uchována v podobě komponenty popsané v kapitole 4.5.4. Prvky v tomto panelu jsou tvarově a barevně rozlišeny tak, aby bylo vždy jasné, který stav je koncový, který byl zpracován funkcí `replace_or_register()`, nebo je právě vybraný z panelu *Register* uživatelem. Význam barev a tvarů získáte prostudováním legendy (Možnosti - Legenda) v aplikaci. Jak už bylo řečeno 4.5.2, rozlišení grafických prvků bylo provedeno tak, aby korespondovalo s událostmi probíhajícími v aplikaci nebo vyvolanými uživatelem.

Navíc můžete získat o prvcích stručnou nápovědu tím, že na ně jednoduše ukážete myší. Zobrazí se Vám doplňkové informace o zvoleném stavu. Prvky si uchovávají poslední dosažený tvar a nápovědu i při průchodu příkladem zpět, aby bylo jasně rozpozna-



Obrázek 4.1: Anatomie komponenty `MyComponent`.

telné a názorné, které z prvků příkladu byly, resp. budou zpracovány optimalizační funkcí `replace_or_register()`.

Pro zlepšení přehlednosti byly implementovány i další funkce, například celoobrazovkový režim nebo zoom panelu *Vizualizace*.

4.5.4 Struktura pro uchování dat

Základním kamenem uchování dat pro účely animace je komponenta s příznačným názvem `MyComponent` - viz obrázek 4.1. Kromě proměnných, které nesou všechny důležité vlastnosti od umístění, až po vazby na další komponenty, obsahuje 2 základní viditelné prvky – upravené tlačítko (*Button*), které zobrazuje písmeno v kruhu, a plátno (*Canvas*) do kterého se kreslí spojnice mezi prvky. Každá komponenta má na svém plátně vždy jen spojnice vedoucí na následující prvek/prvky, nikdy neobsahuje spojnice na prvky předcházející, tyto jsou vykresleny předešlými stavy.

Aby nedošlo k porušení logické kontinuity řetězců, jsou součástí všech těchto komponent 2 pole – `transForward` a `transBackward`, hodnoty v těchto polích vyjadřují unikátní čísla (*child ID*) komponent o 1 pozici vpřed, resp. vzad. Při tom je důležitá nejen vlastní hodnota, ale i index, pod kterým se tato v poli nachází. Při minimalizaci funkcí `replace_or_register()` je totiž nutné přesně roslišovat, kdy byl který prvek přidán. Vycházím při tom z logické úvahy: čím vyšší index, tím později byl prvek do pole přidán, a tím je pak jasně dána i lexikografická pozice celého řetězce. Mimoto na pozicích hodnot v poli `transForward` závisí také správné zobrazování aktuálně viditelných spojnic, jejichž viditelnost v jednotlivých krocích je mapována v poli `visibleArrows` každé komponenty.

Prvky z panelu *Vizualizace*, ať už jen spojnice uložené na plátně komponenty, nebo pak samotné komponenty jako celek, se při minimalizaci nikdy neodstraňují, ale jsou pouze zneviditelněny. Jejich stálé existence se poté využívá při opětovném průchodu již provedenými kroky 4.6, kdy stačí pouze zviditelnit příslušné elementy.

```
// zjistim v kterem stavu programu se nachazim
// stepView = textove pole s cislem aktualniho kroku,
// STATE_FINAL - konstanta nesouci hodnotu posldeniho stavu
var remainder:uint = (int(stepView.text) - 1) % STATE_FINAL;

// bylo-li jiz skonceno pridavani novych prvku a soucasne jde o posl. krok
// musim zajistit, ze se presunu do finalniho stavu
if (!InsertingNotFinished && (int(stepView.text) == Calendar.lastStep))
{
  remainder = STATE_FINAL;
}

// podle stavu hodnoty remainder urcim, co se ma zvyraznit a prip. nahrat
// z kalendare, jde-li o playBack.
switch (remainder)
{
  case STATE_0:
    // VIZ ZDROJOVY KOD
    break;

  case STATE_1:
    // VIZ ZDROJOVY KOD
    break;
  .
  .
  .
  case STATE_FINAL:
    // VIZ ZDROJOVY KOD
    break;
}
```

Obrázek 4.2: Ukázka způsobu řešení krokování.

4.6 Realizace krokování - tam i zpět

Nejprve je nutné upozornit na to, že rozlišujeme 2 typy pohybu vpřed (stisknutím tlačítka *Vpřed*). První nastává pouze při zpracování ještě nenavštíveného kroku, tedy v každém kroku právě jednou, druhý při opětovném zobrazení těch již vizualizovaných. Tlačítkem *Zpět* se můžeme dostat vždy jen do pozice, která musela být nutně vyřešena dříve. Tím je určena doslova jednoduchá funkčnost druhého z tlačítek.

Celá realizace vychází z poměrně jednoduché myšlenky. V každém novém kroku dojde ke změně některého z vizualizovaných elementů, tato situace je zaznamenána do vytvořené komponenty typu `History` reprezentované v programu globální proměnnou `Calendar`. Ta ve svých vnitřních polích uchovává kompletní aktuální stav pro každý provedený krok (můžeme jej nalézt vždy pod indexem *daný krok - 1* – vychází ze základní vlastnosti programovacího jazyka. Pole se indexují od *0*, kdežto počáteční krok je označován jako *1.*).

Stručně shrnuto tedy platí, že pokud zpracovávám krok poprvé, musím zaznamenat jeho stav. Pokud procházíme již vypracovanými stavy, stačí podle záznamů jen zobrazit aktuální hodnoty proměnných, stav plátna *Vizualizace* a zvýraznit patřičná textová pole, nic nového se už nepočítá.

Kapitola 5

Testy a srovnání výsledků

Moje původní představa o testech a hlavně o jejich výsledcích nebyla do jisté míry uskutečněna a musím dát za pravdu všem, kteří zkušeně tvrdí, že testování a následné získávání výsledků může být procedura velmi náročná. Obzvláště tehdy, je-li k testování zapotřebí lidských bytostí, pak se jedná v jistých případech o skutečně extrémně náročný úkol, jehož realizace by měla být raději svěřena do rukou odborníků.

5.1 Cílová skupina

Jako cílovou skupinu pro provedení testů jsem zvolil své kolegy, studenty z 3. ročníku Fakulty informačních technologií na VUT v Brně. Původně jsem se domníval, že obecné povědomí respondentů o informačních technologiích bude dostatečné pro provedení naplánovaných úloh. Jistě mi ale dáte za pravdu, že ne každý vyslovený předpoklad bývá nutně naplněn dle autorových představ.

5.2 Plán testování

Plánování testů proběhlo relativně snadno a také poměrně rychle. Mnohem náročnější pak byla realizace, tedy příprava potřebných materiálů, zajištění dostatečného počtu respondentů a následné provedení samotné akce. Testy byly naplánovány pro 2 skupiny uživatelů. První skupina vůbec nepřišla do styku s aplikací a týkala se jí pouze jedna část testovací procedury – dotazník zjišťující správné pochopení jistých částí algoritmu ve stanoveném čase (viz kapitola 5.3). Druhá skupina dostala k dispozici kromě jiného i vytvořený program animace algoritmu a tím byla kompetentní k vyplnění i druhé testované části – dotazníku o spokojenosti uživatele s provedením vizualizace a aplikace jako takové (viz kapitola 5.4).

5.3 Test 1

První z dvojice testů měl být důkazem toho, že aplikace značným způsobem usnadní pochopení zvoleného algoritmu.

5.3.1 Zadání

Skupina číslo 1 (5 respondentů) obdržela pouze výtah z dokumentace k algoritmu (jen podstatné části), jednoduchý průvodní list k průběhu testování a soubor s otázkami. Druhá

skupina (5 respondentů) dostala totéž a navíc ještě vlastní aplikaci a návod k jejímu užití. Respondenti měli za úkol nastudovat označený algoritmus a následně odpovědět na otázky týkající se hlavní smyčky algoritmu, při čemž měl být zaznamenán čas potřebný k zodpovězení otázek. Test neměl trvat déle než 30 minut (z důvodu, abych respondenty příliš nezatížil).

5.3.2 Zhodnocení testu

Optimista by tvrdil, že jakýkoliv algoritmus je možné pochopit z dobře připravené vizuální reprezentace konkrétních příkladů. Pesimista by přirozeně mohl prohlásit pravý opak. Jak už tomu bývá, pravdu zpravidla nemívá ani jeden z nich. Některé algoritmy jsou natolik jednoduché, že po chvíli experimentování je naprosto jasné, co se v procesu děje. Naopak jiné vyžadují hlubší znalost nejen algoritmu jako takového, ale i dalších příbuzných oblastí, což už nemusí být z hlediska nabytí vědomostí otázka několika málo okamžiků a ani hodin.

První test jsem byl nucen označit za neúspěšný z důvodu, že v podstatě nebyl ani řádně dokončen. Přibližně po 10 minutách se totiž ze strany respondentů začaly objevovat různé dotazy, se kterými jsem při přípravě materiálů skutečně nepočítal. Domníval jsem se, že studenti budou schopni samostatně z dodaných materiálů požadované informace získat. Pravděpodobně jsem si naivně vsugeroval představu o tom, jak všichni hladce projdou mnou naplánovanou cestou. Celá situace postupně gradovala, až jsem nakonec dospěl k závěru, že celý průběh testu byl natolik narušen, že výsledky nelze považovat za průkazné. Tím byl test ukončen.

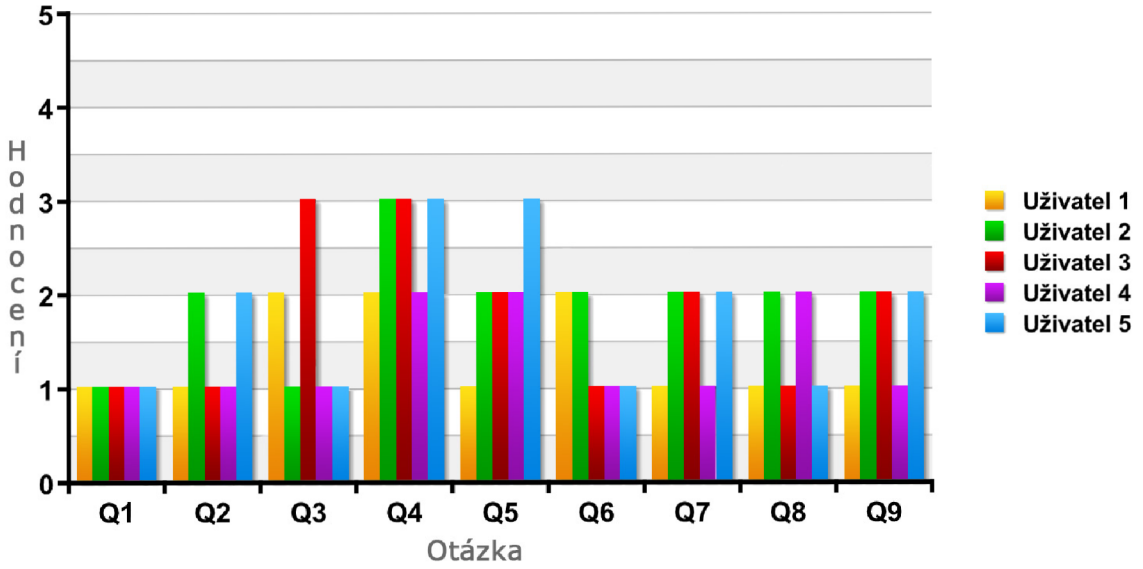
Ztroskotání původního plánu přisuzuji prvotní horší srozumitelnosti zvoleného algoritmu (i když jsem se snažil vymyslet poměrně jednoduché otázky, které byly vypozorovatelné jak z aplikace, tak s trochou snahy i z dodané dokumentace), jeho obecné nerozšířenosti a pak pravděpodobně také nedostatečné odbornosti respondentů, a možná i, nerad bych ovšem někoho křivě nařkl, lenosti snažit se aktivně pochopit cosi nového. Pravdou ale zůstává, že jsem sám ke studiu tohoto algoritmu potřeboval relativně dlouhou dobu, jelikož porozumění všem detailům a získání patřičného nadhledu není samozřejmostí jako u jiných, jednodušších, postupů. Zde je patrně třeba získat nejprve jisté povědomí o problematice jako takové, poté se teprve pustit do studia algoritmu.

I přes nezdařené provedení testu mohu vyvodit stručný závěr. Vizualizovat algoritmus má bezpochyby smysl v případě, že je doplněn v lepším případě kvalitním výkladem, nebo alespoň kvalitními textovými podklady. Dalším faktorem rozhodujícím o úspěchu vizualizace je jistě správné zacílení na skupinu uživatelů. S nadsázkou lze říci, že není možné prodávat osobní automobily na základní škole. Později jsem náhodou narazil na web[1] výzkumné skupiny, kde je výsledný neúspěch mého prvního testu zdůvodněn.

5.4 Test 2

Druhý test byl zaměřen na spokojenost uživatelů, tedy tzv. *user satisfaction test*. Zde byli respondenti dotazováni na provedení jednotlivých částí aplikace a měli vyjádřit své hodnocení především k provedení vizualizace. Tento test dopadl o poznání lépe. Nebyla zde vyžadována žádná neobvyklá znalost nebo nutnost získat nové vědomosti, proto byl pravděpodobně pro uživatele mnohem průchodnější.

[1] http://www.cc.gatech.edu/gvu/softviz/empir/empir.html



Obrázek 5.1: Přehledný graf výsledků 2. testu

5.4.1 Zadání

Jednotlivá kritéria měla být dotázanými uživateli oznámkována užitím stupnice 1–5, kde 1 znamená výborně a 5 nedostatečně.

1. Krokování
   (a) Zvýrazňování kódu [Q1]
   (b) Zvýrazňování proměnných [Q2]
2. Registr
   (a) Možnost označit a zvýraznit reprezentanta dané množiny [Q3]
3. Vizualizace
   (a) Viditelnost vztahů mezi prvky [Q4]
   (b) Názornost [Q5]
4. Aplikace
   (a) Ovládání [Q6]
   (b) Přehlednost [Q7]
   (c) Celková názornost [Q8]
5. Celkový dojem z aplikace [Q9]
6. Slovní hodnocení

Výsledky hodnocení můžete vidět v přehledném grafu na obrázku 5.1.

5.4.2 Zhodnocení testu

Výsledky testu jsou vcelku pozitivní (viz obrázek 5.1), ukazují nám oblibu grafické reprezentace abstraktních informací mezi respondenty a dá se z nich rovněž odvodit, že vizualizovat jinak těžko představitelné procesy má rozhodně z hlediska názornosti smysl. Nejhůře hodnocena byla pak viditelnost vztahů mezi prvky –předpokládám, že z důvodu horší přehlednosti při složitých optimalizacích. Tuto záležitost by bylo vhodné nějakým způsobem zlepšit.

5.5 Shrnutí a důsledky

Závěrem bych chtěl k testování jako takovému dodat, že skutečně nelze srovnávat testování různých typů aplikací. U některých stačí pouze připravit skript s testovacími příklady, změřit např. čas procesoru potřebný pro výpočet úlohy a tím je dán nezpochybnitelný výsledek. Naopak tam, kde se do popředí dostává člověk jako nástroj testování, je náročné připravit kvalitní test tak, aby výsledky skutečně vypovídaly o měřeném aspektu. Je to dáno tím, že člověk není exaktní stroj, tudíž nelze vypracovat testy podle jednoho modelového případu a myslet si, že všichni je budou vnímat stejným způsobem.

Na základě doporučení ve slovním hodnocení 2. testu (kapitola 5.4) byly provedeny ještě některé úpravy v aplikaci. Například bylo zavedeno zvýraznění panelu *Vizualizace* při provádění funkce `add_suffix()`. Původní verze nepočítala s podbarvením tohoto prvku, jelikož jsem byl přesvědčen o tom, že přibývající elementy v tomto panelu už samy o sobě dosti zvýrazňují jeho použití v patřičných krocích. Dalším vylepšením bylo vyměnění pozice tlačítek *Vpřed* a *Zpět* tak, aby tlačítka byla umístěna v souladu se svým logickým významem dle zaužívaného standardu, tedy *Zpět* vlevo a *Vpřed* vpravo. Poslední markantní úpravou na základě doporučení uživatelů bylo provedení grafického odlišení typů stavů v panelu *Vizualizace*, aby byla okamžitě zřejmá příslušnost k určité skupině.

5.6 Motivace k budoucímu rozvoji

Dalšími vylepšeními, která by mohla být v budoucnu provedena, jsou například zvýraznění celé cesty slov po najetí myší na některý ze stavů. Podrobnější vizualizace kódu a krokování s tím, že by si uživatel mohl zvolit, zda chce vizualizaci podrobnou, či jednoduchou.

Z uživatelského hodnocení vyplývá, že by bylo vhodné přepracovat způsob umísťování prvků v panelu *Vizualizace* pro zlepšení přehlednosti. Nový umísťovací systém by mohl také obsahovat podporu tažení prvků po plátně na libovolné místo v panelu.

Posledním návrhem na vylepšení je přesunutí popisků z vnitřního prostoru stavů ke spojnicím (přechodům).

Kapitola 6

Závěr

Cílem této bakalářské práce bylo diskutovat a reálně předvést vlastní implementaci možností animace algoritmů na platformě Flash. Naplnění tohoto záměru byl podřízen vývoj programové části, kde můžete sledovat užití různých vizualizačních prvků v praxi.

Jelikož se jedná o téma spojené s vizuálním vjemem, strávil jsem jistý čas dolaďováním detailů právě vizuálního charakteru programu, aby po této stránce byla zajištěna jistá úroveň.

Při pohledu zpět na průběh vzniku celé práce mě snad jen zamrzí opravdu až zbytečně dlouhý čas věnovaný studiu použitého demonstračního algoritmu, jelikož jsem se zde neměl zaobírat do hloubky žádným konkrétním algoritmem, nýbrž možnostmi jeho vizuální reprezentace jako takové.

Pokračování vývoje aplikací tohoto typu jde ruku v ruce s vývojem výpočetní techniky a postupů používaných v této dynamicky se rozvíjející oblasti. Splněním nejrůžovějšího snu by bylo vytvoření univerzálního vizualizačního nástroje, který by dokázal akceptovat algoritmus v nějakém vhodném jazyce a tento následně kvalitně vizualizovat, což je velmi obtížně dosažitelné z důvodu přílišné rozmanitosti různých typů úloh, a tudíž i algoritmů. Dalo by se říci, že pro každý typ algoritmu by musel být vypracován samostatný modul, který by dokázal pro daný typ úlohy zobrazit maximum informací odpovídajícím a kvalitním způsobem. Krokování skrz řádky zdrojového kódu a zobrazení hodnot proměnných je dnes už běžnou záležitostí, a proto bych si v rámci vizualizačního nástroje představoval vždy alespoň o krok lepší podání, než poskytují například debugovací nástroje.

Za osobní přínos zhotoveného díla považuji osvojení nových schopností ve vývoji aplikací pro platformu Flash, což jsem si kladl za cíl už od samého začátku. Dále bych vyzvedl také získání nových vědomostí z oblastí RIA a vizualizace. Dalším aktivem je i fakt, že jsem měl možnost vyzkoušet si tvorbu této formy textu a tím jsem získal alespoň základní přehled o tom, co taková činnost obnáší.

Literatura

[1] Adobe Systems Incorporated: *Adobe - Adobe AIR: FAQ* [online]. 2009 [cit. 2009-04-20].
URL http://www.adobe.com/products/air/faq/

[2] Adobe Systems Incorporated: *Flash content reaches 99.0% of Internet viewers* [online]. 2009 [cit. 2009-04-21].
URL http://www.adobe.com/products/player_census/flashplayer/

[3] Aigner, W.: *Visualization - InfoVis:Wiki* [online]. 20060-06-07 [cit. 2009-04-26].
URL http://www.infovis-wiki.net/index.php?title=Visualization

[4] Aigner, W.: *Scientific Visualization - InfoVis:Wiki* [online]. 2007-07-16 [cit. 2009-04-26].
URL
http://www.infovis-wiki.net/index.php?title=Scientific_Visualization

[5] Bernard, B.: *Rich Internet Applications v roce 2008* [online]. 2008-04-25 [cit. 2009-03-19].
URL http://interval.cz/clanky/rich-internet-applications-v-roce-2008/

[6] Canoo Engineering AG: *Canoo UltraLightClient: Technical Concepts, Background Info* [online]. 2000-2009 [cit. 2009-04-09].
URL http://www.canoo.com/ulc/home/technicalconcepts.html

[7] Chen, C.: *Information Visualization: Byond The Horizon.* Springer, 2006 [cit. 2009-05-14], ISBN 1-84628-340-X.

[8] Diehl, S.: *Software visualization: visualizing the structure, behaviour, and evolution of software* [online]. Springer, 2007 [cit. 2009-05-15], ISBN 3540465049.
URL http://books.google.cz/books?id=8OADPxGJOa8C&printsec=frontcover

[9] Fry, B.: *Visualizing data.* O'REILLY, 2007 [cit. 2009-05-01], ISBN 0-596-51455-7.

[10] Google: *Product Overview – Google Web Toolkit – Google Code* [online]. 2009 [cit. 2009-04-17].
URL http://code.google.com/intl/cs/webtoolkit/overview.html

[11] Hopkins, D.: *What is OpenLaszlo, and what's it good for?* [online]. 2006-03-24 [cit. 2009-04-05].
URL http://www.donhopkins.com/home/124

[12] Kosara, R.: *Visualization Criticism – The Missing Link Between Information Visualization and Art* [online]. IEEE CS Press, 2007 [cit. 2009-04-26].
URL http://eagereyes.org/references/Kosara_IV_2007.html

[13] Kosara, R.: *What is Visualization? A Definition* [online]. 2008-07-24 [cit. 2009-04-25].
URL http://eagereyes.org/theory/Definition-of-Visualization.html

[14] Laszlo Systems, Inc.: *Architecture — OpenLaszlo* [online]. 2005-2009 [cit. 2009-04-05].
URL http://www.openlaszlo.org/architecture

[15] Moravec, Z.: *RIA - Rich Internet Applications* [online]. 2009-04-14 [cit. 2009-04-19].
URL http://programujte.com/?akce=clanek&cl=2009041200-ria-rich-internet-applications

[16] Novell: *MoonlightRoadmap - Mono* [online]. 2009 [cit. 2009-05-09].
URL http://www.mono-project.com/MoonlightRoadmap

[17] Pichlík, R.: *Google Web Toolkit* [online]. 2006-09-19 [cit. 2009-04-17].
URL http://interval.cz/clanky/google-web-toolkit/

[18] Smashing magazine: *Data Visualization: Modern Approaches* [online]. 2007-08-02 [cit. 2009-04-27].
URL http://www.smashingmagazine.com/2007/08/02/data-visualization-modern-approaches/

[19] Sun Microsystems, Inc.: *JavaFX: Features and Benefits* [online]. 1994-2009 [cit. 2009-04-17].
URL http://www.sun.com/software/javafx/features.xml

[20] University of Lugano: *Knowledge Domain Map* [online]. [cit. 2009-04-28].
URL http://www.elab.usilu.net/usi10anni/knowledge_domain_maps/visualization_scholars/

[21] Wikipedia.org: *Instability.jpg* [online]. 2007-03-16 [cit. 2009-04-27].
URL http://upload.wikimedia.org/wikipedia/en/1/17/Instability.jpg

[22] Wikipedia.org: *Rich Internet application* [online]. 2009-04-09 [cit. 2009-04-20].
URL http://en.wikipedia.org/wiki/Rich_Internet_application

[23] Wikipedia.org: *Ajax (programming)* [online]. 2009-04-13 [cit. 2009-04-16].
URL http://en.wikipedia.org/wiki/Ajax_(programming)

[24] Wikipedia.org: *Microsoft Silverlight* [online]. 2009-04-16 [cit. 2009-04-16].
URL http://en.wikipedia.org/wiki/Silverlight

Dodatek A

Obsah CD

A.1 Aplikace

[CD-ROM jednotka]/Aplikace/index.html

A.2 Aplikace - zdrojový kód

[CD-ROM jednotka]/Aplikace-zdrojovy_kod/

A.3 Dokumentace zdrojového kódu

[CD-ROM jednotka]/Dokumentace/index.html

A.4 Technická zpráva - zdroj

[CD-ROM jednotka]/Technicka_zprava/

A.5 Kompletní manuál k aplikaci

[CD-ROM jednotka]/Manual.pdf

A.6 Plakát

[CD-ROM jednotka]/Plakat.pdf

A.7 Technická zpráva

[CD-ROM jednotka]/xmalci00.pdf