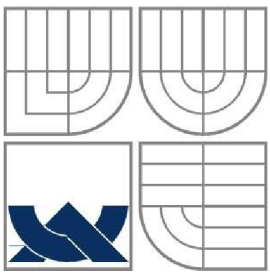
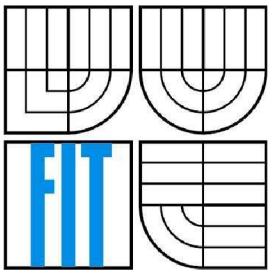


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZŠÍŘENÍ WEBOVÝCH PROHLÍŽEČŮ V NATIVNÍM KÓDU

NATIVE CODE WEB BROWSER EXTENSIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VOJTĚCH VÍTEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2011

Abstrakt

Webové aplikace a prohlížeče jsou v dnešní době vyvíjeny a modernizovány neuvěřitelně rychlým tempem – dá se říct, že pokrok na poli internetových technologií je nezastavitelným fenoménem posledního desetiletí. Potřeba využít maximálního výkonu procesoru ve webové aplikaci k dosažení plynulého a příjemného uživatelského zážitku je enormní a každým rokem přináší, kromě neustálého vylepšování stávajících prostředků, několik nových technologií. Tato bakalářská práce se zabývá tvorbou rozšíření webových prohlížečů v nativním kódu, jejichž primárním cílem je právě využití maximálního výkonu procesoru a též snaha o vylepšení uživatelského zážitku při prohlížení webových stránek.

Abstract

Nowadays, web applications and browsers are undergoing rapid development - we can say that the progress of Internet technologies is unstoppable phenomenon of the last decade. The need for the best-possible CPU performance in web applications to achieve a smooth user experience is enormous – besides the continuous improvements of existing technologies, we can see several new technologies arising every year. This thesis deals with development of native code web browser extensions whose primary purpose is to use maximum CPU performance as well as efforts to improve the user experience when viewing web pages.

Klíčová slova

Plugin, NPAPI, PPAPI, NaCl, PNaCl, Native Client, Pepper, Salt, Pinnacle, Webová aplikace, Rozšíření prohlížeče, Firefox Add-on, Chrome Extension, Opera Widget, Sandbox, JavaScript, DOM, Ajax, Assembler, SSE

Keywords

Plugin, NPAPI, PPAPI, NaCl, PNaCl, Native Client, Pepper, Salt, Pinnacle, Web application, Browser extension, Firefox Add-on, Chrome Extension, Opera Widget, Sandbox, JavaScript, DOM, Ajax, Asembler, SSE

Citace

Vojtěch Vítek: Rozšíření webových prohlížečů v nativním kódu, bakalářská práce, Brno, FIT VUT v Brně, 2011

Rozšíření webových prohlížečů v nativním kódu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, PhD. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Vítek
12. května 2011

Poděkování

Děkuji svému vedoucímu Ing. Radku Burgetovi, PhD. za trpělivost, ochotu a hodnotné rady, které mi poskytl při psaní této technické zprávy. Dále děkuji Marku Vantuchovi, který se podílel na vývoji uživatelské části projektu CoNetServ, o které se tato technická zpráva též okrajově zmiňuje. Dále děkuji Zbyňkovi Michlovi a celé komisi soutěže VIP 2 a VIP 3 laboratoří CZ.NIC, z.s.p.o. za hodnotné připomínky v průběhu vývoje projektu CoNetServ a kvalitní zpětnou vazbu při hodnocení výsledku. A v neposlední řadě také děkuji desítkám testerů a uživatelů jednotlivých balíčků rozšíření CoNetServ, jmenovitě pak Olegu Afanasyevovi, kteří poskytovali zpětnou vazbu a podíleli se tak na zkvalitňování aplikace.

Obsah

1 Úvod

Tato bakalářská práce se zabývá tvorbou rozšíření webových prohlížečů v nativním kódu, jejichž primárním cílem je využití maximálního výkonu procesoru, systémových prostředků a též snaha o vylepšení uživatelského zážitku při prohlížení webových stránek. Dokument popisuje technologie, které lze v dnešní době efektivně využít k tvorbě zásuvných binárních modulů a balíčků rozšíření do jednotlivých webových prohlížečů.

1.1 Shrnutí účelu a cíle projektu CoNetServ

Praktickou částí této práce je projekt CoNetServ (Complex Network Services extension, v překladu „rozšíření pro komplexní síťové služby“), jehož cílem bylo vytvoření aplikace, podporované jako rozšíření co možná největším počtem současných moderních webových prohlížečů, která poskytuje informace o síti a konektivitě uživatele, a to nejlépe v přehledné a srozumitelné grafické podobě.

Síťové nástroje pro zisk základních informací o stavu sítě jsou známy většinou jen administrátorům a pokročilým uživatelům konkrétního operačního systému. Běžný uživatel téměř zcela jistě nezná terminálové/konzolové programy (a natož jejich přepínače a parametry), kterými by např. zjistil, zda-li je vzdálený server dostupný, jakou má latenci jeho připojení k danému cíli v rámci sítě Internet či jak je „daleko“ daný počítač v lokální síti.

Ideou projektu tedy bylo zpřístupnit několik základních síťových nástrojů a informací v grafické formě běžnému uživateli, ihned po stisku jediného tlačítka, a to zcela automatizovaně a bez nutnosti zdlouhavého nastavování programu či zadávání pokročilých parametrů.

1.2 Obsah technické zprávy

Tato technická zpráva popisuje všechny kroky, které jsem podnikl k úspěšnému vytvoření a následnému zveřejnění aplikace CoNetServ.

- **Analýza a výběr vhodných technologií**

  První etapou vývoje aplikace CoNetServ byl výběr vhodné technologie pro tvorbu rozšíření webového prohlížeče v nativním kódu. Tomuto tématu je věnována kapitola 2, která se mj. podrobněji zabývá i novými technologiemi, které jsou toho času ve fázi experimentálního vývoje.

- **Implementace NPAPI pluginu**

  Nejobsáhlejším bodem této technické zprávy je samotná implementace NPAPI pluginu – technologie rozšíření webového prohlížeče v nativním kódu pomocí standardizované architektury. Této části práce je věnována kapitola 3.

- **CoNetServ JavaScript API**

  Další částí tohoto dokumentu je popis návrhu objektového rozhraní projektu CoNetServ, jehož implementace je přístupná prostřednictvím kořenového objektu a technologií DOM a NPRuntime ve front-endu prohlížeče, pomocí jazyka JavaScript (kapitola 4).

- **Implementace balíčků rozšíření a jejich distribuce**

  Kapitola 5 se zabývá implementací balíčků rozšíření pro jednotlivé prohlížeče a jejich následnou distribucí.

- **Zveřejnění projektu a výsledné statistiky**

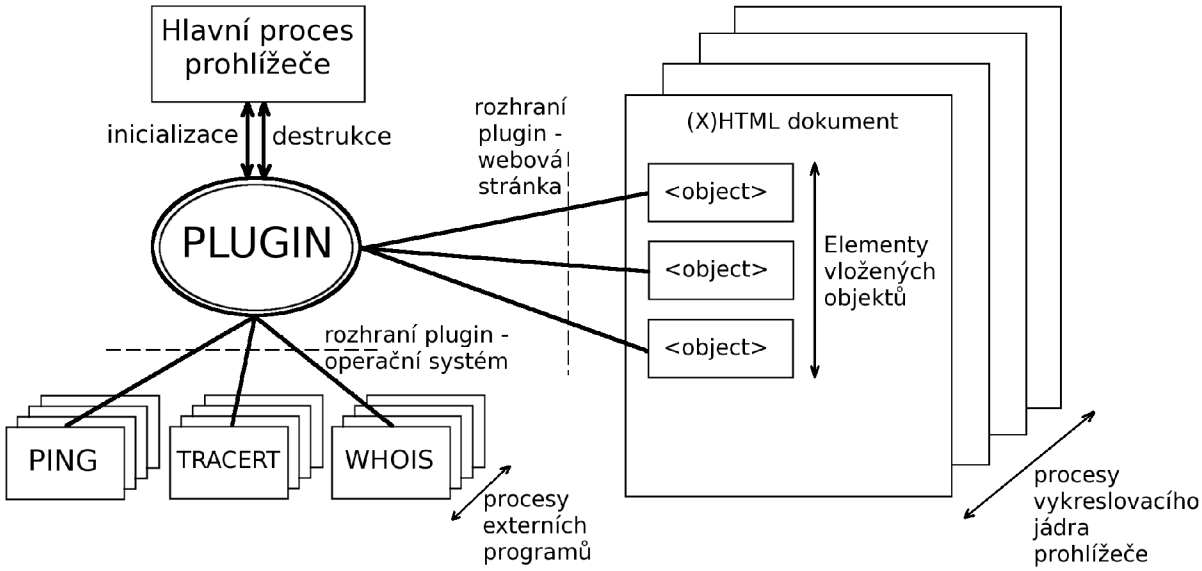
  Závěrečnou kapitolou této práce je stručný popis průběhu zveřejnění projektu na oficiálních repozitářích jednotlivých rozšíření prohlížečů a statistiky užití výsledných aplikací (kapitola 6).

2 Analýza a výběr vhodných technologií

Návrh projektu CoNetServ se dá z technického hlediska rozdělit na dvě části – externí a lokální služby. V prvním případě je nutné využít výstup předem určené externí služby. K tomu je tedy potřeba vytvořit parsér výstupních dat, která klient získá požadavkem na předem určený server – například pomocí asynchronního požadavku, zprostředkovaného technologií Ajax (Asynchronous JavaScript and XML). Získaná data poté stačí jen zpracovat a zobrazit pomocí nějakého toolkitu či grafické knihovny.

V případě druhém, u lokálních služeb, je nutné zpracovat výstup určitého externího programu, spuštěného lokálně na straně klienta. Spuštěný program typicky vyžaduje přístup k systémovým voláním, což značně limituje počet použitelných a vhodných technologií, které nejsou platformně závislé a přitom jsou podporovány co možná největším počtem současných webových prohlížečů. Této problematice se věnuje následující kapitola 2.1.

2.1 Plugin – zásuvný binární modul prohlížeče



Obrázek 1: Znázornění hledané technologie pro projekt CoNetServ

Vztah mezi požadovanou technologií, která spouští lokální služby (tedy typicky pluginem – zásuvným binárním modulem prohlížeče), webovým prohlížečem a procesy externích programů běžících v prostředí operačního systému, je znázorněn na obrázku 1. Výběru konkrétní vyhovující technologie jsou věnovány následující podkapitoly.

2.1.1 Skriptovací jazyky, Adobe Flash a Microsoft Silverlight

Nutnost využít systémová volání automaticky vyloučila použití skriptovacích jazyků standardu ECMAScript a tudíž i jeho kompatibilních a odvozených implementací, běžících na straně klienta, jako jsou JavaScript, JScript, ActionScript, nebo Tamarin JIT. Vyloučila také další rozšířené technologie, jako jsou např. Adobe Flash, či Microsoft Silverlight. Všechny zmiňované technologie totiž běží ve svém specifickém virtuálním stroji a nemají žádnou možnost přístupu k souborům na koncovém počítači uživatele, ani práva spouštět systémová volání, či externí programy.

2.1.2 Java applet

Podobný problém, s nedostatečnými právy k systémovým voláním jako u předchozích technologií, vyvstal také při úvaze o naprogramování Java appletu. Java, jakožto jazyk samotný, sice systémová volání podporuje, ale v rámci appletů je ve výchozím nastavení zakazuje.

Pouze tehdy, pokud je applet (soubor s příponou .jar) podepsán klíčem, který se shoduje s předem vygenerovaným certifikátem - který musí uživatel předem manuálně importovat, potvrdit a vytvořit pro něj bezpečnostní soubor - tak je bytový kód spouštěn bez omezení [1], tedy mimo speciální sandbox v rámci JVM (Java Virtual Machine). Především kvůli tomuto omezení se Java applet jevil jako nedostatečně vhodné řešení.

2.1.3 Native Client

Technologie Native Client společnosti Google, které je podrobně věnována kapitola 2.4, je sice zcela přenositelná mezi různými platformami, ale již z podstaty návrhu není v jejím rámci možné využít systémových volání, spouštět externí programy či nahrávat dynamické knihovny a přistupovat k nim pomocí nějakého rozhraní. Native Client v době vývoje programu neposkytoval žádné API, které by zmiňovanou funkcionalitu mohlo jakkoliv nahradit a proto nebylo možné tuto technologii pro potřeby projektu CoNetServ použít.

2.1.4 Microsoft ActiveX a VBScript

Další dvě technologie, Microsoft ActiveX a VBScript (Visual Basic Scripting Edition), byly vyloučeny hned z několika důvodů. Prvním je téměř nulová multiplatformní nezávislost, neboť jsou obě technologie podporovány pouze na operačních systémech Microsoft Windows. Druhým důvodem bylo velmi limitující omezení pouze na prohlížeče s jádrem Trident, tedy na Microsoft Internet Explorer a jeho deriváty s upraveným grafickým uživatelským prostředím (s výjimkou částečné podpory ActiveX v některých verzích prohlížeče Mozilla Firefox).

Třetím, nejdůležitějším důvodem, byla bezpečnostní rizika spojená s použitím těchto technologií a tedy i reálná možnost znemožnění spuštění vytvořené aplikace antivirovým programem či opravným balíkem systému (tzv. service packem). Je známo mnoho bezpečnostních děr a útoků prostřednictvím ActiveX objektů nebo speciálními konstrukcemi Visual Basic skriptů, které zákeřně čekají na své oběti ve zdrojových kódech webových stránek či (X)HTML emailů. Je tedy pravděpodobné, že by bylo spuštění cílové aplikace na naprosté většině dnešních systémů buď zcela zakázáno a nebo alespoň výrazně omezeno.

2.1.5 XPCOM

XPCOM (Cross Platform Component Object Model) je komponentový model, používaný v aplikacích Mozilla, dostupný de facto na všech platformách. Na rozdíl od dříve jmenovaných technologií umožňuje jak manipulaci s lokálními soubory na počítači cílového uživatele, tak spouštět externí programy – a to prostřednictvím objektu *nslProcess* [2].

Na světlo ale opět vyvstaly určité pochybnosti, zda použít právě XPCOM, jako onu vhodnou technologii pro vývoj aplikace CoNetServ. Tou hlavní byl fakt, že XPCOM je podporován pouze vykreslovacím jádrem Gecko, což by přineslo výrazné omezení v podobě vázanosti pouze na prohlížeče Mozilla Firefox, SeaMonkey, Epiphany (do verze 2.26.3), Camino, Flock (do verze 2.6.2) a několik dalších, méně známých prohlížečů, typicky založených na zdrojových kódech Firefoxu. Muselo by tedy být upuštěno od podpory, toho času velkých hráčů na poli světových prohlížečů, jako jsou např. Internet Explorer, Opera, Google Chrome (Chromium) a Safari.

Druhá, neméně výrazná pochybnost, byla způsobena nepřesvědčivými zprávami o podpoře XPCOM v budoucích verzích jádra Gecko, kterou ještě navíc umocnilo vyjádření technologického šéfa firmy Mozilla [3], který se zmínil o zrušení podpory XPCOM v jádru verze 2.0 a vyšším – a tedy např. Firefoxu 4.0.

2.1.6 NPAPI

Jako vítězná technologie pro projekt CoNetServ byla nakonec vybrána architektura NPAPI (Netscape Plugin Application Programming Interface), která umožňuje nejen manipulovat s lokálními soubory na počítači uživatele, ale také dovoluje využívat systémových volání a spouštět externí programy, paralelní procesy a vlákna.

Nejdůležitějším faktorem pro výběr technologie NPAPI bylo její velké rozšíření a podpora téměř všemi významnými prohlížeči, a to na téměř všech dostupných platformách. Jediným kazem na tomto tvrzení je chybějící podpora Windows Internet Exploreru (dříve Microsoft Internet Exploreru), který NPAPI podporoval pouze do verze 5.5 SP2 [4][5].

Microsoft od podpory NPAPI upustil ve prospěch technologie ActiveX [5] - v oficiálním vyjádření kvůli bezpečnostním důvodům [5], ale dle mého názoru především z důvodu upevnění monopolního postavení na trhu operačních systémů a webových prohlížečů. Toho času měl totiž Internet Explorer kolem 87% podílu veškerého trhu [6] a konkurence tedy nemohla dostatečně pružně reagovat na předem neoznámenou [5] a natolik významnou změnu, jako je okamžité zrušení podpory NPAPI ve prospěch proprietární technologie, kterou ostatní prohlížeče nemohly (a dodnes nemůžou) implementovat a jakkoli podporovat. Architektura NPAPI je blíže popsána v následující kapitole.

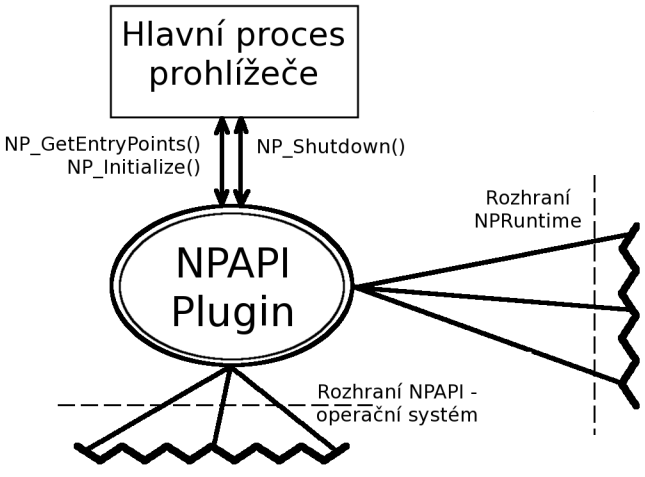
2.2 Netscape Plugin API (NPAPI)

NPAPI je multiplatformní architektura zásuvných modulů, která byla původně vytvořena pro webový prohlížeč Netscape Navigator 2.0 a později byla převzata mnoha dalšími konkurenčními prohlížeči.

Nejznámějšími pluginy architektury NPAPI jsou například Adobe Flash Player, Adobe Reader (PDF), Java, QuickTime, RealPlayer, Adobe Shockwave Player, Windows Media Player a mnoho dalších pluginů, které většinou přidávají funkcionalitu dalšího multimediálního formátu.

2.2.1 Vstupní body NPAPI

Vývojář NPAPI pluginu musí pro správné načtení a fungování pluginu implementovat cca 15 funkcí, z nichž jsou nejdůležitější tři - funkce `NP_GetEntryPoints()` pro získání pozic vstupních bodů dílčích funkcí, `NP_Initialize()` pro inicializaci a `NP_Shutdown()` pro destrukci pluginu. Tyto tři hlavní funkce musí být mezi exportovanými symboly výsledné sdílené binární knihovny, aby ji mohl prohlížeč správně načíst jako plugin NPAPI.



Obrázek 2: Architektura NPAPI a rozhraní NPRuntime vložené do výseku obrázku 1

Na obrázku 2 lze vidět jednotlivá rozhraní mezi architekturou NPAPI, rozšířením NPRuntime a operačním systémem, zasazená do výseku obrázku 1. Znázorněn je též vztah pluginu k hlavnímu procesu prohlížeče a jednotlivé inicializační a destrukční kroky.

2.2.2 Slabý „standard“

NPAPI je označován za tzv. slabý „standard“ [7], protože jeho architektura nebyla standardizována oficiální cestou. Každý prohlížeč tedy technologii NPAPI implementuje svým vlastním způsobem, což přineslo jistou dávku nekompatibility mezi prohlížeči (např. odlišné inicializační kroky, nutnost různých vstupních bodů atp.) a také odlišnosti mezi různými platformami – a to především rozdílný způsob ošetření událostí a vykreslování 2D a 3D grafiky. Tento problém se snaží řešit rozšíření PPAPI, popsané v kapitole 2.3.

2.2.3 NPRuntime

NPRuntime je název technologie, která rozšiřuje funkcionalitu originální architektury NPAPI o možnost skriptování, po vzoru svých předchůdců LiveConnect a XPConnect, ovšem bez závislosti na technologii Java.

Toto rozšíření bylo vytvořeno na konci roku 2004 v rámci spolupráce vývojářů prohlížečů Mozilla 1.7.5+/Firefox, Safari a Opera a umožňuje přístup k objektům uvnitř NPAPI, například prostřednictvím DOM a jazyka JavaScript.

2.3 Pepper Plugin API (PPAPI)

Pepper, v překladu „Pepř“, je projektem firmy Google, jehož původním předsevzetím bylo vyřešit problémy přenositelnosti a slabého výkonu architektury NPAPI. Tento návrh nakonec narostl o implementaci API pro generování 2D a 3D grafiky a rozhraní pro práci se zvuky.

První implementace Pepper API bylo navrhnuta s ohledem na značnou kompatibilitu s původním návrhem NPAPI kvůli usnadnění jejího používání stávajícími pluginy. Tento návrh byl implementován ve vydání prohlížeče Chrome 5.

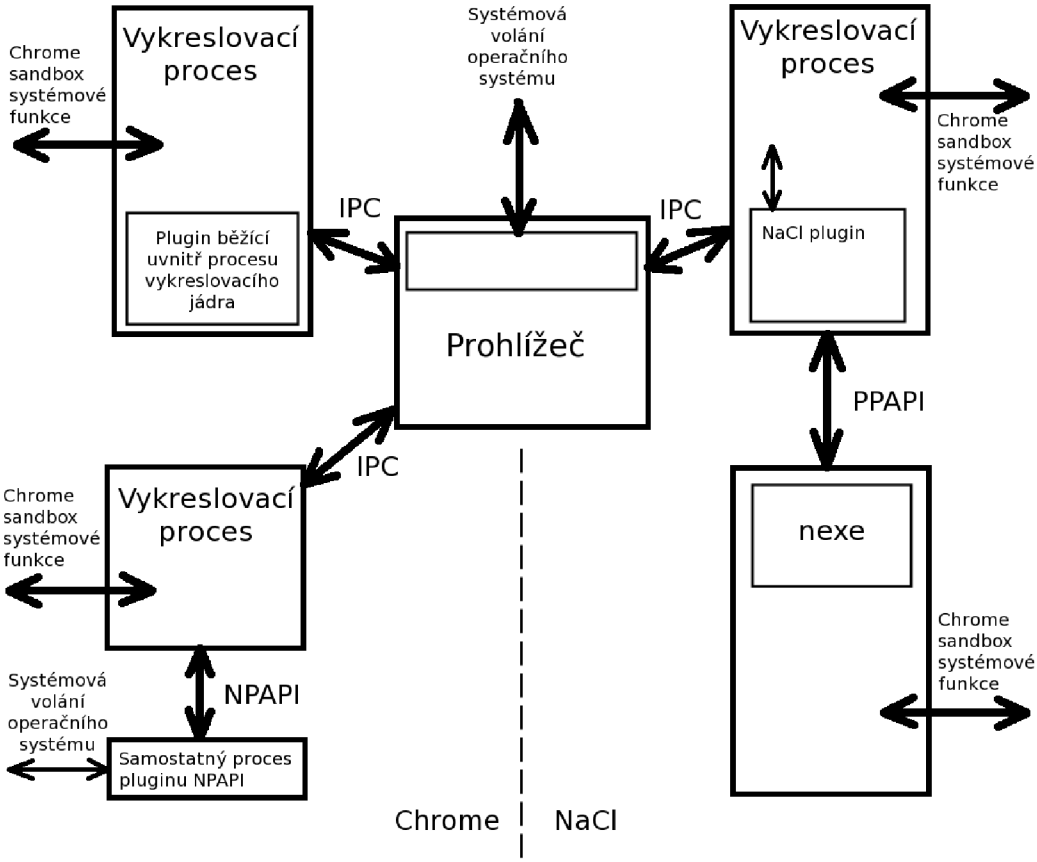
Poté bylo navržené API kompletně přepracováno (na úkor kompatibility s původním NPAPI) s cílem výrazně zvýšit výkon rozhraní. Toto revidované API je označováno jako „PPAPI“ nebo také „Pepper2“ a je dostupné od verze prohlížeče Chrome 6.

2.4 Native Client (NaCl)

Native Client je „sandboxing“ technologie pro bezpečné spouštění platformně nezávislých binárních spustitelných souborů uvnitř webového prohlížeče. Tento ambiciózní projekt firmy Google nese též název „Salt“, odvozený od chemické značky a zkratky projektu „NaCl“.

Native Client umožňuje výkonově náročným webovým aplikacím (tj. např. online hrám, médiím, rozsáhlým datovým analýzám či vizualizacím) přistupovat bezpečně a v reálném čase ke zdrojům procesoru uživatelova počítače a tím mj. omezit náročný síťový provoz.

Příklad napojení technologie NaCl do webového prohlížeče ilustruje obrázek 3.



Obrázek 3: Kombinovaná ilustrace prohlížeče Chrome, vykreslovacího jádra, samostatných a uvnitř běžících procesů pluginů a technologie NaCl

2.4.1 Native Client SDK, sandboxing technologie

Native Client SDK je sada vývojových nástrojů pro vytváření spustitelných souborů .nexe, které představují binární podobu NaCl modulů. Binární soubory NaCl jsou klasickými spustitelnými soubory konkrétní architektury procesoru, které jsou ovšem multiplatformní, protože neobsahují žádné systémově závislé volání ani práci s externími knihovnami. NaCl modul 32-bitové architektury je tedy spustitelný jak v unixových operačních systémech, tak pod systémem Microsoft Windows.

Soubory .nexe, vytvořené pomocí speciálně pozměněných nástrojů gcc, binutils a gdb, omezují výsledný binární kód pouze na sadu předem určených bezpečných instrukcí (příkladem zakázané instrukce je např. instrukce RET) a též se starají o správné zarovnání strojových instrukcí a jejich parametrů, aby například instrukce JMP nemohla skočit mimo rámec modulu nebo doprostřed jiné instrukce a pozměnit tak význam následného kontextu. Správnost .nexe souborů je vždy kontrolována též statickou analýzou na straně klienta, čímž je zaručena bezpečnost a důvěryhodnost daného NaCl modulu.

Obrovskou výhodou spustitelných modulů technologie Native Client je možnost využít rozšířených instrukcí procesoru, především SSE, a též zabudované podpory API rozhraní pro přístup k 2D a 3D grafice a zvukové knihovně (prostřednictvím Pepper API, viz kapitola 2.3).

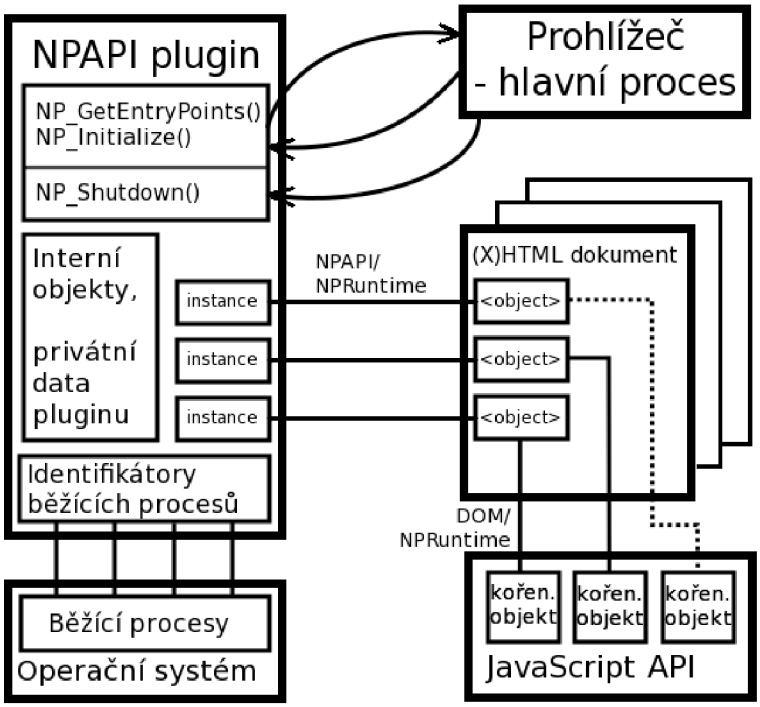
Technologie Native Client je toho času stále ve fázi experimentálního vývoje a není proto dostupná ve výchozím nastavení v žádném webovém prohlížeči současnosti. Podporu lze zapnout v prohlížeči Google Chrome prostřednictvím přepínače `–enable-nacl` nebo v nastavení prohlížeče na speciální adrese `about:flags`. Několik světových firem se ovšem snaží o rychlé prosazení této technologie, kromě firmy Google též například Adobe, a dle posledních informací je plánována podpora NaCl v prohlížeči Mozilla Firefox 4.x [8], což ve výsledku donutí k přijmutí technologie i další konkurenční prohlížeče.

2.4.2 PNaCl

Projekt PNaCl, též vyslovován jako „Pinnacle“, je projekt postavený na technologii Native Client, soustředící se na ještě větší přenositelnost jednotlivých binárních modulů. K tomu PNaCl používá formát bitového kódu překladače LLVM (Low-Level Virtual Machine), který zabaluje funkcionalitu aplikace do výsledných balíčků neutrálního kódu, který je poté na straně klienta přeložen do cílové architektury klienta.

Kapitoly 2.3 a 2.4 byly volně převzaty z [7], [9] a [10].

3 Implementace NPAPI pluginu



Obrázek 4: Projekt CoNetServ: Ilustrace NPAPI pluginu ve vztahu k operačnímu systému, hlavnímu procesu prohlížeče a jednotlivým procesům vykreslovacího jádra prohlížeče

3.1 Zdroje a použité nástroje při vývoji

Binární pluginy do prohlížečů jsou vhodné pouze ke zcela specifickým účelům a vytváří je zcela zanedbatelná, minoritní, skupina vývojářů. Proto také existuje jen značně omezené množství publikací a zdrojů, které se tomuto tématu věnují.

3.1.1 Programátorské dokumentace, referenční manuály

Hlavními zdroji, ze kterých jsem čerpal, byly především programátorské dokumentace a referenční manuály vývojářů jednotlivých prohlížečů, např. Gecko Plugin API Reference [11] a The Opera plug-in interface [12] (tyto dva referenční manuály jsou svým obsahem téměř shodné, liší se pouze v několika implementačních detailech).

V neposlední řadě jsou velmi kvalitním zdrojem informací přímo zdrojové kódy jednotlivých open-source projektů a referenční ukázky – tzv. příklady „Hello World!“.

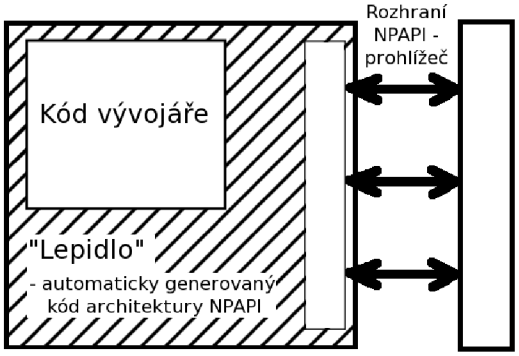
3.1.2 Framework FireBreath

Existuje dokonce několik frameworků, jejichž úkolem je usnadnit tvorbu přenositelného pluginu. Prvním zástupcem je projekt FireBreath, jehož výčet funkcionality [13] uvádí kompatibilitu s prohlížeči Gecko/Firefox, Google Chrome, Apple Safari a dokonce Internet Explorer (prostřednictvím ActiveX) na platformě Windows, dále Gecko/Firefox a Apple Safari na platformě Darwin a konečně Gecko/Firefox na platformě GNU/Linux.

Po vytvoření projektu, přesně podle instrukcí autora, jsem byl projektem spíše zklamán. Výsledkem byl pouze neskutečně nabalený zdrojový kód, obsahující mnoho obskurních výrazů a vnořených podmíněných výrazů preprocesoru jazyka C/C++, složený z mnoha desítek až stovek souborů. Bohužel ani po několikahodinové snaze o opravu mnoha chybových hlášení ve vygenerovaných zdrojových souborech, nebylo možné projekt přeložit. Tento pokus tedy ztroskotal již při prvním pokusu o vytvoření knihovny pomocí překladače *gcc*, na kterém mi, kvůli jeho přenositelnosti, záleželo nejvíce.

3.1.3 Framework Nixysa

Další framework pro tvorbu NPAPI pluginů, který jsem testoval a stojí alespoň za krátkou zmínku, je Nixysa. Tento projekt je svou podstatou velmi podobný výše zmiňovanému projektu FireBreath, ovšem s tím rozdílem, že je kompletně napsán v jazyku Python. Vývojáři na svých wiki stránkách trefně popisují, že framework nedělá de facto nic jiného, než že vytváří tzv. „lepidlo“ [14], tedy kód, který obaluje vlastní kód potenciálního vývojáře automaticky generovanými konstrukcemi rozhraní NPAPI – viz obrázek 5.



Obrázek 5: Ilustrace termínu "lepidlo" (v anglickém originále "glue") v rozhraní NPAPI, použitém mj. v projektu Nixysa

V době, kdy jsem framework Nixysa testoval, se mi bez obtíží podařilo vytvořil knihovnu, fungující jako zásuvný modul v prohlížeči Google Chrome na platformě Microsoft Windows. Tehdy ovšem ještě nebyla dokončena podpora pro alternativní prohlížeče či operační systémy a proto jsem

musel od použití frameworku Nixysa ve výsledku upustit. Při psaní této technické zprávy je situace již jiná – vývojáři na oficiálním webu projektu slibují velmi širokou podporu alternativ [14], která by mě pravděpodobně již přiklonila na svou stranu.

3.1.4 Ukázkové kódy a open-source projekty

Velmi kvalitním zdrojem pro tvorbu NPAPI pluginu je kolekce jednoduchých příkladů projektu Mozilla, ač si jsou její autoři vědomi její neaktuálnosti:

„Příklady už bohužel nemusí být přeložitelné na všech dostupných platformách. Existují ale plány na pročištění stavu stávajících příkladů – např. vylepšit jejich strukturu a organizaci, aktualizovat překladový systém a tím dosáhnout přeložitelnosti na všech platformách. Nicméně, ač dnešní situace nedovoluje přeložit příklady na všech platformách, stále můžou jejich zdrojové kódy sloužit jako velmi hodnotná reference.“ [15]

Za další kvalitní zdroj považuji zdrojové kódy dalších open-source projektů, z kterých jsem čerpal inspiraci. Příkladem jsou například zdrojové kódy projektů NSPluginWrapper a GNU Gnash dostupné z [16], resp. [17].

3.1.5 Projekt NPSimple

Jako základ projektu CoNetServ jsem ale nakonec vybral projekt NPSimple. NPSimple je velmi malý a prostý projekt, který byl vytvořen s jedním cílem – usnadnit potencionálním vývojářům tvorbu jednoduchého NPAPI pluginu – a právě kvůli tomuto důvodu je také šířen pod open-source licencí GPLv2 (GNU General Public License verze 2).

Projekt je založen na překladovém systému GNU Automake, s jehož pomocí je možno dosáhnout široké podpory nejrůznějších platforem a autoři také slibují ve verzi NPSimple 0.3 kompatibilitu s nejvýznamnějšími prohlížeči, které podporují architekturu NPAPI – podrobně viz tabulka 3.1.

Jméno prohlížeče/vykreslovacího jádra	Platforma
Firefox 3.5	Android
Firefox 3.0, Safari 3.1.2	Darwin
Iceweasel 2.0, Iceweasel 3.0, Opera 9.5	Debian Linux
Firefox 2.0, Firefox 3.0, Opera 9.27	Ubuntu Linux
WebKit/GTK c6c3f8ca4996a96a1c7e9d1d	Ubuntu Linux
Firefox 3.0, Opera 9.5	Windows XP
Torchmobile's IRIS browser 1.0.13	Windows Mobile 6

Tabulka 3.1: Projekt NPSimple verze 0.3 – podporované platformy a prohlížeče

3.2 Překladový systém

3.2.1 Problémy a nedostatky původního systému Automake

Zdrojové kódy projektu NPSimple s sebou přinesly závislost na původní překladový systém GNU Automake na unixových systémech a také speciální projektové soubory vývojových prostředí Visual Studio a Xcode na systémech Microsoft Windows, resp. Mac OS X. Ukázalo se, že to bylo nedostatečné řešení, které přineslo celou řadu dalších problémů. Například téměř jakákoliv změna v překladových souborech Makefile přinesla nekompatibilitu mezi jednotlivými unixovými platformami – a to především při změnách závislostí na knihovnách třetích stran či při změnách absolutních cest k hlavičkovým souborům. Původní návrh totiž nepočítal s žádným nástrojem, který by tyto hodnoty zobecnil a generoval je až za běhu překladu. Bylo tedy nutné pro každou platformu vytvořit speciální balík změn, tzv. changeset, který se dal aplikovat až před spuštěním překladu (v praxi se dalo využít tzv. „branchování“ v distribuovaném systému správy verzí *git*, kterému je věnována kapitola 3.3).

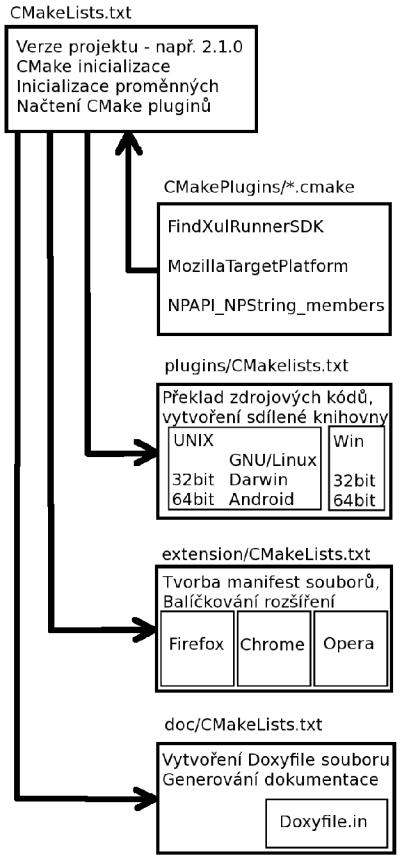
Kvůli výše uvedeným důvodům jsem se rozhodl kompletně přepsat překladový systém a použít při tom moderní technologie. Po hledání a testování moderních překladových systémů, kde byly brány v úvahu mj. systémy Apache Ant [18], Apache Maven [19], qmake [20] a Scons [21], jsem nakonec vybral jako vhodný překladový nástroj systém CMake [22][23].

3.2.2 Překladový systém CMake

Překladový systém CMake byl namísto původního Automake vybrán především z důvodu výrazné přenositelnosti mezi platformami a velké oblibě v komunitě open-source vývojářů, např. The KDE® Community [24].

Soubory CMake mají sice svou vlastní specifickou syntaxi (narozdíl například od výše zmiňovaného projektu SCons, jehož soubory mají syntaxi jazyka Python), ta se ale dá po přečtení několika stran tutoriálu na oficiálních stránkách projektu [22] nebo prvních tří kapitol z [23] poměrně snadno a rychle pochopit.

3.2.3 Hierarchie souborů CMake



Obrázek 6: Hierarchie souborů překladového systému CMake v projektu CoNetServ

Soubory CMake byly v projektu CoNetServ rozděleny do několika sekcí, jak je vidět na obrázku 6. Hlavním souborem je kořenový soubor `CMakeLists.txt`, tzv. zavaděč. Tento soubor obsahuje inicializační kroky překladového systému samotného, deklaruje a inicializuje několik dalších privátních proměnných (např. verzi projektu nebo seznam autorů načtených ze souboru `AUTHORS.txt`), importuje uživatelsky definované pluginy a dále načítá konfigurační soubory v jednotlivých podadresářích. Soubor také rozhoduje, zda je překlad v režimu ladění či vydávání.

Uživatelsky definované pluginy doplňují chybějící funkcionalitu výchozích pluginů, které CMake poskytuje ve své základní verzi. Implementoval jsem plugin `FindXulRunnerSDK`, který vyhledává knihovnu XULRunner na cílovém systému uživatele, dále plugin `MozillaTargetPlatform`, který vytváří řetězec znaků požadovaný konkrétní architekturou a verzí prohlížeče Firefox a poslední plugin `NPAPI_NPString_members`, který zjišťuje implementačně závislé rozdíly NPAPI knihovny automatickou kompilací jednoduchých příkladů v jazyku C.

Soubor `CMakeLists.txt` v podadresáři `plugin/` zajišťuje samotnou kompilaci binárního výsledku sdílené knihovny a také vytváří potřebné soubory k ladění projektu. Dle architektury načítá potřebné informace z podadresářů `unix/`, `win/` a `apple/`.

Podadresář `extension/` obsahuje konfigurační soubor `CMakeLists.txt` pro vytvoření *manifest* souborů jednotlivých rozšíření prohlížečů Firefox, Chrome a Opera (viz kapitola 5), které poté zabalí předem určeným způsobem s dalšími soubory do výsledných balíčků.

Poslední podsekcí hierarchie překladového systému je soubor `CMakeLists.txt` v podadresáři `docs/`. Ten je zodpovědný za vygenerování konfiguračního souboru `Doxyfile`, kterým poté pomocí nástroje Doxygen vytvoří programátorskou dokumentaci celého projektu CoNetServ.

Jednotlivé vztahy mezi vyjmenovanými sekcemi jsou názorně zobrazeny na obrázku 6.

3.2.4 Překlad projektu

Překladový systém CMake vytvoří při překladu projektové soubory vybraného vývojového prostředí – např. Visual Studio projekt v prostředí Windows, Kdevelop projekt na Linuxu či XCode projekt na platformě Darwin/Mac OS X. Vývojář poté spustí vytvořený projekt, vyvíjí v něm a vytváří výsledné binární soubory.

V unixových systémech se často generují soubory Makefile (namísto výše uvedených projektových souborů), kterými následně vývojář překládá projekt – a to většinou z konzole. Oproti ručnímu psaní souborů Makefile dodává CMake zabudovanou podporu smysluplných chybových hlášek, barevný výstup programu, *progress bar* a vyniká ve srovnání s jinými nástroji především výbornými výsledky v rychlosti generování souborů a překladu samotného [24][25].

CMake je tedy jakýmsi multiplatformním prostředníkem, který generuje výsledné projekty či souborové celky, které poté vývojář ovládá – a to bez potřeby hlubokých znalostí cílového vývojového prostředí. Podrobný návod pro přeložení projektu na různých platformách, včetně příkladů použití jednotlivých příkazů, je dostupný v příloze C a také v souboru `README.txt` na přiloženém CD (viz příloha A).

3.3 Distribuovaná správa verzí

Aplikace CoNetServ byla od počátku vyvíjena s pomocí verzovacího nástroje git [26], který zachovává celou historii změn všech souborů projektu. Repozitář se zdrojovými texty je volně dostupný na internetové adrese https://github.com/V-Teq/CoNetServ a je též k dispozici na přiloženém CD (viz příloha A).

3.4 Kompilace knihovny pro různé platformy

Jak již bylo nastíněno v kapitole 3.2.2, projekt CoNetServ byl překládán pomocí několika různých nástrojů, jejichž dílčí soubory byly vygenerovány překladovým systémem CMake. Na platformě GNU/Linux byly vygenerovány soubory Makefile a výsledná knihovna byla generována pomocí překladače gcc a linkeru ld.

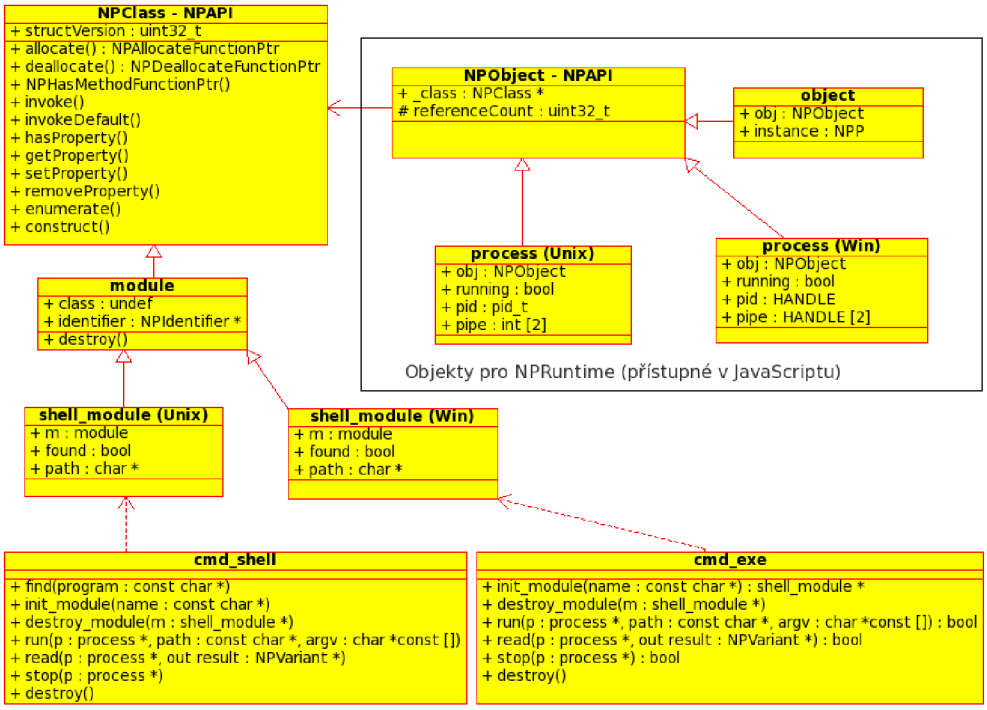
Na platformě Windows bylo použito vývojového prostředí Microsoft Visual Studio a integrovaného překladače a linkeru Microsoft Visual C++ 2010 Express edition. A konečně na platformě Darwin/Mac OS X bylo využito vývojového prostředí XCode (s výchozím překladačem gcc a linkerem ld) a kompilátoru/lokalizátoru Rez.

Všechny binární knihovny musely být vygenerovány zvlášť pro různé architektury, což ukazuje tabulka 3.2.

Platforma	Architektura	Výsledný binární soubor (library/plugin/bundle)
GNU/Linux	32-bit	npCoNetServ_x86-gcc4.so
GNU/Linux	64-bit	npCoNetServ_x86_64-gcc4.so
Microsoft Windows	32-bit	npCoNetServ_x86-msvc.dll
Microsoft Windows	64-bit	npCoNetServ_x86_64-msvc.dll
Darwin/Mac OS X	32-bit	npCoNetServ_x86-gcc4.dylib

Tabulka 3.2: Výsledné binární soubory na různých platformách a architekturách

3.5 Pseudo-objektový návrh uvnitř pluginu



Obrázek 7: Ilustrace části pseudo-objektového návrhu uvnitř pluginu

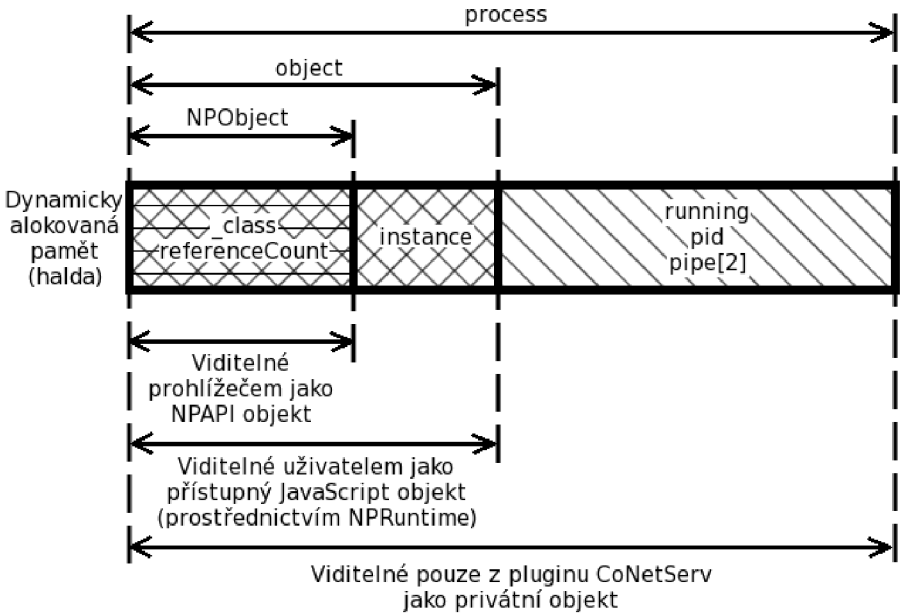
Pseudo-objekty uvnitř pluginu vychází z návrhu architektury NPAPI, jak je vidět na obrázku 7. Z hlediska pluginu se ovšem nejedná o objekty v pravém slova smyslu – plugin je totiž implementovaný v jazyku C – a zmiňované „pseudo-objekty“ jsou de facto jen struktury, které simulují dědičnost objektů překrýváním dynamicky alokované paměti za pomoci přetypování.

3.5.1 Privátní a veřejné dynamicky alokované objekty

Objekty se dají rozdělit do dvou skupin:

- **privátní objekty**,
  které jsou viditelné pouze uvnitř pluginu jako interní data
- a **veřejné objekty**,
  které jsou částečně dostupné prohlížeči jako NPObject objekty a uživateli jako objekty v JavaScriptu (prostřednictvím rozšíření NPRuntime)

Druhá skupina je názorně zobrazena na obrázku 8 na příkladu objektu `process`.



Obrázek 8: Názorná ukázka překrývání dynamicky alokované paměti veřejného pseudo-objektu `process` *a jeho viditelnost v různých úrovních projektu*

3.5.2 Třídy objektů

Plugin CoNetServ je navrhnut modulárně - každý modul představuje oddělenou sadu implementace „abstraktního“ objektu `modul`, `object` nebo `process` (viz obrázek 7). Jednotlivé moduly můžou být rozděleny do dalších sub-modulů, přičemž se liší především třídou `_class` objektu NPObject a privátními daty k dané třídě přidruženými (viz obrázek 8).

Obsah třídy `_class` určuje způsob chování objektu, který je volán z rozhraní JavaScript – NPRuntime. Jednotlivé třídy obsahují ukazatele na funkce – přístupové body modulu, které volá obslužná rutina prohlížeče.

3.5.3 Prototypy funkcí přístupových bodů:

- Prototyp funkce pro zjištění existence metody objektu:

```
static bool hasMethod(NPObject *obj, NPIdentifier identifier);
```

- Prototyp funkce pro zavolání metody objektu:

```
static bool invokeMethod(NPObject *obj, NPIdentifier
identifier, const NPVariant *args, uint32_t argc, NPVariant
*result);
```

- Prototyp funkce pro zavolání výchozí metody objektu:

```
static bool invokeDefault(NPObject *obj, const NPVariant
*args, const uint32_t argCount, NPVariant *result);
```

- Prototyp funkce pro zjištění existence vlastnosti objektu:

```
static bool hasProperty(NPObject *obj, NPIdentifier
identifier);
```

- Prototyp funkce pro získání hodnoty vlastnosti objektu:

```
static bool getProperty(NPObject *obj, NPIdentifier
identifier, NPVariant *result);
```

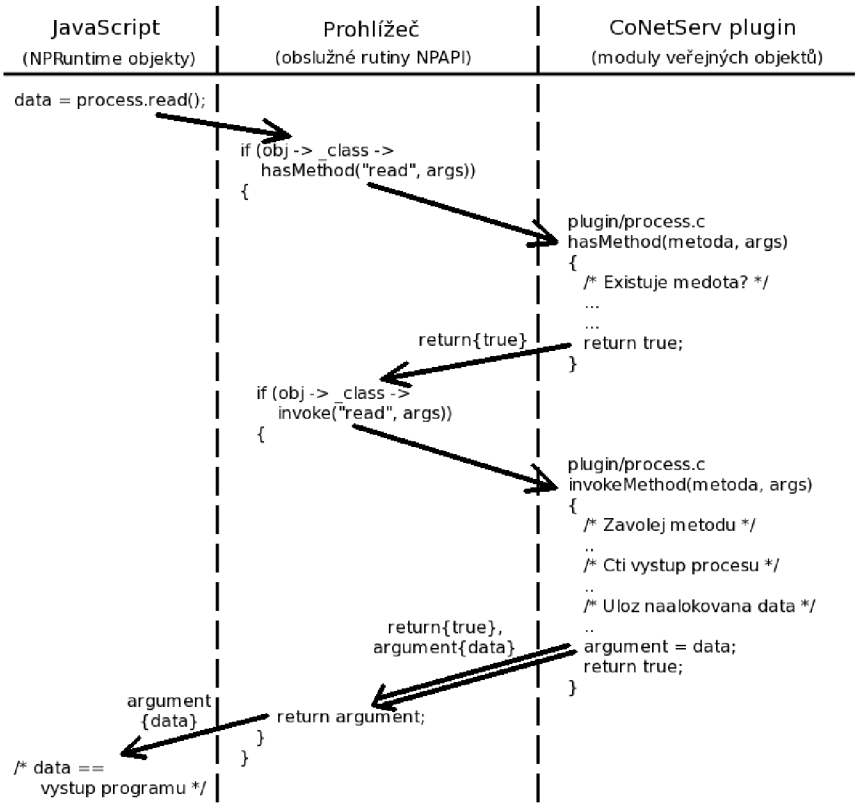
- Prototyp funkce pro nastavení hodnoty vlastnosti objektu:

```
static bool setProperty(NPObject *obj, NPIdentifier
identifier, NPVariant *result);
```

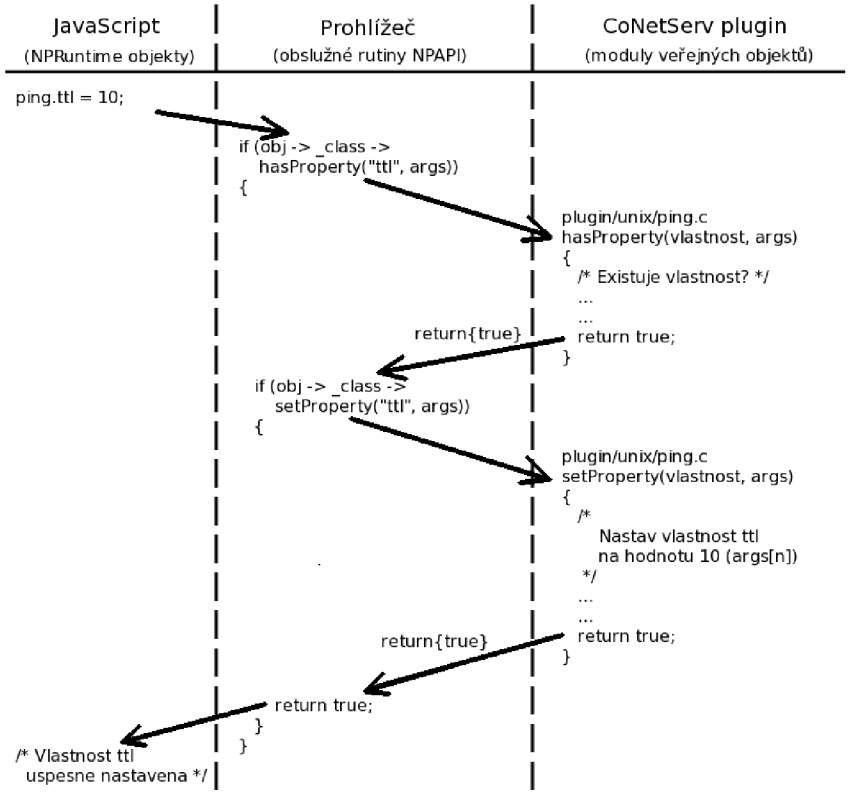
3.5.3.1 Příklady volání jednotlivých metod či vlastností objektu z JavaScriptu

Přistupuje-li JavaScript například k metodě `read()` objektu třídy `process`, zavolá prohlížeč nejdřív funkci `hasMethod()` konkrétní třídy dle ukazatele `_class` daného objektu – v tomto případě tedy statickou funkci `hasMethod()` ze souboru `plugin/process.c`. Jestliže je návratová hodnota volané funkce `true`, zavolá dále funkci `invokeMethod()`, jejímiž výstupními parametry může cílová funkce vrátit pomocí dynamicky alokovaného objektu neomezené množství dat. Tato data ale musí striktně splňovat normu pro pozdější destrukci objektu – buď být speciálním typem, s kterým si poradí sám prohlížeč a nebo musí nastavovat vlastní destruktor, aby nedocházelo k únikům paměti. Postup je znázorněn na obrázku 9.

Výše popsaný postup je analogický i pro přístup k jednotlivým vlastnostem objektu – s výjimkou existence speciální funkce jak pro získání, tak i nastavení hodnoty. Příklad takového volání je znázorněn na obrázku 10



Obrázek 9: Zjednodušená ilustrace komunikace vrstev při volání metody



Obrázek 10: Zjednodušená ilustrace komunikace vrstev programu při nastavování vlastnosti

3.6 Implementované lokální služby

3.6.1 Spouštění a manipulace s procesy - Unix

Absolutní cesty k externí programům jsou na unixových systémech nejprve vyhledány v průběhu inicializace pluginu pomocí funkce `find_program_path()` - důvodem jsou nekompatibilní cesty mezi různými distribucemi a operačními systémy vycházející z Unixu (např. program `ping` může být umístěn v adresáři `/usr/`, `/usr/bin/`, `/sbin/` nebo `/usr/sbin/`). Tato funkce využívá výstupu programu `which`, který se vždy nachází v adresáři `/usr/bin/`, a musí pro něj vždy vytvořit externí proces, ze kterého načte výsledný řetězec.

Toto krkolomné řešení muselo být použito z důvodu nedostupnosti funkce `execvpe()` na mnoha unixových systémech. Funkce sice umí pracovat s relativní cestou programu a zároveň nastavovat proměnné unixového prostředí, ale byla implementována až ve verzi knihovny glibc 2.11.

Jednotlivé procesy externích programů jsou poté spouštěny funkcí `execv()`, která je volána uvnitř nově vytvořeného podprocesu pluginu pomocí funkce `vfork()`. Komunikaci mezi pluginem a externím procesem zajišťují anonymní roury, vytvořené pomocí funkce `pipe()`, které pracují jako fronta FIFO. Ukončení jednotlivých procesů či získání jejich stavu obstarávají funkce `kill()`, resp. `waitpid()`.

3.6.2 Spouštění a manipulace s procesy - Microsoft Windows

Manipulace s externími procesy na platformě Windows je velmi podobná té unixové. Rozdíl je pouze v implementačních detailech, inicializačních krocích a v použití rozdílných datových typů. Narozdíl od unixových zdrojových kódu byly použity např. následující funkce: `CreateProcess()`, `CreatePipe()`, `TerminateProcess()` a `GetExitCodeProcess()`. Jednotlivé externí programy jsou spouštěny pomocí relativní cesty v programu `cmd.exe`.

Narozdíl od unixových systémů, které typicky používají kódování UTF-8, jsem na platformě Windows musel řešit problémy s národním kódováním. Například Firefox běžící pod Windows XP používá interně kódování UTF-8, ale externí program mu prostřednictvím pluginu předává znaky v národním kódování – např. Windows-1250. Řešení nebylo pouze v použití speciálního parametru `'/u'` programu `cmd.exe`, který by měl automaticky přepnout výstup programu do kódování Unicode, ale bylo nutné konvertovat všechny přečtené znaky pomocí funkcí `MultiByteToWideChar()` a `WideCharToMultiByte()` z dané kódovací stránky do UTF-8, za pomoci dočasných bufferů.

3.6.3 Použité programy a přepínače, rozdíly mezi platformami

Platforma	Verze IP	PING - Spuštěný příkaz a jeho parametry
GNU/Linux	IPv4	`%s{PATH}/ping -n -l3 [-c %d{PACKET_COUNT}] [-s %d{PACKET_SIZE}] [-W %d{TIMEOUT}] [-t %d{TTL}] %s{HOST}`
GNU/Linux	IPv6	`%s{PATH}/ping6 -n -l3 [-c %d{PACKET_COUNT}] [-s %d{PACKET_SIZE}] [-W %d{TIMEOUT}] [-t %d{TTL}] %s{HOST}`
Darwin/Mac OS X	IPv4	`%s{PATH}/ping -n [-c %d{PACKET_COUNT}] [-s %d{PACKET_SIZE}] [-m %d{TTL}] %s{HOST}`
Darwin/Mac OS X	IPv6	`%s{PATH}/ping6 -n [-c %d{PACKET_COUNT}] [-s %d{PACKET_SIZE}] [-m %d{TTL}] %s{HOST}`
Microsoft Windows	IPv4	`ping.exe [-n %d{PACKET_COUNT} | -t] [-l %d{PACKET_SIZE}] [-w %d{TIMEOUT}] [-i %d{TTL}] %s{HOST}`
Microsoft Windows	IPv6	`ping.exe -6 [-n %d{PACKET_COUNT} | -t] [-l %d{PACKET_SIZE}] [-w %d{TIMEOUT}] [-i %d{TTL}] %s{HOST}`

*Tabulka 3.3: Program **ping** ve verzi protokolu IPv4 a IPv6 a implementované parametry na operačních systémech GNU/Linux, Darwin/Mac OS X a Microsoft Windows*

Platforma	WHOIS - Spuštěný příkaz a jeho parametry
Unix-like	`%s{PATH}/whois %s{HOST}`

*Tabulka 3.4: Program **whois** a implementované parametry na unixových systémech*

Platforma	DIG - Spuštěný příkaz a jeho parametry
Unix-like	`%s{PATH}/dig %s{PATH}`

*Tabulka 3.5: Program **dig** a implementované parametry na unixových systémech*

Platforma	Verze IP	TRACEROUTE/TRACERT - Spuštěný příkaz a jeho parametry
Unix-like	IPv4	`%s{PATH}/traceroute [-m %d{MAX_HOPS}] [-w %d{WAIT_TIME}] [-n] %s{HOST}`
Unix-like	IPv6	`%s{PATH}/traceroute6 [-m %d{MAX_HOPS}] [-w %d{WAIT_TIME}] [-n] %s{HOST}`
Microsoft Windows	IPv4	`tracert.exe [-h %d{MAX_HOPS}] [-w %d{WAIT_TIME}] [-d] %s{HOST}`
Microsoft Windows	IPv6	`tracert.exe -6 [-h %d{MAX_HOPS}] [-w %d{WAIT_TIME}] [-d] %s{HOST}`

*Tabulka 3.6: Program **tracert/traceroute** ve verzi protokolu IPv4 a IPv6 a implementované parametry na unixových systémech a na operačním systému Microsoft Windows*

Platforma	NSLOOKUP - Spuštěný příkaz a jeho parametry
Unix-like	`%s{PATH}/nslookup -q=AAAA %s{HOST}` `%s{PATH}/nslookup -q=A %s{HOST}`
Microsoft Windows	`nslookup.exe -q=AAAA %s{HOST}` `nslookup.exe -q=A %s{HOST}`

*Tabulka 3.7: Program **nslookup** a implementované parametry na unixových systémech a na operačním systému Microsoft Windows*

Platforma	ROUTE - Spuštěný příkaz a jeho parametry
Unix-like	`%s{PATH}/route %s{HOST}`

*Tabulka 3.8: Program **route** a implementované parametry na unixových systémech*

Platforma	IFCONFIG/IPCONFIG - Spuštěný příkaz a jeho parametry
Unix-like	`%s{PATH}/ifconfig`
Microsoft Windows	`ipconfig.exe /All`

*Tabulka 3.9: Program **ifconfig/ipconfig** a parametry implementované na unixových systémech a na operačním systému Microsoft Windows*

Platforma	Verze IP	Spuštěný příkaz a jeho parametry
Unix-like	IPv4	`%s{PATH}/nmap -sT %s{HOST}` `%s{PATH}/nmap %s{HOST}`
Unix-like	IPv6	`%s{PATH}/nmap -6 -sT %s{HOST}` `%s{PATH}/nmap -6 %s{HOST}`
Microsoft Windows	IPv4	`nmap.exe -sT %s{HOST}` `nmap.exe %s{HOST}`
Microsoft Windows	IPv6	`nmap.exe -6 -sT %s{HOST}` `nmap.exe -6 %s{HOST}`

*Tabulka 3.10: Program **nmap** ve verzi protokolu IPv4 a IPv6 a implementované parametry na unixových systémech a na operačním systému Microsoft Windows*

3.7 Debugování pluginu

Binární knihovny aplikace CoNetServ nejsou samy o sobě spustitelnými soubory a lze je tedy debugovat pouze v rámci spuštěného procesu webového prohlížeče, který je načítá v průběhu své existence. K tomu lze efektivně využít například programu gdb, který se dokáže „připojit“ na jakýkoliv běžící proces operačního systému.

Tento postup byl ovšem nepoužitelný v případě proprietárního prohlížeče Opera, jehož zdrojové kódy nejsou veřejné a chybí tudíž i jeho tzv. „debug-info“ soubory (soubory obsahující ladící informace).

Vytvořil jsem proto speciální testovací aplikaci (kterou lze vytvořit příkazem „make debug“ v překladovém systému – viz kapitola 3.2), kterou lze otestovat základní funkcionalitu binární aplikace CoNetServ bez implementačně závislých rozdílů jednotlivých prohlížečů. Výše popsaný problém s prohlížečem Opera jsem tímto postupem alespoň částečně vyřešil, ovšem ne zcela – platí, že v rámci proprietárního prohlížeče je vždy nutné experimentovat metodou „pokus-omyl“.

Exportované symboly výsledných knihoven jsem ověřoval programy ldd, objdump a DLL Export Viewer.

3.8 Manuální instalace pluginu

Instalace pluginu, který má podobu sdílené binární knihovny, je možná dvěma způsoby – manuální nebo automatizovanou cestou.

Manuální instalace spočívá v nakopírování konkrétního binárního souboru do předem stanoveného adresáře – viz tabulka 3.11, ve kterém jej cílový webový prohlížeč na dané platformě vyhledává.

Prohlížeč	Operační systém	Adresář
Firefox, Chrome, Opera	Microsoft Windows	C:\Program Files\Mozilla Firefox\plugins
Firefox, Chrome, Opera	GNU/Linux	~/.mozilla/plugins/, /usr/lib/mozilla/plugins/
Firefox, Chrome, Opera, Safari	Darwin/Mac OS X	/Applications/Firefox.app/Contents/MacOS/plugins/
Chrome	Microsoft Windows	C:\Program Files\Google Chrome\plugins
Opera	Microsoft Windows	C:\Program Files\Opera\plugins
Chrome	GNU/Linux	/opt/google/chrome/
Opera	GNU/Linux	/usr/lib/opera/plugins/
Opera	Darwin/Mac OS X	/Applications/Opera.app/Contents/MacOS/plugins/
Safari	Microsoft Windows	C:\Program Files\Safari\plugins
Safari	Darwin/Mac OS X	/Applications/Safari.app/Contents/MacOS/plugins/

Tabulka 3.11: Adresáře, ve kterých jednotlivé prohlížeče hledají binární knihovny pluginů

Automatizovaná instalace pluginu je možná buď v rámci balíčku rozšíření (viz kapitola 5) nebo pomocí systémového správce, což je typické pro balíčky linuxových distribucí, nebo posledním způsobem – instalací proprietárním instalátorem (typicky na platformě Windows).

4 CoNetServ JavaScript API

4.1 Komunikace s pluginem prostřednictvím NPRuntime

Webový prohlížeč nahraje a inicializuje plugin CoNetServ typicky ihned po svém spuštění, kdy prohledává příslušné adresáře s binárními knihovnami. Jedinou výjimkou je toho času prohlížeč Google Chrome, který nahrání nového pluginu zvládá i během svého běhu.

Inicializaci konkrétních instancí pluginu ovšem prohlížeč spustí až po detekci párového elementu `object` (případně `embed`) v (X)HTML kódu daného dokumentu, jehož atributem `type` je předem určený řetězec `“application/x-conetserv“`:

```
<object id="conetserv" type="application/x-
conetserv"></object>
```

Po úspěšné inicializaci instancí pluginu už je možné k jednotlivým elementům přistupovat pomocí následujícího JavaScriptu – a získat tak „přístupový NPRuntime objekt“ stejně zvaného rozšíření architektury NPAPI:

```
var plugin = document.getElementById("conetserv");
```

Vlastnosti a metody získaného objektu lze ovládat dynamicky uvnitř pluginu „za běhu programu“. Zjistit, zda-li byl plugin správně načten lze proto jednoduchým způsobem – implementoval jsem totiž v kořenovém objektu vlastnost `version`, která musí vždy vracet řetězec představující verzi pluginu. Kontrolovat, zda-li je objekt správně načtený, lze nejlépe dvojí negativní podmínkou:

```
if (!plugin || !plugin.version) {
  // Plugin nebyl spravne nacten
}
var version = plugin.version; // napr. retezec “2.1.0”
```

4.2 Objektový návrh

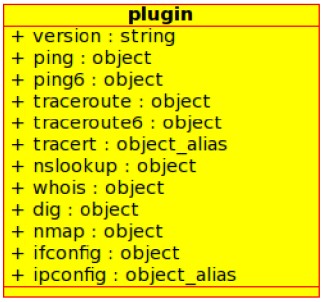
Objektový návrh rozhraní pro JavaScript byl vytvořen až ve verzi projektu CoNetServ v2.0, v dřívějších verzích pluginu nebylo možné vytvářet objekty jednotlivých služeb a procesů, ale bylo nutné přistupovat k daným objektům obskurními konstrukcemi, při kterých často hrozil únik paměti, tzv. „memory leak“.

Nový návrh nepřinesl pouze kvalitní objektový návrh a smysluplnou hierarchii jednotlivých objektů, ale především rychlost, bezpečnost a naprostou kontrolu nad běžícími podprocesy pluginu.

4.2.1 Kořenový objekt pluginu

V kapitole 4.1 bylo popsáno, jak získat prostřednictvím JavaScriptu a DOM kořenový objekt projektu CoNetServ – z postupu je zřejmé, že získaná proměnná může nést jakýkoliv název. Pro názornost je ale v obrázku 11 uveden název objektu `plugin`.

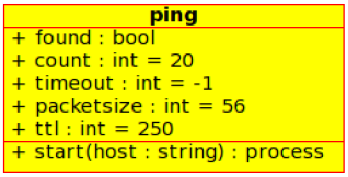
Přistupovat k jednotlivým lokálním službám lze pouze a právě přes tento objekt, protože jednotlivé služby jsou vždy pod-objektem kořenového objektu pluginu.



Obrázek 11: CoNetServ JavaScript API: Kořenový objekt „plugin“

4.2.2 Objekt ping

Příkladem objektu lokální služby je například objekt `ping`. Každý objekt lokální služby má alespoň metodu `start(string host)`, která spouští příslušný externí proces a vrací jej jako návratovou hodnotu, a také vlastnost `found`, která udává, zda-li byla lokální služba na cílovém systému nalezena.



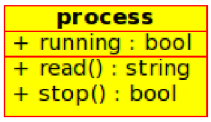
Obrázek 12: CoNetServ API: Objekt ping

Následující ukázka kódu ukazuje testování objektu ping a nastavování jeho parametrů:

```
if (!plugin.ping) {
  // Not implemented for platform
}
if (!plugin.ping.found) {
  // Not installed service in current system
}
```

```
plugin.ping.count = 10; // 10 packets
plugin.ping.ttl = 20; // max 20 TTL / hops
plugin.timeout = 3; // 3 second to wait for response
plugin.packetsize = 56; // set size of packet to 56 bytes
```

4.2.3 Objekt process

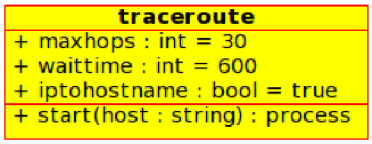


Obrázek 13: CoNetServ API: Objekt process

Objekt process je vždy navrácen jako návratová hodnota po spuštění konkrétní lokální služby. Reprezentuje externí proces, který skutečně existuje a běží na operačním systému uživatele. Ze získaného objektu lze poté číst data, zjistit zda-li je proces stále aktivní a lze jej také ukončit. Následující ukázka kódu ukazuje všechny vyjmenované operace:

```
process = plugin.ping.start("nic.cz");
if (!process.running) {
  // Proces nebezi nebo byl ukoncen
}
var string = process.read(); // Cti data z procesu
if (string) {
  // Vytiskni data
}
process.stop(); // Ukonci proces
```

4.2.4 Objekty zbylých lokálních služeb



Obrázek 14: CoNetServ API: Objekt traceroute (tracert)

Dalšími objekty lokálních služeb jsou: ping6, traceroute, traceroute6, tracert, tracert6, nslookup, whois, dig, nmap, ifconfig, ipconfig. Obrázek 14 zobrazuje jako příklad objekt traceroute – zbylé objekty jsou svým obsahem víceméně podobné.

4.3 Front-end a implementované externí služby

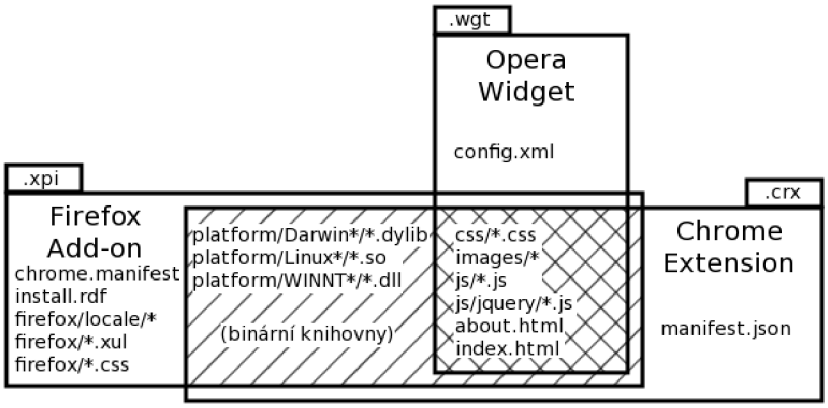
Front-end (neboli viditelná část) projektu CoNetServ byla naprogramována pomocí kombinace technologií XHTML, CSS, JavaScript a knihovny jQuery. Tato práce se touto částí projektu zabývá pouze okrajově – a proto jen vyjmenuji dílčí implementované externí služby a zdroje dat v tabulce 4.1 a také uvedu obrázek běžící lokální služby ping, která je zobrazována do grafu pomocí knihovny jQuery plots. Další screenshoty z běhu aplikace, zobrazení interaktivní mapy a nastavení jednotlivých služeb, jsou dostupné v příloze B.

Za zmínku stojí ještě implementace Punycode – podpory adres internetových domén v tabulce znaků Unicode, která byla přesunuta z pluginu (implementace v jazyku C) do front-endu aplikace (implementace v JavaScriptu) – a to především z důvodu nadměrné zátěže procesu pluginu a zbytečné velikosti výsledných binárních knihoven.

Zdroj parsovaných dat	Externí služba
Google.com – JSAPI	Pozice na interaktivní mapě
MaxMind GeoIP	Pozice na interaktivní mapě
IP Info DB	Pozice na interaktivní mapě
Free IP geolocation webservice by Alexandre Fiori	Pozice na interaktivní mapě
WIPmania.com - WorldIP API	Pozice na interaktivní mapě, externí adresa IPv4
CGI script by Chris F.A. Johnson	Externí adresa IPv4
Check IP by DynDNS.com	Externí adresa IPv4
MojeIP.cz	Externí adresa IPv4, externí adresa IPv6, hostname externí adresy
IPinfo Security Portal	Externí adresa IPv4, hostname externí adresy
RADb, Merit Network Inc.	Externí adresa IPv4, route/adresa sítě, jméno poskytovatele internetového připojení
2 Privacy.com	Externí adresa IPv4, hostname externí adresy
ATMAN Looking glass	Ping IPv4, Ping IPv6
CERN Looking glass	Ping IPv4, Ping IPv6
ILAN Looking glass	Ping IPv4
Telecom Italia Sparkle's Looking Glass	Ping IPv4

Tabulka 4.1: Projekt CoNetServ: Implementované externí služby a zdroje dat

5 Implementace balíčků rozšíření a jejich distribuce



Obrázek 15: Projekt CoNetServ: Společné části (průnik souborů) jednotlivých balíčků rozšíření Firefox Add-on, Chrome Extension a Opera Widget, před jejich archivací

Tvorba jednotlivých balíčků rozšíření je vždy závislá na konkrétní implementaci daného prohlížeče – každý prohlížeč podporuje různé archivy balíčků, popisuje obsah balíčků jiným souborem a odlišnou syntaxí a nabízí též unikátní rozhraní API přístupu ke zdrojům prohlížeče.

Na obrázku 15 jsou názorně zobrazeny společné části a rozdíly mezi jednotlivými balíčky rozšíření aplikace CoNetServ. Všechny zmiňované balíčky mají společnou uživatelsky viditelnou část projektu – od HTML stránek, přes uživatelský JavaScript až po kaskádové styly a soubory obrázků. Firefox Add-on a Chrome Extension mají ještě jednu společnou část – a to soubory binárních knihoven, které zabalují přímo do výsledného archivu. Tato možnost u Opera Widgetu z bezpečnostních důvodů chybí.

Následující podkapitoly popisují implementaci a specifické detaily jednotlivých balíčků rozšíření – nejdříve Firefox Add-on, dále Chrome Extension a nakonec Opera Widget.

5.1 Mozilla Add-ons

Balíček rozšíření do aplikací Mozilla je tzv. Cross-Platform Installer Module, zkráceně XPI. Soubor s příponou .xpi, který jej reprezentuje, není de facto nic jiného, než klasický ZIP archiv s pozměněnou koncovkou, který vždy obsahuje soubor `install.rdf` (dříve `install.js`).

Aplikace společnosti Mozilla Corporation využívají jednotného instalátoru technologie XPInstall [27], který slouží k instalaci nových rozšíření, témat, pluginů nebo jejich kombinace. Mozilla zpřístupňuje jednotlivým rozšířením své API prostřednictvím technologie XUL (XML User Interface Language), která byla v projektu CoNetServ též použita – konkrétně pro vytvoření ikon a tlačítek aplikace v nástrojové liště prohlížeče a k nastavené vlastností nově vytvořeného okna po stisku tlačítka.

Kromě souboru `install.rdf`, který obsahuje základní informace o rozšíření (typ, název, verze, podporované verze aplikací atd.), obsahuje archiv též soubor `chrome.manifest`, který udává rozložení struktury souborů uvnitř archivu (soubory XUL, kaskádových stylů a překladů aplikace). Velikou výhodou rozšíření XPI je možnost ladění aplikace v reálném čase, za pomocí debuggeru FireBug.

Oficiální rozšíření Mozilla jsou hostovány na internetové službě Mozilla ADD-ONS, kde bylo rozšíření CoNetServ publikováno pod adresou https://addons.mozilla.org/firefox/addon/181909, ovšem pouze jako tzv. „untrusted“ rozšíření, které díky přítomnosti NPAPI pluginu nezískalo status ověřené aplikace.

5.2 Google Chrome extension

Balíček rozšíření do prohlížeče Google Chrome je archiv postavený na formátu ZIP, který má ovšem příponu .crx, obsahuje speciální hlavičky s podpisem a RSA klíči autora [28] a obsahuje alespoň jeden soubor s názvem `manifest.json`.

Tento soubor, který musí být zapsán ve formátu JSON (JavaScript Object Notation), obsahuje nejen název projektu, jeho verzi a stručný popis, ale také výčet důležitých souborů rozšíření, které jsou načítány při běhu aplikace, výčet požadovaných oprávnění a minimální verzi podporovaného prohlížeče.

Prohlížeč Google Chrome je díky své multi-procesové architektuře schopen načítat jednotlivé rozšíření za běhu programu, a to i když balíček obsahuje NPAPI plugin. Ten dokáže inicializovat jako dynamickou knihovnu v průběhu své činnosti – bez nutnosti restartu (na rozdíl od prohlížečů Firefox nebo Opera). Jednotlivá rozšíření lze velmi snadno ladit díky zabudovanému debuggeru v prohlížeči Chrome, který nese název Chrome Developer Tools.

Oficiální rozšíření aplikace Google Chrome jsou hostovány službou Chrome Web Store (dříve Google Chrome Extensions), do které vývojář nahrává archiv ZIP a portál jej převede do podoby CRX a opatří jej RSA klíči patřící společnosti Google. Aplikace CoNetServ byla publikována pod adresou https://chrome.google.com/webstore/detail/mmkpilpdijdbifpgkpdndpjlkpjkihee.

Zajímavostí je, že Google přenáší odpovědnost za případné škody způsobené pluginem NPAPI na jednotlivé vývojáře rozšíření. Před vydáním první verze aplikace CoNetServ jsem musel podepsat prohlášení o odpovědnosti v dokumentu „Google Chrome Gallery Developer Agreement“.

5.3 Opera widget

Opera widget byl toho času posledním implementovaným balíčkem rozšíření Complex Network Services. Formát archivu je opět ZIP archiv, ovšem s příponou .wgt a s alespoň jedním souborem nazvaném `config.xml`.

Tento soubor, s daty ve formátu XML (Extensible Markup Language), nese opět základní informace o aplikaci, jako je název „widgetu“, datum poslední změny (která ve formátu *YYYY-MM-DD* funguje principiálně jako verze programu) a dále obsahuje cesty k jednotlivým ikonám aplikace a výšku a šířku okna „widgetu“. Prohlížeč Opera též v archivu očekává soubor `index.html`, který načítá jako výchozí stránku – obsah widgetu.

Opera jako jediná, z výše jmenovaných prohlížečů, žádným způsobem nepodporuje zabalení binárních knihoven do výsledného archivu. Plugin proto musí být instalován manuální cestou, jak je popsáno v kapitole 3.8 a tabulce 3.11.

6 Zveřejnění projektu a statistiky

Projekt CoNetServ byl oficiálně vydán ve verzi 1.0 jako balíček rozšíření daného prohlížeče v oficiálních repozitářích Mozilla Add-ons a Google Chrome Extensions (později Chrome Web Store) na podzim roku 2010. Zpětná vazba od uživatelů, kteří začali jednotlivé balíčky rozšíření instalovat a používat, byla velmi přínosná a inspirativní.

6.1 Testování a zpětná vazba

Aplikace byla díky velkému množství testerů odzkoušena na mnoha platformách – od několika distribucí systému GNU/Linux, přes platformu Darwin/Mac OS X, až po různé verze operačního systému Microsoft Windows, konkrétně Windows XP, Windows Vista a Windows Seven. Téměř každá z vyjmenovaných platforem byla otestována ve své 32-bitové i 64-bitové verzi.

Díky chybovým hláškám, které jednotliví uživatelé zasílali, jsem byl schopen opravit a odladit předem nepředvídané chyby – např. chybnou inicializaci NPAPI pluginu na platformě Darwin/Mac OS X, problémy s verzí dynamicky linkované knihovny glibc na distribuci Debian Linux či chybnou závislost na ladící knihovně MSVCR90.dll (v důsledku automatického překladu projektu v Microsoft Visual Studiu s novějším WinAPI na operačním systému Microsoft Windows XP), která způsobovala pád celého prohlížeče.

6.2 Podporované platformy a prohlížeče

Následující tabulky 6.1 a 6.2 uvádí výčet podporovaných platforem a webových prohlížečů, které jsou podporovány aplikací CoNetServ verze 2.1.0:

Operační systém	Status	Podporované architektury
Android	Ve fázi vývoje	
BeOS	Nepodporováno	
FreeBSD	Ve fázi testování	
GNU/Linux	Podporováno	x86, x86_64
Darwin/Mac OS X	Podporováno	x86
Microsoft Windows	Podporováno	x86, x86_64

Tabulka 6.1: CoNetServ v2.1.0 – podporované platformy

Prohlížeč, verze	Status	Způsob distribuce / Poznámka
Camino	Není podporováno	Prohlížeč nepodporuje NPAPI pluginy
Epiphany	Není podporováno	Prohlížeč nepodporuje NPAPI pluginy
Firefox Fennec 1.1+	Ve fázi testování	Nedostatek testovacích zařízení s Firefox Fennec
Firefox 3.0+	Podporováno	Add-on .xpi – balíček se zabudovanou knihovnou
Flock 2.0 - 2.9	Podporováno	Add-on .xpi – balíček se zabudovanou knihovnou
Flock 3.0+	Podporováno	Extension .crx – balíček se zabudovanou knihovnou
Google Chrome 4.0+	Podporováno	Extension .crx – balíček se zabudovanou knihovnou
Iceweasel 3.0+	Podporováno	Add-on .xpi – balíček se zabudovanou knihovnou
Internet Explorer	Není podporováno	Prohlížeč nepodporuje NPAPI pluginy (od verze 5.5)
Konqueror	Není podporováno	Prohlížeč nepodporuje NPAPI pluginy
Maxthon	Není podporováno	Prohlížeč nepodporuje NPAPI pluginy
Opera 10.5+	Podporováno	Widget a manuálně nainstalovaná knihovna
Opera Mini	Není podporováno	Prohlížeč nepodporuje NPAPI pluginy
Safari 5.0+	Ve fázi testování	Nutno vytvořit balíček rozšíření pro Safari
SeaMonkey 2.0+	Ve fázi testování	Add-on .xpi – balíček se zabudovanou knihovnou

Tabulka 6.2: CoNetServ v2.1.0 – podporované webové prohlížeče

6.3 Otevřenost zdrojových kódů a komunita vývojářů open-source

Zdrojové kódy byly po celou dobu vývoje projektu (a stále jsou) zveřejňovány na veřejném *git* repozitáři portálu github.com jako open-source projekt pod licencí GPLv2 (GNU General Public License v2.0).

Díky otevřenosti zdrojových kódů projevilo několik externích vývojářů zájem o dílčí části projektu a CoNetServ též sloužil jako inspirace pro nově vznikající projekty – například pro aplikaci Network and Internet Tools [29]. Několik zahraničních studentů a programátorů mě žádalo o radu při tvorbě NPAPI pluginu a dva vývojáři dokonce projevili zájem podílet se na testování či vývoji aplikace CoNetServ samotné (první z nich opravdu několik týdnů testoval rozšíření pro webový prohlížeč Google Chrome, a to s velkým nadšením).

Z výše uvedeného je zřejmé, že tvorba projektu s otevřeným zdrojovým kódem, který je pravidelně aktualizován a zveřejňován na síti Internet, nese velké výhody. Komunita open-source vývojářů je dnes opravdu rozsáhlá a spolupráce i s mnohdy anonymními vývojáři pro mě byla nečekaně přínosná.

6.4 Verzování projektu

Projekt CoNetserv je verzován trojčíslím dle vzoru *major.minor[.build[.revision]]*, který je mezi open-source aplikacemi velice rozšířený. První číslo *major* udává hlavní verzi projektu, která se mění pouze s výraznou změnou API – tedy vždy když se jedná o výraznou a zpětně nekompatibilní změnu. Ta potkala projekt CoNetServ zatím pouze jednou – při implementaci nového objektového návrhu API (viz kapitola 4.2). Druhé číslo *minor* používá projekt CoNetServ k odlišení jednotlivých verzí, které přidávají nebo výrazně mění funkcionalitu aplikace. Poslední číslo, které CoNetServ ve svých verzích odlišuje, je tzv. „bugfix“ číslo. Toto číslo je jakousi směsí výše uvedeného *build* a *revision* čísla a značí především menší opravy aplikace či nová vydání, která nepřinášejí výrazné změny v chování programu.

Projekt CoNetServ byl vyvíjen od své testovací verze 0.1.0 až k prvnímu veřejnému vydání pod označením CoNetServ v1.0.0. Celkem byly vydány desítky aktualizací aplikace, přičemž poslední verze toho času byla vydána 14.3.2011 pod označením CoNetServ v2.1.0.

6.5 Statistiky uživatelů, počtu stažení a hodnocení aplikace

Tabulka 6.3 znázorňuje počet uživatelů oficiálně vydaného rozšíření aplikace CoNetServ v2.1.0, dle statistik uvedeného zdroje. Celkový počet stažení či používanost verzí jednotlivých neoficiálních balíčků rozšíření, které jsou dostupné např. z projektové stránky na portálu github.com, nejsou bohužel žádným způsobem dohledatelné a ověřitelné.

Typ rozšíření	**Zdroj statistiky**	**Počet uživatelů denně (týdenní průměr)**	**Počet stažení za poslední týden**	**Počet komentářů**	**Hodnocení uživateli (počet hlasů)**	**Počet stažení celkem**
.xpi add-on	Mozilla Firefox ADD-ONS	427	203	2	7/10 (2)	2846
.crx extension	Google Chrome Web Store	621	147	10	5/5 (16)	1707

Tabulka 6.3: Statistiky uživatelů, počtu stažení a hodnocení jednotlivých rozšíření aplikace CoNetServ, dle oficiálních zdrojů ke dni 3.4.2011

7 Závěr

Cílem této technické zprávy bylo obecně popsat technologie pro tvorbu rozšíření webových prohlížečů v nativním kódu a poté zaznamenat jeden konkrétní příklad vývoje takového rozšíření – pomocí předem vybrané technologie. Praktickou částí této práce byl projekt CoNetServ (plugin vytvořený pomocí technologie architektury NPAPI a též balíčky rozšíření do současných webových prohlížečů), který přináší funkcionalitu síťových nástrojů běžným uživatelům stiskem jediného tlačítka na liště prohlížeče.

7.1 Native Client

Technologie Native Client, kterou jsem testoval a věnoval jí kapitolu 2.4, má velký potenciál rozšířit se během několika let jako běžný prostředek pro urychlení webových aplikací. Dle mého názoru se stane tahounem a světovou jedničkou na poli on-line her, a to především díky zabudované podpoře rozšířených instrukcí procesoru, možnosti přímého přístupu ke grafické kartě a jednotnému rozhraní pro tvorbu zvuků a zobrazování 2D a 3D grafiky. Jednoduchá přenositelnost a multiplatformnost, ve které Native Client vyniká, bude s nástupem *cloudových* operačních systémů naprosto nezbytná.

7.2 Projekt CoNetServ

Projekt CoNetServ naplnil svá předsevzetí a funguje jako jednoduchá síťová aplikace v prohlížečích Mozilla Firefox a Google Chrome, kterou toho času využívají stovky až tisíce uživatelů po celém světě. Podařilo se mi vytvořit aplikaci, která spojuje několik technologií (od binárního kódu až po grafický front-end) do jednoho logického celku, a která funguje spolehlivě na mnoha platformách a architekturách. Projekt je postaven na vlastním objektově orientovaném modelu a API rozhraní a může sloužit jako základ mnoha dalším open-source projektům, které můžou využít jeho zdrojových kódů pod licencí GPLv2+.

Projekt Complex Network Services též vyhrál dvě ocenění v podobě výhry v soutěži VIP 2 a VIP 3 pořádané laboratořemi CZ.NIC, z.s.p.o., což dokazuje jeho smysluplnost a použitelnost v reálném světě informačních technologií.

Doufám a pevně věřím, že bude projekt nadále vyvíjen a vylepšován a bude ještě dlouho sloužit velkému počtu uživatelů jako rychlá alternativa k síťovým nástrojům, které dokáže efektivně používat pouze minoritní skupina pokročilých uživatelů a administrátorů.

7.3 Návrh pokračování projektu CoNetServ

Projekt lze nadále vylepšovat a vyvíjet v moha směrech, ať už z hlediska uživatelské přívětivosti nebo funkcionality samotné. Stále lze například rozšiřovat řady podporovaných webových prohlížečů, platforem a architektur.

Díky modulárnosti pluginu a objektovému návrhu API lze přidávat novou funkcionalitu velmi snadno – a to samé samozřejmě platí také pro vylepšování funkcionality stávající. Budoucí vývoj aplikace můžou nasměrovat například tyto nápady:

- Automatické vyhledání okolních počítačů v lokální síti uživatele a následné vykreslení mapy, pomocí grafu s hranami hodnocenými dle „vzdálenosti“ (latence mezi počítači)
- Multijazyčnost aplikace (systém překladatelných souborů a možnost nastavení jazyka rozšíření)
- Možnost určit rozhraní síťové karty, přes které budou programy ping, traceroute a další posílat své pakety
- Ukazatel rychlosti síťového provozu na počítači uživatele (přijímání/odesílání paketů)
- Rychlé a jednoduché posílání souborů mezi dvěma počítači
- Úspěšné zabalení aplikace do balíčku rozšíření pro prohlížeč Safari
- Plná podpora platformy Android a vytvoření balíčku aplikace
- Instalátor pluginu na platformě Windows (např. pro prohlížeč Opera)

Literatura

[1] Oracle SDN, Sun Developer Network: Signed Java Applets. 8.4.2011 [cit. 11.4.2011]. URL: <http://java.sun.com/developer/onlineTraining/Programming/JDCBook/signed.html>

[2] Mozilla Developer Network, XPCOM Interface Reference – nslProcess [online]. URL: <https://developer.mozilla.org/en/nsIProcess>

[3] Eich, B.: Mozilla 2 Roadmap [online]. 13.10.2006 [cit. 14.4.2011]. URL: <http://weblogs.mozillazine.org/roadmap/archives/2006/10/mozilla_2.html>

[4] Microsoft Support: Description of Internet Explorer Support for Netscape-Style Plug-ins [online]. 31.1.2007 [cit. 13.4.2011]. URL: <http://support.microsoft.com/kb/306790>

[5] Giannandrea, J.: Microsoft breaks Web Plugins in Windows XP [online]. 4.9.2001 [cit. 14.4.2011]. URL: <http://classic-web.archive.org/web/20071016233843/http://www.meer.net/jg/broken-plugins.html>

[6] TheCounter.com: Browser Stats Jul-Aug 2001 [online]. 1.9.2001 [cit. 13.4.2011]. URL: <http://www.thecounter.com/stats/2001/August/browser.php>

[7] The Chromium Project: Native Client - Backround and Basics [online]. 17.4.2011. URL: <http://www.chromium.org/nativeclient/getting-started/getting-started-background-and-basics>

[8] Sullivan J.: Firefox 2011 Roadmap [online]. 7.2.2011. URL: <https://wiki.mozilla.org/Firefox/Roadmap>

[9] Yee B., Dardyk G., Chen J. B., Muth R., Ormandy T., Okasaka S., Narula N., Fullagar N.: Native Client: A Sandbox for Portable, Untrusted x86 Native Code. 19.8.2010.

[10] Donovan A. Muth R., Chen B., Sehr D.: Portable Native Client Executables. 22.2.2010.

[11] Gecko Plugin API Reference: NPAPI [online]. URL: <https://developer.mozilla.org/en/Gecko_Plugin_API_Reference>

[12] Opera Sofware: The Opera plug-in interface [online]. 18.1.2008 [cit. 17.4.2011]. URL: <http://dev.opera.com/articles/view/the-opera-plug-in-interface>

[13] Bateman, R.: FireBreath Project - Features [online]. 13.1.2011 [cit. 17.4.2011]. URL: <http://www.firebreath.org/display/documentation/Features>

[14] Google Code: An introduction to using Nixysa [online]. 26.8.2009. URL: <http://code.google.com/p/nixysa/w/list>

[15] Aas J.: NPAPI Plugin Samples [online]. 6.6.2010 [cit. 3.5.2011]. URL: <https://developer.mozilla.org/en/Plugins/Samples_and_Test_Cases>

[16] Launchpad.net: Bazaar repository of NSPluginWrapper project. 23.5.2011. URL: <http://bazaar.launchpad.net/~ubuntu-dev/nspluginwrapper/ubuntu/files/head:/src/>

[17] The GNU Project: Browsing the Git Repository of GNU Gnash project. 15.4.2011. URL: <http://git.savannah.gnu.org/cgit/gnash.git/tree/plugin/npapi>

[18] Apache Ant 1.8.2 Manual [online]. 3.4.2011. URL: <http://ant.apache.org/manual/index.html>

[19] Apache Maven Project: Introduction [online]. 3.4.2011. URL: <http://maven.apache.org/what-is-maven.html>

[20] Qt Reference Manual: qmake Manual [online]. 17.4.2011. URL: <http://doc.qt.nokia.com/latest/qmake-manual.html>

[21] The SCons Wiki: Getting Started [online]. 17.4.2011. URL: <http://www.scons.org/wiki/>

[22] CMake: Cross Platform Make Tutorial [online]. 18.4.2011. URL: <http://www.cmake.org/cmake/help/cmake_tutorial.html>

[23] Martin K., Hoffman B.: Mastering CMake 4th Edition. ISBN: 1930934203 9781930934207. 2008.

[24] Neundorf A.: Why the KDE project switched to CMake and how. 21.6.2006. URL: <http://lwn.net/Articles/188693/>

[25] Hanwell M.: CMake Performance with Open Babel [online]. 11.4.2009 [cit. 18.4.2011]. URL: <http://blog.cryos.net/archives/217-CMake-Performance-with-Open-Babel.html>

[26] Kernel.org: Git Tutorial [online]. 18.4.2011. URL: <http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html>

[27] Mozilla Developer Network: XPInstall - Cross-Platform Install Technology [online]. 21.4.2011. URL: <https://developer.mozilla.org/en/XPInstall>

[28] Google Chrome Extensions: CRX Package Format [online]. 24.4.2011. URL: <http://code.google.com/chrome/extensions/crx.html>

[29] Chrome Web Store: Network and Internet Tools [online]. 31.12.2010. URL: <https://chrome.google.com/webstore/detail/ekpdpmpcgcmpaeokmclflfpadaklgpji>

Seznam příloh

Příloha A

Obsah CD

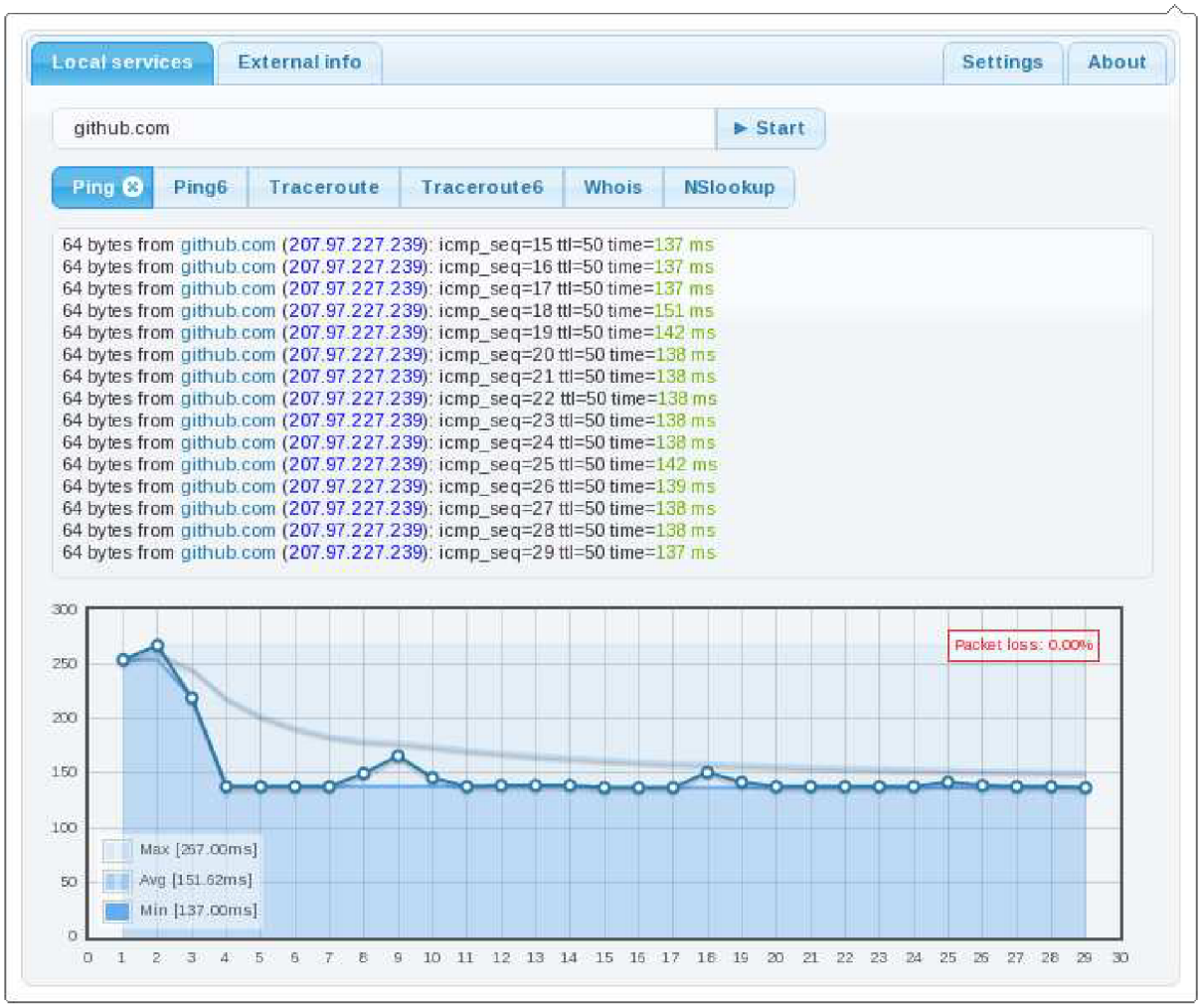
- Zdrojové texty projektu CoNetServ, včetně souborů vygenerovaných při překladu projektu na platformách GNU/Linux, Microsoft Windows a Darwin/Mac OS X na 32 i 64-bitových architekturách
  - source_build/
  - source_build/build/
- Čistý git repozitář projektu CoNetServ ke dni 12. května 2011
  - source_clean_git/

CMakePlugins/	CMake pluginy
extension/	Zdrojové kódy rozšíření
extension/chrome/	Specifické soubory pro Chrome
extension/firefox/	Specifické soubory pro Firefox
extension/opera/	Specifické soubory pro Operu
doc/	Dokumentace (doxygen)
plugin/	Zdrojové kódy NPAPI pluginu
plugin/apple/	Specifické soubory pro Darwin/Mac OS X
plugin/unix/	Specifické soubory pro Unix (GNU/Linux)
plugin/win/	Specifické soubory pro Microsoft Windows

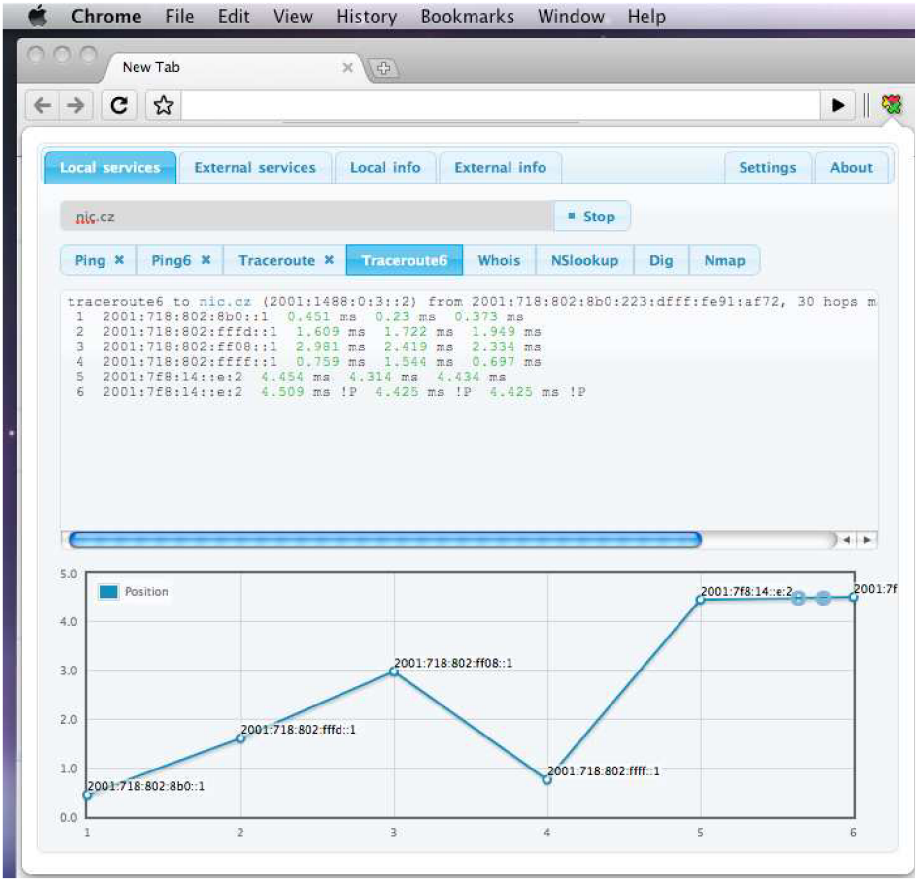
- Výsledné balíčky rozšíření CoNetServ, která lze lokálně nainstalovat v jednotlivých prohlížečích
  - browser_packages/Firefox/
  - browser_packages/Chrome/
  - browser_packages/Opera/
- Výsledné binární soubory – knihovny, které lze nakopírovat do speciálních složek na cílové platformě, dle tabulky 3.11
  - browser_plugins/
- Screenshoty aplikace CoNetServ při běhu v různých prohlížečích a platformách
  - screenshots/
- Programátorská dokumentace Doxygen pluginu verze 2.0.0
  - doc/index.html

Příloha B

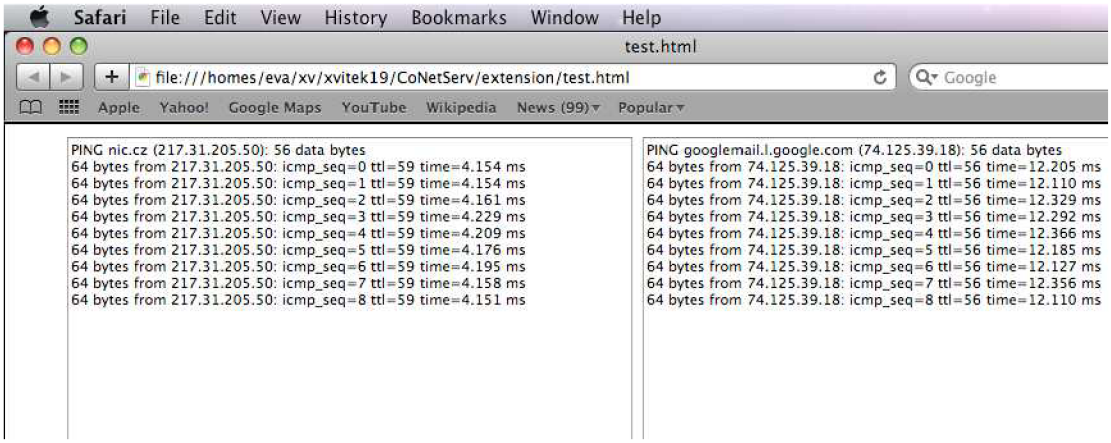
Screenshoty aplikace CoNetServ



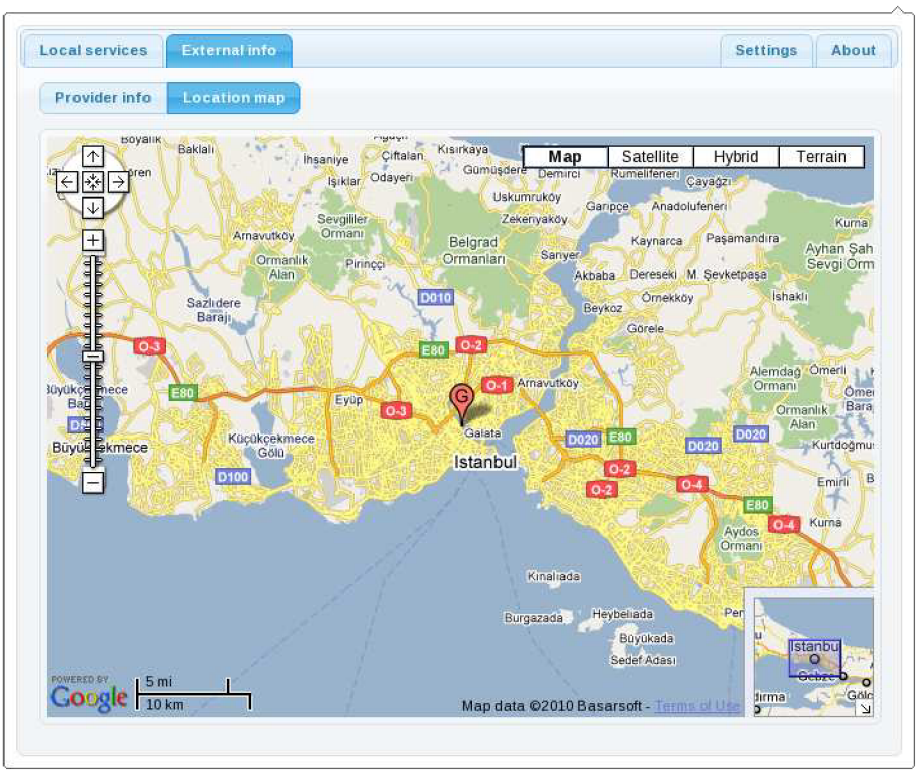
Obrázek 16: CoNetServ v1.3.0: Lokální služba PING (Chromium, Fedora 13 GNU/Linux)



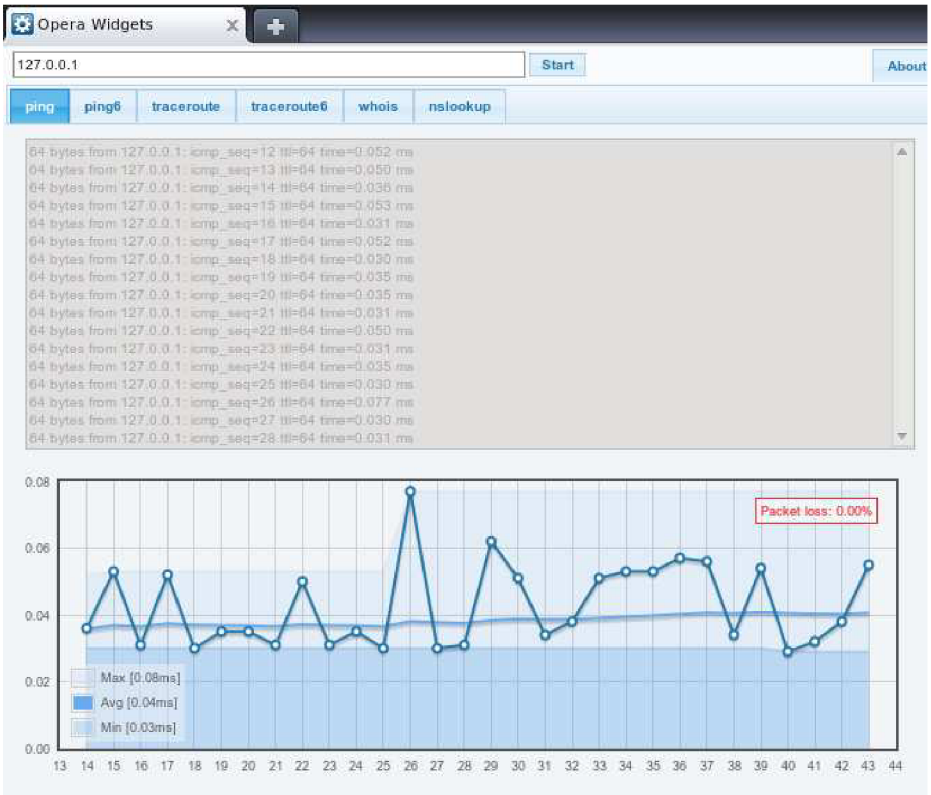
Obrázek 17: CoNetServ v2.1.0: Lokální služba TRACEROUTE IPv6 (Google Chrome, Mac OS X 10.6)



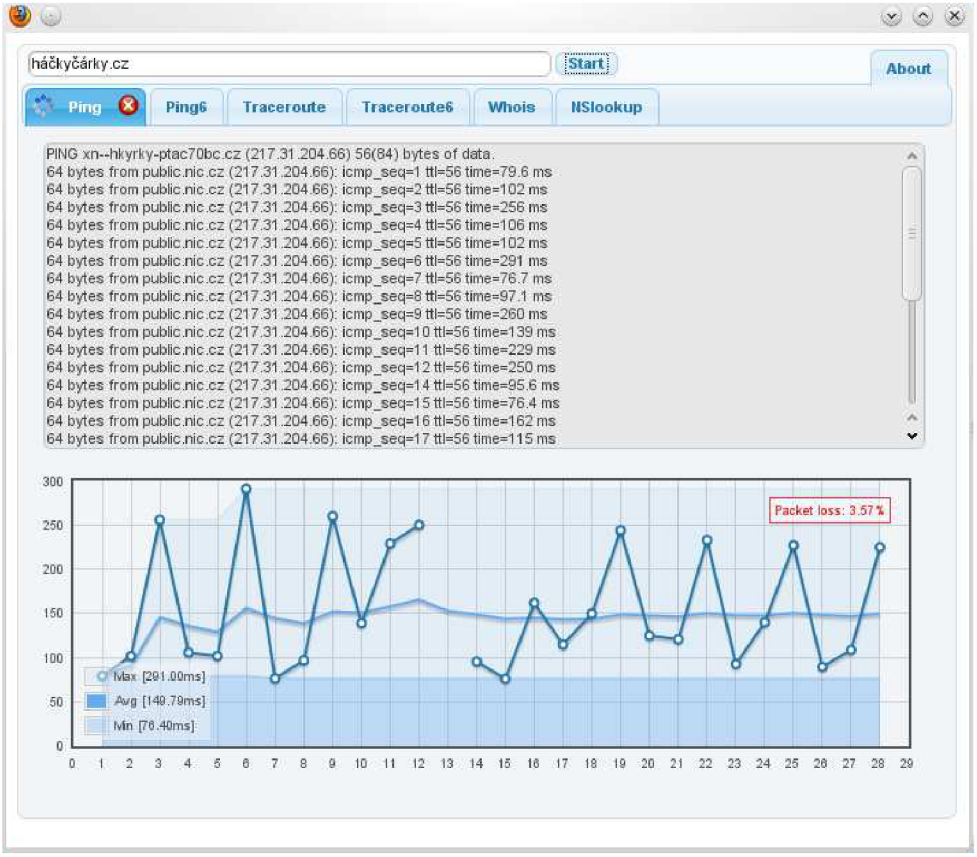
Obrázek 18: CoNetServ v2.1.0 (testovací fáze prohlížeče Safari): Lokální služba PING (Safari, Mac OS X 10.6)



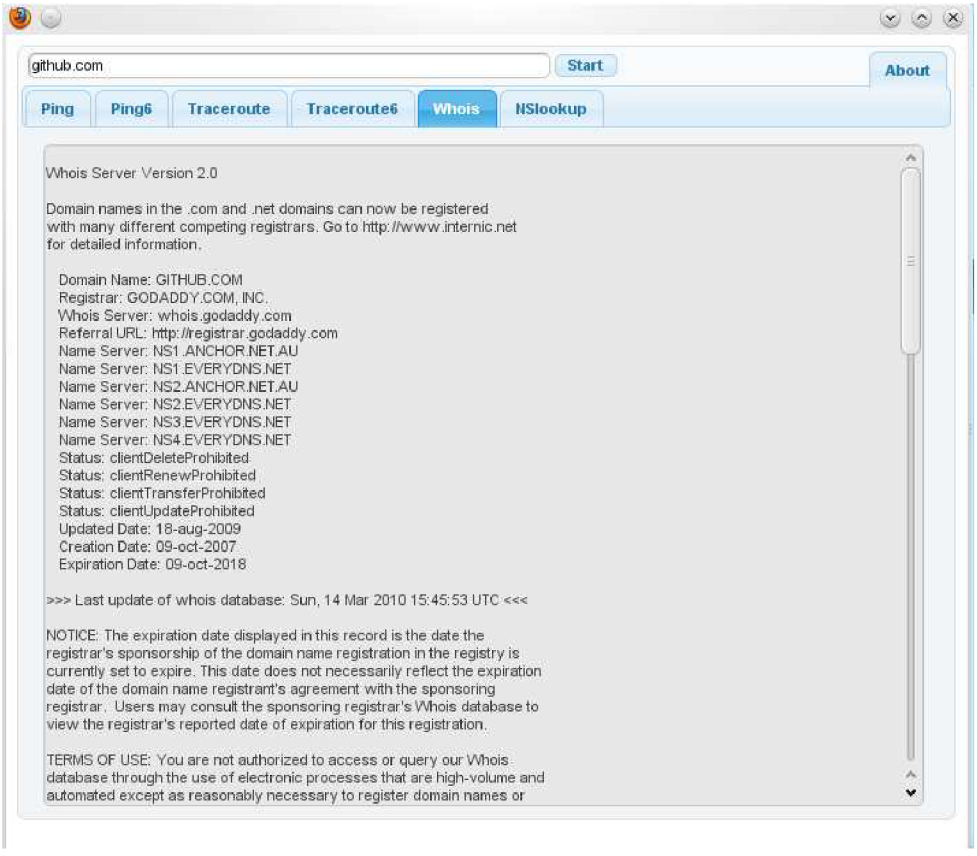
Obrázek 19: CoNetServ v1.3.0: Interaktivní mapa (Google Chrome, Windows XP)



Obrázek 20: CoNetServ v1.3.0: Lokální služba PING (Opera, Ubuntu 10.4 GNU/Linux)



Obrázek 21: CoNetServ v1.0.0: Lokální služba PING - Punycode (Iceweasel 3.5, Debian 5.0 "lenny" GNU/Linux)



Obrázek 22: CoNetServ v1.0.0: Lokální služba WHOIS (Firefox 3.5, Cent OS GNU/Linux)

Příloha C

Manuál k překladu projektu (anglicky)

Building on Unix-like systems

Debug mode

```
$ mkdir -p build/$(uname)/ && cd build/$(uname)
$ cmake ../../
$ make
```

Release mode

```
$ mkdir -p build/$(uname)/ && cd build/$(uname)
$ cmake -DCMAKE_BUILD_TYPE=Release ../../
$ make
$ make Packages
```

NOTE: (building 32bit plugin on 64bit system)

```
$ cmake -DCMAKE_C_FLAGS=-m32 -DARCH=x86 -DCMAKE_BUILD_TYPE=Release ../../
```

NOTE 2: You will need crxmake.py script for generating debug
Chrome extension .crx.
You will probably need to run:

```
$ chmod +x ./crxmake.py
$ PATH="$PATH:$(pwd)"
  # or edit your .bashrc file to add the path during shell init
```

Download crxmake.py from http://github.com/Constellation/crxmake.

Building plugin on Windows

1. Run CMake (download CMake)
2. Set source code path
3. Set build path (eg. source code path + /build/Windows)
4. Select build type
   - Debug mode (development)
     Default
   - Release mode (exports optimized shared library without debug messages)
     Set CMAKE_BUILDTYPE variable to "Release" value
5. Set XULRunnerSDK_INCLUDE_DIR (download XULRunner SDK)
6. Click to Configure button
7. Select generator, eg. Visual Studio 2008 (64bit)
8. Click to Configure button again
9. Click to Generate button
10. Open generated project file and you are ready to develop