VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

Fakulta informatiky a statistiky

Katedra informačního a znalostního inženýrství

Možnosti využití XSLT v současných webových prohlížečích

Bakalářská práce

Ondřej Bittner

Vedoucí práce: Ing. Jiří Kosek

prosinec 2011

Anotace

Cílem této práce je analyzovat možnosti současných webových prohlížečů při zpracování XSLT. Jednotlivé prohlížeče jsou porovnávány z několika hledisek jako jsou: úroveň implementace XSLT 1.0 a XSLT 2.0, možnosti spouštění transformací a výkon při zpracování XSLT. Zkoumány jsou především nedostatky, zvláštní vlastnosti či nestandardní chování jednotlivých prohlížečů. Dále jsou analyzovány možnosti zpracování XSLT v prohlížečích s použitím jiných než nativních prostředků. Závěr práce je věnován popisu vzorové aplikace využívající XSLT transformace.

Annotation

The aim of this thesis is to analyse possibilities of current web browsers in performing XSLT transformations. Individual browsers are compared by variety of different aspects such as: the level of implementation of XSLT 1.0 and XSLT 2.0, possibilities for invoking trasnformations and performance during XSLT processing. The analysis aims on flaws, special features or non-standard behavior of indiviual browsers. Possibilities for XSLT transformations in browsers using non-native instruments are researched as well. The conclusion of this thesis is dedicated to description of sample application that is using XSLT transformations.

Poděkování

Rád bych poděkoval vedoucímu této bakalářské práce Ing. Jiřímu Koskovi za podnětné rady, připomínky a čas, který mé práci věnoval.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a použil pouze literaturu uvedenou v přiloženém seznamu. Nemám námitek proti půjčení práce se souhlasem katedry, ani proti zveřejnění práce, nebo její části.

V Praze dne 6. prosince 2011

Ondřej Bittner

Obsah

Kapitola 1

Úvod

I přes to, že od publikování prvního standardu XSLT 1.0 uběhlo již 12 let, není frekvence výskytu projektů, které využívají XSLT transformací na klientské straně (tedy v prohlížeči) nijak velká a mezi webovými vývojáři se těší spíše malé popularitě. Což víceméně zanechává současný prostor tohoto řešení datové prezentace pouze pro serverové zpracování XSLT transformací. Důvody této situace mohou být různé, ale společným jmenovatelem mnoha z nich bude nedostatečná podpora právě ze strany výrobců webových prohlížečů v počátcích XSLT. Na poli webových prohlížečů se však situace rapidně mění každou chvílí a dnes již všechny běžně používané produkty poskytují nějakou podporu XSLT 1.0. A protože samozřejmě stejně jako v implementaci jiných webových standardů i zde se prohlížeče v některých aspektech liší, si tato práce klade za úkol zmapovat limity, nedostatky v implementaci, nestandartní chování a v některých případech i rozšíření, kterými jednotlivé prohlížeče disponují při práci s XSLT transformacemi. Účelem této práce je tedy nabídnout ucelený pohled na rozdíly mezi současnými prohlížeči, který usnadní vývoj webových aplikací a prezentací používajících tuto technologii.

V druhé kapitole se budu věnovat významu XSLT transformací při prezentaci na webu jako celku. Následně budou rozebrány rozdíly, výhody a nevýhody při použití serverového a klientského řešení. Pokusím se také nastínit situace, ve kterých se domnívám, že je možno efektivně používat transformace na klientovi. Kapitola třetí definuje jednotlivé prohlížeče, které budou dále podrobeny analýze a alternativní možnosti zpracování na klientské straně pomocí nenativních XSLT procesorů. Čtvrtá kapitola pak popisuje jednotlivé možnosti připojení dokumentu a užití parametrů pro kontrolu transformace v jednotlivých prohlížečích a při užití alternativních nástrojů. Pátá kapitola se věnuje úrovni implementaci XSLT 1.0 a XSLT 2.0 v jednotlivých prohlížečích. Šestá kapitola se zabývá měřením výkonu nativních XSLT procesorů pomocí benchmarku v jazyce JavaScript. Kromě popisu samotného benchmarku a metodiky měření samozřejmě prezentuje i porovnatelné výsledky. Sedmá kapitola je věnována popisu názorné ukázky užití XSLT transformací na jednoduché vzorové aplikaci.

Vzhledem k rychlému vývoji na poli webových prohlížečů, webových standardů a obecně ve světě informačních technologií, berte prosím při čtení této práce v potaz, že informace zde obsažené se vztahují k období psaní této práce což je období října až prosince 2011. Nicméně vzhledem k tomu, že při tvorbě webových aplikací je často nutno dbát na zpětnou kompatibilitu, věřím, že závěry této práce budou k užitku i v budoucnu.

Kapitola 2

Reprezentace XML v prohlížeči

V této kapitole bych se chtěl stručně věnovat významu reprezentace XML dokumentů v prohlížeči. A dále rozdílným přístupům ke zpracování XML dokumentů pomocí XSL stylu tedy zpracování na klientovi a zpracování na serveru. V této práci se budu zabývat samozřejmě hlavně prvním způsobem, nicméně je vhodné ukázat si hlavní rozdíly mezi oběma způsoby zpracování.

2.1. Oddělení formy a obsahu

Separace obsahu a formy neboli způsobu, jakým budou data zobrazována, je jedním z hlavních pilířů filozofie XML. K praktickému využití tohoto přístupu je však potřeba mít nástroj k zobrazování dat a produkci rozdílných výstupů. V rodině XML technologií jsou tímto nástrojem XSLT transformace.

Výsledkem XSLT transformace může být prakticky cokoliv. XML dokument s jinou strukturou, HTML nebo XHTML dokument, CSV, SQL dávka – možností je nepřeberné množství. V této práci se však budu zajímat zejména o situaci, kdy je XSLT transformace použita k transformování XML dokumentu do HTML, potažmo XHTML dokumentu zobrazitelného ve webovém prohlížeči.

Zobrazování dat na webu výše uvedeným způsobem by se dalo považovat za další krok v oddělování formy od obsahu. V raných dobách WWW byly data, struktura dokumentu i jeho styl a design zamíchány dohromady v jediném souboru. Nemotornost takového návrhu byla nejvíce patrná při změnách stylu. Primitivní operace jako změna barvy prvku, či velikosti písma znamenala otrocké procházení dokumentu a přepisování atributů HTML tagů.

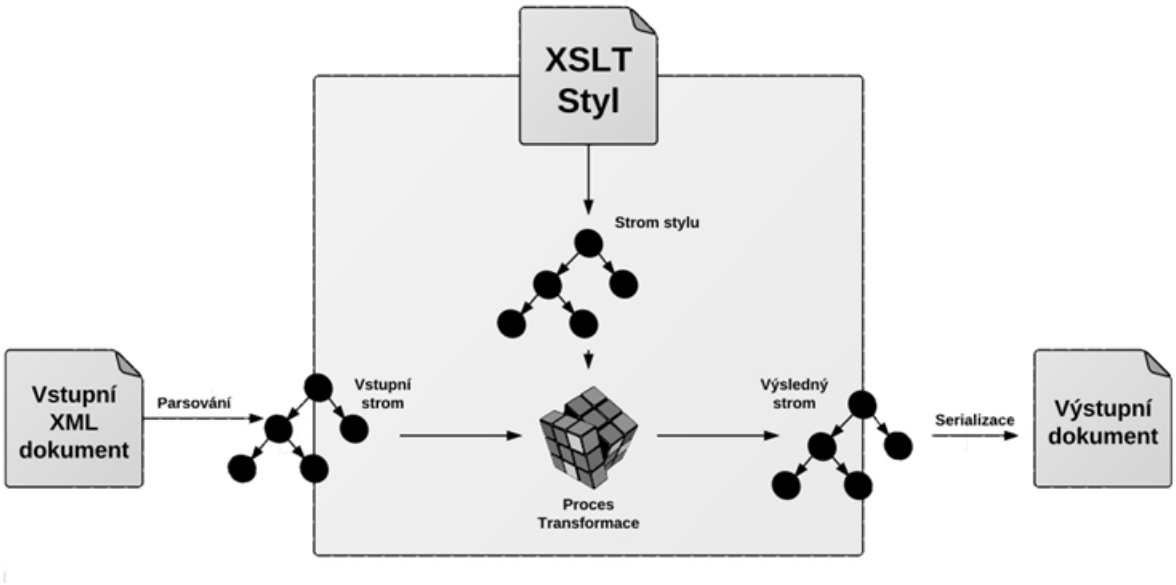
Bylo jasné, že je potřeba změna. Tu přinesli kaskádové styly. Rázem bylo možno definovat zobrazování dokumentu jednoduše a elegantně. CSS se od té doby stalo nedílnou součástí webu. Jeho hlavní výhodou a zároveň nevýhodou se zdá jednoduchost. Je sice schopno měnit vzhled prvků, avšak nemá žádný způsob jak ovlivnit strukturu dat anebo provádět jiné komplexní operace, kterých je XSLT schopno. Použití CSS jako stylu pro XML dokument je tak možné jen v případě, že struktura XML dokumentu je již v takovém stavu, že není potřeba žádných strukturálních změn.[1][9]

Právě úplného oddělení dat, struktury a stylu můžeme dosáhnou pomocí XSL stylu. Ten je schopen nejen měnit strukturu ať už seskupováním, řazením, přeskládáním prvků či jejich úplným vynecháním z výstupního dokumentu, ale také zvládá obsah generovat a importovat části jiných dokumentů. Nad takovýmto komplexním obsahem je pak možno provádět např. početní operace nebo operace s textovými řetězci.

2.2. Server vs. klient při zpracování XSLT

Rozdíl mezi oběma přístupy je samozřejmě v tom, kde se transformace provádí. U serverového zpracování je v zásadě lhostejné, zdali vstupní XML dokument pochází od klienta nebo je jeho zdrojem umístění na serveru či někde úplně jinde. Důležité je, že transformaci provádí server a ke klientovi se vrací již vygenerovaný výstupní dokument. Naopak u klientského zpracování je XSLT transformace prováděna XSLT procesorem, který je obsažen v prohlížeči a který poté zobrazí výsledek transformace.

Porovnáváme-li oba způsoby z obecného hlediska, má serverové zpracování dvě velké výhody. Servery bývají o poznání výkonnější než koncové klientské stanice. Což je pro serverové řešení výhoda, neboť XSLT transformace je ze své povahy poměrně náročná na paměť a procesorový výkon. Při transformaci jsou totiž vstupní dokumenty převedeny do stromové reprezentace, která je poté uchována v paměti. Ta je obvykle několikrát větší než serializovaná forma dokumentu.[8] Kromě toho se v paměti též uchovává stromová reprezentace stylu a výstupních dokumentů, které jsou také reprezentovány ve stromové struktuře, předtím než jsou serializovány. Celé znázornění transformace je možno vidět na obrázku 2.1.



Obrázek 2.1. Proces transformace XML dokumentu (zdroj: [1])

Za povšimnutí stojí, že XSLT procesor má za povinnost transformovat pouze zdrojové stromy ve stromy výstupní.[1] Celému procesu tak logicky musí předcházet parsování vstupního XML dokumentu a následovat serializace výstupního stromu v konečný dokument. Tyto funkcionality nemusí XSLT procesor vůbec obsahovat (nicméně v reálu je zpravidla obsahuje). Funkcionalita serializace stromu reprezentující výsledek transformace ve výstupním dokumentu je však nadbytečná, pokud jsme schopni pracovat už s výsledným stromem. Což je právě případ transformací v některých prohlížečích, které rovnou pracují s výsledným stromem.

Druhou často zmiňovanou výhodou zpracování XSLT transformací na serverové straně je kontrola nad procesorem provádějícím transformaci. Zatímco u zpracování klientského se výsledek plně odvíjí od toho, jaký uživatel používá prohlížeč (tedy jaký používá XSL procesor

implementovaný v daném prohlížeči). Při serverovém zpracování je za transformaci vždy odpovědný jediný procesor a výstupní dokumenty jsou tak konzistentní. U klientského zpracování pak může být rozdílné chování XSL procesorů vážnou překážkou ve zpracování některých stylů. Výstupní dokumenty se mohou v nejrůznějších ohledech lišit podle použitého prohlížeče, ale může nastat i situace, kdy jednoduše jeden z prohlížečů vůbec není schopen danou transformaci provést, nebo je výsledek natolik vzdálený původnímu záměru, že je nepoužitelný.

Právě rozdílné chování prohlížečů nebo jejich omezení připomínají zavádění již zmiňovaných kaskádových stylů. První doporučení CSS bylo položeno v roce 1996. Teprve v roce 2000 byl uvolněn Internet Explorer 5 pro Mac, který implementoval 99% standardu CSS 1.0[11]. Další prohlížeče pak navázali, nicméně napříč prohlížeči stále přetrvávala situace velmi rozdílného vykreslování dokumentů používajících kaskádové styly. A samotné jednotlivé implementace trpěly nejrůznějšími bugy, jejichž odstraňování přicházelo až s dalšími verzemi prohlížečů a situace se zlepšovala. Podobný vývoj je právě možno sledovat u implementace XSLT 1.0. A stejně jako u CSS se situace postupně zlepšila tak, že dnes již většina majoritních prohlížečů poskytuje z XSLT 1.0 alespoň tu nejzákladnější funkcionalitu.[2]

Klientské zpracování přináší jednu zásadní výhodu – snížení nároku na server. Server v klientském zpracování poskytne uživatelské straně pouze vstupní XML dokument spolu s XSLT stylem a dále se již nemusí daným uživatelským požadavkem zabývat, protože transformaci zajistí klientská strana. Další vlastností, kterou lze v některých případech považovat za výhodu, je výrazně lepší práce s uživatelskou interakcí. Pokud v aplikaci zachycujeme javascriptové události, které následně určují podobu transformace, tak při serverovém zpracování musí být tyto interacke složitě předány serveru, který zpět vrací výsledek transformace. Na výsledek je tak nutno čekat, což snižuje uživatelský komfort. V případě, že transformace zpracováváme na klientské straně a styl ovládáme pomocí JS API (podrobně se o tomto postupu zmíním v části 4.3) výsledky transformace mohou být okamžitě do dokumentu zapisovány, bez refreshe stránky.

Proti samozřejmě stojí omezení definována výše a to náročnost na výpočetní kapacity a nemožnost kontrolovat jaký procesor klient používá. I tak si ale můžeme představit podmínky, ve kterých je možno nejen klientské zpracování uplatnit, ale ve kterých se jeví jako vhodnější než řešení serverové.

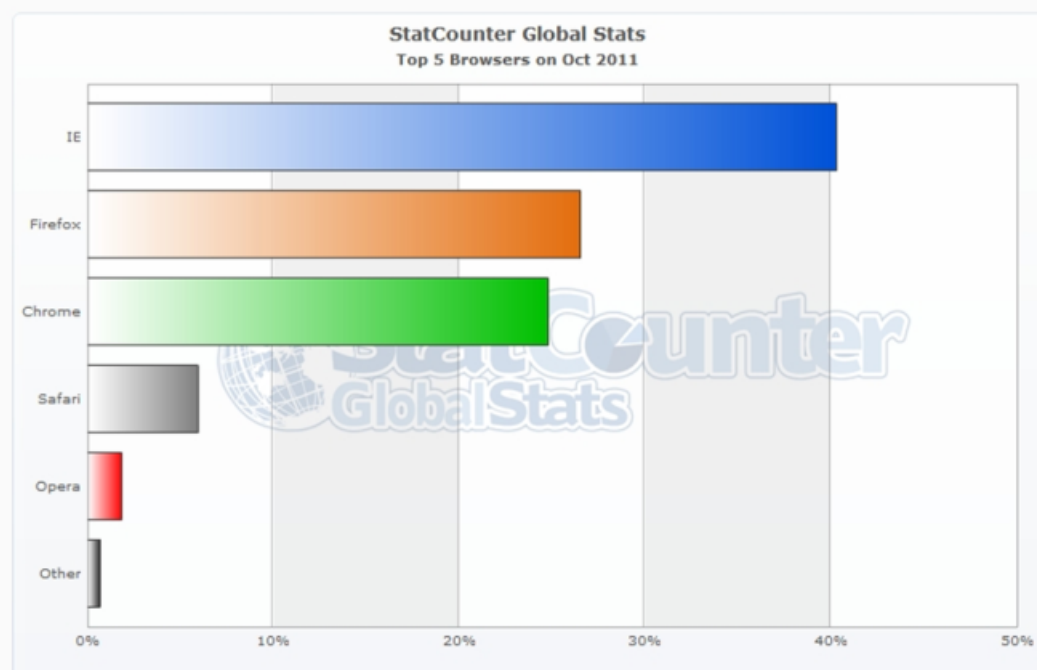
Jednou ze situací, kde se klientské zpracování může prosadit je intranet. V organizacích, kde může docházet k velkému množství požadavků na zobrazování nejrůznějších XML dokumentů z podnikového informačního systému (faktury, objednávky, informace o klientech atd.) je klientské zpracování ve výhodě. Kontrolované prostředí organizace totiž umožňuje vynutit na uživatelích používání prohlížeče, který je zobrazování XML dokumentů s XSLT stylem schopen bez jakýchkoliv potíží. Samozřejmě otázka výkonu zůstává. Klientské zpracování je tak spíše vhodné pro méně komplexní transformace, kde doba zpracování dokumentu bude přijatelně dlouhá.

V prostředí internetu samozřejmě již kontrola nad prohlížečem používaným uživatelskou stranou není možná. Proto je nutné použít metody, které pokud možno unifikují výsledek napříč používanými prohlížeči. Je možné otestovat chování konkrétního stylu v běžně používaných prohlížečích, a pokud některý z nich má problémy se zpracováním stylu, přistoupit k tvorbě alternativního stylu, který bude adaptován na konkrétní prohlížeč. Stále je tak možné použít XSLT pro velké webové projekty, kde by serverové zpracování velkého množství požadavků vyžadovalo přílišnou výpočetní kapacitu.

Kapitola 3

Selekce porovnávaných prohlížečů

Vzhledem k množství existujících prohlížečů jsem do výběru, který bude podroben další analýze, vybral pouze prohlížeče, které se dají označit za majoritní. Prohlížeče jsem tak vybíral podle jejich podílu na současném trhu. Následující údaje byly získány z online webové služby statcounter.com.



Obrázek 3.1. Podíl webových prohlížečů na trhu - říjen 2011 (zdroj: statcounter.com)

Do výběru tak byli zařazeny následující prohlížeče v poslední release verzi:

- Internet Explorer 9.0
- Firefox 7.0
- Chrome 14.0
- Safari 5.1

- Opera 11.5

S tím, že považuji za vhodné zmínit, že výše uvedené prohlížeče mají různě úplné XSLT implementace již od verze:

- Internet Explorer 6.0
- Firefox 3.0
- Chrome 1.0
- Safari 3.0
- Opera 9.0

3.1. Poznámky k jednotlivým prohlížečům

Internet Explorer 9.0 používá jako XML parser i XSLT procesor svoji knihovnu (MS nazývanou kolekcí služeb) MSXML 6.0, proto výsledky obsažené v této práci např. ohledně úplnosti implementace XSLT 1.0 v IE9 (tedy v MSXML 6.0) mohou být uplatněny na aplikace pracující s touto knihovnou.

U Firefoxu 7.0 obstarává XSLT transformace vykreslovací jádro Gecko 7.0, resp. v Gecku obsažený XSLT procesor TransforMiix. Ten aktuálně používá i poslední stable verze prohlížeče SeaMonkey, konkrétně verze 2.4. Ten sice není obsažen ve výše zmíněném výběru, ale výsledky je na něj možné aplikovat právě prostřednictvím Gecka 7.0.

Google Chrome a Safari sdílejí vykreslovací modul WebKit, který se stará o XSLT transformace prostřednictvím knihovny libxslt. Chrome 14.0 využívá verzi jádra WebKit 535.1 a Safari 5.1 verzi 534.50 a z toho důvodu se i přes stejné vykreslovací jádro mohou výsledky lišit.

3.2. Další možnosti klientského zpracování

Kromě nativně zabudovaných XSLT procesorů lze použít ke zpracování XML dokumentu i externí javascriptové knihovny, které suplují funkce nativně implementovaných procesorů. Důvodů k využití tohoto řešení může být několik. Zaprvé je možno touto metodou provádět XSLT transformace i v prohlížečích, kterým tato funkce v nativním prostředí schází.[1] Za druhé je tímto způsobem možno ignorovat rozdíly mezi implementacemi v nativních XSLT procesorech. Tedy stačí pokud je styl funkční v externím XSLT procesoru, není se pak potřeba zabývat kompatibilitou. Což odstraňuje jednu z hlavních překážek v klientském zpracování XSLT.

Zřejmě největší nevýhodou tohoto řešení je fakt, že takovéto knihovny mohou být poměrně velké a při prvním zobrazení cílového dokumentu může docházet k nepříjemnému čekání na to, až se skript stáhne.

Ve své práci se budu věnovat dvěma takovýmto produktům[2], které budu porovnávat s nativními procesory v prohlížečích.

[1] Samozřejmě za předpokladu, že tento prohlížeč podporuje JavaScript

[2] Nepodařilo se mi vypátrat jiné JS XSLT procesory, nicméně je samozřejmě možné, že existují.

3.3. Saxon CE

Celým názvem Saxon Client Edition je další verzí jednoho z nejpoužívanějších XSLT procesorů (vedle Home Edition, Professional Edition a Enterprise Edition). A jak název této verze napovídá je určena pro klientská zpracování. Jedná se o odlehčenou verzi Home Edition (Saxon HE má odhadem 143 tisíc řádků nekomentovaného kódu, Saxon CE 62 tisíc). Odlehčení kódu probíhalo zejména v oblastech, které nejsou pro použití v prohlížečích důležité. Například XQuery funkcionalita, serializace či různá přebytečná API. Saxon CE je původně napsán v Javě a následně pomocí Google Web Toolkit zkompilován do JavaScriptu a nyní se nachází v alpha testovací verzi.[3]

Hlavní předností Saxonu CE je fakt, že jako odnož Saxonu 9.3 je funkčním XSLT 2.0 procesorem (resp. splňuje všechny podmínky standardu XSLT 2.0 podle W3C na úrovni "Basic XSLT 2.0 Processor"), což se nedá říci ani o jednom nativním procesoru, ani o projektu Javeline, o kterém se zmíním níže. Navíc Saxon CE přichází s rozšířením iXSL (interactive XSL), které přináší zajímavý způsob práce s uřivatelskou interakcí.[3].

3.4. Javeline

Projekt Javeline je též javascriptová knihovna suplující funkcionalitu XSLT procesoru. Je úzce spojený s projektem javascriptového API Sarissa (viz 4.3.3). Bohužel se zdá, že projekt již není ve vývoji. Jediná smysluplná reference je právě v dokumentaci Sarissy, kde se navíc nachází v současné době nefunkční odkazy pro stažení (knihovny je však stále možno nalézt jinde), nicméně jakákoliv dokumentace chybí. Na některých vývojářských fórech jsou poté vyjádřeny pochyby o tom, že Javeline je úplnou implementací XSLT 1.0. Z těchto důvodů jsem projekt Javeline do srovnání nezahrnul.[12]

[3] Aktuálně verze 0.3

Kapitola 4

Připojení a zpracování stylu

V této části budou popsány možnosti připojení stylů k XML dokumentu a způsoby, jimiž je možno na klientské straně transformaci kontrolovat a ovlivňovat. Jednotlivé metody připojení budou vždy popsány obecně a následně bude rozebráno, jak se daný způsob v daném prohlížeči chová.

4.1. Přímo připojený externí styl

Nejjednodušším způsobem jak připojit k XML dokumentu styl je za pomocí zvláštní procesní instrukce na úplném začátku dokumentu. To znamená že její výskyt musí být v prologu dokumentu, tedy před prvním XML tagem. Deklarace W3C o připojování stylů k XML dokumentům uvádí, že stylů může být tímto způsobem připojeno libovolné množství[13]. Tento způsob připojení stylu tedy zapisujeme nějak takto:

```
<?xml-stylesheet type="application/xslt+xml" href="example.xsl"?>
```

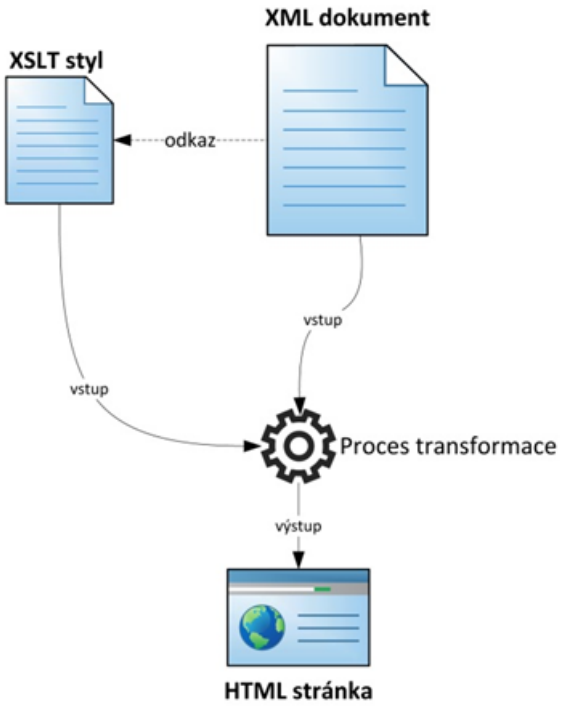
První pseudo-atribut `type` udává MIME cílového volaného dokumentu. Registrovaným typem pro XSL styly je hodnota: `application/xslt+xml`. Do nedávna tato hodnota nebyla registrována a běžně se používala hodnota: `text/xsl` prosazená Microsoftem. Kolem toho, jaký typ uvádět v pseudo-atributu `type` tak panují jisté zmatky[1].

Druhý pseudo-atribut href udává URI stylu. To může mít samozřejmě absolutní nebo relativní odkaz. Tento pseudo-atribut pak může mít i hodnotu identifikátoru.

Oba tyto pseudo-atributy jsou povinné. Dále je možno v procesní instrukci uvést další atributy: `title`, `media`, `charset` a `alternate`. Atribut title se používá v případě, že je k dokumentu pomocí procesní instrukce připojeno více stylů. Uživatel by pak měl být schopen výběru stylu. Atribut `media` charakterizuje výstupní dokument. Seznam možností je součástí specifikace HTML 4.0. Atribut `charset` je v případě volání XSLT stylu zbytečný neboť XSLT jako XML dokument svoje kódování povinně definuje (na rozdíl od CSS, které může být také tímto způsobem voláno). Nakonec atribut `alternate` má možné volby yes a no. V případě, kdy je zvoleno no styl je považován za preferovaný. V případě, kdy je hodnota atributu yes, je považován za alternativní pro uživatelskou volbu.

Zpracování dokumentu s přímo připojeným stylem je velice jednoduché. Zpracovávaným XML dokumentem je dokument, v němž je umístěna procesní instrukce. Z té je následně zjištěna

poloha XSLT stylu. Oba dokumenty jsou pak zpracovány XSLT procesorem. Celý proces je znázorněn na obrázku 4.1.



Obrázek 4.1. Proces transformace XML dokumentu s přímo připojeným stylem.

4.1.1. Firefox

V případě volání stylu přes procesní instrukci přichází Firefox (od verze 2.0) se zajímavým nástrojem. Pomocí další procesní instrukce je možné styl zavolat s parametry.[4] Procesní instrukce má tvar:

```
<?xslt-param name="example" value="something"?>
<?xslt-param name="example" select="//element"?>
```

Pseudo-atribut `name` je název parametru, jemuž chceme předat hodnotu. Ta je obsažena v pseudo-atributu `value`. V druhém případě je místo pseudo-atributu `value` použit atribut select, který obsahuje XPath výraz, jehož výsledek je předaný jako hodnota parametru. Jako kontextový uzel je brán incializační uzel transformace.

Jmenné prostory pro XPathové výrazy je možno nastavovat pomocí celkem jasné procesní instrukce:

```
<?xslt-param-namespace prefix="my" namespace="http://www.example.org/my"?>
```

Další závěry ukazují, že při pokusu o připojení více stylů Firefox akceptuje procesní instrukci, která je uvedena jako poslední. V případě použití pseudo-atributů `alternate` akceptuje ten

poslední s hodnotou: `no`. I v případě doplnění pseudo-atributů `title` nenabízí Firefox žádné uživatelské rozhraní pro výběr stylu.

Firefox respektuje použití obou MIME typů jak oficiální `application/xslt+xml`, tak prakticky používaný typ: `text/xsl`.

4.1.2. Opera a Safari

Při připojení více stylů pomocí procesní instrukce akceptují oba tyto prohlížeče první validní procesní instrukci bez ohledu na hodnotu pseudo-parametru `alternate`. Stejně jako Firefox nenabízejí žádné uživatelské rozhraní pro výběr stylu. Opera ani Safari nerespektují oficiální MIME typ a akceptují pouze styly s neoficiálním typem `text/xsl`.

4.1.3. Chrome

Chrome je totožný s chováním Opery a Safari, nicméně při testování vykázal zvláštní vlastnost. Bezpečnostní politika Chromu totiž neumožňuje nahrávání stylu z lokálního umístění (styl bylo potřeba umístit na webový server). Chybu sice Chrome detekuje, ale zobrazí ji pouze v konzoli pro vývojáře. Navíc na výstupu nic nezobrazí. A to ani původní XML dokument, přesto, že údaje o vzhledu dokumentu chybí (resp. nebyli nahrány). A standartně pokud údaje o stylu XML dokumentu chybí, je prohlížečem zobrazena surová reprezentace XML dokumentu.

4.1.4. Internet Explorer

Internet Explorer při pokusu o připojení více stylů akceptuje pouze první instrukci nehledě na hodnoty pseudo-atributu alternate. Jako MIME typ dokumentu se stylem pak akceptuje obě dvě možnosti.

4.1.5. Saxon CE

Z hlediska technického řešení tohoto XSLT procesoru není možné volat transformaci přes procesní instrukci. Vykonání transformace se ihned po otevření dokumentu ujme nativní XSLT procesor integrovaný v prohlížeči (pokud je).

4.2. Přímo vložený styl

Přímo vložené styly se analogicky podobají přímo vloženému CSS stylu. Styl je součástí dokumentu, který nese obsah. Použití takovéto metody tak zpravidla nachází užití pouze v případě samostatně stojících dokumentů, nebo dokumentů s jednorázovým použitím, kde je pravděpodobné, že styl již jinde nevyužijeme.

Přímo vložený styl je výjimkou pravidlu, že styl je XML dokument. V tomto případě je styl součástí XML dokumentu a je uvozen tagem `<xsl:stylesheet>` nebo `<xsl:transform>` (stejně jako samostatně stojící styl). Princip volání stylu je stejný jako u externího stylu přes procesní instrukci s tím rozdílem, že místo URI je volán identifikátor elementu, který uvozuje styl. Například takto:

```
<?xml-stylesheet type="text/xsl" href="#style1"?>
```

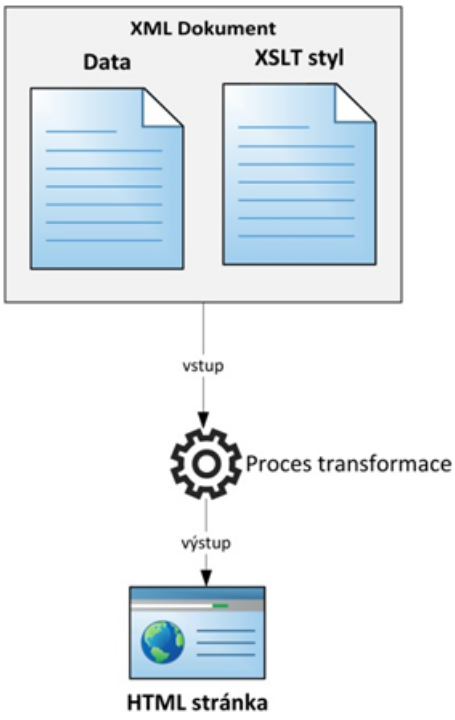
Ovšem vzhledem k faktu, že se styl nachází uvnitř XML dokumentu, je potřeba dbát dvou zásad. Zaprvé je nutné atribut id pomocí DTD nadefinovat v prologu dokumentu jako atribut jehož DTD typ je ID (tak aby ho bylo možné volat pomocí identifikátoru #id). Například takto:

```
<!DOCTYPE example [<!ATTLIST xsl:stylesheet id ID #REQUIRED>]>
```

Druhou možností je použít v elementu `<xsl:stylesheet>` id ve jmenném prostoru XML, které DTD validaci nepotřebuje. Druhou zásadou je pak vložení této šablony na začátek stylu:

```
<xsl:template match="xsl:stylesheet"/>
```

Ta se spustí v momentě, kdy transformace narazí na element `<xsl:stylesheet>` a jak je vidět je prázdná. Tato šablona zajistí, že v momentě, kdy transformace dojde na konec datové části a de facto na začátek stylu samotného, netransformuje sama sebe.[1]



Obrázek 4.2. Proces zpracování XML dokumentu s vloženým stylem

4.2.1. Firefox

Firefox bohužel neimplementuje atribut `xml:id` [4] a tudíž při snaze otevřít dokument s vloženým stylem, jenž je takto identifikován, vrátí parsing error. Jinak se chová standardně.

4.2.2. Opera

Opera se chová téměř standardně. Pouze při pokusu zobrazit dokument, který volá vložený styl přes identifikátor bez DTD validace, je styl navzdory očekávání nalezen a transformace proběhne bez jakýchkoliv problémů.

4.2.3. Safari a Chrome

Oba prohlížeče se při zobrazení dokumentu s vloženým stylem chovají standardně.

4.2.4. Internet Explorer

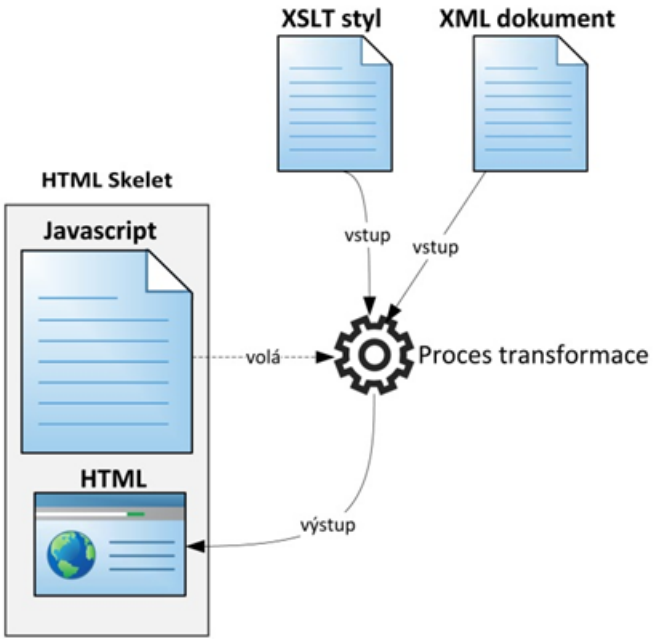
Nenašel jsem žádnou oficiální zmínku, že Internet Explorer nepodporuje vložené styly, avšak zdá se, že tomu tak je. V některých diskuzích na MSDN padly domněnky, že IE má problémy s DTD sekcí a není tak schopno naleznout styl v dokumentu. Nicméně při pokusu s použitím `xml:id`, které DTD validaci nepotřebuje, jsem došel ke stejnému výsledku. Což naznačuje, že má Internet Explorer problém s obojím, nebo se chyba nachází někde jinde.

4.2.5. Saxon CE

V případě vloženého stylu se omezení javascriptových procesorů neliší od případu 4.1.5.

4.3. JavaScript API

Obecně nejpokročilejší metodou, jak připojit styl k dokumentu, je užití JavaScriptu a s ním souvisejícího API k manipulaci XSLT transformací, které daný prohlížeč nabízí. Principiálně je celá záležitost poměrně jednoduchá. Vytvoří se dva DOM objekty obsahující oba potřebné dokumenty, tedy XML dokument, který chceme transformovat a styl, který chceme použít. Následně se vykoná transformace a výsledek se zobrazí jako HTML v okně prohlížeče. javascriptový kód, jenž transformaci řídí, se většinou nachází v předpřipraveném HTML skeletu. HTML kód, jenž je výsledkem transformace, poté JavaScript připojí k elementu se specifikovaným identifikátorem.



Obrázek 4.3. Proces zpracování XML dokumentu řízený JavaScriptem

Výhodou této metody je samozřejmě fakt, že na rozdíl od předešlých, jde spouštět transformace s různými parametry což může např. zajišťovat interakci s uživatelem (tedy je možno se na parametr dotázat uživatele). Obdobně lze volat různé styly na základě okolností (např. verze uživatelova prohlížeče) a pracovat variabilně s výstupem.

4.3.1. Firefox, Opera, Safari a Chrome

Všechny tyto čtyři prohlížeče podporují stejné API. Nahrání souboru je možné provést přes objekt `XMLHttpRequest` :

```
var xhttp=new XMLHttpRequest();
xhttp.open("GET",“example.xml“,false);
xhttp.send("");
var result=xhttp.responseXML;
```

Výsledek se nám již vrací jako DOM objekt, tedy ve stromové reprezentaci. Poté, co výše popsaným způsobem (nebo jiným: můžeme např. jeden z dokumentů chtít zparserovat z řetězce) získáme oba dokumenty, můžeme provést transformaci:

```
xsltProcessor=new XSLTProcessor();
xsltProcessor.importStylesheet(xsl);
var output = xsltProcessor.transformToDocument(xml);
```

Jak je vidět, k transformaci je potřeba vytvořit instanci XSLT Procesoru, do něj poté importovat styl a metodou `transformToDocument` s argumentem zdrojového XML dokumentu provést transformaci.

Zobrazení výsledného dokumentu na výstup pak můžeme provést mnoha způsoby např. použitím `innerHTML`., nebo metody `appendChild`.

Výsledkem transformace může být nejen nový dokument, ale i fragment. Ty jsou užitečné, pokud s nimi dále chceme nějakým způsobem manipulovat. Jediným rozdílem je to, že fragmenty musí mít dokument, jenž je jejich vlastníkem. Poslední řádek výše uvedeného příkladu by se tak dal změnit takto:

```
var output = xsltProcessor.transformToFragment(xml,document);
```

Kde druhým argumentem je právě dokument, který fragment vlastní.

Práce s parametry také není v případě těchto prohlížečů zvláště složitá. `XSLTProcessor()` poskytuje tři metody: `getParameter` (vrací hodnotu prametru), `setParameter` (nastavuje novou hodnotu parametru) a `removeParameter` (maže hodnotu parametru, resp. vrací na výchozí definovanou ve stylu). První dvě metody mají vždy stejné argumenty, jsou jimi: namespace parametru a název parametru. Metoda `setParameter` má ještě třetí argument, který určuje novou hodnotu parametru.[4]

4.3.2. Internet Explorer

Od verze 7.0 je Internet Explorer schopen pracovat s objektem `XMLhttpRequest` a tím pádem dokáže získat dokumenty obdobným způsobem jako ostatní prohlížeče. Do verze 7.0 však pracoval pouze s ActiveX objektem:[1]

```
xhttp=new ActiveXObject("Microsoft.XMLHTTP");
```

Transformace probíhá pomocí metody `transformNode()`. Tuto metodu může používat jakýkoliv XML DOM objekt a jako argument předává umístění XSLT stylu.

```
result = xml.transformNode(xsl);
```

Jedná se tak v podstatě o obdobnou situaci jako u ostatních prohlížečů. Pouze v tomto případě není potřeba vytvářet instanci objektu `XSLTProcessor` a jako argument transformační funkce slouží XSLT styl, a ne transformovaný XML dokument.

Nicméně pokud je potřeba manipulovat s parametry transformace, vytvoření procesoru a asociování stylu je nutné.

```
   processor = xslt.createProcessor();
processor.input = xsl;
```

[1] MSDN i nyní jako oficiální příklad pro IE9 uvádí: ActiveXObject("Msxml2.DOMDocument.6.0")

Objekt XSLT procesoru používá pro manipulaci s parametry metodu `addParameter`, která má tři argumenty: jméno parametru, hodnotu parametru a namespace parametru. V případě, že je potřeba parametr odstranit, volá se tato metodou s novou hodnotou parametru nastavenou na `NULL`. [5]

4.3.3. Sarissa

Pokud jde o javascriptová API a XSLT transformace, neměl by být opomenut projekt Sarissa (http://www.dev.abiss.gr/sarissa/). Tento projekt vznikl jako snaha o vytvoření univerzálního API pro práci s XML (parsování, serializaci DOM objektů, XSLT transformace apod.). Sarissa ve své podstatě zastřešuje nativní API webových prohlížečů svým vlastním API, což samozřejmě usnadňuje práci vývojářům.

Jakožto javascriptový kód je nejprve nutno Sarissu do dokumentu, ve kterém chceme transformace invokovat, vložit. To můžeme učinit jednoduše na začátku dokumentu:

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

Zbytek je velmi podobný JS API např. Firefoxu. Jednoduchá transformace pomocí API Sarissa následně může vypadat třeba takto:

```
var proc = new XSLTProcessor();
var xhttp=new XMLHttpRequest();
xhttp.open("GET","example.xml",false);
xmlDoc=xmlhttp.responseXML;
xhttp.open("GET","example.xsl",false);
xslDoc=xmlhttp.responseXML;proc.importStylesheet(xslDoc);
var result = proc.transformToDocument(xmlDoc);
```

Jak je vidět, i zde je potřeba vytvořit instanci objektu `XSLTProcessor` a následně s ní pracovat. Získávání obou potřebných dokumentů probíhá opět pomocí instance objektu `XMLhttpRequest`, ale jak už bylo řečeno dokumenty je možno získat i jinak (např. parsováním z řetězce). Metoda `importStylesheet` slouží k vzájemnému asociování stylu a procesoru a jediným argumentem je logicky DOM dokument stylu. Metodou `transformToDocument` pak spouštíme samotnou transformaci, argumentem je DOM dokument zdrojového XML. Alternativně je možné transformovat pouze do fragmentu, pomocí metody `transformToFragment`, přičemž stále platí, že každý fragment musí být vlastněn DOM dokumentem.

Sarissa samozřejmě nabízí i sadu metod pro práci s parametry. Metoda `getParameter` vrací hodnotu parametru, argumenty jsou namespace a jméno parametru. Metoda `setParameter` nastaví hodnotu parametru podle argumentu value. A konečně metoda `clearParameters` vrátí všem parametrům výchozí hodnotu definovanou ve stylu.[12]

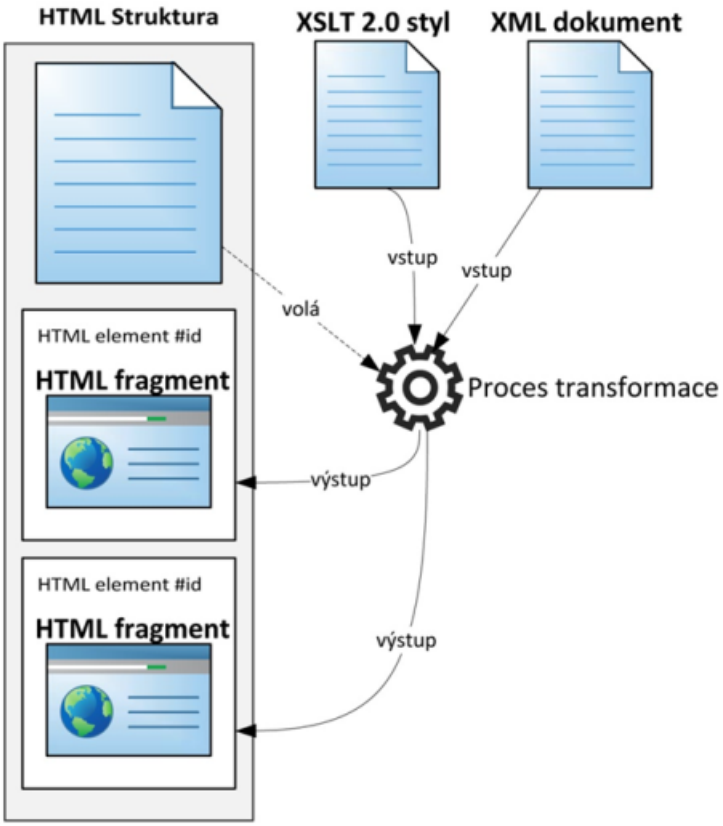
4.3.4. Saxon CE

Invokování transformace v Saxonu CE se výrazně liší od předešlých způsobů. Jak bylo řečeno už dříve, jedná se o JavaScript a je ho tak možno pouštět pouze uvnitř HTML stránky. Inicializačním prvkem je v tomto případě dvojice elementů `<script>` v takovéto podobě:

```
  <script type="text/javascript" language="javascript" ▶
src="../Saxonce.nocache.js">
</script> <script type="application/xslt+xml" language="xslt2.0" ▶
src="books.xsl" data-source="books.xml"></script>
```

První část jednoduše připojuje kód Saxonu CE. Soubor Saxonce.nocache.js je vstupní modul, který volá jeden z větších modulů obsahujících zkompilovaný XSLT procesor např. na základě toho, v jakém prohlížeči chceme výsledky zobrazit.

Druhá část pak definuje polohu použitého stylu a zdrojový XML dokument. Svojí funkcí připomíná funkci procesní instrukce pro připojení externího stylu s jedním rozdílem. V případě externího stylu definujeme pouze polohu stylu, protože zdrojovým dokumentem je nám samotný dokument obsahující procesní instrukci. Výsledkem transformace je pak kompletní HMTL stránka. V případě Saxonu CE určujeme polohu jak stylu, tak zdrojového XML dokumentu. XSLT 2.0 a Saxon CE totiž poskytují efektivnější způsob, jak generovat HTML obsah. Výše uvedený skript je vložen do HTML stránky obsahující strukturu budoucího výsledku (např. tagy `<div>` s identifikátory). Transformace pak vyplní jednotlivé sekce stránky fragmenty HTML obsahující data ze zdrojového XML dokumentu. Celý proces je znázorněn na Obrázku 4.4.



Obrázek 4.4. Proces zpracování XML dokumentu pomocí Saxon CE

Napojení do předpřipravené HTML struktury pak neprobíhá pomocí JavaScriptu, jako v případě JS API, kde je výsledek transformace také připojován k již existujícímu HTML, ale pomocí XSLT instrukce `<xsl:result-document>` přímo uvnitř stylu. Tato se používá zejména v případě, kdy je potřeba provádět transformaci s více výstupy, každá instrukce `<xsl:result-document>` produkuje samostatný strom jako výsledek transformace a je schopna kontrolovat umístění tohoto výsledného dokumentu. K tomu používá atribut `href`, jehož hodnotou je URI identifikátor umístění, kam má být výsledek zapsán. Může to být samozřejmě umístění na disku, ale také fragment HTML dokumentu, do kterého chceme výsledek zapsat. Druhým atributem, který při konstrukci HTML pomocí Saxonu CE používáme, je atribut `method`. Ten standardně slouží k určení výstupní serializace (XML, HTML, text atd.) a přepisuje pro konkrétní výsledný strom hodnotu atributu `output` (platnou pro celou transformaci). Pomocí rozšíření Interactive XSL (podrobně o něm později) obsažené v Saxonu CE můžeme v atributu method také použít hodnoty `ixsl:replace-content` a `ixsl:append-content`, které kontrolují zda výsledný strom nahradí obsah nacházející se na daném URI, nebo bude k tomuto obsahu pouze připojen.[3] Stavba dokumentu pak vypadá následovně:

```
    <xsl:result-document href="id1" method="ixsl:replace-content">
     <xsl:text>Hello</xsl:text>
</xsl:result-document>
<xsl:result-document href="id2" method="ixsl:replace-content">
```

```
    <xsl:text>World!</xsl:text>
</xsl:result-document>
```

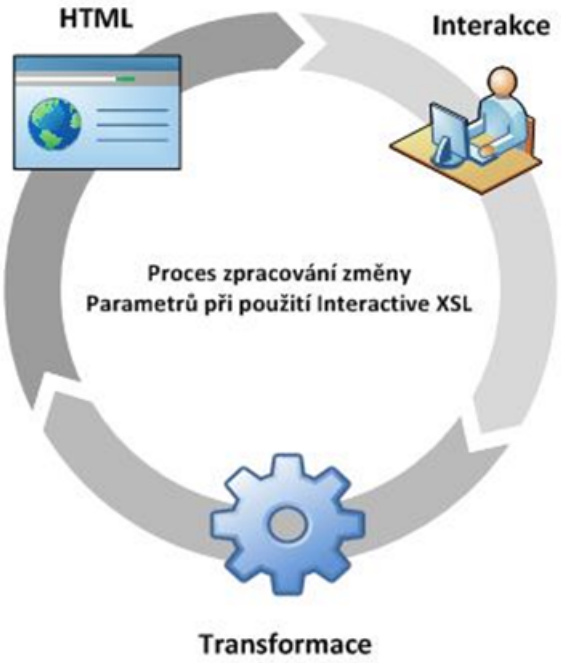
Tento styl vytvoří dva výsledné dokumenty, které připojí do HTML struktury na příslušné místo určené v atributu href. Takovým způsobem, že původní obsah (existoval-li nějaký) je přepsán. Výsledné HTML může vypadat např. takto:

```
   <div id="id1">Hello</div>
<div id="id2">World!</div>
```

Práce s parametry se při použití Saxonu CE jako XSLT procesoru významně liší. Stejně jako se liší běžné metody (XSLT procesor a JS API) a metody Saxonu CE pro tvorbu výsledného dokumentu, stejně tak pracuje Saxon CE odlišně i s interakcí uživatele.

Při použití stylu, jenž má pracovat s uživatelskou interakcí, se aplikace používající JS API řídí následujícím principem. JavaScript detekuje uživatelskou interakci (typicky se jedná např. o změnu formulářového pole). Na základě nové hodnoty formulářového pole (či jiného uživatelského vstupu) určí JavaScript novou hodnotu parametru, se kterou opětovně zavolá transformaci a prezentuje uživateli pozměněnou HTML stránku. Celý princip je zobrazen na obrázku 4.5. Saxon CE však umožňuje tvořit styly, které zachycují a vyhodnocují uživatelské interakce bez nutnosti psát pro tuto funkcionalitu JavaScriptovou obálku.[2] Tato funkcionalita je realizována pomocí rozšíření Interactive XSLT (obrázek 4.6).

Obrázek 4.5. Proces zpracování změny parametrů při použití JS API



Obrázek 4.6. Proces zpracování změny parametrů při použití iXSL

Toto rozšíření přináší několik funkcí a instrukcí, které umožňují stylu v reálném čase reagovat na uživatelskou interakci. Nachází se ve jmenném prostoru http://saxonica.com/ns/interactiveXSLT a běžně se pro jeho identifikaci používá prefix ixsl (je samozřejmě možno použít i jiný).

Princip použití Interactive XSL závisí na použití atributu `mode` v instrukci `<xsl:template>`. Pomocí toho atributu je možno definovat větší množství šablon zpracovávajících stejný uzel vstupního XML dokumentu různými způsoby. Běžně jsou jednotlivé šablony volány v závislosti na preferenci v instrukci `<xsl:apply-templates>` obsažené právě v atributu `mode`. Interactive XSL do tohoto principu přináší změnu. Šablony lze volat nejen klasicky pomocí příslušné instrukce, ale i po uživatelské interakci na daném elementu. Pokud se typ interakce shoduje s typem interakce zapsaným v atributu `mode` šablony, daná šablona se aplikuje. Na následujícím příkladu vidíme šablonu, která je aplikována při kliknutí na element `<div>`.

```
<xsl:template match="div" mode="ixsl:onclick">
```

Interactive XSL dále definuje řadu funkcí a instrukcí, které umožňují práci s některými prvky objektového modelu HTML, s JS objekty a s externě definovanými JS funkcemi.

Funkce `ixsl:page()` vrací dokumentový uzel HTML DOM dokumentu. Jedná se o nejsvrchnější uzel ve stromu, jenž nemá žádného předka a jeho potomky jsou všechny ostatní uzly stromu (pozor, nejedná se o kořenový element). Tímto způsobem se tak například můžeme dostat k elementu title HTML dokumentu, ve kterém styl invokujeme. Například pomocí následujícího XPath výrazu:

```
ixsl:page()/html/head/title
```

Funkce `ixsl:get($objekt, $vlastnost)` vrací vlastnost javascriptového objektu, jenž je identifikován prvním argumentem. Druhý argument definuje název požadované vlastnosti. Konkrétně k pseudo-atributům (vlastnostem) elementů v HTML DOM objektu lze přistupovat i jednoduší cestou a lze se na ně přímo dotazovat XPath výrazem s tím, že tyto vlastnosti jsou vázány ve jmenném prostoru `http://saxonica.com/ns/html-property`, který je běžně reprezentován prefixem `prop`.

Funkce `ixsl:call($objekt, $funkce, $argumenty...)`, volá JavaScriptovou funkci náležící danému objektu s poskytnutými argumenty. Výsledek, jenž je vrácen touto funkcí, je vrácen jako výsledek volání `ixsl:call`. Pokud je funkce globální, je ji možno volat i přímo zevnitř XPath výrazu. Například definujeme-li funkci:

```
<script type="text/javascript" language="javascript">
function obsahkruhu(r) { return Math.PI*String(x*x) }
</script>
```

Můžeme ji zavolat zevnitř XPath výrazu například takto:

```
js:obsahkruhu(5)
```

Výsledek volané funkce je předán jako výsledek XPath výrazu. Prefix `js` (nebo jiný použitý pro identifikaci globální JS funkce) musí být vázán na jmenný prostor `http://saxonica.com/ns/globalJS`.

Pokud je potřeba jako argument předat těmto funkcím aktuální objekt window, je možno zavolat funkci `isxl:window()`, která tento objekt vrací jako svůj výsledek. Obdobně je-li potřeba manipulovat s objektem event aktuální uživatelské interakce, je nutno zavolat funkci `ixsl:event()`, která jako svůj výsledek vrátí tento objekt.

Konečně funkce `ixsl:eval($string)` provede exekuci JavaScriptu, jenž je poskytnut této funkci ve formě textového řetězce jako argument.[2]

Interactive XSL pak dále definuje dvě nové XSLT instrukce. Instrukce `<ixsl:set-attribute>` slouží ke změně hodnot atributů HTML elementu. Atribut `name` definuje jméno atributu, jenž má být změněn. Atribut `select` je XPath výraz, jehož výsledek bude novou hodnotou atributu.

Druhá instrukce je `<ixsl:scheduled-action>`. Ta se dá použít v případě, že chceme pozdržet zpracování. Atribut `wait` obsahuje XPath výraz, jehož hodnota musí být typu Integer a je to doba v milisekundách, která uběhne předtím, než je tělo instrukce zpracováno. Obsahem této instrukce může být pouze jedna instrukce `<xsl:call-tepmlate>`, která se po vypršdení daného času zpracuje[3].

[2] V současné release verzi zatím nefunkční.

Kapitola 5

Implementace XSLT a XPath

Při porovnání možností současných prohlížečů je třeba vzít v úvahu úroveň implementace XSLT 1.0 a XPath 1.0. Toto srovnání je důležité zejména z hlediska toho, jak se implementace XSLT v jednotlivých prohlížečích vyvíjela. Jak už bylo nastíněno v části 2.2, tento proces neprobíhal od roku 1999, kdy bylo XSLT 1.0 přijato jako standard, na straně prohlížečů rychle a stejně jako u ostatních webových standardů docházelo i u XSLT k chybným implementacím některých částí, resp. k úplnému zanedbání implementace. To samozřejmě vede k nestandardnímu chování nativního procesoru v prohlížeči v momentě, kdy na takovou situaci narazí.

Následující údaje byly získány z on-line dokumentací a bug-reporting systémů jednotlivých prohlížečů nebo XSLT procesorů, které jsou implementovány jako nativní.

5.1. Firefox

Vykreslovací jádro Gecko podporuje plně všechny XSLT elementy, funkce a XPathové osy mimo následujících, které jsou podporovány jen částečně nebo vůbec.

5.1.1. Element xsl:fallback

Určuje k jakému zpracování má dojít, pokud nadřazený element ve stylu není v použitém procesoru, nebo verzi XSLT implementován. Používá se zejména k zachování kompatibility mezi verzemi XSLT, nebo tam, kde je potřeba užít rozšiřujících funkcí, které však nemusí být zahrnuty ve všech procesorech a je proto nutné takové situace ošetřit. Tento element není podporován[4].

5.1.2. Element xsl:namespace-alias

Definuje dočasný alias pro prefix jmenného prostoru v atributu `stylesheet-prefix` . Ve výstupním dokumentu jsou pak všechny výskyty tohoto prefixu nahrazeny prefixem obsaženým v atributu `result-prefix`. Používá se zejména v případě, kdy chceme z jednoho stylu generovat styl jiný. Tento element není podporován[4].

5.1.3. Element xsl:number

Je podporován pouze částečně. Tento element se používá buď pro formátování čísel, nebo pro zjištění sekvenčního pořadí elementu v dokumentu. Atribut `count` XPathovým výrazem speci-

fikuje, které elementy se mají do sekvence zahrnout. Atribut `from` stejným způsobem specifikuje element, od kterého má sekvence začínat. Nakonec atribut `level` určuje jak bude vygenerované číslo odrážet hiearchii dokumentu. Hodnota `single` (která je výchozí) počítá veškeré předešlé sourozence elementu získaného z atributu select. Hodnota `multiple` určuje, že v pořadovém čísle se bude odrážet struktura dokumentu. Často se takto vytvářejí hierarchická čísla kapitol, oddílů aj. Poslední možná hodnota je hodnota `any`, která do úvahy bere všechny elementy vybrané výrazem v atributu `count` bez ohledu na jejich hiearchii. Tato hodnota není ve Firefoxu podporována. Dále je možné měnit formát vygenerovaného číslování. Pokud je použit formát číslování obsahující písmena, atribut `lang` určuje jazyk, jehož abeceda se použije. Tento atribut též není ve Firefoxu podporován[4].

5.1.4. Element xsl:output

Je částečně podporován. Tento element ovlivňuje vlastnosti výstupního dokumentu, jeho atributy se odvíjejí od typu výstupního dokumentu (XML, XHTML, nebo text). Firefox nepodporuje atributy `standalone`, `media` a `indent`. Atribut `standalone` je směrodatný pouze u XML výstupu a deklaruje, zdali je dokument standalone. Tedy zda je závislý na externě definovaném DTD schématu nebo ne. Atribut `media` definuje MIME typ výstupního dokumentu a nakonec atribut `indent` zdali má být výstupní dokument odsazován podle hiearchie prvků v dokumentu. Což vzhledem k faktu, že Firefox výstupní dokument neserealizuje, nepředstavuje vážný problém[4].

5.1.5. Element xsl:stylesheet

Tento element není plně podporován. Atribut extension-element-prefixes, který definuje prefixy pro elementy v tomto stylu, jež jsou součástí nějakého rozšíření, není implementován[4].

5.1.6. Element xsl:value-of a element xsl:text

Oba tyto elementy jsou podporovány částečně. V obou případech není podporován atribut `disable-output-escaping`, který ovlivňuje, zda budou ve výstupu escapovány zvláštní znaky. Například pokud bude element `<xsl:text>` zapisovat do výsledku řetězec obsahující znak „<“ při volbě `disable-output-escaping=no` bude výsledkem escapovaná hodnota „<“, naopak při volbě `disable-output-escaping=yes` zůstane „<“ v neescapované podobě. Opět je potřeba zdůraznit, že Firefox výstup neserealizuje a absence tohoto atributu tak není v klientském zpracování relevantní[4].

5.1.7. XPath osa namespace

Osa namespace slouží k vybrání všech namespace uzlů kontextového elementu. Tato osa není XPathovou implementací Firefoxu podporována[4].

5.1.8. Funkce unparsed-entity-uri

Tato funkce vrací jako svůj výsledek URI neparsované entity definované v DTD deklaraci dokumentu. Jako argument je této funkci předáno jméno entity, jejíž URI se má vrátit. Tato funkce není podporovaná[4]..

5.2. Opera

Vykreslovací jádro Presto plně podporuje všechny XSLT elementy, XPathové osy a funkce mimo elementu `<xsl:namespace-alias>` (viz 5.1.2)[6].

5.3. Internet Explorer

Knihovna MSXML 6.0, která se v Internet Exploreru o XSLT transfromace stará, je podle oficiální dokumentace MSDN úplnou implementací XSLT 1.0 a XPath 1.0[5].

5.4. Google Chrome a Safari

Knihovna libxslt, pomocí níž provádí Google Chrome a Safari XSLT transformace, je podle její dokumentace úplnou implementací XSLT 1.0[7].

5.5. XSLT 2.0

V současné době je jedinou možností, jak používat XSLT 2.0 transformace na klientské straně použití Saxonu CE (ten je úplnou implementací XSLT 2.0 procesoru [3]). Z nativních procesorů, žádný nemá podporu XSLT 2.0. Což je ostatně jeden z důvodů proč byl Saxon CE vytvořen.[2]

5.6. Funkce document()

Přestože je funkce `document()`, implementována ve všech nativních procesorech, rozhodl jsem se jí věnovat zvláštní sekci. Důvodem jsou bezpečnostní politiky jednotlivých prohlížečů ohledně získávání obsahu z jiné domény, než na které se nachází aktuální stránka. Žádný z testovaných prohlížečů neumožní funkci `document()` nahrát obsah z cizí domény, nicméně chování se při takovéto události liší. Zatímco Firefox, Opera, Chrome a Safari jednoduše funkci `document()`, která požaduje obsah z cizí domény, ignorují (nicméně neignorují instrukci, která tuto funkci obsahuje). Internet Explorer styl, který obsahuje funkci `document()` volající obsah z jiné domény, prostě a jednoduše nebere v potaz a na výstup vypíše pouze textové řetězce ze zdrojového dokumentu.

Kapitola 6

Benchmarking výkonu

Otázka výkonu je v porovnání prohlížečů resp. nativních implementací XSLT procesorů velmi důležitá. Jak už bylo řečeno v části 2.2 XSLT, transformace jsou operace nad stromy, které jsou ze své povahy náročné na operační paměť a procesorový výkon. Proto je důležité, aby prohlížeč disponoval dostatečně výkonným procesorem, který umožní provádět transformace alespoň tak, aby splnil minimální požadavky uživatelů na odezvu.

V závislosti na typu aplikace, která XSLT transformace na klientské straně používá a na typu obsahu, který daný styl prezentuje, je vhodné určit max. dobu odezvy. Například pokud bude zvažovaná aplikace běžet ve firemním intranetu, kde bude sloužit ke generování rozsáhlých reportů např. o stavu skladu. Nebude zřejmě velkým problémem, pokud bude transfromace trvat několik sekund (samozřejmě snaha o snížení doby odezvy by měla být stále přítomna). Naopak bude-li zvažovaná aplikace, resp. styl generovat obsah webové prezentac,e nebude doba odezvy v jednotkách sekund pro uživatele pravděpodobně pohodlná a je dokonce možné, že pro něj bude nepřijatelná a web opustí.[1] Maximální dobu odezvy je tak potřeba promítnout i do designu stylu, který by v každém případě měl být navržen tak, aby ho bylo možno procesovat s využitím co nejmenších prostředků za co nejkratší čas.

6.1. Metodika měření

Měření výkonu jednotlivých XSLT procesorů bude prováděno pomocí JavaScriptu a javascriptového API prohlížeče pro ovládání XSLT transformací. Faktorem, jenž bude měřit výkon XSLT procesoru je samozřejmě čas. Konkrétně čas v milisekundách, který uběhne od začátku měřené události do jejího konce.

U každého testovacího stylu se bude samostatně měřit doba transformace, doba získání a parsování XML dokumentu a doba získání a parsování stylu. Směrodatná pro tento test je pak doba trvání všech těchto částí. Protože ačkoliv není parsování XML záležitostí XSLT procesoru (viz 2.2), bude ve většině případů při klientském zpracování transformaci předcházet. Z toho plyne, že cílem testu není čistě technicky pouze jedna část procesu (např. XSLT transformace), ale spíše modelová situace, ke které běžně dochází při klientském zpracování.

Každý XSLT procesor bude v tomto benchmarku zpracovávat řadu testovacích stylů. Tyto styly jsou navrženy tak, aby testovaly specifickou funkcionalitu XSLT. Ke každému stylu pak bude přidán odpovídající XML dokument jako zdroj dat pro transformaci. Samotná transformace

[1] Jako všobecně přijatelný čas, po který uživatel ještě vnímá odezvu jako okamžitou, bývá uváděna jedna desetina sekundy.[15]

(stejně jako parsing XML a XSL) pak bude probíhat u každého testovacího stylu v desítkách iteracích (konkrétně bude upřesněno u výsledků testů), hodnoty jednotlivých měření budou zprůměrovány, pro lepší vypovídací hodnotu testu. Ke každému testovanému stylu taky bude uveden rozsah vstupních dat, či vstupní proměnné, závisle na tom zdali daný styl pracuje s předpřiravenými daty, nebo generuje data sám.

6.2. Testovací styly

U tvorby testovacích stylů byl kladen důraz zejména na variabilitu tak, aby zpracování jednotlivých stylů prověřilo různé aspekty testovaného XSLT procesoru. Následuje popis jednotlivých stylů s krátkou anotací stylu, popisem testovacích dat a popisem funkcionality procesoru (či modelové situace), kterou by měl daný styl prověřit.

6.2.1. Styl bottles.xsl

Tento styl generuje text anglické písně "99 Bottles of Beer", resp. generuje tuto píseň o určitém počtu slok. Vstupními daty je soubor bottles.xml, obsahující v elementu <bottles> počet veršů, který má styl vygenerovat. Tento styl prověřuje schopnost procesoru zpracovávat rekurzivní styly a generovat data na výstup.

6.2.2. Styl reverse.xsl

Tento styl obrací pořadí prvků v souboru vstupních dat. Tím je soubor two_gent.xml, jedná se o přepis Shakespearovy komedie "Dva kavalíři z Verony" do XML podoby. Při zpracování tohoto stylu je obráceno pořadí veršů v každém výstupu jednotlivých postav. Tento styl se opět zaměřuje na rekurzi.

6.2.3. Styl sort.xsl

Tento styl jednoduše řadí elementy zpracovávaného XML dokumentu do abecedního pořadí. Vstupními daty je v tomto případě seznam fiktivních osob a jejich osobních informací nacházející se v souobru 1000rows.xml. Elementy jednotlivých záznamů jsou řazeny nejprve podle křestního jména a poté podle příjmení. Data byla vygenerována pomocí programu DataGen 2.1. Styl má prověřit rychlost třídících algoritmů v XSLT procesorech.

6.2.4. Styl lookup.xsl

Styl lookup.xsl pouští na zdrojovými daty několik XPath výrazů, které vrací od jednoho po několik stovek záznamů. Vstupními daty je opět soubor 1000rows.xml. Tento styl má prověřit rychlost XSLT procesorů při prohledávání širokého stromu.

6.2.5. Styl hanoi.xsl

Tento styl řeší úlohu Hanojských věží se zadaným počtem kotoučů. Vstupním souborem je soubor hanoi.xml, který má v atributu size elementu <hanoi> počet kotoučů, s nimiž bude úloha

řešena. Výstupem této transformace je seznam tahů vedoucích k vyřešení úlohy. Tento styl je opět zaměřen na rekurzi.

6.2.6. Styl math.xsl

Styl math.xsl vypočte požadovaný faktoriál a odhad čísla π pomocí Liebnitzovi řady. Vstupním souborem je soubor math.xml, ve kterém je jednak hodnota faktoriálu, který má bých vypočten a také počet iterací výpočtu Liebnitzovi řady. Tento styl se zaměřuje na rychlost početních operací a rekurzi[14].

6.2.7. Styl string.xsl

Tento styl se zaměřuje na operace s řetězci a prověřuje rychlost XSLT procesoru v této oblasti. Vstupní dokumentem je soubor rich_iii.xml, což je přepis Shakespearovy hry Richard III. v XML podobě. Při použití tohoto stylu je manipulováno s řetězci různým způsobem zejména za použití funkcí translate(), contains() a normalize-space() (např. všechny dialogy vévody z Clarence jsou převedeny do leetspeaku).

6.3. Testovací prostředí

Všechny testy byli prováděny na stejném testovacím prostředí s parametry.[2]

- Operační systém: Microsoft Windows 7 Home Premium (version 6.1.7600; build 7600)
- Procesor: Intel(R) Core(TM) i3 CPU U 380 @ 1.33GHz (architecture: x64; 1333 MHz)
- Paměť: 4.0 GB

6.4. Výsledky benchmarku

V následující sekci jsou uvedeny výsledky benchmarku pro jednotlivé prohlížeče. V každé tabulce je nejprve uveden název testu. Poté číslo, jež uvádí počet iterací daného testu. Následuje průměrný čas zpracování v milisekundách. A nakonec minimum a maximum z jednotlivých časů zpracování též v milisekundách. V další sekci se nachází komentář k výsledkům benchmarku a souhrný graf, který ukazuje průměrné časy jednotlivých prohlížečů při zpracování všech testovacích scénářů. Na závěr je uvedena tabulka a graf ukazující nejlepší a nejhorší hodnoty pro jednotlivé testovací scénáře a rozdíly mezi nimi.

Z dílčích i souhrných výsledků jsem se též rozhodl vyřadit hodnoty časů parsování jednotlivých stylů. Nejedná se o velké soubory, a tak parsování zabralo v nejhorším případě ve všech prohlížečích 5ms. Vzhledem k absenci jakýchkoliv rozdílů mezi prohlížeči tedy nepovažuji tyto hodnoty za příliš směrodatné.

Pro správné interpretování výsledků je potřeba ještě uvést u některých testovacích stylů rozsah vstupních proměnných, které byli použity pro vygenerování výsledků uvedených níže:

- Styl bottles.xsl generuje text písně o 999 slokách.

[2] Údaje byli získány za pomocí programu Free PC Audit 2.1

- Styl hanoi.xsl generuje řešení pro problém Hanojských věží s počtem kotoučů 10.

- Styl math.xsl vypočítá faktoriál 28 a Liebnitzovu řadu o 2500 iteracích[3] (odhadne tak číslo π na 2 desetiná místa).

6.4.1. Firefox

Tabulka 6.1. Výsledky benchmarku - XSLT transformace - Firefox

Test	Iterací	Průměrně (ms)	Minimum (ms)	Maximum (ms)
bottles	50	16.7	15	22
reverse	50	44.74	41	55
sort	50	48.02	46	55
lookup	50	29.36	27	35
hanoi	50	12.82	11	17
math	50	6.06	5	7
string	50	124.74	119	158

Tabulka 6.2. Výsledky benchmarku - XML parsing - Firefox

Test	Iterací	Průměrně (ms)	Minimum (ms)	Maximum (ms)
bottles.xml	50	2,58	2	3
two_gent.xml	50	36,08	34	63
rows1000.xml	50	36,04	34	62
hanoi.xml	50	2,64	2	25
math.xml	50	1,54	1	2
rich_iii.xml	50	36,28	33	66

6.4.2. Opera

Tabulka 6.3. Výsledky benchmarku - XSLT transformace - Opera

Test	Iterací	Průměrně (ms)	Minimum (ms)	Maximum (ms)
bottles	50	91.6	89	101
reverse	50	74.28	72	87
sort	50	46.2	44	50
lookup	50	17.14	16	21

[3]Původně byl počet iterací stanoven na 40000 nicméně, kromě Firefoxu všechny prohlížeče na takovémto počtu iterací selhali. Počet tak musel být snížen.

Test	Iterací	Průměrně (ms)	Minimum (ms)	Maximum (ms)
hanoi	50	56.92	55	68
math	50	17.86	17	25
string	50	1137.34	1089	1240

Tabulka 6.4. Výsledky benchmarku - XML parsing - Opera

Test	Iterací	Průměrně (ms)	Minimum (ms)	Maximum (ms)
bottles.xml	50	0.78	0	4
two_gent.xml	50	31.66	30	41
rows1000.xml	50	40.68	38	49
hanoi.xml	50	0.66	0	2
math.xml	50	0.64	0	1
rich_iii.xml	50	48.98	44	65

6.4.3. Google Chrome

Tabulka 6.5. Výsledky benchmarku - XSLT transformace - Google Chrome

Test	Iterací	Průměrně (ms)	Minimum (ms)	Maximum (ms)
bottles	50	46.62	42	63
reverse	50	97.02	90	124
sort	50	76.28	71	94
lookup	50	52.5	47	73
hanoi	50	31.16	25	45
math	50	7.6	6	12
string	50	1072.98	1014	1201

Tabulka 6.6. Výsledky benchmarku - XML parsing - Google Chrome

Test	Iterací	Průměrně (ms)	Minimum (ms)	Maximum (ms)
bottles.xml	50	2.14	1	3
two_gent.xml	50	37.72	32	50
rows1000.xml	50	48.1	43	55
hanoi.xml	50	2.02	1	3
math.xml	50	1.3	1	2
rich_iii.xml	50	58.06	53	67

6.4.4. Safari

Tabulka 6.7. Výsledky benchmarku - XSLT transformace - Safari

Test	Iterací	Průměrně (ms)	Minimum (ms)	Maximum (ms)
bottles	50	51.48	48	67
reverse	50	91.2	84	113
sort	50	61.66	59	71
lookup	50	40.74	38	48
hanoi	50	29.42	28	32
math	50	8.34	8	9
string	50	964.02	941	1000

Tabulka 6.8. Výsledky benchmarku - XML parsing - Safari

Test	Iterací	Průměrně (ms)	Minimum (ms)	Maximum (ms)
bottles.xml	50	1.44	1	3
two_gent.xml	50	27.02	22	74
rows1000.xml	50	33.66	29	76
hanoi.xml	50	1.46	1	2
math.xml	50	1.48	1	2
rich_iii.xml	50	43.84	37	9

6.4.5. Internet Explorer

Tabulka 6.9. Výsledky benchmarku - XSLT transformace - Internet Explorer

Test	Iterací	Průměrně (ms)	Minimum (ms)	Maximum (ms)
bottles	50	108.9	105	115
reverse	50	25.72	24	29
sort	50	27.54	25	49
lookup	50	163.5	158	171
hanoi	50	16.88	15	24
math	50	20.7	19	24
string	50	81.46	76	91

Tabulka 6.10. Výsledky benchmarku - XSLT transformace - Internet Explorer

Test	Iterací	Průměrně (ms)	Minimum (ms)	Maximum (ms)
bottles.xml	50	0.88	0	1
two_gent.xml	50	16.94	15	22
rows1000.xml	50	42.2	20	1042
hanoi.xml	50	0.88	0	1
math.xml	50	0.9	0	1
rich_iii.xml	50	25.9	24	28

6.4.6. Saxon CE

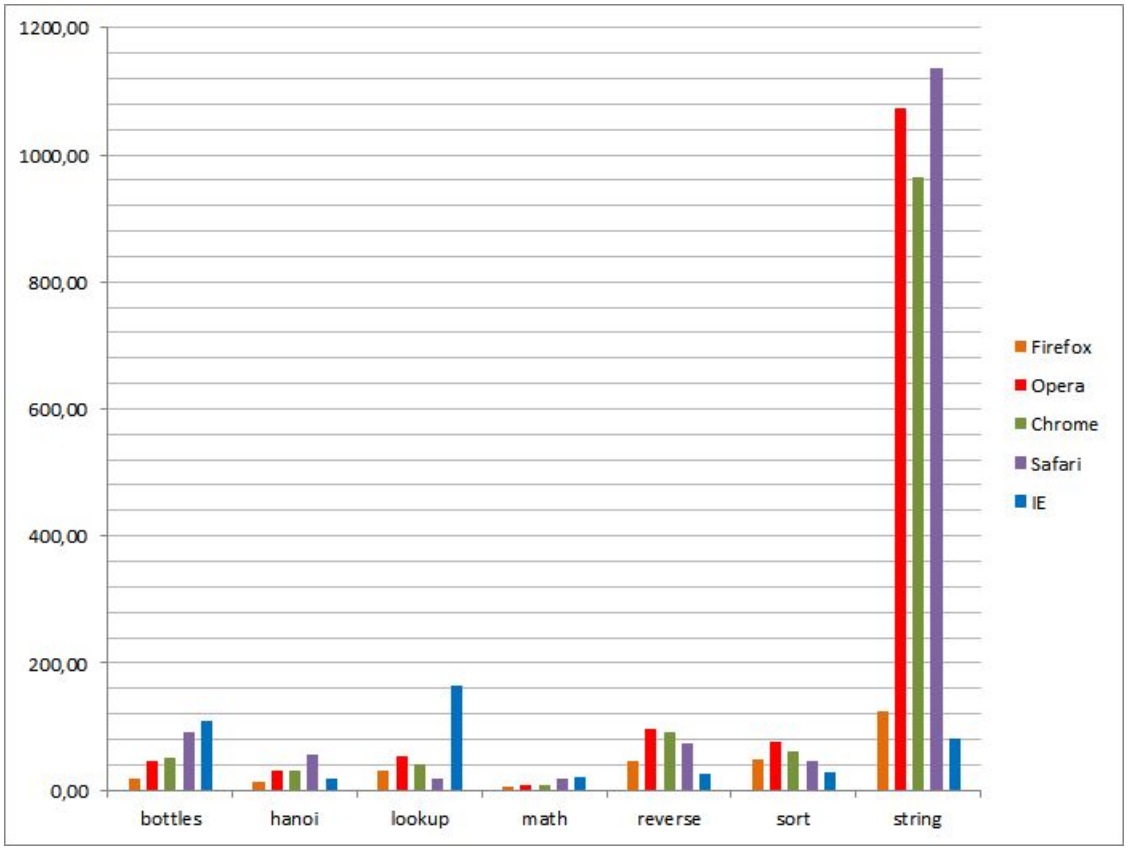
V současné verzi neobsahuje Saxon CE javascriptové API (pro připojení stylu se používá pouze element <script> viz 4.3.4). Což představuje pro takto navržený benchmark problém, protože transformaci není možno dostatečnm způsobem kontrolovat. Z tohoto důvodu nebyl u Saxonu CE benchmark proveden.

6.4.7. Souhrn - XSLT transformace

Na následujícím grafu a tabulce jsou znázorněny průměrné výsledky jichž dosáhli prohlížeče na jednotlivých testech.

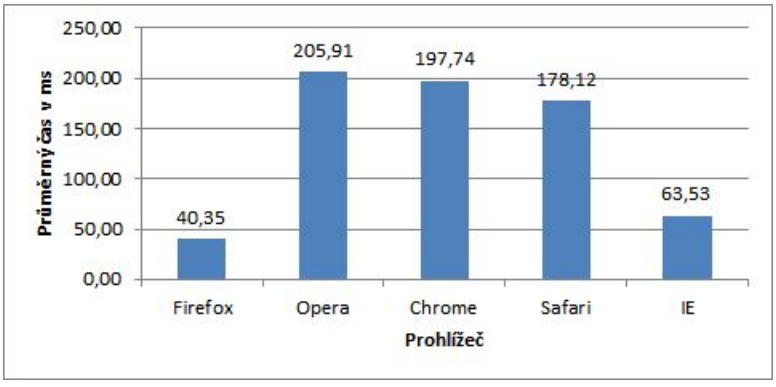
Tabulka 6.11. Průměrné výsledky prohlížečů na jednotlivých testech

Test	Firefox	Opera	Chrome	Safari	IE
bottles	16,70	46,62	51,48	91,60	108,90
hanoi	12,82	31,16	29,42	56,92	16,88
lookup	29,36	52,50	40,74	17,14	163,50
math	6,06	7,60	8,34	17,86	20,70
reverse	44,74	97,02	91,20	74,28	25,72
sort	48,02	76,28	61,66	46,20	27,54
string	124,74	1072,98	964,02	1137,34	81,46



Obrázek 6.1. Graf průměrných výsledků prohlížečů na jednotlivých testech.

Průměrné časy ke zpracování všech testovacích scénářů (tedy průměry průměrů dosažených dílčích časů při 50 iteracích) jsou znázorněny v následujícím grafu. Na nejlepší hodnotu dosáhl Firefox. Zpracování všech testovacích scénářů mu průměrně zabralo 40,35 milisekundy. Nejhůře pak dopadla Opera, jejíž souhrný výsledek činí 205,91 milisekundy. K tomuto číslu se také blíží Chrome a Safari.



Obrázek 6.2. Graf průměrného času ke zpracování všech testovacích scénářů.

Následující tabulka ukazuje nejlepší a nejhorší prohlížeč (a odpovídající čas v milisekundách) pro jednotlivé testovací scénáře a rozdíl mezi oběma hodnotami.

Tabulka 6.12. Nejlepší a nejhorší výsledky pro jednotlivé testy

	Nejlepší		Nejhorší		
Test	**Prohlížeč**	**Průměrný čas (ms)**	**Prohlížeč**	**Průměrný čas (ms)**	**Rozdíl (ms)**
bottles	Firefox	16,70	IE	108,90	92,20
hanoi	Firefox	12,82	Opera	56,92	44,10
lookup	Opera	17,14	IE	163,50	146,36
math	Firefox	6,06	IE	20,70	14,64
reverse	IE	25,72	Chrome	97,02	71,30
sort	IE	27,54	Chrome	76,28	48,74
string	IE	81,46	Opera	1137,34	1055,88

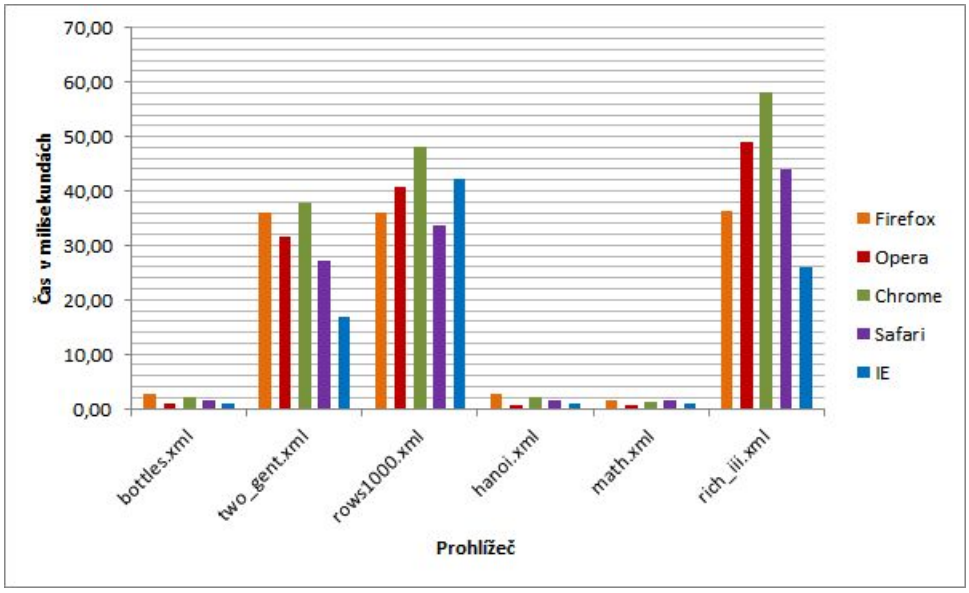
Z uvedených výsledků je patrné, že výkony jednotlivé prohlížečů se významně liší v závislosti testovacím scénáři. Tedy, že nativní XSLT procesory v testovaných prohlížečích jsou silné ve zpracovávání některých úloh, zatímco v jiných naopak zaostávají. Nicméně z celkového hlediska je možno pozorovat, že nejrychlejším prohlížečem je Firefox. Průměrný čas trojice prohlížečů blížícím se nebo překračujícím 200 ms (Chrome, Safari, Opera) je pak zcela jasně ovlivně zejména průměrným časem z testovacího scénáře string, který se pohybuje okolo desetinásobku času, jenž k tomuto testu potřeboval Firefox a Internet Explorer.

6.4.8. Souhrn - XML parsing

V následujícím grafu a tabulce jsou znázorněny průměrné výsledky, jichž dosáhli prohlížeče při parsování jednotlivých XML dokumentů.

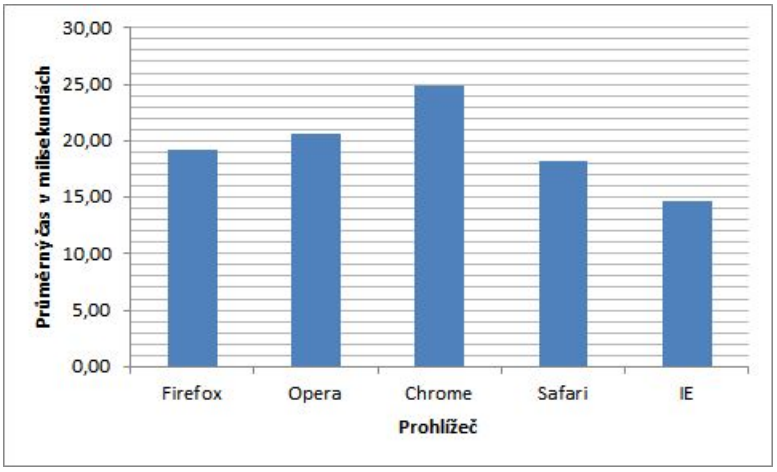
Tabulka 6.13. Průměrný čas parsování jednotlivých XML dokumentů.

Dokument	**Firefox**	**Opera**	**Chrome**	**Safari**	**IE**
bottles.xml	2,58	0,78	2,14	1,44	0,88
two_gent.xml	36,08	31,66	37,72	27,20	16,94
rows1000.xml	36,04	40,68	48,1	33,66	42,2
hanoi.xml	2,64	0,66	2,10	1,46	0,88
math.xml	1,54	0,64	1,30	1,48	0,9
rich_iii.xml	36,28	48,98	58,06	43,84	25,900



Obrázek 6.3. Graf průměrného času parsování jednotlivých XML dokumentů.

Na následujícím grafu jsou vidět celkové průměrné výsledky parsování všech XML dokumentu obsažených v benchmarku. Nejpomalejší byl Chrome a nejrychlejší Internet Explorer, nicméně rozdíly mezi prohlížeči nejsou natolik zásadní jako v případě XSLT transformací.



Obrázek 6.4. Graf průměrného času ke zpracování všech XML dokumentů.

Následující tabulka uvádí nejlepší a nejhorší časy při parsování XML dokumentů a rozdíly mezi nimi. Jak už bylo řečeno u předchozího grafu, rozdíly v celkových časech nejsou veliké a tudíž ani dílčí časy, resp. rozdíly mezi nejlepším a nejhorším časem, nejsou tak signifikantní jako v případě benchmarku XSLT transformací.

Tabulka 6.14. Nejlepší a nejhorší výsledky pro jednotlivé testy

	Nejlepší	Nejhorší			
Test	**Prohlížeč**	**Průměrný čas (ms)**	**Prohlížeč**	**Průměrný čas (ms)**	**Rozdíl (ms)**
bottles.xml	Opera	0,78	Firefox	2,58	1,80
hanoi.xml	Opera	0,66	Firefox	2,64	1,98
math.xml	Opera	0,64	Firefox	1,54	0,90
rich_iii.xml	IE	25,90	Chrome	58,06	32,16
rows1000.xml	Safari	33,66	Chrome	48,10	14,44
two_gent.xml	IE	16,94	Chrome	37,72	20,78

Kapitola 7

Vzorová aplikace

Vzorová aplikace byla navržena za účelem demonstrovat možnosti využití XSLT Transformací pro prezentaci dat na klientské straně v prohlížečích Firefox, Chrome a Safari. V ostatních prohlížečích nefunguje správně nebo vůbec, při návrhu aplikace jsem se soustředil zejména na možnosti XSLT místo na úplnou kompatibilitu. Aplikace zobrazuje v tabulce seznam knih ze zdrojového souboru `books.xml`, kde každý řádek tabulky reprezentuje jednu knihu (položku) z tohoto zdrojového souboru a zobrazuje autora, název a cenu publikace. Zobrazení je řízeno jednak stylem `booklist.xsl` a jednak JavaScriptem v souboru `index.htm`, kde se také nachází HTML skelet aplikace. Navržené řešení by mělo být schopno třech základních funkcionalit:

- Po kliknutí na název sloupce by měla aplikace seřadit položky podle hodnot tohoto sloupce, a to v opačném pořadí, než jak jsou položky aktuálně seřazeny (jsou-li řazeny).
- Po kliknutí na ikonu "informace" na konci každého řádku, by měla aplikace rozbalit pod touto položkou box obsahující náhled obalu publikace a krátkou anotaci. Po opětovném kliknutí by měl box zmizet.
- Formulářem ve vrchní části stránky by mělo být možno filtrovat zobrazené výsledky podle žánru.

7.1. Zdroj dat

Jak už bylo řečeno, zdrojem dat pro aplikaci je XML dokument `books.xml`. Údaje o jednotlivých publikacích jsou uchovávány v elementu `<book>`. Příklad jednoho záznamu spolu s popisem obsahu elementů, které jsou obsaženy v elementu `<book>`, je možné vidět níže:

```
<book>
    <title>Název publikace</title>
    <cover href="URL obrázku obalu knihy"/>
    <author>Příjmení a jméno autora</author>
    <price>Cena</price>
    <synopsis>
      Krátká anotace.
    </synopsis>
```

```
        <genre>Žánr</genre>
    </book>
```

7.2. JavaScript a HTML

Soubor `index.htm` obsahuje jak HTML strukturu stránky, která je plněna výsledkem transformace, tak JS kód, který transformace invokuje a řídí jejich průběh. Klíčovým prvkem je funkce `diplayResult(arg)`, která invokuje transformaci a předává XSLT procesoru parametry. Těmi jsou:

- parametr `info`, obsahující pořadí položky, u které má být zobrazen informační box. Hodnota parametru 0 znamená, že se nemá zobrazit žádný box.
- parametr `col`, obsahuje sloupec, resp. název elementu, podle kterého se mají položky seřadit. Hodnota parametru `nosort` znamená, že se položky nemají řadit.
- parametr `ord`, určuje, zda se mají položky řadit sestupně či vzestupně.
- parametr `gen`, určuje žánr, podle, kterého se mají položky filtrovat. Hodnota `ALL` znamená, že se mají zobrazit všechny položky.

Funkce `displayResult(arg)` je volána z HTML kódu pouze při nahrání stránky z elementu `<body>` pomocí atributu `onload`. V ostatních případech je funkce `displayResult(arg)` volána zevnitř jiných funkcí. Transformace je totiž vždy volána jako celek se všemi parametry a je tak potřeba ošetřit i hodnoty parametrů, které přímo s danou akcí nesouvisí tak, aby výsledek, který bude zobrazen uživateli, zůstal konzistentní. Proto je při uživatelské interacki (např. kliknutí na informační ikonu) nejprve volána funkce obstarávající danou interakci a jako jediný argument má novou hodnotu parametru. V těle funkce se pak vyhodnocují i hodnoty ostatních parametrů, které jsou buď implicitně nastaveny v závislosti na typu akce, která se má vykonat (např. při řazení stránky jsou všechny informační boxy skryty), nebo jsou nahrány z globálních proměnných hodnoty parametrů, jež byli determinovány v předchozích interakcích (např. při kliknutí na informační box je zachováno řazení položek a aplikovaný filtr). Transformace je poté spuštěna funkcí `displayResult()` se všemi parametry zevnitř pomocné funkce a do globální proměnné je uložena nová hodnota parametru pro potřeby další transformace.

HTML kód stránky je velmi jednoduchý. Při nahrání elementu `<body>` je spuštěna první transformace, která zobrazí uživateli data. Stránka již dále obsahuje pouze formulářové pole typu `<select>` se seznamem žánrů a JavaScript eventem, který při kliknutí na název žánru zavolá funkci `genre(arg)` s argumentem žánru, podle kterého se má filtrovat. A předpřipravené záhlaví tabulky, kde jsou jednotlivé sloupce vytvořeny tak, aby byla po kliknutí zavolána funkce `sort(arg)` s argumentem sloupce, podle něhož mají být položky seřazeny.

7.3. XSLT styl

Na začátku stylu se postupným větvením pomocí instrukcí `<xsl:if>` zjišťuje, zdali mají být zobrazeny všechny položky, nebo zdali se má aplikovat patřičný filtr. Pokud je parametr `$genre` nastaven na hodnotu `ALL`, vypíšou se položky podle XPath výrazu `//book`, tedy všechny ele-

menty <book> v dokumentu. Pokud je zvolen konkrétní žánr, jsou vypsány položky podle XPath výrazu s predikátem //book[genre=$genre], tedy všechny elementy <book>, jejichž element <genre> obsahuje stejnou hodnotu jako má paramter $genre.

Dalším větvením se poté determinuje, mají-li se položky řadit pomocí instrukce <xsl:sort> či nikoliv. A pokud ano, podle kterého elementu se mají seřadit a zda-li má být pořadí vzestupné či sestupné. Dále se aplikuje šablona pro element <book>. Kromě standardního výstupu řádku tabulky v HTML, určuje styl každému tagu <tr> atribut třídy, a to v závislosti na tom, zdali je řádek lichý nebo sudý (později jsou liché a sudé řádky pro přehlednost odlišeny CSS stylem). Činí tak pomocí matematické operace modus:

```
<xsl:attribute name="class">
 <xsl:if test="(position() mod 2)='0'">
     <xsl:value-of select="'suda'"/>
 </xsl:if>
 <xsl:if test="(position() mod 2)!='0'">
     <xsl:value-of select="'licha'"/>
 </xsl:if>
</xsl:attribute>
```

Poslední sloupec v tabulce je vyhrazen pro informační ikonu. Jeho obsah je řízen šablonou info. Pokud se parametr $info rovná pořadí záznamu, znamená to, že informační box u dané položky je aktivní a k elementu <img>, který vykresluje ikonu, se přidá atribut onclick s hodnotou displayInfo(0). To zajistí, že je-li informační box u dané položky aktivní, kliknutím na ikonu je informační box skryt. Pokud se parametr info nerovná pozici položky je hodnota tohoto atributu nastavena na concat('displayInfo(',position(),')') . Tedy takovým způsobem, že po stisknutí ikony je předána funkci displayInfo() pozice položky, u které se má box zobrazit.

Šablona info-row, pracuje s podobným principem. Také porovnává parametr $info s pozicí položky. Pokud se rovnají, šablona přidá pod řádku dané položky další, která obsahuje informace o publikaci (náhled obalu a anotaci publikace).

Kapitola 8

Závěr

Tato práce měla nastínit možnosti současných prohlížečů na poli XSLT transformací na klientské straně, tedy v současných webových prohlížečích. Velmi pravděpodobně jsem nepokryl úplně všechny odlišnosti prohlížečů, kterým jsem se v této práci věnoval. Nicméně si myslím, že hlavní rozdíly, které mohou markantně ovlivnit funkcionalitu nebo chování aplikace, jsou v této práci pokryty dostatečně. V tomto ohledu je důležité zejména porovnání implementací jednotlivých XSLT procesorů.

Ve druhé kapitole jsou nastíněny rozdíly mezi klientským a serverovým zpracováním. Pro závěry této práce je zejména důležitý fakt, že ikdyž se všeobecně nedostává XSLT transformacím na klientské straně tolik prostoru, je zde velké množství případů, kdy by mohla tato metoda fungovat bez větších potíží. Zlepšení v této oblasti, tedy větší užití XSLT transformací na klientské straně, je však do značné míry závislé na tom, jak bude v prohlížečích implementováno XSLT 2.0, které nabízí výrazně lepší možnosti pro prezentaci dat ve webovém prohlížeči. Nicméně přítomnost XSLT 2.0 nativního procesoru v prohlížeči je minimálně v nejbližší době velmi málo pravděpodobná. K této skutečnosti přispívá i fakt, že prohlížeče se stávají jakousi monokulturou nejpopulárnějších technologií. Na ostatní technologie tak nezbývá při vývoji příliš místa a jsou odsouvány na okraj, což pro ně vytváří poněkud paradoxní situaci, ze které je obtížně se dostat.

V tomto ohledu může být klíčový vývoj kolem Saxonu CE [1]a potencionálně i kolem jiných javascriptových implementací XSLT procesorů, které možná v budoucnu vzniknou. Na jejich příkladě může být demonstrováno, že využití XSLT transformací na klientské straně není technologie bez užitku. Samozřejmě za předpokladu, že se tato řešení alespoň částečně uchytí v praktické tvorbě webových stránek a aplikací.

Kromě funkce komparativní měla tato práce za cíl i demonstraci možností XSLT procesorů. Měla by tak sloužit jako sada základních vodítek při tvorbě webových aplikací využívajících XSLT transformace na klientské straně. Důležitá je v tomto ohledu kapitola 4, kde je popsána řada způsobů, jak styl k dokumentu připojit a jak předávat a nastavovat parametry transformace. Další kapitolou, kde jsou demonstrovány možnosti XSLT na klientské straně, je kapitola 7. V té je rozbráno fungování vzorové aplikace, kterou jsem pro potřeby této práce vytvořil.

Kapitola 6 se věnuje benchmarkingu nativních XSLT procesorů v současných webových prohlížečích. I zde se ukázaly zajímavé rozdíly mezi XSLT procesory. Přínos tohoto benchmarku je zejména v porovnání zde publikovaných výsledků s výsledky budoucích verzí porovnávaných prohlížečů. Bude tak možno pozorovat, zdali se výkon, XSLT procesorů vestavěných v rpohlí-

[1] 31. prosince expiruje release 0.3 a je přislíbena další alpha verze.

žečích mění k lepšímu, nebo zůstává stejný. Budoucí práci na tomto benchmarku vidím v rozšíření testovacích stylů a úpravách JS kódu. Vítané by také bylo zlepšení podmínek měření v JS, tedy například lepší granulace času a konzistentnější chování funkce `getTime()`.

Uplným závěrem bych chtěl načrtnout náměty pro další práce v této oblasti. Jistě by byla zajímavé téma věnující se metodikám při tvorbě interaktivních aplikací za pomoci použití XSLT transformací na klientské straně, tedy jakýsi přehled a ukázka použití best practices v této oblasti. A to jak ve verzi XSLT 1.0, ale i 2.0. Druhou prací, která by řešila aktuální problematiku by bylo detailnější prozkoumání některé z budoucích kompletních verzí Saxonu CE. Práce by se mohla zabývat omezeními, které má klientská verze Saxonu oproti svým standardním verzím, vývojem v oblasti Interactive XSL a porovnáním výkonu s jinými XSLT procesory.

Příloha A

Přílohy

Obsah složky `benchmark`:

`bottles.xsl`, `math.xsl`, `sort.xsl`, `lookup.xsl`, `hanoi.xsl`, `string.xsl`, `reverse.xsl`
: Jednotlivé testovací styly pro účely benchmarkingu.

`bottles.xml`, `math.xml`, `1000rows.xml`, `1001rows.xml`, `hanoi.xsl`, `two_gent.xml`, `rich_iii.xml`
: XML dokumenty pro účely benchmarkingu.

`bench.htm`, `bench_ie.htm`
: Rozhraní pro spuštění benchamarku a verze pro spuštění benchmarku v IE.

Obsah složky `bp`:

`bp.xml`
: Tato práce ve formátu DocBook.

`bp.pdf`
: Tato práce ve formátu pdf.

`bp.html`
: Tato práce ve formátu HTML.

Obsah složky `aplikace`:

`index.xhtml`
: HTML a JS kód aplikace.

`books.xml`
: XML dokument s daty vzorové aplikace.

`booklist.xsl`
: XSLT styl, který aplikace používá.

Literatura:

[1] KAY, Michael. XSLT 2.0 and XPath 2.0 Programmer's Reference Město:Wrox Press, 2008 ISBN 978-0-470-19274-0

[2] KAY, Michael. XSLT in the Browser. In: XML Prague 2011 – Conference Proceedings, Praha: MATFYZPRESS, 2011, s. 125 - 134. ISBN: 978-80-7378-160-6. Dostupný z WWW: http://www.xmlprague.cz/2011/files/xmlprague-2011-proceedings.pdf

[3] Saxonica. Saxonica: XSLT and XQuery Processing : Saxon Documentation [online]. 2011, Generated at 11 September 2011 at 09:01 [cit. 2011-11-20]. Dostupné z WWW: http://www.saxonica.com/welcome/welcome.xml.

[4] Mozilla Foundation. Mozilla Dveloper Network : XSLT [online]. c2011, Page last modified 21:17, 12 May 2011 [cit. 2011-11-22]. Dostupné z WWW: https://developer.mozilla.org/en/XSLT.

[5] Microsoft. MSDN Library : XSLT for MSXML [online]. c2011, [cit. 2011-11-22]. Dostupné z WWW: http://msdn.microsoft.com/en-us/library/ms759204%28v=VS.85%29.aspx.

[6] Opera Software ASA. Opera : XML support in Opera Presto 2.9 [online]. c2011, [cit. 2011-11-21]. Dostupné z WWW: http://www.opera.com/docs/specs/presto29/xml/.

[7] libxslt : The XSLT C library for GNOME [online]. [cit. 2011-11-21]. Dostupné z WWW: http://xmlsoft.org/XSLT/.

[8] BUSATTO, Giorgio ; LOHREY, Markus; MANETH, Sebastian. Efficient memory representation of XML document trees. Information Systems [online]. June-July 2008, Volume 33, Issues 4-5, [cit. 2011-11-15]. Dostupný z WWW: http://parsys.informatik.uni-oldenburg.de/~skript/fs-pub/efficient-xml-is.pdf. ISSN 0306-4379.

[9] WESTIN, Ken. Digital Web Magazine [online]. 2004 [cit. 2011-11-30]. An Introduction to Client-Side XSLT: It's Not Just for Server Geeks Anymore. Dostupné z WWW: http://www.digital-web.com/articles/client_side_xslt/.

[10] W3C. World Wide Web Consortium (W3C) [online]. 1999 [cit. 2011-11-27]. XSL Transformations (XSLT) Version 1.0. Dostupné z WWW: http://www.w3.org/TR/xslt.

[11] W3C. World Wide Web Consortium (W3C) [online]. c2011, Last updated Fri 02 Dec 2011 06:20:10 AM CET [cit. 2011-12-04]. CSS Software. Dostupné z WWW: http://www.w3.org/Style/CSS/software.

[12] Abiss.gr. Sarissa : Sarissa Home Page [online]. c2008 [cit. 2011-12-04]. Dostupné z WWW: http://dev.abiss.gr/sarissa/.

[13] W3C. World Wide Web Consortium (W3C) [online]. 2010 [cit. 2011-11-04]. W3C Associating Style Sheets with XML documents 1.0 (Second Edition). Dostupné z WWW: http://www.w3.org/TR/xml-stylesheet/.

[14] DUCHARME, Bob. XML.com [online]. 2001 [cit. 2011-11-05]. Math and XSLT. Dostupné z WWW: http://www.xml.com/pub/a/2001/05/07/xsltmath.html.

[15] JAKOB, Nielsen. Useit.com: Jakob Nielsen on Usability and Web Design [online]. 2010 [cit. 2011-11-06]. Website Response Times (Jakob Nielsen's Alertbox). Dostupné z WWW: http://www.useit.com/alertbox/response-times.html.