TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N 2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

Webový XML editor

Web XML editor

Diplomová práce

Autor: Bc. Martin Vagenknecht

Vedoucí práce: Ing. Přemysl Svoboda

V Liberci 11. 5. 2010

Prohlášení

Byl (a) jsem seznámen (a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském zejména 60(školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé BP a prohlašuji, že **souhlasím** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom (a) toho, že užití své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených universitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval (a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Rád bych poděkoval vedoucímu diplomové práce Ing. Přemyslu Svobodovy za rady, čas a trpělivost, při zhotovení této práce.

Abstrakt:

Cílem diplomové práce je popsání současných možností jazyka XML, jeho schémat a způsoby zpracování XML dokumentů. Jádrem práce je pak implementace webového XML editoru ve vhodně zvolené webové technologii. Aplikace by měla umožňovat editaci a vytváření XML dokumentů a podporu XML schémat. Vizí je vytvořit plnohodnotný webový XML editor, který by dokázal konkurovat aplikacím určeným pro desktop. Zvolené prostředí webového prohlížeče je zajímavou výzvou pro zhotovení aplikace, která bohužel s sebou ponese i určitá technologická omezení.

Klíčová slova:

- Webová aplikace
- Webové technologie
- Xml
- Silverlight
- Xml editor
- WCF

Abstract:

This diploma thesis describes a deployment and current options of XML language, XML schemes and XML documents processing. The diploma thesis focuses on implemantation of a web based XML editor, using an appropriate web technology. The application should edit and create XML documents along with a support of XML schemes. The vision is to create a fully fledged web based XML editor, that could compete with applications developed for a desktop. A web browser environment is a challenging task of developing an application, that will unfortunately bring about specific technological limitations.

- Web application
- Web technology
- Xml
- Silverlight
- Xml editor
- WCF

Obsah

Seznam obrázků

Seznam příkladů

Seznam symbolů, zkratek a termínů

SGML – Standard Generalized Markup Lanuguage: univerzální značkovací jazyk, který umožňuje definovat značkovací jazyky jako své podmnožiny.

W3C – World Wide Web Consortium: mezinárodní konsorcium, jehož členové vyvíjejí standardy pro World Wide Web.

URI – Uniform Resource Identifier: Jednotný indikátor zdroje.

URL – Uniform Resource Locator: Jednotný lokátor zdrojů.

SOAP – Simple Object Access Protocol

WSDL – Web Services Description Language: Xml formát pro popis SOAP komunikace.

API - Application Programming Interface: rozhranní pro programování aplikací.

RIA – Rich Internet Application

1 Úvod

Informace a jejich efektivní zpracování dnes mají velmi strategický význam. S tím je spojena potřeba využívat nějaký jednoduchý otevřený formát, který není úzce svázán s nějakou platformou nebo proprietálních technologií.

XML znamená budoucnost, která právě začala. Za původně prostou ideou neomezeného rozšiřitelného jazyka pro flexibilní výměnu dat mezi různými systémy se dnes skrývá široká řada produktů, technologií a standardů, kde se tento jazyk využívá. Příležitostí, kde a jak aplikovat XML je celá řada, ať už je využita jako prostředí pro webové služby, či výměnu dat a mnoho dalšího.

S těmito vizemi souvisí i nutnost editovat XML dokumenty. Aby uživatel mohl například vytvořit webovou stránku, napíše kód pro tvorbu internetových stránek v jednom z vybraných jazyků, který je po zhotovení nahrán na server a tím zpřístupněn okolí. Tímto jazykem pro zhotovení internetových stránek může být XHTML (*eXtensible HyperText Markup Language*). XHTML je forma XML, což je textový formát a může být editován v nejjednodušší formě například v aplikaci *Notepad* (Poznámkový blok), který je součástí operačního systému. K editaci XML, ovšem mohou být využity sofistikovanější produkty zaměřené právě pro práci s XML. Tyto aplikace disponují funkcemi, jako například formátování XML dokumentu, zvýrazňování syntaxe pro lepší orientaci.

V informačním světě je stále více a více informací využívajících elementy a atributy ve formátu XML a roste potřeba nejen tyto informace prohlížet a prezentovat, ale také vytvářet a editovat.

1.1 Popis problému a specifikace cíle

1.1.1 Uživatelské potřeby

Webová aplikace pro práci s XML dokumenty dostala název Web XML Editor. Uživaleli nabídne základní funkce, které jsou očekávány od aplikace podobného typu. Základními vlastnostmi každého editoru, ať se jedná o klasický textový nebo XML editor, jsou vytvoření nového dokumentu, uložení a načtení dokumentu. Po nahrání dokumentu bude

aplikace poskytovat obecné informace a na základě analýzy XML dokument zobrazí. Samozdřejmostí je podpora XML schémat. Dle schématu umožňuje aplikace validaci XML dokumentu spolu s výpisem možných chyb.

Technické požadavky:

- Webová aplikace
    - Serverová část
        - Technologie webových aplikací
    - Klientská část
        - Webový prohlížeč
        - Dolpněk Microsoft Silverlight 4.0

Populárnost webových aplikací je především díky webovému prohlížeči, který je standardním vybavením počítače. Prohlížeč slouží jako klient (nazývá se také *tenkým klientem*), který nezná aplikační logiku, ale pouze interpretuje danou aplikaci. Hlavní výhodou je schopnost aktualizovat a spravovat webové aplikace bez nutnosti instalovat software. Na aplikace, jako jsou e-mailový klient, jsme už zvyklí, ale postupně se přidávají kancelářské programy, plánovací kalendáře nebo třeba i grafické editory.

1.1.2 Specifikace cílů

Cílem této práce je navrhnout a implementovat aplikaci, podle předem daných požadavků s ohledem na možnosti zvolených technologií a vybraných postupů. Webová aplikace musí poskytovat předem dané informace a možnosti. Návrh designu je ponechán vlastní tvorbě.

2 XML

XML je zkratka pro *Extensible Markup Language* (česky *rozšířitelný značkovací jazyk*) je obecně značkovací jazyk, který slouží pro popis, transport a ukládání dat. Ve své nejprostší podstatě je specifikace XML sadou doporučení, kterou definovalo *W3C* pro popis strukturovaných dat v čistém textu.

Definice XML podle W3C:

- XML je značkovací jazyk podobně jako HTML
- XML byl navrhnut pro uložení a zobrazování dat
- Značky XML jazyka nejsou předem definované. Je na vývojáři, aby si vytvořil vlastní definici tagů.

Podobně jako HTML je i XML jazykem založeným na značkách psaných uprostřed špičatých závorek. Stejně jako v případě HTML je i tento formát snadno přenositelný díky textovému formátu XML. To umožňuje vytvářet a editovat dokumenty v jakémkoliv textovém editoru.

Na rozdíl od HTML nemá XML fixní sadu značek. XML je metajazyk, který umožňuje vytvářet jiné jazyky značek, protože XML specifikuje sadu jednotlivých pravidel pro pojmenování a uspořádání prvků, přičemž vlastní formát vytváříte pomocí vlastní sady prvků. Značky popisující strukturu mohou mít obecně jakékoliv názvy prvků, které budou nejlépe popisovat vyše data. Právě takto flexibilita byla hlavním důvodem k popularitě a úspěchu jazyka.

XML specifikace ukládá následující cíle.

1. XML musí být jednoduché pro využití na internetu. Uživatelé musí mít možnost zobrazit XML dokumenty, tak snadno a rychle jako HTML dokumenty. V praxi to znamená to, že XML prohlížeče musí být stejně robustní jako pro klasické HTML, ale princip zůstává.
2. XML musí podporovat široké spektrum aplikací. Mělo by být pro tyto aplikace přínosné (authoring, prohlížení, obsahová analýza).
3. Kompatibilita se *SGML*.
4. Snadné napsání programu pro zpracování XML.

5. Množství volitelných rysů XML musí být omezeno na absolutní minimum, nejlépe na nulu.
6. XML dokumenty musí být čitelné i pro člověka a přiměřeně srozumitelné.
7. XML design musí být připraven rychle.
8. Design XML musí být formální a stručný.
9. Musí být snadné tvořit XML dokumenty.
10. Stručnost XML značkování má minimální důležitost.

Značkovací jazyk se v podstatě skládá ze dvou typů značek:

- **formátovacích**, které slouží k formátování dokumentu
- **logických**, zajišťující logické oddělení dat

HTML míchalo oba typy těchto značek, a tím bylo dosaženo poměrně slušných možností. Časem se ukázalo, že je daleko lepší tyto typy značek oddělit od sebe. Struktura dokumentu bude tvořena pomocí logických značek a to jak bude dokument zobrazen, budou specifikovat formátovací značky, které budou z větší části separované. Tuto techniku zprostředkovávají kaskádové *CSS* styly. XML tuto separaci značek posouvá o něco dále. XML soubor neobsahuje žádné informace o formátování. Ty jsou obsahem formátovacího souboru, který používá *CSS, XSL* styly nebo jiný formátovací jazyk. Jestli je potřeba zobrazit dynamicky data v jazyku HTML, pak je příliš pracné editovat HTML dokument při každé změně. XML nabízí řešení v podobě uložení dat v XML souboru a o rozvržení a zobrazení dat se postará HTML, které nebudeme muset měnit.(Načtení XML dokumentu zajistí *JavaScript,* který dovede číst externí zdroj XML dat a upravit obsah HTML stránky.)

Využití jazyka XML je především pro ukládání hierarchicky uspořádaných dat, jako mohou být například adresáře, katalogy, atd. Soubory XML nejsou nic speciálního. Jsou to jen informace v podobě textu a obsluhovat tyto soubory může kterýkoliv software, který zvládne práci s textem. Použití speciálního software ovšem navíc umožňuje přistupovat k jednotlivým elementům speciálně. Další výhodou plynoucí z této vlastnosti je nezávislost na použitých platformách. Této výhody využívají vývojáři při problému komunikace a výměny dat přes internet u aplikací na nekompatibilních systémech.

2.1 Výhody použití XML

Úspěch jazyka XML spočíval hlavně v jednoduchosti. Pravidla jsou totiž mnohem stručnější a jednodušší než pravidla jeho předchůdce, kterým bylo *SGML* (*Standard Generalized Markup Language*). Tento fakt umožňuje, že jednodušší dokumenty XML jsou čitelné i pro lidi. Mezi další zisky, které plynou z použití XML v aplikacích patří:

- **Akceptace.** XML je všudypřítomné. Kdykoliv je potřeba sdílet nějaký data, XML je nejpřijatelnějším a prvním uvažovaným řešením.
- **Rozšiřitelnost a flexibilita.** XML neobsahuje žádná pravidla pro sémantiku dat a je možné XML použít pro jakýkoliv druh dat. Proto je snadná a levnější imlementace.
- **Příbuzné standardy a nástroje.** Další příčinou úspěchu XML jsou nástroje a související standardy (například *XML schema, Xpath, XSLT*, ...), které pomáhají při práci s XML. Programátor má k dispozici téměř v každém jazyku hotové komponenty pro ověřování XML, zda vyhovuju předem určeným pravidlům, vyhledávání v XML nebo pro transformaci a úpravu XML dokumentu.

2.2 XML dokument

XML je z určitého pohledu dost striktní standard. Tyto pravidla jsou ovšem vyváženy zachováním široké kompatibility. Důvodem k přísným pravidlům XML je obtížnost rozlišit neškodnou odchylku od závažné chyby. Ještě horší by byla zkutečnost, že různé parsery XML by mohli na tyto odchylky reagovat různě, což by vedlo k nejjednoznačnostem při zpracování takového XML dokumentu. Typickým příkladem této nejjednoznačnosti jenž není založen na XML je jazyk HTML.

Všechny parsery XML proto provádějí kontrolu validity kódu, aby bylo zabráněno podobným problémům tohoto druhu. Pokud XML dokument nevyhovuje předem definovaným standardům je okamžitě odmítnut.

Správně strukturovaný XML dokument musí obsahovat na začátku XML deklaraci. Je v ní uvedeno, jaká je použita verze XML a jaké je použito kódování. Definice kódování může chybět. V takovém případě je použito kódování ISO 10646. Obvykle se pro komunikaci používá UTF-8, které je kompatibilní s ASCII. Pro použití českého jazyka lze využít ISO-

8859-2 nebo Windows-1250. Následuje definice struktury XML dokumentu, která ovšem není povinná. Po této části je již samotný obsah XML dokumentu.

Aby se mohl XML dokument považovat za dobře strukturovaný musí splňovat následující kritéria:

- Každá otevírací značka musí mít odpovídající uzavírající značku.
- Prázdný prvek musí končit na />.
- Prvky se nikdy nesmějí překrývat (křížit).
- Názvy prvků a atributů musejí používat stejná velká i malá písmena.
- Prvek nemůže obsahovat dva atributy se stejným názvem, protože by nebylo možné je od sebe navzájem rozlišit. Prvek ovšem může obsahovat dva vnořené prvky se stejným názvem.
- Dokument může mít pouze jediný kořenový prvek. Kořenový prvek je prvek nejvyšší úrovně, jímž dokument začíná, a ve kterém se nachází veškerý obsah dokumentu.
- Všechny atributy musí mít hodnoty vloženy do uvozovek, nebo apostrofů.
- Komentáře nemohou být umístěny uvnitř značek. Komentáře XML mají stejný formát jako komentáře HTML – jsou obklopeny značkamami <!--a -->.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- XML poznámka -->
<note>
  <to>Petr</to>
  <from>Martin</from>
  <heading>Připomínka</heading>
  <body>Nezapomeň na tento víkend</body>
</note>
```

Příklad 1: XML dokument

2.3 Definice typu dokumentu

Velkou zásluhu na úspěchu standardu XML má jeho flexibilita. XML dává možnost, jak přesně vytvořit takový značkovací jazyk, jaký je potřeba. Taková flexibilita s sebou, ale přináší jisté problémy. Pokud bude formát XML používat i někdo jiný, jak bude zajištěno, aby dodržel stanovená pravidla?

Tento problém vyřeší vytvoření formálního dokumentu, ve kterém budou specifikována všechna pravidla vlastního značkovacího jazyka. Tento dokument se nazývá schéma jazyka. Tyto pravidla nezahrnují podrobnosti o syntaxi, protože tyto požadavky jsou součástí základního standardu XML. Jsou zde stanovena pravidla, která se musejí dodržovat s vlastním typem dat.

- **Slovník dokumentu.** Určuje názvy prvků a atributů.
- **Struktura dokumentu.** Zde je stanoveno, kde se dané značky mohou vyskytovat. To zahrnuje pravidla, jež se týkají umístění značky před nebo za jiné, nebo umístění uvnitř značky. Je možné také určit počet výskytů konkrétní značky.
- **Podporované datové typy.** Specifikace, zda jsou data v podobě obyčejného textu nebo je mají číselný formát, datum, apod.
- **Povolené rozsahy dat.** Nastavení různých omezení. Například omezení rozsahu číselného formátu.

Ověřování platnosti je sice důležitou součástí projektů, v nichž dochází k výměně dokumentů mezi heterogeními platformami, znamená to ovšem i zvýšenou režii celé aplikace. Kontrola dokumentů znamená vyčkat na dokončení analýzy podstatných uzlů. Prověřeny musí být informace o počtech, typech, hodnotách atributů a závyslosti mezi uzly. Dostane-li aplikace plně ověřený XML dokument, má zaručenu nejen jeho správnou syntaxi, ale také korektnost obsahu. Z uvedených faktů vyplývá, že při čtení XML bez kontroly platnosti poběží *reader* rychleji než kdyby byla kdyby byl dokument kontrolován a proto *readery* XML nabízejí ověřování pouze jako volbu a ne nutnost.

2.3.1 DTD

Nejspíše asi nejznámější jazyk pro popis schématu dokumentu je **DTD**. Ten má odlišnou strukturu od XML a proto je pro jeho analyzování potřeba použít jiné speciální procesory. Tato deklarace schema dokumentu může být obsažena uvnitř XML dokumentu nebo může být v podobě externího zdroje.

Deklarace uvnitř dokumentu musí být obalena definici typu (DOCTYPE) s následující syntaxí.

<!DOCTYPE root-element [element-declarations]>

V DTD se musí nejprve určit kořenový element(root), který je pouze jeden. Dále se potom nadefinují ostatní elementy, které kořenový element uzavírá v sobě. Jsou to jeho potomci.

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
<note>
  <to>Petr</to>
  <from>Martin</from>
  <heading>Připomínka</heading>
  <body>Nezapomeň na tento víkend</body>
</note>
```

Příklad 2: zápis DTD schématu

Pro definici přípustného výskytu potomků slouží operátory plus, hvězdička, svislá čára, otazník a čárka. Plus, hvězdička a otazník označují počet výskytů. Hvězdička značí libovolný počet, otazník znamená žádný nebo jeden a plus slouží k označení alespoň jednoho výskytu elementu. Vyjádření vztahu je možné pouze rodič – potomek a nelze nadefinovat dva elementy se stejným názvem, ale jiným obsahem. Čárka mezi elementy určuje posloupnost i pořadí daných elementů. Svislá čára nám odděluje elementy, mezi kterými máme na výběr. Nahrazení by bylo možné logickém operátorem „nebo“.

Každý element může navíc obsahovat atributy. Definice atributu určuje jejich povinnost výskytu a datový typ. Nejprve se určí, ke kterému elementu atribut patří, jeho název a potom případně povinnost výskytu.

2.3.2 XDR

Za zmínku stojí i tento formát pro tvorbu XML schémat. Vychází z návrhu, který Microsoft předložil konzorciu *W3C* v roce 1998. Schema XDR jsou flexibilní a překonávají některá omezení DTD. XDR narozdíl od DTD popisuje strukturu dokumentu stejnou syntaxí,

kterou je popsán samotný dokument XML. Kromě toho v DTD tvoří veškerý datový obsah znaková data. Jazyk schémat umožňuje určit datový typ elementu nebo atributu.

2.3.3 XML Schema

DTD bylo velmi dlouho považováno za standard pro různé počítačové platformy. S rostoucímy požadavky bylo potřeba vyvynout nový jazyk pro popis struktury dokumentu a proto byl přijat novější standard – **XML Schema**, který je z technického hlediska mnohem vyspělejší. Důvodů pro použití XML Schema oproti DTD je hned několik. XML Schema je mnohem lépe uzpůsobeno pro budoucí rozšíření a je mnohem bohatší než DTD. Výkonnost je spojováná i s podporou datových typů a jmenýných prostorů. V dnešní době je XML Schema podporováno téměř všemy parsery na všech platformách.

XML Schema (dále jen XSD) popisuje syntaxi a semántiku dokumentů XML prostřednictvím standardní syntaxe XML. XSD specifikuje omezení pro obsah dokumentu a slovník, kterému musí dokumenty vyhovět, aby byli podle daného schematu platné. Dokumenty musí splňovat všechny závyslosti mezi uzly, přiřazovat atributy správného typu a dodržet přesnou kardinalitu uzlů.

```
<?xml version="1.0"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading"
type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Příklad 3: zápis XSD schéma

2.4 Jmenné prostory

Jakmile získal standard XML solidní základy, objevilo se mnoho značkovacích jazyků XML (často nazývané jako gramatiky XML). Tyto jazyky byli specifické pro různá odvětví, procesy a druhy informací. Bylo důležité, aby se dal jeden typ rozšířit o další dodatečné prvky specifické pro dané odvětví. Musela být umožněna i kombinace několika různých gramatik, ze kterých je dokument XML složen. Tyto předpoklady s sebou nesou i určité problémy. Co dělat, když je potřeba zkombinovat dvě gramatiky, které u prvků používají stejné názvy? Jak se odliší od sebe navzájem?

Řešením této situace je standard XML Namespaces, neboli jmenné prostory XML. Tato technika umožňuje, aby každý značkovací jazyk XML měl svůj vlastní jmenný prostor, který jednoznačně definuje všechny prvky, jež se k tomuto jmennému prostoru vztahují. Z formálního hlediska umožňují jmenné prostory jednoznačnost prvků, ze kterého značkovacího jazyka je použit.

Všechny jmenné prostory XML používají **URI** (*universal ressource identifier*). *URI* je na pohled podobná klasickému *URL* webové stránky. Například typickým jmenným prostorem by mohlo být http://www.mojedomena.cz/mujstandard. Použití *URI* pro jmenné prostory XML je z toho důvodu, že je mnohem více pravděpodobné, že budou jedinečné, protože jestli vlastníte web nebo doménu, kterou máte pod svojí kontrolou, budete *URI* směřovat na ni. Jmenný prostor ovšem nutně nemusí být *URI*. Obecně by se dalo říci, že je akceptována jakákoliv posloupnost textu.

Specifikace prvků o tom, že patří do konkrétního jmenného prostoru probíhá tak, že do úvodní značky je přidán atribut xmlns s názvem jmenného prostoru. Jak již bylo řečeno pomocí jmenných prostorů je možné od sebe oddělit jednotlivé části dokumentu. K tomuto oddělení slouží prefixy jmenných prostorů. Prefix je krátká posloupnost znaků, která se vkládá před název značky, čímž se specifikuje, do kterého jmenného prostoru patří.

2.5 Využití XML v praxi

Každý, kdo někdy použil internet a zobrazil si nějakou stránku, přišel do kontaktu s XML, který byl ovšem zpracovaný XML procesorem, webovým prohlížečem a zobrazeným

na obrazovku. Díky textovému formátu XML lze snadno k jeho přenostu využít protokol HTTP *(HyperText Transfer Protokol*). K tvorbě webových stránek se tudíž čím dál tím více začíná používat norma XHTML, která vznikla úpravou známé normy HTML. Úprava měla za následek to, že webová stránka byla XML dokument a musela splňovat všechny pravidla správně strukturovaného XML dokumentu.

Zobrazování XML je založeno na oddělení dat, která jsou v XML dokumentu, od způsobu formátování, který se může měnit a je uložen v dalších souborech. Tedy při zobrazení XML, procesor načte data z XML dokumentu a podle zadaného formátování tyto data vykreslí na obrazovku. Toto je velice užitečné pro prohlížení dat na různých zařízeních. Pro různá zařízení jiný druh formátování. Díky této variabilitě můžeme plně využít možnosti zařízení, na kterém jsou data zobrazována. Pro transformaci XML dat do uživateli příjemné podoby se používá *XSL* (*eXtensible Stylesheet Language*). Tento formát je opět založen na XML.

Používané XML formáty by se dali rozdělit do dvou skupin podle toho, kdo daný standard navrhl. První ze skupiny formátů patří *W3* konsorciu, které je díky jeho autoritě označilo jako standardy.

SVG -*Scalable Vector Graphics* je formát pro popis vektorové grafiky a animací na ní založených. Hlavní použití by měl tento formát najít zejména na webových stránkách jako standardní formát pro popis vektorové grafiky.

MathML - zde se jedná o formát určený k popisu matematických výrazů.

XHTML – značkovací jazyk pro tvorbu hypertextových dokumentů

Druhou skupinou jsou formáty vyvinuté jinými skupinami než *W3C* a nebo vývojáři jednotlivých programů jako formáty ukládání dat.

WML – formát určený pro tvorbu internetových stránek pro mobilní zařízení. Těmito zařízeními jsou myšleny hlavně ty zařízení, které mají malý displej, paměť a pomalý procesor.

OpenOffice – XML formát pro popis Office dokumentů.

RSS – XML formát určený pro čtení novinek na webových stránkách, obecněji na syndikaci obsahu.

XML se také používá v aplikacích, kde dochází k výměně dat voláním vzdálených metod. Příkladem takové komunikace jsou webové služby (*Web Services*), které ke komunikaci používají protokol *SOAP*(*Simple Object Access Protokol*). Služby funguje na principu přijmutí požadavku a odeslání odpovědi. Požadavkem je jméno volané metody se zadanými parametry. Server se postará o spuštění příslušné metody s parametry z požadavku a výsledek vrátí klientovi. Požadavek i odpověď jsou ve formátu XML. S webovými službami také souvisí zkratka *WSDL* (*Web Services Description Language*). Pomocí tohoto nástroje je nadefinován seznam metod s názvy, pořadím a typy parametrů, které mohou být na daném serveru volány klientem. *WSDL* je vlastně konfiguračním souborem pro klienty webové služby, podle kterého se mohou automaticky nastavit a nabídnout uživateli všechny metody, co tato konfigurace obsahuje i s názvy a typy parametrů. *WSDL* je samozřejmě také XML.

2.6 Možnosti zpracování XML

Pro práci s XML dokumenty v nějaké aplikaci máme situaci jednodušší než by se zdálo. Není potřeba si psát svůj vlastní analyzátor XML, ale máme možnost použití některého z mnoha již existujících parserů. Parser může být ve formě programu nebo programátorské knihovny, která čte XML dokument ze souboru (nebo jiného zdroje, takovým zdrojem může být i soubor získaný z webového serveru prostřednictvím protokolu *HTTP*) a nízkoúrovňově provádí syntaxtickou analýzu XML dokumentu. Samotný XML dokukument může být zpracováván několika způsoby, které prošli historickým vývojem a proto je k dispozici několik různých *API*, hodících se pro různé aplikace.

2.6.1 Readery XML

První parsery nabízeli **rozhranní řízené událostmi.** To znamená, že parser postupně četl Xml dokument a byli volány funkce pro obsluhu událostí. Takovými událostmi jsou například začátek a konec elementu, obsah elementu ve formě textu apod. Funkce navíc obsahovali další užitečné parametry jakými mohou být název elementu, seznam atributů apod. Výhody, které přináší událostmi řízené zpracování XML dokumentu je rychlost a nízké paměťové nároky. V aplikacích, kde je kladen důraz na rychlost se proto tento přístup aplikuje nejčastěji. Nevýhodou tohoto přístupu je naopak nutnost přečíst dokument během jednoho sekvenčního průchodu.

Reader umožňuje procházet obsah XML dokumentu uzel po uzlu (nikoliv bajt po bajtu). Dokument tímto přestává být textovým souborem, ale kolekcí uzlů. Uzel znamená v podání readeru jakýkoliv vyjadřovací prvek ve struktuře XML dokumentu. Tedy uzlem je například element, komentář, ale i text uvnitř elementu. Jak již bylo napsáno, readery XML nevyžadují, aby bylo do pamětí ukládáno více dat než je nutné. Po otevření XML dokumentu pracuje se s jednoduchými ukazateli na uzly. Následuje jednoduché procházení a vyhledání konkrétních uzlů, které jsou potřeba. Pomocí tohoto postupu není paměť zatěžována celým dokumentem, ale do paměti je načten pouze obsah aktuálně procházeného uzlu.

Readery jsou předstastavitelé tzv. *pull modelu* (tahání dat), v protikladu k typickému *push modelu* **SAX** (*Simple API for XML parsing*). Je možné vytvořit push model (tlačení dat) na základě *API*, vycházejícího z pull modelu. Opak však není možný. To má za následek to, že v aplikacích *.NET Framework* (dále .NET) schází jakákoliv podpora modelu *SAX*. Parser *SAX* řídí přímo průběh procesu čtení dokumentu a odesílá data klientské aplikaci. Klasický Xml reader je mnohem pasivnější a ponechává řízení celého procesu klientské aplikaci.

Při využití parseru *SAX* pro čtení Xml dokumentu je potřeba nejprve vytvořit třídu, která má rozhraní se kterým umí parser komunikovat. Tato třída se při inicializaci předá parseru. Pro začátek čtení XML dokumentu slouží metoda *parse()*. Parser postupně čte dokument a vždy, když narazí na nějaký uzel, je zavolána příslušná metoda v programátorem definované třídě, která byla na začátku registrována při inicializaci. Při volání metod parserem jsou dle potřeby předány i parametry, obsahující dodatečné informace. Programátor si sám nadefinuje obsah metod, které reagují na příslušné uzly a tím se může zpracovat celý dokument. Příkladem může být zpracování začátečního elementu. Musí být nadefinována metoda startElement, ve které jsou parametry určující jmenný prostor, ve kterém se nachází zpracovávaný element, jméno elementu s prefixem a atributy elementu.

2.6.2 Zpracování XML v paměti

Zpracování XML, které je založeno na proudu, sice nabízí nejmenší zatížení, ovšem na druhou stranu je také nejméně flexibilní. V mnoha scénářích zpracování XML pravděpodobně nebudete chtít pracovat na tak nízké úrovni. Často také budete chtít, abyste měli k dispozici nějaký snadný způsob, jakým byste mohli vybrat obsah prvku, o který se zajímáte, pouze s několika jednoduchými řádky kódu. V paměti se s XML pracuje daleko pohodlněji. Pro

zpracování XML v paměti neexistuje pouze jeden způsob, ale hned několik. *.NET* nabízí následující třídy, umožňující číst obsah souboru XML a navigovat se po něm.

XmlDocument – třída *XmlDocument* implementuje úplný *XML DOM Level 2 Core,* který byl specifikován *W3C* organizací. V současnosti se jedná o nejvíce standardizované rozhraní k datům XML. *XmlDocument* ukládá informace jako strom uzlů. Uzel je základní stavební materiál souborů XML. Mohou jimi být prvky, atributy, komentář nebo hodnota v prvku. Každý z uzlů je reprezentován oddělneným objektem. Uzly se seskupují do kolekcí. V *.NET* reprezentuje uzel objekt *XmlNode* a skupina těchto objektů, které existují na stejné úrovni, se zabaluje do kolekcí *XmlNodeList* .

XPathNavigator. Využití této třídy nabizí rychlejší a praktičtější model než *XML DOM* společně s rozšířenými rysy vztahující se k vyhledávnání. Oproti *XmlDocument* ovšem neposkytuje možnost provádět změny v dokumentu a ukládat je. Systém práce je s touto třídou podobný jako *XmlDocument*. Jakmile jsou načteny všechny informace do paměti, je omožněn pohyb prostřednictvím uzlů. Klíčovým rozdílem je to, že používá přístup založený na kurzoru. Pohyb po datech se provádí pomocí metod jako *MoveToNext().* Objekt *XpathNavigator* může být v daném okamžiku umístěn pouze na jednom uzlu.

XDocument. XDocument poskytuje nejvíce intuitivní a efektivní *API* pro práci s XML. Formálně se jedná o součást **LINQ toXML**, nicméně je užitečné i když nesestavujete dotazy *LINQ*.

3 Webové technologie pro zhotovení Web XML Editoru

Webová aplikace je v softwarovém inženýrství aplikace, která je poskytovaná uživatelům z webového serveru prostřednictvím celosvětové sítě internet, nebo vnitropodnikovou podobu - intranet. Populárnost získaly především díky všudypřítomnosti webového prohlížeče jako klienta (ten je pak nazýván *tenkým klientem*, protože nemusí znát logiku aplikace) a schopnosti aktualizovat a spravovat webové aplikace bez nutnosti šířit a instalovat software pro všechny uživatele.

V současné době existuje několik technologií, které by byli použitelné pro zhotovení webové aplikace. Každá s sebou nese své klady i zápory. Je na vývojáři, aby zvážil, co od dané aplikace očekává, shrnul její přednosti a podle toho se rozhodl, jakou ze zvažovaných technologií zvolit. Pro toto rozhodnutí je potřeba shrnout všechny klíčové vlastnosti webové aplikace. Takovými vlastnostmi mohou být:

- **Design a celkový vzhled** – Pro případného uživatele aplikace je toto jednou z důležitých a rozhodujích vlastností. Je to první věc, kterou uživatel uvidí při zobrazení aplikace.
- **Použitelnost** – Při použití webové aplikace je nutné být připojen k síti internet, což může způsobit někdy problémy. (Technologie Silverlight od verze 3.0 umožňuje offline běh aplikace)
- **Dostupnost** – Aplikace by měla být technologicky nezávislá na použité platformě.
- **Výkonnost** – Tento požadavek u webových aplikací spíše souvisí s rychlostí a stabilitou připojení k internetu, protože se ve většině případů se jedná o výpočetně méně náročné aplikace.
- **Bezpečnost** – Jelikož jsou všechny data přenášena přes síť je nutné počítat i s bezpečností a zvážit důležitost přenášených dat.

3.1 Skriptovací technologie PHP a ASP

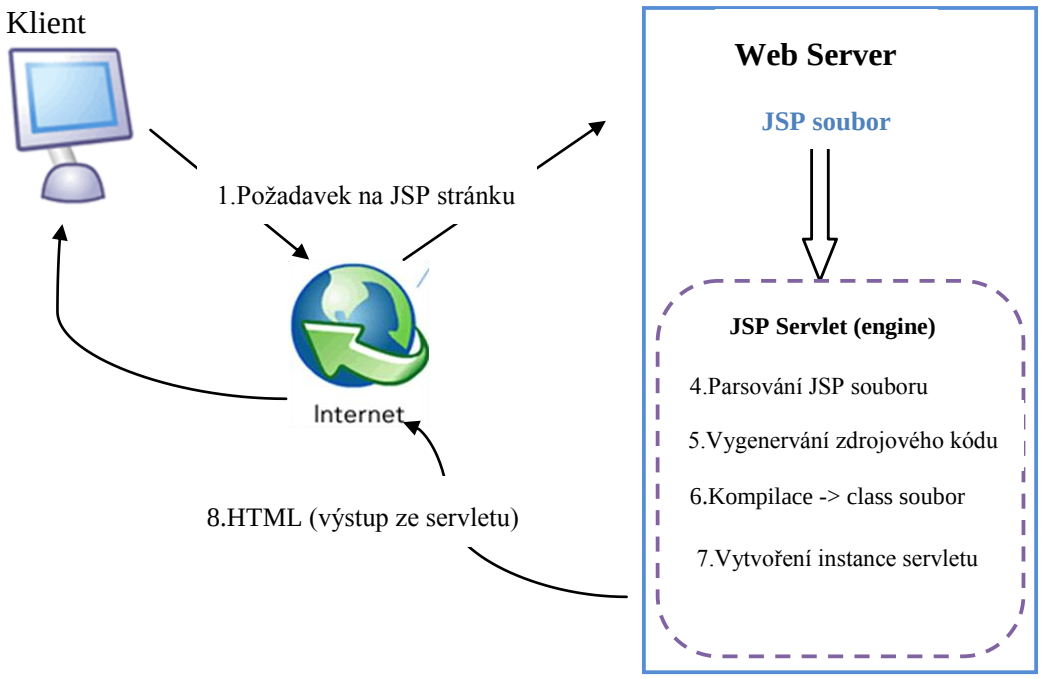
Princip vývoje dynamických webových stránek pomocí skriptovacích technologií je založen na jednoduchém vkládání částí kódu do HTML stránky. Tyto části kódu se zpracují parserem na webovém serveru, který na jejich základě vygeneruje skutečný obsah HTML stránky. Tento proces je mnohem méně efektivní, než kdyby se vykonával zkompilovaný kód.

Použití skriptovacích technologií ovšem není vhodné pro rozsáhlejší projekty, což je způsobeno chybějící pokročilou infrastrukturou a slabší podpora objektově orientovaného přístupu k vývoji. V poslední době však začínají vynikat a určitě i převažovat objektové technologie *J2EE* a *Microsoft .NET Framework*, nabízející robustnější architekturu a větší výkonnost než skriptovací jazyky. [9]

3.2 JavaServer Pages

JavaServer Pages (dále *JSP*) je technologie vyvinutá společností ***Sun Microsystems***, která poskytuje jednoduchou a rychlou cestu pro vytváření dynamického obsahu webových stránek na straně serveru. V praxi to vypadá tak, že na serveru jsou uloženy *JSP* soubory, které jsou vlastně HTML stránky, do nichž jsou vložené speciální tagy obsahující *Java* kód. Tento kód poté vytváří dynamický obsah stránky.

Princip činnosti:



3.1 Princip činnosti JSP

JSP a XML

Technologie *JSP* poskytuje řadu možností, které se hodí pro práci s XML. *JSP* stránky mohou obsahovat jakékoliv textové data, proto je jednoduché vytvářet dokumenty, které obsahují XML značky. *JSP* stránky mohou také využít plnou sílu platformy *JAVA* pro zpracování a transformaci XML zpráv a dokumentů. V aplikaci je možné použít parser s rozhranním *DOM* nebo *SAX*. Mezi nejznámější Java parsery dnes patří *Xerxes*, který byl vyvinut firmou IBM.

3.3 ASP.NET

Technologie ASP.NET je součástí ***.NET Frameworku*** pro tvorbu webových aplikací a služeb. Nabízí úplný objektově orientovaný model, který obsahuje architekturu řízenou událostmi, založenou na ovládacích prvcích, což umožňuje zapouzdřování kódu a jeho opětovné využívání. ASP.NET je založen na *CLR* (*Common Language Runtime*) prostředí, které je společné pro všechny aplikace postavené na *.NET Frameworku*. Projekty mohou být realizovány ve kterémkoliv programovacím jazyce podporujícím *CLR*. Hlavními podporovanými jazyky jsou C# a VB.NET. Vývoj a principy návrhu webové aplikace jsou velice podobné jako u klasického desktop provedení - koncept *ASP.NET WebForms*. Stránky jsou poskládány z objektů a serverových ovládacích prvků, které stejně jako v běžných „těžkých“ aplikacích mají své vlastnosti a na základě činnosti uživatele vyvolávají události, které jsou zpracovány na webovém serveru. Výhody technologie ASP.NET se dají shrnout do několika bodů:

1) ASP.NET je integrováno s *.NET Frameworkem*.
2) ASP.NET se neinterpretuje, ale kompiluje.
3) Možnost použití několika programovacích jazyků (pro které existuje překladač do *MSIL*).
4) Běh aplikace uvnitř společného runtime jazyků (Automatická správa paměti, typová bezpečnost, rozšiřitelná Metadata, strukturované zpracování chyb, *MultiThreading*)
5) Objektově orientovaná technologie.
6) Podpora různých zařízení a prohlížečů.
7) Snadné rozmístění a konfigurace.

ASP.NET a XML

Zpracování XML spíše než s ASP.NET souvisí s infrastrukturou *.NET Frameworku*, konkrétně *BCL* (*Base Class Library*), což je rozsáhlá sada tříd a knihoven. *.NET Framework* obsahuje celou řadu funkcí, datových typů a objektů pro práci s XML. Tyto funkce mohou používat všechny aplikace spuštěné na tomto frameworku. Tudíž je možné je používat i na stránkách ASP.NET. Další zajímavou možností je použití **LINQ** (od verze 3.5). LINQ je sada rozšíření pro jazyky C# a VB.NET, které umožňují manipulovat s daty v paměti pomocí dotazování, které je velice podobné SQL dotazům nad databází. LINQ v zásadě definuje asi 40 dotazovacích operátorů, jako jsou *select, from, where, orderby,* s jejichž pomocí je zakódován dotaz. Existují různé typy dat nad nimiž lze dá dotaz vykonat. Pro práci s XML je to **LINQ to XML**. Tento postup lze samozřejmě použít pouze když máme XML dokument uložený v paměti. Podrobnější informace ke zpracování XML pomocí této technologie jsou v kapitole 4.3.

3.4 ASP.NET AJAX

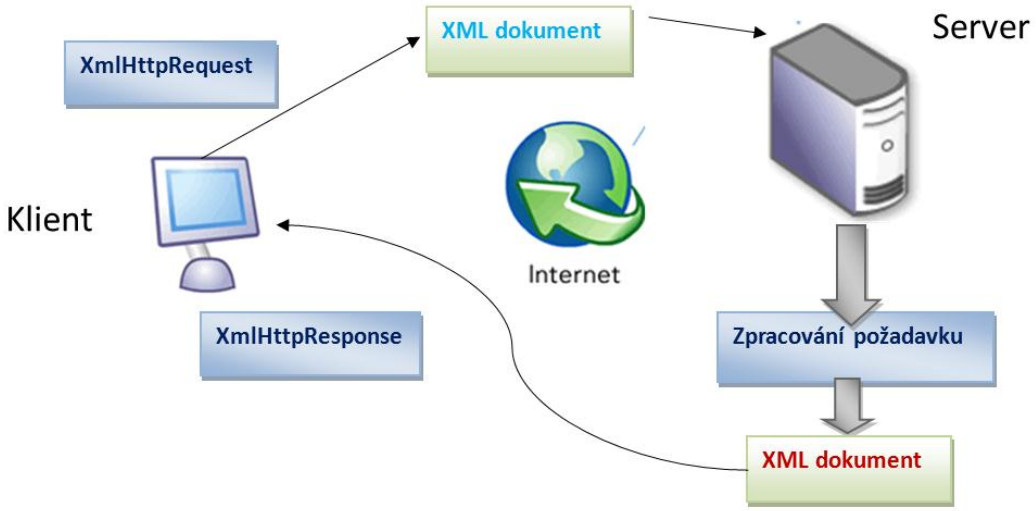
AJAX zkratka pro asynchronní *JavaScript* a XML (***A****synchronous* ***J****avaScript* ***A****nd* ***X****ML*). *AJAX* není sám o sobě novou technologií, ale jedná se o obecnou aplikaci JavaScriptu, která pro získání nových informací komunikuje na pozadí se serverem. Kód na straně klienta dostane tyto informace a na jejich základě provede nějaké akce. Hlavní výhodou tohoto přístupu je větší interaktivita stránky a přívětivější prostředí pro uživatele, protože pro získání a zobrazení nových informací není nutné obnovovat celou stránku. Nabídnou tak uživateli vetší flexibilitu a výhody desktop aplikací v podobě okamžité reakce na podněty a události uživatele.

Princip:

Standardně protokol HTML funguje takovým způsobem, že po akci uživatele, kterou může být například odeslání formuláře, klient odešle všechna data na server, na kterém běží webová aplikace, která dokáže komunikovat se světem přes http protokol. Po zpracování požadavku server znovu vygeneruje nový aktualizovaný HTML dokument a odešle zpět na klienta. S každým požadavkem se odesílají nejen data, ale i celé uživatelské rozhranní.

Obecná logika technologie *AJAX* spočívá na komplikovanějším postupu. Po akci uživatele, kterou může být stisk tlačítka, nám klientský *JavaScript* vygeneruje nějaký Xml

dokument, který obsahuje informace o tom, co se stalo. Například při stisku tlačítka pro přidání zboží do košíku bude Xml dokument obsahovat informace, jaké zboží chceme přidat a kolik kusů.



3.2 Princip AJAX

Tento XML dokument se za pomocí objektu *XmlHttpRequest* odešle na server. Toto odeslání pomocí tohoto objektu je pro uživatele „neviditelné“. To znamená, že uživatel neví o komunikaci na pozadí, protože internetový prohlížeč nedokáže rozpoznat tuto komunikaci. Když se klasicky odesílá formulář pomocí *POST*, tak posílání dat nebo stahování HTML stránky, je nějak graficky znázorněno (např. pomocí rotujícího kolečka, a tak podobně). V případě odeslání přes *XmlHttpRequest* tomu tak není. Na serveru je požadavek zpracován. Po zpracování je vygenerován opět XML dokument, obsahují informace o tom, co se změnilo, a je odeslán zpět na klienta. Pro pokračování z předchozího příkladu jím může být přidání zboží do košíku a aktuální celková cena. Klientský skript obdrží tento dokument a je tímto skriptem nějak interpretován a pomocí *DHTML* a práce s *DOM* je promítnut do zobrazení stránky.

ASP.NET AJAX:

Bohatý obsah a vyspělost serverových ovládacích prvků se nedá srovnat s poněkud omezenějším *HTML DOM*. Napsání bezpečného kódu, bez možnosti ladění, *intelisence* a dalších výhod vývojového nástroje Visual Studio, je velice složité. Dalším problémem je odlišnost dnešních prohlížečů, protože mají menší odlišnosti v implementaci.

Částečně lze tyto problémy vyřešit funkcionalitou zpětného volání ASP.NET, která poskytuje serverový model, s jehož pomocí lze snadno vygenerovat potřebný kód na straně klienta. Tento model zpětného volání má ale řadu nevýhod. Rozhranní vypadá neohrabaně a implementace ve stránce není moc vhodná pro přehlednost. Mezi nevíhody rovněž patří nutnost vymyslet, jak data serializovat a deserializovat na straně klienta, aby je bylo možné odeslat respektive přijmout ve formě řetězce.

Možností jak tyto problémy obejít je využít ASP.NET AJAX. Použití této sady nástrojů, poskytující funkcionalitu, nám pomůže při tvorbě ajaxových stránek bez velkého úsilí.

ASP.NET AJAX je složen ze 2 základních částí. Část na straně klienta a část na straně serveru. Klientská část je složena z knihoven *JavaScriptu*. Ty nejsou nijak provázány s ASP.NET a proto je lze používat i na jiných technologií než je ASP.NET. JavaScriptové knihovny obsahují základ, který je nutný při využití ASP.NET AJAX. Patří jsem rozšíření jazyka *JavaScript,* aby vyplnil jeho mezery, jako jsou například dědičnost, a poskytuje základní infrastrukturu.

Serverová část obsahuje ovládací prvky a komponenty, které využívají právě tyto knihovny *JavaScriptu*. Příkladem by mohl být panel na formuláři, který lze volně přetahovat po okně prohlížeče. V pozadí toho procesu běží *JavaScriptový* kód, který využívá výše zmiňované knihovny na straně klienta. Vytvoření potřebného klientského skriptu má na starosti komponenta na straně serveru. Tím tak odpadá složité psaní kódu. [1]

3.5 Shrnutí

Slabiny webových aplikací jsou nejvíce evidentní v nedostatečné možnosti reagovat na podněty (responsiveness). Veškerou práci je totiž nutné provádět na straně webového serveru. Pokaždé, když dojde ve stránce k nějaké akci, potřebuje prohlížeč odeslat nějaká data spolu s požadavkem na server, získat novou kopii stránky a aktualizovat zobrazení. Tento proces je rychlý, ale i tak znatelně narušuje hladký běh činnosti s aplikací. Protože tento proces trvá určitou dobu je prakticky nemožné reagovat na události, které se vyvolávají velmi často jako například pohyb myši, úhozy na klávesnici. Nabízeným řešením v případě webových aplikací mohou být technologie, které zpracovávají události a reagují na podněty uživatele na straně klienta. Jednou z variant je použití techniky AJAX, která dovoluje asynchroně volat server, kde jsou zpracovány požadavky. Druhou ze zvažovaných možností je technologie Silverlight, která pro své výhody rozhodla o volbě technologie pro Web Xml Editor.

4 Silverlight

Ačkoliv je web nejpopulárnějším prostředkem pro firemní software, existují určité věci, které webové aplikace prostě dělat neumějí nebo je neumějí dělat podle našich představ. I když jsou webové stránky vybaveny nejnovějšími výkřiky JavaScriptu, přesto není možné duplikovat mnohé ze schopností, které se u desktopových aplikací považují za samozdřejmé. I když je pomocí JavaScriptu možné reagovat na straně klienta na jisté akce, není možné budovat složitá uživatelská rozhranní, která by byla vybavená a vstřícná jako okno v bohaté klientské aplikaci. Výhody, které s sebou přináší webové aplikace (široká kompatibilita, vysoká bezpečnost, nízké náklady na distribuci, apod.), často vyváží ztrátu vzniklou nedostatečným komfortem pro prostředí webového prohlížeče. Limity webu jsou ale neustále vyvojáři překračovány. V současné době není vůbec neobvyklé například sledování animované reklamy. Taková výbava není přímou součástí obyčejných standardů HTML, CSS a JavaScriptu. Vyřešení toho problému spočívá v nainstalování *plug-inu* do prohlížeče.

Nový silverlight od společnosti Microsoft je multiplatformní implementace *.NET Frameworku* určená pro tvorbu aplikací *RIA* a mediálních prezentací na webu. Podobně jako Flash je i Silverlight technologií, která umožňuje vytvářet interaktivní obsah, který je spuštěný skrz webový prohlížeč na straně klienta. Podporuje dynamickou grafiku, média a animace, které dalece přesahují možnosti HTML. Využití této technologie je možné najít i při tvorbě webových aplikací. Silverlight je zároveň také „cross-browser“ technologií, což znamená, že by měl být podporován všemi verzemi internetových prohlížečů. Tato vlastnost je samozdřejmě spojena a limitována s operačním systémem ***Windows***. Pro uživatele operačního systému ***Linux*** ovšem existuje řešení v podobě projektu nazvaného *Moonlight*, který je open source implementací Silverlightu primárně určených pro Linux a ostatních systémech založených na *UNIX/X11*.[8]

Silverlight a ASP.NET

Mezi těmito technologiemi existuje jistá spojitost. Obě využívají platformu *.NET*. **Silverlight** je spojen s .NET framework 3.0 a 3.5. Je zde možné zavézt jistou analogii. Stejně jako systém **Windows Forms** (Windows XP a starší) přešel **WPF** – *Windows Prezentation Foundation* (Windows Vista, Windows 7), tak z technologie ASP.NET využívající grafické knihovny Windows Forms, vznikla technologie Silverlight, která využívá knihovny WPF.

Hlavní rozdíl mezi ASP.NET a Silverlight spočívá ve způsobu komunikace uživatele se serverem. V ASP.NET uživatel webové aplikace posílá požadavky na jednotlivé stránky a server zpracuje požadavek, vyrenderuje novou HTML stránku při každém vyžádání uživatele. Při využití Silverlightu se s webovou aplikací pracuje jako s celistvou aplikací, přes kterou se odesílá na server pouze dílčí požadavky na informace. Server odešle uživateli příslušná data a je na aplikaci spuštěnou v prohlížeči, aby požadavky zpracovala sama. Důležitým faktem je, že se data zpracovávají na straně uživatele. Tím je i zátěž při zpracování dat převedena ze serveru na uživatelský počítač.

4.1 Způsob tvorby Silverlight aplikace

Technologie Silverlight je založena na kombinaci jazyka XAML se standardními .NET jazyky. **XAML** zajišťuje grafický návrh a design aplikace. Funkčnost a obsluha událostí je popsána pomocí kteréhokoliv .NET jazyka, nejčastěji **C#.**

4.1.1 XAML

Koncepčně je XAML jazyk jistou analogií k HTML. Prostřednictvím HTML definujeme prvky, které dohromady tvoří obyčejnou HTML stránku. XAML umožňuje definovat prvky, které dohromady tvoří blok obsahu XAML. Při manipulaci s prvky HTML je použit JavaScript na straně klienta (nebo je opakovaně stránka renderována serverem, při práci s ASP.NET a odesílnáním stránky na server). S prvky XAML je manipulováno s klientským kódem napsaným v některých z programovacích jazyků .NET. Nejčastěji pomocí C#. Stejně jako XHTML, tak i jazyk XAML je založen na XML. Prvky lze vnořovat v jakémkoliv uspořádání, čímž je specifikováno zařazení v nějakém rámci.

4.1.2 XAML a kód v pozadí

Aby byla zajištěna funkčnost aplikace je potřeba napojit kód pro obsluhu událostí na XAML objekty. Vývojový nástroj automaticky vytvoří pro soubor *Page.xaml* i třídu kódu v pozadí s názvem *Page.xaml.cs* (přípona cs závisí na zvoleném programovacím jazyku). Třída obsahuje výchozí konstruktor *InitializeComponent(),* pro vytvoření instance Silverlight stránky. V tomto konstruktoru jsou analyzovány použité značky, vytvořeny odpovídající objekty, provedeno nastavení jejich vlastností a přiřazení definovaných událostí. Nelze

opomenout důležitý atribut XAML objektů, Name, který slouží pro identifikaci a pomocí něho lze manipulovat s ovládacími prvky programátorsky.

Tento model je podobný vývoji webové ASP.NET stránky, ale konstrukce jsou zcela odlišné. Značkování XAML je analyzováno enginem Silverlightu na straně klienta pomocí zredukované verze *CLR* prostředí. Konečný obsah je vložen do stránky pomocí specializovaného ovládacího prvku Silverlightu. Oproti tomu engine ASP.NET zpracovává stránku na serveru a ne na klientovi.

Programovací modely

Silverlight aplikace využívají řízený (*managed*) kód, který běží na úrovni vrstvy *CLR*. Je možné využívat kompilované programovací jazyky Visual Basic .NET a C# přes řízené API, nebo dynamické jazyky jako například IronPython nebo IronRuby. Pro tyto jazyky je k dispozici Silverlight Dynamic Languages SDK. Kvůli špatné kompatibilitě se Silverlight 1.0, který nevyužíval .NET jazyky, ale jen JavaScript je k dispozici JavaScript API pro Silverlight. Při využití tohoto modelu je kód interpretovaný na úrovni webového prohlížeče.

4.1.3 Vývojové nástroje

Jak již bylo napsáno, silverlight aplikace lze vyvýjet v jakémkoliv .NET programovacím jazyce. Tím jsou specifikovány i vývojové nástroje, které musí navíc spolupracovat se Silverlight, podporovat ***Silverlight CoreCLR*** pro hostování aplikace. Microsoft nabízí produkt **Expression Blend**, který je nyní ve verzi 3. Ten slouží převážně pro grafický návrh aplikace, zhotovení aplikací a celkovému návrhu v jazyce XAML. Další vývojářským produktem ze stejné dílny je **Visual Studio.** Expression Blend lze použít i pro programování funkcionality v jazyce C#, ale Visual Studio nabízí větší komfort a možnosti pro psaní kódu. Pro vytvoření Silverlight projektu a kompilaci je potřeba do Visual Studia doinstalovat balíček Silverlight Tools for Visual Studio. Ve spojení s vydáním Silverlight 2 je možné využít i konkurenční nástroj **Eclipse**.

4.2 Prostředí .NET a XML

.NET Framework nabízí bohatou sadu voleb pro práci s XML. Patří sem třídy pro zpracování XML založené na proudech, třídy pro manipulaci s XML daty v paměti a také webové ovládací prvky s nimiž lze kód XML zobrazovat a vázat. Zpracování XML je obsahem kapitoly 4.3.

Od verze .NET 3.5 se k API XML připojila nová skupina – **LINQ**. Tato technologie je sadou rozšíření jazyka, která umožňuje vykonávat dotazy v jazyce C# (nebo VB.NET). Přesněji LINQ definuje klíčová slova jimiž je možné tyto dotazy budovat. Tyto výrazy mohou data vybírat, filtrovat, řadit, seskupovat a transformovat.

Jednotlivá rozšíření LINQ:

- LINQ to Objects
- LINQ to Dataset
- LINQ to SQL
- LINQ to XML

Základy LINQ

Základní představa o práci LINQ se dostane pomocí vyložení logiky, jak pracuje s kolekcemi uloženými v paměti. Nejjednodušší a nejobecnější formou je LINQ to Object, které v podstatě umožňuje nahradit iterační logiku (jako blok cyklu foreach) deklarativním výrazem LINQ. Vykonávání dotazů je závislé na typu dotazovaných dat. LINQ to SQL transformuje dotaz na databázové T-SQL příkazy. S tím souvisí i další podkladové konstrukce jako otevření spojení do databáze, vykonání dotazu a získání návratových dat.

V LINQ je jedna důležitá symetrie. Výrazy LINQ pracují s objekty, implementující rozhranní *IEnumerable<T>* a zároveň výsledkem dotazu je objekt, který rovněž implementuje toto rozhranní. To znamená, že výsledek jednoho dotazu LINQ, může být použit jako zdroj dat pro další jiný výraz LINQ. Řetěz výrazů LINQ se vyhodnocuje až na konci, když se iteruje přes finální data. V závislosti na typu dat, na něž je dotazováno, je LINQ schopno často sloučit řetězec výrazů dohromady, do jediné operace, takže je dosaženo nejefektivnějšího způsobu.

Spolu s integrací LINQ je jazyk C# 3.0 obohacen o další novinky, které zlepšují efektivitu vývoje a zjednodušují psaní LINQ dotazů. Mezi tyto nové vlastnosti patří **automatické odvození typu proměnné**. Přibylo nové klíčové slovo *var*, které se použije místo určení typu proměnné.

```
var promenna = 3;
if (promenna > 2)
promenna++;

// při tomto přiřazení kompilátor hlásí chybu
// - zachování silné typovosti objektu
promenna = "chyba";
```

Příklad 4: Použití klíčového slova var

Je ovšem zachována silná typovost jazyka, protože typ proměnné je odvozen v době kompilace. Této vlastnosti jazyka je nejvíce využíváno při získávání výsledků z LINQ dotazu. Není potřeba znát typ proměnné, protože o dosazení se postará právě kompilátor.

Další z novinek, které je možné využít při tvoření dotazů LINQ je **inicializátor objektů** a kolekcí. Díky těmto inicializátorům je možné už v době tvorby objektu nebo kolekce jednoduše definovat hodnotu veřejných vlastností respektive v případě kolekce určit hodnoty prvků.

```
Osoba newOsoba = new Osoba() { jmeno = "Petr", vek = 3 };
```

Příklad 5: Inicializátor objektů

Za výhodu se může považovat i estetická funkce – kód zabere pouze jeden řádek a je lépe čitelný. Při využití tohoto přístupu není nutné u objektů tvořit speciální konstruktor pro inicializaci.

Anonymní typy, to je další z užitečných vlastností nového C# 3.0. Díky nim lze definovat typ bez jména, určit jeho veřejné vlastnosti, a použít jeho instanci. Hodnoty těchto vlastností se definují při vytváření tohoto anonymního typu a jsou určené pouze pro čtení. Použití nachází tato novinka v projekci vytvořené pomocí LINQ. Jelikož tyto typy nemají jméno je nutné použitá klíčového slova *var*.

```
var mojePromenna = new { Jmeno = "Petr", vek = 20};
Console.WriteLine("Vek osoby je " + mojePromenna.vek);
```

Příklad 6: Anonymní typy

Výhodou těchto novinek je, že nemění *CIL* kód. To znamená, že kód, který vzniká ze C# 3.0 je stejný jako kdyby vznikl na jeho předchůdcích (C# 2.0), protože změny se týkají pouze vylepšením kompilátoru. Nedochází ke změnám ve specikaci a dalších důležitých částech .NET Framework. [4]

Výrazy LINQ

Podobnost LINQ výrazů je vzhledově srovnatelná dotazům SQL, nicméně klauzule jsou vzájemně proházené. Všechny výrazy musejí mít klauzuli *from*, která vyjadřuje zdroj dat, a *select*, jež vyjadřuje data, které je potřeba získat.

```
matches = from zamestnanec in zamestnanci
                                ...;
```

Příklad 7: ukázka LINQ

Klauzule from specifikuje dvě důležité informace. Slovo za in určuje zdroj dat. V ukázkovém příkladu je to objekt kolekce s názvem zamestnanci, která obsahuje instance jednotlivých detailů o zaměstnancích. Slovo následující za *from* přiřazuje alias, který repreznetuje jednotlivé prvky ze zdroje dat - z kolekce zamestnanci. Tento alias se později používá při sestavování dalších částí výrazu, například pro filtrování.

Jazyk C# obsahuje mnoho dalších operátorů. Nejdůležitějšími pro potřeby webového vývoje jsou *select*, *where*, *orderby* a *group*. Funkce těchto operátorů je vysvětlena u konkrétních použití.

Projekce

Schopnost, která umožňuje transformovat data, na něž je dotazováno, do výsledků s jinou strukturou se nazývá projekce. Pro získání pouze podmnožiny dat se využije klazule *select*. Například pro vybrání pouze křetsních jmen zaměstnanců uložených v kolekci, by výraz vypadal podle ukázky:

```
IEnumerable<string> matches;
matches = from zamestnanec in zamestnanci
          select zamestnanec.jmeno;
```

Příklad 8: Projekce

Při projekci je často využíváno dynamicky definovaných tříd, které obalí pouze ty informace, které chceme získat. Pro získání křestního jména a přijmení uložených v oddělených řetězcích, je nejvhodnější zvolit funkcionalitu C# známou pod názvem anonymní typy. Do klauzule *select* se přidá klíčové slovo *new*.

```
var matches = from zamestnanec in zamestnanci
              select new { jmeno = zamestnanec.jmeno,
             prijmeno = zamestnanec.prijmeni };
```

Příklad 9: Projekce s anonymním typem

Po vykonání tohoto dotazu bude vrácena sada objektů, které používají implicitně vytvořenou třídu. Definici třídy vytvoří kompilátor a přidělí ji automaticky generovaný název. Tuto třídu lze lokálně využívat, číst její vytvořené vlastnosti a využít ji jako zdroj pro vázaní dat.

Filtrování a řazení

Za pomoci klauzule where se dají výsledky filtrovat, aby obsahovali jen ty, které vyhovují zadané podmínce. Příklad ukazuje kód pro výběr zaměstance, kde jeho jméno začíná konkrétním zadaným písmenem.

```
var matches = from zamestnanec in zamestnanci
              where zamestnanec.jmeno.StartsWith("P")
              select zamestnanec;
```

Příklad 10: Filtrování a řazení

Klauzule *where* vezme podmínkový výraz a vyhodnotí ho pro každý prvek. Je-li výsledek vyhodnocen *true* (pravda), prvek je zařazen do výsledků. Podmínkové výrazy se dají kombinovat nejenom s logickými operátory AND (&&) a OR (||), ale tak s relačními operátory (<, <=, >, >=).

Zajímavým rysem výrazů LINQ je to, že je možné zavolat vlastní metodu přímo (*inline*). Tato vlastní metoda obsahuje vlastní vyhodnocovací kód a musí vracet hodnotu *true* nebo *false*.

Podle syntaxe příkazu SELECT v SQL byl také namodelován operátor *orderby*. Uvedením hodnoty nebo seznamu hodnot, oddělených čárkami, lze výsledky setřídit. Pro třídění v opačném opařadí slouží slovo *descending*, které se uvádí za název prvního sloupce. [1]

4.3 Zpracování XML v .NET Framework pro Silverlight

Zpracování XML v Silverlight aplikaci je trochu odlišné od Verze Silverlight .NET Framework obsahuje podmnožinu klasické sady třídy .NET Framework a několik přidaných typů, které nejsou součástí klasické sady tříd.

V technologii Silverlight existují 2 přístupy, jak parsovat XML data. Prvním je použití třídy *XmlReader* nebo novější *LINQ to XML* (zpracování XML v paměti).

XmlReader je způsob založený na proudu a proto redukuje nároky na paměť a je rychlejší čili efektivnější. Pokud ovšem je potřeba s XML dokumentem pracovat na složitějších úlohách, které jsou náročné na čas, je lepší dát přednost druhému přístupu práce

se zpracováním v paměti. Zkrátí se tím doba, po kterou přistupujete k souboru XML a dáváte tím možnost ostatním uživatelům, kteří chtějí s XML dokumentem pracovat. Přečíst soubor XML s tímto objektem je nejjednodušší, ale přináší s sebou méně flexibilní přístup. Soubor se čte sekvenčně a není zde možnost volně se pouvat po uzlech dokumentu (rodičovské, dceřiné, sourozenecké uzly). Tento problém řeší druhý přistup, zpracování XML v paměti.

Obecně by se dalo říci, že XmlReader se využívá v případech, kdy se načítají rozsáhlé dokumenty, protože načtení zabere velké místo v paměti a snižuje výkon. Když je použit systém, který pracuje s menšími XML dokumenty je vhodné použít výhod, které LINQ to XML nabízí.

Zpracování XML schémat a validace dokumentu je pro technologii Silverlight **slabinou** stejně tak jako *XSLT*, protože v současné době nejsou podporovány. Důvodem může být zredukování velikosti .NET Framework pro Silverlight. Jakýkoliv z těchto nedostatků potřebné absence třídy lze vyřešit zpracováním na webovém serveru, který obsahuje plnohodnotný .NET Framework, s využitím **WCF** pro komunikaci.

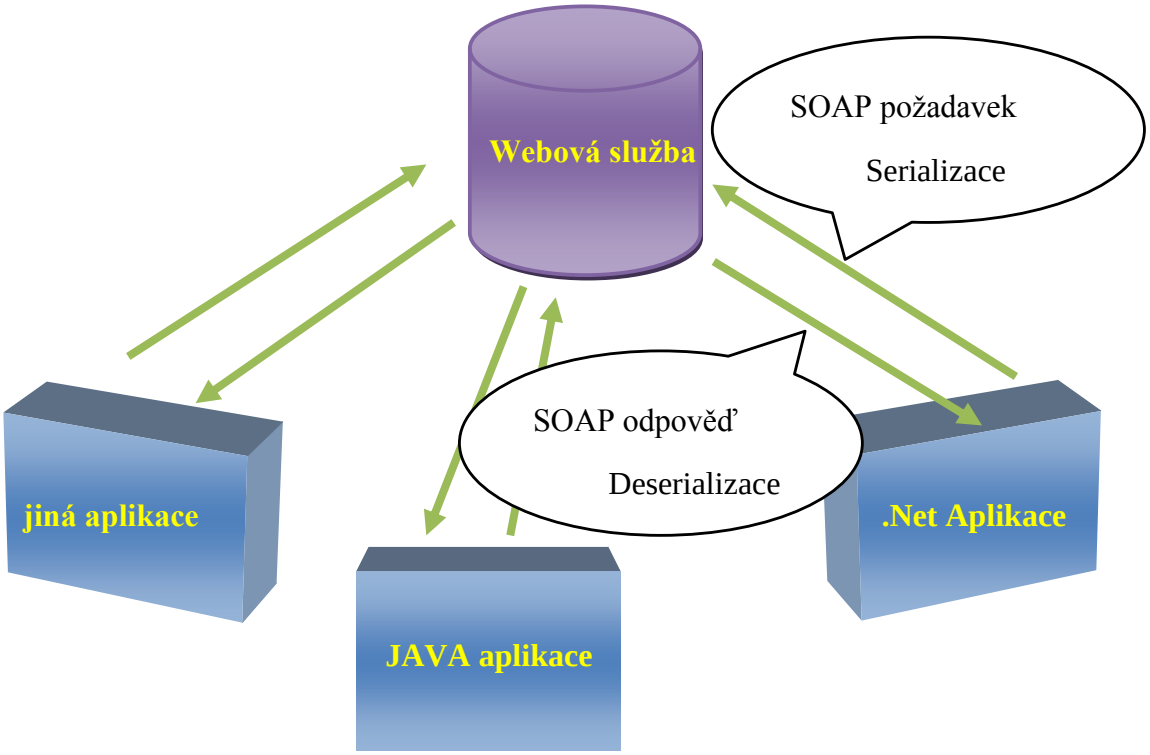
4.4 Technologie WCF – Windows Communication Foundation

WCF neboli *Windows Communication Foundation* je robustní technologie, která se zrodila společně i s dalšími (WPF, WF, CardSpace) při uvedení .NET Framework 3.0. Tato technologie umožňuje tvorbu SOA (*Service-Oriented Architecture*, servisně orientovaná architektura) aplikací prakticky na libovolném komunikačním protokolu. Sjednucuje již dříve dostupné technologie určené pro komunikaci - *.NET remoting*, webové služby (*web services*) a další. Obecně poskytuje komunikaci respektive výměnu dat aplikace s aplikací (služby s klientem). WCF slouží jako distribuovaná aplikce nejčastěji v podobě jako server (WCF služba) a klient(aplikace napsaná v .NET jazyce).

Základním stavebním prvkem WCF aplikace je služba. Služba je systém, poskytující jeden nebo více koncových bodů (endpointů), které slouží pro příjem a odeslání *SOAP* zpráv. *SOAP* je multiplatformní standard, který je používán pro výměnu zpráv založených na formátu XML zasílaných mezi serverem a klientem. SOAP vyniká díky jeho flexibilitě, protože je možné ho použít pro zasílání na jiných transportních protokolech než klasické

HTTP. Služba navíc poskytuje metadata, sloužící pro popis služby. Metadata obsahují potřebné informace pro konfiguraci klienta. [3]

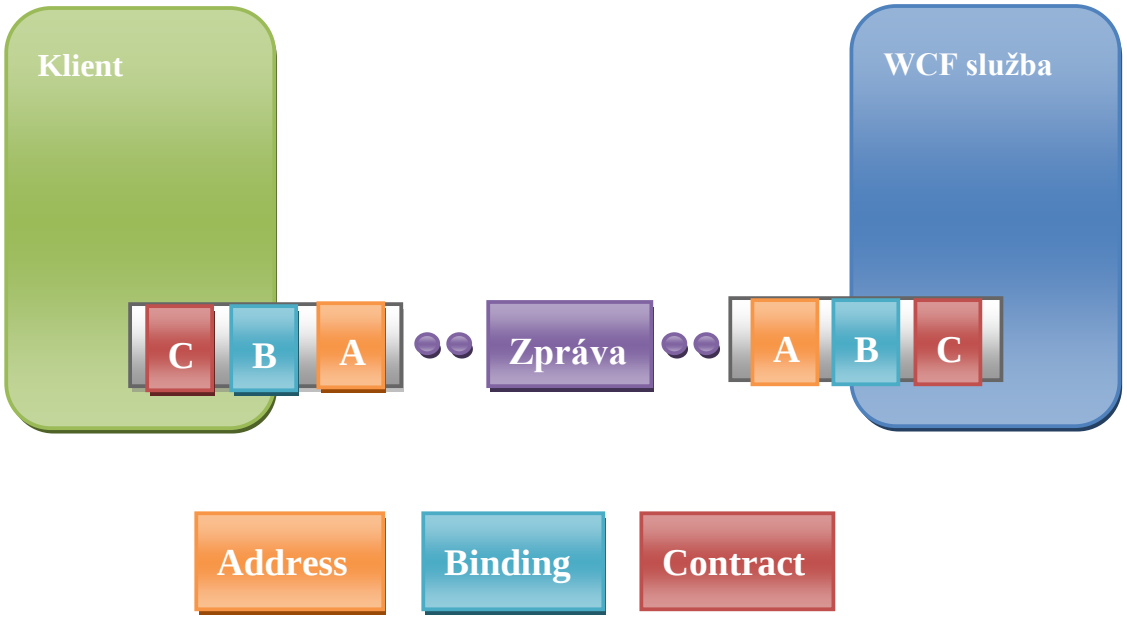
Komunikace s webovou službou:



4.1 Komunikace s webovou službou

Webová služba komunikuje na základě zasílání zpráv a proto se službou může komunikovat aplikace na různých platformách a napsaná v jakémkoliv jazyce.

Podobně WCF služba je ve skutečnosti objekt, obsahující sadu operací, které nabízí svým klientům pomocí **endpointu**.



4.2 Komunikace s WCF

Endpoint je definován pomocí:

- **Address** – Informace o tom, na které adrese je služba dostupná. Tvar adresy je závislí na použitém protokolu. Např.
  - http://localhost/Service.svc
  - https://localhost/Service.svc
  - net.tcp://localhost:8090/Service
- **Binding** – Popisuje jakým způsobem se může klient připojit a jaký formát dat je očekáván ze služby. Binding zahrnuje následující informace:
  - Protokol
    - HTTP, HTTPS, TCP
    - *Named pipes*
    - *Message Queues*
  - Kódování zpráv
    - XML (standartně využíváno)
    - Binární kódování (Obrázky, Stream)
  - Bezpečnost
    - *Message Security* – zabezpečení na úrovni zpráv
    - *Transport Security*
  - Transakce (metoda může být prováděna v rámci transakce)

    - Spolehlivost – Využití protokolu, který zaručí doručení všech zpráv. Umožňuje obnovení komunikace při přerušení.
- **Contract** – *Service contract* = rozhranní, které definuje operace dostupné pro klienty.

Contract se definuje pomocí klasického rozhraní, které je navíc označeno atributem *ServiceContract*. Aby byli jednotlivé metody publikovány pro klienty, musí se označit atributem *OperationContract*. Pokud tímto atributem metoda označena není, klient ji nevidí.

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    string GetData(int value);
```

Příklad 11: Definice rozhranní WCF

Funčnost se definuje pomocí třídy, která vytvořené rozhranní implementuje.

```
public class Service1 : IService1
{

    public string GetData(int value)
    {
        return string.Format("You entered: {0}", value);
    }
```

Příklad 12: Definice třídy WCF

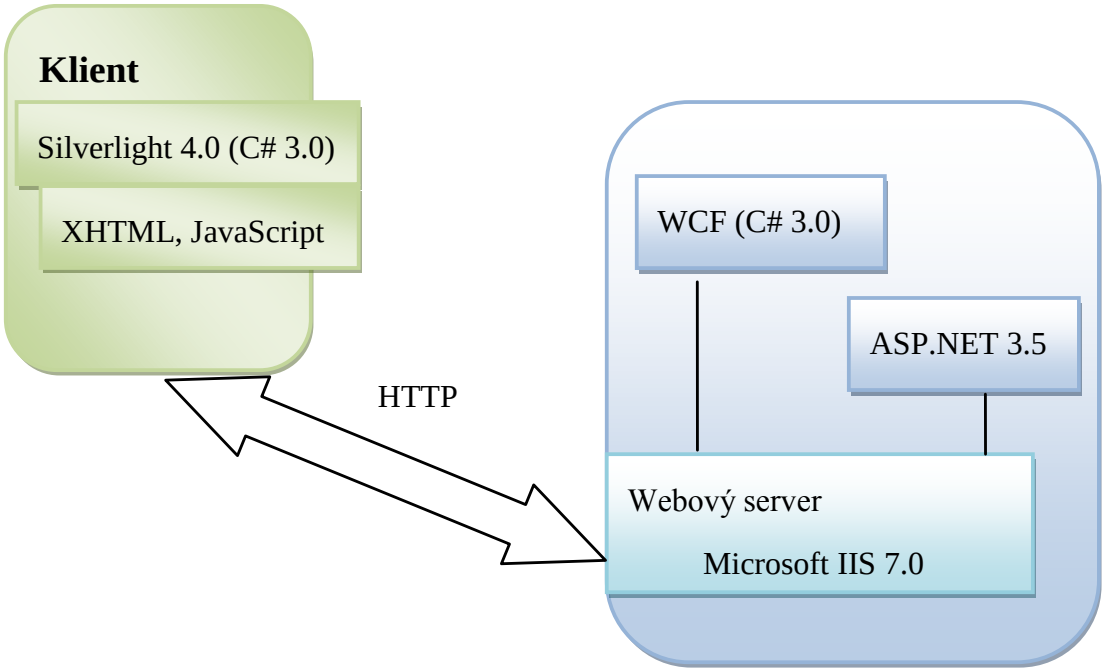
Pro publikaci WCF služby je navíc nutné provézt konfiguraci. Konfigurace je stejně jako v klasickém ASP.NET provedena v souboru *Web.config* (popřípadě *App.config*).

5 Vývoj aplikace

Tato kapitola popisuje celkový průběh vývoje webové aplikace, která nese název Web XML Editor. Dále jsou popsány problémy, které se vyskytli při jejím vývoji a jejich řešení s využitím možností dosavadních technologií. Pro návrh a celkové zhotovení aplikace je využito vývojářského nástroje Visual Studio 2010 (ve verzi beta, oficiání Release 20. dubna 2010). Pro grafické zpracování uživatelského prostředí je použit nástroj Microsoft Expression Blend 3, flexibilní a grafické vývojové prostředí. Pomáhá při návrhu moderních a vizuálně propracovaných aplikací s interaktivní podporou 3D zobrazování a zobrazování multimedií. Pro návrh posloužilo při vytvoření prezentační vrstvy webové aplikace. Není ovšem zcela nezbytně nutné použít tento nástroj, primárně určený pro návrh vzhledu aplikace.

5.1 Architektura aplikace

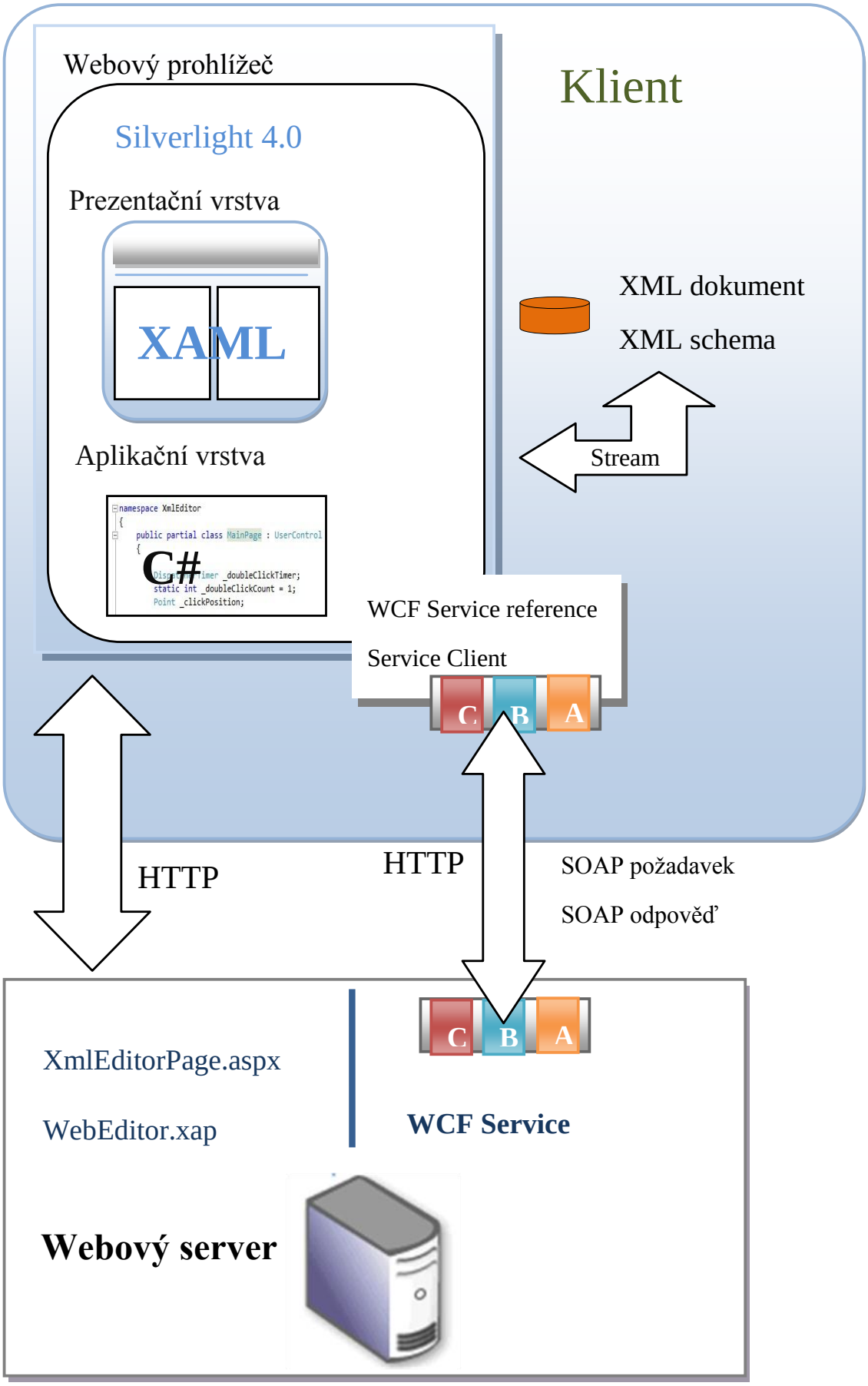
Aplikace z hlediska použitých technologií:



5.1 Aplikace z hlediska použitých technologií

Webová aplikace je dostupná prostřednictvím webového prohlížeče, který je samozdřejmostí v operačním systému. Navíc musí být do prohlížeče doinstalován doplněk pro zobrazování Silverlightu. Doinstalování je zajištěno pomocí úvodní stránky (*XmlEditorPage.aspx*), která hostuje Silverlight aplikaci. Při zjištění nepřitomnosti doplňku je uživatel přesměrován na informační stránku a vyzván k instalaci. Zhotovená Silverlight aplikace je zabalena do souboru s příponou xap (*WebEditor.xap*) a s požadavkem uživatele je spolu s úvodní HTML stránkou stažena do klientského počítače, kde je spuštěna pomocí již nainstalovaného doplňku, obsahující prostředí .NET Framework určený pro Silverlight. Server není dále zatěžován dalšími požadavky. Silverlight je rozdělen na 2 základní části: Prezentační vrstva obsahuje komponenty a služby orientované na generování uživatelského rozhranní a interakci s uživatelem. Uživatelské rozhranní může obsahovat využívat renderování vektorové i bitmapové grafiky, textový výstup a animace. Aplikační vrstva zahrnuje obsluhu událostí generovaných uživatelem pomocí myši nebo klávesnice.

Navíc webová aplikace obsahuje referenci na WCF službu, která aplikaci dovoluje se službou komunikovat. Tato komunikace probíhá na základě vytvoření instance *proxy* třídy *Service Client*, která obsahuje všechny metody a vlastnosti dostupné z endpointu WCF služby.



5.2 Architektura aplikace

5.2 Návrh prezentační vrstvy

Každou aplikaci tvoří minimálně aplikační a prezentační vrstva Úlohou prezentační vrstvy je zobrazení dat z aplikační vrstvy ve vhodné formě, aby měli co největší informační podobu pro uživatele.

GUI (*Graphical User Interface* - uživatelské rozhraní) je navrženo tak, aby co nejvíce usnadňovalo práci s webovou aplikací a poskytovalo požadované informace. To je základním požadavkem pro design.

5.3 Návrh aplikační vrstvy

Úlohou aplikační vrstvy je pracovat s údaji, daty a připravit je pro zobrazení uživateli pomocí prezentační vrstvy. Složení aplikační vrstvy je z hlavní třídy *MainPage.cs*, ve které jsou zpracovány události objektů Silverlight stránky.

5.3.1 Základní funkce editoru

U každého editoru, ať už se jedná o textový či v tomto případě Xml editor, jsou očekávány funkce pro uložení, načtení z lokálního disku a vytvoření nového dokumentu. V Silverlightu jsou pro tyto úlohy k dispozici file dialogy. Při otevření respektive uložení jakéhokoliv obecného dokumentu je nutné upravit obsah Streamu do podoby pro objekt, který bude obsah zobrazovat, obdobně získat stream z objektu pro uložení. Pro tuto práci je navržena třída *DocumentPersister*.

Otevření Souboru

Pro zobrazení okna s výběrem souboru pro načtení je nutné vytvořit objekt *OpenFileDialog*. Možná základní nastavení tohoto dialogového okna jsou v podobě filtru pro výběr souborů (určení konkrétních přípon souborů) a umožnění výběru více souborů najednou.

```
private void btnOpen_Click(object sender, RoutedEventArgs e)
{

    OpenFileDialog ofd = new OpenFileDialog();
    ofd.Multiselect = false;
    ofd.Filter = "Xml Files|*.xml|All Files|*.*";

    if (ofd.ShowDialog().Value)
    {
        FileInfo fi = ofd.File;
        StreamReader stream = fi.OpenText();
        DocumentPersister.parseLoadFile(stream, rta.Blocks);

        stream.Close();
    }
}
```

Příklad 13: Otevření souboru

Uložení Souboru

```
private void btnSave_Click(object sender, RoutedEventArgs e)
{


    StringBuilder sb = DocumentPersister.prepareDocumentXML(rta.Blocks);

    if (sb == null)
    {
        MessageBox.Show("Saving documents with images is not supported");
        return;
    }

    SaveFileDialog sfd = new SaveFileDialog();
    sfd.DefaultExt = ".xml";
    sfd.Filter = "Xml Files|*.xml|All Files|*.*";

    if (sfd.ShowDialog().Value)
    {
        using (FileStream fs = (FileStream)sfd.OpenFile())
        {
            System.Text.UTF8Encoding enc = new System.Text.UTF8Encoding();
            byte[] buffer = enc.GetBytes(sb.ToString());
            fs.Write(buffer, 0, buffer.Length);
            fs.Close();
            IsDirty = false;
        }
    }

}
```

Příklad 14: Uložení souboru

5.3.2 Validace XML dokumentu

Validace je proces ověřování shody XML dokumentu se schématem. Umožňuje odhalit nekonzistence dat, které by později mohli vadit při dalším zpracování. Dostane-li aplikace plně ověřený dokument, má zaručenu jeho správnost. Korektnost XML dokumentů lze měřit podle dvou odlišných a navzájem se doplňujících metrik. Jedná se o správnost struktury XML (*well-formed XML*) a platnost dokumentu. Správnost struktury se vztahuje na celkovou syntaktickou analýzu. Platnost je rozbor na nižší úrovni a zahrnuje sémantiku dokumentu, která musí odpovídat uživatelsky definované struktuře. Ověřování platnosti je sice důležitou součástí, hlavně v aplikacích, kde dochází k výměně dokumentů, ale zároveň však znamená zvýšenou režii celé aplikace, protože kontrola dokumentů znamená vyčkat na dokončení analýzy všech podstatných uzlů.

Pro kontrolu syntaxe XML je v aplikaci použita třída XDocument, která načítá XML dokument do paměti, protože daný dokument je dále používán. Ověřování chyb je programově vyřešeno přes blok, který slouží pro odchytávání a zpracovávání výjimek. Případné chyby respektive správnost jsou uživateli zobrazeny na spodní informační liště.

Správnost zpracovávaného dokumentu ověřuje třída *XmlReader*, ale ta nedovoluje další podrobnější analýzu. Pro tyto účely je potřeba použít *XmlValidatingReader*. Tyto dvě třídy jsou zástupci metody zpracování pomocí Readerů, které jsou popsány v kapitole 2.6.1. Pro ověření správnosti dokumentu a provedení hlubší analýzy nemá Silverlight dostatečné množství tříd. Proto je nutné najít jiné vhodně řešení, které by poskytovalo lepší zázemí pro práci s XML. Propojením technologií Silverlight a WCF dostaneme elegantní způsob, jak obejít tyto nedostatky. Na server, kde je vytvořená WCF služba, odešleme XML dokument spolu se schématem a provedeme zpracování. Je-li potřeba znát podrobné údaje o chybě, ke kterým dojde během ověřování, je potřeba definovat událostní rutinu a předat odkaz na ni ověřovacímu readeru. Událost je vyvolána pro všechny detekované chyby při čtení. Interní mechanismy readeru, odpovědné za kontrolu správně strukturovaného dokumentu a kontrola se schématem jsou dvě odlišné věci. Jestliže ověřovací reader kontroluje špatně strukturovaný XML dokument, není vyvolána přiřazená událost, ale jen vyjímka typu *XmlException*. Jednotlivé chyby je možné poté zpracovat samostně. Jak bude dokument analyzován určuje vlastnost ValidationType, kterou je potřeba nastavit ještě před tím, než je dokument čten metodou *Read()*.

Typy ověřování:

- None
- Auto
- DTD
- Schema
- XDR

Při zvolení možnosti *None* se vytvoří reader, který neprovádí kontrolu a všechny chyby spojené s ověřováním jsou ignorovány. Jestliže je nastaven typ ověřování na *Auto* reader určí nejvhodnější typ ověřování. Pokusí se nejdříve vyhledat v dokumentu DTD (tato kontrola má vždy přednost před ostatnímy typy). V případě nenalezení je postupně hledáno XSD (odkaz nebo in-line schéma) a nakonec XDR. Pokud není nalezeno žádné schéma, je vytvořen reader bez kontroly.

5.3.3 Formátování dokumentu

Jednou z výhod jazyka Xml je dobrá čitelnost i pro člověka. Tato vlastnost ovšem platí pokud není Xml dokument příliš složitý a musí být správně naformátován(well-formed dokument). Správně zformátovaný dokument obsahuje na jednom řádku pouze jeden element, který je odsazen od kraje dle úrovně, na které se nachází.

```
<DvdList>
  <DVD ID="1" Category="Science Fiction">
    <Tittle>Matrix</Tittle>
    <Director>Larry Wachowski</Director>
    <Price>18.74</Price>
    <Starring>
      <Star>Keanu Reeves</Star>
      <Star>Laurence Fishburne</Star>
    </Starring>
  </DVD>

     ...

</DvdList>
```

Příklad 15: Zformátovaný dokument

Web Xml Editor dokáže aplikovat tyto pravidla na Xml dokument a zformátovat ho. Toto formátování probíhá na základě získání obsahu dokumentu v podobě textového řetězce, vytvořením Xml dokumentu v paměti z tohoto řetězce a následně opětovným vkládáním vytvořených bloků do komponenty RichTextBox s ohledem na stanovená pravidla.

5.3.4 Odvození schématu XSD podle XML dokumentu

Vytvoření schématu Xml je možné provézt několika způsoby. Visual Studio obsahuje možnost klasického ručního návrh nebo je možné využít vizuální editor *XML Editor for XSD files*. Tvorba XSD schématu a jeho jednotlivých komponent probíhá za pomoci myši a příkazů místních nabídek. Další příjemnou vlastností Visual Studia je schopnost dynamicky odvodit schéma ze souboru Xml. Tuto úlohu provádí xsd.exe, který se spouští pomocí integrovaného příkazového řádku (*Command Prompt*). Tento příkaz zprostředkovává například ty důležité převody:

- **XDR → XSD** – generuje XML schéma ze schématu XDR, který je starší.
    - příklad: xsd mojeSchema.xdr
- **XML → XSD** – z existujícího Xml dokumentu vygeneruje XSD schema.
    - xsd XmlDocument.xml
- **XSD → CLR třída** – vytvoří třídu ze schématu, ve zvoleném programovacím jazyce (C#, VB.NET, JScript, Visual J#).
    - xsd /c /language:CS xsdSchema.xsd

Webový Xml Editor poskytuje podobnou možnost odvození XSD schématu dle Xml dokumentu. Vygenerování Xml schématu probíhá na straně serveru za pomocí technologie WCF. Aplikace odešle Xml dokument a jako výsledek obdrží schema ve formátu XSD. Pro porovnání ukázka xml dokumentu, ručně vytvořeného schématu, za pomoci nástroje xsd.exe a pomocí Web Xml Editoru.

```xml
<?xml version="1.0" encoding="utf-8"?>
<DvdList>
  <DVD ID="1" Category="Science Fiction">
    <Tittle>Matrix</Tittle>
    <Director>Larry Wachowski</Director>
    <Price>18.74</Price>
    <Starring>
      <Star>Keanu Reeves</Star>
      <Star>Laurence Fishburne</Star>
    </Starring>
  </DVD>
...
</DvdList>
```

Příklad 16: Xml dokument pro odvození XSD schemma

```xml
<?xml version="1.0" encoding="utf-16"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="DvdList">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="DVD">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Tittle" type="xs:string" />
              <xs:element name="Director" type="xs:string" />
              <xs:element name="Price" type="xs:decimal" />
              <xs:element name="Starring">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element maxOccurs="unbounded" name="Star" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="ID" type="xs:unsignedByte" use="required" />
            <xs:attribute name="Category" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Příklad 17: Odvozené XSD schema

5.3.5 Syntax highlighting

Zvýrazňován syntaxe Xml dokumentu je jednou z důležitých částí Xml Editoru, která pomáha snadné orientaci v dokumentu. Pro editaci a zvýraznění textu je nutné v aplikaci použít takovou komponentu, která dovoluje nastavit různé styly pro textový obsah. Komponenta, jakou je například klasický TextBox, může nastavit styl textu pouze pro celý obsah, nikoli pro jeho části. Proto je nutné najít jiné řešení v podobě vytvoření vlastní komponenty, která by toto umoňovala, nebo využít Silverlight komponentu, dostupnou od verze 4, Rich Text Box. (poznámka: komponenta RichTextBox existuje již od verze Silverlight 4 beta, ale pod názvem RichTextArea. Novějšího RichTextBox navíc obsahuje některé další užitečné vlastnosti, které ve verzi beta nebyli možné. Jedná se například o zjištění polohy textového kurzoru při označování textu)

Řešení, v podobě vytvoření vlastní komponenty, je možné realizovat mnoha způsoby. Jedním z ních by mohlo být spojení komponent TextBox a TextBlock. Komponenta textbox slouží pro editaci a TextBlock pro zobrazování textu. Překrytím objektu TextBlock komponentou TextBox, přičemž průhlednost TextBoxu by byla nastenvena na maximum, dosteneme pár, který spolu spolupracuje. Uživatel by psal do TextBoxu, který je pro něho neviditelný, a na pozadí se po zpracování zobrazí zvýraznený text. Pro každou zvýrazněnou část by existoval prvek TextBlock.

S vydáním Silverlight 4 RC, ovšem vytváření vlastní komponenty není nutné. Rich Text Box pracuje na principu jako již existující komponenta technologie WPF. Obsah je tvořen kolekcí bloků, určitých typů. Každý z těchto bloků může mít nastaveny jiná formátovací pravidla, což umožňuje zobrazovat text s různým fotmátováním.

Seznam bloků:

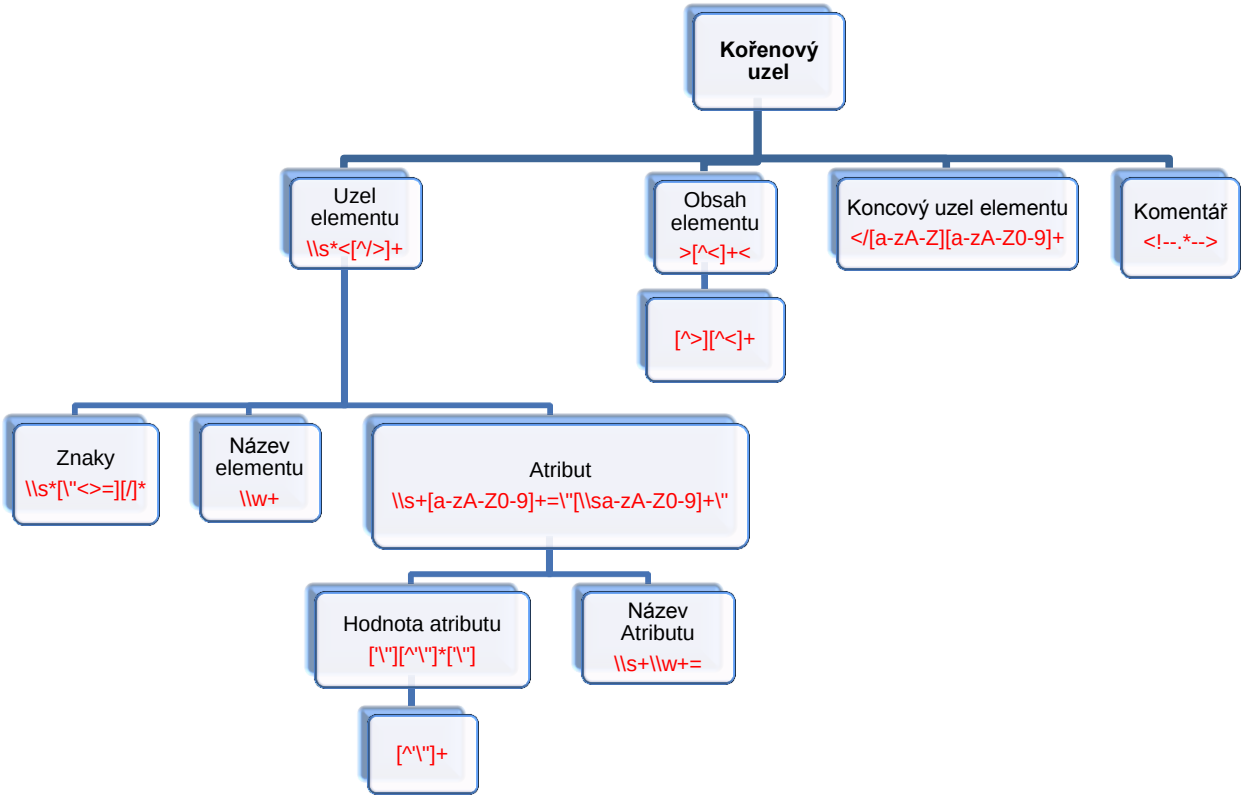
- *Paragraph* – Nejdůležitější objekt komponenty RichTextBox. Slouží jako kořenový element pro ostatní bloky. Text zapsaný v tomto bloku je zobrazen jako odstavec. To znamená na začátku a na konci je zalomení řádku
- *LineBreak* – Odřádkování uvnitř odstavce.
- *Run* – Blok run může být pouze uvnitř objektu paragraph a je využit pro formátování uvnitř odstavce.

- *InlineUIContainer* – Pomocí tohoto bloku je možné přidávat do komponenty RichTextBox jiné než textové informace. Unitř tohoto bloku může být vložen například obrázek, kalendář, tlačítko a další.

První způsob, kterým je možné realizovat highlighting, je založen na konečném automatu. Tento konečný automat čte postupně znaky v dokumentu a podle stavu, ve kterém se nachází, a přechodové tabulky se posune do dalšího stavu. Při kažném čtení automatu je zaznamenána pozice v textu a druh značky, která je čtena. Výstupem tohoto algoritmu je seznam pozic začátků značek, jejich délky a jim odpovídají druh Xml dokumentu. Vznikne tím seznam objektů pro popis značek. Procházením seznamu se programově označí text v komponentě RichTextBox a je zavolána metoda pro změnu barvy textu. Barvu určuje druh značky. Velká nevýhoda využití konečného automatu je v neustálém označování a obarvování textu. To průběh zpracování zpomaluje.

Další způsob analýzy Xml dokumentu je pomocí regulárních výrazů, které mají stejnou vyjadřovací schopnost jako konečné automaty. Jejich prostřednictvím je možné vytahovat z textových dat potřebné údaje a přetvářet je do jiné podoby.

V aplikaci Web Xml Editor je použit druhý způsob zvýrazňování syntaxe pomocí regulárních výrazů. Další užitečnou vlastností editoru je zformátování Xml dokumentu do well-formed podoby, aby byl dokument lépe čitelný. Toho je využito i při zvýrazňování syntaxe. Dokument je nejprve parsován a zformátován do určité podoby a poté jsou na jednotlivé řádky aplikovány připravené regulární výrazy, kterou jsou aplikovány hierarchicky.



5.3 Hierarchie aplikování regulárních výrazů

Při parsování regulárnímy výrazy jsou nalezené části zpracovány a uloženy do kolekce, která uchovává navíc informaci o pozici, kde byl výraz nalezen. Tato pozice později slouží pro opětovné sestavení dokumentu.

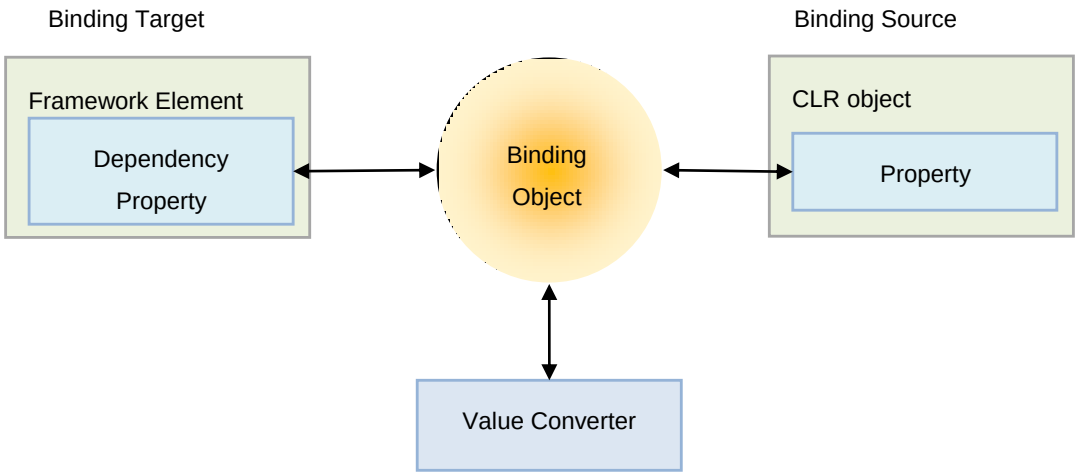
```
<DvdList>
  <DVD ID="1" Category="Science Fiction">
    <Tittle>Matrix</Tittle>
    <Director>Larry Wachowski</Director>
    <Price>18.74</Price>
    <Starring>
      <Star>Keanu Reeves</Star>
      <Star>Laurence Fishburne</Star>
    </Starring>
  </DVD>
  <DVD ID="2" Category="Drama">
    <Tittle>Forrest Gump</Tittle>
    <Director>Robert Zemeckis</Director>
    <Price>23.99</Price>
    <Starring>
      <Star>Tom Hanks</Star>
      <Star>Robin Wright</Star>
    </Starring>
  </DVD>
  <DVD ID="3" Category="Horror">
    <Tittle>The Others</Tittle>
```

5.4 Zvýraznění syntaxe

5.3.6 Databinding

Databinding stanovuje způsob, kterým Silverlight aplikace může zobrazovat a provádět interakci s daty. Jde o sytém, jakým se zobrazení dat oddělí ze správy dat. Připojení nebo provázání mezi UI a datovými objekty dovoluje datový tok mezi těmito vrstavami. Když jsou data provázány, tak veškeré změny zobrazených dat v UI elementu mohou reflektovat změny do datového objektu automaticky. Například pokud uživatel změní hodnotu v textovém poli, které je provázáno s nějakým datovým objektem.

Při každém bindingu je nutné specifikovat zdroj a cíl. Následující obrázek zobrazuje základní koncept bindingu.



Infrastruktura pro bindování potřebuje znát následující informace:

- Cílovou vlastnost UI objektu, která zobrazuje data. Cílem může být jakákoliv DependencyProperty elementu.
- Zdroj obsahující data, která proudí mezi zdrojem a cílem. Zdrojem může být kterýkoliv CLR objekt.
- Směr, kterým data proudí. Ten je určen nastavením módu objektu na který je prováděn binding.

Pro vázání dat na cílové UI objekty musí být zdroj datový objekt typu kolekce, přičemž každý prvek v kolekci reprezentuje jeden záznam. Kolekce musí mít určena i svůj datový typ. Jestliže je obsah Xml dokumetnu známý, stačí vytvořit třídu, která bude reprezentovat jednotlivé záznamy Xml dokumentu. Například pro Xml dokument obsahující záznamy o zaměstnancích (jméno, věk, plat) by třída měla stejné vlastnosti jméno, věk, plat. Tato třída je použita jako typ kolekce. Problém může nastat, jestliže není známa struktura Xml dokumentu. Pro tento případ je vytvořena třída XmlElement, která je použitelná pro jakýkoliv obecný Xml dokument. Díky této třídě je možné vytvořit kolekci a provézt databinding.

```
namespace XmlEditor
{

    public class XmlAttribute
    {
        public int Id {get; set;}
        public string Name { get; set; }
        public string Value { get; set; }
        public int ElementId { get; set; }

        public XmlAttribute()
        {

        }
    }

    public class XmlElement
    {
        public int Id {get; set;}
        public int ParentId { get; set; } // If ParentId = 0 -> root element
        public String Name {get; set;}
        public int[] AttributesId {get; set;} // null -> has no attributes
        public List<XmlElement> Elements { get; set; }
        public string Value { get; set; }
        public Boolean HasElements { get; set; } // if HasElements is null -> element have only value

        public XmlElement()
        {

        }
    }
}
```

Příklad 18: Třída reprezentují Xml dokument

5.4 Návrh WCF služby

Aplikace pro některé funkce editoru, jako je například validace dokumentu, potřebuje využít některých tříd, které jsou dostupné pouze v kompletním .NET Framework. Proto je navržena WCF služba, která po zavolání vykoná potřebné operace a vrátí výsledek aplikaci.

5.4.1 Cross domain policy

Aby mohla Silverlight aplikace konzumovat data z WCF služby v jiné doméně, musí to služba povolit pomocí cross domain policy specifikace, což je soubor zásad, který umožňuje tento přístup. Toto zabezpečení je z důvodů cross-site scripting útoku. Z pravidla stačí do kořenového adresáře služby umístit cross domain policy soubor, který je ve formátu xml. Jestliže je služba hostována pro Silverlight aplikaci je potřeba zajistit, aby byl tento soubor nalezen. To bude provedeno změnou výchozího nastavení služby a ručního obsloužení požadavku na cross domain policy soubor. Obsluha bude provedena pomocí *REST* služby

(*Representational State Transfer*) použitím WCF služby, která bude poskytovat *ClientCccessPolicy.xml*. Tímto postupem bude dosaženo, aby byl soubor dostupný z kořenového adresáře, i když je umístěn do adresáře Bin.

```xml
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-request-headers="*">
        <domain uri="*"/>
      </allow-from>
      <grant-to>
        <resource path="/" include-subpaths="true"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

Příklad 19: Obsah souboru *ClientAccessPolicy.xml*

```
using System.ServiceModel;
using System.ServiceModel.Web;
using System.ServiceModel.Channels;

namespace XMLServiceLibrary
{
    [ServiceContract]
    public interface ICrossDomainService
    {
        [OperationContract]
        [WebGet(UriTemplate = "ClientAccessPolicy.xml")]
        Message ProvidePolicyFile();


    }
}
```

Příklad 20: Definice Contractu pro obsluhu požadavku na ClientAccessPolicy.xml

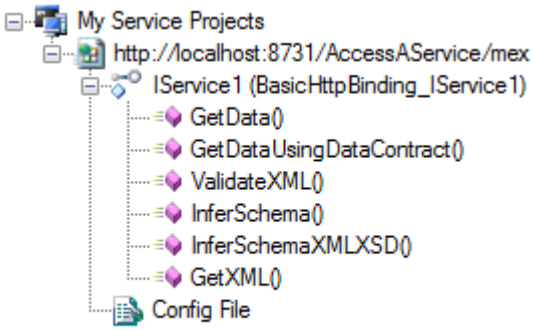
```
using System.IO;
using System.Xml;
using System.ServiceModel.Channels;

namespace XMLServiceLibrary
{
    public class CrossDomainService : ICrossDomainService
    {


        #region ICrossDomainService Members

        public System.ServiceModel.Channels.Message ProvidePolicyFile()
        {
            FileStream filestream = File.Open(@"ClientAcessPolicy.xml", FileMode.Open);
            XmlReader reader = XmlReader.Create(filestream);
            System.ServiceModel.Channels.Message result =
                Message.CreateMessage(MessageVersion.None, "", reader);

            return result;
        }

        #endregion
    }
}
```

Příklad 21: Obsluha požadavku pro ClientAccessPolicy.xml

5.4.2 Definice kontraktu a metod

Jednotlivé metody, které WCF služba poskytuje okolí, jsou definovány v kontraktu *IService.cs*. Metody slouží pro

- validaci Xml dokumentu (ValidateXML)
- odvození XSD schema dle Xml dokumentu (InferSchema)
- odvození XSD schema dle Xml dokumentu a původního XSD schématu(InferSchemaXMLXSD)
- získání Xml dokumentu vygenerovaného na základě XSD schema (GetXML)



Data, která jsou posílány prostřednictvím *SOAP* protokolu, jsou typu *String*, který je snadno serializovatelný.

5.4.3 Vytvoření reference na WCF službu

Pro komunikaci Silverlight aplikace s WCF službou je nutné v projektu vytvořit referenci. Tato reference nám v *Namespace* aplikace vytvoří třídu. Po vytvoření instance této třídy proxy je možné volat WCF službu. Silverligt dovoluje použít pro komunikaci pouze asynchronní volání metod, proto je nutné zpracovat výsledek až po úplném doručení.

```
private void ValidateXML_Click(object sender, RoutedEventArgs e)
{

    Popisek.Content = "Status";
    try
    {
        WcfServiceReference.Service1Client proxy = new WcfServiceReference.Service1Client();
        proxy.ValidateXMLCompleted +=
            new EventHandler<WcfServiceReference.ValidateXMLCompletedEventArgs>(proxy_ValidateXMLCompleted);
        proxy.ValidateXMLAsync(NactiText(rta), NactiText(xsdRTA));
    }
    catch (Exception ex)
    {
        Popisek.Content = ex.Message.ToString();
    }
}

void proxy_ValidateXMLCompleted(object sender, WcfServiceReference.ValidateXMLCompletedEventArgs e)
{
    if (e.Error == null)
        Popisek.Content = e.Result;
    else
        HtmlPage.Window.Alert(e.Error.Message);
}
```

Příklad 22: Vytvoření proxy třídy pro validaci

6 Uživatelská dokumentace

6.1 Potřebná nastavení

Hardwarové požadavky:

Jediným hardwarovým požadavkem je počítač s dostatečně rychlím připojením k síti internet (doporučení minimálně 256 kb/s). Celkové požadavky na výkonost počítače jsou závislé na nainstalovaném operačním systému.

Softwarové požadavky:

Webová aplikace ke svému běhu potřebuje funkční a správně nastavený internetový prohlížeč, který navíc obsahuje doplněk Silverlight 4.0. Tento plug-in podporuje následující operační systémy.

Microsoft Windows:

- Microsoft Windows 7
- Microsoft Windows XP SP2
- Microsoft Windows 2000
- Microsoft Windows 2003
- Microsoft Windows 2008

Macintosh

- Mac OS 10.4.8 + (PowerPC)
- Mac OS 10.4.8+ (Intel based)

Linux

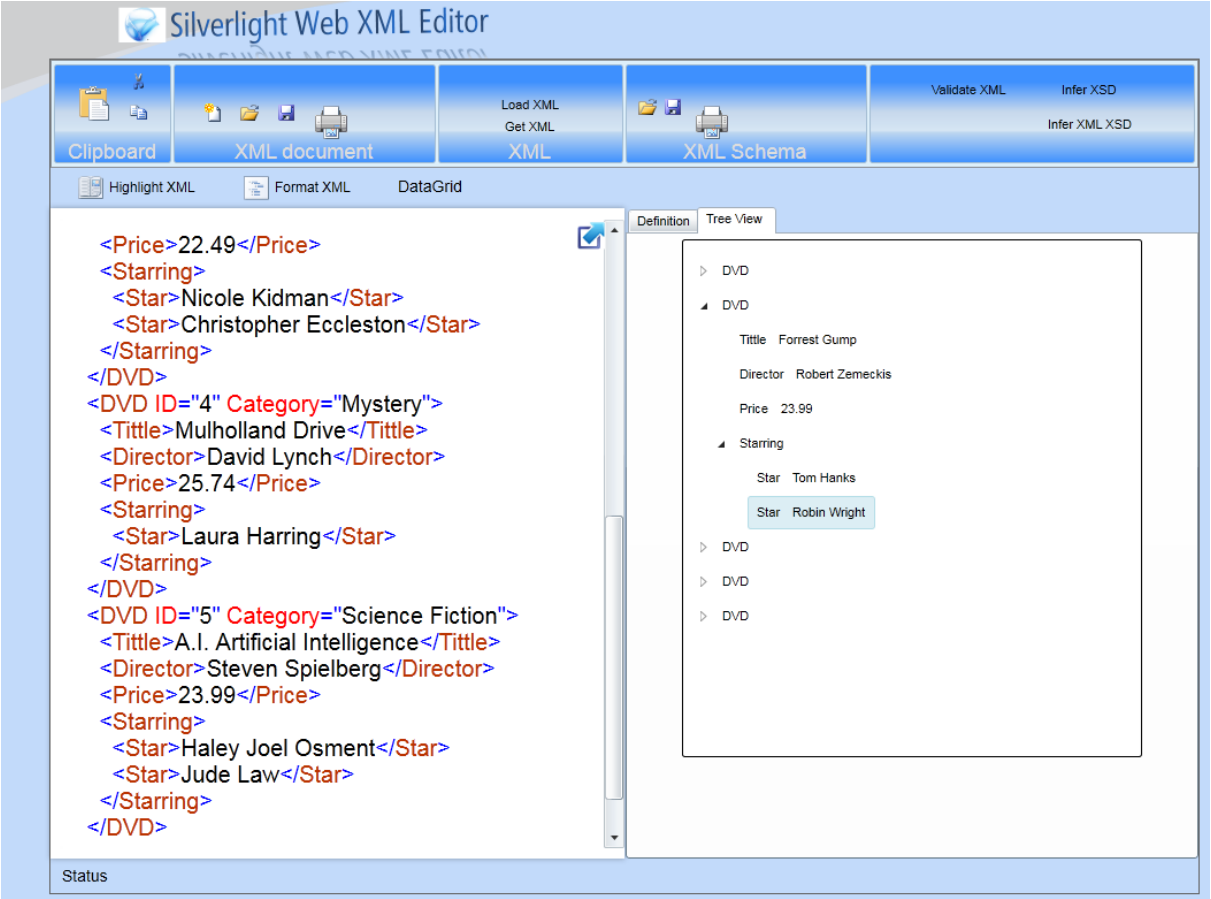
Implementace technologie Silverlight do operačního systému je známa pod názvem Moonlight. V současné době je podporována pouze Silverlight verze 2.0. Finální verze Moonlight 4, která by podporovala i Silverlight 4 je očekávána ke konci roku 2010.

Pokud není doplněk nainstalován aplikace zobrazí okno s možností instalace.

6.1 Instalace doplňku Microsoft Silverlight

6.2 Uživatelské prostředí

Pracovní plocha je rozdělena do několika částí. Horní panel obsahuje základní ovládací prvky sloužící pro ovládání editoru. Prvky jsou rozděleny do kategorií podle funkce, kterou plní. Podrobnější popis tlačítek je zajištěn pomocí malé nápovědy v podobě tooltipu. Hlavní část okna obsahuje textový editor pro práci s Xml dokumenty v levé častí a pravá část obsahuje více oken rozdělených do záložek. Jednotlivé záložky slouží pro vyhledávání, zobrazení stromu Xml dokumentu a práci s Xml schématem. Dolní část je určena pro výpis možných chyb nebo zpráv. Velikost okna aplikace je stanovena pevně i s ohledem na menší širokoúhlý displej notebooků. Při použití prohlížeče Internet Explorer je dobré maximalizovat zobrazení stránky na maximum (klávesa F11).



6.2 Uživatelské prostředí

6.3 Ovládání

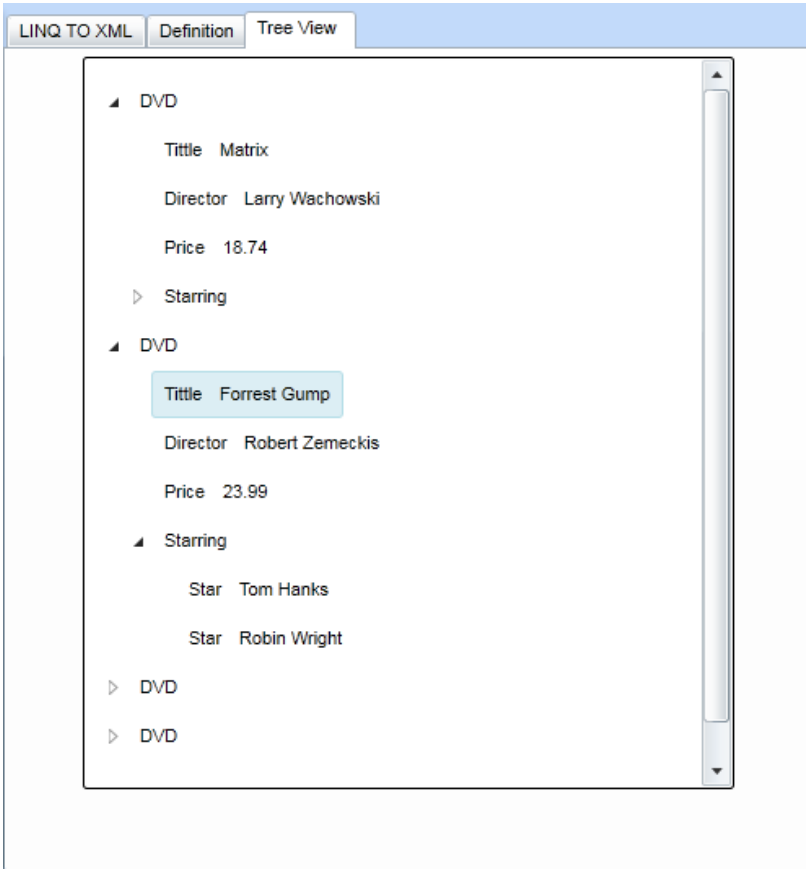
Hlavní menu obsahuje tyto položky:

- Clipboard
  - *Copy* – zkopíruje označený text do schránky
  - *Cut* – vyjme označený text do schránky
  - *Paste* – vloží text ze schránky na pozici kurzoru
- Xml File
  - *New* – vytvoří nový Xml dokument
  - *Open* – otevře dialogové okno pro výběr Xml souboru
  - *Save* – otevře dialogové okno pro uložení Xml souboru
  - *Print* – zobrazí okno s náhledem na tisk a umožní tisk Xml dokumentu

- XML
    - *Load XML* – načte Xml dokument do paměti, zkontroluje jeho správnost a zobrazí strom v záložce TreeView
- XML Schema
    - *Open* – otevře dialogové okno pro výběr Xml schématu
    - *Save* – otevře dialogové okno pro uložení Xml schématu
    - *Print* – zobrazí okno s náhledem na tisk a umožní tisk schématu dokumentu
    - *Validate XML* – zvaliduje Xml dokument dle schematu
    - *Infer XSD – vytvoří XSD* schéma dle otevřeného Xml dokumentu
    - *Infer XML XSD* – vytvoří nové schéma na základě původního a podobného Xml dokumentu
-

6.3.1 Zobrazení stromu Xml dokumentu

Web Xml Editor obsahuje funkci pro zobrazení Xml dokumentu pomocí hierarchického stromu. Pro načtení dokumentu slouží tlačítko Load Xml. Aby bylo zobrazení dostupné, Xml dokument musí být správně strukturován. V opačném případě je zobrazena podrobná zpráva o chybě. Strom je zobrazen v záložce Tree View.



6.3.2 Validace Xml dokumentu

Pro validaci dokumentu je nutné mít načtený Xml dokument a příslušné schéma. Tlačítkem Validate Xml je dokument zvalidován a na spodní liště je zobrazena zpráva o výsledku.

6.3.3 Odvození XSD schématu

Odvození XSD schématu může být dvojího druhu. První možností je odvodit schéma dle otevřeného Xml dokumentu. Pro tuto volbu slouží tlačítko Infer XSD, které po vytvoření zobrazí schéma v záložce Definition, do které je uživatel automaticky přepnut. Toho je možné využít i při tvoření schémat. Je daleko jednodušší vytvořit Xml dokument a schema odvodit, než ho psát ručně. Vytvořené schéma je samozdřejmě možné podle představ i ručně upravit.

Druhou volbou je rozšíření stávající schématu o další prvky na základě Xml dokumentu. Aby bylo možné využít této schopnosti Web Xml Editoru, je zapotřebí mít otevřeno původní XSD schema a Xml dokument, dle kterého bude schema doplněno.

Příkladem použití může být knihovna, která má knihy uloženy v rozdílných Xml formátech, které se nepatrně liší (knihy1.xml, knihy2.xml). Postup pro získání schématu, které by bylo validní pro všechny dokumenty je následující:

1. Otevřít knihy1.xml
2. Pomocí Infer XSD odvodit první schéma, které odpovídá knihy1.xml
3. Otevřít knihy2.xml, přičemž v záložce Definitions je stále otevřeno první schema
4. Použít tlačítko Infer XML XSD
5. Záložka Definitions nyní obsahuje nové schéma, které odpovídá oběma dokumentům
6. Tento postup lze opakovat tolikrát, kolik je dokumentů

6.3.4 Generování Xml

Jednou z užitečných vlastností Web Xml Editoru je automatické generování Xml dokumentu dle jeho schema. Pro tuto volbu je nutné mít otevřené schema a pro generování slouží tlačítko GetXml.

6.3.5 Tisk

Aplikace obsahuje podporu pro tisk. Za tímto účelem jsou v menu dvě ikonky, které jsou určeny pro tisk Xml dokumentu nebo schématu. Při zvolení tisku je zobrazen náhled na tištěný dokument a po potvrzení (tlačítko OK) je zobrazeno okno pro výběr tiskárny a nastavení tisku.

Závěr

Webové rozhranní v některých směrech znamená omezení funkčnosti a možností klienta a metody známé z desktopových aplikací nejsou standartními technologiemi prohlížečů podporovány. Cílem diplomové práce bylo vytvoření funkční webové aplikace pro editaci a vytváření Xml dokumentů ve vhodně zvolené technologii, která by uživateli nabídla podobný komfort při práci jako aplikace určená pro desktop. Využití najde převážně díky dostupnosti a všudypřítomnosti webového prohlížeče jako klienta.

Vývoj probíhal na předem stanovených požadavcích pro aplikaci, které zahrnují základní vlastnosti každého Xml editoru. Dle těchto požadavků byla vybrána nejvhodnější webová technologie pro zhotovení Web Xml Editoru. Implementace webové aplikace byla provedena pomocí technologie Silverlight 4, která uživateli nabízí interaktivní a působivé prostředí pro práci. Zvolená technologie ovšem obsahuje pouze redukovanou sadu .Net tříd umožňující práci s Xml dokumenty, a proto je využito i technologie WCF.

Pro zhotovení aplikace bylo použito vývojářského nástroje Visual Studio 2008, později 2010 s využitím programovacího jazyka C# 3.0. Uživatelské rozhraní bylo vytvořemo pomocí jazyka XAML, který slouží pro vytváření UI elementů Silverlight aplikace. V počátcích vývoje bylo nutné použít Silverlight verze 3, avšak po vydání nejnovější verze 4, byla aplikace upravena a umožňovala více funkcí, jako například tisk dokumentu. Aplikace umožňuje uživateli funkce pro prohlížení Xml dokumnetu jako je formátování a zvýrazňování syntaxe nebo zobrazení dokumentu v podobě stromu elementů. Může posloužit jako užitečný nástroj pro validaci dokumentu, odvození XSD schématu nebo rozšíření stávajícího dle Xml dokumentu.

Aplikace byla vytvořena v souladu s moderními trendy programování a využívá dostupné technologie, proto je dále rozšiřitelná a její správa udržovatelná. Je zde další prostor pro další rozšíření možností webové aplikace. Další vývoj by se mohl ubírat směrem k usnadnění návrhu Xml schémat pomocí grafických ovládacích prvků, které jsou známy z vývoje na některých z komerčních Xml editorů. Příjemným doplněm pro uživatele by byla schopnost personalizace uživatelského prostředí dle svých představ. Jelikož se jedná o webovou aplikaci, zajímavým rozšířením by byly uživatelské účty s možností uložit Xml dokument na server, aby byl snadněji dostupný i z jiných počítačů.

Web Xml editor vznikl z důvodů absence podobného xml editoru zhotoveného jako webová aplikace. Tímto se stává jediným nejrychlejším a dostupným řešením za účelem potřeby editovat Xml data bez nutnosti instalace desktopového programu.

Seznam použité literatury

[1] M. MacDonald, M. Szpuszta: ASP.NET 3.5 a C# 2008 tvorba dynamických stránek profesionálně. Zoner press, Brno 2008

[2] D. Esposito: XML – efektivní programování pro .NET. Grada, 2004

[3] Justin Smith: INSIDE WINDOWS COMMUNICATION FOUNDATION. Microsoft Press, 2007

[4] John Sharp: Microsoft Visual C# 2008 Step by Step. Microsoft Press, 2008

[5] Shannon Horn: Microsoft Silverlight 3: A Beginner´s Guide. The McGraw-Hill Companies, 2010

[6] Microsoft ASP.NET, URL: <http://asp.net/>[online][10. 5. 2010]

[7] MSDN home – Microsoft.com, URL:<http://msdn.microsoft.com>

[online][10. 5. 2010]

[8] Microsoft Silverlight, URL: <http://www.silverlight.net/>[online][10. 5. 2010]

[9] XML pro každého. URL:< http://kosek.cz/xml/> [online][10. 4. 2010]

[10] Server .NET technologií – dotnetcurry.com: Použití Silverlight pro přístup k WCF službě. URL:< http://www.dotnetcurry.com/ShowArticle.aspx?ID=208>

[online][10. 4. 2010]

Příloha A –Odvození XSD schema

Předloha - Xml dokument:

```
<?xml version="1.0" encoding="utf-8"?>
<DvdList>
  <DVD ID="1" Category="Science Fiction">
    <Tittle>Matrix</Tittle>
    <Director>Larry Wachowski</Director>
    <Price>18.74</Price>
    <Starring>
      <Star>Keanu Reeves</Star>
      <Star>Laurence Fishburne</Star>
    </Starring>
  </DVD>
...
</DvdList>
```

Ručně napsané XSD schéma:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="Dvdlist" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="DvdList">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="DVD" type="DVDType" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="DVDType">
    <xs:sequence>
      <xs:element name="Tittle" type="xs:string" />
      <xs:element name="Director" type="xs:string" />
      <xs:element name="Price" type="xs:decimal" />
      <xs:element name="Starring" type="StarringType" />
    </xs:sequence>
    <xs:attribute name="ID" type="xs:integer" />
    <xs:attribute name="Category" type="xs:string" />
  </xs:complexType>
  <xs:complexType name="StarringType">
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="Star" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Odvozené schéma pomocí Web Xml Editoru:

```
<?xml version="1.0" encoding="utf-16"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="DvdList">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="DVD">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Tittle" type="xs:string" />
              <xs:element name="Director" type="xs:string" />
              <xs:element name="Price" type="xs:decimal" />
              <xs:element name="Starring">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element maxOccurs="unbounded" name="Star" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="ID" type="xs:unsignedByte" use="required" />
            <xs:attribute name="Category" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Odvozené XSD schéma pomocí nástroje Visual Studia – xsd.exe:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="DvdList" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="DvdList" msdata:IsDataSet="true" msdata:Locale="en-US">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="DVD">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Tittle" type="xs:string" minOccurs="0" msdata:Ordinal="0" />
              <xs:element name="Director" type="xs:string" minOccurs="0" msdata:Ordinal="1" />
              <xs:element name="Price" type="xs:string" minOccurs="0" msdata:Ordinal="2" />
              <xs:element name="Starring" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Star" nillable="true" minOccurs="0"
maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:simpleContent msdata:ColumnName="Star_Text" msdata:Ordinal="0">
                          <xs:extension base="xs:string">
                          </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="ID" type="xs:string" />
            <xs:attribute name="Category" type="xs:string" />
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Příloha B – Konfigurační soubor WCF služby

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <!-- When deploying the service library project, the content of the config file must be
added to the host's
  app.config file. System.Configuration does not support config files for libraries. -->
  <system.serviceModel>
    <services>
      <service behaviorConfiguration="XMLServiceLibrary.Service1Behavior"
        name="XMLServiceLibrary.Service1">
        <endpoint address="" binding="basicHttpBinding"
contract="XMLServiceLibrary.IService1">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8731/AccessAService" />
          </baseAddresses>
        </host>
      </service>
      <service name="XMLServiceLibrary.CrossDomainService">

        <endpoint address="" binding="webHttpBinding"
contract="XMLServiceLibrary.ICrossDomainService"
                  behaviorConfiguration="XMLServiceLibrary.CrossDomainServiceBehavior"/>
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8731/" />
          </baseAddresses>
        </host>
      </service>
    </services>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />
    <behaviors>
      <serviceBehaviors>
        <behavior name="XMLServiceLibrary.Service1Behavior">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
      </serviceBehaviors>
      <endpointBehaviors>
        <behavior name="XMLServiceLibrary.CrossDomainServiceBehavior">
          <webHttp/>
        </behavior>
      </endpointBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```