UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY
A INFORMATIKY

DIPLOMOVÁ PRÁCE

2010 Bc. Jan Kašpar

Univerzita Pardubice

Fakulta Elektrotechniky a Informatiky

Tvorba a zpracování souborů XML v Javě

Bc. Jan Kašpar

Diplomová práce

2010

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jan KAŠPAR**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Tvorba a zpracování souborů XML v Javě**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

- V úvodu práce je nutné zpřehlednit zadanou problematiku a provést její kategorizaci. Přehledně popsat používané technologie a jazyky pro tvorbu XML souborů. Popsat metody pro transformaci, kompresi a modelování XML dat při jejich přenosu a ukládání např. do objektově - relačních databází. - Hlavním cílem diplomové práce je vytvoření vzorových implementací výše uvedených metod v jazyce Java s použitím moderních nástrojů (XML Schema, SAX, JDOM atd.).

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. McLaughlin B.: Java and XML, Second Edition. O'Reilly Media, 2001. 2. Mlýnková I., a kol.: XML technologie - Principy a aplikace v praxi. Grada Publishing a.s., 2008. 3. Herout P.: Java a XML. Koop, 2007.

Vedoucí diplomové práce: **Ing. Zdeněk Šilar**
Katedra informačních technologií

Datum zadání diplomové práce: **30. října 2009**

Termín odevzdání diplomové práce: **21. května 2010**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

doc. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 10. listopadu 2009

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 20. 8. 2010

Bc. Jan Kašpar

ABSTRAKT

Hlavním tématem diplomové práce je seznámení s technologiemi pro zpracování XML souborů v jazyce Java. Dále pak seznámení s možnostmi transformace a komprese XML dat při ukládání do objektově-relačních databází.

V praktické části práce jsou použity popsané technologie. Tyto technologie jsou implementovány v systému z reálného světa. Analýza, návrh a implementace systému pro evidenci nákladních vozů železničního dopravce proběhla podle standardizovaných metodik.

KLÍČOVÁ SLOVA

Java, XML, XML technologie, použití XML v praxi, XSLT, JDOM, JAXB, StAX, SAX, validace XML

TITLE

Creation and processing XML files in Java.

ABSTRACT

The main theme of this thesis is to introduce the technology for processing XML files in Java. Furthermore, acquaintance with the possibilities, and transform XML data compression when saving the object-relational databases.

Described technologies are used in the practical part of the work. These technologies are implemented in the system of the real world. Analysis, design and implementation of a system for registration of trucks of the rail carrier was conducted according to standardized methodologies.

KEYWORDS

Java, XML, XML technologies, XML use in practice, XSLT, JDOM, JAXB, StAX, SAX, XML validation

Poděkování

Na tomto místě bych rád poděkoval Ing. Zdeňku Šilarovi za odborné vedení diplomové práce a za všechny odborné rady, které mi byly poskytnuty během zpracovávání této práce.

Obsah

1. Úvod

XML je velmi oblíbený díky svým nesporným výhodám při jeho použití. Jednou z nejvýhodnějších vlastností XML je platformní nezávislost. Této vlastnosti se využívá při komunikaci mezi systémy. Příkladem použití jsou servisně orientované systémy, které jsou na XML založeny. Nejčastěji se však s XML můžeme setkat při migraci dat. Díky flexibilitě popisu obsahu XML se používá také pro Export a Import dat. Tento trend je nejvíce patrný v současných databázových systémech.

Mezi dalšími možnostmi použití jsou elektronické publikace. V současné době jsou například technické dokumentce vytvářeny v různých formátech a to z důvodů rozdílných potřeb uživatelů. Uživatel může mít nejrůznější podmínky pro zobrazení a proto se s výhodou používá díky možnosti konverze do různých formátů.

Cílem této práce je čtenáře seznámit s problematikou zpracování XML souborů v Javě. Vzhledem k velké oblibě XML existuje velké množství technologií, které s tímto formátem pracují. Tyto technologie se liší přístupem ke XML nebo oblastí použití. Proto je v této práci souhrn nejpoužívanějších technologií a jejich popis. Práce se dále zabývá možnostmi uložení XML do objektově-relačních databázových systémů.

V praktické části jsou ukázky, jak se dané technologie pro zpracování XML používají. Technologie pro práci s XML jsou aplikovány do navrhnutého vzorového systému, který slouží k evidenci nákladních vozů železničního dopravce. Největší důraz bude kladen na části systému, kde budou použity technologie pro práci s XML.

2. XML

Extensible Markup Language, ve zkratce XML, je, jak už název napovídá, značkovacím jazykem. XML je velmi rozšířeným způsobem, jak uchovávat data. Dále také slouží ke komunikaci v servisně orientovaných systémech a hojně se využívá v různých integračních platformách.

V XML si uživatel může sám nadefinovat sadu značek, které bude používat, což zajišťuje vysokou flexibilitu tohoto jazyka pro reprezentaci dat. S největší pravděpodobností právě tato vlastnost stojí za velkou oblibou tohoto formátu ukládání dat. A z toho důvodu, že si každý může libovolně nadefinovat sadu značek přesně pro účely své aplikce. XML má hned několik vlastností, které jsou jeho velkou výhodou.[7]

V zásadě ke XML můžeme přistupovat dvěma různými způsoby. Těmito způsoby jsou:

- dokumentově orientovaný přístup,
- datově orientovaný přístup.

2.1. Dokumentově orientovaný přístup

Jedná se o XML dokument vytvářený uživateli především ručně a má značně nepravidelnou strukturu. Ve většině případů záleží na pořadí elementů na stejné úrovni. Obsah elementů je velmi často smíšený. Dokumentově orientovaný přístup se vyznačuje tím, že velikost výsledného souboru není do takové míry ovlivněná znaky pro značky XML jazyka, jelikož značky zapouzdřují většinou velké bloky textu, a tak nejsou tolik datově náročné.

2.2. Datově orientovaný přístup

Jak již název napovídá, jedná se o přístup, při němž se do XML dokumentu ukládají data. Většinou automatické vytváření i zpracovávání. XML dokument obsahuje strukturovaná data a naopak neobsahuje elementy se smíšeným obsahem. Také se zde nevyskytují kometáře. Pořadí uzlů na stejné úrovni není významné.

Z výše zmíněného vyplývá, že takovýto XML dokument se hodí pro výměnu nebo uchování dat. Využívá se především v servisně orientovaných systémech. U tohoto přístupu je nevýhodou

velikost značek uvozujících data. Proto by se z hlediska datové náročnosti zdálo lepší využití klasických binárních souborů, které mají výrazně menší velikost než XML dokumenty. Avšak XML dokument má nespornou výhodu v poměrně snadné kontrole vstupních dat.[1, str. 148-149]

2.3. Vlastnosti XML

Standardizovaný formát pro výměnu dat

XML je platformě nezávislý, otevřený formát. Je jednoduše strojově zpracovatelný a i běžný uživatel z něho dokáže bez obtíží vyčíst potřebné informace.

Multijazyčnost

Již od začátku vývoje XML jeho návrháři počítali s širokou podporou jazyků a nejen anglického jazyka. Podporuje celou řadu kódování, které však musí být v daném XML dokumentu uvedeno (utf-8, windows-1250).

Velký obsah informací

Jelikož se v XML neřeší vzhled informací, tak je daleko vyšší míra užitné informace. Toho se dá výhodně využít například při vyhledávání.

Jednoduchá konvertovatelnost do různých formátů

XML dokument lze za pomocí XSL šablon konvertovat do různých formátů. Počínaje HTML přes RTF až tiskovým PDF konče. Tyto šablony tak umožňují konvertovat XML dokument podle firemních konvencí a požadavků.

Automatizovaná validace dokumentu

XML dokument lze na vstupu automaticky validovat. Tato validace se provadí proti schématům, která definují strukturu XML dokumentu a někdy i použité datové typy. To ovšem v závislosti na tom, jaký schématický jazyk je použit a o jaký druh XML dokumentu se jedná.

Možnost tvorby referencí

V XML dokument je možné vytvořit reference neboli odkazy. Tyto odkazy mohou být v rámci jednoho XML dokumentu, ale i v rámci více XML dokumentů. K tomuto adresování se využívají jazyky XPath a XLink.[7]

2.4. Pravidla pro tvorbu XML dokumentu

Pro tvorbu XML dokumentu existuje několik základních pravidel, podle kterých se vytváření XML dokumentu musí řídit, aby jeho syntaxe byla v pořádku. Tato pravidla jsou podobná pravidlům pro tvorbu XHTML. Výše zmíněná pravidla jsou následující:

- Elementy jsou párové,
- Elementy se nesmí křížit,
- Obsah XML dokumentu musí být uzavřený právě v jednom kořenovém elementu,
- Hodnoty atributů musí být v uvozovkách (čísla také),
- Jména elementů mohou obsahovat písmena, čísla, tečky, podtržítka a pomlčky (pokud se použijí akcentovaná písmena, tečky, podtržítka a pomlčky, nastává problém při programovém zpracování, jelikož použití různých kódování je vysoce pravděpodobné)
- Element nesmí začít tečkou, pomlčkou nebo podtržítkem,
- XML rozlišuje velká a malá písmena,
- Má stromovou strukturu (každý element má nejvýše jednoho předka)
- Speciální znaky z hlediska XML se nahrazují znakovými entitami,
- Komentáře se uvozují znaky <!-- a ukončují se --> .[13]

Znaková entita	Znak
&amp;	&
&lt;	<
&gt;	>
&quot;	“
&apos;	‘

Tabulka 1: Seznam znakových entit pro znaky, které nejdou standartně zapsat do XML [2 str. 30]

2.5. Schémové jazyky

Při validaci se ověřuje, zda daný XML dokument vyhovuje veškerým omezujícím podmínkám uvedeným ve schématu. Z toho vyplývá, že schéma obsahuje popis struktury XML dokumentů. Toho se hojně využívá na rozhraních systémů, jelikož v těchto místech nemůžeme zajistit správný formát XML dokumentů, které do systému vstupují z jiného systému či od uživatele. V rámci jednoho systému je pak validace nadbytečná. Předpokládáme totiž, že navržený systém vytváří validní XML dokumenty. Některé typy schémat podporují i definice datových typů a mají možnost omezení jejich hodnot.

2.5.1. Document Type Definition

Definice Typu Dokumentu, ve zkratce DTD, je schématickým jazykem pro popis XML dokumentu. Jedná se o starý, ale široce podporovaný způsob definice XML dokumentu. Ikdyž se v současné době přechází na XML Schema, je stále ještě DTD používaným způsobem definice. Především z hlediska historické kompatability, kdy se navazuje na dříve vytvořenou aplikaci, která používá DTD.

DTD má však mnoho nevýhod, díky nimž je vytlačován konkurentem XML Schema. Nejmarkantnějším je asi ta, že DTD není oproti jeho konkurenci v konvencích XML. DTD má svou vlastní syntaxi a ačkoliv je tato syntaxe značně jednoduchá, tak neumožňuje podobnou kontrolu správnosti, jako ji umožňuje syntaxe XML. Dále je DTD vytlačován také díky tomu, že DTD nemá možnost definovat datové typy a jejich omezení není vůbec vhodné pro použití při definici datově orientovaných XML dokumentů. Poslední zmiňovaná nevýhoda vychází z toho, že DTD je zde od samého počátku XML a neumí tak pracovat se jmennými prostory. Ty se využívají především v rozsáhlých XML dokumentech.[2 str. 51-69]

2.5.2. XML Schema

XML Schema obsahuje popis struktury XML dokumentu, typy jednotlivých elementů a možnosti jejich výskytu, z čehož vyplývá, že se jedná o jednoznačnou definici použitých datových typů a díky tomu je vyloučena rozdílná interpretace dat. XML Schema je nástupcem DTD schémat. Jedná se o soubory nejčastěji s příponou XSD.

Syntaxe XML Schema

Narozdíl od DTD je XML Schema psáno syntaxí jazyka XML. To má samozřejmě výhodu v tom, že není nutnost ovládat další syntaxi. Avšak XML Schema je od DTD značně složitější, jelikož postihuje širší spektrum možností pro definici XML typů. Na druhou stranu však tyto možnosti zajišťují přesnou definici datového typu a také rozmezí hodnot daného typu.

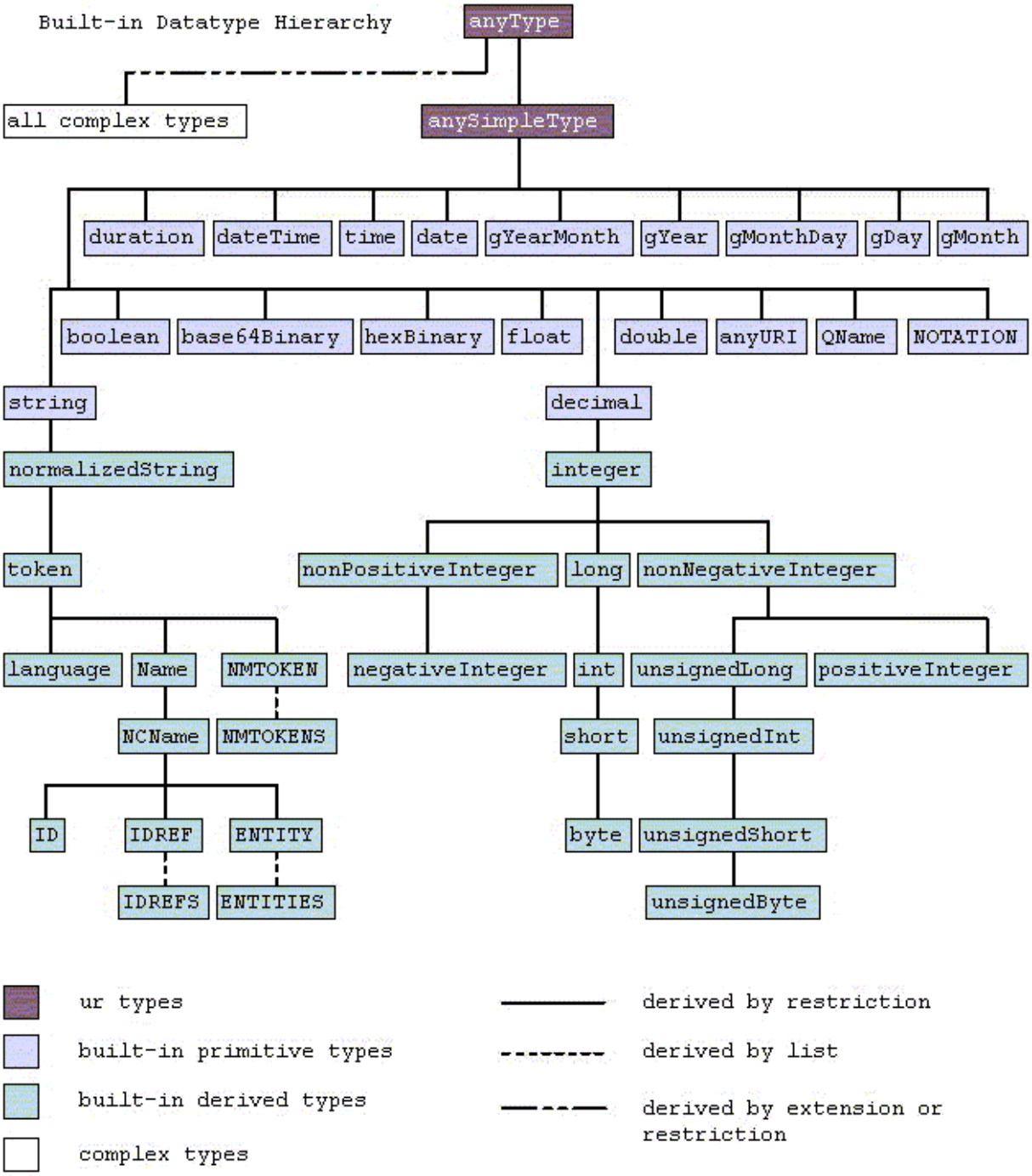
Podpora datových typu a jejich omezení

XML Schema je výhodné pro definování obsahu datově orientovaných XML dokumentů. Ale jeho poněkud vyšší náročnost na zápis však může vést k tomu, že i definice poněkud malého XML dokumetnu zabere v podání XML Schema více místa než XML dokument samotný.

XML Schema nabízí několik základních datových typů. Mezi ty nejzákladnější patří tyto následující:

- String,
- Boolean,
- Decimal,
- Float,
- Double,
- DateTime,
- Time,
- Date.

Pomocí těchto základních typů může vytvořit definice jednoduchých datových typů. Ovšem XML Schema má také možnost definice složitých datových typů. Ty se mohou skládat z mnoha elementů jednoduchého datového typu, ale i z dalších elementů složitého datového typů.



Obrázek 1: Schéma všech základních datových typů podporovaných v XML Schema [5]

Element jednoduchého datového typu se zapisuje pomocí tagu simpleType. Do tohoto uzlu lze ještě přidat několik omezení daného datového typu. Příkladem omezení může být povolený rozsah hodnot u číselného typu nebo délka u řetězcového typu. Přitom atributy omezení délky se dají ještě rozepsat na minimální a maximální délku. Pro omezení lze také využít regulární výrazy, omezení číselných typu shora i sdola.

Element složitého datového typu se zapisuje pomocí tagu complexType. Tento uzel může obsahovat několik elementů i atributů, přičemž element může obsahovat další complexType. Tento způsob se využívá především při definování datových struktur pro tvorbu instancí „javovských“ tříd pomocí technologie JAXB. Navíc uzel sequence uvozuje seznam elementů, které se mohou vyskytovat v dané struktuře jen v daném pořadí a počet výskytů lze ovlivnit pomocí atributu minOccurs a maxOccurs. Naproti tomu uzel choice pak uvozuje seznam elementů, které se na daném místě mohou vyskytnout. Vždy však pouze jeden z nich.

Pokud by na pořadí nezáleželo, byl by použit uzel all namísto sequence. Dále se může vyskytnout v XML dokumentu smíšený obsah a to především u dokumentově orientovaných XML dokumentů. Pokud se tedy u daného složitého typu očekává smíšený obsah, je nutné do uzlu complexType přidat atribut mixed a jeho hodnotu nastavit na true. To samotné by nestačilo, jelikož tento zápis by říkal, že text se může objevit kdekoliv v daném elementu. Proto se musí použít konstrukce choice, která bude definovat elementy, které se budou mísit s textem v daném elementu typu coplexType.

```
<Clanek>Odstavce typicky obsahují <pojem>smíšený obsah</pojem>. Text se může střídat s <odkaz
url="http://www.kosek.cz">odkazy</odkaz> a dalšími <pojem>elementy</pojem>.</Clanek>
<odstavec>Odstavec může obsahovat i jen text.</odstavec>
<odstavec><pojem>Nebo jen element.</pojem></odstavec>

<xs:element name="odstavec">
   <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="pojem" type="xs:string"/>
         <xs:element name="odkaz">
            <xs:complexType>
               <xs:simpleContent>
                  <xs:extension base="xs:string">
                  <xs:attribute name="url" type="xs:anyURI"/>
                  </xs:extension>
               </xs:simpleContent>
            </xs:complexType>
         </xs:element>
      </xs:choice>
   </xs:complexType>
</xs:element>
```

Zdrojový kód 1 Ukázka XML dokumentu a XSD souboru s elementem se smíšeným obsahem [5]

Podpora jmenných prostorů

Namespaces, v překladu jmenné prostory, slouží k použití definic prvků z různých skupin XSD souborů. Jeden jmenný prostor zahrnuje skupinu schémat, které popisují určitou problematiku (matematické vzorce, vektorovou grafiku, chemické vzorce). A tyto různé definice se pak v daném XML dokumentu využívají za pomocí prefixu. To pro XML Schema znamená, že máme skupinu XML schémat, z nichž můžeme použít nějaký typ pro námi vytvořená schémata.

Názvy prvků z různých prostorů se mohou shodovat a tak se používají pro přesnou definici, z jakého jsou jmenného prostoru prefixy. Jednoznačné určení jmenného prostoru je díky lokátoru URI, který musí být zapsán v absolutním tvaru. Použité jmenné prostory se zapisují do kořenového elementu, aby se daly používat v celém XML dokumentu. Jmenné prostory se dědí potomky elementu, pro něž je jmenný prostor definován. To znamená, že u potomků nemusíme zapisovat prefix daného jmenného prostoru, jelikož je již jmenný prostor určen v rodiči tohoto elementu. Pokud však chceme použít prvek z jiného jmenného prostoru, musíme zapsat prefix požadovaného jmenného prostoru a název prvku jenž chceme použít.[5][8][1 str.47-67]

```
<xsd:simpleType name="fakultaType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="FCHT"/>
      <xsd:enumeration value="FEI"/>
      <xsd:enumeration value="DFJP"/>
      <xsd:enumeration value="FES"/>
      <xsd:enumeration value="FF"/>
      <xsd:enumeration value="FZS"/>
      <xsd:enumeration value="FR"/>
   </xsd:restriction>
</xsd:simpleType>
```

Zdrojový kód 2 Ukázka schemata, které bude použito v následujícím schématu za pomocí jmenných prostorů [autor]

```
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:a="http://kaspar.xml.cz/pokus/upce" >
<xs:element name="studentAddress">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="student" type="xs:string" />
         <xs:element name="CP" type="xs:string" />
         <xs:element name="Ulice" type="xs:string" />
         <xs:element name="Mesto" type="xs:string" />
         <xs:element name="Zeme" type="xs:string" minOccurs="0" />
         <xs:element name="PSC" type="xs:string" />
         <xs:element name="fakulta" type="a:fakultaType"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```

Zdrojový kód 3 Ukázka schématu, které za pomocí jmenných prostorů používá typy z jiných schémat [autor]

2.6. Komprese XML

Ačkoliv je formát XML velmi oblíbený má jednu nevýhodu, která je mu v poslední době velmi často vytýkána. Jedná se o velký prostor, který zaujímají pouze značky. Může se stát, že právě tyto značky, jenž určují strukturu informace, zabírají v některých případech několikanásobek prostoru oproti neseným informacím. Zde je nasnadě použití jiného formátu, jenž by nezabíral tolik prostoru. To by se většinou řešilo implementací vlastního formátu, který by zajišťoval stejnou informační hodnotu, ale s daleko menšími nároky na prostor.

Jistě by se jednalo o efektivní řešení z hlediska úspory prostoru, ale naráží se zde na další problém a tím je možnost integrace aplikace, která by na vstupu očekávala daný formát. XML má tu výhodu, že je to přenositelný, platformě nezávislý formát s možností validace proti schématům. Ale formát, který by byl vlastní produkce, rozhodně podobné vlastnosti mít nebude. Navíc by se musela řešit složitá kontrola vstupu, aby špatný formát dat na vstupu nezpůsobil pád aplikace. To je u XML řešeno validací, která tyto problémy s chybami na vstupu přenáší na stranu producenta XML dokumentů, nikoliv však na stranu konzumenta.

Právě z těchto důvodu je v současné době velmi skloňovaným tématem komprese XML dokumentů. Přece jenom se jedná o znakový formát, a tak se komprese doslova sama nabízí. Ale pokud by na XML dokument byla použita standardní znaková komprese, nebyl by kompresní

poměr příliš dobrý. Navíc by se tím ztratila možnost pracovat s XML. Proto se pro kompresi používají různé algoritmy pro zlepšení možností komprese XML dokumentů a přístupu k nim v komprimovaném stavu.

Standardní kompresory typu gzip neumožňují práci s komprimovaným XML dokumentem. Nevýhodou je, že při použití v praxi ztrácí na efektivitě při zpracování. To z důvodu, že nejdříve musí daný komprimovaný XML dokument dekomprimovat a až potom se s ním mohou provádět jakékoliv operace. Tento přístup zabírá nejen čas, ale i systémové prostředky.

Moderní algortimy pro kompresi XML mají tu výhodu, že i nad komprimovaným XML dokumentem mohou provádět dotazovací operace. Tento příklad šetří jak čas, tak i systémové prostředky.

Tyto algoritmy se dají dělit do dvou následujících skupin:

- Off-line komprese
- a On-line komprese.

Off-line kompresory se používají v případech, kdy nepotřebujeme proudové generování komprimovaných XML dokumentů. Je to způsob vhodný pro archivaci XML dokumentů na úložišti s možností pozdějšího přístupu.

Naproti tomu On-line kompresory umožňují proudové generování komprimovaných XML dokumentů. To se využívá především u messagingových systémů, kdy se zasílá velké množství XML zpráv a je nutné tyto zprávy komprimovat, aby se nezahlcovaly přenosové linky.

XML mají pro kompresi jednu výhodnou vlastnost a to možnost definice struktury pomocí schémat. To v praxi znamená, že schémata jsou předpisy všech možných výskytů elementů, jejich názvů i názvů jejich atributů. Pokud se jedná o strukturu XML dokumentu, tak ve schématu může být uvedeno, že na některém místě se vyskytuje buď jeden element nebo element druhý. To se pak v kompresi projeví pouze jednobitovým kódem.

Je-li použito XML Schema, dostává komprese nový rozměr z důvodu definice datových typů a podle nich zvolené kompresní techniky. Problém nastává, pokud schémata nejsou k dispozici například při dekompresi. Tyto komplikace se řeší tak, že se schémata stanou součástí komprimovaného XML dokumentu. [1 str.200-216]

Kompresní algoritmus	Dotazy	Komprese	Využití
XMill	-	kontejnerová, slovníková	Archivace
XMLPPM	-	PPM	archivace, on-line přenos
XGrind	XPath	Huffman	Archivace, dotazování
XPress	XPath	Huffman, typově orientovaná	archivace, dotazování
XQueC	XQuery	kontejnerová, Huffman, ALM	archivace, dotazování
BiM	-	automat, typově orientovaná	on-line přenos
ASN.1 XER	-	kódovací pravidla	on-line přenos
Fast Infoset	-	slovníková, typově orientovaná	on-line přenos

Tabulka 2: Přehled kompresních algoritmů a jejich vlastností [1str.216]

3. Mapování XML do databáze

Propojení databázových systémů a XML je logické. Slouží totiž ke stejnému účelu a tím je uchovávání dat. Navíc některé vlastnosti databázových systémů mohou navíc příznivě ovlivnit práci s XML dokumenty. Pro příklad si můžeme uvést indexy a transakční zpracování. Mapování XML dokumentů do relačně-objektových databázových systémů je proces, při kterém se XML dokumenty transformují tak, aby vyhovovaly navrhnutému datovému modelu v databázi. Existuje mnoho různých technik, jak data z XML dokumentů uložit do databáze. Navíc jejich použití musí být logické a efektivní. Jako při každém zpracovávání XML dokumentů, jsou i zde jsou zohledněny výše zmíněné přístupy ke XML dokumentům a těmi jsou [1 str.148-149]:

- Dokumentově orientované XML
- a datově orientované XML.

3.1. Dokumentově orientované techniky

S dokumentově orientovaným XML souvisí pojem round tripping, který se vztahuje k procesu uložení XML dokumentu do databázového systému a jeho opětovnému načtení. Úroveň round trippingu pak uvádí podobnost mezi původním XML dokumentem a XML dokumentem načteným z databázového systému. Čím vyšší úroveň, tím větší podobnost mezi ukládaným XML dokumentem a načteným XML dokumentem.

Dokumentově orientované XML se do databáze ukládá jako celistvý blok XML. Využívají se k tomu datové formáty BLOB[1] a CLOB[2]. Tyto formáty umožňují uložení XML dokumentu tak, že se jednoduše zpětně načte celý XML dokument. Znovuzískání XML dokumentu z databáze je rychlé a úroveň rountrippingu je maximální, jelikož jsou XML dokumenty shodné. Tento přístup nám sice dovoluje jednoduché načtení z databázového systému, ale o to těžší dotazování. Specificky je v tomto případě je možné použít pouze fulltextové vyhledávání.[1 str. 1149-150]

[1] Binary Large Object je obecný datový typ, který explicitně vyjadřuje, že databáze nemá informace o tom, jak jeho obsah interpretovat.

2 Character Large Object je kolekce znakových dat vi, obvykle uložena na jiném místě, které je odkazováno v tabulce samotné.

3.2. Datově orientované techniky

V datově orientovaných technikách se již objevuje pojem mapování, který představuje převod XML dokumentů do databází a zpět. K tomuto účelu slouží mapovací metody. XML dotazy jsou transformovány do formy SQL dotazů nad tabulkou, popřípadě tabulkami. Datově orientované techniky mají nízkou úroveň round trippingu, poněvadž nenačítají XML dokument jako celek. Není kladen důraz na přesnou posloupnost elementů a neexistuje podpora pro elementy se smíšeným obsahem.

Příklady datově orientovaných technik mohou být databáze s podporou XML nebo middleware. Databáze s podporou XML realizuje přenos dat pomocí svých vlastních implementovaných funkcí v rámci databáze. Middleware je software pro integraci, který zasílá data formou XML a ta se pak ukládají do DB. Software sám řeší převod XML dokumentů do formy vhodné k uložení do tabulek v databázi. Příkladem takového softwaru je TIBCO, WebSphere, SAP a NetWeaver.[1 str.150]

3.3. Mapovací metody

Mapovací metody slouží k přenosu dat mezi XML dokumenty a tabulkami relační databáze. Mapovacích metod je několik a můžeme je dělit následovně:

- Generické mapování,
- schématem řízené mapování,
- uživatelsky definované mapování.

3.3.1. Generické mapování

Tento způsob mapování nevyužívá žádná schémata. To umožňuje ukládat i XML dokumenty, které splňují jen základní pravidla pro vytvoření XML dokumentu. To řeší především situaci, kdy nejsou schémata dostupná. Tento přístup nám sice poskytuje značnou volnost při ukládání, ale ta nemusí být vždy žádána. V následující části bude uvedeno několik metod.

Tabulkové mapování

Nejjednodušší varianta generického mapování, která ukládá jen určitou kolekci XML dokumentů. Předpoklad definice databázového schématu přímo v XML dokumentu.[1 str. 151]

```
<Databáze>
   <Tabulka název="Studenti" sloupce="ID Jméno Prijmeni Login" typy="number varchar ...">
      <Řádek>
         <Sloupec1>1</Sloupec1>
         <Sloupec2>Jan</Sloupec2>
         <Sloupec3>Hejda</Sloupec3>
         <Sloupec4>jhejda1</Sloupec4>
      </Řádek>
      <Řádek>…</Řádek>
      <IntegritniOmezeni>…</IntegritniOmezeni>
   </Tabulka>
</Databáze>
```

Zdrojový kód 4: Ukázka příkladu tabulkového mapování [1 str. 151]

Mapování generického stromu

Tato metoda má na vstupu XML dokument, jehož datovým modelem je orientovaný strom. Vnitřní uzly tohoto stromu obsahují jednoznačné identifikátory, listy stromu mají přiřazeny hodnoty atributů nebo textový obsah elementů, přičemž každá hrana se označuje názvem elementu nebo atributu. Hrany, které vycházejí z uzlu, pak přestavují podelementy nebo atributy elementu představovaného hranou do uzlu vcházející.

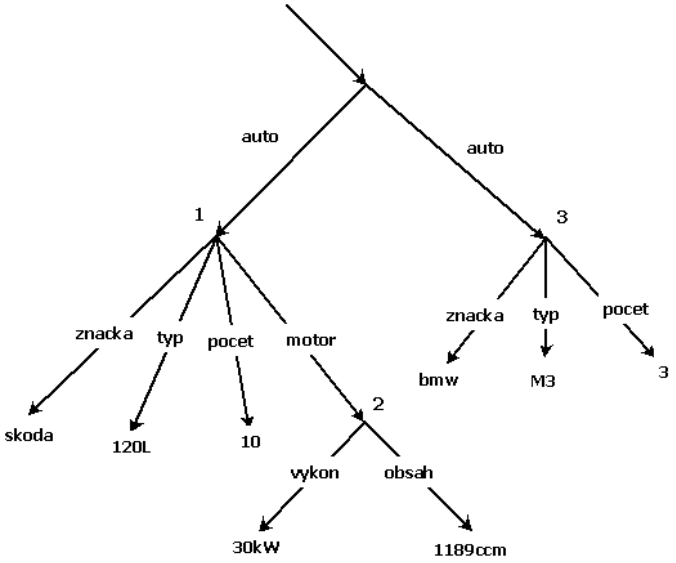
Tento strom lze do databáze zapsat hned několika způsoby. Všeobecně jsou doporučovány tyto:

- Mapování hran,
- atributové mapování,
- univerzální mapování,
- normalizované univerzální mapování.

```
<auto id=1 znacka=”skoda">
   <typ>120L</typ>
   <pocet>10</pocet>
   <motor id=2>
      <vykon>30kW</vykon>
      <obsah>1189ccm</obsah>
   </motor>
</auto>

<auto id=3 znacka="bmw">
   <typ>M3</typ>
   <pocet>3</pocet>
</auto>
```

Zdrojový kód 5: Ukázka XML dokumentu pro techniku mapování generického stromu [autor]



Obrázek 2: Ukázka stromové reprezentace XML dokumentu [autor]

Mapování hran ukládá hrany stromu do jedné jediné tabulky se složeným klíčem ze sloupců IDZdroje a pořadí. Jeden řádek udává identifikátor počátečního uzlu hrany, jméno, typ hrany (element nebo atribut), pořadí hrany v rámci hran sousedních uzlů a nakonec také identifikátor cílového uzlu. Následuje popis struktury.

`Hrana(IDzdroje, pořadí, název, typ, IDcíle)`

Další metodou je metoda atributového mapování. Tato metoda používá pro každý název hrany zvláštní tabulku. Tím nám ve struktuře tabulky na rozdíl od předchozí metody ubyde sloupec název, ale zvýší se počet tabulek a s tím související spojování tabulek. To má samozřejmě za následek zneefektivnění dotazování. Jak již bylo řečeno, metoda atributového mapování má shodnou strukturu jako metoda mapování hran, pouze s jedním rozdílem, že je vynechán sloupec název.

`Hrana(IDzdroje, pořadí, typ, IDcíle)`

Metoda univerzálního mapování má opět všechny hrany uloženy v jediné tabulce. Tato tabulka odpovídá vnějšímu spojení tabulek atributového mapování. Dotazy jsou na této tabulce vykonávány efektivně, ale nevýhodou je však řídkost informací v řádku, což tedy znamená velký počet prázdných hodnot sloupců.

`Universal (IDzdroje, pořadí_atrib1, název_atrib1, typ_atrib1, IDcíle_atrib1, pořadí_atrib2, název_atrib2, typ_atrib2, IDcíle_atrib2,…, pořadí_atribk, název_atribk, typ_atribk, IDcíle_atribk)`

Metoda normalizovaného univerzálního mapování kombinuje předchozí přístupy tak, aby odstranila jejich nevýhody a naopak využila jejich výhod. Hlavní tabulka obsahuje jen jeden záznam pro každý název hrany. Ostatní hrany jsou uloženy v tabulkách přetečení. Tyto tabulky přetečení mají shodnou strukturu jako tabulky atributového mapování. Tímto se metoda snaží co nejvíce omezit počet tabulek a počet prázdných hodnot.[1 str. 151-153]

Mapování struktury stromu

Opět se jedná o metodu, která transformuje ukládaný XML dokument do orientovaného stromu. Každý uzel stromu je definován následovně:

$$N=(t,l,c,n) \tag{1}$$

t ... typ uzlu

l ... název uzlu

c ... textový obsah uzlu

n ... seznam jeho dětí

Uložení této n-tice do databáze je možné několika možnými způsoby. V následující části budou nastíněny některé možnosti realizace tohoto uložení. Složky t, l a c mohou být uloženy v jedné tabulce.

```
Uzel_Stromu(IdUzlu, TypUzlu, NázevUzlu, Text)
```

Realizace jednotlivých algoritmů pro mapování se budou lišit různými způsoby ukládání seznamu dětí do databáze. Jako první je možné uvést způsob, při němž se využívá primárních a cizích klíčů. To v praxi znamená, že každý uzel obsahuje svůj identifikátor a identifikátor rodičovského uzlu, kdy je identifikátor rodičovského uzlu je cizím klíčem. Tento způsob nám zajišťuje velmi snadnou aktualizaci dat v tabulce, avšak dotazy nad tabulkou budou značně náročné, jelikož bude docházet ke spojování tabulek. Struktura tabulky pak bude vypadat následovně.

```
Uzel_Stromu_FK(IdUzlu, IdRodiče, TypUzlu, NázevUzlu, Text)
```

Další variantou zpracování je DF algoritmus. Tento algoritmus pracuje tak, že každý uzel identifikuje pomocí dvojice hodnot, které reprezentují dobu při průchodu stromu do hloubky. Přičemž hodnota u_{min} je doba navštívení uzlu a u_{max} je doba opuštění uzlu, které budou zároveň klíčem. Z toho vychází následující struktura tabulky.

```
Uzel_Stromu_DF(IdMin, IdMax, TypUzlu, NázevUzlu, Text)
```

Tento algoritmus umožňuje efektivní dotazování a to z důvodu, že pouze z identifikátoru lze zjistit vztahy dvou uzlů. Máme-li například dva uzly u a v, tak v případě, kdy $u_{min} < v_{min}$ a současně $v_{max} < u_{max}$ lze tvrdit, že v je nepřímým potomkem uzlu u. Navíc díky tomuto algoritmu má rekonstrukce XML dokumentu lineární složitost.[1 str. 153-154]

Mapování jednoduchých cest

Tato generická metoda také reprezentuje XML dokument jako orientovaný strom. Předpokládá dotazy nad daty v jazyce Xpath. Tento orientovaný strom sestává ze tří různých typů uzlů a to konkrétně:

- Elementové,
- atributové,
- textové.

Elementové uzly mají název a narozdíl od ostatních typů obsahují i seznam uzlů dětí. Atributové mají pouze název a hodnotu. Textové pak mají pouze textovou hodnotu. Tato metoda používá algoritmus, při němž je ukládána úplná jednoduchá cesta ke každému uzlu. Takto uložená cesta by nestačila k úplné rekonstrukci XML dokumentu, jelikož nespecifikuje pozici uzlu v dokumentu a ani pořadí. Proto se vše ukládá do relačního schématu, které tyto informace bude udržovat.

```
Element(IdDok, IdCesty, pořadí, pozice)
Atribut(IdDok, IdCesty, hodnota, pozice)
Text(IdDok, IdCesty, hodnota, pozice)
Cesta(IdCesty,textCesty)
```

Jak je již patrné z názvů tabulek, tak tabulka Cesta obsahuje všechny cesty, zatímco ostatní tabulky obsahují pouze informace o elementových, atributových a textových uzlech. Cesta je

definována identifikátorem (*IdCesty*) a skutečnou hodnotou, která popisuje cestu k uzlu ve sloupci *textCesty*. Identifikátor dokumentu (*IdDok*) definuje, do jakého dokumentu uzel patří. Sloupec *pořadí* udržuje informaci o pořadí elementového uzlu v rámci sousedních elementů. Navíc atributové a textové uzly také udržují informaci o své hodnotě ve sloupci *hodnota*. A všechny uzly také udržují informaci o své pozici v rámci dokumentu ve sloupci *pozice*.[1 str. 154-155]

3.3.2. Schématem řízené mapování

Tento druh metod využívá pro tvorbu databázového schématu definici XML schémat. Do vytvořeného databázového schématu jsou následně uložena data z XML dokumentů. Metody se snaží o vytvoření co možná nejoptimálnějších databázových schémat z poskytnutých XML schémat.

Metody využívají z hlediska XML schémat buď DTD nebo silnější XML Schema, které ovšem není tak rozšířené jako DTD. Z hlediska databázových schémat využívají buď relační (DTD), nebo objektově relační (XML Schema).

Základní idea, z níž vychází veškeré schématem řízené metody je definována následujícími charkteristikami:

Shrnutí principů a charakteristik schématem řízených metod

- Podelementy, které se vyskytují v dokumentu maximálně jednou, jsou mapovány do stejné tabulky jako rodičovský element, tedy odpovídající elementy jsou spojeny (inlining).
- Podelementy s nepovinným výskytem jsou mapovány na sloupce s prázdnými hodnotami.
- Podelementy s vícenásobným výskytem jsou mapovány do samostatných tabulek, vztahy element-podelement jsou pak mapovány pomocí primárních klíčů, cizích klíčů a referenčních integrit.
- Alternativní podelementy jsou mapovány buďto do samostatných tabulek (podobně jako v předchozím případě), nebo do jediné univerzální tabulky, ovšem s mnoha prázdnými hodnotami.

- Pokud je třeba zachovat pořadí sousedních elementů, je tato informace obvykle uložena do speciálního sloupce.
- Elementy se smíšeným obsahem obvykle vůbec podporovány nejsou. Důvodem je následné zkomplikování a zpomalení algoritmu.
- Přes předchozí optimalizace vyžaduje obvykle rekonstrukce XML fragmentů spojení většího množství tabulek, operace tudíž není příliš efektivní.

Metody řízené schématem se dají ještě rozdělit na metody fixní a flexibilní, přičemž fixní metody vytváří databázové schéma striktně podle XML schémat a flexibilní se databázové schéma snaží dále vylepšit s využitím ukázkové sady XML dokumentů.[1 str.155-156]

Mapování spojující tabulky

Jedná se o fixní, schématem řízenou metodu, která, jak již název napovídá, pracuje na principu spojování tabulek. Je zde několik verzí tohoto algoritmu s různou úrovní optimalizace. Tato metoda pracuje s DTD schématy, které jsou převáděny do paměťové reprezentace DTD grafu. Uzly tohoto grafu reprezentují následující prvky:

- Elementy vyskytující se ve schématu pouze jednou,
- atributy,
- a operátory (reprezentováno každé jejich použití).

Hrany pak v tomto grafu reprezentují jednotlivé vazby mezi těmito prvky. Mezi operátory jsou v grafu použity jen následující * a ?. Ostatní operátory, které jsou v DTD schématu použity, jsou těmito operátory nahrazeny. Převod vzniklého DTD grafu na relační schéma se řídí pravidly, která jsou v různých algoritmech různě optimalizována.

Základní verze pravidel pro mapování na relační schéma jsou poněkud jednoduchá a skýtají pouze dvě pravidla:

- Pro každý výskyt elementu vytvořena jedna tabulka.
- Maximální množství potomků je mapováno do tabulky rodiče, v ideálním případě jsou tabulky rodičů a potomku spojené.

Výjimku u druhého pravidla tvoří podelementy s operátorem * a rekurzivní podelementy. Tyto podelementy jsou uloženy v samostatných tabulkách a kromě primárních a cizích klíčů jsou také spojeny pomocí referenčních integrit. Nevýhodou této metody je tvorba vysokého počtu relací, které vyplývají z druhého pravidla.

Na tuto verzi navazuje verze další, která se velké množství relací snaží eliminovat sdílenými elementy. To znamená, že pokud je někde v DTD grafu použit jeden a ten samý element vícekrát, je pro něho vytvořena pouze jedna tabulka. Tato tabulka je posléze sdílená všemi výskyty tohoto elementu za pomocí primárních a cizích klíčů. Zde se používá pojem vstupní stupeň, který znamená počet rodičovských uzlů. Pravidla, jimiž se řídí tato verze mapování, jsou následující:

- Uzly se vstupním stupněm vyšším než jedna jsou uloženy do vlastních sdílených relací.
- Uzly se vstupním stupněm jedna jsou mapovány do tabulky rodičů.
- Uzly se vstupním stupněm nula jsou mapovány do vlastních relací.
- Podelementy operátoru * jsou uloženy do vlastních relací.
- Jen jeden ze vzájemně rekurzivních elementů je uložen do vlastní tabulky.

Poslední verze této metody se snaží optimalizovat jak počet relací, tak i počet tabulek, vzniklých při převodu na relační schéma. A tak k předchozím pravidlům bylo doplněno ještě jedno následující:

- Uzly se vstupním stupněm vyšším než jedna, které nejsou rekurzivní nebo podelementem operátoru *, jsou namapovány do tabulky rodiče.

Jsou zde i další verze této metody, které mapují další informace o schématu. Informace mohou být následující:

- Pořadí elementů na stejné úrovni,
- seznam přípustných hodnot,
- nepovinný výskyt elementu,
- atd.

Mapování zachovávající integritní omezení

Fixní schématem řízená metoda mapující schémata XML Schema do schématu v relační databázi. Tato mapovací metoda zachovává integritní omezení, která jsou zapsána pomocí jazyka XML Schema v mapovaném schématu. Integritní omezení definované ve schématech se mapují na databázové integritní omezení. Těmito omezeními jsou myšleny přípustné hodnoty daného typu, případně jiné omezení jednoduchých datových typů.

Metoda reprezentuje XML schéma pomocí orientovaného grafu. Metoda pracuje s EER modelem reprezentovaným grafem. Uzly grafu představují elementy, složité datové typy, jejichž atributy jsou reprezentací XML atributů elementů. Vztahy jsou reprezentovány vztahy a kardinalitami mezi prvky ER modelu a jsou uloženy do pomocných struktur.[1 str. 158]

LegoDB mapování

Jedná se o flexibilní schématem řízené mapování. Nejprve je definováno fixní mapování pro dané schéma. Flexibilita pak spočívá v tom, že pro dané schéma jsou vytvořeny všechny různé transformace. Poté se vybere ta, která nejvíce vyhovuje dané aplikaci. Aplikace je zastoupena sadou XML dokumentů a sadou XML dotazů. Nad sadou XML dokumentů jsou vytvořeny podrobné statistiky (minimální a maximální hodnoty, počet různých hodnot, velikost datových typů, atd.). To v praxi znamená, že se musí vybrat takové schéma, nad kterým budou dané dotazy vykonávány nejefektivněji.

Příklad postupu flexibilní metody:

- Na XML schéma je aplikována XML transformace, a tak vznikne nové XML schéma.
- Nové XML schéma je fixní metodou mapováno na relační schéma.
- Sada XML dotazů je hodnocena vzhledem k vzniklému relačnímu schématu.

Transformacemi jsou myšleny změny XML schématu, jejichž použitím vzniknou stejné nebo obecnější schémata. Výsledné schéma po transformaci musí být alespoň tak efektivní jako fixní varianta metody.[1 str. 158-159]

Hybridní objektově relační mapování

Strukturované části dokumentů jsou mapovány do relací a neuspořádané či neúplné části dokumentů jsou ukládány do speciálního datového typu XML, který je navržen pro ukládání XML segmentů. Má velmi silnou podporu XML dotazů a jiných XML funkčností. To znamená, že algoritmus musí rozhodovat, co jsou strukturované části dokumentu a co nikoliv. To činí v několika krocích:

- Vytvořen DTD graf pro používané DTD schéma.
- Pro každý uzel vypočtena váha strukturovanosti ω.
- V grafu jsou nalezeny uzly s těmito podmínkami
- Uzel není listem
- Uzel a jeho potomci mají ω menší než je požadovaná úroveň detailu výsledného schématu
- Uzel nemá předka, jenž by splňoval předchozí podmínky
- Podgraf tvořený uzlem, jenž je definován v předchozím bodu, a jeho potomky se nahradí výše zmíněným datovým typem XML, čímž vzniká upravený graf.
- Takto upravený graf je fixní metodou mapován na relační databázové schéma.

Váha strukturovanosti se definuje následujícím vztahem:

$$\omega = \frac{1}{2}\omega_S + \frac{1}{4}\omega_D + \frac{1}{4}\omega_Q \qquad (2)$$

ω_S – kombinace hodnot pozice uzlu v grafu (hloubka) a jeho obsahu (míra strukturovanosti)

ω_D – poměr počtu dokumentů obsahující daný uzel ku počtu všech dokumentů

ω_Q – poměr počtu dotazů obsahující daný uzel ku počtu všech dotazů

[1 str. 158-159]

3.3.3. Uživatelsky definované mapování

Toto mapování je zcela v rukou uživatele. Uživatel vytvoří databázové schéma, do kterého bude probíhat ukládání XML dokumentů. Největší výhodou je maximální flexibilita. Mapování přesně vyhovuje požadavkům uživatele (aplikace), předpokládá ovšem uživatelovu znalost mnoha technologií. Určování optimálního databázového schéma na netriviálních aplikacích je poměrně složité.[1 str. 160]

3.3.4. Uživatelsky řízené mapování

Tato metoda mapování vylepšuje metodu uživatelsky definovaného mapování. Zejména díky tomu, že v menší míře spoléhá na uživatele. Využívá se zde implicitní metody, ale je zde umožněna interakce s uživatelem. Interakcí je myšleno doplnění atributů do schématu v případě, že pozměněné části XML mají být mapovány jiným způsobem, než jak je mapuje implicitní metoda. Většinou se jedná o fragment XML dokumentu, v jehož kořenovém uzlu ve schématu jsou přidány uživatelem definované atributy.

Uživatel má na výběr několik atributů, které může použít. Metoda umožňuje uživateli spojit tabulky nebo oddělit tabulky, uložit fragmentu do bloku (BLOB). Dále může také specifikovat názvy tabulek a sloupců. [1 str. 160]

Atribut	Hodnota	Funkce
INLINE	0	Pokud je přiřazen elementu, je daný fragment mapován do tabulky rodiče
TABLE	0	Pokud je přiřazen elementu, je daný fragment uložen do vlastní tabulky
BLOB_ONLY	0	Pokud je přiřazen elementu, je daný fragment uložen do sloupce typu BLOB
STORE_BLOB	0	Pokud je přiřazen elementu, je daný fragment uložen implicitně i do sloupce typu BLOB
RENAME	String	Specifikuje název sloupce nebo tabulky, do níž je uložen odpovídající fragment schématu
DATATYPE	String	Specifikuje datový typ sloupce, do něhož je uložen odpovídající fragment schématu

Tabulka 3: Přehled atributů pro uživatelsky řízené mapování [1 str. 161]

4. XSLT

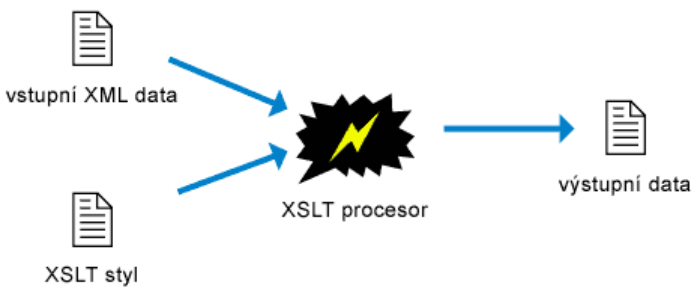
XML, jak již bylo výše zmíněno, je formát, který je pouze nositelem informace, ale už nenese informaci o vzhledu. Toto je vlastnost, která se drží základní myšlenky značkovacích jazyků, což znamená úplné oddělení obsahu od jeho vzhledu. Značky v XML dokumentu vyjadřují význam informace. Vzhled se řeší odděleně v dalším jazyce založeném na XML, je jím XSL.

Takovéto oddělení obsahu od vzhledu má příznivý vliv na informace nesené v XML dokumentu. Pokud však chceme dané informace prezentovat, zřejmě se nespokojíme s prostým výpisem XML dokumentu například ve webovém prohlížeči. Právě z tohoto důvodu je tu XSLT. eXtensible Stylesheet Language Transformations je jazyk pro transformaci zdrojových XML souborů do libovolných formátů.

XSLT byl dříve součástí XSL, jenž slouží k definování vzhledu jednotlivých elementů ze zdrojového XML dokumentu. Při zavádění standardu XSL byl však XSLT vyčleněn, jelikož by pak XSL sloužil ke dvěma různým věcem. XSL je alternativa CSS s využitím pro XML dokumenty. [17]

4.1. XSL Transformace

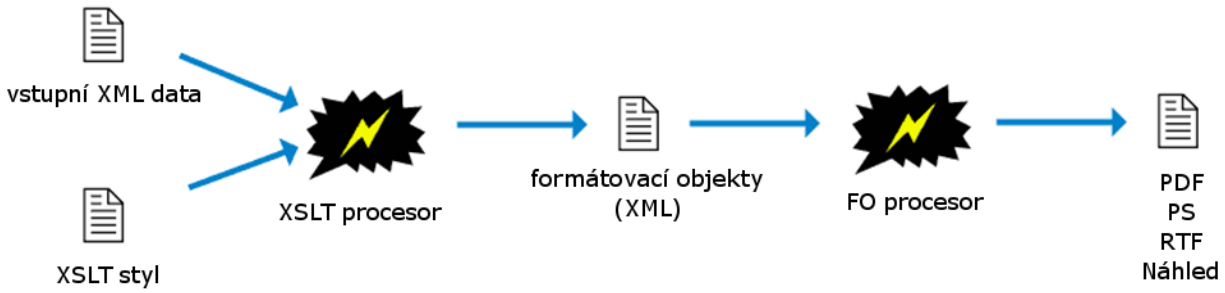
XSLT nedělá vlastně nic jiného než, že prochází zdrojový XML dokument a za pomocí XSLT stylu mění značky, v nichž jsou informace uvedeny v XML dokumentu. Tak dostaneme z XML dokumentu úplně jiný formát výstupních dat. Daná transformace na jiný formát se provádí v XSLT procesoru. Tímto procesorem se myslí programy nebo knihovny, které tyto transformace provádějí. [3]



Obrázek 3: Diagram procesu transformace XML na jiný formát [3]

4.2. XSL FO

XSL FO aneb formátovací objekty slouží k převodu XML dokumentů do formátů, které jsou vhodné pro tisk. Formátovací objekty jsou jen mezistupněm mezi XML a výsledným formátem. Transformace probíhá ve dvou základních krocích. Nejdříve se ze zdrojového XML dokumentu a XSLT stylu vytvoří pomocí XSLT procesoru XML dokument s formátovacími objekty. Následně se takto vzniklý XML dokument zpracuje FO procesorem. Výsledkem je pak hotový dokument vhodný pro tisk.



Obrázek 4: Diagram procesu transformace XML na tiskové formáty pomocí formátovacích objektů [autor]

4.2.1. Syntaxe

XSL šablona má syntaxi XML. Kořenovým elementem je element stylesheet, ve kterém jsou obsažena veškerá pravidla pro transformaci XML dokumentů. XSL je komplexní jazyk. Mohou se používat následující konstrukce:

- Podmínky,
- cykly,
- přepínače.

V kořenovém elementu se může nacházet mnoho šablon, které jsou „volány“ v případě, že se v XML dokumentu nachází element vyhovující šabloně. Šablona se uvozuje značkou template a vzor, pro který má být šablona použita, je atributem elementu template. Jedná se o atribut match.

```
<xsl:template match="faktura">
```

Zdrojový kód 6: Příklad použití šablony XSLT [autor]

Pro zpracování vnitřní struktury zpracovávaného elementu se může použít element apply-templates. Ten má za následek vytvoření výstupu do místa jiné šablony, odkud byla zavolána jiná šablona. Toto vnořené použití je výhodné použít pro složitě strukturovaný XML dokument.

Získání hodnoty elementu se provádí přes XSL element value-of. Do atributu select se zadá jen název požadovaného elementu XML dokumentu.

Pro částečně automatizované zpracování sekvence uzlů lze s výhodou použít konstrukci cyklu for-each. Následující příklad ilustruje vypsání hodnot všech elementů obsažených v elementu *„polozka“*.

```
   <xsl:for-each select="./polozka/*">
      <xsl:value-of select="local-name()" />
   </xsl:for-each>
```

Zdrojový kód 7: Příklad použití cyklu v XSLT [autor]

Podmínky se s výhodou využívají v případech, kdy se dotazujeme na určitou hodnotu. Podmínky v XSLT nemají možnost ELSE. Pro tento účel se využívají přepínače choose.

```
<xsl:if test="price &gt; 10">
   <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
   </tr>
</xsl:if>
```

Zdrojový kód 8: Příklad použití konstrukce if v XSLT [14]

Přepínače se v XSLT používají v případě, že nelze vystačit s podmínkou if. A to především tehdy, je-li nutné ošetřit stav, kdy daná hodnota nevyhovuje hodnotě očekávané.

```
<xsl:choose>
   <xsl:when test="(local-name(.)='simpleType') or (local-name(.)='complexType')">
      <xsl:value-of select="concat('type_', @name)"/>
   </xsl:when>
   <xsl:when test="(local-name(.)='key') or (local-name(.)='unique')">
      <xsl:value-of select="concat('anyKey_', @name)"/>
   </xsl:when>
   <xsl:otherwise>
      <xsl:value-of select="concat(local-name(.), '_', @name)"/>
   </xsl:otherwise>
</xsl:choose>
```

Zdrojový kód 9: Příklad použití přepínače choose v XSLT

Proměnné jsou další vyhledávanou funkcionalitou XSLT. Slouží jako uložiště hodnot, které můžeme později použít. Její hodnotu lze definovat atributem select, do něhož zapisujeme výraz Xpath. Nebo se dá také definovat jako obsah elementu variable.

```
<xsl:variable name="polozky" select="/polozky/*" />
<!—nebo-->
<xsl:variable name="polozky">
<tr>
  <th>Element</th>
  <th>Popisek</th>
</tr>
</xsl:variable>
<xsl:value-of select="$polozky"/>
```
Zdrojový kód 10: Příklad použití proměnné v XSLT [autor]

Parametry mají podobnou funkci jako proměnné. Stejně tak přečtení hodnot i jejich naplnění. Tyto parametry pak mohou být použity v rámci celé šablony, pro kterou byly definovány. [1 str. 103-113]

```
<xsl:param name="nazev" select="'Zeme'" />
<xsl:param name="vyber" select="''" />
<xsl:call-template name="SelectCountry">
   <xsl:with-param name="nazev" select="'UserCountry'"/>
   <xsl:with-param name="vyber" select="'CZ'"/>
</xsl:call-template>
```
Zdrojový kód 11: Příklad použití parametrů v XSLT [3]

5. XML a Java

5.1. Java

Seznámení s jazykem Java. Java je moderní objektově orientovaný programovací jazyk. Hlavní je, že Java je interpretovaný jazyk. To znamená, že kód v jazyce Java není přeložen přímo do strojového kódu počítače, ale do kódu, který vykonává nainstalovaný virtuální stroj JVM, takzvaný interpretr Javy. Pokud chceme programovat v Javě, je nutné mít nainstalován potřebný balík, jenž obsahuje JVM. JVM zajišťuje jakési rozhraní mezi vytvořenou aplikací a systémem. Vytvoříme-li aplikaci, kterou chceme dále provozovat na systému unixového typu, stačí na tento systém nainstalovat patřičnou verzi Javy. I přesto, že byla aplikace vyvíjena na Win32 systému, bude ji možné provozovat na výše zmíněném systému unixového typu.[10]

5.1.1. Historie Javy

Jazyk Java se vyvinul během 90. let z předchůdce s názvem Oak. Jazyk Oak byl původně navrhován pro použití v domácích elektronických spotřebičích. Hlavní myšlenkou bylo vytvořit heterogenní prostředí pro vzájemnou komunikaci různých zařízení mezi sebou. Tato technologie se však uplatnila až na poli Internetu, který v té době zaznamenal velký rozmach. Jazyk Oak představoval možnost, jak přetvořit do té doby statické stránky na dynamické.

Jazyk Oak se na stránkách vyskytoval v podobě malé sekce zdrojového kódu, který interpretoval webový prohlížeč. A tak vznikl client-side[3] jazyk, který se nazývá aplet. V roce 1995 byla zjištěna existence programovacího jazyka se shodným názvem Oak a tak byl Oak přejmenován do podoby, jak ji známe dnes, tedy jazyk s názvem Java.

5.1.2. Vlastnosti jazyka Java

Mezi základní vlastnosti jazyka Java patří:

- Zjednodušená syntaxe,
- objektový přístup,
- interpretovanost,

[3] Skript zpracovávaný na straně klienta.

- robustnost,
- automatická správa paměti,
- nezávislost na architektuře,
- přenositelnost,
- víceúlohovost.

Ikdyž je Java díky těmto vlastnostem oblíbeným jazykem, má i své nevýhody. Jednou z nich je samozřejmě delší prodleva při startu, která je zapříčiněná tím, že se program musí nejprve přeložit a posléze spustit. Další poměrně omezující vlastností je větší paměťová náročnost Javy (běhové prostředí načítáno do paměti).

5.1.3. Balíčky instalací Javy

Pro spuštění aplikace v Javě nebo snad dokonce pro její vyvoj, je nutné mít nainstalované prostředí Javy. Java je nabízena od firmy Sun ve dvou balíčcích pro instalaci potřebného prostředí. Jsou to následující:

- Java RunTime Environment (JRE) je balíček s prostředím pro spouštění již vytvořených Java aplikací (obsahuje základní knihovny).
- Java Development Kit (JDK) slouží k vývoji (kompilace, ladění a spouštění) Java aplikací. Samozřejmě že obsahuje i JRE pro spouštění Java aplikací.

V této práci bude hlavní důraz kladen na tvorbu a zpracování XML za pomocí programovacího jazyku Java a technologií s tímto jazykem spojených. Java podporuje mnoho různých technologií, které umí pracovat s XML. Jako příklad je možné uvést následující technologie-parsery:

- DOM,
- JDOM,
- SAX,
- StAX,
- JAXB.

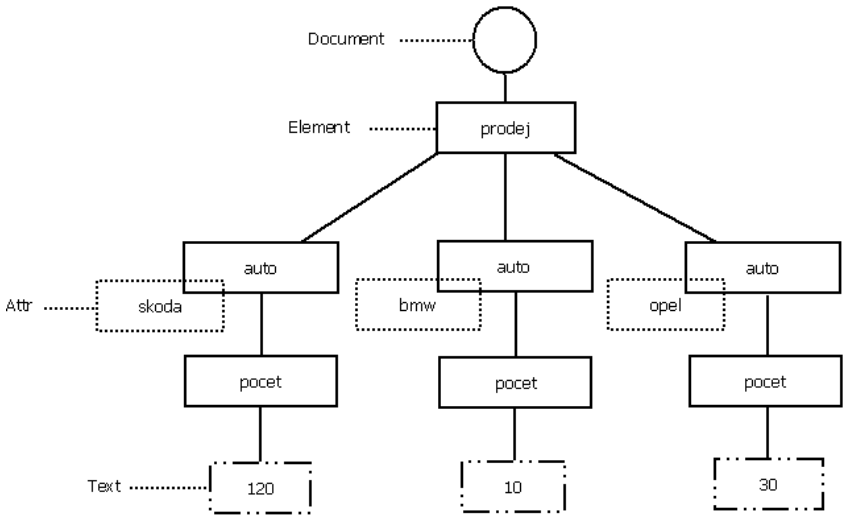
5.2. DOM

Document Object Model (DOM) je vlastně parser XML dokumentů. DOM je standardem konsorcia W3C. Tento parser se chová tak, že načte celý XML dokument a do paměti ho uloží jako stromovou strukturu. Jednotlivý prvek struktury je uzel. Používá se i anglická verze terminologie. DOM je navržen tak, že dokáže upravovat existující XML dokumenty nebo vytvářet nové. Všechny třídy, které jsou třeba pro fungování DOM jsou umístěny v Java Core API, což je standardní aplikační rozhraní pro Javu.

Pro vytvoření stromově objektového modelu XML dokumentu postačí pět základních uzlů z rozhraní od konsorcia W3C. Těmito uzly jsou Node, Document, Attr, Element, Text.

```
<?xml version="1.0"?>
<prodej>
   <auto typ="skoda">
      <pocet>120</pocet>
   </auto>
   <auto typ="bmw">
      <pocet>10</pocet>
   </auto>
   <auto typ="opel">
      <pocet>30</pocet>
   </auto>
</prodej>
```

Zdrojový kód 12: Ukázkový příklad XML dokumentu [autor]



Obrázek 5: Grafická reprezentace stromu, jenž je v paměti při použití DOM [autor]

DOM si ve struktuře stromu drží i informace o formátování (konce řádků, formátovací mezery). Tyto informace si udržuje mimo daný uzel do uzlů Text, a proto je nutné pracovat se stromem s vědomím, že se v něm tyto uzly vyskytují. [2 str. 146-156]

5.3. JDOM

Java Document Object Model je pokročilou implementací rozhraní pro přístup ke XML dokumentům dle standardu DOM. Jelikož je DOM všeobecný pro mnoho programovacích jazyků, byl speciálně vyvinut pro jazyk Java JDOM. Je na míru postavený pro jazyk Java, proto dokáže naplno využít všech jeho vlastností (přetěžování metod, kolekce, atd.). JDOM podporuje jak SAX tak i DOM zpracování XML dokumentu.

Balíček JDOM má dva základní podbalíčky sloužící ke zpracování XML dokumentů. Jsou pojmenovány příhodně Input a Output. Je velmi jednoduché odvodit, k čemu třídy v těchto balíčcích slouží. Input, jak již název napovídá, slouží k vytváření JDOM dokumentu a to buď vytvořením v aplikaci, nebo načtením ze souboru. Output pak slouží k převodu JDOM dokumentu na jiné dokumenty (výstup). Nejvýznamnější je v tomto podbalíčku třída XMLOutputter, která vypisuje dokument do proudu. Můžeme se také zmínit o podbalíčku Transform, který slouží ke XSLT transformacím dokumentu.

Usnadňuje operace s dokumentem. Načítání a editace je lehčí. Toto API optimalizuje kód, jenž je napsán. Na rozdíl od DOMu jsou konstrukce JDOMu jednodušší a kratší. Z toho vyplývá přehlednější a úspornější kód než při použití DOM. Pro parsování XML dokumentů používá SAX parser nebo DOM parser. Označeny jsou SAXBuilder a DOMBuilder. [11][12]

Příklad vytvoření dokumentu pomocí JDOMu:

```
Document doc = new Document(new Element("root").setText("Toto je kořenový element"));
```

Zdrojový kód 13: Příklad vytvoření dokumentu JDOMem [15 str. 69]

A teď jak by se vytvářel stejný dokument, ale za pomocí DOMu:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.newDocument();
Element root = doc.createElement("root");
Text text = doc.createText("Toto je kořenový element");
```

```
root.appendChild(text);
doc.appendChild(root);
```

Zdrojový kód 14: Příklad vytvoření dokumentu DOMem [15 str. 69]

V následujících příkladech si představíme nejpoužívanější konstrukce.

```
//Získání kořenového elementu
Element root = doc.getRootElement();
// Získání seznam potomků kořenového element
List allChildren = root.getChildren();
// Získání seznamu potomků element s danným jménem
List namedChildren = root.getChildren("name");
// Získání prvního výskytu potomka s daným jménem
Element child = root.getChild("name");
// Odstranění 4. potomka v seznamu
allChildren.remove(3);
// Odstranění potomka s daným názvem ze seznamu
allChildren.removeAll(root.getChildren("jack"));
// Přidání elementu na konec
allChildren.add(new Element("jane"));
// Přidání elementu na začátek
allChildren.add(0, new Element("jill"));
//Získání hodnoty atributu s daným jménem
String val = table.getAttributeValue("width");
//Je možné získat atribut jako object pro vykonávání speciálních operací
Attribute border = table.getAttribute("border");
int size = border.getIntValue();
// Nastavení nebo změna atributu
table.setAttribute("vspace", "0");
// K odstranění atributu slouží removeAttribute():
table.removeAttribute("vspace");
// Získání textového obsahu elementu
String desc = description.getText();
// Získání obsahu element bez bílých znaků
String betterDesc = description.getTextTrim();
// Nastaveni textového obsahu elementu
description.setText("A new description");
// Nastaveni textoveho obsahu funguje tak, že speciální znaky nahrazuje znakovou entitou
element.setText("<xml/> content");
// Lze vkládat I sekci CDATA do dokumentu
element.addContent(new CDATA("<xml/> content"));
```

Zdrojový kód 15: Příklady jednotlivých možných operací dokumentem pro JDOM [15 str. 69]

5.4. SAX

Simple API for XML je parser se sekvenčním přístupem ke XML dokumentům. Tento událostně řízený parser slouží pouze ke čtení XML souborů. Jedná se o alternativu technologie DOM. Oproti DOM parseru však mají výhodu menší spotřeby paměti, jelikož si v paměti neudržují strom elementů daného XML. Tím však přichází o některé výhodné vlastnosti, které DOM má.

SAX parsování je jednosměrné. To znamená, že předešle zpracovaná data nemohou být opětovně načtena bez znovu spuštění parsovací funkce. Díky těmto vlastnostem se hodí především v případech, kdy je soubor potřeba jen jednoduše zpracovat bez nutnosti pohybu mezi elementy. [2 str. 122-127]

5.5. StAX

Streaming API for XML ve zkratce StAX je rozhraní pro práci s XML jako s proudem dat. Je to výhodný přístup při vytváření a zpracování rozsáhlých XML dokumentů, pokud není potřeba znát strukturu elementů v něm obsažených. Tím dosáhneme efektivnější práce se systémovými prostředky, především s pamětí. StAX nabízí hned dvojí zpracovávání XML dokumentu, kurzorové a událostní. Kurzorové je rychlé a pracuje na nízké úrovni. Využívá rozhraní XMLSteamReader a XMLSteamWriter. Naproti tomu událostní zpracování je propracovanější a pracuje s rozhraními XMLEventReader a XMLEventWriter.

Výhody vůči rozhraní SAX:

- XML dokumenty dokáže vytvářet i zapisovat.
- Proudové zpracování velkých XML dokumentů (výhoda vůči DOM).
- Zdrojový kód na jednom místě při čtení (SAX uchovává ve 3 provázaných metodách).
- Textový obsah elementu vracen v celku (není nutnost použití StringBufferu).
- Dokument čte na žádost (pull-parser).
- Nevaliduje XML dokumenty (výhoda při neexistenci nebo nedostupnosti schémat).

Pro čtení lze využít třídu XMLStreamReader, která poskytuje mnoho metod pro zpracování XML dokumentu. Zde se může využít metoda getEventType(). Dále je nutné zmínit konstanty z rozhraní XMLConstants, které tato metoda vrací.

Nejpoužívanější konstanty jsou:

- START_ELMENT,
- END_ELEMENT,
- ATRIBUT.

Druhou možností je využití následujících metod:

- isStartElement,
- isEndElement,
- getLocalName,
- getElementText.

```
public String   getLocalName()        // Získání jména elmentu
public String   getText()            // Získání textovou hodnotu aktuálního prvku
public String   getElementText()     // Získání obsahu elementu
public int      getEventType()      //Získání typu elementu
public Location getLocation()       // Získání místa kde je XML document uložen
public int      getAttributeCount()
```

Zdrojový kód 16: Příklady použití metod StAX pro XMLStreamReader [9]

Na rozdíl od SAX čte StAX na požádání, což znamená , že StAX si pamatuje pozici, na které naposledy probíhalo čtení. A tak na rozdíl od SAX nemusí číst celý XML dokument najednou. Data tedy posílá jen v okamžiku, kdy jsou požadována.

```
URL u = new URL("http://www.cafeconleche.org/");
InputStream in = u.openStream();
XMLInputFactory factory = XMLInputFactory.newInstance();
XMLStreamReader parser = factory.createXMLStreamReader(in);
while (true) {
      int event = parser.next();
      if (event == XMLStreamConstants.END_DOCUMENT) {
         parser.close();
         break;
      }
      if (event == XMLStreamConstants.START_ELEMENT) {
         System.out.println(parser.getLocalName());
      }
}
```

Zdrojový kód 17: Jednoduchý příklad načtení pomocí technologie StAX [9]

Při zápisu XML dokumentu do proudu můžeme zapsat mnohem rozsáhlejší XML dokumenty než v případě použití DOM. Při vytvoření XMLStreamWriter je možnost nastavení kódování. Přitom tento XMLStreamWriter umí nahrazovat speciální znaky znakovými entitami (< nahrazen <). Odřádkování a osazování je nutné provádět ručně (zanky /r a /n). [9] [2 str. 196-211]

```
//nastaveni kódování při pokusu zapsat jiné kódování v XML dokumentu by nastala chyba
writer=factory.createXMLStreamWriter(out,"UTF-8");
writer.writeStartDocument("UTF-8", "1.0");
writer.writeDTD("<!DOCTYPE svg PUBLIC \"-//W3C//DTD SVG 1.1//EN\"
\"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd\">");
writer.writeStartElement("svg");
writer.writeAttribute("version", "1.1");
writer.writeAttribute("id", "Layer_1");
writer.writeAttribute("xmlns","http://www.w3.org/2000/svg");
writer.writeAttribute("xmlns:xlink", "http://www.w3.org/1999/xlink");
writer.writeAttribute("x", "0px");
writer.writeAttribute("y", "0px");
writer.writeAttribute("width","100%");
writer.writeAttribute("height","100%");
writer.writeAttribute("xml:space", "preserve");
writer.writeStartElement("defs");
   writer.writeStartElement("marker");
   writer.writeAttribute("id", "arrow");
   writer.writeAttribute("viewBox", "0 0 80 80");
   writer.writeAttribute("refX", "60");
   writer.writeAttribute("refY", "40");
   writer.writeAttribute("markerUnits", "strokeWidth");
   writer.writeAttribute("markerWidth", "8");
   writer.writeAttribute("markerHeight", "7");
   writer.writeAttribute("orient", "auto");
   writer.writeStartElement("polyline");
      writer.writeAttribute("points", "0,0 80,40 0,80");
   writer.writeEndElement();
   writer.writeEndElement();
writer.writeEndElement();
```

Zdrojový kód 18: Příklad tvorby dokumentu pomocí StAX [autor]

5.6. JAXB

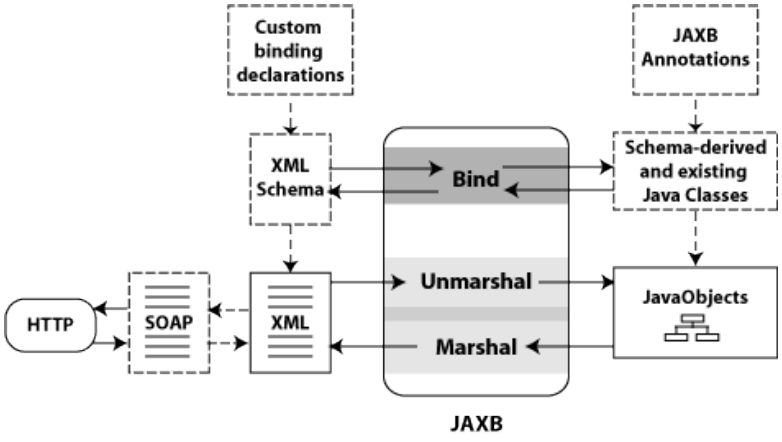
Java API for XML Binding ve zkratce JAXB. Jedná se o rozhraní, které slouží k práci s XML a třídami v jazyce Java. Za pomocí XSD schémat transformuje třídy do XML dokumentů a samozřejmě také transformuje XML dokumenty zpět do tříd v jazyce Java. Pokud je struktura XML dokumentu definována pomocí XSD schéma je možné generovat další třídy automaticky.

To se dá s výhodou využít u komunikace mezi aplikacemi, kdy jedna aplikace pošle druhé XML dokument a ta jej za pomocí XSD schématu převede do třídy v jazyce Java, se kterou následně může pracovat.

JAXB nabízí dvě hlavní funkčnosti. Jsou jimi schopnost marshalingu objektů v jazyce Java do XML a inverzní funkce, tj. unmarshaling zpět XML do objektů v jazyce Java. Jinými slovy, JAXB umožňuje ukládání a získávání dat z paměti v libovolném formátu XML, aniž by bylo nutné provádět konkrétní rutiny načítání a ukládání XML souboru pro třídní struktury vytvořené v programu. Tento přístup je podobný s xmlserializers v .Net Framework.

JAXB je zvláště užitečný, když se specifikace mění a je složitá. Pravidelné měnění XML Schema definice udržuje tyto definice synchronizovány s Java definicemi, což může být časově náročné a náchylné k výskytu chyb.

JAXB je jedno z API v EE platformě Java a je součástí Java Web Services Development Pack (JWSDP). JAXB je součástí Javy verze SE 1.6.



Obrázek 6: Schéma technologie JAXB a její vazby [4]

Jak je uvedeno v předchozím obrázku, JAXB data Binding Proces se skládá z následujících úkolů:

Bind – Váže XML Schema ke schématu odvozených JAXB Java tříd, nebo hodnotám tříd. Každá třída poskytuje přístup k obsahu přes sadu JavaBean-styl přístupových metod (to je get a set). Vazba je řízena JAXB překladačem schématu.

Unmarshal – Převede XML dokument na strom prvků Java programu, nebo objekty, které reprezentují obsah a organizaci dokumentů, které mohou být přístupné Java programem. V obsahu stromu jsou složité typy mapovány na hodnoty tříd. Jednoduché typy jsou ukládány do vlastností nebo polí uvnitř třídy a je k nim přístup pomocí get a set metod.

Marshal – Převede Java objekty zpět na obsah XML. V tomto případě Java metody, které jsou nasazeny jako WSDL operace určí součásti schématu v sekci WSDL: types.[4]

```
<xsd:complexType name="BagType">
   <xsd:sequence>
      <xsd:element name="Content" type="xsd:anyType"/>
   </xsd:sequence>
</xsd:complexType>

public class BagType {
   protected Object content;
   public Object getContent() {  return content;  }
   public void setContent(Object value) {  this.content = value; }
}
```

Zdrojový kód 19: Příklad předpisu pro typ vXSD a v Javě [16]

Příklad pro lepší pochopení bude uveden v části implementace. Bude řešen konkrétní případ z reálného světa, který nastíní většinu případů, jak lze JAXB použít.

6. Implementace evidenčního systému

6.1. Analýza systému

Ukázkové příklady použití výše zmíněných technologií jsou v této práci implementovány na systému, který byl v rámci studia zadán jako ročníkový projekt. Projekt systému pro evidenci nákladních vozů dosud nepoužíval výše zmíněné technologie. Zadání bylo pro účel diplomové práce rozšířeno, a také rozsah implementace se zvýšil. Implementace tohoto systému má za úkol ukázat použití technologií pro práci s XML dokumenty. Velký důraz byl kladen na použitelnost navrhovaného systému v praxi. V této části budou primárně popsány části systému, které používají technologie, kterými se tato práce zabývá.

6.1.1. Postup analýzy

V následující části práce bude teoreticky popsán postup tvorby systému. Od analýzy až po implementaci daného systému. Nejdříve je nutné zjistit základní požadavky na systém, které se v průběhu analýzy, návrhu a komunikace se zadavatelem budou rozšiřovat a upravovat. Dále je nutné zjistit, do jakého prostředí bude systém navrhován, kdo s ním bude pracovat.

Sběr požadavků

Pro základní orientaci sloužil dokument poskytnutý zadavatelem, který shrnoval představy o poptávaném systému. Nejvyšší důraz byl kladen na evidenci nákladních vozů a jejich životního cyklu. Životní cyklus představuje operace, které jsou s nákladním vozem prováděny v rámci firmy (nákup, opravy, revize, prodej, likvidace, atd.).

Po zpracování zadání a konzultaci se zadavatelem byly zjištěny následující funkční požadavky:

- **Evidence NV** – Zajištění evidence nákladních vozů a jejich charakteristik.
- **Evidence událostí NV** – Zajištění evidence operací s nákladními vozy, které se s nimi provádějí v rámci jejich životního cyklu (nákup, pronájem, oprava, revize, prodej, likvidace).
- **Evidence oprav** – Zaznamenání oprav a souvisejících změn charakteristických vlastností vozu.

- **Evidence napalánovaných revizí** – Umožnění komplexní správy revizí vozů. Naplánování revize pracovníkem, upozornění na končící platnost revize a záznam o provedené revizi.
- **Výpisy dat dle stanovených kritérií** – Uživatelé budou mít možnost vytváření výpisů vozového parku podle určitých kritérií.
- **Výpis přednastavených sestav** – Pro tisk bude umožněn výstup ve formě tiskových sestav vozového parku.
- **Evidence vystavených faktur** – Příslušní uživatelé budou moci vytvářet a zasílat faktury zákazníkům, kteří si od firmy zakoupí nebo pronajmou nákladní vozy.
- **Zabezpečený přístup do systému** – Přístup do systému bude podmíněn přihlášením daného uživatele.
- **Evidence najetých kilometrů** – Možnost rozšíření se naskýtá v připojení na externí systém pro monitorování pozice vozů a tím i najetých kilometrů.

Nefunkční požadavky jsou pak následující:

- **On-line přístup** – Přístup k systému odkudkoliv a kdykoliv pomocí webového rozhraní.
- **Použití technologií Java a XML** – Požadavek na použití technologií pro práci s XML dokumenty v Javě.
- **Výstup faktur a tiskových sestav ve formátu PDF** – Požadavek na možnost exportu do tiskového formátu PDF.

6.1.2. UML diagramy evidenčního systému

Ze všeho nejdříve je nutné najít a identifikovat aktéry, kteří budou se systémem pracovat. Jelikož na těchto aktérech se začnou stavět jednotlivé případy užití systému. Při konzultacích se zadavatelem bylo identifikováno hned několik typů aktérů, kteří budou popsáni v následující části.

Aktéři

Se systémem pracuje hned několik aktérů, kteří odpovídají struktuře firmy. Aktéři, kteří pracují se systémem, odpovídají zaměstnanecké struktuře firmy a mají různá oprávnění v rámci systému. Tím pádem i různé přístupy do jednotlivých částí systému. Zde je jejich shrnutí:

- **Manažer** – Manažer bude mít komplexní přehled nad vozovým parkem firmy a také komplexní správu uživatelů, kteří budou se systémem pracovat.
- **Evidenční pracovník 1** – Evidenční pracovník 1 bude mít na starosti nákup, zavedení do provozu, prodej, likvidaci, pronájem NV od firmy a pronájem NV firmě. Informace ohledně NV bude moci kontrolovat díky výstupním sestavám.
- **Evidenční pracovník 2** - Evidenční pracovník 2 bude komunikovat s opravnou NV. Dále bude zadávat a kontrolovat správné zadávání charakteristik týkající se NV, zaznamenávat a plánovat revize NV a dále bude přidělovat NV na opravu. Všechna data bude moci kontrolovat přes výstupní sestavy systému.
- **Externí opravna** – Externí opravna bude zaznamenávat informace o provedených opravách a revizích.
- **Opravna NV** – Opravna NV bude zaznamenévat informace o interních menších opravách, které si firma provádí sama.
- **Čas** – Časový údaj bude sloužit k naplánování revizí a oprav. Pokud se bude blížit vypršení revize, systém na tuto skutečnost upozorní.
- **Externí systém** – Možnost, jak rozšířit systém, je implementovat rozhraní pro přístup externího systému pro evidenci najetých kilometrů.
- **Zákazník** – Další možností, jak rozšířit systém, je zavedení zákaznických účtů pro lepší přehledzakázníků nad svými fakturami. Ale také pro lepší zpětnou vazbu od zákazníků a efektinější práci pří péči o zákazníka včetně statistik.

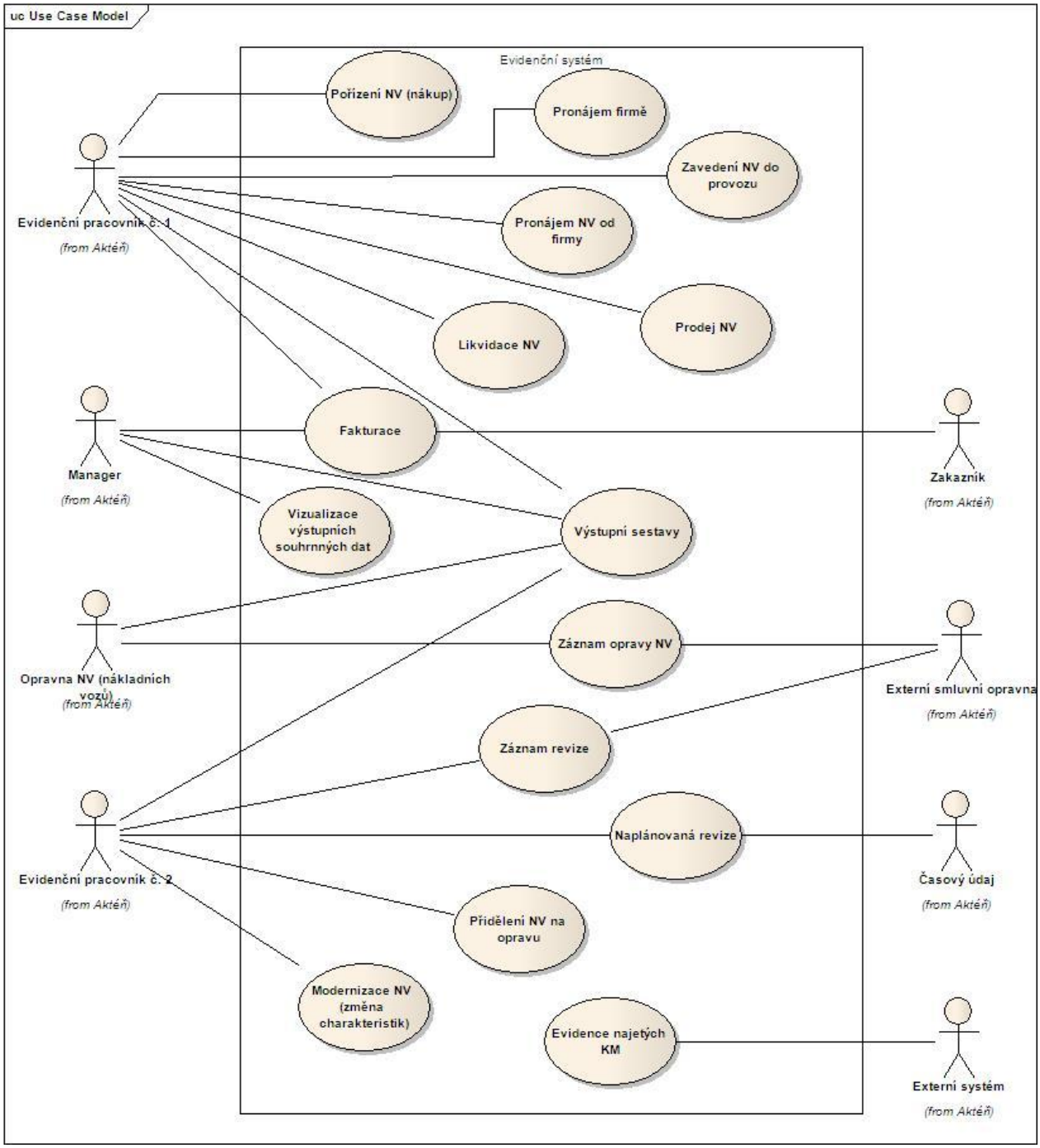
Z hlediska velké časové náročnosti nebyla uživatelská část implementována, jelikož vyšší priorita byla kladena na evidenci nákladních vozů a částí systému, které používají technologie pro zpracování XML dokumentů.

Use Case diagramy

Vytváření případů užití mělo dvě hlavní fáze. Nejprve byl vytvořen obecný případ užití pro veškeré funkčnosti požadované zadavatelem. V druhé fázi byly funkčnosti rozděleny do jednotlivých případů užití, které byly rozděleny do případů užití pro jedotlivé aktéry. Tato bližší specifikace jednotlivých případů užití pak slouží k lepšímu pochopení chování systému a k odstranění nejednoznačností. Díky tomu se předejde chybám v možnosti různé interpretace jednotlivých případů užití.

Obecný Use Case

Tento případ užití je zobecněnou realizací funkčních požadavků, které byly na počátku specifikovány. Zahrnuje také všechny identifikované aktéry. A tak poskytuje obecný, ale ucelený pohled na navrhovaný systém.



Obrázek 7: Obecný Use Case diagram pro evidenční systém [autor]

UC – Manager

Tento případ užití popisuje možnosti aktéra *Manager*, jak může pracovat se systémem. Manager má po úspěšném přihlášení komplexní přehled o vozovém parku a může v něm vyhledávat jednotlivé nákladní vozy. Manager, jak je z obrázku vidět, má možnost vytvářet faktury, výstupní sestavy a vizualizace. Dále je tento případ užití rozšířen o *Vyhledávání NV* a *Uložení dat do centrálního uložiště*.

UC – Evidenční pracovník 1

Jedná se o případ užití popisující interakci systému s aktérem *Evidenční pracovník 1*. Tento aktér má za úkol evidenci NV a operace s tím spojené. Navíc může vystavovat faktury. I tento případ užití je rozšířen o *Vyhledávání NV* a *Uložení dat do centrálního uložiště*. Jmenovitě jsou v tomto diagramu zastoupeny následující případy užití:

- Fakturace,
- pořízení NV (nákup),
- zavedeni do provozu,
- pronájem NV od firmy,
- pronájem firmě,
- prodej NV,
- likvidace NV,
- přečíslování NV
- a výstupní sestavy.

UC – Evidenční pracovník 2

Tento diagram popisuje interakci systému s aktérem *Evidenční pracovník 2*. Ten má za úkol evidenci veškerých revizí a oprav NV. Navíc je tento diagram jako v předchozích případech rozšířen o *Vyhledávání vozů* a *Uložení dat do centrálního uložiště*. To znamená, že má k dispozici následující funkčnosti systému:

- Naplánování revize,
- záznam revize,
- přidělení vozu na opravu,

- modernizace vozu,
- přečíslování vozu,
- vyhledání vozu
- a tvorbu výstupních sestav

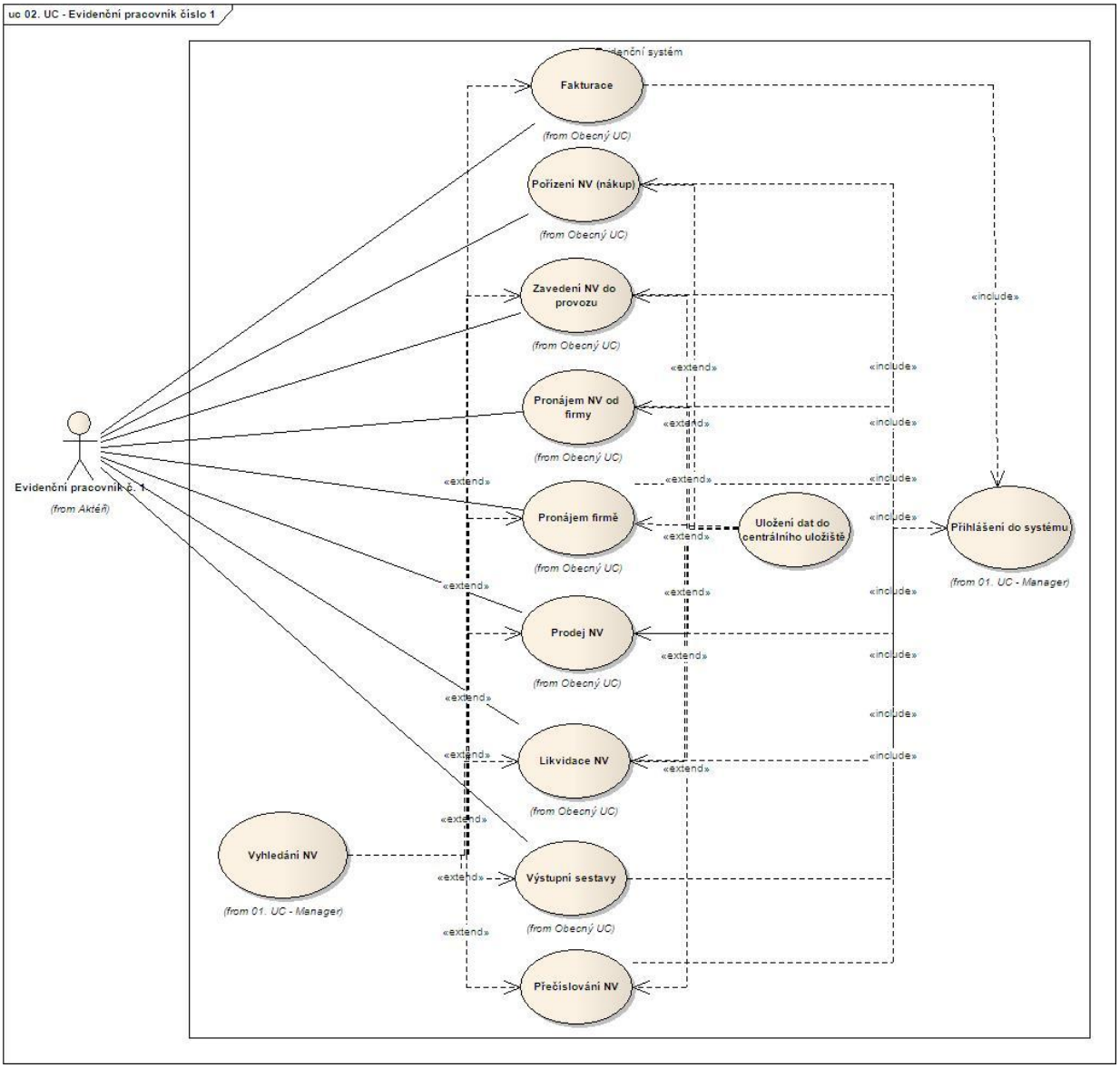
UC – Opravna nákladních vozů

Diagram popisující interakci systému s aktérem *Opravna NV*. Tato opravna reprezentuje interní opravnu firmy, která zajišťuje menší opravy. Opravna disponuje funkčnostmi *Záznam opravy*, *Výstupní sestavy*. Případ užití je rozšířen stejně jako předchozí o *Vyhledávání vozů* a *Uložení dat do centrálního uložiště*.

UC - Externí smluvní opravna

Tento diagram případu užití popisuje funkčnosti přístupné pro aktéra reprezentujícího externí smluvní opravny. Jedná se jmenovitě o *Záznam revizí* a *Záznam opravy*, které slouží k zaznamenání provedené opravy a revize do systému. Opět je tento diagram rozšířen o *Vyhledávání vozů* a *Uložení dat do centrálního uložiště*.

Další diagramy případů užití popisují části systému, které nebyly zařazeny mezi priority pro implementaci, ale spíše jako možnost pro budoucí rozšíření. Jedná se například o externí systém evidující najeté kilometry nákladních vozů. Další je diagram pro popis interakce systému s časem, který popisuje funkčnost automatické kontroly propadnutí revizí.



Obrázek 8: Diagram Use Case pro aktéra typu Evidenční pracovník 1 [autor]

6.1.3. Analytický model

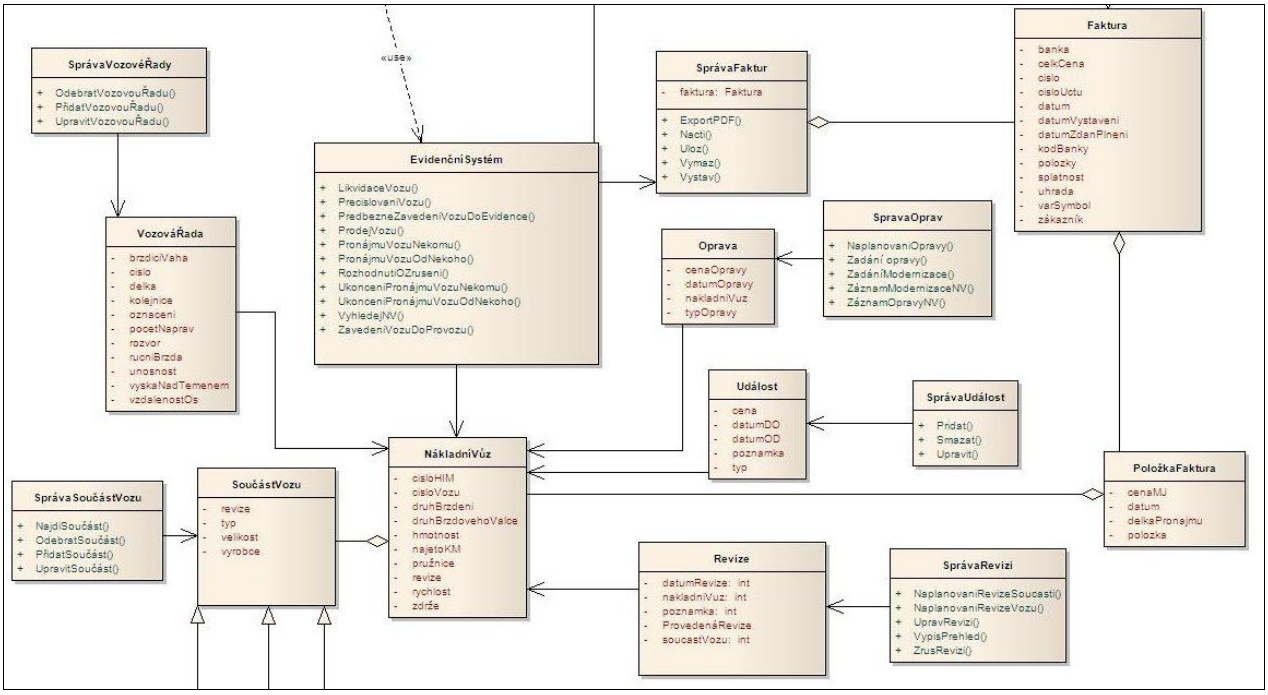
Po úspěšném vytvoření případů užití nastává další fáze. Jedná se o hledání analytických tříd, které budou obecnou reprezentací tříd, jež mají potenciál se stát třídami návrhovými. A tyto třídy pak budou implementovány.

Analytické třídy se mohou hledat několika způsoby:

- **Metoda podstatných jmen a sloves** – Za pomocí analýzy dokumentace od zadavatele je možné z podstatných jmen získat jednotlivé třídy a ze sloves operace těchto tříd.

- **Metoda CRC štítků** – Tato metoda určuje zodpovědnost tříd a možnost spolupráce s ostatními třídami za pomocí štítků. Následně se rozhoduje, které štítky budou reprezentovat třídy a jaké jejich atributy.

Pro nalezení analytických tříd byla použita metoda podstatných jmen a sloves. Z dokumentu od zadavatele bylo nalezeno základní uskupení tříd a jejich relací. Celý systém byl navrhován tak aby splňoval podmínky MVC modelu. Celkem bylo identifikováno devět základních tříd typu *Model* a 8 tříd typu *Controller*.



Obrázek 9: Výřez z diagramu analytických tříd [autor]

Z analytického modelu byl vytvořen návrh datového modelu. Tedy návrh, jak by měla vypadat struktura dat v databázi. Pro implementaci v rámci diplomové práce, muselo být databázové schéma upraveno tak, aby bylo možné co nejefektivněji použít technologii JAXB. To znamená, že data z java tříd, které reprezentují třídy typu *Model*, jsou do databáze ukládána formou XML.

6.2. Implementace systému s využitím XML technologií

6.2.1. Použité nástroje a XML technologie

Použité nástroje pro implementaci:

- **Enterprise Architekt** – nástroj pro návrh systému a datového modelu.
- **Oracle Database 10g Express Edition** – jako nejvhodnější relační databázový systém byl zvolen systém od společnosti Oracle pro svou širokou podporu XML formátu, který bude v systému hojně využíván.
- **Oracle JDeveloper Studio 11** – toto vývojové prostředí bylo zvoleno, jelikož je zde předpoklad dobré spolupráce s databázovým systémem od stejné firmy. Navíc obsahuje server pro spouštění a testování vyvíjeného systému. Toto vývojové prostředí umožňuje vývoj webových aplikací v jazyce Java. Za použití Java Server Pages, tak lze zpřístupnit aplikaci napsanou v jazyce Java.

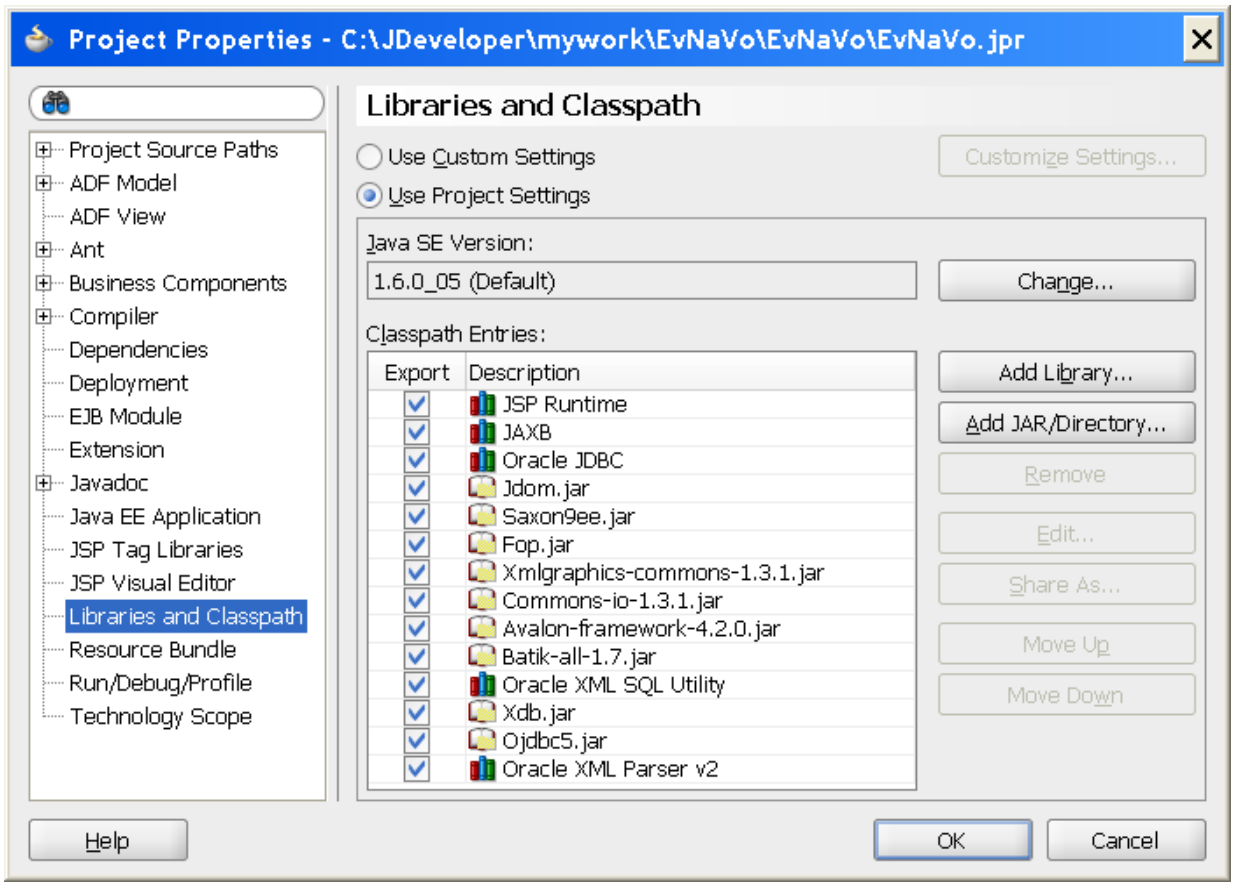
Použité XML technologie:

- **XML Schema,**
- **JDOM,**
- **StAX,**
- **XSL-FO,**
- **JAXB.**

Pro použití všech technologií bylo nutné do projectu v JDeveloperu přidat několik externích knihoven. Byly importovány následující knihovny:

- JAXB,
- JDOM,
- Saxon9ee,
- FOP,
- Xmlgraphics-commons-1.3.1,
- Commons-io-1.3.1,
- Avalon-framework-4.2.0,

- Batik-all-1.7,
- Oracle JDBC.



Obrázek 10: Ukázka externích knihoven pro funkci všech technologií [autor]

6.2.2. Vytvořené návrhové třídy

Návrhové třídy jsou upřesňující předpisy navrhovaných tříd dle analytického modelu. Tyto třídy již mají konkrétní datové typy atributu a návratových typů metod jednotlivých tříd. Slouží jako předpis pro programátora, který navrhovaný systém implementuje v konkrétním jazyce a za použití konkrétních technologií.

Vzhledem k tomu, že veškeré třídy jsou navženy podle modelu *MVC*, tak je systém složen z jednotlivých nezávislých částí. Tyto části pak zastřešuje jeden *Controller*, a tak jsou části přístupné z jednoho místa. Podrobnější popis všech částí, které byly pro účel diplomové práce změněny nebo vytvořeny nové, bude uveden v následující části práce.

Všechny výstupy z následujících tříd jsou zobrazeny pomocí JSP stránek, které tvoří vrstvu *View* z *MVC* modelu. Spolupracují s třídami typu *Controller*.

Správa faktur (package spravaFaktur)

Modul pro správu faktur spolupracuje se systémem a umožňuje uživateli vytvořit fakturu, která bude popisovat prodej nebo pronájem nákladních vozů zákazníkům. Faktura bude uložena formou XML v databázi. Také zde bude možnost exportu faktury do tiskového formátu PDF. Tato část systému bude používat hned několik technologií, jimiž se tato práce zabývá. Jmenovitě jsou zde použity technologie JDOM a JAXB.

Nejprve se na webovém formuláři vyplní jednotlivé prvky potřebné pro definici faktury. Tyto hodnoty se pomocí useBean v JSP stránce použijí a naplní se s nimi objekty, které jsou potřebné pro vytvoření faktury. Po vytvoření objektu faktury, je tato faktura transformována do podoby XML dokumentu a uložena do databáze.

Pokud je faktura uložena v databázi ve formě XML dokumentu, tak je možné tento dokument transformovat do tiskového formátu PDF za pomocí technologie XSLT. XML dokument je transformován pomoci XSL šablony do formátovacích objektů a tyto objekty jsou následně transformovány do pdf souboru. Ukázka vytvoření XML dokumentu pomocí JDOM rozhraní je v příloze.

Správa vozů

Tato část zabezpečuje správu nákladních vozů. Veškerá manipulace s nákladními vozy je realizována pomocí metod v této třídě. Slouží pro přečíslování a změnu některých charakteristik již existujícího vozu. Také poskytuje operace pro přidání a odstranění vozů.

Správa vozové řady (*package spravaVozoveRady*)

Správa vozových řad slouží k přidávání, odebírání a editaci vozových řad. Vozová řada je model, který popisuje charakteristické vlatsnosti dané vozové řady. Pokud se některému z vozů změní tyto vlastnosti, jedná se automaticky o novou vozovou řadu.

Tento balíček byl upraven pro použití technologie JAXB. Proto i odpovídající tabulky v databázi byly změněny tak, aby vyhovovaly použití JAXB. Všechna data o vozové řadě jsou

uložena ve formátu XML. Takto uložená data se jednoduše převedou na Javovské objekty, s nimiž objekty se pak nadále pracuje. Jednotlivé události jsou ukládány do elementů *udalost*. Tyto elementy jsou pak vnořeny do elementu *Udalosti*.

Správa revizí

Správa revizí slouží ke komplexní evidenci revizí vozů a jejich součástí. Revize se nejprve musí naplánovat. Když ji uživatel naplánuje, může s touto naplánovanou revizí ještě manipulovat. Pokud je už však od opravny označena za provedenou, je tato revize zaznamenána do systému jako událost s vozem a nelze jí měnit.

6.2.3. Správa oprav

Správa oprav funguje podobně jako správa revizí, jelikož jde o podobný druh manipulace s nákladními vozy. I zde se mohou opravy naplánovat. Tato část však není implementována.

6.2.4. Správa událostí

Správa událostí se zabývá evidencí a záznamem událostí v průběhu životního cyklu nákladních vozů. Postupně se přidávají události do databáze. Tyto události jsou zaznamenávány automaticky při manipulaci s nákladními vozy.

Tato část byla přepracována pro použití JAXB. Jednotlivé události jsou přidávány a ukládány ve formátu XML do databáze. Události, které se vztahují k jednomu vozu jsou ukládány do jednohozáznamu v tabulce a jsou zde ukládány identifikátor vozu a události ve formě XML. V XML je seznam všech událostí daného vozu.

6.2.5. Správa součástí vozů

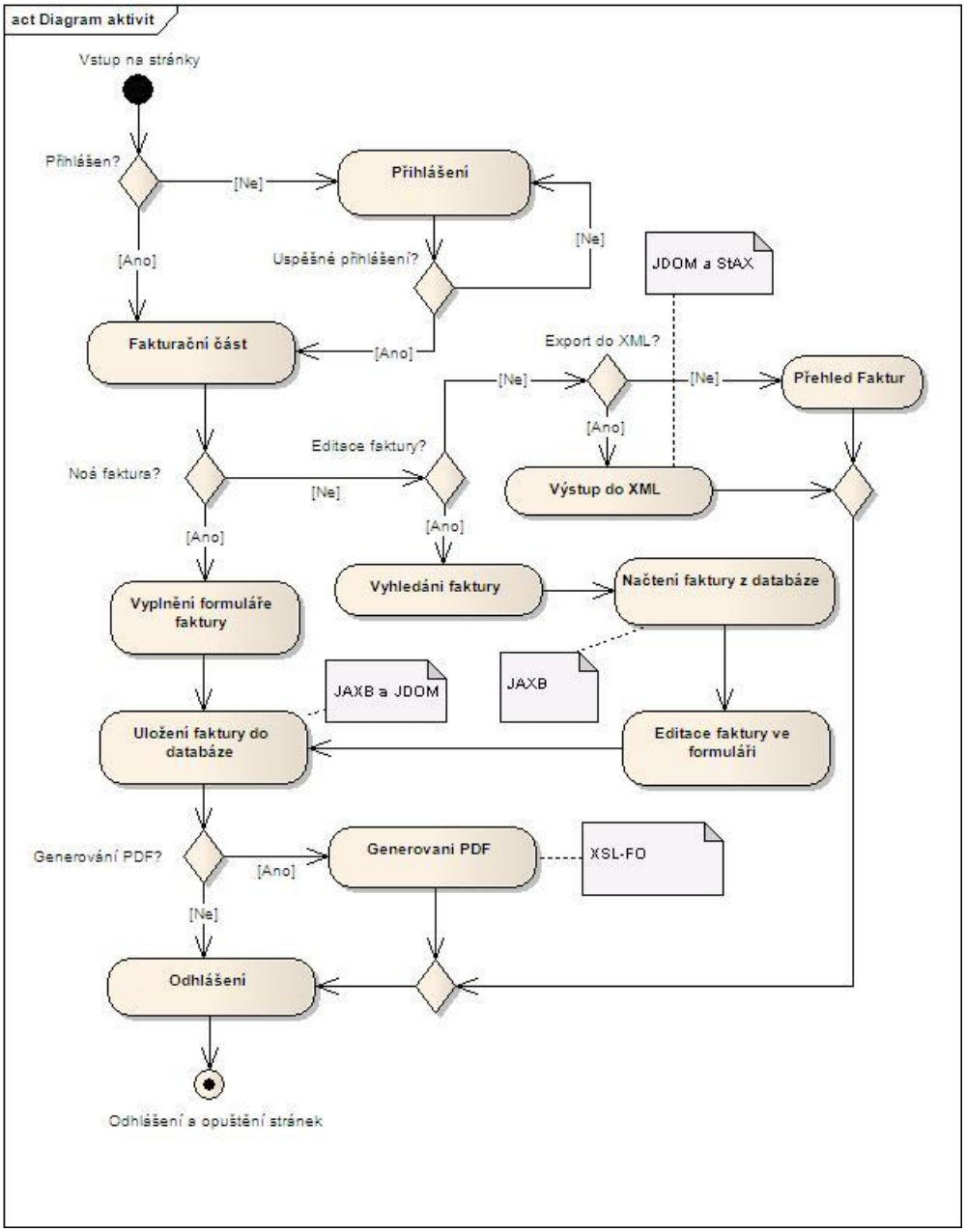
Správa součástí vozů eviduje veškeré součásti jednotlivých vozů. Tyto součásti podléhají nutnosti revizních kontrol, a tak musí být přehled o jejich existenci.

6.2.6. Evidenční systém

Tato část zabezpečuje centralizovaný přístup k systému. Jedná se o implementaci rozhraní pro celkový systém. To znamená, že implementuje veškeré operace, které jsou pro uživatele ze systému přístupné.

6.2.7. Diagram aktivit a použitých technologií

V této části budou popsány části systému využívající výše zmíněné technologie pomocí diagramů aktivit. Diagramy aktivit slouží k popsání jednotlivých aktivit uvnitř nějakého systému. Každý Diagram aktivit musí mít svůj počátek a konec.

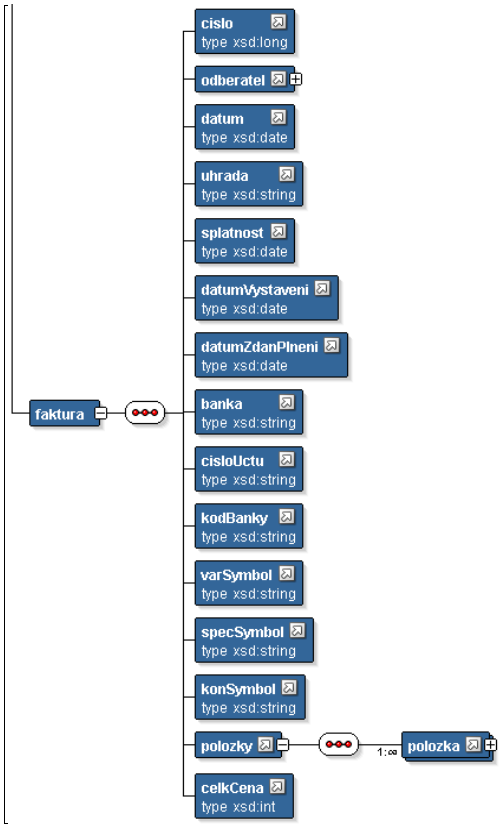


Obrázek 11: Activity diagram pro fakurační část [autor]

V předchozím diagramu je popsáno choování části systému a to konkrétně fakturační části. Pomocí poznámek jsou pak označena místa, kde jsou použity technologie pro práci s XML.

6.2.8. Využití XML Schema

XML Schema bylo použito pro vytvoření schémat jako předloha pro tvorbu javovských objektů za pomocí JAXBu. Jako základ pro vytvoření schémat byly již vytvořené návrhové třídy. Z těchto tříd byly vybrány jen třídy typu *Model*, které slouží k udržení dat. Takto vytvořené XSD soubory byly následně tranformovány na třídy typu *Model*, které se inicializovaly v systému za pomocí Unmarshalleru. JAXB Unmarshaller zajišťuje konverzi mezi XML dokumentem a java třídou.



Obrázek 12: Grafická reprezentace XSD souboru pro popis nákladního vozu [autor]

XML Schema se dá dále využít, jako validační schéma na rozhraní systému. Například pokud by se implementovalo rozhraní pro Import/Export dat z/do jiných systémů. Validace v rámci jednoho systému je nadbytečná a byla by s ní spjata zbytečná režie, jelikož se předpokládá, že v rámci systému budou všekeré XML dokumenty validní.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- definition of complex elements -->
<xsd:element name="udalost">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="typ"/>
   <xsd:element ref="poznamka" minOccurs="0"/>
   <xsd:element ref="datumOd"/>
   <xsd:element ref="datumDo"/>
   <xsd:element ref="cena"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name="Udalosti">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="udalost" maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Zdrojový kód 20: XSD soubor obsahující popis pro udalosti vozu [autor]

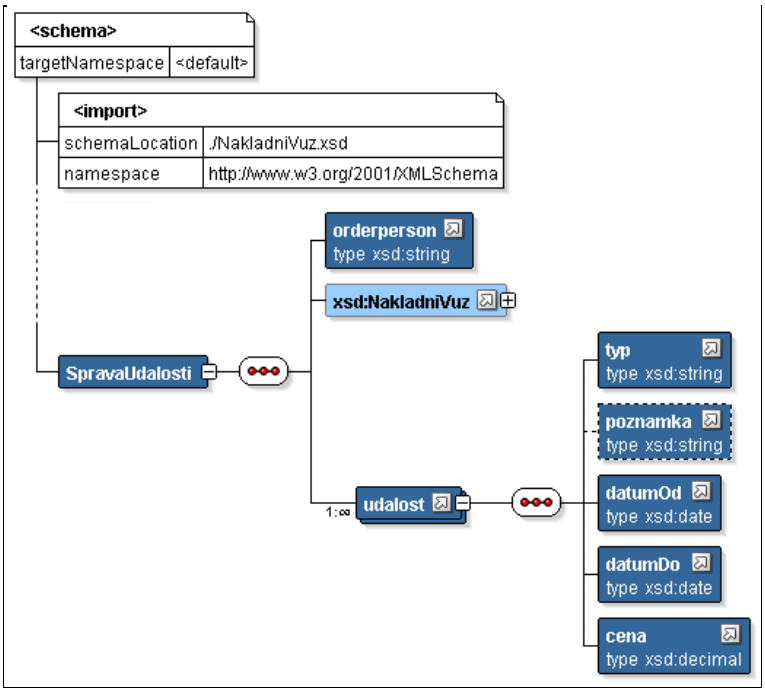
Pokud lze předpokládat, že některé ze schémat bude použito v jiném schématu jako typ, je nutné v jeho definici zadat cílový jmenný prostor. Potom lze pomocí tohoto jmenného prostoru jednoduše typ importovat.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3.org/2001/XMLSchema" >
<xsd:element name="NakladniVuz">
 <xsd:complexType>
  <xsd:sequence>
    ....
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Zdrojový kód 21: Ukázka schématu pro pozdější import do jiného schématu [autor]

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:import  namespace="http://www.w3.org/2001/XMLSchema" schemaLocation="./NakladniVuz.xsd"/>
...
<xsd:element name="SpravaUdalosti">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="orderperson"/>
   <xsd:element ref="xsd:NakladniVuz"/>
   <xsd:element ref="udalost" maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Zdrojový kód 22: Ukázka použití importovaného datového typu [autor]



Obrázek 13: Grafická reprezentace XSD souboru v JDeveloperu s importovaným typem [autor]

6.2.9. Použití parseru JDOM

Obě tyto technologie jsou si podobné v tom smyslu, že umožňují tvorbu, zápis a čtení XML dokumentů. Z toho důvodu byly použity na stejném místě v systému, jako dvě alternativy s různým přístupem ke XML dokumentu. JDOM na rozdíl od StAx nabízí mnohem více možností, jak pracovat s XML dokumentem. Avšak StAX zase nezabírá tolik paměti, jelikož si v ní nevytváří stromovou strukturu dokumentu.

```
Document doc = new Document(new Element("faktura"));
Element root = doc.getRootElement();
root.setAttribute("schemaLocation","C:/JDeveloper/mywork/EvNaVo/EvNaVo/faktura.xsd");
Element cislo=new Element("cislo");
cislo.addContent(""+tmpfaktura.getCislo());
root.addContent(cislo);
....
root.addContent(celkCena);
```

Zdrojový kód 23: Ukázka práce s rozhraním JDOM [autor]

```
XMLOutputter out = new XMLOutputter(Format.getCompactFormat().setEncoding("UTF-8"));
String faktura=out.outputString(doc);
OutputStream os;
try {
   os = new FileOutputStream("C:\\JDeveloper\\mywork\\EvNaVo\\EvNaVo\\faktura.xml");
   out.output(doc,os);
} catch (FileNotFoundException e) {
   e.printStackTrace();
} catch (IOException e) {
   e.printStackTrace();
}
```

Zdrojový kód 24: Ukázka použití JDOMu při výstupu do souboru [autor]

Uložení XML dokumentu do databáze pak probíhá následujícím způsobem:

```
public void ulozFakturu(Faktura faktura){
   DBAcces spoj = new DBAcces("java","root");
   String xml;
   try {
      xml=vytvorXML(faktura);
      String insert = "INSERT INTO faktury (faktura_id,faktura)
      VALUES ("+faktura.getCislo()+",'"+xml+"')";
      spoj.execXMLUpdate(insert);
   } catch (JAXBException e) {
      e.printStackTrace();
   } catch (IOException e) {
      e.printStackTrace();
   } catch (SQLException e) {
      e.printStackTrace();
   } finally{
      spoj.finalize();
   }
}
```

Zdrojový kód 25: Metoda pro uložení faktury do databáze [autor]

Načtení faktury v XML formátu je implemento váno následovně:

```
public void nactiXMLzDB(int idFaktury) throws SQLException {
   String poxml="";
   DBAcces spoj= new DBAcces("java","root");
   String select="select e.faktura.getStringVal() poString from faktury e where e.faktura_id=";
   select+= idFaktury;
   OracleResultSet orset = spoj.execXMLQuery(select);
   while(orset.next()){
      poxml = orset.getString(1);
   }
   is = new ByteArrayInputStream(poxml.getBytes());
}
```

Zdrojový kód 26: Metoda pro načtení faktury z databáze ve formě řetězce a její převod do Streamu [autor]

```
DBAcces spoj= new DBAcces("java","root");
ResultSet rs= spoj.execQuery(select);
Document doc = new Document(new Element("sestava"));
Element root = doc.getRootElement();
try {
   while(rs.next()){
      Element vuz=new Element("nakladniVuz");
   Element cisloVozu=new Element("cisloVozu");
   cisloVozu.addContent(""+rs.getLong("cisloVozu"));
   vuz.addContent(cisloVozu);
   Element cisloHIM=new Element("cisloHIM");
   cisloHIM.addContent(""+rs.getInt("cisloHIM"));
   vuz.addContent(cisloHIM);
   ...
   Element zdrze=new Element("zdrze");
   zdrze.addContent(rs.getString("zdrze"));
   vuz.addContent(zdrze);
   Element pruznice=new Element("pruznice");
   pruznice.addContent(rs.getString("pruznice"));
   vuz.addContent(pruznice);
   Element rychlost=new Element("maxRychlost");
   rychlost.addContent(""+rs.getInt("maxRychlost"));
   vuz.addContent(rychlost);
   root.addContent(vuz);
   }
   XMLOutputter out = new XMLOutputter(Format.getCompactFormat().setIndent("  "));
   OutputStream os;
   os = new ByteOutputStream();
   out.output(doc,os);
```

Zdrojový kód 27: Ukázka metody pro export nákladních vozů

6.2.10. Použití StAX

Použití StAXu, jak již bylo zmíněno, je výhodné pro rozsáhlé XML. Toho bylo využito v aplikaci pro import dat nákladních vozů. Vzhledem k tomu, že StAX neumí formátovat XML, tak pro export není příliš vhodný. Export by byl špatně čitelný, ale v případě velkých exportů, by použití StAXu bylo výhodnější.

Import je řešen tak, že do formuláře se vloží XML, stiskne se tlačítko *Import* a data se začnou zpracovávat. Data jsou postupně čtena jako proud a jsou vyhledávány počáteční a koncové elementy. Podle jejich názvů se pak provádějí potřebné operace.

```
   XMLStreamReader r = f.createXMLStreamReader(is);
   while(r.hasNext()){
      int eventType = r.next();
      if(eventType == XMLStreamReader.START_ELEMENT){
         if(r.getLocalName().equals("nakladniVuz")){
         }
         if(r.getLocalName().equals("cisloVozu")){
            cisloVozu=Long.parseLong(r.getElementText());
         }
...
...
         if(r.getLocalName().equals("datumRevize")){
            DatatypeFactory dtf;
            try {
               dtf = DatatypeFactory.newInstance();
               datumRevize = dtf.newXMLGregorianCalendar(r.getElementText());
            } catch (DatatypeConfigurationException e) {
               e.printStackTrace();
            }
      }
      }
      if(eventType == XMLStreamReader.END_ELEMENT){
         if(r.getLocalName().equals("nakladniVuz")){
            if(cisloVozu!=0)
            insert="INSERT INTO NakladniVuz ( cisloVozu, cisloHIM, idStav, idVozovaRada, hmotnost,
zdrze, pruznice, najetoKM, maxRychlost, datumRevize) values("+cisloVozu+", "+cisloHIM+",
"+idStav+", "+idVozovaRada+", "+hmotnost+", '"+zdrze+"', '"+pruznice+"', "+najetoKm+",
"+rychlost+", TO_TIMESTAMP('"+datumRevize+"','YYYY-MM-DD'))";
            DBAcces spoj= new DBAcces("java","root");
            spoj.execUpdate(insert);
```

Zdrojový kód 28: Ukázka použití rozhraní StAX pro čtení XML v metodě pro import nákladních vozů

6.2.11. Použití JAXB

Použití technologie JAXB je výhodné z hlediska jednoduchého převodu dat z databáze do javovských objektů. Díky již dřívější implementaci této části byly k dispozici dostatečné podklady pro vytvoření schémat XML Schema. Z vytvořených XSD souborů byly vygenerovány třídy typu *Model* pro použití v aplikaci.

Pro práci s těmito třídami byly vytvořeny *Controllery*, které zprostředkovávají veškeré operace s těmito třídami. Použití JAXB velmi zeštíhluje kód nutný pro namapování XML na objekty v Javě.

```
private String vytvorXML(Faktura f) throws JAXBException, IOException {
   OutputStream os=new ByteArrayOutputStream();
   context = JAXBContext.newInstance("spravaFaktur");
   Marshaller m =context.createMarshaller();
   m.marshal(f,os);
   ByteArrayOutputStream byte1=new ByteArrayOutputStream();
   os.write(byte1.toByteArray());
   String s=byte1.toString();
   return s;
}
```

Zdrojový kód 29: Metoda pro vytvoření XML z Java objektu [autor]

```
private Faktura vytvorObbjekt(InputStream xml) throws JAXBException {
   context = JAXBContext.newInstance("spravaFaktur");
   Unmarshaller unmarshaller;
   unmarshaller = context.createUnmarshaller();
   return (Faktura)unmarshaller.unmarshal(xml);
}
```

Zdrojový kód 30: Metod vytvářející Java objekty ze Streamu [autor]

6.2.12. Použití XMLType v databázi

Vzhledem k tomu, že databáze od společnosti Oracle má XML rozšíření, je použití typu XMLType nasnadě. Dále pak použití funkcí pro správu XML a dotazovacích jazyků XML. XMLType skýtá mnohé možnosti, jak pracovat s XML, ale v případě tenkého klienta nejsou dostupné veškeré funkce XMLType.

Pokud je použit typ XMLType, je možné v databázi využít dotazování nad XML dokumenty za pomocí XPath výrazů. Oracle sám nabízí sadu funkcí, která práci s XML značně usnadňuje. Mezi tyto funkce patří následující:

- extract(XMLType, 'XPath výraz')
- existNode(XMLType, 'XPath výraz')

Pro vyhledání faktury s daným číslem je použito fce extract, ze které se následně bere číselná hodnota pomocí funkce *getNumberVal()*. Samozřejmě, že existuje i její textová varianta *getStringVal()*, která se používá pro získávání textových hodnot.[6]

```
SELECT f.faktura.getStringVal() FROM faktury f
WHERE f.faktura.extract('//faktura/cislo/text()').getNumberVal() = 20100811;
```

Zdrojový kód 31: Příklad SQL dotazu za použití XMLType a funkce extract() [autor]

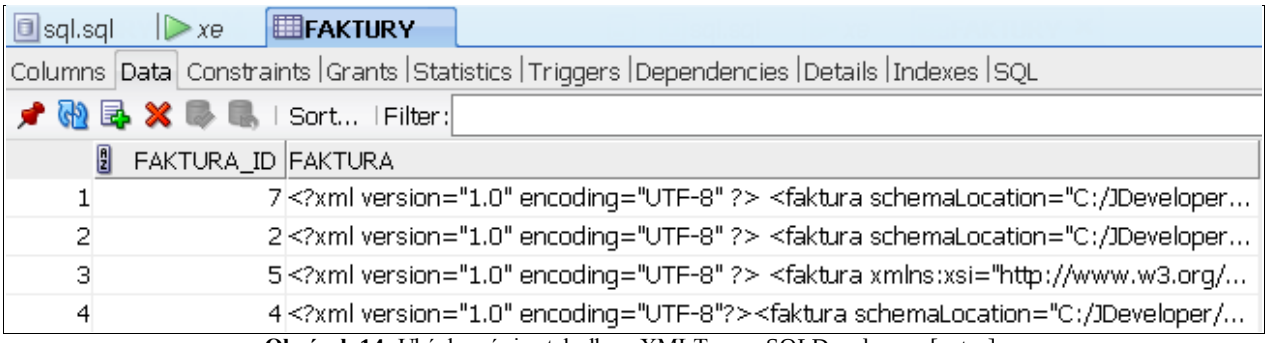
Pomocí funkce *getStringVal()* dotaz vrátí XML dokument jako řetězec. Tento řetězec se poté převede v Javě do InputStreamu a ten se již zpracuje pomocí JAXB Unmarshalleru do javovských objektů.

```
public InputStream nactiXMLzDB(int idFaktury) throws SQLException {
   String poxml="";
   DBAcces spoj= new DBAcces("java","root");
   String select="SELECT e.faktura.getStringVal() FROM faktury e
   WHERE e.faktura.extract('//faktura/cislo/text()').getNumberVal() =";
   select+= idFaktury;
   OracleResultSet orset = spoj.execXMLQuery(select);
   while(orset.next())
   {
      poxml = orset.getString(1);
   }
   InputStream is = new ByteArrayInputStream(poxml.getBytes());
   return is;
}
```

Zdrojový kód 32: Metoda pro získání XML jako řetězec z databáze a jeho transformace na Stream [autor]

```
SELECT count(*) FROM faktury f WHERE existsNode(value(f),'//faktura[cislo=20100811]') = 1;
```

Zdrojový kód 33: Příklad SQL dotazu za použití XMLType a funkce existNode() [autor]



Obrázek 14: Ukázka výpisu tabulky s XMLType v SQLDeveloperu [autor]

6.2.13. Transformace do formátu PDF

Transformace do formátu PDF se provádí především proto, aby bylo možné data lépe tisknout. Transformace probíhá tak, že se XML převede do FO a ty jsou posléze převedeny do formátu PDF. Pomocí FO lze XML transformovat i do jiných formátů. Za zmínku stojí například formát RTF. Pro transformaci XML do různých formátů je nutné mít knihovnu FOP, která tuto funkcionalitu poskytuje. Tuto knihovnu je nutné najít na Internetu a a stáhnout. Poté se musí přidat k projektu jako externí knihovna. Nejdůležitější jsou pak třídy *FopFactory* a *Fop*.

```
xslfoTransformer = getTransformer(transformSource);
xslfoTransformer.setOutputProperty(OutputKeys.ENCODING,"UTF-8");
Fop fop;
try{
   fopFactory.setUserConfig(new File("C:/Jdeveloper/mywork/EvNaVo/EvNaVo/fonts/conf/fop.xconf"));
   fopFactory.setFontBaseURL("file:///C:/JDeveloper/mywork/EvNaVo/EvNaVo/fonts/");
   fop = fopFactory.newFop(MimeConstants.MIME_PDF, foUserAgent, outStream);
   Result res = new SAXResult(fop.getDefaultHandler());
   try{
      xslfoTransformer.transform(source, res);
      File pdffile = new File("C:/Jdeveloper/mywork/EvNaVo/EvNaVo/Result.pdf");
      OutputStream out = new java.io.FileOutputStream(pdffile);
      out = new java.io.BufferedOutputStream(out);
      FileOutputStream str = new FileOutputStream(pdffile);
      str.write(outStream.toByteArray());
      str.close();
      out.close();
   }catch (Exception e) {
   }
```

Zdrojový kód 34: Ukázka generování do PDF [autor]

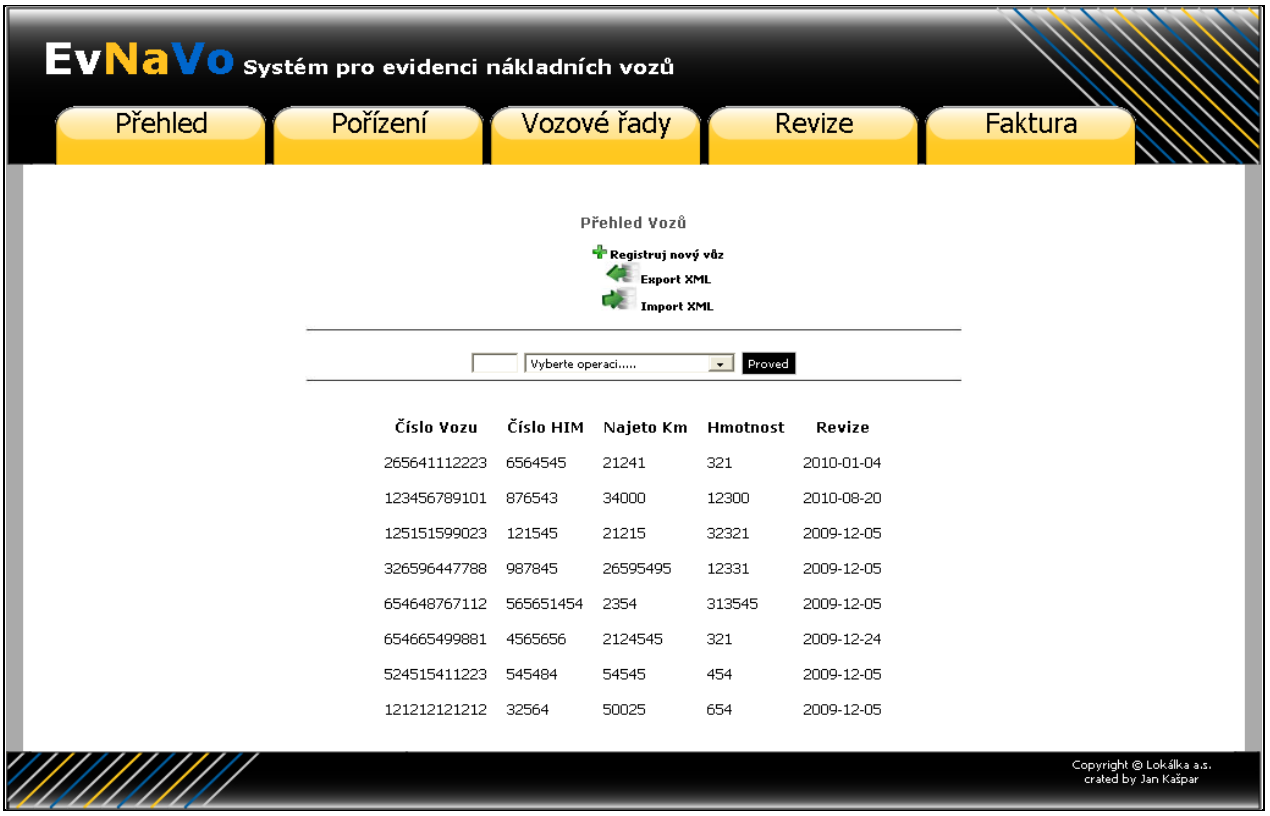
Velký problém nastává, pokud chceme použít české znaky. Standardní fonty, které jsou dostupné, je nepodporují. Proto je nutné doplnit do konfigurace fonty s českou diakritikou. Fonty s českou diakritikou je nutné zkopírovat do patřičného adresáře. Je také nutné zkontrolovat i konfigurační soubor těchto fontů *fop.xconf*. Tento soubor se pak následně v aplikaci použije pro nastavení fopFactory, aby tato třída měla k dispozici českou verzi fontů. Příklad použití je v předchozí části *Zdrojový kód 33*.

```
<!DOCTYPE fop [
<!ENTITY fop.home "file:///C:/JDeveloper/mywork/EvNaVo/EvNaVo/fonts/">
<!ENTITY fonts.dir "file:///c:/windows/fonts">
]>
...
...
<font metrics-url="&fop.home;conf/times.xml" kerning="yes" embed-url="&fonts.dir;/times.ttf">
   <font-triplet name="TimesNewRoman" style="normal" weight="normal"/>
   <font-triplet name="serif" style="normal" weight="normal"/>
</font>
<font metrics-url="&fop.home;conf/timesi.xml" kerning="yes" embed-url="&fonts.dir;/timesi.ttf">
   <font-triplet name="TimesNewRoman" style="italic" weight="normal"/>
   <font-triplet name="serif" style="italic" weight="normal"/>
</font>
```

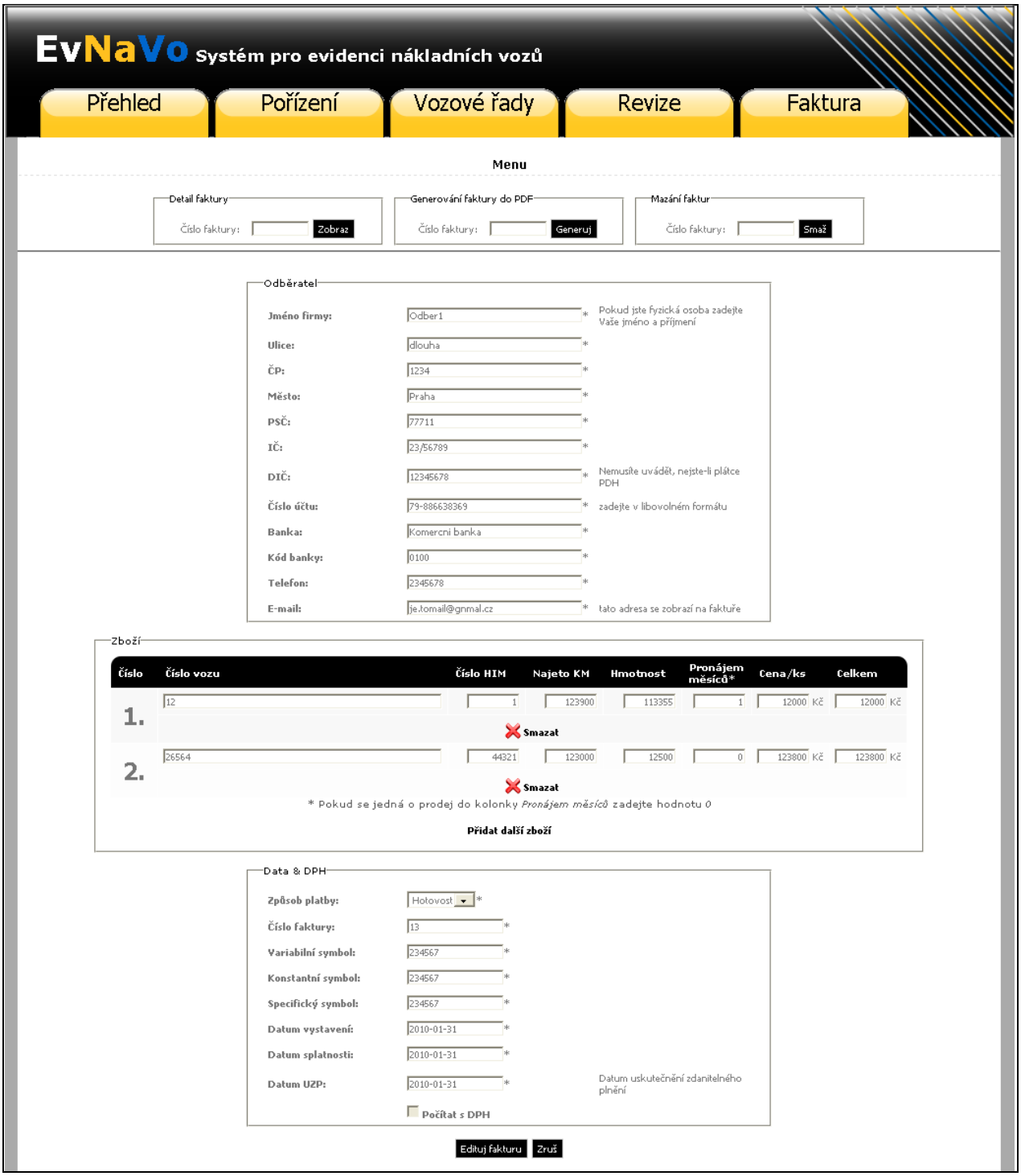
Zdrojový kód 35: Ukázka konfiguračního souboru balíčku FOP pro zpřístupnění českých znaků [autor]

6.3. Příklady formulářů

Obrázek 15: Ukázka formuláře pro přidání nákladního vozu [autor]



Obrázek 16: Ukázka výpisu nákladních vozů z databáze do aplikace [autor]



Obrázek 17: Ukázka formuláře pro vytvoření faktury[autor]

Obrázek 18: Ukázka formuláře pro přidání vozové řady[autor]

Číslo faktury: 14

Adresa odběratele: Lokálka a.s. Bezručova 3517 532 10 Pardubice 2 ICO: 23/56789 DIC: 12345678	Adresa odběratele: Odber1 dlouha 1234 77711 Praha ICO: 23/56789
Platební podmínky: Způsob platby: Hotovost Banka: Komercni banka - 0100 Číslo účtu: 79-886638369 Variabilní symbol: 234567	Datum vystavení: 2010-01-31 Datum zdanitelného plnění: 2010-01-31
	Splatnost: 2010-01-31

Fakturované položky

No.	Datum	Délka pronájmu	Cena
Číslo HIM: 1 Číslo Vozu: 12 Hmotnost: 113355 Najeto Km: 123900	2010-01-31	1	12000
Číslo HIM: 6564545 Číslo Vozu: 265641112223 Hmotnost: 321 Najeto Km: 21241	2010-01-31	0	120000
		Celkem cena:	**132000 Kč**

Obrázek 19: Ukázka výstupu faktury ve formátu PDF

7. Závěr

Tato práce měla za cíl seznámit čtenáře s problematikou zpracování XML v Javě a uvést technologie pro zpracování XML. V teoretické části byly podrobně rozebrány jednotlivé technologie, které jsou v této práci použity. Teoreticky jsou zde také rozebrána mapování XML do databáze, avšak popis této problematiky by vystačil na celou diplomovou práci. Z toho důvodu je tato problematika zmíněna jen okrajově.

Na praktickém příkladu měla tato práce ukázat technologie pro práci s XML, které jsou v současnosti k dispozici. Praktická ukázka byla postavena na již vytvořené aplikaci, která byla v průběhu navazujícího studia zadána jako ročníkový projekt. Tato aplikace byla upravena a byly do ní přidány další části jako její rozšíření. A tak byla přepracována i její analýza a návrh, který byl doplněn o přidané části. Existující části, na kterých bylo vhodné použít XML, byly přepracovány tak, aby využívaly technologie pracující s XML. Vzhledem k tomu, že by komprese XML v databázi znemožnila použití XMLType, nebyla nakonec implementována.

Aplikace by se dala rozšířit o uživatelskou část, kde by bylo naimplementované rozhraní pro zákazníky firmy. Zákazníci by tak měli možnost správy svých účtů. Dále by bylo možné rozšířit část Importu a Exportu dat, který je v současné době realizován aplikačně, ale jen na malé části dat.

Implementace aplikace by šla realizovat jako implementace desktopové aplikace s přístupem na databázový server. Toto řešení by přineslo výhody z hlediska úspor. Nebyl by nutný aplikační server a využíval by se výpočetní výkon pracovních stanic.

V průběhu tvorby této práce jsem získal velké množství znalostí a i praktických zkušeností s XML a technologiemi, které XML využívají. Tyto znalosti budou jistě přínosem v mém budoucím profesním životě.

Seznam použité literatury

[1] POKORNÝ, Jaroslav, et al. *XML technologie : Principy a aplikce v praxi*. [s.l.] : [s.n.], 2008. 272 s.

[2] HEROUT, Pavel. *Java a XML*. [s.l.] : [s.n.], 2007. 316 s.

[3] BŘÍZA, Petr *Interval.cz* : XML. In *Kompletní průvodce XSLT : Úvod do problematiky* [online]. [s.l.] : [s.n.], 27.4.2004 [cit. 2010-08-16]. Dostupné z WWW: <http://interval.cz/clanky/kompletni-pruvodce-xslt-uvod-do-problematiky/>.

[4] *Oracle.com* [online]. 2008 [cit. 2010-07-16]. Using JAXB Data Binding. Dostupné z WWW:<http://download.oracle.com/docs/cd/E12840_01/wls/docs103/webserv/data_types.html>.

[5] KOSEK, Jiří. *Kosek.cz* [online]. 2004 [cit. 2010-07-06]. Kapitola 3. XML schémata. Dostupné z WWW: <http://www.kosek.cz/xml/schema/wxs.html>.

[6] *Oracle.com* [online]. 2002 [cit. 2010-08-02]. Using XMLType. Dostupné z WWW: <http://download.oracle.com/docs/cd/B10501_01/appdev.920/a96620/xdb04cre.htm>.

[7] Extensible Markup Language. In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2001-09-27, last modified on 2010-07-12 [cit. 2010-08-18]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Extensible_Markup_Language>.

[8] XML Schema (W3C). In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2003-02-20, last modified on 2008-10-23 [cit. 2010-08-18]. Dostupné z WWW: <http://en.wikipedia.org/wiki/XML_Schema_(W3C)>.

[9] *XML.com* [online]. 2003 [cit. 2010-07-18]. An Introduction to StAX. Dostupné z WWW: <http://www.xml.com/pub/a/2003/09/17/stax.html>.

[10] Java (programming language). In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2001-11-13, last modified on 2010-08-13 [cit. 2010-08-18]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Java_(programming_language)>.

[11] *IBM.com* [online]. 2001 [cit. 2010-08-18]. Simplify XML programming with JDOM. Dostupné z WWW: <http://www.ibm.com/developerworks/java/library/j-jdom/>.

[12] *Linuxzone.cz* [online]. 2002 [cit. 2010-08-05]. Programově na XML, 3.díl - JDOM. Dostupné z WWW: <http://www.linuxzone.cz/index.phtml?ids=2&idc=377>.

[13] *Tvorba-webu.cz* [online]. 2008 [cit. 2010-07-28]. XML základy. Dostupné z WWW: <http://www.tvorba-webu.cz/xml/>.

[14] *W3schools.com* [online]. 2001 [cit. 2010-07-20]. XSLT Element. Dostupné z WWW: <http://www.w3schools.com/xsl/xsl_if.asp>.

[15] HUNTER, Jason. *JDOM AND XML PARSING* [online]. [s.l.] : [s.n.], 2002 [cit. 2010-08-20]. JDOM and XML Parsing, Part 1, s. . Dostupné z WWW: <http://www.jdom.org/docs/oracle/jdom-part1.pdf>.

[16] *Jaxb.dev.java.net* [online]. 2005 [cit. 2010-08-20]. Elements With Any Type. Dostupné z WWW: <https://jaxb.dev.java.net/tutorial/section_2_2_16_1-Elements-With-Any-Type.html>.

[17] KOSEK, Jiří. *Kosek.cz* [online]. 2004 [cit. 2010-06-20]. Kapitola 1. Úvod. Dostupné z WWW: <http://www.kosek.cz/xml/xslt/uvod.html#principy>.

Seznam obrázků

Seznam zkratek

- **XML** – *eXtensible Markup Language*. Rozšiřitelný značkovací jazyk pro popis dat.
- **RTF** – *Rich Text Format.* Na platformě nezávislý textový formát.
- **PDF** – *Portable Document Format*. Přenositelný formát dokumentů.
- **HTML** – HyperText Markup Language. Značkovací jazyk pro hypertext.
- **XSL** – *eXtensible Stylesheet Language*. Stylový jazyk.
- **Xpath** – *XML Path Language*. Jazyk pro adresování částí XML.
- **Xlink** – *XML Linking Language*. Jazyk pro tvorbu vazeb mezi XML.
- **XHTML** – *eXtensible HyperText Language*. Rozšiřitelný jazyk pro hypertext.
- **DTD** – *Document Type Definition*. Jazyk pro popis struktury XML.
- **XSD** – *XML Schema Definition*. Jazyk pro popis struktury XML.
- **JAXB** – *Java Architecture for XML Binding*.
- **URI** - *Uniform Resource Identifier*. Unikátní identifikátor zdroje.
- **DF** – *Depth First*.
- **EER** – *Extended Entity Relationship*.
- **ER** –*Entity Relationship*.
- **XSLT** - *eXtensible Stylesheet Language Transformations*. Transformační jazyk.
- **XSL-FO** - *eXtensible Stylesheet Language – Formating Objects.* Jazyk pro formátování XML ve vysoké typografické kavlitě.
- **JVM** – *Java Virtual Machine*.
- **W3C** - *World Wide Web Consortium*. Společnost vyvíjející webové standardy.
- **DOM** – *Document Object Model*.
- **JDOM** - *Document Object Model for Java*.
- **StAX** – *Streaming API for XML*.
- **SAX** – *Simple API for XML*.
- **NV** – Nákladní vůz.