VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

Fakulta informatiky a statistiky

Katedra informačního a znalostního inženýrství

Vizualizace XML schémat

Bakalářská práce

Václav Slavětínský

Vedoucí práce: Ing. Jiří Kosek

květen 2008

Poděkování

Rád bych poděkoval Ing. Jiřímu Koskovi za odborné vedení, cenné rady a hodnotné podněty, jež mi poskytl při psaní této práce.

Prohlášení

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

V Praze dne 1. května 2008

Václav Slavětínský

Abstrakt

Bakalářská práce se zabývá návrhem a implementací aplikace, která bude sloužit k převodu *W3C XML schématu* do interaktivního diagramu ve formátu *SVG (Scalable Vector Graphics)*. Diagram bude možné použít jako součást dokumentace XML souborů vyhovujících schématu; bude znázorňovat strukturu dokumentů a údaje o jednotlivých položkách – elementech, atributech. Cílem je generovat interaktivní a přehlednou grafiku, abychom uživatelům usnadnili orientaci a pochopení XML dokumentů.

První část práce obsahuje stručný popis technologií a způsobu jejich použití v rámci aplikace. Je zde zařazeno i zhodnocení několika open source implementací abstraktního datového modelu XML schématu. Z nich je vybrána jedna – *Xerces2-J*. Xerces poskytne jednoduchý přístup ke kompontám schématu a jejich vlastnostem a vytvoří tak programový základ. Další kapitoly se zabývají návrhem. Nejprve jde o návrh grafického modelu včetně zajištění jeho interaktivity. Naleznete zde obrázky grafických symbolů, které znázorňují jednotlivé komponenty XML schématu, a popis jejich parametrů. Dodatečně je možné měnit vzhled, především barvy, pomocí externího stylu. Následně je uveden algoritmus, jehož úkolem je projít komponentami a vytěžit z nich data pro vizualizaci.

Praktickým výsledkem bakalářské práce je program napsaný v jazyce *Java*, který provádí zde popsanou transformaci XML schématu do jeho grafické podoby.

Abstract

This thesis deals with design and implementation of an application, that will serve to transform *W3C XML Schema* into an interactive diagram in SVG format (*Scalable Vector Graphics*). The diagram can be used as a part of documentation for schema conforming XML files; it will represent structure of XML files and it will provide information about individual items – elements and attributes. The goal is to generate interactive and well arranged graphics, so that we simplify the orientation and understanding of XML documents for users.

The first part of this work contains a brief description of technologies and their usage within the application. A review of some open-source XML Schema abstract data model implementations is also included. One of them – *Xerces2-J* – is chosen to provide simple access to schema components and their properties. By doing so, it will create the base for the program. Next chapters deal with the design. At first they are concerned to design the graphical model including its interactivity. You can find here pictures of graphical symbols, that represents the individual XML Schema components, and a description of their parameters. Additionally, it is possible to change the appearance of the symbols via an extern style, especially colors. After that, an algorithm is presented, that should walk through the components and extract data for the visualization.

The practical result of this thesis is a program, written in language *Java*. It performs the transformation of XML Schema into its graphical form that is described here.

Obsah

Kapitola 1

Úvod

Jazyk XML umožňuje vývojářům, při dodržení syntaktických pravidel, vytvářet vlastní formáty pro uchování a sdílení dat. Pokud chceme tyto formáty automatizovaně zpracovávat a validovat, potřebujeme jejich formální deklaraci a dokumentaci, tedy popis který umožní jejich sdílení, definování a použití. Pro tyto účely vzniklo několik jazyků. Patří k nim především W3C *XML schéma*, DTD (*Document Type Definition*) a Oasis *Relax NG*.

Díky začlenění do dalších standardů pracujících s XML, díky širokým možnostem a podpoře velkých softwarových společností je velmi často využíváno XML schéma. Jeho specifikace (viz [2], [3]) je však poměrně složitá a schémata se mohou stát pro uživatele nepřehledná. Důvodem jsou různé možnosti zápisu, vyplývající z objektových prvků tohoto jazyka. V XML schématu je umožněno používání datových typů, jejich rozšiřování a omezování; vytváření referencí na elementy, atributy a skupiny elementů či atributů; elementy lze vzájemně nahrazovat mechanismem substitučních skupin; dále je možné importovat, vkládat a předefinovat externí schémata, respektive jejich části.

Téma této práce je vizualizace XML Schémat. Cílem je navrhnout a implementovat aplikaci, která by na vstupu načetla instanci W3C XML schématu a jako výstup by uložila její interaktivní grafickou reprezentaci ve formátu SVG (*Scalable Vector Graphics*) [5]. Tuto grafickou reprezentaci či diagram bude možné použít jako součást dokumentace XML souborů vyhovujících schématu. Diagram bude zobrazovat struktury použitelných elementů a atributů, jejich jména, jmenné prostory, datové typy a dodatečné informace – kardinalitu elementů, povinné/volitelné použití atributů, definované unikátní, primární a cizí klíče atd. Umožněno bude také procházení struktur, rozevírání a skrývání podskupin elementů a atributů, s využitím *ECMAScriptu*. K důležitým požadavkům patří, aby byla grafika interaktivní, intuitivní a přehledná, neboť smysl práce spočívá především v usnadnění orientace a pochopení XML schémat ze strany jejich uživatelů – tvůrců XML dokumentů.

SVG soubory lze otevřít v moderních prohlížečích webových stránek (u některých je třeba nainstalovat SVG plug-in) nebo v aplikacích určených k prohlížení či editaci vektorové grafiky. Mnohé z nich jsou volně dostupné. Vektorová grafika používá k reprezentaci obrázků geometrické objekty a díky tomu ji lze snadno programově upravovat. Jisté omezení však vyplývá z náročnosti zpracování a pomalé implementci ECMAScriptu v prohlížečích. To začne být patrné u velkých schémat, nebo spíše velkých struktur popisovaných dokumentů (i poměrně krátkým schématem lze definovat rozsáhlé struktury). Omezení se projevuje v pomalejší reakci na události a obecně v pomalejším vykreslování diagramu. V této oblasti lze jen doufat, že do budoucna vznikne lepší podpora.

Samotná aplikace bude napsána v programovacím jazyce Java a bude využívat balíčky parseru *Xerces*, který je open source implementací rozhraní definovaného v [2] a [3]. Pro spuštění bude třeba mít nainstalované běhové prostředí (*Java Runtime Envioronment*) příslušné verze. Dnes je však JRE běžnou součástí většiny počítačů. Díky využití Javy bude aplikace snadno přenositelná na různé platformy.

Nyní se stručně zmíním o struktuře práce. Cílem druhé kapitoly je seznámit čtenáře s technologiemi použitými v rámci aplikace a se způsobem jejich použití. Jde o W3C XML schéma, Scalable Vector Graphics, ECMAScript a Document Object Model, Cascading Style Sheets, jazyk Java.

Zvláštní kapitola (třetí) je věnována možnostem načtení a interpretace modelu XML schématu. Je tu popis několika open source implementací abstraktního datového modelu XML schématu a jejich zhodnocení. Z těchto implementací jsem nakonec vybral procesor Xerces2-J. Vytvoří základ programu tím, že poskytne jednoduchý přístup ke kompontám schématu a jejich vlastnostem.

Čtvrtá kapitola se zabývá návrhem grafického modelu. Nejprve je vybrán vhodný typ, logický model; ten přehledně zobrazí strukturu definovaných XML dokumentů. Následuje rozbor jednotlivých prvků – grafických symbolů kompenent schématu – a jejich struktury. U každého symbolu naleznete obrázky, význam a popis parametrů ve vztahu k vlastnostem komponent schématu. Dále je popsán mechanismus zajištění interaktivity modelu a možnost dodatečných úprav vzhledu pomocí externího stylu.

Cílem páté kapitoly je navrhnout algoritmus zpracování schématu. Je třeba projít některými komponentami a vytěžit z jejich vlastností data pro vizualizaci. Zpracování komponent, jejichž význam je zde také popsán, obstarávají jednotlivé metody. Ty se navzájem volají, předávají si parametry a vytváří grafické symboly pro pozdější uložení do SVG souboru.

V závěru se pokusím o shrnutí výsledků práce a uvedu, kde je možné aplikaci stáhnout.

Kapitola 2

Použité technologie

2.1. XML schéma

XML, *Extensible Markup Language*, rozšiřitelný značkovací jazyk umožňuje vývojářům vytvářet vlastní formáty pro uchování a sdílení dat. Vznikají tím jakési nové slovníky položek – elementů a jejich vlastností – atributů. Pokud chceme tyto formáty automatizovaně zpracovávat a validovat, potřebujeme jejich formální deklaraci a dokumentaci, tedy popis který umožní sdílení, definování a použití vzniklých slovníků.

XML schéma[1] je specifikace vydaná konsorciem W3C.[2] Definuje jazyk, využívající objektově orientovaný přístup, který lze použít pro formální popis XML dokumentů; poskytuje prostředky pro definování jejich struktury, obsahu a sémantiky. Konkrétně můžeme pomocí schématu definovat elementy a atributy použitelné v dokumentu, včetně jejich struktury – vzájemného zanořování a kombinování; můžeme také určit datový typ pro obsah elementu nebo atributu, implicitní hodnoty a další integritní omezení. K těm patří především unikátní klíče (*unique*), primární klíče (*key*) a na ně odkazující cizí klíče (*keyref*).

XML schéma bývá často srovnáváno s jiným obvyklým schémovým jazykem – DTD, *Document Type Definition*. Ten byl zahrnut již v XML 1.0 specifikaci W3C [6]. Mezi nejdůležitější rozdíly patří to, že DTD využívá vlastní syntaxe, zatímco XML schéma je aplikací XML, to znamená, že využíva jeho syntaxi. Dalším důvodem pro volbu XML schématu je možnost přiřazení datových typů atributům a obsahu elementů. Tuto základní vlastnost DTD nemělo, takže se všemi hodnotami zacházelo jako s textovými řetězci. Stejně tak nebylo možné pracovat se jmennými prostory. Přesto jsou DTD stále velmi užívané, především kvůli jednoduchosti.

Specifikace XML schématu a jeho XML zápis jsou naopak komplexní a v některých místech poměrně obtížné na pochopení a použití. Vznikly proto i další jazyky: *Relax NG* s běžnou (XML) a kompaktní syntaxí a *Schematron* s odlišným přístupem – pro validaci podle pravidel. XML schéma má však nejširší využití, také proto, že ho delší dobu podporují velké softwarové firmy (Microsoft, IBM, Oracle, Sun).

XML schéma formalizuje omezení, vyjádřená jako pravidla nebo model struktury, kterým odpovídá určitá třída XML dokumentů. Schémata slouží často jako nástroje designu, zakládající framework, na němž mohou být postaveny konkrétní implementace. Existuje několik typických oblastí použití schémat. Jedná se především o validaci, dokumentaci, dotazování, data binding a editaci [4]:

[1] Druhé vydání je z 28. října 2004, první bylo schváleno 2. května 2001.
W3C: *XML Schema*. http://www.w3.org/XML/Schema

[2] World Wide Web Consortium, oficiální webové stránky. http://www.w3.org/

- *Validací* se rozumí ověření shody XML dokumentu se schématem. Často se využívá pro kontrolu XML vstupu do aplikace. Funguje jako filtr; dokumenty vyhovující schématu projdou, ostatní nebudou použity. Tato externí kontrola značně zjednodušuje aplikace, které by jinak musely ošetřovat spoustu výjimek vzniklých při zpracování nevyhovujících XML vstupů.

- *Dokumentace* znamená v tomto případě popis vyhovujících XML dokumentů. Je formální a strojově čitelná. Jelikož je XML schéma XML dokumentem, lze formální dokumentaci snadno převést tak, aby byla snadno čitelná i pro člověka, například použitím XSLT transformací, nebo jiných transformací. *Také aplikace, jež je výsledkem této bakalářské práce provádí takový převod, na podobu obrazovou.*

- Schémata poskytují podporu pro *dotazování*, vyhledávání obsahu v XML dokumentech. To je sice možné provádět i bez nich, ale dodatečná informace o struktuře a datových typech pomůže zrychlit a zefektivnit jak vyhledávání, tak i řazení a porovnávání. Využití nalezneme v jazycích *XPath 2.0*, *XSLT 2.0*, *XQuery 1.0*.

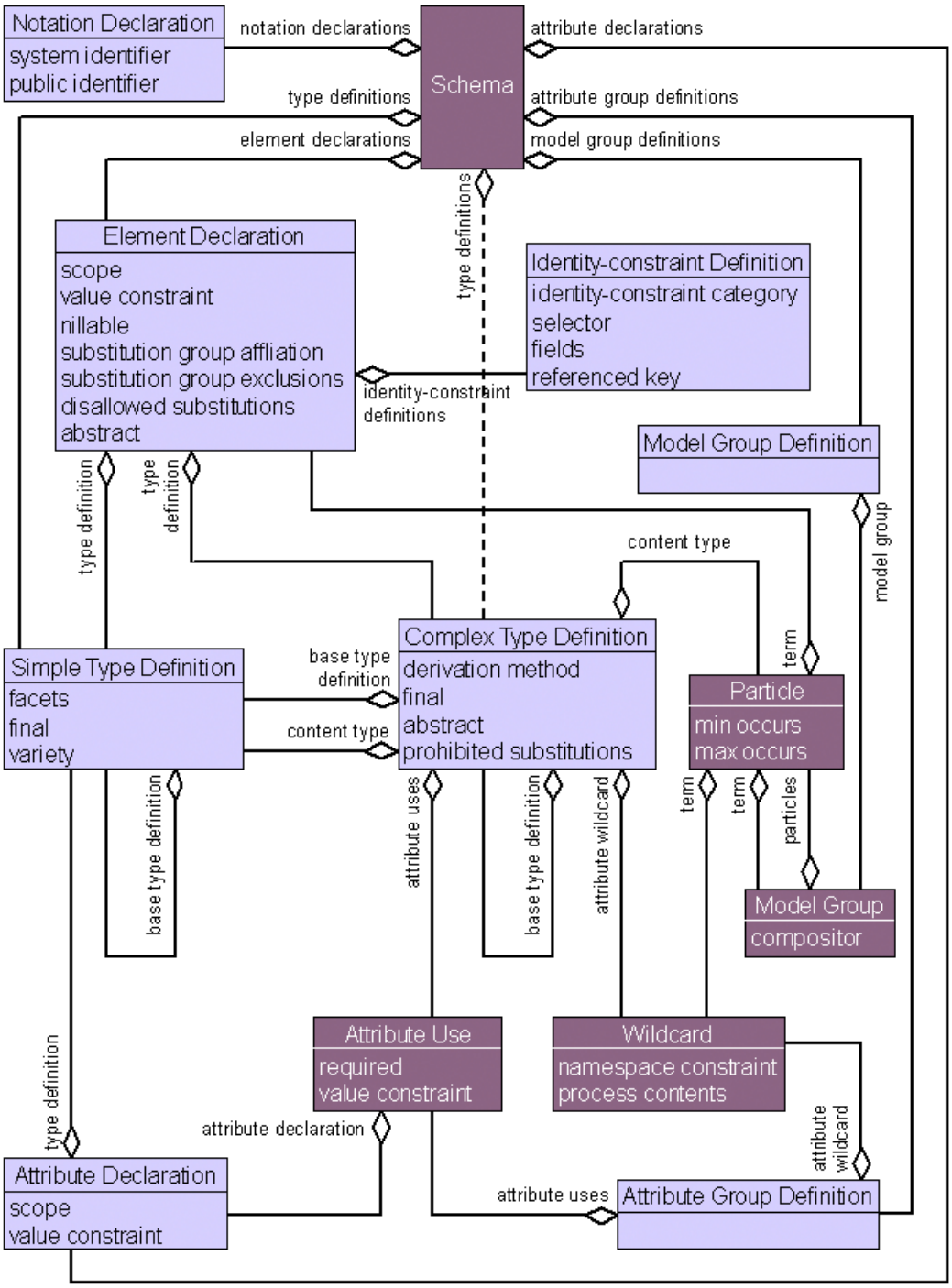
- XML schémata jsou od svého počátku využívána pro *data binding*. Tím se zde rozumí převádění dat obsažených v XML dokumentech do struktur konkrétních aplikací, jakými jsou objekty v objektově orientovaných systémech a relační tabulky relačních databázových systémů. Schémata poskytují dodatečné informace o datových typech, které se převedou do datových typů té které aplikace.

- Schémata obecně je možné použít při *editaci* XML dokumentů. Například tato práce vzniká v *DocBooku*, což je také aplikace XML. Editor, který používám, mi dáva na výběr pouze elementy, které jsou podle schématu na daném místě přípustné. To výrazně usnadňuje práci. Editor využívá DTD DocBooku, poskytující informaci o struktuře, XML schémata nadto poskytnou informaci o datových typech.

Komponenty schématu

Procesory XML schémat, vyhovující specifikaci W3C [2], musí pracovat s informacemi obsaženými ve schématu tak, jak je to ve specifikaci popsáno. Na XML schéma se zde nahlíží jako na abstraktní datový model konceptuální úrovně, nezávislý na implementaci. Datový model se skládá z jednotlivých, přesně definovaných součástí – *komponent*. Každá z nich zahrnuje popis XML reprezentace, vlastností a omezení, popis validačních pravidel, příspěvků k PSVI (*Post Schema Validation Infoset*),[3] a případných vestavěných instancí těchto komponent.
Definice zní takto: „*Komponenta schématu* je obecný termín, který zastupuje stavební prvky tvořící abstraktní datový model schématu… *XML schéma* je množinou komponent schématu.“[4]
Existuje 13 druhů komponent, spadajících do 3 skupin. *Aplikace s nimi bude pracovat způsobem popsaným v kapitole 5 – „Zpracování XML schématu“*. Pro rychlé pochopení vazeb mezi komponentami slouží obrázek 2.1 – „Diagram komponent XML schématu“.[5]

[3] Jedná se o XML infoset (množinu informací), rozšířený o dodatečné informace o typu jednotlivých položek – elementů, atributů, obecně uzlů.
[4] [2] XML Schema Abstract Data Model. http://www.w3.org/TR/xmlschema-1/#concepts-data-model
[5] Tamtéž.



Obrázek 2.1. Diagram komponent XML schématu

Primární komponenty mohou nebo musí být pojmenované:

- definice jednoduchých typů (*Simple type definitions*),
- definice komplexních typů (*Complex type definitions*),
- deklarace atributů (*Attribute declarations*),
- deklarace elementů (*Element declarations*).

Sekundární komponenty musí mít jména:

- definice skupin atributů (*Attribute group definitions*),
- definice identitních omezení (*Identity-constraint definitions*),
- definice modelových skupin (*Model group definitions*),
- deklarace notací (*Notation declarations*).

Pomocné komponenty poskytují přístup k jiným komponentám. Jsou závislé na kontextu:

- anotace (*Annotations*),
- modelové skupiny (*Model groups*),
- částice (*Particles*),
- divoké karty (*Wildcards*),
- užití atributů (*Attribute Uses*).

2.2. SVG

Scalable Vector Graphics, škálovatelná vektorová grafika vznikla také na půdě konsorcia W3C. Základ vývoje tvoří specifikace SVG 1.1 [5]. Kolem ní však existuje a vzniká řada dalších: *SVG Tiny 1.2*, *SVG Mobile 1.1*, *SVG Print*, *SVG Filters*, *SVG Requirements*. Jejich předmětem je jazyk pro popis dvourozměrné grafiky a grafických aplikací pomocí XML. Jedná se přitom o vektorovou grafiku, která používá k reprezentaci obrázků geometrické objekty. To přináší možnost škálování – zmenšování a zvětšování – bez ztráty kvality; objekty, z nichž se obraz skládá, jsou odlišeny a vzniklé soubory zabírají obvykle méně místa než soubory bitmapové. Je samozřejmé, že se vektorová grafika hodí pouze na některé druhy obrázků, jako jsou třeba symboly, diagramy nebo grafy. *Výstup aplikace, jež je předmětem této práce, bude diagram XML schématu v SVG.*

Formát SVG byl navržen hlavně pro použití na webu, díky tomu je ho dnes možné přímo otevřít v prohlížečích Mozilla Firefox, Opera; v MS Internet Explorer bohužel až po nainstalování příslušného zásuvného modulu (Adobe SVG Viewer). Stejně tak je formát podporován i v komerčních (Adobe Illustrator, CorelDraw) či open source (Inkscape, Sodipodi) editorech.

SVG poskytuje tři typy grafických objektů. Jsou to tvary vektorové grafiky, například cesty složené z úseček a křivek, obrázky a text. Tyto objekty mohou být seskupovány, lze je transformovat, předrenderovat a přiřadit jim styly. K dalším funkcím patří vnořené transformace, ořezávání objektů podle cest, alpha masking, filtrování obrazu a objekty sloužící jako šablony.

Výsledné obrázky mohou být interaktivní a dynamické. Animace lze definovat a spouštět deklarativně pomocí speciálních SVG elementů nebo skriptováním. Skriptovacím jazykům jsou zpřístupněny všechny elementy, atributy a vlastnosti přes SVG *Document Object Model* (DOM). K libovolným grafickým objektům lze přiřadit různé ovladače událostí, jako onmouseover, onclick.[6]

Dále je možné SVG vkládat do jiných XML formátů, k tomu se využívá technologie jmenných prostorů (*namespace*). Zajištěna je i kompatibilita v tom smyslu, že skriptování lze provádět zároveň například na XHTML (*Extensible HyperText Markup Language*) webové stránce, do které byly vloženy elementy SVG. Podobně snadné je využití kaskádových stylů (*Cascading Style Sheets*, CSS) pro popis vzhledu jednotlivých objektů.

[6] Při najetí myši, při kliknutí myši.

Elementy použité ve výstupu aplikace

Následuje stručná charakteristika SVG elementů, které využívá aplikace pro grafickou prezentaci schématu. U elementů jsou dále uvedeny i některé důležité atributy. Vedle nich jsou ještě potřeba běžné atributy: *id* pro jednoznačnou identifikaci elementu v rámci dokumentu a *class* pro zařazení elementu do určité skupiny, které pak lze například přiřadit styl. Ukázku konkrétního použití naleznete v příloze C.2 – „SVG výstup (XML reprezentace)“.

2.2.1. `<svg>`[7]

Fragment SVG dokumentu se skládá z libovolného počtu SVG elementů obalených tagem `<svg>`. V našem případě nepůjde o fragment, ale o samostatný dokument, jehož kořenovým elementem bude `<svg>`. Krom obvyklých atributů (*id*, *class*) zde budou:

- *xmlns* – deklarace jmenných prostorů,
- *onload* – spustí skript při nahrání tohoto elementu.

2.2.2. `<title>`[8]

Titulek, krátký popisek může být přidán každému kontejneru nebo grafickému elementu. Nebude přímo renderován, může být ale zobrazen jako *tooltip*. Titulek přidaný `<svg>` elementu zobrazí prohlížeč v záhlaví okna.

2.2.3. `<script>`[9]

Slouží pro vložení skriptu, bude zařazen jako dítě `<svg>` elementu. Samotný skript je třeba obalit sekcí `<![CDATA[` *skript* `]]>`, aby interpret chápal obsah pouze jako text a nehledal v něm značky.

- Atribut *type* – určuje skriptovací jazyk, hodnotou je MIME typ, v případě ECMAScriptu `text/ecmascript`.

2.2.4. `<defs>`[10]

Obsahuje elementy, na které je v dokumentu odkazováno. Obsah může být stejný jako u kontejneru `<g>`, rozdíl spočívá v tom, že grafické objekty zde uvedené nebudou přímo renderovány. Do definic ve výstupu aplikace zahrneme styl `<style>` a často využívané symboly `<symbol>`.

2.2.5. `<style>`[11]

Slouží pro vložení stylu, popisujícího grafickou úpravu objektů a jejich skupin. Opět bude vhodné styl obalit sekcí `<![CDATA[` *styl* `]]>`.

[7] [5] http://www.w3.org/TR/SVG11/struct.html#NewDocument
[8] [5] http://www.w3.org/TR/SVG11/struct.html#DescriptionAndTitleElements
[9] [5] http://www.w3.org/TR/SVG11/script.html#ScriptElement
[10] [5] http://www.w3.org/TR/SVG11/struct.html#DefsElement
[11] [5] http://www.w3.org/TR/SVG11/styling.html#StyleElement

- Atribut *type* – určuje jazyk stylu, hodnotou je MIME typ,[12] v případě kaskádového stylu `text/css`.

2.2.6. `<symbol>`[13]

Symbol definuje grafický vzor, který se nerenderuje, dokud není použit v SVG dokumentu elementem `<use>`. Definovat symboly má smysl u často používaných objektů. V této aplikaci to budou zatím dva – *plus* a *minus* pro rozevírání a skrývání podstromu grafických symbolů (boxů). Využijeme pouze běžné atributy.

2.2.7. `<use>`[14]

Odkazuje na jiný element a signalizuje, že jeho grafický obsah má být zahrnut a vykreslen v místě, kde je element `<use>` uveden. Může se jednat o grafické elementy, `<g>`, `<svg>`, `<use>` a (v našem případě pouze) `<symbol>`. Kromě obvyklých atributů zde využijeme:

- *x* – určuje souřadnici na ose X, kde bude umístěn odkazovaný objekt,
- *y* – určuje souřadnici na ose Y, kde bude umístěn odkazovaný objekt,
- *xlink:href* – URI odkaz na objekt, realizováno pomocí identifikátoru,
- *onclick* – spouští skript při kliknutí na grafický objekt, tím bude symbol *plus* nebo *minus*. Skript provede rozbalení nebo skrytí podstromu grafických symbolů (boxů).

2.2.8. `<g>`[15]

Představuje kontejner pro seskupení souvisejících grafických elementů. Použijeme ho pro sdružení SVG elementů tvořících konkrétní grafický symbol komponenty schématu (box).

- Atribut *transform* – pro určení grafické transformace obsahu. Grafiku lze posouvat, škálovat, otáčet a zešikmit. My budeme pouze posouvat boxy o *x* jednotek po ose X a o *y* jednotek po ose Y pomocí *transform* = `translate`(*x*, *y*).

2.2.9. `<text>`[16]

Definuje textový grafický element. Ten je renderován stejnými metodami jako ostatní grafické elementy. Bude sloužit ke zobrazení různých názvů, vlastností a popisků v boxech. Budeme potřebovat atributy:

- *x* – reprezentuje absolutní pozici textu, počítanou od prvního znaku, na ose X. Také je možné uvést více (*n*) hodnot, které pak reprezentují pozice prvních *n* znaků.
- *y* – reprezentuje absolutní pozici textu, počítanou od prvního znaku, na ose Y. I zde můžete uvést více hodnot, oddělených mezerou nebo čárkou, s významem jako u atributu *x*.

[12] RFC specifikace 2045: *Multipurpose Internet Mail Extensions*. http://www.ietf.org/rfc/rfc2045.txt
[13] [5] http://www.w3.org/TR/SVG11/struct.html#SymbolElement
[14] [5] http://www.w3.org/TR/SVG11/struct.html#UseElement
[15] [5] http://www.w3.org/TR/SVG11/struct.html#Groups
[16] [5] http://www.w3.org/TR/SVG11/text.html#TextElement

- *visibility* – určuje, jestli bude text viditelný, skrýtý, nebo zdědí viditelnost po svém rodiči. Tento atribut bude třeba pro skrývání a odkrývání některých popisků symbolů komponent schématu.

2.2.10. `<line>`[17]

Definuje úsečku. Pomocí úseček a křivek budou propojeny jednotlivé grafické symboly (boxy), Z úseček budou složeny i další prvky. Použijeme atributy:

- *x1* – souřadnice začátku úsečky na ose X,
- *y1* – souřadnice začátku úsečky na ose Y,
- *x2* – souřadnice konce úsečky na ose X,
- *y2* – souřadnice konce úsečky na ose Y.

2.2.11. `<polyline>`[18]

Definuje množinu propojených úseček, typicky tvoří neuzavřené tvary. Element `<polyline>` použijeme pouze v některých grafických symbolech komponent schématu.

- Atribut *points* – seznam párů souřadnic podle os X a Y, začátky a konce úseček, z nichž se skládá tento útvar.

2.2.12. `<polygon>`[19]

Definuje uzavřený tvar, skládající se z množiny propojených úseček. Element `<polygon>` použijeme pouze v některých grafických symbolech komponent schématu.

- Atribut *points* – seznam párů souřadnic podle os X a Y, tvořících polygon.

2.2.13. `<rect>`[20]

Definuje obdélník. Ten poslouží k vykreslení boxů, jejich stínů a dalších grafických prvků. Pro klasifikaci použijeme atribut *class*, dále:

- *x* – souřadnice strany obdélníku podle osy X, té strany, jejíž souřadnice má nižší hodnotu v uživatelském souřadnicovém systému (většinou jde o levou stranu).
- *y* – souřadnice strany obdélníku podle osy Y, strana je určena analogicky jako u atributu *x*, většinou jde o horní stranu.
- *width* – šířka obdélníku,
- *height* – výška obdélníku,
- *rx* – poloměr elipsy, jež tvoří zaoblené rohy obdélníku, podle osy X.
- *onclick* – stejný význam jako u elementu `<use>`, atribut bude nastaven u obdélníkových ovládacích tlačítek.

[17] [5] http://www.w3.org/TR/SVG11/shapes.html#LineElement
[18] [5] http://www.w3.org/TR/SVG11/shapes.html#PolylineElement
[19] [5] http://www.w3.org/TR/SVG11/shapes.html#PolygonElement
[20] [5] http://www.w3.org/TR/SVG11/shapes.html#RectElement

2.2.14. `<circle>`[21]

Definuje kružnici, kruh. Ty ve výstupu aplikace poslouží pouze jako dodatečné grafické prvky. Použijeme atributy:

- *cx* – souřadnice středu kruhu na ose X,
- *cy* – souřadnice středu kruhu na ose Y,
- *r* – poloměr.

2.2.15. `<path>`[22]

Reprezentuje obrys tvaru. Může být vyplněn, obtažen nebo sloužit jako cesta, podle které budou ořezávány jiné objekty. V této aplikaci bude sloužit jako koncová část propojení boxů.

- Atribut *d* – definice obrysu tvaru. Obecně je tvořená příkazy *moveto*, *lineto*, *curveto*, *arc* a *closepath*, buď absolutními, nebo relativními.[23] K příkazům se dále zadávají páry souřadnic podle os X a Y.

2.3. ECMAScript

ECMAScript je skriptovací jazyk, standardizovaný organizací *Ecma International* ve specifikaci *ECMA-262*.[24] Je široce používán na webu a bývá často označován jako *JavaScript* nebo *JScript*, podle hlavních dialektů tohoto jazyka. Syntaxe se záměrně podobá syntaxi *Javy*, ale je uvolněnější, aby se docílilo snazšího použití, například není třeba deklarovat typ proměnných. Zároveň však tento přístup zvyšuje pravděpodobnost výskytu chyby.

ECMAScript je objektově orientovaný programovací jazyk pro provádění výpočtů a manipulaci s objekty *v hostitelském prostředí*. Nefunguje tedy samostatně, vstup a výstup dat a zpracovávané objekty poskytuje jiný, existující systém, jehož možnosti jsou rozšiřovány skriptováním. To je podstatou skriptovacích jazyků. *V této aplikaci bude skript sloužit k zajištění interaktivity SVG modelu schématu v prostředí prohlížeče.* Konkrétnější informace jsou v oddílu 4.4 – „Interaktivita modelu“.

ECMAScript byl původně vyvíjen pro web za účelem oživení webových stránek a přenesení části výkonu na klienta – prohlížeč – v klient-server architektuře. Pokud se jedná o HTML (*HyperText Markup Language*) stránku, poskytne prohlížeč objekty reprezentující okna, menu, dialogové boxy, textové oblasti, odkazy, rámy a další. Dále poskytne způsoby, jak spustit skriptovací kód při událostech, například nahrávání stránky, změna velikosti okna, pohyb myši, kliknutí.

My však nepotřebujeme speciální prvky jazyka HTML, ale rozhraní k SVG nebo obecně XML objektům. Obecné rozhraní je standardizováno v další specifikaci konsorcia W3C, nazývá se DOM (*Document Object Model*).[25] DOM existuje v několika úrovních – level 1 až 3. V této

[21] [5] http://www.w3.org/TR/SVG11/shapes.html#CircleElement
[22] [5] http://www.w3.org/TR/SVG11/paths.html#PathElement
[23] Přesun kurzoru na zadanou pozici bez kreslení čáry; s vykreslením úsečky; s vykreslením podrobněji stanovené křivky; s vykreslením oblouku; s vykreslením úsečky vedoucí k počátečnímu bodu tak, aby vznikl uzavřený tvar.
[24] Ecma International: *Standard ECMA-262*. ECMAScript Language Specification 3rd edition (December 1999). http://www.ecma-international.org/publications/standards/Ecma-262.htm
[25] W3C: *Document Object Model (DOM)*. http://www.w3.org/DOM/

aplikaci by stačila úroveň druhá. Pro programovou manipulaci s objekty škálovatelné grafiky vznikl speciální SVG DOM. Ten je součástí specifikace SVG[26] a dále rozšiřuje DOM Level 2. SVG DOM bychom mohli dobře využít, ale museli bychom se pak potýkat s nekompatibilitou některých prohlížečů. Nakonec proto zůstaneme u základního DOM úrovně 2, který je více rozšířen. Budeme potřebovat metody a atributy blíže popsané v [9] a [10], jejich konkrétní uplatnění je ukázáno na příkladu C.2 – „SVG výstup (XML reprezentace)“:

- Document: Element `getElementById`(in DOMString *`elementId`*); pro získání elementu podle jeho identifikátoru, který je uveden v atributu *id*.
- Document: NodeList `getElementsByTagName`(in DOMString *`tagname`*); pro získání seznamu uzlů se zadaným jménem tagu.
- Node: `readonly attribute` NodeList *childNodes*; obsahuje seznam všech dětí daného uzlu.
- Element: DOMString `getAttribute`(in DOMString *`name`*); pro získání hodnoty atributu se zadaným názvem.
- Element: `void setAttribute`(in DOMString *`name`*, in DOMString *`value`*); slouží k nastavení hodnoty atributu s daným názvem nebo k vytvoření nového atributu tohoto uzlu se zadaným názvem a hodnotou.
- Element: DOMString `getAttributeNS`(in DOMString *`namespaceURI`*, in DOMString *`local-Name`*); slouží k získání hodnoty atributu s daným lokálním jménem a jmenným prostorem.
- Element: `void setAttributeNS`(in DOMString *`namespaceURI`*, in DOMString *`qualified-Name`*, in DOMString *`value`*); slouží k nastavení hodnoty atributu s daným názvem a v daném jmenném prostoru nebo k vytvoření nového atributu tohoto uzlu se zadaným názvem a hodnotou, v daném jmenném prostoru.

2.4. CSS

CSS, *Cascading Style Sheets*, kaskádové styly jsou opět výsledkem aktivity organizace W3C.[27] Specifikace definuje jazyk pro popis stylů, který umožňuje tvůrcům i uživatelům připojit styl (například fonty a odsazení) strukturovaným dokumentům, jako jsou HTML dokumenty a aplikace XML. Lze tak docílit oddělení definice vzhledu dokumentu od jeho obsahu a zjednodušit tvorbu webu i jeho správu.

Jazyk CSS je navržen tak, aby byl snadno čitelný pro člověka, stylový předpis je vyjádřen v terminologii, jež je běžná v DTP. Jednou ze základních vlastností je, že styl je *kaskádový*. To znamená, že se na sebe může vrstvit více definic stylu, ale platí pouze ta poslední. Tvůrce připojí k dokumentu preferovaný styl, ale uživatel ho může překrýt svým vlastním, přizpůsobeným lidskému nebo technologickému handicapu [11].

Předpis se skládá z jednotlivých *pravidel*. Každé pravidlo pak určuje vzhled jednoho nebo více elementů – těch elementů, které jsou vybrány *selektorem*, první částí pravidla. Za selektorem

[26] [5] Appendix B: SVG Document Object Model (DOM). http://www.w3.org/TR/SVG/svgdom.html
[27] W3C: *Cascading Style Sheets*. Home page. http://www.w3.org/Style/CSS/

následuje seznam deklarací uzavřených složenými závorkami, jednotlivé deklarace jsou odděleny středníkem. Každá deklarace je tvořena *vlastností* (následuje dvojtečka) a *hodnotou* vlastnosti. Selektor může vybrat všechny elementy s určitým názvem nebo elementy obsahující zadané atributy, může je označovat podle toho, kde jsou umístěny vzhledem k jiným elementům a vybírat podle *pseudotříd* (`:hover`) a *pseudoelementů* (`:first-line`). Pokud lze aplikovat na určitý element více pravidel, použije se to s větší prioritou. Zjednodušeně se jedná o pravidlo s konkrétnějším selektorem.

Kaskádový styl, který použijeme pro SVG výstup této aplikace, umožní snadnou dodatečnou manipulaci se vzhledem jednotlivých tříd grafických prvků, blíže v oddílu 4.5 – „Styl“.

2.5. Java

Java je objektově orientovaný programovací jazyk pro všeobecné použití. Byl vyvinut společností *Sun Microsystems* a po svém představení v roce 1995 se stal jedním z nejpoužívanějších programovacích jazyků. Od května roku 2007 je Java vyvíjena jako open source.

Charakteristickou vlastností je přenositelnost na různé platformy. Jednou napsaný program lze spustit na libovolném podporovaném operačním systému a hardwaru. Napsaný zdrojový kód se předkompiluje do takzvaného *bytecode*, který je pro všechny platformy stejný; odlišují se pouze virtuální stroje (*Virtual Machine*). Ty bytecode interpretují, případně za běhu přeloží do nativního kódu. Program je možné spustit všude, kde je odpovídající běhové prostředí (*Java Runtime Environment*).

Záměrem tvůrců Javy podle [12] bylo vytvořit jazyk, který by byl:

- *Jednoduchý*: Java staví na několika základních konceptech, které se vývojáři snadno naučí.
- *Podobný zavedeným technologiím*: syntaxe je založena na syntaxi populárního jazyka C++, je však snížena jeho složitost.
- *Objektově orientovaný*: programy pracují s objekty. Definují se třídy objektů, které mohou dědit od jiných tříd, mohou implementovat připravená rozhraní. Objekty nebo celé třídy poskytují vlastnosti (atributy) a metody. Metody je možné překrývat a přetěžovat. Implementace je ukrytá; objekty poskytují veřejné rozhraní umožňující manipulaci s nimi, jinak jsou zapouzdřené. Jazyk je silně typový.
- *Robustní*: javové programy jsou před spuštěním striktně kontrolovány, jazyk vynechává různé možnosti C a C++, jež bývají náchylné k chybám.
- *Bezpečný*: Java obsahuje speciální nástroje zajištění bezpečnosti, programy běžící přes síť nemohou poškodit soubory v počítači nebo obsahovat viry.
- *Přenositelný*: programy mohou být snadno přesunuty z jedné platformy na jinou, a to s minimálními změnami nebo beze změn.
- *Vysoce výkonný*: javové programy běží dostatečně rychle vzhledem k požadovaným účelům.
- *Interpretovaný*: souvisí s přenositelností, viz výše.
- *Užívající programová vlákna*: to umožňuje programu vykonávat několik úkolů naráz a zvětšit tak výkon.
- *Dynamický*: programy se mohou přizpůsobovat změnám prostředí i za jejich běhu.

Javu jsem pro tuto aplikaci zvolil hlavně kvůli přenositelnosti a velkému rozšíření, tento jazyk je mi blízký také díky kurzům absolvovaným na VŠE.

Kapitola 3

Možnosti načtení a interpretace modelu XML schématu

V předchozí kapitole jsem stručně popsal technologie užité v aplikaci pro interaktivní vizualizaci schémat. Teď bude třeba rozhodnout se, jak schéma otevřít, načíst ho do paměti, interpretovat a zpracovat. Na XML schéma lze nahlížet z několika úrovní:

Obyčejný textový soubor

Pokud bychom schéma chápali pouze jako textový soubor, museli bychom naprogramovat kompletní způsob interpretace značkování, aby pak šlo s výsledkem pracovat jako s XML dokumentem. Toto naštěstí už dávno řeší standardizovaná rozhraní jako SAX (*Simple API for XML*) a DOM (*Document Object Model*).

XML dokument

Dokument by stačilo načítat pomocí SAXu a přitom vyhledávat typické struktury tvořené převážně názvy elementů a hodnotami atributů, důležité pro interpretaci schématu. Tyto struktury by byly reprezentovány jako objekty a jejich vlastnosti, a to buď tak aby tyto objekty vyhovovaly svému účelu – chceme pouze vizualizovat schéma –, nebo aby odpovídaly specifikaci. Shoda se specifikací zajišťuje kompatibilitu, funkčnost a ulehčuje práci v případě změn. Po přečtení souboru je však ještě třeba řešit další úlohy, mezi které patří:

- Načtení importovaných, vložených a předefinovávaných schémat a zajištění, aby přitom nedošlo k zacyklení.
- Vložení typů, které jsou podle specifikace zabudované jako součást schématu.
- Musí se řešit redefinice datových typů, skupin elementů a skupin atributů.
- Je třeba poskládat reference na globálně definované typy, primární a unikátní klíče, globálně deklarované elementy a atributy, globálně definované skupiny elementů a atributů.

I v této oblasti však existují hotové implementace, kterým stačí zadat vstupní soubor, nastavit je několika parametry a nechat načíst schéma. Některou z nich určitě využijeme (viz 3.1 – „Výběr vhodné implementace XML schématu“), ušetří to hodně práce. Nakonec tedy budeme pracovat s nejvyšší úrovní, s abstraktním datovým modelem.

Abstraktní datový model

S interpretací schématu podle [2] získáme jednoduchý přístup ke všem komponentám a vlastnostem. Bližší popis zpracování je v kapitole 5 – „*Zpracování XML schématu*“.

3.1. Výběr vhodné implementace XML schématu

Nejlepší variantou bude vyhledat přijatelnou open source implementaci vyhovující specifikaci a naučit se s ní zacházet. Následuje popis čtyř známých otevřených aplikací, které se schématem pracují, a výběr jedné z nich.

3.1.1. Eclipse: Model Development Tools – XSD[1]

Model Development Tools (MDT) tvoří součást projektu organizace *Eclipse*, který je zaměřen na rozvoj a propagaci technologií pro vývoj založený na modelech – Eclipse Modeling Project. Eclipse poskytuje řadu frameworků, nástrojů a implementací technologických standardů. MDT konkrétně má nabídnout implementaci standardních metamodelů a ukázkové nástroje pro vývoj modelů, postavených na těchto metamodelech. Součástmi MDT jsou:

- Business Process Model and Notation (BPMN2),
- Ontology Definition Metamodel (EODM),
- Information Management Metamodel (IMM),
- Object Constraint Language (OCL),
- Semantics of Business Vocabulary and Business Rules (SBVR),
- Unified Modeling Language (UML2),
- UML2 Tools,
- XML Schema Infoset Model (XSD).

XML Schema Infoset Model je knihovna, která poskytuje rozhraní pro aplikace, jež prohledávají, tvoří nebo modifikují W3C XML schémata. Pro manipulaci s komponentami je možné využít rozhraní popsáné ve specifikaci, ale stejně tak lze pracovat s DOM reprezentací schématu. Při modifikacích se mění obě reprezentace odpovídajícím způsobem. Knihovna zahrnuje i služby pro serializaci a deserializaci schémat. *Cílem projektu je zcela obsáhnout funkcionalitu reprezentace XML schématu*, není ale nutné poskytnout validační služby, obvyklé u validujících parserů (Xerces-J).

3.1.2. Apache: Xerces2 Java Parser – XML Schema[2]

Xerces2 je open source XML parser vyvinutý organizací *Apache*, jeho výhodou je vysoký výkon a shoda se standardy. Krom jiného zahrnuje Xerces Native Interface, framework pro stavbu komponent a konfigurací parserů.

Xerces dokáže parsovat dokumenty napsané podle doporučení XML 1.1 a správně pracuje také se jmennými prostory podle specifikace XML Namespaces 1.1. Dále poskytuje kompletní

[1] Eclipse Modeling: *Model Development Tools (MDT)*. http://www.eclipse.org/modeling/mdt/?project=xsd#xsd

[2] The Apache XML Project: *Xerces2 Java Parser Readme*. http://xerces.apache.org/xerces2-j/ [http://xerces.apache.org/xerces2-j/xml-schema.html]

implementaci DOM Level 3 Core, Load and Save, implementuje XML Inclusions (jsou to doporučení W3C) a poskytuje podporu pro OASIS XML Catalogs v1.1.

Xerces2 je také XML schéma procesor, který až na pár drobných výjimek plně vyhovuje specifikaci [2] a [3].

3.1.3. Saxonica: Saxon[3]

Saxon je kompletní implementace XSLT 2.0, XQuery 1.0 a XPath 2.0 doporučení konsorcia W3C. Je zveřejňován společností *Saxonica*, a to zároveň pro platformu Java a .NET. Vydává se ve dvou verzích: Saxon-B je open source produkt, implementuje XSLT 2.0 a XQuery tak, že vyhovuje specifikacím pouze v základní úrovni požadavků. Saxon-SA je produkt komerční, umožňuje však aplikovat XSLT a XQuery s využitím schémat. Lze tedy importovat schéma a validovat oproti němu vstup nebo výstup a vybírat položky podle jejich typu. Saxon-SA obsahuje také samostatný XML schéma validátor a další rozšíření oproti produktu Saxon-B.

Bohužel v open source verzi není rozhraní pro přístup ke komponentám XML schématu a proto Saxon nemůžeme použít.

3.1.4. ExoLab: Castor – Source Generator XML Schema Support[4]

Castor je open source framework pro Javu, slouží pro data binding, převod dat mezi objekty Javy, XML dokumenty a relačními tabulkami.

Součástí je mimo jiné i XML Source Code Generator. Ten vytváří javové třídy, reprezentující objektový model podle vstupního XML schématu. Castor proto podporuje specifikaci W3C XML Schema [2], [3]. Objektový model reprezentuje XML schéma v paměti počítače, zatímco generátor zdrojového kódu převádí datové typy a struktury schématu do odpovídajících typů a struktur Javy. *Objektový model schématu dokáže číst i zapisovat dokumenty a manipulovat s nimi. Vyhovuje specifikaci bez omezení.* Generátor kódu zatím nenabízí mapování pro všechny komponenty.

3.1.5. Volba Xerces2

Z popsaných implementací nemůžeme použít Saxon, ostatní se zdají rovnocenné a plně vyhovují specifikaci. Nakonec jsem vybral XML schéma procesor, který je součástí parseru *Xerces2-J*. Je jednoduchý a poskytuje přesně ty možnosti, které budeme potřebovat. Třídy reprezentující jednotlivé komponenty obsahují metody pro přístup k vlastnostem, jak jsou definovány ve specifikaci. Modifikace vlastností není podporována a my bychom ji stejně nevyužili. Snadné je také ovládání procesoru, načtení schématu a jeho zpracování, použití je dobře dokumentováno. Krom toho je Xerces využíván pro účely vizualizace schématu ve známém XML editoru oXygen. Eclipse MDT a Castor by posloužili stejně dobře, jejich funkcionalita je však zbytečně široká (práce s DOM stromem, tvorba a modifikace schémat).

[3] *Saxon, The XSLT andXQuery Processor*. http://saxon.sourceforge.net
[4] *The Castor Project*. http://www.castor.org/xmlschema.html

Kapitola 4

Návrh grafického modelu

Předtím, než začnu programovat aplikaci, která bude generovat grafickou reprezentaci XML schémat, musím navrhnout, jak by tato reprezentace měla vypadat. Budu se muset rozhodnout pro správný typ modelu, navrhnout symboly – prvky modelu – a jejich propojení, budu se zabývat interaktivitou výsledného modelu. To vše je předmětem této kapitoly.

4.1. Dva typy modelů

Existují dva základní způsoby, jak zobrazit XML schéma. Oba samozřejmě vycházejí ze stromové struktury XML dokumentů, liší se ale svou podrobností a zamýšleným účelem použití. Rozdíl ukážu na výstupech aplikace *oXygen*,[1] která nabízí vykreslení obou typů.

Jako příklad poslouží toto jednoduché XML schéma:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="contact" type="ContactType"/>

    <xs:complexType name="ContactType">
        <xs:sequence>
            <xs:element name="phone" type="xs:string"/>
            <xs:element name="address" type="AddressType"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="AddressType">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="street" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

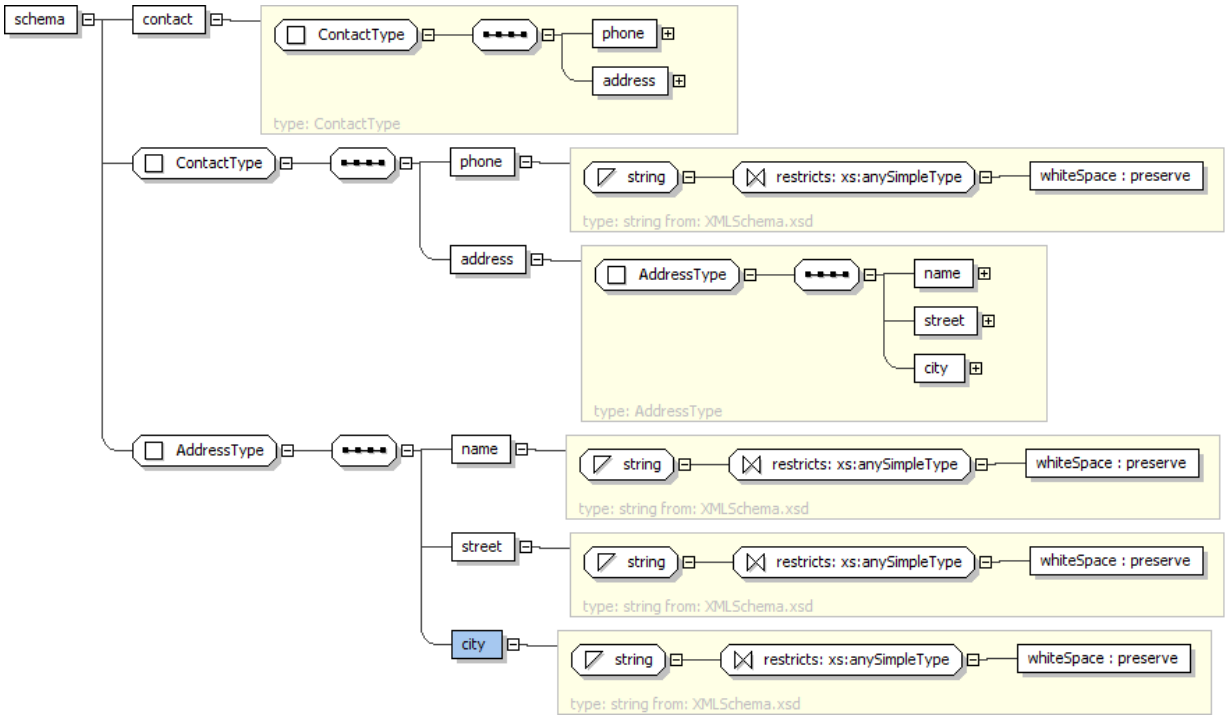
[1] Oxygen XML editor, oficiální stránky. http://www.oxygenxml.com

4.1.1. Úplný model

Úplný grafický model kopíruje XML reprezentaci schématu. Pro každý element je vykreslen vlastní symbol. Krom toho je možné rozevírat strom symbolů dál a podívat se, co se skrývá za odkazy na globálně definované komponenty schématu, jako jsou definice typů, globální deklarace elementů, jejich skupin, atd.

Například viz obrázek 4.1 – „Znázornění schématu úplným modelem“. Za symbolem elementu `contact` jsem rozbalil větev s definicí jeho typu `ContactType`. Jedná se o sekvenci elementů `phone` a `address`. Dalším rozevíráním stromu bych se dostal na definice typů těchto elementů. Stejně tak je ale mohu vidět níže jako potomky definice ContactType.

Úplný model je velmi podrobný. Nabízí komponenty, jež je možno použít při rozšiřování schématu, ale zatím nejsou aplikovatelné v popisovaném dokumentu; ukazuje detailně všechna nastavení a hodnoty, globální definice typů, deklarace skupin; uvádí importované a vložené soubory. Je tedy určen spíše tvůrcům XML schémat a ne uživatelům.



Obrázek 4.1. Znázornění schématu úplným modelem

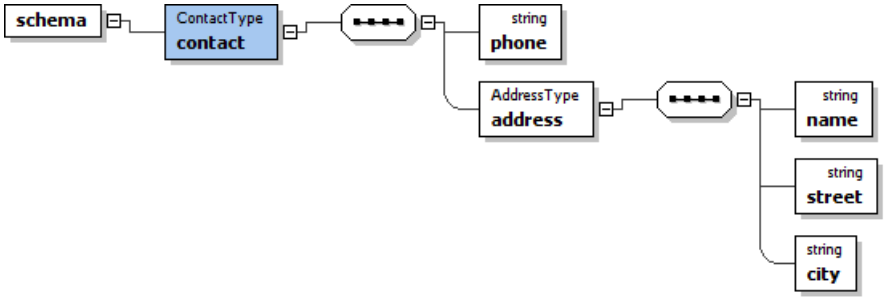
4.1.2. Logický model

Logický grafický model oproti tomu zobrazuje pouze základní informace, důležité pro tvůrce XML dokumentů, vyhovujících schématu. Symbolů je podstatně méně než v případě úplného modelu. Jde o to, ukázat uživateli možnou strukturu elementů a atributů. Reference na globální komponenty musí být poskládány stejně jako u úplného modelu, při procházení stromu ale není na první pohled patrné, ve které části schématu je prvek definován.

Například viz obrázek 4.2 – „Znázornění schématu logickým modelem“. Kořenovým elementem validních XML dokumentů je `contact`. Uvnitř něj se musí objevit sekvence elementů

`phone` a `address`. Zatímco `phone` obsahuje textový řetězec (*string*), je uvnitř elementu `address` další posloupnost, a to elementy `name`, `street` a `city`. Každý z nich je typu textový řetězec.

Logický model je tedy vhodný pro uživatele XML schémat, usnadňuje jim pochopení definovaných struktur; ale i tvůrcům umožní rychlejší kontrolu jejich práce. Právě logický model bude výstupem mé aplikace, neboť cílem je vytvořit dokumentaci schématu, která bude intuitivní a každý se v ní rychle vyzná. Dokumentace nebude popisovat XML schéma, ale XML dokumenty schématu vyhovující; konkrétně strukturu (kombinování, zanořování) elementů, jejich atributy a datové typy. Elementy schématu, které nejsou nezbytné pro pochopení účelu, budou z modelu vypuštěny, stejně tak komponenty, jež nejsou přímo využity.

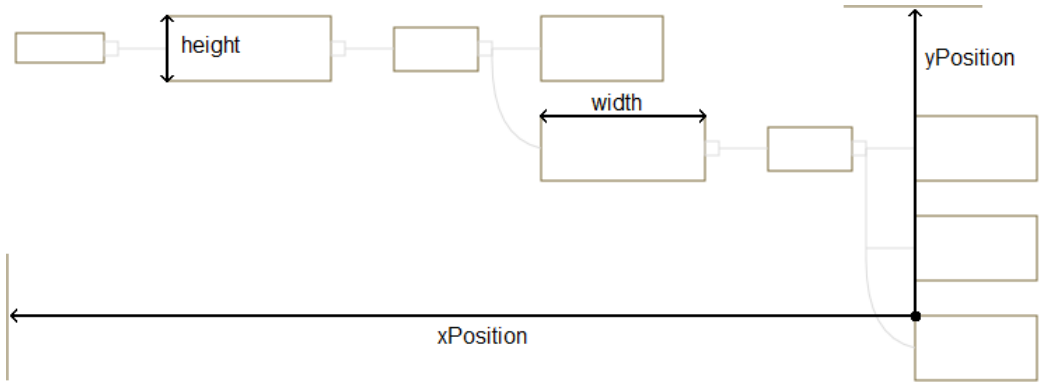


Obrázek 4.2. Znázornění schématu logickým modelem

4.2. Model jako stromová struktura abstraktních symbolů

Z uvedených ukázek a z podstaty XML dokumentů plyne, že lze symboly, tvořící grafický model, chápat jako uzly stromové struktury. Na základní úrovni proto bude existovat abstraktní symbol (`AbstractSymbol`) s odkazem na svého rodiče a řazeným seznamem odkazů na své děti. `AbstractSymbol` bude poskytovat metody pro získávání a nastavování těchto odkazů a dotazování na další informace.

Dále je pro vykreslení každého symbolu třeba znát horizontální (*`xPosition`*) a vertikální pozici (*`yPosition`*), každý symbol má šířku a výšku. `AbstractSymbol` definuje také metody pro nastavení správné šířky a výšky a pro vykreslení symbolu. Ty musí být přepsány konkrétním symbolem.



Obrázek 4.3. Model abstraktních symbolů

4.3. Symboly jednotlivých komponent schématu

Konkrétní symbol rozšiřuje vlastnosti abstraktního symbolu (`AbstractSymbol`). Povinně přepisuje metody pro nastavení správné šířky a výšky a pro vykreslení symbolu. Šířka musí být nastavena s ohledem na délku textových řetězců, které se mají vejít do symbolu. Při použití proporcionálních písem se toto dá řešit pouze přibližně.

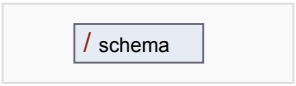
Symbol také poskytuje metody pro získávání a nastavování dále popsaných parametrů. Pokud nebude některý z parametrů uveden (například proto, že zastupuje volitelnou vlastnost), nebude jeho hodnota vypsána nebo jinak zpracována.

Následuje přehled konkrétních symbolů a jejich vztahů k XML schématu. V popisech parametrů vycházím z vlastností a hodnot definovaných ve specifikaci [2].

4.3.1. `schema`

`SymbolSchema` znázorňuje kořenový element schématu.

Pro vykreslení není třeba uvádět žádné parametry.



Obrázek 4.4. Symbol `schema`

4.3.2. `element`

`SymbolElement` zobrazuje jméno informačních položek element, jejich typ a zařazení ke jmennému prostoru a další vlastnosti dostupné z komponenty deklarace elementu (*Element declaration*).

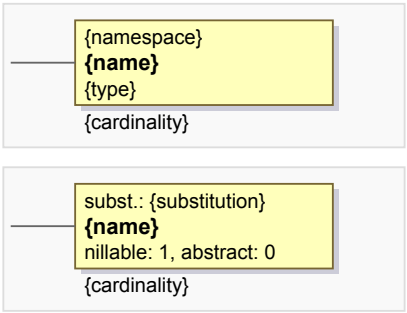
- Parametr *`name`* je textový řetězec, obsahující lokální část jmen informačních položek element, jež jsou validovány. Hodnotu poskytne deklarace elementu ve vlastnosti *name*.
- Parametr *`namespace`* je řetězec se jmenným prostorem. Ten kvalifikuje informační položky element. Cílový jmenný prostor udává deklarace elementu ve vlastnosti *namespace*.
- Parametr *`type`* je řetězec, který obsahuje jméno typu informačních položek element, pokud je tento typ pojmenovaný; nebo, pokud je typ anonymní a zároveň jednoduchý, obsahuje jméno základního typu, ze kterého je typ informačních položek element odvozen. Základem je vlastnost *name* definice typu (*Type definition*).
- Parametr *`cardinality`* je řetězec, který udává minimální a maximální počet výskytů informační položky element. Pokud není uveden, znamená to, že hodnoty jsou implicitní (min = max = 1), a řetězec nebude vykreslen. Minimální a maximální počet výskytů je dán vlastnostmi částice (*Particle*) obsahující tuto deklaraci elementu. Jejich názvy jsou *min occurs* a *max occurs*.

- Parametr *nillable* booleovského typu říká, zda mohou mít informační položky element prázdný obsah (hodnota `true`), nebo ne (`false`). Hodnotu udává deklarace elementu ve vlastnosti *nillable*.

- Parametr *abstr* je booleovského typu. Pokud je hodnota nastavena na `true`, není tato deklarace sama o sobě použita k validování obsahu elementů. Hodnotu udává deklarace elementu ve vlastnosti *abstract*.

- Parametr *substitution* je řetězec s názvem substituční skupiny, do které tato deklarace patří. Skutečnou hodnotu poskytuje deklarace elementu ve vlastnosti *substitution group affiliation*.

Do symbolu bude nutné vtěsnat hodně informací a přitom nezaplnit velkou plochu. Proto budou poslední tři vlastnosti zobrazeny teprve při najetí myši místo vlastností *namespace* a *type*.



Obrázek 4.5. Symbol `element` (normální režim; při najetí myši)

4.3.3. `attribute`

`SymbolAttribute` zobrazuje jméno informační položky atribut, její typ a zařazení ke jmennému prostoru a další vlastnosti dostupné z komponent užití atributu (*Attribute use*) a deklarace atributu (*Attribute declaration*).

- Parametr *name* je textový řetězec, obsahující lokální část jména informační položky atribut, jež je validována. Hodnotu poskytne deklarace atributu ve vlastnosti *name*.

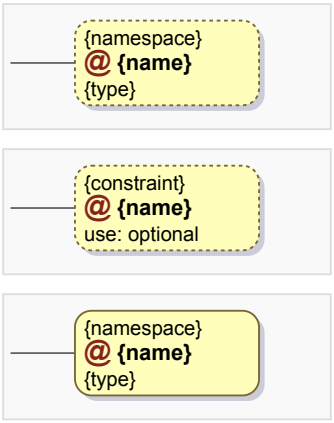
- Parametr *namespace* je řetězec se jmenným prostorem. Ten kvalifikuje informační položky atribut. Cílový jmenný prostor udává deklarace atributu svou vlastností *namespace*.

- Parametr *type* je řetězec, který obsahuje jméno typu informační položky atribut, pokud je tento typ pojmenovaný; nebo, pokud je typ anonymní, obsahuje jméno základního typu, ze kterého je typ informační položky atribut odvozen. Základem je vlastnost *name* definice jednoduchého typu (*Simple type definition*).

- Parametr *required* booleovského typu říká, zda musí být přítomna odpovídající informační položka atribut (hodnota `true`), nebo zda je volitelná (`false`). Kromě vypsání odpovídajícího řetězce bude volitelný atribut ohraničen přerušovanou čárou, zatímco povinný atribut plnou. Parametr má stejnou hodnotu jako vlastnost užití atributu s názvem *required*.

- Parametr *constraint* je textový řetězec, který specifikuje omezení pro hodnotu informační položky atribut (fixní nebo defaultní hodnota). Řetězec je vytvořen z vlastnosti *value constraint*. Tu poskytuje komponenta užití atributu nebo deklarace atributu.

  Hodnoty posledních dvou parametrů se zobrazí teprve po najetí myši na symbol `attribute`.



Obrázek 4.6. Symbol `attribute` (volitelný atribut v normálním režimu; a po najetí myši; povinný atribut v normálním režimu)

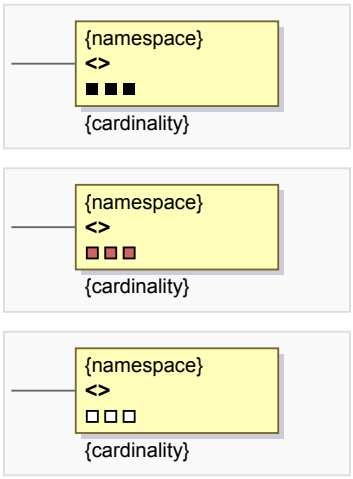
4.3.4. `any`

`SymbolAny` představuje divokou kartu (*Wildcard*) pro elementy. Na jejím místě mohou být informační položky element s libovolným lokálním jménem, které ale musí vyhovět omezením kladeným na jejich jmenný prostor.

- Parametr *namespace* je textový řetězec, představující omezení kladené na jmenný prostor informačních položek element. Řetězec je utvořen z vlastnosti *namespace constraint* komponenty divoká karta.

- Parametr *processContents* je celé číslo, které slouží k rozlišení způsobů zpracování informačních položek element. Hodnota vychází z vlastnosti *process contents* divoké karty a závisí na ní vyobrazení symbolu. Parametr nabývá hodnot:
  - `3` – způsob zpracování `lax`. Pokud existuje unikátní deklarace položky, musí položka této deklaraci vyhovovat.
  - `2` – odpovídá způsobu zpracování `skip`. Nevzniká žádné omezení, položka musí být pouze dobře strukturované XML.
  - `1` (a ostatní přípustné hodnoty) – způsob zpracování `strict`. Informační položka musí mít přiřazen xsi:type, nebo musí být k dispozici její deklarace, oproti které je prováděna validace.

- Parametr *cardinality* je řetězec, který udává minimální a maximální počet výskytů informační položky element. Pokud není uveden, znamená to, že hodnoty jsou implicitní (min = max = 1), a řetězec nebude vykreslen. Minimální a maximální počet výskytů je dán vlastnostmi částice (*Particle*) obsahující tuto divokou kartu. Jejich názvy jsou *min occurs* a *max occurs*.



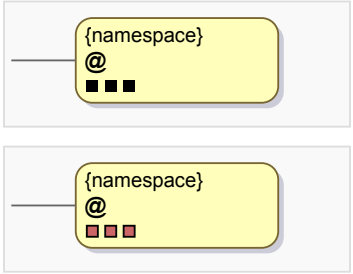
Obrázek 4.7. Symbol `any` (způsob zpracování `strict`; způsob zpracování `skip`; způsob zpracování `lax`)

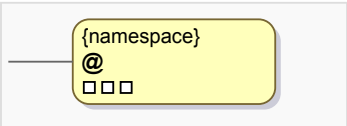
4.3.5. `anyAttribute`

`SymbolAnyAttribute` představuje divokou kartu (*Wildcard*) pro atributy. Na jejím místě mohou být informační položky atribut s libovolným lokálním jménem, které ale musí vyhovět omezením kladeným na jejich jmenný prostor.

- Parametr *`namespace`* je textový řetězec, představující omezení kladené na jmenný prostor informačních položek atribut. Řetězec je utvořen z vlastnosti *namespace constraint* komponenty divoká karta.

- Parametr *`processContents`* je celé číslo, které slouží k rozlišení způsobů zpracování informačních položek atribut. Hodnota vychází z vlastnosti *process contents* divoké karty a závisí na ní vyobrazení symbolu. Parametr nabývá hodnot:
  - `3` – způsob zpracování `lax`. Pokud existuje unikátní deklarace položky, musí položka této deklaraci odpovídat.
  - `2` – odpovídá způsobu zpracování `skip`. Nevzniká žádné omezení, položka musí být pouze dobře strukturované XML.
  - `1` (a ostatní přípustné hodnoty) – způsob zpracování `strict`. Informační položka musí mít přiřazen xsi:type, nebo musí být k dispozici její deklarace, oproti které je prováděna validace.



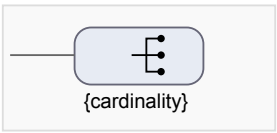


Obrázek 4.8. Symbol `anyAttribute` (způsob zpracování `strict`; způsob zpracování `skip`; způsob zpracování `lax`)

4.3.6. `all`

`SymbolAll` znázorňuje kompozitor `all` modelové skupiny (*Model group*). Ten říká, že se děti informační položky element, definované v modelové skupině jako *particles*, mohou vyskytovat v libovolném pořadí.

- Parametr *`cardinality`* je řetězec, který udává minimální a maximální počet výskytů dětí informační položky element. Pokud není uveden, znamená to, že hodnoty jsou implicitní (min = max = 1), a řetězec nebude vykreslen. Minimální a maximální počet výskytů je dán vlastnostmi částice (*Particle*) obsahující tuto modelovou skupinu. Jejich názvy jsou *min occurs* a *max occurs*.

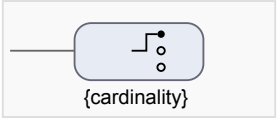


Obrázek 4.9. Symbol `all`

4.3.7. `choice`

`SymbolChoice` znázorňuje kompozitor `choice` modelové skupiny (*Model group*). Pouze jedna z částic (*Particle*), definovaných v modelové skupině jako *particles*, se může v XML dokumentu objevit jako dítě informační položky element.

- Parametr *`cardinality`* je řetězec, který udává minimální a maximální počet výskytů dětí informační položky element. Pokud není uveden, znamená to, že hodnoty jsou implicitní (min = max = 1), a řetězec nebude vykreslen. Minimální a maximální počet výskytů je dán vlastnostmi částice (*Particle*) obsahující tuto modelovou skupinu. Jejich názvy jsou *min occurs* a *max occurs*.

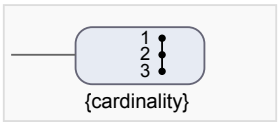


Obrázek 4.10. Symbol `choice`

4.3.8. `sequence`

`SymbolSequence` znázorňuje kompozitor `sequence` modelové skupiny (*Model group*). Každá z částic (*Particle*), definovaných v modelové skupině jako *particles*, se může objevit v XML dokumentu jako dítě informační položky element, a to ve specifikovaném pořadí.

- Parametr *`cardinality`* je řetězec, který udává minimální a maximální počet výskytů dětí informační položky element. Pokud není uveden, znamená to, že hodnoty jsou implicitní (min = max = 1), a řetězec nebude vykreslen. Minimální a maximální počet výskytů je dán vlastnostmi částice (*Particle*) obsahující tuto modelovou skupinu. Jejich názvy jsou *min occurs* a *max occurs*.



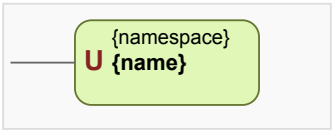
Obrázek 4.11. Symbol `sequence`

4.3.9. `unique`

`SymbolUnique` znázorňuje část definice identitního omezení (*Identity-constraint definition*), konkrétně jméno a jmenný prostor kategorie `unique`. Ta zajišťuje jedinečnost hodnot v rámci obsahu vymezeného selektorem 4.3.12 – „`selector`“, které jsou výsledkem vyhodnocení XPath výrazů uvedených v polích 4.3.13 – „`field`“.

- Parametr *`name`* je textový řetězec, obsahující jméno definice identitního omezení. Hodnotu poskytne tato definice ve vlastnosti *name*.

- Parametr *`namespace`* je řetězec se jmenným prostorem. Ten kvalifikuje definici identitního omezení. Jmenný prostor udává definice svou vlastností *namespace*. Dvojice jméno a jmenný prostor identifikuje definici identitního omezení, a proto musí být v rámci XML schématu unikátní.



Obrázek 4.12. Symbol `unique`

4.3.10. `key`

`SymbolKey` znázorňuje část definice identitního omezení (*Identity-constraint definition*), konkrétně jméno a jmenný prostor kategorie `key`. Ta zajišťuje jedinečnost a přítomnost hodnot v rámci obsahu vymezeného selektorem 4.3.12 – „`selector`“, které jsou výsledkem vyhodnocení XPath výrazů uvedených v polích 4.3.13 – „`field`“.

- Parametr *name* je textový řetězec, obsahující jméno definice identitního omezení. Hodnotu poskytne tato definice ve vlastnosti *name*.

- Parametr *namespace* je řetězec se jmenným prostorem. Ten kvalifikuje definici identitního omezení. Jmenný prostor udává definice svou vlastností *namespace*. Dvojice jméno a jmenný prostor identifikuje definici identitního omezení, a proto musí být v rámci XML schématu unikátní.

Obrázek 4.13. Symbol `key`

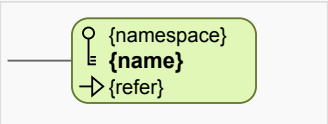
4.3.11. `keyref`

`SymbolKeyref` znázorňuje část definice identitního omezení (*Identity-constraint definition*), konkrétně jméno a jmenný prostor kategorie `keyref`. Ta zajišťuje, že hodnoty, které jsou výsledkem vyhodnocení XPath výrazů uvedených v polích 4.3.13 – „`field`“, budou odpovídat hodnotám, jež specifikuje vlastnost *referenced key* definice identitního omezení. Tato podmínka se vyhodnocuje v rámci obsahu určeného selektorem 4.3.12 – „`selector`“.

- Parametr *name* je textový řetězec, obsahující jméno definice identitního omezení. Hodnotu poskytne tato definice ve vlastnosti *name*.

- Parametr *namespace* je řetězec se jmenným prostorem. Ten kvalifikuje definici identitního omezení. Jmenný prostor udává definice svou vlastností *namespace*. Dvojice jméno a jmenný prostor identifikuje definici identitního omezení, a proto musí být v rámci XML schématu unikátní.

- Parametr *refer* je řetězec, který slouží jako odkaz na jinou definici identitního omezení kategorie `key` nebo `unique`. Hodnotu poskytne tato definice ve vlastnosti *referenced key*.



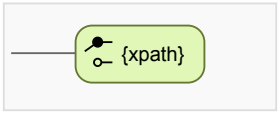
Obrázek 4.14. Symbol `keyref`

4.3.12. `selector`

`SymbolSelector` slouží ke zobrazení vlastnosti *selector* definice identitního omezení (*Identity-constraint definition*).

- Parametr *xpath* je textový řetězec. Specifikuje omezený XPath výraz, relativní k instanci deklarovaného elementu. Výraz musí identifikovat množinu uzlů – podřízených elementů,

na které se vztahuje omezení. Hodnota parametru se získá z vlastnosti *selector* definice identitního omezení.



Obrázek 4.15. Symbol `selector`

4.3.13. `field`

`SymbolField` slouží ke zobrazení jednoho prvku ze seznamu – vlastnosti *fields* definice identitního omezení (*Identity-constraint definition*).

- Parametr *`xpath`* je textový řetězec. Specifikuje omezený XPath výraz, relativní ke každému elementu, který je vybrán selektorem 4.3.12 – „`selector`“. Výraz musí identifikovat konkrétní uzel (element nebo atribut), jehož obsah nebo hodnota musí být jednoduchého typu a je použita v omezení. Hodnota parametru se získá z vlastnosti *fields* definice identitního omezení.

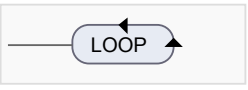
Obrázek 4.16. Symbol `field`

4.3.14. smyčka

`SymbolLoop` je pomocný symbol, který nepatří do XML schématu. Vykreslí se, pokud by mělo dojít k zacyklení. Deklarace elementu totiž může nepřímo obsahovat samu sebe, a to za splnění těchto podmínek:

1. Deklarace elementu je komplexního typu.
2. Komplexní typ (1) obsahuje částici (*Particle*).
3. Termínem (*term*) částice (2) je modelová skupina (*ModelGroup*) – vždy složená z dalších částic.
4. Termínem alespoň jedné z částic (3) je deklarace elementu totožná s deklarací uvedenou v bodě (1); nebo je termínem alespoň jedné z částic (3) modelová skupina a v tom případě se postupuje rekurzivně od bodu (3), dokud nebude nalezena deklarace elementu totožná s deklarací uvedenou v bodě (1).

Nemá parametry.



Obrázek 4.17. Symbol smyčky

4.4. Interaktivita modelu

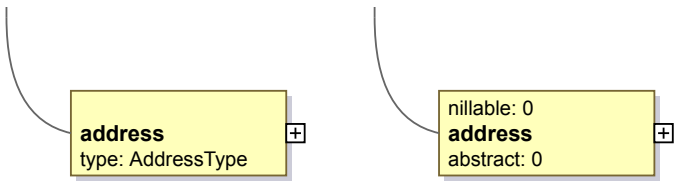
Jedním z požadavků kladených na grafický model je jeho interaktivita. SVG se dá rozpohybovat pomocí ECMAScriptu (JavaScriptu). Jednoduše půjde zajistit změnu zobrazovaných informací při najetí kurzoru myši na symboly 4.3.2 – „`element`“ a 4.3.3 – „`attribute`“. Složitější je implementovat rozbalování a skrývání podstromů jednotlivých symbolů. Aby to bylo možné, musí SVG dokument znát umístění symbolů v rámci jejich stromu. K vysvětlení použiji následující příklad:

```
<g id='_1_1_1_2' class='box' transform='translate(395,121)'>
    <rect class='shadow' x='3' y='3' width='117' height='46'/>
    <rect class='boxelement' x='0' y='0' width='117' height='46'
        onmouseover='makeVisible("_1_1_1_2")' onmouseout='makeHidden("_1_1_1_2")'/>
    <text class='hidden' visibility='hidden' x='5' y='13'>nillable: 0</text>
    <text class='hidden' visibility='hidden' x='5' y='41'>abstract: 0</text>
    <text class='strong' x='5' y='27'>address</text>
    <text class='visible' x='5' y='41'>type: AddressType</text>
    <line class='connection' id='p_1_1_1_2' x1='-35' y1='-48' x2='-35' y2='-40'/>
    <path class='connection' d='M-35,-40 Q-35,15 0,23'/>
   <use x='116' y='17' xlink:href='#plus' id='s_1_1_1_2' onclick='show("_1_1_1_2")'/>
</g>
```

Jedná se o XML reprezentaci symbolu `address`. Ten se skládá z několika SVG elementů a je zobrazen jako 4.18 – „Symbol elementu address (v normálním režimu; při najetí myši)“.

- Symbol je tvořen dvěma obdélníky. První je mírně posunutý a tvoří lehký stín. Druhý z nich tvoří hlavní box a je citlivý na najetí myši.
- Dále symbol obsahuje čtyři textové položky. První dvě s obsahem „nillable: 0“ a „abstract: 0“ jsou teď skryty. Řetězce „address“ a „type: AddressType“ jsou naopak viditelné.
- Následuje úsečka a na ní navazující křivka, která zleva připojuje symbol ke svému rodiči.
- Nakonec je použit předem definovaný symbol „plus“, což je malý čtverec se znakem plus. Je citlivý na kliknutí myši.

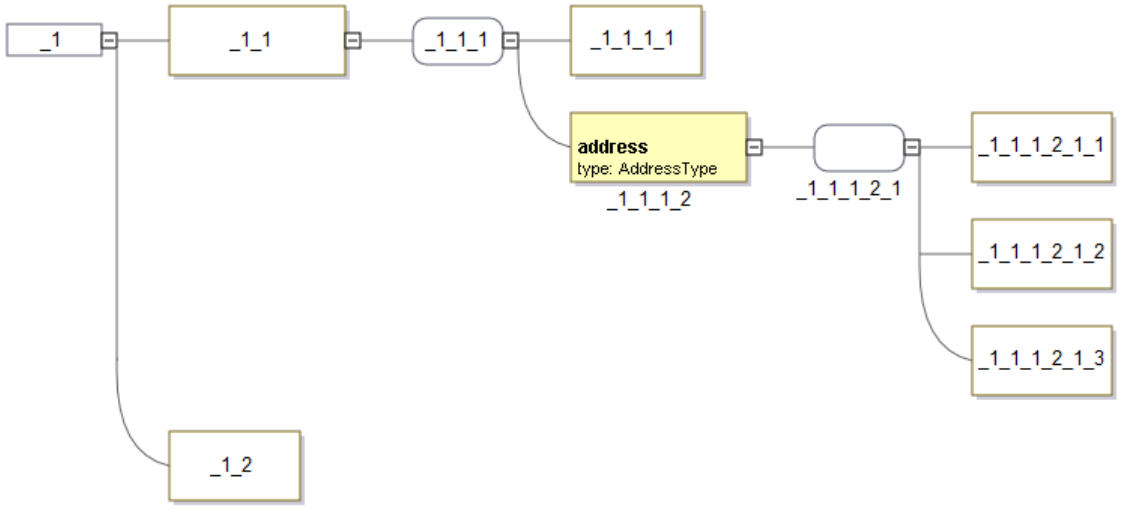
Navíc, aby se se symbolem dalo pracovat jako s jedním celkem, jsou všechny popsané položky obaleny jako skupina elementem `<g>`. Ten dále umožňuje transformaci pozice symbolu a jeho jednoznačnou identifikaci v rámci stromu.



Obrázek 4.18. Symbol elementu address (v normálním režimu; při najetí myši)

Pro změnu zobrazovaných položek při najetí myši stačí, aby měl každý symbol jednoznačný identifikátor. Ten je předáván ECMAScriptovým metodám `makeVisible(id)` a `makeHidden(id)` při najetí kurzoru myši na hlavní box symbolu, respektive při jeho sjetí z boxu. Změna spočívá v tom, že položky, které měly nastavenou třídu (*class*) na hidden, budou nyní viditelné. Jedná se o textové řetězce „nillable: 0“ a „abstract: 0“. Naopak položky ve třídě visible budou dočasně skryty. Sem patří pouze řetězec „type: AddressType“. Viz obrázek 4.18 – „Symbol elementu address (v normálním režimu; při najetí myši)“.

Zde použitý identifikátor symbolu je poměrně složitý, jedná se o řetězec `_1_1_1_2`. Jeho použití je však opodstatněné. Už jsem napsal, že pro práci s podstromy symbolů, musí být v SVG dokumentu patrné, do které části stromu symbol spadá. To lze jednoznačně určit pomocí takovýchto identifikátorů, uvedením cesty od kořene stromu až ke konkrétnímu symbolu. Podívejte se na obrázek 4.19 – „Model s kódy symbolů“.



Obrázek 4.19. Model s kódy symbolů

Kořenový symbol má identifikátor `_1`. Potržítko je nutné kvůli přípustným hodnotám attributu `id`, je tedy zvoleno i jako oddělovač. Kořenový symbol má dále dvě děti. Jejich identifikátory jsou tvořeny identifikátorem rodiče a přidanou vlastní částí, která udává jejich pozici. Dostáváme řetězce `_1_1` a `_1_2`. Analogicky symbol `_1_1` má dítě `_1_1_1`. Symbol s identifikátorem `_1_1_1` má dvě děti a druhé z nich (`_1_1_1_2`) je popisovaný prvek, symbol elementu `address`.

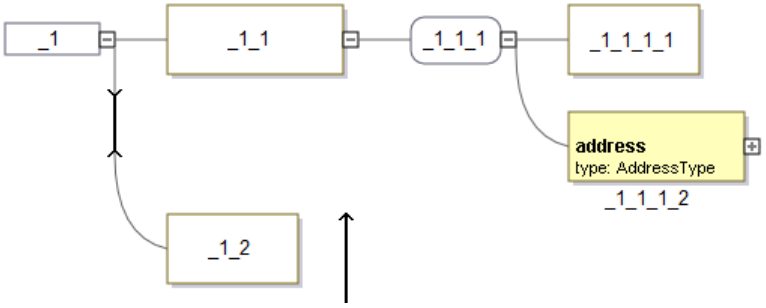
Při kliknutí na symbol minus za boxem `address` se zavolá ECMAScriptová metoda `show("_1_1_1_2")`. Ta provede následující:

- Změní tu symbol minus na plus, přepsáním atributu `xlink:href` elementu `<use>` tak, aby odkazoval na předem definovaný grafický objekt `plus`.
- Schová celý podstrom symbolů připojený zprava k boxu. Patří do něj všechny symboly, jejichž identifikátor začíná také na `_1_1_1_2`. U elementů `<g>` obalujících objekty, z nichž se symbol skládá, se nastaví atribut `visibility` na hodnotu *hidden*.
- Vzniklý prostor se zaplní vedlejšími větvemi. Ty se posunou směrem nahoru, přenastavením hodnoty atributu `transform` u seskupujícího elementu `<g>`. Výpočet vzdálenosti, o kterou

se mohou symboly posunout, je nutné provést pouze pro jeden symbol, pro ostatní je vzdálenost stejná. Zde poslouží také systém identifikace. Navíc je třeba zkrátit úsečku propojující rodičovské symboly s posouvanou větví.

Výsledek je možné vidět na obrázku 4.20 – „Model po skrytí podstromu elementu address“. Pokud teď klikneme na symbol plus, zavolá se opět metoda `show("_1_1_1_2")`. Ta teď ale bude pracovat přesně naopak, než bylo popsáno.

Kromě výše uvedené funkčnosti budou poskytnuta tlačítka pro zobrazení a skrytí všech symbolů, až na kořenový. Časem přibudou i škálovací tlačítka *zoom in* a *zoom out*, zatím se lze obejít bez nich a využít funkce prohlížeče.



Obrázek 4.20. Model po skrytí podstromu elementu address

4.5. Výsledná SVG reprezentace schématu

Když splním všechny podmínky, získám z aplikace výstup ve formátu SVG, jehož grafickou reprezentaci si můžete prohlédnout na obrázku 4.21 – „Výstup aplikace pro výše uvedený příklad XML schématu“. V příloze C – „*Příklad*“ je pro přehlednost znovu uvedeno jak vstupní XML schéma, tak i SVG výstup v XML i grafické reprezentaci.

Styl

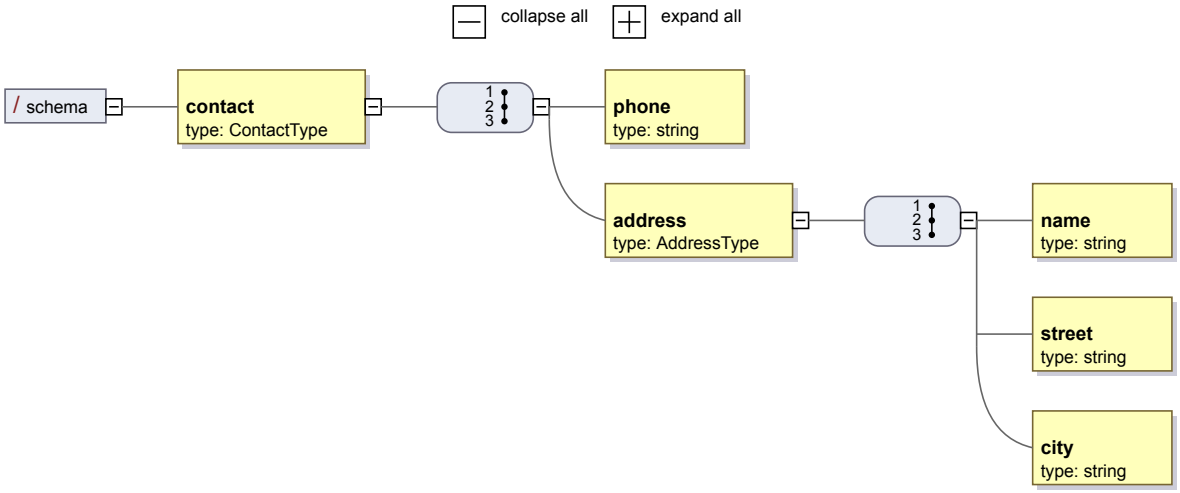
Výsledný diagram se ale dá ještě dodatečně upravovat, jeho vzhled je totiž nastaven v CSS stylu. Aplikace dokáže styl generovat jako součást SVG dokumentu nebo zvlášť, nebo pouze připojit existující externí styl. Kaskádové styly využívají selektory pro výběr určitých elementů, kterým pak nastaví vzhled. Výběr může být určen názvem tagu, pokud však chceme jednotně upravit určité logické celky, lze využít jejich zařazení do tříd (*class*). V SVG výstupu vznikly tyto třídy:

- *strong* pro důležité nápisy jako jsou názvy elementů a atributů. Písmo je větší a tučné.
- *small* pro text, jenž je součástí grafiky, psaný menším písmem.
- *big* pro text, jenž je součástí grafiky, psaný větším písmem.
- *button* pro tlačítka; podstatou je, že má v parametru *pointer-events* zapnutou citlivost na určité události, především nám jde o kliknutí myši.
- *shadow* je třída pro stín zobrazovaný za některými boxy.
- *connection* zahrnuje úsečky a křivky propojující jednotlivé grafické symboly (boxy).
- *empty* pro tvary obtažené, bez výplně.

- *filled* pro tvary vyplněné, bez obtažení.
- *boxelement* pro box symbolu 4.3.2 – „`element`“.
- *boxattribute1* pro box symbolu 4.3.3 – „`attribute`“, pokud je použití atributu povinné.
- *boxattribute2* pro box symbolu 4.3.3 – „`attribute`“, pokud je použití atributu volitelné.
- *boxany* pro box symbolu 4.3.4 – „`any`“.
- *boxanyattribute* pro box symbolu 4.3.5 – „`anyAttribute`“.
- *boxschema* pro box symbolu 4.3.1 – „`schema`“.
- *boxcompositor* pro boxy symbolů 4.3.6 – „`all`“, 4.3.7 – „`choice`“, 4.3.8 – „`sequence`“.
- *boxloop* pro box symbolu 4.3.14 – „smyčka“.
- *boxidc* pro boxy symbolů 4.3.9 – „`unique`“, 4.3.10 – „`key`“, 4.3.11 – „`keyref`“.
- *boxselector* pro box symbolu 4.3.12 – „`selector`“.
- *boxfield* pro box symbolu 4.3.13 – „`field`“.
- *lax* slouží k dalšímu nastavení vzhledu u boxů 4.3.4 – „`any`“ a 4.3.5 – „`anyAttribute`“, pokud je u nich způsob zpracování nastaven na *lax*.
- *skip* slouží k dalšímu nastavení vzhledu u boxů 4.3.4 – „`any`“ a 4.3.5 – „`anyAttribute`“, pokud je u nich způsob zpracování nastaven na *skip*.
- *strict* slouží k dalšímu nastavení vzhledu u boxů 4.3.4 – „`any`“ a 4.3.5 – „`anyAttribute`“, pokud je u nich způsob zpracování nastaven na *strict*.

Původní styl generovaný aplikací

```
svg {pointer-events: none;}
text {font-family: arial; font-size: 11px;}
line, polyline, polygon {fill: none; stroke: black;}

.strong {font-size: 12px; font-weight: bold;}
.small {font-size: 10px;}
.big {font-size: 15px; fill: #882222;}

.button {fill: white; stroke: black; pointer-events: all;}
.shadow {fill: #ccccd8; stroke: none;}
.connection {fill: none; stroke: #666666;}
.empty {fill: none; stroke: black;}
.filled {fill: black; stroke: none;}

.boxelement, .boxany, .boxattribute1, .boxanyattribute
  {fill: #FFFFBB; stroke: #776633; pointer-events: all;}
.boxattribute2
  {fill: #FFFFBB; stroke: #776633; pointer-events: all; stroke-dasharray: 2;}
.boxschema, .boxloop, .boxcompositor {fill: #E7EBF3; stroke: #666677;}
.boxselector, .boxfield, .boxidc {fill: #E0F7B7; stroke: #667733;}

.lax {fill: white; stroke: black;}
.skip {fill: #cc6666; stroke: black;}
.strict {fill: black; stroke: none;}
```



Obrázek 4.21. Výstup aplikace pro výše uvedený příklad XML schématu

Kapitola 5

Zpracování XML schématu

V této kapitole ukážu způsob procházení mezi komponentami schématu a vytvořím tak koncept jádra aplikace. Jednotlivé oddíly se věnují zpracování konkrétních komponent a vedle algoritmu zpracování obsahují také popis převzatý ze specifikace [2]. Odkaz na konkrétní část specifikace uvádím u každého oddílu v poznámce pod čarou.

Jako základní knihovnu, na které postavím aplikaci, jsem po předchozím uvážení zvolil open-source parser Xerces. Ten poskytuje plnou podporu XML schémat podle [1], [2] a [3], až na několik omezení, které v naprosté většině případů nebudou tvořit překážku.[1] Implementace Xerces tak bude pracovat v souladu se specifikací W3C a bude užívat její termíny.

Obrázek 2.1 – „Diagram komponent XML schématu“ nám umožní udělat si snadno představu o vazbách a základních vlastnostech jednotlivých komponent XML schématu, jež jsou definovány v třetí části specifikace.[2]

5.1. Zpracování samotného schématu[3]

Schéma (*Schema*, v Xercesu třída `XSModel`) slouží na abstraktní úrovni jako kontejner pro jednotlivé komponenty. Jedná se o komponenty globální, tedy pojmenované a anotace.

Metoda `processModel(XSModel` *`model`*`)` bude mít na starost zpracování schématu.

1. Vytvoří se symbol 4.3.1 – „`schema`“ a bude vložen jako kořen do stromové struktury symbolů.
2. Bude volána pomocná metoda `processElementDeclarations(`*`elementDeclarations`*`)` [5.2 – „Zpracování kolekce deklarací elementů“], parametrem jsou všechny globálně deklarované elementy.

Ostatní vlastnosti schématu (*attribute declarations*, *model group definitions*, *attribute group definitions*, *type definitions*, *notation declarations*, *annotations*) pro tvorbu logického modelu grafické reprezentace takto přímo nevyužijeme; Xerces nám některé z nich poskytne později skrze reference jako globálně definované typy, atributy, elementy a jejich skupiny. Při vykreslování tedy vůbec nebudeme muset pracovat s komponentami definice skupiny atributů (*Attribute group definition*), definice modelové skupiny (*Model group definition*) a deklarace notace

[1] The Apache XML Project: *Xerces2 Java Parser Readme*. XML Schema. http://xerces.apache.org/xerces2-j/xmlschema.html
[2] [2] Schema Component Details. http://www.w3.org/TR/xmlschema-1/#components
[3] [2] http://www.w3.org/TR/xmlschema-1/#Schemas

(*Notation declaration*), které jsou dostupné pouze ze schématu, a nebudeme potřebovat ani anotace (*Annotation*).

5.2. Zpracování kolekce deklarací elementů

Kolekce deklarací elementů (v Xercesu obalené obecnou třídou `XSNamedMap`).

Metoda `processElementDeclarations(XSNamedMap map)` provádí zpracování globálních deklarací elementů. Metoda obsahuje cyklus.

1. Prochází se kolekcí deklarací elementů:
   a. Pro každý prvek kolekce se volá `processElementDeclaration(elementDeclaration, null)` [5.3 – „Zpracování deklarace elementu“]. První parametr je konkrétní deklarace elementu (*Element declaration*), druhý je řetězec vyjadřující kardinalitu elementu. V případě globální deklarace má vždy hodnotu `null`.

5.3. Zpracování deklarace elementu[4]

Deklarace elementu (*Element declaration*, v Xercesu třída `XSElementDeclaration`) umožňuje:

- lokální validaci hodnot informační položky element za použití definice typu;
- specifikování defaultních nebo fixních hodnot pro informační položku element;
- zajištění jedinečnosti hodnot a nastavení referenčních omezení v rámci hodnot příbuzných elementů a atributů;
- kontrolování vzájemné nahraditelnosti elementů pomocí mechanismu substitučních skupin (*element substitution groups*).

Metoda `processElementDeclaration(XSElementDeclaration elementDeclaration, String cardinality)` zpracuje deklaraci elementu:

1. Dojde k vytvoření symbolu 4.3.2 – „`element`“ a jeho připojení na odpovídající místo stromové struktury. U symbolu je třeba nastavit základní informace, poskytované přímo objektem *elementDeclaration*, ale také některé dodatečné informace, například řetězec s typem elementu, získaný voláním pomocné metody `getTypeString(typeDefinition)` [5.8 – „Zpracování definice jednoduchého typu“], a řetězec uvádějící minimální a maximální počet výskytů, získaný z parametru *cardinality*. Popis jednotlivých položek naleznete u grafického návrhu symbolu `element`.
2. Zjišťuje se, zda se stejná deklarace nevyskytuje mezi předky této deklarace. Došlo by k zacyklení a vykreslování modelu by se nikdy řádně neukončilo. Kontrolu provádí metoda `processLoop(XSElementDeclaration elementDeclaration)`, která má za úkol v případě objevení cyklu připojit symbol 4.3.14 – „smyčka“ a zastavit zpracování následníků.
3. Pokud má element komplexní datový typ, proběhne na tomto místě zpracování jeho definice. To obstarává metoda `processComplexTypeDefinition(complexTypeDefinition)` [5.9 – „Zpracování definice komplexního typu“]

[4] [2] http://www.w3.org/TR/xmlschema-1/#cElement_Declarations

4. Bude volána pomocná metoda `processIdentityConstraints(IdentityConstraints)` [5.10 – „Zpracování kolekce identitních omezení“], parametrem jsou všechna identitní omezení definovaná v rámci této deklarace.

5.4. Zpracování kolekce užití atributů

Kolekce užití atributů (v Xercesu obalené obecnou třídou `XSObjectList`).

Metoda `processAttributeUses(XSObjectList attributeUses)` provádí zpracování kolekce užití atributů. Metoda obsahuje cyklus.

1. Prochází se kolekcí:
   a. Pro každý prvek kolekce se volá metoda `processAttributeUse(attributeUse)` [5.5 – „Zpracování užití atributu“]. Parametr je konkrétní užití atributu (*Attribute use*).

5.5. Zpracování užití atributu[5]

Užití atributu (*Attribute use*, v Xercesu třída `XSAttributeUse`) je pomocná komponenta, která kontroluje výskyt a defaultní chování deklarace atributu. Pro deklaraci atributu plní v rámci komplexního typu podobnou úlohu jako částice (*Particle*) pro deklaraci elementu.

Metoda `processAttributeUse(XSAttributeUse attributeUse)` slouží ke zpracování užití atributu a zároveň i samotné deklarace atributu:

1. Na tomto místě proběhne zpracování deklarace atributu [5.6 – „Zpracování deklarace atributu“].

5.6. Zpracování deklarace atributu[6]

Deklarace atributu (*Attribute declaration*, v Xercesu třída `XSAttributeDeclaration`) umožňuje:

- lokální validaci hodnot informační položky atribut za použití definice jednoduchého typu;
- specifikování defaultních nebo fixních hodnot pro informační položku atribut.

Pro jednoduchost je deklarace atributu zpracována uvnitř metody `processAttributeUse(XSAttributeUse attributeUse)` [5.5 – „Zpracování užití atributu“].

1. Dojde k vytvoření symbolu 4.3.3 – „`attribute`“ a jeho připojení do stromové struktury. Předtím se musí zjistit základní informace, poskytované přímo deklarací atributu, a také dodatečná informace o typu atributu, získaná voláním pomocné metody `getTypeString(typeDefinition)` [5.8 – „Zpracování definice jednoduchého typu“]. Popis jednotlivých položek naleznete u grafického návrhu symbolu `attribute`.

[5] [2] http://www.w3.org/TR/xmlschema-1/#cAttributeUse
[6] [2] http://www.w3.org/TR/xmlschema-1/#cAttribute_Declarations

5.7. Zpracování divoké karty[7]

Divoká karta (*Wildcard*, v Xercesu třída `XSWildcard`) umožňuje validaci informačních položek element a atribut, závisející na jmenném prostoru, ale nezávislou na lokálním jméně.

Metoda `processElementWildcard`(`XSWildcard` *`wildcard`*, `String` *`cardinality`*) slouží ke zpracování divoké karty pro element:

1. Vytvoří symbol 4.3.4 – „`any`“ a připojí ho na odpovídající pozici ve stromové struktuře. Pomocná metoda `getNamespaceString`(`XSWildcard` *`wildcard`*) umožní sestavit řetězec, uvádějící omezení jmenných prostorů, parametr `cardinality` dodá informaci o minimálním a maximálním počtu výskytů.

Metoda `processAttributeWildcard`(`XSWildcard` *`wildcard`*) slouží ke zpracování divoké karty pro atribut:

1. Vytvoří symbol 4.3.5 – „`anyAttribute`“ a připojí ho na odpovídající pozici ve stromové struktuře. Pomocná metoda `getNamespaceString`(`XSWildcard` *`wildcard`*) umožní sestavit řetězec, uvádějící omezení jmenných prostorů.

5.8. Zpracování definice jednoduchého typu[8]

Definice jednoduchého typu (*Simple type definition*, v Xercesu třída `XSSimpleTypeDefinition`) umožňuje omezení znakových informačních položek – dětí informačních položek element a atribut.

Pomocná metoda `getTypeString`(`XSTypeDefinition` *`typeDefinition`*) je volána při zpracování deklarací elementů a atributů. U jednoduchého typu je pouze třeba zjistit název, nebo název základního typu a ten zobrazit jako typ elementu / atributu.

1. Pokud je typ pojmenovaný (může být i komplexní), vrátí jméno typu.
2. Pokud je typ anonymní a zároveň je jednoduchý, vrátí jméno základního typu, ze kterého je tento typ odvozen.
3. Jinak vrátí `null`.

5.9. Zpracování definice komplexního typu[9]

Definice komplexního typu (*Complex type definition*, v Xercesu třída `XSComplexTypeDefinition`) umožňuje:

- omezení informačních položek element, přidáním deklarací atributů, určujících výskyt a obsah atributů;
- omezení informačních položek element tak, že musí mít buď prázdný obsah, nebo musí vyhovět specifikovanému obsahu smíšenému, nebo tvořenému pouze elementy; nebo omezuje znakové informační položky tak, aby vyhověly specifikované definici jednoduchého typu;

[7] [2] http://www.w3.org/TR/xmlschema-1/#Wildcards
[8] [2] http://www.w3.org/TR/xmlschema-1/#Simple_Type_Definitions
[9] [2] http://www.w3.org/TR/xmlschema-1/#Complex_Type_Definitions

- využití mechanismu hierarchie definic typů (*Type definition hierarchy*) k odvození komplexního typu z jiného jednoduchého či komplexního typu;
- specifikování příspěvků k post-schema-validation infosetu elementů;
- omezení možnosti odvozovat další typy z tohoto komplexního typu;
- kontrolovat nahrazování elementů odvozeného typu za elementy deklarované v modelu obsahu, který je tohoto komplexního typu.

Definice komplexního typu je zpracována metodou `processComplexTypeDefinition(XSComplexTypeDefinition complexTypeDefinition)`.

1. Zjistí se, zda definice obsahuje částici (*Particle*), pokud ano, volá se metoda `processParticle(particle)` [5.13 – „Zpracování částice“]. Parametrem je částice získaná z definice komplexního typu.
2. Volá se pomocná metoda `processAttributeUses(attributeUses)` [5.4 – „Zpracování kolekce užití atributů“], parametrem je kolekce užití atributů.
3. Pokud je definována divoká karta (*Wildcard*) pro atributy, zavolá se metoda `processAttributeWildcard(wildcard)` [5.7 – „Zpracování divoké karty“]. Parametrem je divoká karta získaná z definice komplexního typu.

5.10. Zpracování kolekce identitních omezení

Kolekce deklarací identitních omezení (v Xercesu obalené obecnou třídou `XSNamedMap`).

Metoda `processIdentityConstraints(XSNamedMap identityConstraints)` provádí zpracování deklarací identitních omezení. Metoda obsahuje cyklus.

1. Prochází se kolekcí:
   a. Pro každý prvek kolekce se volá metoda `processIdentityConstraintDefinition(identityConstraintDefinition)` [5.11 – „Zpracování definice identitního omezení“]. Parametrem je konkrétní definice identitního omezení (*Identity-constraint definition*).

5.11. Zpracování definice identitního omezení[10]

Definice identitního omezení (*Identity-constraint definition*, v Xercesu třída `XSIDCDefinition`) zajišťuje jedinečnost a referenční omezení v rámci hodnot množiny elementů a atributů.

Metoda `processIdentityConstraintDefinition(XSIDCDefinition identityConstraintDefinition)` má na starost zpracování definice identitního omezení.

1. Zjistí se kategorie této definice:
   a. Pokud jde o unikátní klíč (`unique`), vytvoří se symbol 4.3.9 – „`unique`“.
   b. Pokud jde o primární klíč (`key`), vytvoří se symbol 4.3.10 – „`key`“.
   c. Pokud jde o cizí klíč (`keyref`), vytvoří se symbol 4.3.11 – „`keyref`“.
2. Vytvoří se symbol 4.3.12 – „`selector`“.
3. Prochází se kolekcí polí (`field`):
   a. Pro každý prvek kolekce se vytvoří symbol 4.3.13 – „`field`“.

[10] [2] http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions

Potřebné informace pro vykreslení všech symbolů poskytne přímo objekt *identityConstraintDefinition*. Jejich přehled naleznete u návrhu jednotlivých symbolů. Každý ze symbolů je připojen na odpovídající místo stromové struktury.

5.12. Zpracování kolekce částic

Kolekce částic (v Xercesu obalené obecnou třídou `XSObjectList`).

Metoda `processParticles`(`XSObjectList` *particles*) provádí zpracování kolekce částic. Metoda obsahuje cyklus:

1. Prochází se kolekcí částic:
   a. Pro každý prvek kolekce se volá `processParticle`(*particle*) [5.13 – „Zpracování částice“]. Parametrem je konkrétní částice (*Particle*).

5.13. Zpracování částice[11]

Částice (*Particle*, v Xercesu třída `XSParticle`) přidává dodatečné informace o minimálním a maximálním počtu výskytů k definici modelu obsahu.

Metoda `processParticle`(`XSParticle` *particle*) zpracuje částici následujícím způsobem:

1. Pomocná metoda `getCardinalityString`(`XSParticle` *particle*) sestaví řetězec `cardinality`, obsahující informaci o minimálním (*minOccurs*) a maximálním (*maxOccurs*) počtu výskytů částicí obaleného termínu (*term*).
2. Zavolá pomocnou metodu `processTerm`(*term*, *cardinality*) [5.14 – „Zpracování termínu“], která termín zpracuje, a pošle jí i řetězec `cardinality`.

5.14. Zpracování termínu

Termín (*Term*, v Xercesu třída `XSTerm`) je obecná vlastnost zastupující modelovou skupinu, deklaraci elementu, nebo divokou kartu pro elementy.

Metoda `processTerm`(`XSTerm` *term*, `String` *cardinality*) provádí tento algoritmus:

1. Zjistí se typ termínu:
   a. Pokud se jedná o modelovou skupinu (*Model group*), volá se metoda `processModelGroup`(*modelGroup*, *cardinality*) [5.15 – „Zpracování modelové skupiny“]. Prvním parametrem je termín – modelová skupina, druhým parametrem se předává dál informace o kardinalitě.
   b. Pokud jde o deklaraci elementu (*Element declaration*), volá se `processElementDeclaration`(*elementDeclaration*, *cardinality*) [5.3 – „Zpracování deklarace elementu“]. Prvním parametrem je termín – deklarace elementu, druhým parametrem se předává dál informace o kardinalitě.
   c. Pokud jde o divokou kartu (*Wildcard*) pro elementy, volá se metoda `processElementWildcard`(*wildcard*, *cardinality*) [5.7 – „Zpracování divoké karty“]. Prvním parametrem je termín – divoká karta, druhým parametrem se předává dál informace o kardinalitě.

[11] [2] http://www.w3.org/TR/xmlschema-1/#cParticles

5.15. Zpracování modelové skupiny[12]

Modelová skupina (*Model group*, v Xercesu třída `XSModelGroup`). Pokud není definováno, že mají být děti informační položky element prázdné, nebo že mají vyhovovat určité definici jednoduchého typu, může být obsah posloupnosti dětí informační položky element specifikován detailněji pomocí modelové skupiny. Díky tomu, že součástí částice může být modelová skupina a modelová skupina obsahuje částice, může modelová skupina nepřímo obsahovat jiné modelové skupiny.

Metoda `processModelGroup`(`XSModelGroup` *`modelGroup`*, `String` *`cardinality`*) provádí algoritmus:

1. Zjistí se typ kompozitoru (*compositor*):
   a. Pokud se jedná o kompozitor `all`, vytvoří se symbol 4.3.6 – „`all`“.
   b. Pokud jde o kompozitor `choice`, vytvoří se symbol 4.3.7 – „`choice`“.
   c. Jinak jde o kompozitor `sequence`, vytvoří se symbol 4.3.8 – „`sequence`“.
2. Zavolá se pomocná metoda `processParticles`(*`particles`*) [5.12 – „Zpracování kolekce částic“], parametrem je kolekce částic, obsažených uvnitř modelové skupiny.

U každého ze symbolů bude vypsán řetězec `cardinality` s informací o minimálním a maximálním počtu výskytů. Vytvořený symbol je vždy připojen na odpovídající pozici stromové struktury.

Konečné uložení připravené struktury do SVG

Poté, co se dokončí procházení abstraktního datového modelu XML schématu a připravené symboly utvoří stromovou strukturu, dojde k uložení stromu do SVG souboru. Nejprve se vypíše jakási hlavička s XML deklarací, doctypem, titulkem, přidá se skript, předem definované symboly a popřípadě styl; následují jednotlivé symboly komponent. Přesný průběh vypadá tak, že se vezme kořenový symbol, nastaví se u něj správné rozměry a umístění na pomyslném plátně a uloží se, totéž se pak opakuje se všemi dětmi a jejich potomky, až se dojde k listům stromu. Tím jsou symboly vloženy do souboru a následuje už jen koncový tag `<svg>`.

[12] [2] http://www.w3.org/TR/xmlschema-1/#Model_Groups

Kapitola 6

Závěr

Cílem práce bylo navrhnout a implementovat aplikaci pro převod XML schématu do interaktivního diagramu ve formátu SVG. Bylo proto třeba nastudovat principy, na kterých staví specifikace W3C XML schématu. Ta silně odděluje abstraktní datový model od XML reprezentace. Právě abstraktní datový model, jeho komponenty a vlastnosti komponent poskytují přímý a jednoduchý přístup k důležitým údajům. Abych nemusel vše programovat od začátku, vybíral jsem mezi hotovými open source implementacemi modelu a zvolil procesor tvořící součást parseru Xerces. Ten vyhovuje specifikaci; zpřístupnil komponenty schématu a jejich struktury programovým prostředkům jazyka Java.

Dále bylo třeba rozhodnout se pro vhodný způsob zobrazení struktur definovaných schématem. Zde jsem se inspiroval výstupy úspěšného XML editoru oXygen. Z představy grafického modelu vzešel návrh tříd objektů – obecného abstraktního symbolu a konkrétních symbolů komponent schématu –, návrh jejich vzhledu, parametrů a propojení i ve vazbě k zajištění interaktivity modelu. Přitom bylo nutné alespoň v základu poznat možnosti formátu SVG.

Po načtení instance XML schématu do objektové reprezentace jazyka Java se spustí procházení těmito objekty a začne se vytvářet stromová struktura definovaných symbolů. Ta je nakonec uložena do SVG souboru. Na něj jsou ale ještě vázány další technologie. Především jde o ECMAScript, který je součástí souboru a provádí jeho zpracování při otevření v prohlížeči. Zajišťuje interaktivitu reakcemi na události spuštěné pohybem a klikáním myši. Aby skript mohl pracovat s elementy SVG dokumentu, potřebuje k nim získat přístup. Přístup zajišťuje DOM úrovně 2. Poslední technologií jsou kaskádové styly, které mohou tvořit součást SVG souboru nebo mohou být připojeny z externího souboru a umožňují drobné úpravy výsledného vzhledu.

Na závěr mohu konstatovat, že se podařilo splnit cíl práce. Praktickým produktem je funkční program, jehož výstupem je přehledný, interaktivní diagram zobrazující strukturu XML souborů, vyhovujících schématu. Konkrétně je to struktura použitelných elementů a atributů, jejich jména, jmenné prostory, datové typy, kardinalita elementů, definované klíče atd. Postromy symbolů lze zobrazovat a skrývat. Dle mého názoru může diagram skutečně usnadnit orientaci a pochopení struktur, může tvořit kvalitní doplněk dokumentace.

I zde však existují určitá omezení. První vyplývá z pomalé implementce SVG a ECMAScriptu v prohlížečích. Projevuje se v pomalejší reakci na události a obecně v pomalejším vykreslování velkých diagramů. V této oblasti lze jen doufat, že do budoucna vznikne lepší podpora. Další problém se může vyskytnout při programovém zpracování schémat, popisujících velmi rozsáhlé struktury. Může dojít k pádu aplikace v důsledku překročení paměťového prostoru. Tento problém lze odstranit úpravou aplikace, vzhledem k prvnímu omezení však není nutné s úpravou moc spěchat.

Tím se dostávám k faktu, že aplikace včetně zdrojových kódů bude šířena jako svobodný software. Momentálně je dostupná na přiloženém CD (viz B – „*Obsah CD-ROM*“) a na webové stránce http://st.vse.cz/~XSLAV14/. To přináší možnost budoucího rozvoje aplikace v podobě vylepšování nebo rozšiřování funkčnosti. Například by šla přidělat tlačítka pro zoom in a zoom out; rozšířit program o podporu dalších schémových jazyků (pokud vím, je jejich vizualizace předmětem jiných bakalářských prací); nebo integrovat do systému pro generování kompletních dokumentací.

Literatura

Hlavní

[1] Fallside, D. C., Walmsley, P. a kol.: *XML Schema Part 0: Primer Second Edition* [online]. W3C, 2004. Dostupný z WWW: http://www.w3.org/TR/xmlschema-0/

[2] Thompson, H. S., Beech, D., Maloney, M., Mendelsohn, N. a kol.: *XML Schema Part 1: Structures Second Edition* [online]. W3C, 2004. Dostupný z WWW: http://www.w3.org/TR/xmlschema-1/

[3] Biron, P. V., Malhotra, A. a kol.: *XML Schema Part 2: Datatypes Second Edition* [online]. W3C, 2004. Dostupný z WWW: http://www.w3.org/TR/xmlschema-2/

[4] van der Vlist, E.: *Using W3C XML Schema*. XML.com, 2001. Dostupný z WWW: http://www.xml.com/pub/a/2000/11/29/schemas/part1.html?page=1

[5] Ferraiolo, J., Fujisawa, J., Jackson, D.: *Scalable Vector Graphics (SVG) 1.1 Specification* [online]. W3C, 2003. Dostupný z WWW: http://www.w3.org/TR/SVG/

Vedlejší a doplňková

[6] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F. a kol.: *Extensible Markup Language (XML) 1.0 (Fourth Edition)* [online]. W3C, 2006. Dostupný z WWW: http://www.w3.org/TR/REC-xml/

[7] Cowan, J., Tobin, R.: *XML Information Set (Second Edition)* [online]. W3C, 2004. Dostupný z WWW: http://www.w3.org/TR/xml-infoset/

[8] Bray, T., Hollander, D., Layman, A., Tobin, R.: *Namespaces in XML 1.0 (Second Edition)* [online]. W3C, 2006. Dostupný z WWW: http://www.w3.org/TR/REC-xml-names/

[9] Wood, L. a kol.: *Document Object Model (DOM) Level 1 Specification* [online]. W3C, 1998. Dostupný z WWW: http://www.w3.org/TR/REC-DOM-Level-1/

[10] Wood, L. a kol.: *Document Object Model (DOM) Level 2 Core Specification* [online]. W3C, 2000. Dostupný z WWW: http://www.w3.org/TR/DOM-Level-2-Core/

[11] Lie, H. W., Bos, B.: *Cascading Style Sheets, level 1* [online]. W3C, 1996. Dostupný z WWW: http://www.w3.org/TR/CSS1/

[12] Hall, W., Keynes, M.: *M254 Java everywhere* [online]. The Open University, 2005. Dostupný z WWW: http://computing.open.ac.uk/m254/

Příloha A

Termíny

API	*Application Programming Interface*, rozhraní aplikačních programů. Sada procedur, funkcí nebo tříd určité knihovny, programu nebo jádra operačního systému, která může být využita v aplikacích.
CSS	*Cascading Style Sheets*, kaskádové styly. Jednoduchý mechanismus pro přidání grafické úpravy webovým dokumentům. Standard organizace W3C mimo jiné v [11].
Data binding	Převod dat mezi různými formáty. Například mezi XML dokumenty, objekty objektově orientovaných systémů a relačními tabulkami databázových systémů.
DOM	*Document Object Model*, objektový model dokumentu. Rozhraní nezávislé na platformě a jazyku, které umožňuje programům a skriptům dynamický přístup k obsahu, struktuře a stylu dokumentu a jeho editaci. Poskytuje objektově orientovanou reprezentaci XML nebo HTML dokumentu. Specifikace W3C v [9], [10] a dalších.
DTD	*Document Type Definition*, definice typu dokumentu. DTD poskytuje nástroje pro definování povolených struktur elementů a atributů, umožňuje stanovení implicitních hodnot atributů. V rámci DTD lze také definovat znovupoužitelný obsah (entity) a dodatečné informace (notace). Specifikace je součástí doporučení organizace W3C o XML – [6] a další.
ECMAScript	Skriptovací (programovací) jazyk, standardizován organizací *Ecma International* ve specifikaci *ECMA-262*. Je široce používán na webu a bývá často označován jako JavaScript nebo JScript, podle hlavních dialektů tohoto jazyka.
Framework	Základní konceptuální struktura užívaná k řešení nebo zaměření se na komplexní problémy. V oblasti softwaru se často jedná o knihovny, podpůrné programy, návrhové vzory a doporučené postupy. Framework řeší typické úlohy, které jsou ve většině aplikací shodné, a usnadňuje tak vývoj.
HTML	*HyperText Markup Language*, hypertextový značkovací jazyk. Jazyk pro tvorbu webových stránek, aplikace SGML. Specifikaci vydala organizace W3C.

Informační položka	*Information item*. Představuje uzel stromové struktury XML infosetu. Specifikace [7] definuje jedenáct různých typů položek. Každá má určité vlastnosti jako je rodičovský element, lokální jméno atd. V této práci se zmiňuji o znakové informační položce, informační položce element a atribut.
Infoset	*XML Information Set* je abstraktní datový model XML dokumentu, skládá se z jednotlivých informačních položek. Specifikace W3C [7].
Java	Objektově orientovaný, silně typový, na platformě nezávislý programovací jazyk od společnosti *Sun Microsystems*.
Namespace	Jmenný prostor. XML poskytuje jednoduchou metodu pro kvalifikování jmen elementů a atributů jejich zařazením do jmenného prostoru s URI identifikátorem. Specifikace W3C [8].
Parsování	Analýza předložených dat a vyhledávání prvků odpovídajících definici určitého jazyka. Obvykle jde o syntaktickou analýzu zdrojových kódů napsaných v daném programovacím jazyce.
PSVI	*Post Schema Validation Infoset*, infoset po validaci dokumentu oproti schématu. Je to infoset XML dokumentu rozšířený o informace o datových typech položek. Příspěvky k PSVI jsou definovány ve specifikaci XML schématu [2] a [3].
Renderování	*Rendering*, proces, při němž se vykresluje obraz ze zadaných dat.
SAX	*Simple API for XML*, jednoduché aplikační rozhraní pro XML. Poskytuje mechanismus pro čtení dat z XML dokumentů. Čtení probíhá sekvenčně a je založeno na událostech (přečtení počátečního tagu, přečtení ukončujícího tagu atd.) Není proto možná modifikace struktury dokumentu, na rozdíl od DOM.
SVG	*Scalable Vector Graphics*, škálovatelná vektorová grafika. SVG je jazyk pro popis dvourozměrné grafiky a grafických aplikací pomocí XML. Základ vývoje tvoří specifikace SVG 1.1, vydaná jako doporučení konsorcia W3C 14. ledna 2003 [5].
Tag	*Značka* vymezující začátek a/nebo konec elementu.
Validace	Ověření shody XML dokumentu se schématem.
W3C	*World Wide Web Consortium*. Zabývá se vývojem interoperabilních technologií – specifikací, směrnic, softwaru a nástrojů – s cílem sjednotit používané technologie a dosáhnout plného rozvoje a využití možností webu.
XHTML	*Extensible HyperText Markup Language*, rozšiřitelný hypertextový značkovací jazyk. Jazyk pro tvorbu webových stránek, aplikace XML. Specifikaci vydala organizace W3C.
XML	*Extensible Markup Language*, rozšiřitelný značkovací jazyk. Umožňuje vývojářům vytvářet vlastní formáty pro uchování a sdílení dat. Specifikaci vydalo konsorcium W3C, [6] a další.
XML schéma	XML schémata poskytují prostředky pro definování struktury, obsahu a sémantiky XML dokumentů. XML schéma bylo schváleno jako doporučení organizace W3C 2. května 2001 a druhá, upravená verze byla vydána 28. října 2004 [1], [2], [3].

Příloha B

Obsah CD-ROM

Součástí této práce je přiložený CD-ROM s textem práce, zdrojovými kódy a hotovou aplikací. Poslední verze bude přístupná na webové stránce: http://st.vse.cz/~XSLAV14/. Konkrétně na CD naleznete následující strukturu adresářů a souborů:

1. `/thesis` – adresář týkající se psaní bakalářské práce, obsahuje:
   - `/images` – složka použitých obrázků kromě symbolů komponent schématu
   - `/symbols` – složka s obrázky symbolů komponent schématu ve formátu SVG
   - `thesis.xml` – vlastní text práce ve formátu DocBook
   - `thesis.xsl`, `tp-fo.xsl` – styly upravující standardní výstup DocBooku
   - `kizi.pdf` – logo katedry pro titulní stranu[1]
   - `thesis.pdf` – vygenerované PDF s touto prací

2. `/src` – adresář zdrojových kódů Javy, struktura vychází ze zařazení tříd do balíčků

3. `/res` – adresář dalších zdrojů obsahující:
   - `/examples` – složka s příklady XSD vstupů
   - `/licenses` – složka s licencemi parseru Xerces
   - `readme.txt` – textový dokument, popisuje uživatelské rozhraní

4. `/lib` – adresář obsahuje implementaci parseru Xerces – `xercesImpl.jar`

5. `/dist` – složka s aplikací připravenou k použití, obsahuje:
   - `/examples` – překopírováno z adresáře `/res`
   - `/licenses` – překopírováno z adresáře `/res`
   - `/lib` – adresář knihoven:
     a. kopie souboru `xercesImpl.jar` a případných dalších z adresáře `/lib`
     b. archiv s vlastní aplikací `xsdvi.jar`
   - `examples.bat` – spouští transformaci příkladů, schémat ze složky examples
   - `readme.txt` – překopírováno z adresáře `/res`

6. `build.properties` – nastavení vlastností využívaných v `build.xml`

7. `build.xml` – XML skript pro sestavovací program Ant

[1] Upravené styly a logo katedry pochází ze stránek Jiřího Koska, věnovaných výuce na VŠE: http://www.kosek.cz/vyuka/.

Příloha C

Příklad

C.1. XML schéma – ukázkový vstup

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="contact" type="ContactType"/>

    <xs:complexType name="ContactType">
        <xs:sequence>
            <xs:element name="phone" type="xs:string"/>
            <xs:element name="address" type="AddressType"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="AddressType">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="street" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

C.2. SVG výstup (XML reprezentace)

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE svg PUBLIC '-//W3C//DTD SVG 1.1//EN'
  'http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd'>
<svg id='svg' onload='loadSVG();'
  xmlns='http://www.w3.org/2000/svg' xmlns:xlink='http://www.w3.org/1999/xlink'>
<title>XsdVi</title>

<script type='text/ecmascript'><![CDATA[
  var efBoxes = [];
  var eSvg = null;
```

```
function loadSVG() {
  efBoxes = getElementsByClassName('box', document.getElementsByTagName('g'));
  eSvg = document.getElementById('svg');
  expandAll();
}

function getElementsByClassName(sClass, nlNodes) {
  var elements = [];
  for (var i=0; i<nlNodes.length; i++) {
    if(nlNodes.item(i).nodeType==1
      && sClass==nlNodes.item(i).getAttribute('class')) {
      elements.push(nlNodes.item(i));
    }
  }
  return elements;
}

function show(sId) {
  var useElement = document.getElementById('s'+sId);
  var moveNext = false;
  var eBoxLast;
  var maxX = 500;

  if (notPlus(useElement)) {
    eBoxLast = document.getElementById(sId);
    setPlus(useElement);
    for (var i=0; i<efBoxes.length; i++) {
      var eBox = efBoxes[i];
      if (moveNext) {
        move(eBoxLast, eBox);
      }
      else if (isDescendant(sId, eBox.id)) {
        eBox.setAttribute('visibility', 'hidden');
      }
      else if (isHigherBranch(sId, eBox.id)) {
        move(eBoxLast, eBox);
        moveNext = true;
      }
      if (eBox.getAttribute('visibility') != 'hidden') {
        eBoxLast = eBox;
        x = xTrans(eBox);
        if (x > maxX) maxX = x;
      }
    }
  }
  else {
    setMinus(useElement);
    var skipDescendantsOf;
    for (var i=0; i<efBoxes.length; i++) {
      var eBox = efBoxes[i];
      if (moveNext) {
```

```
        move(eBoxLast, eBox);
      }
      else if (isDescendant(sId, eBox.id) && (!skipDescendantsOf
        || !isDescendant(skipDescendantsOf.id, eBox.id))) {
        eBox.setAttribute('visibility', 'visible');
        move(eBoxLast, eBox);
        if (nextClosed(eBox)) skipDescendantsOf = eBox;
      }
      else if (isHigherBranch(sId, eBox.id)) {
        move(eBoxLast, eBox);
        moveNext = true;
      }
      if (eBox.getAttribute('visibility') != 'hidden') {
        eBoxLast = eBox;
        x = xTrans(eBox);
        if (x > maxX) maxX = x;
      }
    }
  }
  setHeight(yTrans(eBoxLast)+71);
  setWidth(maxX+300);
}

function collapseAll() {
  for (var i=0; i<efBoxes.length; i++) {
    var eBox = efBoxes[i];
    var useElement = document.getElementById('s'+eBox.id);
    if (useElement) setPlus(useElement);
    if (eBox.id != '_1') eBox.setAttribute('visibility', 'hidden');
  }
  setHeight(400);
  setWidth(500);
}

function expandAll() {
  var eBoxLast;
  var maxX = 0;
  for (var i=0; i<efBoxes.length; i++) {
    var eBox = efBoxes[i];
    var useElement = document.getElementById('s'+eBox.id);
    if (useElement) setMinus(useElement);
    move(eBoxLast, eBox);
    eBox.setAttribute('visibility', 'visible');
    eBoxLast = eBox;
    var x = xTrans(eBox);
    if (x > maxX) maxX = x;
  }
  setHeight(yTrans(eBoxLast)+71);
  setWidth(maxX+300);
}
```

```
function makeVisible(sId) {
  var childNodes = document.getElementById(sId).childNodes;
  var hidden = getElementsByClassName('hidden', childNodes);
  var visible = getElementsByClassName('visible', childNodes);
  inheritVisibility(hidden);
  hiddenVisibility(visible);
}

function makeHidden(sId) {
  var childNodes = document.getElementById(sId).childNodes;
  var hidden = getElementsByClassName('hidden', childNodes);
  var visible = getElementsByClassName('visible', childNodes);
  inheritVisibility(visible);
  hiddenVisibility(hidden);
}

function inheritVisibility(efElements) {
  for (var i=0; i<efElements.length; i++) {
    efElements[i].setAttribute('visibility', 'inherit');
  }
}

function hiddenVisibility(efElements) {
  for (var i=0; i<efElements.length; i++) {
    efElements[i].setAttribute('visibility', 'hidden');
  }
}

function nextClosed(eBox) {
  var useElement = document.getElementById('s'+eBox.id);
  return (useElement && !notPlus(useElement));
}

function isHigherBranch(sSerialLower, sSerialHigher) {
  var sLower = sSerialLower.split('_');
  var sHigher = sSerialHigher.split('_');
  for (var i=0; i<sLower.length; i++) {
    if (Number(sHigher[i]) > Number(sLower[i])) return true;
    else if (Number(sHigher[i]) < Number(sLower[i])) return false;
  }
  return false;
}

function isOnHigherLevel(eBoxLower, eBoxHigher) {
  var sLower = eBoxLower.id.split('_');
  var sHigher = eBoxHigher.id.split('_');
  for (var i=0; i<sLower.length; i++) {
    if (Number(sHigher[i]) > Number(sLower[i])) return true;
  }
  return false;
}
```

```
function isDescendant(sSerialAsc, sSerialDesc) {
  return (sSerialDesc.length > sSerialAsc.length &&
    sSerialDesc.indexOf(sSerialAsc) === 0);
}

function getParent(eBox) {
  var serial = eBox.id.substring(0, eBox.id.lastIndexOf('_'));
  return document.getElementById(serial);
}

function move(eBoxLast, eBox) {
  if (!eBoxLast) return;
  if (isOnHigherLevel(eBoxLast, eBox)) {
    eBox.setAttribute('transform',
      'translate('+xTrans(eBox)+','+(yTrans(eBoxLast)+71)+')');
    var parent = getParent(eBox);
    var line = document.getElementById('p'+eBox.id);
    if (!parent || !line) return;
    line.setAttribute('y1', String(yTrans(parent)-yTrans(eBox)+23));
  }
  else {
    eBox.setAttribute('transform',
      'translate('+xTrans(eBox)+','+yTrans(eBoxLast)+')');
  }
}

function notPlus(eUseElement) {
  return (eUseElement.getAttributeNS('http://www.w3.org/1999/xlink',
    'href') != '#plus');
}

  eUseElement.setAttributeNS('http://www.w3.org/1999/xlink',
    'href', '#plus');
}

function setMinus(eUseElement) {
  eUseElement.setAttributeNS('http://www.w3.org/1999/xlink',
    'href', '#minus');
}

function setHeight(nHeight) {
  eSvg.setAttribute('height', nHeight);
}

function setWidth(nWidth) {
  eSvg.setAttribute('width', nWidth);
}

function yTrans(eBox) {
  var transform = eBox.getAttribute('transform');
```

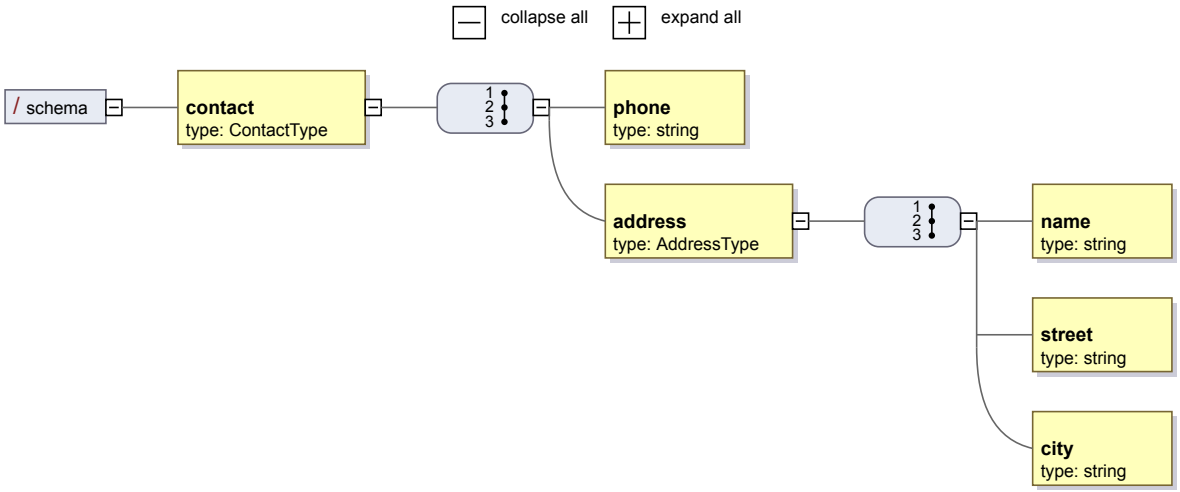
```
    var y = Number(transform.substring(10,
      Number(transform.length)-1).split(',')[1]);
    if(!y) y = 0;
    return y;
  }

  function xTrans(eBox) {
    var transform = eBox.getAttribute('transform');
    var x = Number(transform.substring(10,
      Number(transform.length)-1).split(',')[0]);
    if(!x) x = 0;
    return x;
  }
]]></script>

<defs>

<style type='text/css'><![CDATA[

svg {pointer-events: none;}
text {font-family: arial; font-size: 11px;}
line, polyline, polygon {fill: none; stroke: black;}

.strong {font-size: 12px; font-weight: bold;}
.small {font-size: 10px;}
.big {font-size: 15px; fill: #882222;}

.button {fill: white; stroke: black; pointer-events: all;}
.shadow {fill: #ccccd8; stroke: none;}
.connection {fill: none; stroke: #666666;}
.empty {fill: none; stroke: black;}
.filled {fill: black; stroke: none;}

.boxelement, .boxany, .boxattribute1, .boxanyattribute
  {fill: #FFFFBB; stroke: #776633; pointer-events: all;}
.boxattribute2
  {fill: #FFFFBB; stroke: #776633; pointer-events: all; stroke-dasharray: 2;}
.boxschema, .boxloop, .boxcompositor {fill: #E7EBF3; stroke: #666677;}
.boxselector, .boxfield, .boxidc {fill: #E0F7B7; stroke: #667733;}

.lax {fill: white; stroke: black;}
.skip {fill: #cc6666; stroke: black;}
.strict {fill: black; stroke: none;}

]]></style>

  <symbol class='button' id='plus'>
    <rect x='1' y='1' width='10' height='10'/>
    <line x1='3' y1='6' x2='9' y2='6'/>
    <line x1='6' y1='3' x2='6' y2='9'/>
  </symbol>
```

```
  <symbol class='button' id='minus'>
    <rect x='1' y='1' width='10' height='10'/>
    <line x1='3' y1='6' x2='9' y2='6'/>
  </symbol>

</defs>

<rect class='button' x='300' y='10' width='20' height='20'
  onclick='collapseAll()'/>
<line x1='303' y1='20' x2='317' y2='20'/>
<text x='330' y='20'>collapse all</text>
<rect class='button' x='400' y='10' width='20' height='20'
  onclick='expandAll()'/>
<line x1='403' y1='20' x2='417' y2='20'/>
<line x1='410' y1='13' x2='410' y2='27'/>
<text x='430' y='20'>expand all</text>

<g id='_1' class='box' transform='translate(20,50)'>
<rect class='boxschema' x='0' y='12' width='63' height='21'/>
<text x='5' y='27'><tspan class='big'>/ </tspan>schema</text>
<use x='62' y='17' xlink:href='#minus' id='s_1' onclick='show("_1")'/>
</g>

<g id='_1_1' class='box' transform='translate(128,50)'>
<rect class='shadow' x='3' y='3' width='117' height='46'/>
<rect class='boxelement' x='0' y='0' width='117' height='46'
  onmouseover='makeVisible("_1_1")' onmouseout='makeHidden("_1_1")'/>
<text class='hidden' visibility='hidden' x='5' y='13'>nillable: 0</text>
<text class='hidden' visibility='hidden' x='5' y='41'>abstract: 0</text>
<text class='strong' x='5' y='27'>contact</text>
<text class='visible' x='5' y='41'>type: ContactType</text>
<line class='connection' x1='-35' y1='23' x2='0' y2='23'/>
<use x='116' y='17' xlink:href='#minus' id='s_1_1' onclick='show("_1_1")'/>
</g>

<g id='_1_1_1' class='box' transform='translate(290,50)'>
<rect class='boxcompositor' x='0' y='8' width='60' height='31' rx='9'/>
<circle cx='42' cy='14' r='2'/>
<circle cx='42' cy='23' r='2'/>
<circle cx='42' cy='32' r='2'/>
<text class='small' x='30' y='17'>1</text>
<text class='small' x='30' y='26'>2</text>
<text class='small' x='30' y='35'>3</text>
<line x1='42' y1='14' x2='42' y2='32'/>
<line class='connection' x1='-35' y1='23' x2='0' y2='23'/>
<use x='59' y='17' xlink:href='#minus' id='s_1_1_1'
  onclick='show("_1_1_1")'/>
</g>

<g id='_1_1_1_1' class='box' transform='translate(395,50)'>
<rect class='shadow' x='3' y='3' width='87' height='46'/>
```

```
<rect class='boxelement' x='0' y='0' width='87' height='46'
  onmouseover='makeVisible("_1_1_1_1")'
  onmouseout='makeHidden("_1_1_1_1")'/>
<text class='hidden' visibility='hidden' x='5' y='13'>nillable: 0</text>
<text class='hidden' visibility='hidden' x='5' y='41'>abstract: 0</text>
<text class='strong' x='5' y='27'>phone</text>
<text class='visible' x='5' y='41'>type: string</text>
<line class='connection' x1='-35' y1='23' x2='0' y2='23'/>
</g>

<g id='_1_1_1_2' class='box' transform='translate(395,121)'>
<rect class='shadow' x='3' y='3' width='117' height='46'/>
<rect class='boxelement' x='0' y='0' width='117' height='46'
  onmouseover='makeVisible("_1_1_1_2")'
  onmouseout='makeHidden("_1_1_1_2")'/>
<text class='hidden' visibility='hidden' x='5' y='13'>nillable: 0</text>
<text class='hidden' visibility='hidden' x='5' y='41'>abstract: 0</text>
<text class='strong' x='5' y='27'>address</text>
<text class='visible' x='5' y='41'>type: AddressType</text>
<line class='connection' id='p_1_1_1_2' x1='-35' y1='-48' x2='-35' y2='-40'/>
<path class='connection' d='M-35,-40 Q-35,15 0,23'/>
<use x='116' y='17' xlink:href='#minus' id='s_1_1_1_2'
  onclick='show("_1_1_1_2")'/>
</g>

<g id='_1_1_1_2_1' class='box' transform='translate(557,121)'>
<rect class='boxcompositor' x='0' y='8' width='60' height='31' rx='9'/>
<circle cx='42' cy='14' r='2'/>
<circle cx='42' cy='23' r='2'/>
<circle cx='42' cy='32' r='2'/>
<text class='small' x='30' y='17'>1</text>
<text class='small' x='30' y='26'>2</text>
<text class='small' x='30' y='35'>3</text>
<line x1='42' y1='14' x2='42' y2='32'/>
<line class='connection' x1='-35' y1='23' x2='0' y2='23'/>
<use x='59' y='17' xlink:href='#minus' id='s_1_1_1_2_1'
  onclick='show("_1_1_1_2_1")'/>
</g>

<g id='_1_1_1_2_1_1' class='box' transform='translate(662,121)'>
<rect class='shadow' x='3' y='3' width='87' height='46'/>
<rect class='boxelement' x='0' y='0' width='87' height='46'
  onmouseover='makeVisible("_1_1_1_2_1_1")'
  onmouseout='makeHidden("_1_1_1_2_1_1")'/>
<text class='hidden' visibility='hidden' x='5' y='13'>nillable: 0</text>
<text class='hidden' visibility='hidden' x='5' y='41'>abstract: 0</text>
<text class='strong' x='5' y='27'>name</text>
<text class='visible' x='5' y='41'>type: string</text>
<line class='connection' x1='-35' y1='23' x2='0' y2='23'/>
</g>
```

```
<g id='_1_1_1_2_1_2' class='box' transform='translate(662,192)'>
<rect class='shadow' x='3' y='3' width='87' height='46'/>
<rect class='boxelement' x='0' y='0' width='87' height='46'
  onmouseover='makeVisible("_1_1_1_2_1_2")'
  onmouseout='makeHidden("_1_1_1_2_1_2")'/>
<text class='hidden' visibility='hidden' x='5' y='13'>nillable: 0</text>
<text class='hidden' visibility='hidden' x='5' y='41'>abstract: 0</text>
<text class='strong' x='5' y='27'>street</text>
<text class='visible' x='5' y='41'>type: string</text>
<line class='connection' x1='-35' y1='23' x2='0' y2='23'/>
</g>

<g id='_1_1_1_2_1_3' class='box' transform='translate(662,263)'>
<rect class='shadow' x='3' y='3' width='87' height='46'/>
<rect class='boxelement' x='0' y='0' width='87' height='46'
  onmouseover='makeVisible("_1_1_1_2_1_3")'
  onmouseout='makeHidden("_1_1_1_2_1_3")'/>
<text class='hidden' visibility='hidden' x='5' y='13'>nillable: 0</text>
<text class='hidden' visibility='hidden' x='5' y='41'>abstract: 0</text>
<text class='strong' x='5' y='27'>city</text>
<text class='visible' x='5' y='41'>type: string</text>
<line class='connection' id='p_1_1_1_2_1_3'
  x1='-35' y1='-119' x2='-35' y2='-40'/>
<path class='connection' d='M-35,-40 Q-35,15 0,23'/>
</g>

</svg>
```

C.3. SVG výstup (grafická reprezentace)



Obrázek C.1. Grafický výstup