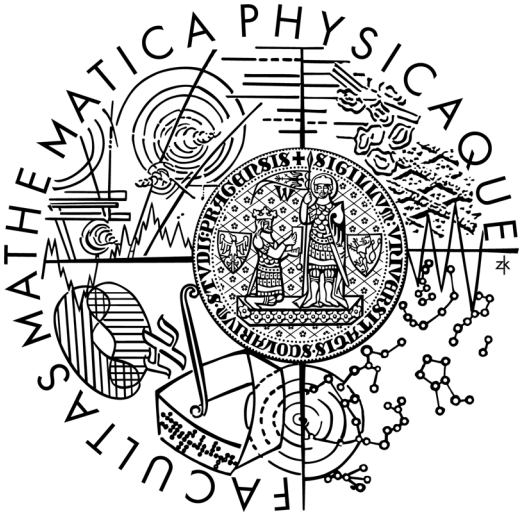
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Lukáš Berka

Generátor dokumentace databázových schémat

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Michal Kopecký, Ph.D.

Studijní program: informatika, obor programování

2007

Na tomto místě chci poděkovat RNDr. Michalu Kopeckému, Ph.D. za vedení, četné konzultace a další odbornou pomoc při tvorbě bakalářského projektu. Rovněž chci poděkovat své rodině a přátelům za morální podporu a trpělivost.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.
V Praze dne 27. května 2007

Lukáš Berka

Obsah

Název práce: Generátor dokumentace databázových schémat
Autor: Lukáš Berka
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Michal Kopecký, Ph.D.
E-mail vedoucího: *Michal.Kopecky@mff.cuni.cz*
Abstrakt: Práce popisuje návrh a implementaci generátoru uživatelské dokumentace databázového schématu. S pomocí aplikace Schemagic, která načítá informace o schématu do strojově čitelného XML souboru, vytváří přehlednou dokumentaci v uživatelem zvoleném výstupním formátu. Jsou podporovány výstupní formáty HTML, PDF, Postscript, RTF, DocBook a FO (XML soubor formátovacích objektů). Aplikace Gendok je rovněž schopna generovat grafické informace jako například grafy závislostí mezi tabulkami.
Klíčová slova: generátor, databázové schéma, DocBook, XML, XSLT

Title: Database Schema Documentation Generator
Author: Lukáš Berka
Department: Department of Software Engineering
Supervisor: RNDr. Michal Kopecký, Ph.D.
Supervisor's e-mail address: *Michal.Kopecky@mff.cuni.cz*
Abstract: The bachelor thesis discribes design and implementation of database schema documentation generator. It uses Schemagic application to create well-arranged documentation in user-defined output format. Schemagic is used to load information about schema into a XML document. For output documentation HTML, PDF, Postscript, RTF, DocBook and FO (XML file of formating objects) formats are supported. The Gendok application also generates some graphic information, e.g. graphs of dependencies between tables.
Keywords: generator, database schema, docbook, XML, XSLT

Úvod

S databázemi se v dnešním světě setkáváme v celé řadě běžných denních situací. Pokud zrovna není informatika naším profesním či studijním oborem, často ani netušíme jak moc v současti náš život na databázích závisí. A ať už si to přejeme nebo ne, tato závislost se neustále zvyšuje. Smyslem projektu, kterému se věnuje tento úvodník, však není nějaká forma pomoci lidem vypořádat se s touto skutečností. Přesto se jeho cíl má dotknout jejich životů.

Mým osobním přáním je, aby projekt přispěl k tomu, že se odborníci v databázích lépe zorientují, lépe jim porozumí, a snáze a rychleji tak budou moci podpořit jejich vývoj, optimalizace nebo odhalit a odstranit chyby a nedostatky v nich už zahrnuté. Tím se mohou stát databázová schémata bezchybnějšími - kvalitnějšími, bezpečnějšími a tedy vhodnějšími pro nasazení do interakce s našimi všedními povinnostmi a potřebami.

Uvažme práci lékaře. Návštěvy klinik se staly běžnou součástí lidských životů, ať už bývají motivovány jakkoliv. Může se jednat o preventivní prohlídky, ale také nemusí. Lidské tělo je ohromně složitý „stroj“, skládá se z celé řady nepostradatelných součástí, a každá z nich nese o naší osobě jedinečné informace.

Aby mohl lékař správně pochopit, jak lidské tělo funguje a co přesně se v něm odehrává, musí k tomu být nejprve vhodně vzdělán. Během svého studia absolvuje student lékařské fakulty ohromný počet přednášek a praktických cvičení, přečte velké množství literatury o jednotlivých částech našeho těla a v neposlední řadě musí složit všechny vyžadované zkoušky. Teprve potom se o něm může říct, že lidskému tělu (nebo jeho součásti) dobře rozumí.

Tedy velmi hrubě řečeno, každý lékař musí před výkonem své profese nastudovat rozsáhlou kvalitní dokumentaci o lidském těle a úspěšně absolvovat patřičná zaškolení, jak s ním zacházet. Obě zmíněné části – dokumentace a školení – pak hrají významné role nejen co se týká studia medicíny, ale prakticky kdykoliv se člověk snaží přijít něčemu takříkajíc „na kloub“, dobře dané věci porozumět a korektně se ji naučit využívat.

Zatímco zaškolení bývá aktem jednorázovým a svým rozsahem omezeným tak, aby školitel nevyslovil více informací, než může být školený schopen si pamatovat, dokumentace by naproti tomu měla být literaturou opakovaně přístupnou a přehlednou tak, aby v ní člověk kdykoliv dohledal cokoliv, co ho o produktu aktuálně zajímá.

Dokumentace jsou tudíž zapotřebí i pro uživatele a vývojáře databázových schémat. Samozřejmě nemám ambice srovnávat složitost databází se složitostí

lidských těl, ale přesto tato analogie může pomoci ujasnit si fakt, proč jsou dokumentace jako prostředky osvětlující vnitřní stavy věcí tolik důležité.

1.1 Cíl projektu

Cílem bakalářského projektu je navrhnout a implementovat generátor dokumentace databázových schémat. Dokumentací databázového schématu můžeme rozumět kvalitní a přehledný popis databázového schématu, tj. všech významných objektů v něm obsažených a jejich vzájemných vztahů. Aplikace by měla být schopna připravit výslednou dokumentaci tak, aby ji bylo možné vytisknout, měla by tedy podporovat výstup do některého z formátů, primárně určených pro tištěný výstup. Zároveň by měla být schopna vygenerovat výstup vhodný pro prezentaci stavu databázového schématu prostřednictvím Internetu a také výstup, jehož obsah bude snadno upravitelný, následně rozšiřitelný o komentáře, vysvětlivky apod.

Generátor dokumentace by měl být schopen vydat výstup nejen ve formě textu, ale mohl by také obsahovat například grafy vazeb mezi některými objekty. Aplikace by měla být široce nastavitelná, měla by uživateli umožnit vybrat si požadovaný výstup podle aktuální potřeby. Cílem projektu není implementovat grafické uživatelské rozhraní (GUI), které by ostatně bylo jen obálkou nad funkční aplikací. Pro účely, pro něž má být generátor určen, je postačující jeho implementace jako konzolové aplikace, neboť se očekává že počet volání aplikace mnohonásobně převýší počet jejího nastavování. GUI by přinášelo velkou výhodu pouze v případě, že by ke změnám parametrů aplikace docházelo často a ve významné míře.

Výsledný generátor by měl svou funkcí přímo navazovat na již existující aplikaci Schemagic[1] [1], která zjišťuje a ukládá stav databázového schématu do souboru o formátu XML. Tento strojově čitelný XML soubor bude tedy vstupem vyvíjeného generátoru dokumentace v podobě snadno čitelné pro člověka. Výsledná aplikace by měla být nezávislá na platformě, měla by fungovat minimálně pod operačními systémy Windows a Linux. Stejně jako v případě aplikace Schemagic by měla být dostupná jako open-source aplikace pod licencí GNU/GPL2. Vícejazyčnost není přímo požadována, ale bylo by vhodné přidat aplikaci kromě angličtiny i podporu českého jazyka.

[1] http://sourceforge.net/projects/schemagic

1.2 Motivace

Motivací pro návrh a implementaci generátoru dokumentace databázových schémat byla nedostupnost nebo případně nevhodnost freeware aplikací, které by generování dokumentace zajišťovaly. Existující generátory dokumentací k databázovým schématům buď nejsou open-source, jsou nedokončené, generují neúplné informace nebo nepodporují možnost tvorby více druhů výstupních formátů.

Aplikace, kterou chceme vyvinout pro účely tvorby dokumentace databázového schématu, by měla být co nejvíce (a pokud možno úplně) založena na freeware technologiích, neboť projekt má za cíl stát se dostupným široké veřejnosti bez jakéhokoliv finančního zatížení jeho uživatelů.

1.3 Struktura práce

Práce je členěna do kapitol, řazených podle významu. V kapitole 0 Analýza, jsou analyzovány jednotlivé aspekty návrhu a požadované vlastnosti implementace porovnány s existujícími generátory dokumentací.

Třetí kapitola knihy nese název Design a obsahuje popis postupů použitých k dosažení cílů projektu, stanovených během analýzy. Kapitola rovněž dává přehled o pomocných technologiích a aplikacích, kterých vyvíjený generátor dokumentace využívá.

Kapitola 0 Implementace osvětluje skutečnou implementaci postupů, navržených ve fázi analýzy a designu. Přehledně popisuje užité algoritmy a vnitřní i vnější podoby jednotlivých součástí aplikace.

Pátá kapitola práce je věnována instalaci a konfiguraci výsledné aplikace nese. Jejím smyslem je poskytnout uživateli maximální množinu informací potřebných ke správnému nastavení a úspěšnému zprovoznění generátoru dokumentace. Věnuje se také instalaci a konfiguraci jednotlivých pomocných programů, jejichž přítomnost je pro funkčnost generátoru nezbytná.

Kapitola 0 Uživatelská příručka definuje postupy, jak generátor dokumentace databázových schémat správně využívat a jak mu sdělit, aby se choval přesně tak, jak uživatel právě požaduje.

Shrnutí výsledků práce a náměty na další možná rozšíření jsou uvedeny v závěrečné kapitole.

Obsah CD, které je součástí práce, je možné nálezt v příloze A .

Analýza

Před přikročením k samotnému vývoji aplikace bylo třeba detailně analyzovat jednotlivé aspekty návrhu a implementace generátorů dokumentací. Bylo nutné si uvědomit, v čem konkrétně jednotlivé existující aplikace nevyhovují našim požadavkům a vytvořit tak seznam vlastností, které by naopak vyvíjená aplikace měla ve finální verzi poskytovat. Z následného rozboru, jak vlastností dosáhnout, vyplývají problémy, které je třeba vyřešit.

1.4 Formáty výstupu

První významnou otázkou, na níž se musela najít odpověď, bylo jaké konkrétní výstupní formáty by měly být generovány. V části 1.1 jsme si dali za cíl podporu minimálně tří různých typů výstupu: výstup pro tisk, pro webovou prezentaci a formát, editovatelný v textovém editoru.

1.4.1 Formát pro tisk

Patrně nejrozšířenějším formátem pro tisk je dnes produkt společnosti Adobe, Portable Document Format, známý pod zkratkou PDF. Jak na svých stránkách uvádí sama společnost Adobe, *„PDF je formátem, vzniklým před více než patnácti lety,… je funkční pro libovolnou aplikaci na libovolném počítačovém systému.“* [3] Jedná se o velice spolehlivý formát. Stránka vytisknutá z PDF by měla vypadat stejně na jakémkoli výstupním zařízení, pochopitelně v rámci jeho technických možností. Formát PDF je velmi blízký formátu PostScript (PS), jazyku, kterým si předávají informace vyspělé tiskárny, osvitky a další zařízení používaná v DTP[2] průmyslu.

K prohlížení formátu PDF slouží například program společnosti Adobe Acrobat Reader, který je dostupný zdarma v mnoha verzích pro různé operační systémy. V tomto programu lze soubory PDF pouze prohlížet a dělat si k nim komentáře. Pro zobrazení dokumentů tohoto formátu navíc existuje i celá řada jiných programů, jeho podpora je dnes ve světě informačních technologií již na vysoké úrovni. K vytváření PDF dokumentů jsou určeny ostatní produkty společnosti Adobe (Adobe Acrobat, Adobe Exchange, Ilustrator,...), ty ale nejsou zdarma dostupné.

[2] DTP – desktop publishing, tvorba tištěného dokumentu za pomoci počítače

Zmíněná fakta i mé osobní zkušenosti s PDF mne utvrdily v názoru, že volba tohoto produktu jako formátu dokumentace určené pro tisk, by měla být ideálním, nebo alespoň rozumným rozhodnutím.

1.4.2 Webová prezentace

Pro výstup aplikace ve formátu webové prezentace je vhodné implementovat generování dokumentace ve formátu HyperText Markup Language, tedy HTML. Autorem původní verze jazyka HTML, která vznikla roku 1991, je Tim Berners-Lee. Formát HTML by měl uživateli dokumentace databázového schématu nabídnout velmi sympatické možnosti, jak ji pročítat a jak v ní rychle dohledat potřebné informace o některém objektu. Jazyk HTML umožňuje propojit jednotlivé součásti dokumentace pomocí odkazů, umí do sebe zahrnout grafy vazeb mezi objekty, je přehledný a podporovaný všemi současnými prohlížeči stránek systému World Wide Web (WWW).

1.4.3 Upravitelná dokumentace

Kandidátů na výstupní formát pro pozdější úpravy bychom nalezli hned několik. Může jím být ten nejjednoduší, tedy plain text, složitější formáty DOC, RTF, nebo další, které ale nejsou tolik rozšířené a proto je pro náš účel nebudeme brát v potaz. I samotné formáty HTML a PDF lze do této kategorie zařadit, protože i k těm existují editory, umožňující jejich úpravy – tyto formáty jsme už ale jako výstupy generátoru odsouhlasili a proto se podíváme na ostatní.

Plain text (TXT) je velmi jednoduchým formátem s několika výhodami, ale řadou nevýhod. Pokud hledáme formát pro zobrazení čistě textových dat, pak je TXT tou nejvýhodnější, nejbezpečnější a nejméně problematickou volbou. Znaky jsou v něm uloženy v tzv. ASCII[3] kódu (každému znaku přísluší jedno číslo v ASCII tabulce). Plain text je možné otevřít v jakémkoli programu na kterémkoli počítači a vždy se objeví to samé.

V praxi má formát TXT více variant, především co se týká zalamování řádek. Na konci každého řádku v plain textu se může vyskytovat znak CR (Carriage return, tj. „návrat vozíku“[4]) nebo se tentýž znak může objevit vždy až na konci odstavce. Popsaný jev se může projevit jako problém ve chvíli, kdy si v operačním systému

[3] *American Standard Code for Information Interchange* (*americký standardní kód pro výměnu informací)*. Jde o kódovou tabulku která definuje znaky anglické abecedy.

[4] pojmenování z éry psacích strojů

Windows v programu „Poznámkový blok“ otevřeme TXT dokument s odrážkami jen na konci odstavců. Text se nám poté zobrazí v dlouhých řádkách.

Dalším problémem může být, že se pro plain text často nevyužívají přímo ASCII formáty znaků, ale různé jejich mutace podle použitého editoru a také různá kódování diakritiky.

Formát TXT se mi pro náš účel nejeví jako vhodný formát z následujících důvodů: Nepodporuje obrázky, které si v dokumentaci databázového schématu přejeme mít obsaženy. Dále plain text neobsahuje žádné formátovací instrukce, což znamená, že editor nepozná, která část textu má být zobrazena tučně, větším písmem, barevně, centrovaně a podobně.

DOC je příponou souborů formátu aplikace Microsoft Word. Konkrétní podoby souborů ve formátu DOC se mohou lišit v závislosti na verzi MS Word, ve které byly uloženy. Mezi jednotlivými novějšími verzemi Microsoft Wordu jsou však tyto soubory v dnešní době už plně přenositelné. Hlavní výhodou formátu DOC je fakt, že s sebou nese všechny informace o formátování dokumentu a také jeho jednoduchá rozšiřitelnost. Nevýhodou ale zůstává velmi významná skutečnost, že stejně jako do většiny produktů společnosti Microsoft, ani do formátu DOC dobře nevidíme. Jedná se o utajený proprietární formát a jsou mi známé jen dva způsoby, jak v něm vytvořit dokument. Jedním je placená aplikace MS Word (distribuuje se jako součást kancelářského balíčku MS Office), druhým pak aplikace OpenOffice. O automatickém generování formátu DOC mi není nic známo.

Posledním formátem z probírané kategorie je RTF neboli Rich Text Format. Je to univerzální formát navržený firmou Microsoft, který s sebou na rozdíl od TXT nese i formátovací instrukce. RTF je značkovacím jazykem, podobně jako například TeX nebo HTML.

Výhodou tohoto formátu je, že mu rozumí řada textových editorů „třetích stran“, a to nejen aplikace pod operačním systémem Windows. Dokáže s ním pracovat i nejpoužívanější textový editor Microsoft Word. Coby textový formát se dá do jiných formátů konvertovat snadněji než binární formáty.

Chceme-li dokument ve formátu RTF vytvořit, máme hned několik možností. Buď formátovaný soubor uložíme jako RTF v některém pokročilém textovém editoru (např. zmíněný MS Word, OpenOffice, Wordpad, nikoli ale např. Poznámkový blok) nebo se postaráme o konverzi z jiného formátu (TeX, LaTeX, XML). Další, ale pro běžného uživatele netriviální, možností je vytvořit takový dokument ručně v kterémkoliv jednoduchém editoru typu Poznámkový blok.

V dnešní době je formát RTF už široce využíván a je poměrně oblíbený. Pro projekt generátoru dokumentace databázového schématu jsem ho proto nakonec vybral jako výstupní formát pro možnost pozdější úpravy.

1.5 Příbuzné aplikace

Popis příbuzných aplikací se nevěnuje pouze programům pro tvorbu dokumentace k databázovému schématu. Dnešní databázová schémata obsahují i nezanedbatelné množství spustitelného kódu v triggerech, uložených procedurách a mnoha dalších typech procedurálních objektů. Jsou zde proto popsány i aplikace, pro generování programátorské dokumentace. V popisovaných programech bylo možné nalézt výbornou inspiraci pro následný vývoj.

1.5.1 Schemagic

Popis aplikací začínám tou, která má pro vyvíjený generátor největší význam. Schemagic je aplikace, jejíž výstup má být vstupem vyvíjeného programu. Smyslem Schemagicu je „vytáhnout“ data z databáze a dát tak jejímu uživateli přehled o jejím obsahu. Na výstup však vydává pouze strojově zpracovatelná data a tak se nemůže jednat o generátor dokumentace k databázovému schématu v pravém slova smyslu.

Tato data je třeba dále zpracovávat do uživateli přívětivých formátů a tuto činnost má za úkol zastat právě předmět tohoto projektu.

Schemagic je aplikací psanou v jazyku Java. Je dostupná jako open-source pod licencí GNU/GPL2 a je volně ke stažení na adrese http://sourceforge.net/projects/schemagic. Instalační balíček obsahuje adresář s knihovnami, dokumentací a také s konfiguračními soubory. V těch je možné určit databázové schéma, k němuž se má aplikace připojit, požadovanou strukturu výstupu a jeho obsah.

Z tohoto hlediska je určitě nejzajímavější podoba XML souboru “./def/machine/oracle.machine.xml”, kde adresa je relativní z domovského adresáře aplikace.

V elementu <model-def> tohoto souboru je určeno požadované rozložení objektů v generovaném výstupu.

V elementech <object-load> se pak definuje, který typ objektu má být získán kterým SQL příkazem SELECT. Podívejme se na jednoduchý příklad:

```
<object-load object_type="ora_table" class_name="org.schemagic
                .plugin.database.load.impl.JdbcSqlDbObjectLoader">
   <parameters>
      <parameter name="SQL_COMMAND"
         value="select TABLE_NAME, TABLESPACE_NAME from ALL_TABLES
                  where USER_NAME=':global[@SCHEMA_NAME]:'"/>
   </parameters>
   <pre-processing class_name="org.schemagic.plugin
                  .database.load.impl.SQLPreprocessor"/>
</object-load>
```

Příklad ukazuje, že tabulky, které jsou ve struktuře výstupu označeny atributem *type="ora_table"*, budou z databáze získány voláním třídy

org.schemagic.plugin.database.load.impl.JdbcSqlDbObjectLoader.

Pod tabulkami budou ve výstupu Schemagicu zavěšeny jejich vlastnosti, které se vybírají SQL SELECTem, konkrétně tedy hodnoty sloupců TABLE_NAME a TABLESPACE_NAME. Dále je zde uvedena informace, že má být před získáním tabulek zavolán plugin, jehož jméno udává hodnota atributu *class_name* elementu *pre-processing*. Tento plugin zajistí nahrazení parametrů obsažených v příkazu SELECT, v tomto případě hodnotu :global[@SCHEMA_NAME]: upraví na jméno, odpovídající požadovanému schématu.

Kompletní definice SELECTů, použité pro tento projekt, jsou uvedeny na přiloženém instalačním CD v souboru "./def/machine/oracle.machine.xml" (adresováno relativně z domovského adresáře aplikace Schemagic).

Aplikace Schemagic má v současné době implementovánu podporu práce s databázemi Oracle, je však snadno rozšiřitelná i na další typy databází. Umožnění práce s dosud nepodporovanými databázovými objekty je rovněž vcelku jednoduchou záležitostí.

1.5.2 Javadoc

Jedná se o aplikaci od společnosti Sun Microsystems, tvůrce jazyka Java. Slouží výhradně ke generování HTML dokumentace zdrojových kódů tohoto jazyka.

Javadoc je dnes široce využíván a jeho podpora je součástí prakticky každého vývojového prostředí pro Javu. Je nezávislý na platformě, neboť je součástí standardní distribuce JDK. Na svém vstupu očekává zdrojový kód Javy. Neumí generovat dokumentaci ve všech formátech, očekávaných od generátoru dokumentace k databázovému schématu, přesto alespoň jeho HTML výstup nabízí zajímavou inspiraci.

1.5.3 Doxygen

Doxygen je generátorem dokumentace pro programovací jazyky C++, C, Java, Python, IDL, PHP, C#, D a ActionScript. Funguje na většině Unixových systémů, pod operačním systémem Windows i pod Mac OS X. Většinu kódu Doxygenu napsal Dmitri van Heesch. K prosinci roku 2005 byla aplikace k dispozici ve verzi 1.4.

Doxygen generuje standardně dokumentaci ve formátu HTML. Narozdíl od Javadoc však umí vytvářet i formáty CHM, RTF, PDF, LaTeX, PostScript a manuálové stránky. Na svém vstupu ale neumí přijmout XML soubor, takže pro

účely tohoto projektu je nevyužitelný. Jeho princip je však principu vyvíjené aplikace velmi blízký a proto může být Doxygen zdrojem inspirace.

Tvorbu dokumentace provádí stejně jako Javadoc parsováním struktury dokumentu (programu) a zohledňováním komentářů při jednotlivých objektech. Výstupní dokumentace je obohacena obrázky vazeb mezi třídami zpracovávaného programovacího jazyka. V závislosti na nastavení programu může tyto obrázky generovat například vizualizační nástroj GraphViz.

1.5.4 SchemaSpy

SchemaSpy[5] je označen jako „prohlížeč metadat databázového schématu“ a vyvíjí ho John Currier. Jedná se o open-source aplikaci. Je psána v jazyku Java, aktuální verzí produktu je verze 3.1.1 vydaná v prosinci roku 2006. Smyslem programu je získat z databáze informace o tabulkách a pohledech a přehledně je zobrazit v HTML souboru.

Tato aplikace se hodně podobá cíli našeho projektu. Při jejím nastudování byla objevena řada předností i vážných nedostatků. Před spuštěním aplikace musí uživatel pomocí parametrů příkazové řádky definovat především typ databáze, její jméno, jméno uživatele a požadovaného schématu, dále heslo a volitelně také regulární výraz názvu tabulek a pohledů, které si přeje zdokumentovat.

Nic jiného než tabulky a pohledy ale výsledná dokumentace neobsahuje. Na domovských stránkách je k dispozici ukázka typického výstupu aplikace. Dává uživateli přehled o tabulkách a pohledech v databázi, přítomny jsou i grafy vazeb přes cizí klíče a vše je okomentováno databázovými komentáři. Výstup aplikace je standardně laděn do zelenobílých barev a výstup lze pomocí zaškrtávacích polí dynamicky ovlivňovat. K tomu je zřejmě použit jazyk Javascript.

Ke generování grafů vazeb mezi tabulkami slouží aplikaci program Graphviz.

SchemaSpy dále obsahuje také výpis anomálií přítomných v databázovém schématu. Umí uživateli dát varování o tabulce bez indexů, o existenci nulových a přitom unikátních sloupcích apod.

Předním nedostatkem aplikace SchemaSpy je schopnost generovat výstup pouze do formátu HTML, který není příliš vhodný pro tisk. Navíc nedává aplikace žádné informace o dalších objektech v databázi, jako jsou procedury, funkce, datové typy, triggery, balíky, těla balíků, atd.

Výhodou aplikace je kvalita generovaného HTML výstupu a spojení procesů získání dat z databáze a tvorby přehledné dokumentace. Výborná je také podpora typů databází. V současnosti aplikace podporuje dvanáct typů databází.

[5] http://schemaspy.sourceforge.net

1.5.5 DocFlex / XML

DocFlex / XML od společnosti Filigris Works je součástí rozsáhlé technologie DocFlex. Je generátorem dokumentace k XML schématu, na vstupu přijímá XSD soubor. Umí generovat dokumentaci ve formátu HTML (single-file i multi-framed), RTF a TXT. Generování je prováděno s použitím speciálních šablon (DocFlex technologie). Aplikace je programována v Javě a je ji možné spustit z příkazového řádku nebo z grafického uživatelského rozhraní. Zdarma jsou ke stažení pouze třicetidenní zkušební verze produktu. Pro účely našeho projektu může být DocFlex / XML opět významným zdrojem inspirace. Více o této aplikaci na domovských stránkách společnosti Filigris Works[6].

1.6 Další požadavky

V této sekci jsou popsány upřesňující požadavky na chování výsledné aplikace, případně upřesnění požadavků, uvedených v kapitole 1.1. Způsob splnění těchto požadavků je diskutován v následující kapitole.

1.6.1 Přenositelnost

Volba samotného programovacího jazyka pro jádro aplikace je sama o sobě velmi důležitá. Programovací jazyk musí umožňovat splnění podmínek, které jsem si vymezil v definici cíle projektu. Týká se to především toho, aby byla aplikace spustitelná na více platformách, minimálně na MS Windows a hlavních distribucích Linuxu.

1.6.2 Univerzální meziformát

Po aplikaci požadujeme, aby byla z jednoho vstupu schopna generovat výstup do většího počtu formátů. Toho lze dosáhnout přímo, tj. tak, že se kód programu rozvětví v závislosti na požadovaném výstupním formátu a tvorba každého z výstupů se bude řešit jako samostatný problém. Alternativou pro tuto myšlenku je využití nějakého prostředníka. Tím myslíme meziformát, který je

[6] http://www.filigris.com

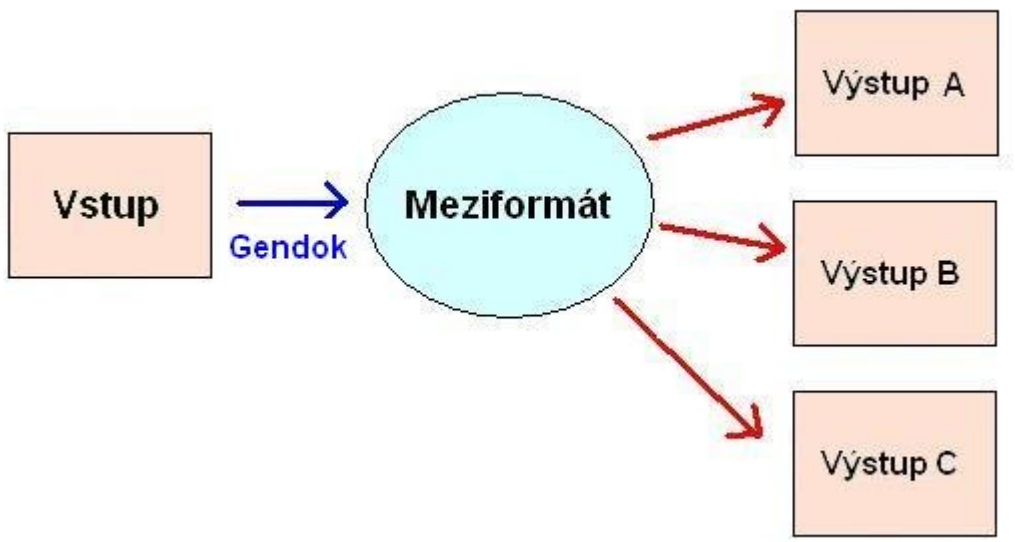
dostatečně silný

Daný meziformát musí umět zachytit kompletní obsah výsledné dokumentace včetně vložené grafiky, odkazů mezi jednotlivými objekty, členění do kapitol a sekcí a všechny další aspekty dokumentace

podporovaný aplikacemi třetích stran

Pro tento meziformát by měly existovat dostupné volně šiřitelné aplikace pro jeho zobrazování a především pro převody do potřebných cílových formátů.

Pokud by takový prostředník existoval a byl skutečně vhodný pro naše účely, dosáhli bychom jeho využitím ještě jedné nezanedbatelné výhody. Transformace vstupního XML souboru, generovaného Schemagicem, do patřičného výstupu by se potom dělila na dva kroky: Generování výstupu ve formátu prostředníka a následné už dobře nadefinované generování finálního výstupu.



Obrázek 0.1 Využití meziformátu

Vhodné bude zajistit, aby se i tento meziformát mohl stát vstupem generátoru. To může být důležité v případě generování dokumentace ve více formátech, kdy odpadne jeho opakované vytváření.

1.6.3 Generování grafických prvků

Již v popisu cílů tohoto projektu byl zmíněn požadavek na možnost vkládat do výsledné dokumentace nejen čistě textové informace, ale rovněž data ve formě obrázků nebo grafů. Grafické informace poskytnou uživateli podstatně lepší přehled o propojení objektů v databázovém schématu, například pomocí názorného zobrazení vazeb přes cizí klíče mezi tabulkami. U těchto grafických výstupů bude rozhodně vhodné pokusit se o to, aby byly vkládané grafy klikatelné, speciálně při jejich začlenění do HTML výstupu dokumentace, tj. aby měl uživatel možnost při prohlížení obrázku, znázorňujícího objekt A, přesunout se kliknutím na daný obrázek k popisu objektu A v příslušné části dokumentace.

K zajištění této funkcionality slouží minimálně v jazyku HTML tzv. obrázkové mapy, jež popisují, které části obrázku jsou odkazem na které URL adresy. Při generování obrázků bude tedy zřejmě nutné zajistit i to, aby byly vygerenovány i příslušné obrázkové mapy. V případě, že toto nezajistí přímo program produkující obrázky, bude nutné využít použitého programovacího jazyka.

1.6.4 Vícejazyčnost

Zajištění vícejazyčnosti, tedy toho, aby si uživatel mohl zvolit jazyk výstupní dokumentace, je jedním z méně důležitých problémů, který ostatně není určen jako cíl projektu. Přesto si dokáži představit princip, jakým této vlastnosti dosáhnout. Místo přímých slov a vět v konkrétním jazyku by měla aplikace pracovat s určitými proměnnými – nastavení těchto proměnných by potom mělo záviset právě na požadovaném kódu výstupního jazyka. Určení proměnných pro konkrétní jazyk by mělo být řešeno hromadně na jednom místě – pro každý jazyk by měl existovat například jeden konkrétní soubor s definicemi. Takové „slovníky“ pro jazyky by pak mělo být možné snadno doplňovat.

1.6.5 Změna vzhledu dokumentace

V případě HTML výstupu bude třeba zajistit, aby výsledná dokumentace měla v sobě definováno, který CSS styl hodlá využívat pro své zobrazení. Tyto styly by mohly být předdefinovány a uživatel by si mohl snadno některý z nich vybrat. V případě výběru stylu pak bude muset aplikace zajistit, aby se příslušný .css soubor správně zkopíroval do výstupního adresáře. Pro bližší určování vzhledu výstupu PDF, PS nebo RTF, bude potřeba nastavit vlastnosti výstupu FO, tedy formátovacích objektů.

Design

Tato kapitola obsahuje popis postupů použitých k dosažení cíle projektu. Kapitola rovněž dává přehled o pomocných technologiích a aplikacích, kterých vyvíjený generátor dokumentace využívá. Věnuje se rovněž možnostem budoucí rozšiřitelnosti aplikace pomocí rozhraní pro zásuvné moduly a popisuje očekávanou funkčnost, kterou s nimi lze dosáhnout.

1.7 Zvolené technologie

1.7.1 Java - jádro aplikace

Coby programovací jazyk pro tvorbu jádra aplikace jsem zvolil objektově orientovaný jazyk Java, vyvinutý v roce 1995 firmou Sun Microsystems. Java je jedním z nejpoužívanějších programovacích jazyků na světě. Jeho největší výhodou je velká přenositelnost, „*díky které je používán pro programy, které mají pracovat na různých systémech počínaje čipovými kartami (zajišťuje platforma JavaCard), přes mobilní telefony a různá zabudovaná zařízení (platforma Java Micro Edition), aplikace pro desktopové počítače (platforma Java Standard Edition) až po rozsáhlé distribuované systémy, pracující na řadě spolupracujících počítačů, rozprostřené po celém světě (platforma Java Enterprise Edition).*“ [2]

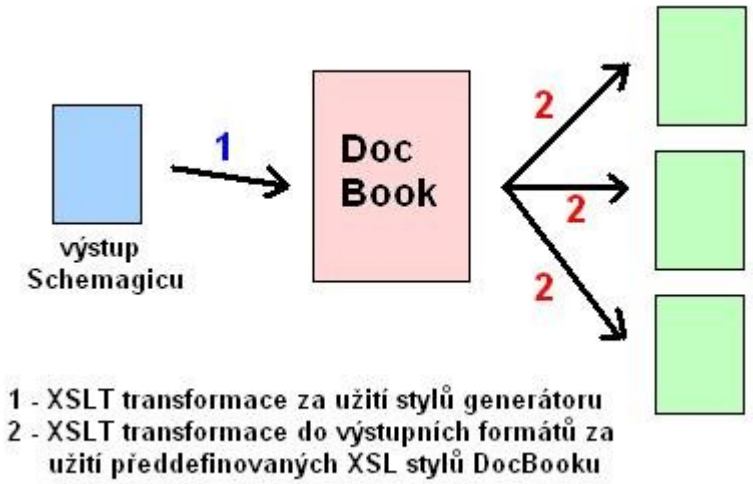
Jazyk Java je interpretovaným jazykem. Místo skutečného strojového kódu se vytváří pouze tzv. mezikód (bajtkód). Tento formát je nezávislý na architektuře počítače nebo zařízení. Program pak může pracovat na libovolném počítači nebo zařízení, který má k dispozici interpret Javy, tzv. virtuální stroj Javy - Java Vitrual Machine (JVM).

Nevýhodou tohoto jazyka je fakt, že je mírně pomalejší oproti například jazyku C++. Ke zpomalení dochází právě z toho důvodu, že je při spouštění programu nejprve potřeba přeložit bajtkód. Naproti tomu je jazyk Java obecně považován za velmi výkonný, bezpečný, jednoduchý a elegantní.

Při rozhodování o volbě jazyka (mezi Javou a C++) mne nakonec ve prospěch Javy mírně podpořil i fakt, že samotný Schemagic, ze kterého má vyvíjený generátor vycházet, je psán v jazyku Java, stejně tak jako řada aplikací, které bude aplikace nakonec nutna využít. Konkrétní verzí Javy, kterou jsem pro vývoj zvolil, je Java 1.5.0.

1.7.2 Meziformát DocBook

Na základě analýzy dostupných meziformátů vhodných pro proces generování dokumentace byla zvolena technologie DocBook XML[7]. Po jejím podrobném nastudování bylo usouzeno, že splňuje všechny požadavky uvedené v části 1.6.2. Tato technologie disponuje především sadou XSL stylů, použitelných pro převod dokumentu typu DocBook do formátů HTML, vícestránkové HTML, FO, HTMLHelp a dalších. Zmíněný formát FO (XML soubor formátovacích objektů) je dále použitelný pro generování PDF, Postscript a RTF výstupů. Verze DocBook XSL stylů, které jsem se rozhodl využít, nese označení 1.72.0.



Obrázek 0.1 Schéma využití formátu DocBook

Vzhledem k tomu, že se jedná o XSL styly, bude třeba zajistit, aby byla aplikace schopna vyvolávat XSLT transformace. Ty se zároveň mohou uplatnit už při převodu vstupního schématu, tj. výstupu aplikace Schemagic, do meziformátu DocBook.

1.7.3 Vizualizační nástroj Graphviz

Při hledání vhodného programu pro tvorbu grafů vazeb v databázovém schématu, jsem se na základě doporučení zaměřil především na aplikaci Graphviz, kterou pro svou práci využívají i generátory Doxygen a SchemaSpy, zmíněné v části

[7] DocBook – http://www.docbook.org/

1.5. Byl jsem s jejími výstupy natolik spokojen, že už jsem ji během následného pročítání dokumentací k jiným aplikacím za žádnou jinou nevyměnil.

Graphviz (Graph Visualization Software) je balíčkem open-source nástrojů, pocházejících z dílny AT&T Research Labs. Je určen pro tvorbu grafů, definovaných pomocí speciálních skriptů v tzv. jazyce DOT. Vztahuje se k němu celá řada knihoven pro různé programovací jazyky, čímž se usnadňuje jeho využívání v nejrůznějších programech. Jedná se o freeware aplikaci s licencí Common Public Licence.

Výše zmíněnými nástroji Graphvizu jsou: *dot, neato, twopi, circo, fdp, dotty a lefty*. Pro tvorbu jednoduchých orientovaný grafů je plně postačující nástroj *dot*, který bude také jako jediný pro účely generátoru dokumentace využíván.

1.7.4 XSLT procesor

Uvažujeme-li, že zvoleným programovacím jazykem pro implementaci jádra bude Java 1.5.0, neměl by být s voláním XSLT transformací žádný problém. Jazyk java disponuje balíčkem *javax.xml.transform*, který definuje volání těchto transformací. Pro vykonání samotné transformace je nezbytná přítomnost některého z XSLT procesorů. Nejznámějšími XSLT procesory jsou Saxon a Xalan. Volba konkrétního procesoru se v Javě určuje nastavením systémové proměnné *javax.xml.transform.TransformerFactory* na jméno třídy TransformerFactory zvoleného procesoru. Po pečlivém studiu informací na internetu jsem se rozhodl pro potřebu generátoru využít XSLT procesoru Saxon ve verzi 6.5.5 a Xalan ve verzi 2.7.0.

1.7.5 FO procesor FOP

Jak už bylo uvedeno, styly DocBook XSL nedefinují nic víc, než jakým způsobem je možné uskutečnit převod meziformátu DocBook do formátů HTML, FO a podobných. Aby bylo možné generovat výstup ve formátu PDF a RTF, je nutné zajistit dodatečnou transformaci z formátu FO (XML dokument formátovacích objektů).

Během analýzy jsem nalezl minimálně dvě poměrně populární aplikace, které požadovaný převod včetně formátu PostScript v současnosti zajišťují. První z nich nese označení XEP[8]. Jedná se o kvalitní program, který však není freeware a proto jsem možnost jeho využití odmítl. Druhým programem, je FO procesor FOP[9]. V

[8] http://www.renderx.com/tools/xep.html

[9] http://xmlgraphics.apache.org/fop/

době návrhu aplikace existoval FOP pouze ve verzi 0.20.5, která nebyla nedokonalá. Zdaleka neobsahovala implementaci všech elementů a atributů souboru formátovacích objektů. Převod FO výstupu do výstupu ve formátech PDF, Postscript nebo RTF tedy probíhal s celou řadou chybových hlášení a o výsledné dokumentaci se pak zdaleka nedalo mluvit jako o kvalitní.

Přesto jsem FOP neváhal zvolit za aplikaci, která bude mít za úkol převést FO výstup do dalších formátů. Na domácích stránkách FOPu se totiž už dlouho dopředu oznamoval příchod mnohem propracovanější, stabilnější a výkonnější verze 0.92. Tato verze byla skutečně koncem roku 2006 vydána. V současnosti generátor dokumentace využívá Java aplikaci FOP ve verzi 0.93.

1.8 Nastavení aplikace

Dalším bodem, který musel být při analýze aplikace dobře promyšlen, byla otázka způsobu nastavení aplikace. Generátor bude schopen přijímat více druhů vstupu a měl by zároveň umět vytvořit kterýkoliv z podporovaných výstupů dle aktuálního přání uživatele. Nabízí se tedy otázka, jakým způsobem bude moci uživatel aplikaci své přání sdělit.

Lze předpokládat, že ne každý uživatel bude chtít po aplikaci vytvoření dokumentace k databázovému schématu za stejným účelem, se stejným objemem údajů a ve stejném vzhledu. Zatímco někdo bude preferovat, aby výstup dobře popisoval definice tabulek v databázi, jiného uživatele mohou zajímat pouze procedury, funkce a obsahy specifikací a těl balíků. Samotný obsah výsledné dokumentace by tedy měl být dobře nastavitelný.

V dané otázce jde však o víc, než jen o pouhá přání uživatele, jejichž splnění má tomu či onomu usnadnit práci. Jde také o to, aby samotná aplikace věděla při svém spuštění vše, co si ona a programy, které využívá, přejí a potřebují vědět. Rozhodně by měl mít uživatel povinnost nastavit správnou polohu pomocných aplikací v adresářové struktuře svého počítače, tj. domovský adresář XSL stylů DocBooku, procesoru FOP, Saxonu, Xalanu a dalších. Pro běh aplikace bude rozhodující i to, v jakém kódování si uživatel přeje svůj výstup ukládat, případně v jakém jazyku, kam má být výstup umístěn a zda má aplikace ponechávat či mazat případné mezivýsledky své činnosti.

Takových nastavitelných vlastností může být celá řada a jeví se jako rozumné řešení umožnit jejich určení na jednom konkrétním místě. Pro tento případ jsem se rozhodl definovat pro generátor dokumentace jeho vlastní konfigurační XML soubor. V něm by pak byly určeny všechny uvedené (a třeba ještě ani neuvedené) parametry aplikace, které by si následně Java kód programu načetl a následně je poctivě využíval pro rozhodování ve své činnosti.

Stejně tak by konfigurační soubor měl obsahovat definici obsahu výsledné dokumentace, a tedy které kapitoly a podkapitoly se v ní mají objevit, a v jakém přesně pořadí. Aplikace by pak na základě tohoto určení mohla „chytře“ poskládat XSLT šablonu pro transformaci ze vstupního schématu do podoby souboru DocBooku.

Parametry, u kterých lze předpokládat, že se budou častěji měnit, je možné aplikaci předat přímo při jejím volání na příkazové řádce. Je pochopitelné, že tímto způsobem lze programu jen obtížně poskytnout informaci, jakou má mít přesně výstupní dokumentace strukturu. Určitě by na ní mělo být možné určit, kde se nachází konfigurační soubor s kompletním nastavením aplikace. Stejně tak je vhodné umožnit s její pomocí volbu vstupního a výstupního formátu.

1.9 Tvorba formátu DocBook

Tato sekce je věnována podrobnému popisu převodu vstupního schématu do meziformátu DocBook za použití „poskládané“ šablony. Systém DocBook v sobě definuje více typů dokumentů, mezi které patří mimo jiné článek, kniha a referenční stránky. Pro účely dokumentace k databázovému schématu je nejvhodnější druhý z nich. Kniha DocBooku se skládá z kapitol, ty dále ze sekcí. Mohou být rovněž přítomny prvky jako záhlaví, zápatí, jméno autora, datum vydání, rejstřík apod. Převod vstupního souboru generátoru do formátu DocBook tedy znamená vytvoření příslušných kapitol, podkapitol a sekcí.

Díky tomuto členění a díky možnostem XSLT transformací se nabízí možnost tvorby DocBook dokumentace pomocí řady malých šablon. Je možné vytvořit například XSL šablonu pro vytvoření sekce s grafy vazeb mezi tabulkami, dále šablonu vytvářející přehled o triggerech ve schématu a následně si pak z konfiguračního souboru aplikace zjistit, v jakém pořadí si uživatel přeje zahrnout tyto sekce do výstupní dokumentace.

Představ o podobě generované dokumentace může být několik, nastíníme si tu jednu z nich. Kapitoly jako hlavní části knihy by mohly být určeny podle druhů objektů v nich popsaných. Mohla by být tedy generována zvláštní kapitola pro tabulky ve schématu, druhá pro pohledy, jiná pro funkce, procedury atd.

V takovém případě se nabízí představa, že další úrovní, tedy přímými podkapitolami, byly tyto konkrétní objekty. Pod těmi by pak mohly být zařazeny sekce s jejich jednotlivými charakteristikami. Jako příklad této myšlenky uvedu schematický zápis:

```
<kapitola nazev=”Tabulky”>
   <podkapitola nazev=”tabulka AUTOMOBILY”>
      <sekce nazev=”Omezeni tabulky”>
      <sekce nazev=”Triggery tabulky”>
      <sekce nazev=”Indexy tabulky”>
   </podkapitola>
   <podkapitola nazev=”tabulka PUJCOVNY”>
      <sekce nazev=”Omezeni tabulky”>
      <sekce nazev=”Triggery tabulky”>
      <sekce nazev=”Indexy tabulky”>
   </podkapitola>
</kapitola>
```

Z tohoto zápisu lze vytušit, že proces generování sekcí pro obě podkapitoly bude podobný. Mohla by tedy (a dokonce by měla) existovat například XSL šablona, generující sekci s definicí omezení pro tabulku, kde název této tabulky by byl jejím parametrem. Je také možné vytvořit více šablon, generujících data s různým stupněm podrobnosti a v konfiguraci zvolit tu, která uživateli nejvíce vyhovuje. Analogicky pro sekce „Triggery tabulky“ a „Indexy tabulky“.

K realizaci této úvahy by tedy bylo zapotřebí vytvořit sadu XSL šablon pro každý popisovaný typ objektu. Každá z nich by měla jiný konkrétní úkol, ale všechny by výsledně definovaly vytvoření některé součásti DocBook knihy. Jádro aplikace by pak mělo za úkol obeznámit se prostřednictvím konfiguračního souboru s přáním uživatele, jakou má mít výstupní dokumentace strukturu, a na základě těchto údajů by rozhodlo, jakým způsobem z malých XSL šablon pro tvorbu částí knihy vytvořit velkou převodní šablonu pro vznik plnohodnotné knihy DocBooku.

Definice struktury výstupu by mohla být v konfiguračním souboru určena přibližně následovně:

```
<kniha>
   <kapitola nazev=”Tabulky”>
      <pro-kazdy-objekt typ=”tabulka”>
         <zahrn-sablonu nazev=”sablona-omezeni-tabulky”>
         <zahrn-sablonu nazev=”sablona-triggery-tabulky”>
         <zahrn-sablonu nazev=”sablona-indexy-tabulky”>
      </pro-kazdy-objekt>
   </kapitola>
</kniha>
```

Očekává se, že aplikace bude vstupní soubor načítat do své vnitřní struktury – Java má pro manipulaci s XML dokumenty vyvinutou technologii JAXP[10], jejíž součástí je tzv. DOM dokument. Jedná se o datový typ, reprezentující soubor XML. Ve chvíli, kdy jádro aplikace načte do vnitřní paměti vstupní soubor, sestaví podle

[10] Java API for XML Processing – rozhraní jazyka Java pro zpracování XML dokumentů

přání uživatele převodní šablonu a následně zavolá XSLT transformaci do formátu DocBook.

1.10 Princip pluginů

Málokterá aplikace se okamžitě po své implementaci dá považovat za dokonalou a úplnou. Většinou se přímo nabízí myšlenky na její rozšíření tím či oním směrem, na různá doplnění, rozšíření funkcionality. Pro tento účel bývá u aplikací vyžadována podpora pluginů – zásuvných modulů, bez nichž se aplikace sice plně obejde, ale přítomností libovolného z nich získá novou funkčnost, která může být naplno využívána. Pluginy samy o sobě většinou spustit nelze, a nebývá obvyklé, aby byly implementovány už ve chvíli, kdy je dokončena samotná aplikace. Je pro ně většinou definováno rozhraní, které musí každý z nich implementovat, ale dále se jim už nic jiného nenařizuje. Plugin může vykonávat libovolnou činnost s prostředky, které mu jsou poskytnuty.

Pro aplikaci generátoru dokumentace databázových schémat bylo rozhodnuto naimplementovat podporu zásuvných modulů. Bylo tedy potřeba určit, jakým způsobem uživatel zadá zavolání určitého pluginu a co vše mu vlastně bude předáno. Jako nejlepší byla určena možnost dát pluginu prostředky k manipulaci už přímo se vstupním souborem aplikace (tedy s výstupem schemagicu), načteným ve vnitřní struktuře DOM.

V takovém případě, ať už bude mít plugin na starosti cokoliv, může výsledek své práce vždy připojit na určité místo v tomto vstupním souboru dříve, než se provede jeho XSLT transformace do formátu DocBook. Definované styly pro tuto transformaci pak mohou naplno počítat s tím, že tuto informaci (získanou pluginem) ve vstupu naleznou a mohou s ní naložit podle své libosti.

Pro možnost jednoduchého volání pluginů jsou jejich jména a parametry začleněny přímo do konfiguračního souboru.

```
<kniha>
   <kapitola nazev=”Tabulky”>
      <pro-kazdy-objekt typ=”tabulka”>
         <zahrn-sablonu nazev=”sablona-omezeni-tabulky”/>
         <zahrn-sablonu nazev=”sablona-triggery-tabulky”>
            <volej-plugin nazev=”gendok.plugin.Plugin”>
               <parametr jmeno=”p1” hodnota=”h1”/>
               <parametr jmeno=”p2” hodnota=”h2”/>
            </volej-plugin>
         </zahrn-sablonu>
         <zahrn-sablonu nazev=”sablona-indexy-tabulky”/>
      </pro-kazdy-objekt>
   </kapitola>
</kniha>
```

Jestliže v něm bude uvedena požadovaná struktura výsledné dokumentace podobně, jako v tomto příkladě, budou v něm zřejmě pro každou požadovanou kapitolu nebo podkapitolu také názvy XSLT šablon, kterými má být konkrétní kapitola - například *sablona-triggery-tabulky* apod. Když potom uživatel bude vědět, že některá z těchto šablon automaticky využívá informaci, která bude do vstupu doplněna až pluginem, měl by mít možnost zavěsit informaci o nutnosti zavolání pluginu právě pod název této šablony.

Pluginy, které se pro generátor dokumentace nabízejí v první řadě, jsou popsány v následujících sekcích. Realizace některých z nich je popsána v kapitole 1.16.

1.10.1 Generátor zdrojového kódu

Tento plugin by pro každý objekt v databázovém schématu na základě informací ve vstupním souboru vygeneroval přesný zdrojový SQL kód, kterým byl daný objekt vytvořen, případně SQL kód, kterým může být objekt zrušen, pozměněn apod. Plugin by příslušný SQL kód vytvořil dle svých pravidel a před ukončením své činnosti by ho zavěsil pod daný objekt ve vstupním souboru, například pod značku s označením <code>. Šablona, definující XSLT transformaci tohoto objektu by pak definovala, že obsah případného elementu <code> má být v DocBook dokumentu uveden tím nebo oním způsobem.

Konkrétní příklad: V databázovém schématu se nachází tabulka ZAKAZNIK. Schemagicem je vygenerován XML soubor, popisující toto schéma, a tedy se v něm rovněž nachází element pro tabulku ZAKAZNIK. Jsou pod ním zavěšeny elementy popisující její vlastnosti, její primární klíč, cizí klíče a všechna ostatní omezení. Stejně tak jsou tu k dispozici i indexy přes danou tabulku. V konfiguračním souboru je určeno, že pro šablonu GENERUJ-VYTVORUJICI-KOD ma byt zavolan plugin s nazvem A. Tento je tedy na vstupni dokument zavolán, sesbírá si všechny uvedené informace o tabulce a s jejich pomocí vytvoří řetězec příkazu, kterým byla tabulka vytvořena. Tento řetězec následně zavěsí zpátky do DOM reprezentace vstupního dokumentu, například přímo pod element tabulky ZAKAZNIK do nového elementu <code>.

Šablona GENERUJ-VYTVORUJICI-KOD má potom definovat, jakým způsobem má vzniknout kapitola (nebo sekce) DocBook knihy, která bude uživateli ukazovat, kterým CREATE příkazem byla tabulka ZAKAZNIK vytvořena. Pro tuto činnost stačí šabloně obsah o jednoduchém příkazu: zobraz v příslušné sekci DocBooku hodnotu elementu <code>, zavěšeného pod hlavním elementem tabulky ZAKAZNIK.

1.10.2 Generátor grafů vazeb mezi objekty

V tomto případě se jedná o zajištění výše diskutované funkcionality. Výsledná dokumentace má mimo obyčejný text obsahovat i grafy vazeb mezi objekty. Tyto grafy by měly vzniknout v podobě obrázků. Pro jejich generování byla zvolena aplikace Graphviz (viz 1.7 Zvolené technologie), používaná pro podobné účely i v jiných aplikacích pro tvorbu dokumentace.

Zdá se rozumné zajištění této funkcionality formou pluginu. Při nepřítomnosti pluginu pak bude generátor fungovat dobře, jen nebude vytvářet grafy. Plugin by měl na základě patřičných informací ve vstupním souboru vytvořit příslušné grafy a následně do vstupního souboru připojit informace, tzv. obrázkové mapy, které v HTML dokumentacích zajistí klikatelnost těchto grafů.

1.10.3 Prettyprinter, syntaxhighlighter

Základním úkolem tohoto pluginu by měla být úprava zdrojových kódů, obsažených ve vstupním souboru. Ty jsou ve vstupu uloženy bez jakýchkoliv informací o vzhledu, především chybí zvýraznění syntaxe a zlomy řádek jsou reprezentovány pouze speciální sekvencí znaků, nikoliv přímo znakem zlomu řádky. Bude nutné, aby tento plugin znal základy syntaxe jazyka PL-SQL. Měl by pracovat s kódy funkcí, procedur, pohledů, typů, triggerů, a také se specifikacemi a těly balíků.

Pro splnění požadavků je potřebné implementovat pouze druhý z popisovaných pluginů. Jeho implementace může zároveň sloužit jako dobrý příklad pro další rozšiřování aplikace.

Implementace

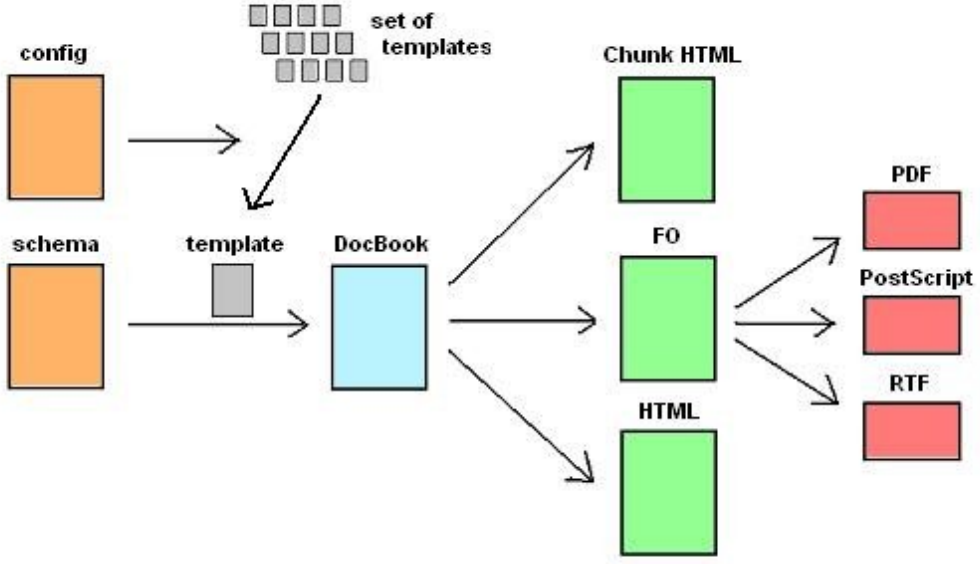
Tato kapitola detailněji popisuje skutečnou implementaci vlastností a postupů, popsaných ve fázi analýzy a designu aplikace. V některých případech bylo nutné návrh dále mírně upravit, aby bylo možné jej řádně implementovat. Změny však nebyly nijak zásadní. Řešení jednotlivých součástí aplikace jsou podrobněji popsány v samostatných podkapitolách.

1.11 Obecný princip

Obecný princip fungování aplikace odpovídá popisu, uvedenému v podrobné analýze. Jádro aplikace je programováno v jazyku Java ve verzi 1.5.0 a stará se postupně o následující procesy:

- Správné načtení uživatelského nastavení z příkazové řádky a z konfiguračního souboru a jeho uložení do k tomu určené vnitřní struktury
- Volání všech požadovaných pluginů, v daném pořadí.
- Volání příslušných XSLT a FO transformací.
- V případě jakéhokoliv selhání se jádro aplikace stará o úklid dočasných souborů a o zápisu chyby do logu, který je směrován na stardadní výstup.

Postup práce generátoru nejlépe ozřejmuje následující obrázek. Šipky v něm vyznačují směry XSLT a FO transformací.



Obrázek 0.1 Znázornění práce generátoru

Povolenými vstupními formáty pro aplikaci jsou Schemagic XML (v obrázku označený jako *schema*), DocBook a FO. Uvažujme první případ, neboť ostatní jsou pouze jeho podúlohami.

Java aplikace si načte do své vnitřní struktury soubor *schema* a následně se seznámí s obsahem konfiguračního souboru *config*. Tam uživatel definoval, jaké kapitoly a podkapitoly (a v jakém pořadí) si přeje mít v meziformátu DocBook a tedy i ve výsledné dokumentaci. Na základě tohoto přání aplikace sestaví XSL šablonu *template*, která popisuje převod souboru *schema* na soubor *DocBook* a bude pro tento převod využita.

V adresáři *xsl* v domácím adresáři instalace gendoku jsou přítomny malé XSL šablony (v obrázku označené jako *set of templates)*, pro jednotlivé kapitoly, podkapitoly či sekce. Právě z nich je sestavena celá převodní XSL šablona. Ve chvíli, kdy je správně vytvořena, je automaticky zavolána transformace (metoda *transform* třídy *javax.xml.transform.Transformer*).

Je nutno zdůraznit, že transformace neprobíhá z původního vstupního souboru *schema*, ale z jeho pozměněné verze, která vznikla po zavolání všech vyžadovaných pluginů. Pluginy se uvádí společně se strukturou výstupu v konfiguračním souboru – viz 1.10. V závěrečné fázi transformace se Java aplikace postará o to, aby se do výstupního adresáře k DocBook souboru zkopírovaly i všechny potřebné obrázky.

Další fází převodu, která může následovat, pokud není požadovaným výstupem přímo soubor DocBooku, je generování HTML, vícestránkového HTML

nebo FO dokumentace. Zde je úkolem aplikace zavolat XSLT transformaci, kde vstupem bude soubor DocBooku a šablonou popisující převod bude některá z předdefinovaných šablon DocBooku. Jak je uvedeno v podkapitole 1.12.2, konfigurační soubor v sobě vyžaduje určení vlastnosti *docbook-xsl*, jejíž hodnota udává místo, kde se v uživatelově file systému nachází adresář s XSL šablonami technologie DocBook. Po zavolání transformace by měl vzniknout jeden z výstupů c*hunk HTML, HTML output, FO output*.

V případě *HTML* nebo *chunk HTML* výstupu je vše hotovo. Aplikace se pouze ještě postará o to, aby se do výstupního adresáře zkopíroval také požadovaný CSS styl a všechny potřebné obrázky. Cestu k CSS stylu lze opět určit v konfiguračním souboru definicí vlastnosti *css-stylesheet* – viz 1.12.2.

Pokud se ale jedná o FO výstup, může tvorba dokumentace pokračovat. Formát FO slouží jako mezistupeň pro vytvoření dokumentace nejen ve formátu PDF, ale také ve formátu Postscript nebo RTF. Pro tuto závěrečnou transformaci je vyžadována přítomnost balíku *org.apache.fop.apps* v proměnné classpath pro volání Java aplikace.

Generátor na závěr provede volání aplikace FOP 0.93, které předá FO soubor formátovacích objektů. Proběhne-li vše bez problémů, vznikne výstup v jednom z požadovaných formátů – PDF, Postscript nebo RTF.

1.12 Konfigurační soubor

Jak je uvedeno v kapitole 1.8, konfigurační soubor je vedle příkazové řádky hlavním místem, na kterém se očekává, že dá uživatel vědět o svých přesných požadavcích pro proces generování dokumentace.

1.12.1 Popis DTD

Struktura souboru je vymezena pomocí DTD[11], která je k dispozici v souboru „./conf/config.dtd“ – relativně adresováno z domovského adresáře aplikace Gendok. Validita konfiguračního souboru podle tohoto DTD je z jádra aplikace ověřována a pokud je soubor nevalidní, aplikace ohlásí chybu a je zastavena. Pro snazší orientaci čtenáře tu ale popis povolené podoby konfiguračního souboru uvedu. Jedná se o XML soubor, který může obsahovat následující skupinu značek:

[11] Dokument Type Definition – definice typu dokumentu

<documentation>

je kořenovým elementem konfiguračního souboru. Nesmí obsahovat žádný atribut. Jeho potomkem musí být právě jeden element „properties“ a právě jeden element “book“ v tomto pořadí.

<properties>

je elementem obsahujícím hodnoty nastavitelných vlastností aplikace. Jemu povolenými potomky jsou elementy „property“, kterých může být libovolný počet, i nulový. Pro element properties není definován (a tedy ani povolen) žádný atribut.

<property>

je element pro definici hodnoty jedné konkrétní vlastnosti. Jméno této vlastnosti je určeno hodnotou atributu „name“, její hodnota potom obsahem atributu „value“. Element property je koncovým uzlem XML stromu a nesmí tedy obsahovat žádný podelement.

<book>

je element, uzavírající v sobě definici struktury výstupní dokumentace. Obsahuje tedy informace, která kapitola bude ve výstupní dokumentaci předcházet které, případně které sekce pod ní budou zavěšeny a které pluginy je nutno zavolat, aby vše proběhlo korektně. Pro element book nejsou definovány žádné atributy. Od elementu se očekává, že bude obsahovat minimálně jeden podelement „chapter“ a žádný jiný.

<chapter>

je elementem, určujícím, že si ve výstupní dokumentaci přejeme mít vygenerovanou kapitolu s názvem, určeným hodnotou atributu „title“. Atribut je však nepovinný a v případě jeho neuvedení bude mít název příslušné kapitoly defaultní prázdnou hodnotu. V dokumentaci pak bude tato kapitola uvedena pouze jako „Kapitola 1“. Element book musí obsahovat alespoň jeden podelement „foreach“.

<foreach>

je element určující, ke kterému typu objektu se vztahují elementy, zavěšené pod ním. Očekává se pod ním pouze libovolné množství (i nulové) elementů „section“. Vyžadovaným atributem elementu je atribut „elem“, volitelným atributem je „elem-name“.

<section>

je element definující, že si ve výstupní dokumentaci v dané kapitole přejeme mít vygenerovanou sekci, která vznikne voláním šablony odvozené z hodnoty povinného atributu „type“. Tato sekce bude zavěšena pod každým objektem typu, který je určen nadřazeným elementem „foreach“. Element může obsahovat libovolný počet přímých potomků s názvem „call-plugin“ (tedy i nulový).

<call-plugin>

je element určující, že pro správné zobrazení výstupu si tato sekce, která je předkem značky „call-plugin“ vyžaduje zavolání pluginu, jehož jméno je v hierarchii balíčku a tříd Javy uloženo v hodnotě povinného atributu „name“. Jiný atribut element call-plugin mít nesmí. Mohou ale nemusí pod ním být zavěšeny elementy „param“ v libovolném množství.

<param>

je element pro vyjádření parametru, předávaného některému pluginu. Element nesmí mít žádné potomky, zato musí mít právě dva atributy – „name“ a „value“. První z nich definuje jméno parametru předávaného pluginu, druhý jeho hodnotu.

Konfigurační soubor je tedy logicky rozdělen na dvě významově rozdílné části. První tvoří element „properties“ a jeho potomci, druhou potom element „book“ a jeho potomci. První část má význam definice jmen a hodnot vlastností pro proces generování výstupní dokumentace, druhá část definuje strukturu této výstupní dokumentace.

Příklad konfiguračního souboru je uveden v Příloze B tohoto dokumentu.

1.12.2 Definice vlastností

Definice vlastností v elementu „properties“ je poměrně jednoduchou záležitostí. Může obsahovat pouze elementy „property“, které mají atributy, označující jméno (atribut „name“) a hodnotu (atribut „value“) vlastnosti. Následuje popis všech podporovaných vlastností:

output-directory

Jméno adresáře, kam bude umístěn výstup. Povolenou hodnotou je každá lokální URI adresa, která navíc může obsahovat některý ze dvou speciálních symbolů ::FORMAT:: a ::TIMESTAMP::. První z nich bude nahrazen malými písmeny jména požadovaného výstupního formátu, druhý pak časovou známkou ve

tvaru YYMMDD-HHmmSS, například 070531-235959 pro vteřinu před půlnocí dne 31. května 2007.

output-encoding

Výstupní kódování. Zde určené kódování bude uvedeno v hlavičce všech výstupních XML souborů. Neznámá kódovní většinou nahrazují transformační procesory svým defaultním kódováním. Toto chování neovlivňuje naše aplikace a proto tu není podrobně rozebráno. Obvyklými hodnotami output-encoding pro češtinu jsou WINDOWS-1250, ISO-8859-2, UTF-8.

output-language

Zde se předpokládá libovolná textová hodnota, odkazující se na soubor adresáře „xsl/lang“ (relativně adresováno z domovského adresáře aplikace). Je-li hodnotou této vlastnosti například řetězec „cs“, potom musí v adresáři „xsl/lang“ existovat soubor jménem „cs.xsl“, který bude includován pro úvodní XSLT transformaci ze vstupního schematu Schemagicu do formát DocBook. Očekává se, že tento soubor obsahuje tzv. slovník. Aplikace disponuje minimálně dvěmi základními slovníky „cs.xml“ a „en.xsl“ pro češtinu a angličtinu.

V případě pokusu o vytvoření nového slovníku, vytvořte prosím soubor se stejnou strukturou a přepište do nového jazyka pouze hodnoty atributů SELECT jednotlivých elementů <xsl:param>.

erase

Hodnotou této vlastnosti může být buď „yes“ nebo „no“ v závislosti na tom, zda si uživatel přeje, aby byly mazány mezivýsledky procesu generování dokumentace („yes“) nebo byly ponechány („no“). Například při generování výstupu do formátu PDF se jedná o soubory typu DocBook, FO a adresář s příslušejícími obrázky.

docbook-xsl-home

Jméno adresáře s XSL styly technologie DocBook. Je vyžadována struktura tohoto adresáře v takové podobě, v jaké byla minimálně pro verzi DocBook XSL 1.72.0. V jiném případě by se volání těchto XSL šablon z Java kódu nezdařilo. Předpokládá se ale, že ze stejného důvodu vývojáři DocBook stylů tuto strukturu nebudou měnit.

css-stylesheet

Cesta k CSS stylu pro HTML výstup. Adresa je relativní z domovského adresáře aplikace. Předdefinované styly jsou uloženy v adresáři *„css“*. Příklad hodnoty této vlastnosti: „./css/html_01.css“.

fop-cfg-file

Jedná se o určení cesty ke konfiguračnímu souboru pro aplikaci FOP. Defaultní konfigurační soubor bývá většinou umístěn v podadresáři „conf“ domovského adresáře této aplikace a obvykle nese název „fop.xconf“.

xslt-procesor

Označuje jméno Java třídy, která je implementací rozhraní XSLT procesoru *javax.xml.transform.TransformerFactory*. Podporovanými XSLT procesory jsou Saxon 6.5.5 a Xalan 2.7.0 a tak může tato vlastnost reálně nabývat jedné z těchto hodnot:

com.icl.saxon.TransformerFactoryImpl
org.apache.xalan.processor.TransformerFactoryImpl

html-pix-format, fo-pix-format

Určují typ obrázků, jaký si uživatel přeje mít přítomen pro výstup v daném formátu. Pro HTML jsou povolenými hodnotami *png* a *gif*, pro FO jsou poveleny formáty *svg* a *png*. Právě jedna hodnota musí být nastavena pro každou z těchto vlastností.

docbook-pix-format

Význam parametru je stejný jako dvou předchozích, liší se ale v podstatné věci. Jako hodnota může být určeno více obrázkových formátů, oddělených čárkami. Povolenými kombinacemi jsou: "*png,svg,gif*", "*png,svg*", "*png,gif*", "*gif,svg*", "*png*", "*svg*", "*gif*".

1.12.3 Definice struktury výstupu

Druhou součástí obsahu konfiguračního souboru pro aplikaci generátoru dokumentace je definice struktury výstupního dokumentu. Jak musí vypadat stromové určení této struktury je dobře patrné z DTD pro konfigurační soubor, které je vypsáno i na začátku této podkapitoly. Zbývá tedy podívat se na věc komplexně.

Uživatel má možnost vytvořit pro své potřeby jakoukoliv XSL šablonu pro vyjmutí některé informace ze vstupního souboru a její uložení do sekce nebo kapitoly vytvářeného formátu DocBook. Nechce-li si psát vlastní šablonu, může využít šablon už definovaných, které jsou všechny volány z defaultního konfiguračního souboru.

Podívejme se na výtah z běžného konfiguračního souboru:

```
<chapter title="::L-TABLES::">
   <foreach elem="table" elem-name="::L-TABLE::">
      <section type="table-columns"/>
      <section type="table-comments"/>
   </foreach>
</chapter>
```

Z ukázky lze vyčíst následující:

Výstupní dokumentace, vzniklá voláním s tímto konfiguračním souborem, bude obsahovat kapitolu, jejíž název je určen proměnnou L-TABLES. Její hodnotu je nutno hledat pro aktuální jazyk volání programu v adresáři „./xsl/lang“ v souboru o jméně shodném s kódem aktuálního jazyka a příponou *.xsl*. Přesto hodnotou atributu title nemusí být pouze název proměnné, ale i přímo požadovaný název kapitoly bez ohledu na jazyk.

Dále je z příkladu patrné, že v této kapitole bude tolik sekcí, kolik je ve schématu, které dokumentace popisuje, objektů typu TABLE – typ těchto objektů určuje hodnota atributu „elem“. Ve vstupním souboru (v XML souboru vytvořeném Schemagicem) odpovídá hodnota tohoto atributu hodnotě atributu „family“ u značky „object“.

Každá sekce v kapitole, o které hovoříme, bude tedy v tomto případě popisovat jeden z objektů typu TABLE. Dále je ve výtahu z konfiguračního souboru uvedeno, že každá taková sekce má mít dvě podsekce. První z nich vznikne zavoláním XSL šablony s názvem *table-columns* a druhá vznikne zavoláním šablony s názvem *table-comments*.

Vyžaduje se, aby po nahrazení pomlček v názvech šablon lomítky a doplnění na konec přípony *.xsl*, vznikla relativní cesta k souboru z adresáře „./xsl“, v němž se daná šablona nachází. Java aplikace se automaticky postará o zahrnutí tohoto souboru do velké šablony pro transformaci „Vstup --> DocBook”.

Pro obsah takové šablony pak platí, že jsou v ní kontextovými uzly jednotlivé elementy daného typu ze vstupního souboru.

Závěrem je nutno dodat klíčovou vlastnost konfiguračního souboru, a tou je „podřízenost příkazové řádce“. Je-li možno některou z vlastností aplikace definovat jak při jejím volání na příkazové řádce, tak v konfiguračním souboru, potom platí, že při definicích na obou místech je rozhodující ta na příkazové řádce.

1.13 Jádro aplikace

Přesnou podobu struktury zdrojového kódu v Javě lze vyčíst z JavaDoc dokumentace, přiložené k projektu. Přesto je zde potřeba uvést aspoň základní informace a ozřejmit rozdělení práce mezi jednotlivé Java třídy.

Jádro aplikace se skládá z řady tříd zahrnutých do balíků *„gendok“*, *„fopemb“* a *„graphviz“*. Poslední dva uvedené jsou však využívané pouze ke snazšímu volání externích programů přímo z Java kódu. Samotná aplikace je pak umístěna v balíku *„gendok“* a jemu podřízených balících.

package gendok

V balíku *„gendok“* je veřejnou třídou se statickou metodou *„main“* třída Gendok. Tam veškerá činnost aplikace začíná. Daná třída umožňuje načíst parametry z příkazové řádky a dále volá třídy pro načtení a uchování informací z konfiguračního souboru a pro tvorbu výsledné dokumentace. Na závěr má tato hlavní třída za úkol odchytit kritickou výjimku, která v aplikaci může nastat, zalogovat ji a postarat se o patřičný úklid dočasných a nadbytečných souborů.

Třídou volanou pro načtení a uchování informací z konfiguračního souboru je statická třída *Configuration*. Třídami majícími na starost tvorbu dokumentace jsou třídy: *BookCreater*, *JustHtmlCreater*, *ChunkHtmlCreater*, *FoCreater*, *PdfCreater*, *PsCreater*, *RtfCreater* z balíku *gendok.create*. O jejich významu a funkci dobře vypovídají jejich názvy.

Statická třída *Configuration* zastřešuje nastavení aplikace při daném spuštění. Je to souhrn informací, předaných programu přes příkazovou řádku a přes konfigurační soubor. Mezi tyto informace patří určení struktury výstupního souboru, vstupní formát, výstupní formát, informace, kde nalézt potřebné externí programy (DocBook XSL home directory, Graphviz home directory, atd.), CSS styl pro HTML výstup, jazyk výstupu, atd. Třída disponuje celou řadou funkcí pro ukládání i načítání v ní uložených vlastností aplikace.

Jinou důležitou funkcí je funkce *load*, která dostane jako parametr třídu File s Java reprezentací konfiguračního souboru. Funkce se postará o načtení informací přímo ze struktury konfiguračního souboru, využívá k tomu třídu *org.w3c.dom.Document*, která je v Javě reprezentací DOM dokumentu (vnitřní struktury pro uchování struktury XML souboru).

Poslední významnou funkcí třídy Configuration je funkce *check*, která má za úkol zkontrolovat správnost vnitřního nastavení aplikace a pokud správné není, doplnit ho defaultními hodnotami. Neexistuje-li pro některou vlastnost defaultní nastavení, vrátí funkce *„check“* hodnotu *false*, aby o tom dala vědět třídě Gendok, která ji zavolala.

Celý postup prací v hlavní třídě Gendok by se dal zjednodušeně zapsat následujícím algoritmem, kde „Načti_Konfiguraci“ je výše zmíněnou funkcí *load* a „Ověř_Konfiguraci“ odpovídá výše zmíněné funkci *check*.

```
procedure Gendok
begin
   Nastav_Logovani(Logovaci_Level);
   Zpracuj_Parametry(Příkazový_Řádek);
   Configuration.Načti_Konfiguraci(Konf_Soubor);
   if (Configuration.Ověř_Konfiguraci) then
      Vytvoř_Výstup;
   Uklid_Po_Sobě;
end;
```

package gendok.create

Balík s názvem *gendok.create* obsahuje třídy pro tvorbu výstupu a mezivýstupů generátoru dokumentace databázových schémat. Všechny tyto třídy jsou potomky abstraktní třídy *OutputCreater*. Klíčovou abstraktní metodou této třídy je metoda *„create“*, jejíž tělo si jednotliví potomci definují podle svých potřeb. Jedná se o jedinou metodu, kterou hlavní třída Gendok volá, kdykoliv chce generovat výstup.

```
procedure Vytvoř_Výstup
begin
   if (Configuration.Vstupní_Formát = “schemagic”
         and Configuration.Výstupní_Formát = “pdf”) then
   begin
      var Schemagic = Configuration.Vstupní_Soubor;
      var DocBook = BookCreater.create(Schemagic);
      var FOFile = FOCreater.create(DocBook);
      var PDFFile = PDFCreater.create(FOFile);
   end;
   else ...
end;
```

Jak už bylo uvedeno, jsou k dispozici třídy *BookCreater*, *ChunkHtmlCreater*, *JustHtmlCreater*, *FoCreater*, *PdfCreater*, *PsCreater* a *RtfCreater*. Nejrozsáhlejší a jistě nejzajímavější z nich je určitě třída *BookCreater*. Ta si totiž pro tvorbu DocBook knihy musí nejprve připravit složitou převodní šablonu. Využije k tomu požadovou strukturu výstupu, kterou třída Configuration načetla z konfiguračního souboru do své vnitřní proměnné, a dále reálné soubory s těly XSL šablon. Když proces tvorby převodní šablony proběhne bez problémů, zavolá třída *BookCreater* samotnou transformaci. Výstupní XML soubor typu DocBook umístí do požadovaného výstupního adresáře.

package gendok.plugin

Jedná se o balík přednostně určený pro umístění pluginů. V základní verzi aplikace už tedy obsahuje třídy *Painter* a *SourcePrinter* a kromě nich také rozhraní, definující, jakou podobu mají pluginy mít. Každý současný i budoucí plugin by měl toto rozhraní implementovat. Rozhraní i implementované pluginy jsou podrobně popsány v samostatné kapitole 1.16, která se tomuto tématu věnuje.

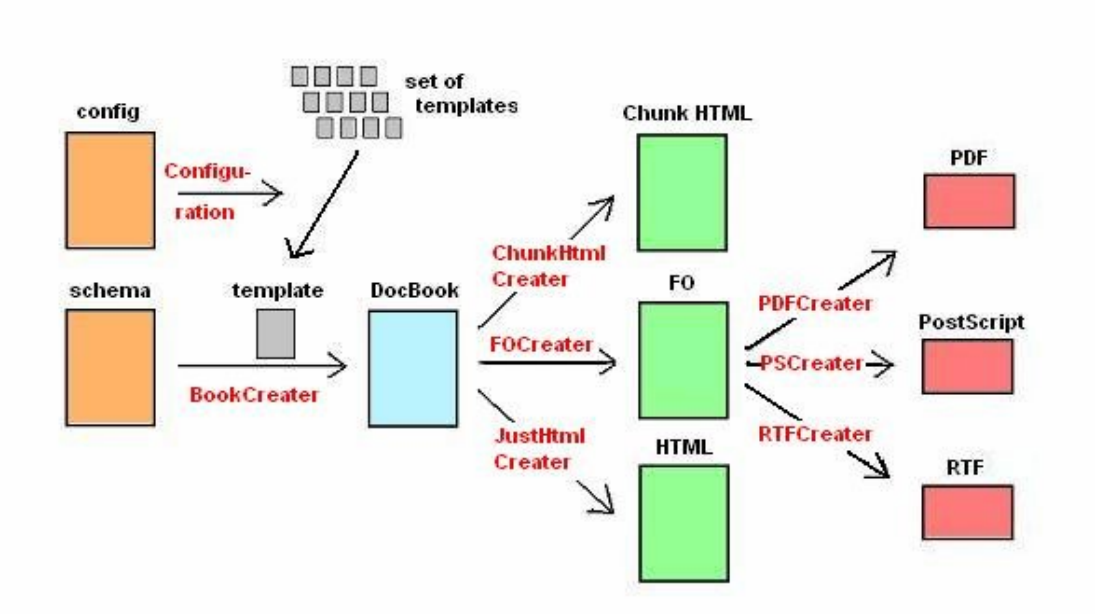
```
function BookCreater.create(File Schemagic)
begin
   var DOM = Načti_Vstup_Do_DOM(Schemagic);
   var Šablona = Vytvoř_Šablonu(StruktVýstupu, DOM);
   //Funkce Vytvoř_Šablonu zjistí i požadované pluginy

   for each X in Požadované_Pluginy do
      DOM = Zavolej_Plugin(X, DOM);
   var DocBook = Transformuj(DOM, Šablona);
   return DocBook;
end;
```

package gendok.exception

Balík pro implementaci výjimek pro aplikaci generátoru dokumentace databázových schémat. Obsahuje výjimky *FatalException, ConfigException, WrongArgumentsException*, tyto všechny jsou přímými potomky třídy *java.lang.Exception*.

O významu jednotlivých tříd aplikace dobře vypovídá následující obrázek. Staví třídy do přehledných souvislostí s operacemi, které má aplikace za úkol.



Obrázek 0.2 Význam tříd pro práci generátoru

1.14 XSLT šablony

V podadresáři „xsl“ domovského adresáře aplikace Gendok se nachází šablony - XSL styly definující převod vstupního souboru ve formátu Schemagic do mezivýstupu ve formátu DocBook. Uživatel si sám v konfiguračním souboru určí, jakým způsobem se z těchto šablon vytvoří celá velká převodní šablona.

Dále se v adresáři „xsl“ nachází soubory a adresáře se speciálním významem. V uvedených podkapitolách vysvětlím jejich přínos pro aplikaci.

1.14.1 Vícejazyčnost výstupu

V podadresáři „lang“ adresáře „xsl“ jsou uloženy slovníky podporovaných jazyků. Přiložená verze aplikace Gendok podporuje češtinu a angličtinu, v adresáři „lang“ se tedy nachází soubory „cs.xsl“ a „en.xsl“. Jméno takového souboru bez přípony se označuje jako “kód jazyka” a je definováno podle RFC1766[12]. Kód

[12] RFC – Request for Comments (žádost o komentáře), používá se pro označení standardů a dalších dokumentů; lze je získat na www.ietf.org/rfc.html

jazyka, požadovaného pro výstupní dokumentaci, musí uživatel před spuštěním aplikace uvést v konfiguračním souboru jako hodnotu vlastnosti *output-language*.

Samotný slovník jazyka se skládá z deklarací XSL parametrů, které jsou chápány jako globální proměnné pro převodní šablonu. Parametr musí obsahovat jméno a hodnotu.

```
<xsl:param name="L-TABLE" select="'Tabulka'"/>
<xsl:param name="L-VIEW" select="'Pohled'"/>
```

Uvedený příklad je výňatkem z českého slovníku „cs.xsl“. Uvádí se v něm, že proměnná L-TABLE nabývá pro češtinu hodnoty řetězce ‘Tabulka’ a proměnná L-VIEW hodnoty ‘Pohled’. V praxi to znamená, že kdekoliv bude v šablonách pro převod formátu Schemagic na formát DocBook použita proměnná L-TABLE, bude nahrazena řetězcem ‘Tabulka’. Analogicky pro ostatní deklarace proměnných.

1.14.2 Přizpůsobení šablon DocBook XSL

Podadresář „db_custom“ adresáře „xsl“ vznikl z důvodu nedostatečnosti stylů DocBook XSL 1.72.0 pro aplikaci Gendok. Bylo nutné provést tzv. “customizaci“ neboli přizpůsobení DocBooku. Ta využívá jednoduché vlastnosti XSL stylů: *Pokud je do XSL stylu šablona stejného jména začleněna vícekrát, použije se při jejím zavolání poslední z nich.* V praxi to znamená, že když se při tvorbě velké převodní šablony nejprve začlení standardní styly DocBook XSL a poté všechny naše úpravy, budou pro převod platné právě tyto úpravy.

Adresář „db_custom“ se dále člení na podadresáře podle toho, pro který výstupní formát je přizpůsobení DocBooku určeno. Pro výstup do formátu FO bylo nutné upravit původní soubor DocBook XSL stylů „block.xsl“. Přibyla v něm konkrétně tato část:

```
<!-- BEGIN CUSTOMIZATION - LBERKA, GENDOK APPL. 2007 -->
<xsl:template match="simplesect[@role='imagemap']">
</xsl:template>
<!-- END CUSTOMIZATION -->
```

Jedná se o šablonu, která říká, že narazí-li XSLT procesor v souboru DocBooku na značku *simplesect*, která má atribut *role* s hodnotou *imagemap*, tak pro takovou značku nemá vykonávat nic. Tato úprava má význam, aby se v PDF, RTF a

Postscript výstupech neobjevovali texty tzv. obrázkových map, které mají smysl pouze pro klikatelnost obrázků ve výstupní HTML dokumentaci.

Další úprava se týká souboru *lists.xsl*, kde bylo nutno opravit chybu standardních DocBook XSL 1.72.0. Styly u elementu *simplelist* s atributem *vert* neobsahovaly urceni sirky tabulky, kterou ovšem FOP 0.93 od svého vstupu bezpodmínečně požaduje. DocBookem generovaný soubor formátovacích objektů byl tak nevyhovující a oprava byla tudíž nezbytná. Konkrétní zásahy do kódu šablon jsou opět uvedeny mezi komentáři BEGIN a END CUSTOMIZATION.

Poslední úpravou ohledně výstupu do formátu FO je přizpůsobení úvodních stránek. Soubor „mytitlepages.xsl“ zajistí, aby soubory FO, PDF, RTF, či Postscript, neobsahovaly mezi úvodní stranou a obsahem prázdnou stránku.

Přizpůsobení stylů DocBooku pro výstupní formát HTML nebo vícestránkové HTML se týká pouze souboru „graphics.xsl“. Tady se jedná o úpravy nutné k zajištění klikatelnosti grafů, vygenerovaných aplikací Graphviz, a následně zahrnutých do HTML stránek. Konkrétní zásahy do kódu šablon jsou opět uvedeny mezi komentáři BEGIN a END CUSTOMIZATION.

1.14.3 Nastavení DocBook stylů

K nastavení vlastností pro DocBook styly slouží soubory „db2html.xsl“, „db2chunk.xsl“ a „db2fo.xsl“ v závislosti na požadovaném výstupu transformace z formátu schemagic. Tyto vlastnosti jsou definovány a dobře popsány v dokumentaci k XSL stylům DocBooku. Příslušné kapitoly jsou označeny jako “HTML Parameter Reference” a “FO Parameter Reference”.

Při tvorbě výsledné XSLT šablony pro převod formátu DocBook na FO, HTML nebo vícestránkové HTML se nejdříve do šablony vloží klasické XSL styly DocBooku a pak teprve tyto soubory. Pro vysvětlení a lepší představu uvádím ukázku z defaultního souboru „db2chunk.xsl“ aplikace Gendok.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:include href="../xsl/db_custom/chunk/graphics.xsl"/>

   <xsl:param name="chunk.quietly">1</xsl:param>
   <xsl:param name="use.id.as.filename">1</xsl:param>
   <xsl:param name="toc.section.depth">2</xsl:param>
   <xsl:param name="section.autolabel" select="1"/>
   <xsl:param name="section.autolabel.max.depth" select="2"/>
</xsl:stylesheet>
```

První parametr udává, že byla provedena “customizace” základních XSL stylů DocBooku a že je tedy třeba vložit soubor s touto “customizací”, konkrétně soubor „graphics.xsl“. Další parametry pak XSLT procesoru radí, že nemá vypisovat

chybová hlášení definovaná přímo ve stylech, že má použít identifikátory objektů jako názvy souborů pro vícestránkové HTML, že v obsahu má uvádět pouze kapitoly a sekce první a druhé úrovně, atp.

V souborech „db2html.xsl“ a „db2chunk.xsl“ není potřeba uvádět, jakou má mít výstup podobu – barvy, odsazení textu, font písma, velikost a sklon fontu apod, protože to vše na závěr zajistí připojený CSS[13] styl. Naproti tomu je potřeba tyto požadované vlastnosti nastavit v souboru „db2fo.xsl“, neboť k výstupu FO se žádný další styl nepřipojuje.

1.14.4 Tvorba DOT souboru pro Graphviz

Soubor jazyka DOT, který se předá aplikaci Graphviz, aby vytvořila graf vazeb mezi objekty, je potřeba nejdříve nějak vytvořit. Informace o tom, jak má tento graf vypadat, je obsažena ve vstupním souboru aplikace – ve formátu schemagic. Plugin Painter, který má tvorbu grafů za úkol, musí proto mimo jiné zavolat XSLT transformaci, která z formátu schemagic vytvoří požadovaný DOT soubor. Více se o chování pluginu Painter dočtete v části 1.16.3.

Převodní šablona pro vytvoření DOT souboru je umístěna v podadresáři „make-dot“ adresáře „xsl“ a nese jméno „for-each-table.xsl“, neboť grafy vazeb se týkají propojení přes primární a cizí klíče tabulek.

1.15 Přizpůsobení Schemagicu

Zajištění tvorby dokumentace k databázovému schématu se neobešlo bez drobných zásahů do aplikace Schemagic. Kromě úprav konfiguračních souborů aplikace došlo i ke změně jejích zdrojových kódů. Následující řádky v bodech odůvodňují a popisují tyto úpravy.

Konce řádků ve zdrojových kódech

Nepozměněná verze aplikace Schemagic získávala z určené databáze zdrojové kódy triggerů, procedur, funkcí, typů, specifikace a těla balíků. Na svém výstupu je pak ukládala stejně jako všechny ostatní požadované informace, tedy coby hodnoty atributů „value“ elementů „property“. Protože ale příslušné zdrojové kódy obsahovaly oddělovače řádků, byl tento postup nepřijatelný. Výstupem Schemagicu je XML soubor, přičemž o XML je známo, že v hodnotách atributů nepovoluje jiný „bílý znak“ než mezeru. Není dokonce povoleno ani více mezer za sebou.

[13] CSS styl, kaskádový styl – definuje vzhled HTML dokumentů.

Jakmile se tedy aplikace Gendok pokusila chápat výstupní soubor Schemagicu jako XML dokument (konkrétně se jedná o načtení vstupního souboru do vnitřní DOM struktury aplikace Gendok), došlo k automatickému nahrazení všech oddělovačů řádků za mezery. Tomuto chování parseru, který dokument načítá, nebylo možno zabránit. Konce řádků přitom nebylo možné zpětně odhadnout žádnou formou heuristiky, protože jejich ztrátou došlo zároveň ke ztrátě důležitých informací, kde ve zdrojových kódech končí komentáře.

Ukládání oddělovačů řádků do hodnot atributů XML souboru bylo tedy považováno za chybné chování Schemagicu a došlo k úpravě jeho zdrojových kódů. Konkrétně se jedná o úpravy ve třídách:

org.schemagic.plugin.database.load.impl. JdbcSqlDbObjectLoader

org.schemagic.plugin.database.load.impl. SqlConcatPostLoader

Do první uvedené třídy bylo přidáno nahrazování oddělovačů řádků speciální hodnotou --EOLN--, tak aby se konce řádků ve zdrojových kódech objektů skutečně projevily ve výstupu Schemagicu. Příslušná úprava je v souboru uvedena komentářem */* uprava L.B. 14.05.2007 */*.

V druhé uvedené třídě byla provedena stejná změna kódu a je také označena příslušným komentářem.

Problém s triggery

Problém s exportem triggerů ve Schemagicu se projevoval jen pro Oracle verzi 8. Jednalo se o to, že hodnoty vlastností WHEN_CLAUSE a TRIGGER_BODY, zapisované do výstupu Schemagicu, byly ukončeny znakem '\0'. To způsobovalo, že se výpis do výstupního souboru zastavil na tomto znaku, a ne až za ukončovací lomenou závorkou XML značky. Opět muselo dojít k úpravě třídy

org.schemagic.plugin.database.load.impl.JdbcSqlDbObjectLoader.

Třída nově obsahuje funkci *RTrim*, která svému argumentu zprava odstraní znaky '\0', CR, LF a mezery. Tato funkce je volána při výpisu hodnoty atributu *value* elementu *property* do výstupního souboru.

Úprava konfiguračních souborů

Konfigurační soubory Schemagicu byly upraveny z toho důvodu, aby jeho výstup obsahoval celou množinu informací, kterou aplikace Gendok vyžaduje pro vytváření dokumentací databázových schémat. Konkrétně byly provedeny úpravy v souborech

„./def/object/objects.xml“

„./def/object/ora_objects.xml“

„./def/machina/oracle.machine.xml“

Upravená podoba těchto souborů je k dispozici společně se Schemagicem na přiloženém instalačním CD aplikace Gendok.

1.16 Pluginy

Rozhraní *Gendok.plugin.PluginInterface* definuje, jak má vypadat vnitřní struktura pluginů pro aplikaci Gendok. Její součástí jsou už první pluginy pojmenované *Painter* a *SourcePrinter*. První má na starost umožnit aplikaci generovat grafy, druhý má zpřehlednit výpis zdrojových kódů databázového schématu.

1.16.1 Rozhraní pluginu

Rozhraní zásuvného modulu neboli pluginu je stručně řečeno výpis metod, které musí implementovat každý plugin pro danou aplikaci. Původní program potom přítomnost těchto metod v pluginu předpokládá a automaticky je volá.

Rozhraní pluginu pro aplikaci Gendok vypadá následovně:

```
interface PluginInterface
begin
   procedure run( Document inputDocument,NodeList args );
   procedure run4each( Document doc,String object,NodeList args );
function getNameAndVersion() returns String;
end;
```

Ačkoliv se nejedná o zápis ve skutečném programovacím jazyce, je z něj dobře patrné, že každý plugin bude implementovat metody „run“, „run4each“ a „getNameAndVersion“. První z nich byla vytvořena pro případ, že bude plugin zavolán, aby vykonal svou práci pro celý vstupní dokument formátu schemagic bez ohledu na objekty v něm. V takovém případě nepotřebuje plugin znát, ke kterému typu objektu databázového schématu se má jeho práce vztahovat. Aplikace Gendok zavolá první metodu pluginu v případě, že bylo volání pluginu definováno v konfiguračním souboru jinde než uvnitř elementu <for-each>.

Pro opačný případ je připravena metoda „run4each“, která dostane jméno objektu jako řetězec ve svém druhém parametru. Řetezec jména objektu odpovídá ve vstupním souboru hodnotě atributu „family“ elementu „object“. V konfiguračním souboru je jméno tohoto objektu rovno hodnotě atributu „elem“ elementu „for-each“, který je přímým předkem volání pluginu.

Prvním parametrem obou metod je odkaz na DOM reprezentaci vstupního souboru aplikace. Parametr je předán odkazem, plugin tedy má možnost strukturu tohoto dokumentu pozměnit. Posledním parametrem je pak seznam elementů „param“, zavěšených v konfiguračním souboru pod voláním pluginu. Tyto elementy definují názvy a hodnoty parametrů pro konkrétní pluginy.

Třetí metoda rozhraní *PluginInterface* je určena pouze k výpisu informace, jaký je název a verze konkrétního Plutonu. Aplikace Gendok tuto informaci loguje.

1.16.2 Plugin SourcePrinter

Implementace pluginu SourcePrinter nebyla cílem projektu a nutno říct, že je stále ve vývoji. Jeho první verzi však aplikace Gendok obsahuje.

Smyslem pluginu je ze vstupního souboru zdrojové kódy jednotlivých objektů databázového schématu a proměnit je ze stavu prostého textu do podoby s formátovacími instrukcemi. Touto úpravou je myšleno doplnění informací o barvě, stylu a velikosti jednotlivých částí kódu. Zároveň by měl být plugin schopen zajistit správné lámání řádků a levé odsazení tam, kde je ho zapotřebí.

Generátor dokumentace volá plugin SourcePrinter prostřednictvím metody run4each, neboť požaduje úpravu zdrojových kódů vždy konkrétního typu objektu v databázovém schématu.

Plugin je z konfiguračního souboru volán přibližně následovně:

```
<call-plugin name="gendok.plugin.SourcePrinter">
    <param name="object-family" value="view"/>
</call-plugin>
```

Parametr *object-family* udává, který konkrétní typ podobjektu má mít upraven zdrojový kód. Povolenými hodnotami tohoto parametru jsou v současnosti: *ora_table_trigger*, *view*, *procedure*, *function*, *ora_type_body*, *ora_package* a *ora_package_body*.

SourcePrinter využívá pro obarvení částí kódu neoficiální seznam klíčových slov jazyka PL/SQL. Řetězce, konstanty a operátory jazyka v zatím vytvořené verzi rozpoznávat neumí.

1.16.3 Plugin Painter

Třída *Gendok.plugin.Painter* je pluginem, není tedy pro aplikaci Gendok „životně“ důležitá, ale její význam je přesto velký. Společně s aplikací Graphviz, kterou využívá, dává generátoru dokumentace sílu vytvářet grafy a zpřehledňovat tak

výslednou dokumentaci databázového schématu. Při svém zavolání přes metodu run4each od uživatele vyžaduje předání dvou klíčových parametrů:

dotxslt

Jeho hodnota udává, kde se nachází XSL šablona k vytvoření DOT souboru pro aplikaci Graphviz. Aplikace Gendok umí vytvářet zatím pouze grafy vazeb mezi tabulkami a příslušná šablona se nachází v „./xsl/make-dot/for-each-table.xsl“.

graphviz-home

Jeho hodnota udává adresář, v němž se nachází hlavní aplikace Graphvizu DOT.EXE (případně DOT v systému Linux). Cesta může být buď absolutní nebo relativní z domovského adresáře aplikace Gendok.

Ukázkové volání pluginu vypadá v konfiguračním souboru následovně:

```
<call-plugin name="gendok.plugin.Painter">
   <param name="dotxslt"
      value="./xsl/make-dot/for-each-table.xsl"/>
   <param name="graphviz-home" value="../graphviz/bin"/>
</call-plugin>
```

Plugin Painter má zajímavější vnitřní strukturu než SourcePrinter, jeho činnost naznačím výpisem zjednodušeného algoritmu.

```
procedure Painter.run4each( .. arguments .. )
begin
   var adresar = Zjisti_Adresar_Graphvizu(arguments);
   var sablona = Zjisti_Sablonu(arguments);
   var objekt = Zjisti_Jmeno_Objektu(arguments);
   var vstupniDok = Zjisti_Vstupni_Dokument(arguments);
   for each object X in vstupniDok
   begin
      if (X = objekt) then
         ZavolejGraphViz(adresar);
   end;
end;
```

1.17 Problém s názvy objektů

Během implementace projektu byl zjištěn problém s českými názvy databázových objektů. Ty aplikace Gendok využívá mimojiné jako součást identifikátorů kapitol a sekcí v DocBook dokumentaci. XSL styly DocBooku pro

tvorbu výstupní dokumentace jsou ale standardně nastaveny tak, aby při tvorbě vícestránkového HTML používaly jako jména souborů právě tyto identifikátory sekcí. Tímto postupem mohou vzniknout soubory s českými znaky v názvech.

Při prohlížení takto vytvořených dokumentací se ukázalo, že s nimi jednotlivé webové prohlížeče zacházejí nejednotně.

Problém hlásil rovněž pomocný vizualizační nástroj Graphviz.

Funkce ke kódování řetězců je v XSLT k dispozici až od verze 2.0, která je ještě velice mladá a aplikace Gendok ji nevyužívá. S pravděpodobným budoucím přechodem aplikace Gendok na XSLT 2.0 bude problém vyřešen.

Instalace a konfigurace

Vzhledem ke skutečnosti, že popisovaná aplikace využívá ke své činnosti řadu pomocných programů, nemůže se popis správné instalace a konfigurace týkat jen jí samotné. Návody k práci s jednotlivými aplikacemi sice většinou lze objevit na internetu, přesto ale vzhledem ke svým zkušenostem cítím potřebu některé věci zdůraznit nebo podrobněji vysvětlit.

Tato kapitola si bere za cíl poskytnout běžnému uživateli tolik informací a rad, aby byl bez výraznějších problémů schopen generátor dokumentace databázových schémat nainstalovat a správně nakonfigurovat. Nejdříve se podívejme na pomocné aplikace, které jsou však pro chod samotného generátoru nezbytné.

V části 1.23.2 je popsána instalace kompletního balíčku, zahrnujícího aplikaci Gendok i pomocné programy. Tento balíček se nachází na přiloženém CD a má uživateli zjednodušit zprovoznění aplikace.

1.18 Instalace FOP 0.93

Význam tohoto programu byl objasněn v podkapitole 1.7. Stručně řečeno má FOP za úkol vygenerovat PDF, RTF nebo Postscript výstup z XML souboru formátovacích objektů. Uvažujeme použití jeho verze 0.93.

Aplikace FOP 0.93 je psána v jazyce Java a její součástí je kromě dokumentace a pomocných *.jar* knihoven především hlavní spustitelný *.jar* archív a konfigurační soubor, který nese povětšinou označení „fop.xconf“. Jelikož aplikace Gendok FOP využívá, je třeba při volání tvorby dokumentace v některém z formátů PDF, Postscript, RTF přidat spouštěcí JAR soubor FOPu a všechny knihovny z podadresáře „lib“ domovského adresáře FOPu do Java proměnné CLASSPATH. Jedině tak budou balíčky Fopu k dispozici pro volání této aplikace přímo z Java kódu.

Trochu těžší je u aplikace FOP 0.93 zprovoznit podporu českého jazyka pro jím vygenerované dokumenty. Nebyl jsem ale první, kdo tento problém řešil, a tak jsem na internetu objevil stručný návod a soubory, které je potřeba k instalaci FOPu 0.93 připojit. Na stránkách Jiřího Koska [3] je volně ke stažení ZIP archív „fop-cs2.zip“. V případě, že by tento odkaz nebyl do budoucna k dispozici, umístil jsem tento archív i na instalační CD aplikace.

V ZIP archívu se nacházejí adresáře „conf“ a „lib“. Archív je potřeba rozbalit do domovského adresáře aplikace FOP. V jeho podadresáři „lib“ touto akcí přibude

soubor „fop-hyph.jar“ a v podadresáři „conf“ se objeví metrické XML soubory a nový konfigurační soubor FOPu s názvem „myfop.xconf“. Tento konfigurační soubor je stejný jako defaultní konfigurační soubor FOPu, jen je navíc obohacen o odkazy na metriky českých fontů. Na prvním a druhém řádku jsou takto deklarovány XML entity:

```
<!ENTITY fop.home "file:///d:/Projects/gendok/progs/fop-0.93/">
<!ENTITY fonts.dir "file:///c:/WINDOWS/Fonts">
```

Je potřeba, aby hodnota první entity odpovídala URI domovského adresáře FOPu, hodnota druhé entity musí být URI umístění českých fontů v systému.

Nyní je FOP připraven pro podporu českého jazyka, při jeho spuštění je ale potřeba uvést, že jeho konfiguračním souborem je *<fop-home>/conf/myfop.conf*. Cestu k tomuto souboru je nutné určit jako hodnotu vlastnosti *fop-cfg-file* v konfiguračním souboru aplikace Gendok.

Tímto postupem zajistíme podporu českých znaků pro fonty Arial, Courier a Times New Roman. V případě, že by to uživateli nestačilo, je potřeba, aby si vlastnoručně zkompiloval fonty podle vlastního výběru. K této kompilaci poslouží opět aplikace FOP 0.93. Přesný návod je uveden v dokumentaci aplikace FOP. Stačí otevřít HTML dokument *<fop-home>/docs/0.93/fonts.html*, kde je postup popsán. Kompilací vzniknou XML soubory stejné, jaké jsme rozbalili z archívu. Je opět potřeba cestu k nim uvést v konfiguračním souboru FOPu.

Pro zjednodušení jsem na přiložené instalační CD umístil i verzi FOPu 0.93 se zprovozněnou podporou češtiny. ZIP archív s takto upraveným programem nese označení „fop-0.93-bin-jdk1.4-CS.zip“. Původní verze bez češtiny se nachází v archívu „fop-0.93-bin-jdk1.4.zip“ a je ji také možné stáhnout z domovských stránek produktu[14].

1.19 Instalace XSLT procesoru

Saxon a Xalan jsou procesory, které mají být využity k XSLT transformacím, například z formátu DocBook do formátu HTML. Vybral jsem pro tuto aplikaci verze Saxon 6.5.5 a Xalan 2.7.0.

Saxon a Xalan jsou podstatně jednoduššími aplikacemi než FOP a tak je jejich instalace a použití jednodušší. Skládají se z jediného JAR archívu,

[14] http://xmlgraphics.apache.org/fop/

pojmenovaného v případě Saxonu „saxon.jar“ a v případě Xalanu „xalan-2.7.0.jar“. Tyto JAR soubory je nutno přidat do Java proměnné CLASSPATH při každém volání aplikace Gendok. Soubor „xalan-2.7.0.jar“ je navíc automatický přítomný v podadresáři „lib“ domovského adresáře aplikace FOP 0.93.

Současná podoba aplikace vyžaduje přítomnost tzv. rozšíření pro XSLT procesor a proto je navíc nutné připojit k aplikaci soubor „saxon65.jar“, případně „xalan27.jar“. Nikoliv však oba dva najednou. Uživatel musí být předem rozhodnutý, který z XSLT procesorů hodlá pro volání aplikace Gendok využít.

Zmíněné soubory rozšíření pro XSLT procesor jsou automaticky přítomny v podadresáři „extensions“ domovského adresáře DocBook XSL stylů.

Instalační CD obsahuje v adresáři „Jednotlive_aplikace“ podadresář pro Xalan verze 2.7.0 i pro Saxon 6.5.5. Oba produkty jsou také volně ke stažení na internetu[15].

1.20 Aplikace Graphviz

Graphviz je pluginem Painter aplikace Gendok využíván k tvorbě grafů vazeb mezi tabulkami. Konkrétně je zapotřebí pouze jediný soubor, a to „dot.exe“ v případě instalace pod operačním systémem Windows a „dot“ v případě instalace pod některou z distribucí Linuxu.

Aplikace Gendok předpokládá využití GraphVizu ve verzi 2.8.

Kompaktní disk, přiložený k bakalářské práci, obsahuje pouze instalační archív pro operační systém Microsoft Windows. Uživatelé Linuxu by si měli stáhnout Graphviz prostřednictvím programu pro správu balíků. Alternativou je stáhnout si Graphviz přímo z domovských stránek produktu[16].

Požaduje-li uživatel, aby se při spuštění aplikace Gendok zavolal i plugin Painter, potom je potřeba uvést v konfiguračním souboru, kde se nainstalovaný program *dot* nachází. Konkrétně se tato URI předá jako hodnota parametru *graphviz-home* v definici volání pluginu Painter. Nastavení pluginu je popsáno v části 1.16.3 tohoto dokumentu.

[15] Saxon - http://saxon.sourceforge.net, Xalan - http://xalan.apache.org/

[16] www.graphviz.org

1.21 Styly DocBook XSL

Aplikace Gendok vyžaduje přítomnost XSL stylů DocBooku ve verzi 1.72.0. Tyto styly jsou k dispozici na přiloženém kompaktním disku. Konkrétně v podadresáři „docbook-xsl-1.72.0“ adresáře „Jednotlive_aplikace“. Lze je také stáhnout z adresy http://sourceforge.net/projects/docbook/. Nutno však podotknout, že se jedná o styly pro DocBook 4 nikoliv pro nově vyvíjený DocBook 5.

Instalace stylů spočívá v pouhém určení domovského adresáře a nakopírování obsahu archívu „docbook-xsl-1.72.0.zip“ do tohoto adresáře. URI domovského adresáře je nutno přidat jako hodnotu vlastnosti *docbook-xsl-home* v konfiguračním souboru aplikace Gendok.

Stáhnout z uvedené adresy nebo zkopírovat z přiloženého disku lze rovněž dokumentaci k XSL stylům DocBooku. Ta je ve formě vícestránkového HTML a nachází se v archívu „docbook-xsl-doc-1.72.0.zip“.

1.22 Instalace aplikace Schemagic

Aplikace Schemagic je rovněž přítomna na přiloženém kompaktním disku. Konkrétně v adresáři „Jednotlive_aplikace/schemagic”. Jedná se o verzi s úpravami, které jsou popsány v části 1.15 tohoto dokumentu. Schemagic obsahuje spouštěcí dávkový soubor pro operační systém Windows, archív „schemagic.jar“, pomocné knihovny v adresáři „lib“, dokumentaci, zdrojové kódy a především konfigurační soubory, které jsou pro jeho použití nezbytné.

Pro správné použití je nejdříve potřeba vytvořit (nebo editovat již existující) schéma v adresáři „schemas“. Zde je potřeba definovat, ze které databáze chceme získat data a pod kterým loginem a heslem se přihlásíme. Poté je možno aplikaci spustit. Je však potřeba, aby dávkový soubor „r.bat“ obsahoval odkaz na námi definované schéma.

Dalším významným konfiguračním souborem je soubor „./def/machine/oracle.machine.xml“. Zde je možno určit objem dat, které Schemagic získá z databázového schématu. Ke každému objektu ve výstupním schématu je tu definován SQL SELECT, jímž se informace o daném objektu získají.

Parametry příkazové řádky aplikace Schemagic jsou *--plugin* a *--log*. Za prvním z nich je očekáváno jméno pluginu. Druhý z nich je standardním mechanismem logování v Javě. Hodnota následující za *--log* by měla být jednou z hodnot definovaných podle *java.util.logging.Level*. Typické volání aplikace Schemagic pro získání informací z databázového schematu je následující:

```
java -jar schemagic.jar --plugin org.schemagic.plugin.schema
schemas/gendok.xml data/gendok.loaded.xml --log all
```

Soubor „schemas/gendok.xml" z tohoto příkladu je konfiguračním souborem aplikace, obsahujícím definici připojení k databázi a schématu, z něhož má program získat informace. Soubor „data/gendok.loaded.xml" je výstupním souborem aplikace.

1.23 Instalace aplikace Gendok

Z přiloženého kompaktního disku se nabízejí dvě varianty instalace. Aplikace je zde přítomna v kompletním balíku i se všemi pomocnými programy. Je tu ale rovněž jako samostatná aplikace, která vyžaduje správné nastavení cest k také samostatně instalovaným pomocným programům. Postup instalace a konfigurace pro tyto dva případy popisují následující podkapitoly.

1.23.1 Samostatná aplikace

Instalace samostatné aplikace Gendok, generátoru dokumentace k databázovému schématu, spočívá ve zkopírování archívu „Jednotlive_aplikace/gendok-1.0/gendok-1.0.zip" z přiloženého CD na lokální disk. Dále je zapotřebí jeho rozbalení a správné nastavení cest k programům, které aplikace využívá.

Především je nutné upravit podobu souboru „run.bat" - případně „run" pro uživatele Linuxu. Na prvních řádcích tohoto souboru je třeba určit domovské adresáře samostatně instalovaných programů. Týká se to Javy, FOPu, Saxonu a XSL stylů DocBooku. Domovský adresář Xalanu není třeba nastavovat ani v případě, že ho uživatel zvolí za XSLT procesor, který chce využít. Xalan je totiž distribuován společně s aplikací FOP 0.93 a rozšíření pro něj je obsaženo v adresáři „extensions" XSL stylů DocBooku.

Rovněž není potřeba uvádět domovský adresář aplikace Graphviz. Cesta k ní se předává jako parametr přímo pluginu Painter, konkrétně v konfiguračním souboru při definici jeho volání. Přesná podoba tohoto volání je uvedena v 1.16.3.

K dosažení správného nastavení aplikace je dále potřeba upravit cestu ke konfiguračnímu souboru, který bude aplikací využíván.

Níže je uveden příklad nastavení zmíněných parametrů pro soubor „run.bat". Pro soubor „run" je v Linuxu postup analogický – jen s rozdílnou syntaxí shell-skriptů.

```
set JAVA_HOME=C:\PROGRA~1\Java\jdk1.5.0_06\bin\java
set FOP_HOME=C:\ PROGRA~1\progs\fop-0.93
set SAXON_HOME=C:\ PROGRA~1\saxon-6.5.5
set DOCBOOK_XSL_HOME=C:\ PROGRA~1\docbook-xsl-1.72.0
set CONFIG_FILE=.\conf\myconfig.xml
set XSLT_PROCESOR=xalan
```

Poslední uvedený řádek určuje, zda bude aplikace využívat XSLT procesoru Xalan 2.7.0 nebo Saxon 6.5.5. Hodnotou proměnné XSLT_PROCESOR může být buď slovo „xalan“ nebo „saxon“.

O zahrnutí potřebných knihoven do Java proměnné CLASSPATH se postarají přímo spouštěcí skripty aplikace. Nutným předpokladem pro tuto skutečnost je ale použití předepsaných verzí pomocných aplikací – tedy FOPu 0.93, Saxonu 6.5.5, Xalanu 2.7.0 a XSL stylů DocBooku ve verzi 1.72.0.

Rovněž je zapotřebí mít k dispozici Javu ve verzi alespoň 1.5.0.

1.23.2 Kompletní instalace

Výhodou instalace kompletního balíku je usnadnění konfigurace generátoru. S aplikací Gendok se na určené místo zkopírují i všechny pomocné programy a generátor se na ně poté odkazuje za pomoci relativních cest. Jeho přednastavená konfigurace by proto měla být až na drobné výjimky postačující.

Instalační balík se na kompaktním disku nachází v ZIP archívu „Kompletni_instalace/komplet.zip”. Od struktury samostatné aplikace Gendok se liší pouze přítomností podadresáře “app”. Ten v sobě sdružuje všechny pomocné aplikace. Soubor “run.bat” (nebo “run”) se na ně automaticky odkazuje, není proto potřeba provádět v něm tolik změn jako při instalaci samostatné aplikace Gendok.

Je potřeba upravit nebo alespoň zkontrolovat dvě proměnné v úvodu spouštěcího skriptu “run.bat”. Domovský adresář Javy a cestu ke konfiguračnímu souboru aplikace Gendok.

```
set JAVA_HOME=C:\PROGRA~1\Java\jdk1.5.0_06\bin\java
set CONFIG_FILE=.\conf\myconfig.xml
```

Defaultním XSLT procesorem balíku je Saxon 6.5.5, k dispozici je ale opět i Xalan 2.7.0, jehož volbu můžeme určit nastavením proměnné XSLT_PROCESOR na hodnotu “xalan”.

Uživatelská příručka

Tato definuje postupy, jak generátor dokumentace databázových schémat správně využívat a jak mu sdělit, aby se choval tak, jak právě požadujeme. Zároveň dává uživateli instrukce, kde se nacházejí jednotlivé prvky aplikace.

1.24 Umístění součástí

Domovský adresář aplikace Gendok obsahuje standardně sedm podadresářů a spouštěcí skripty s názvem „run.bat“ pro operační systém Windows, respektive „run“ pro distribuce Linuxu.

V podadresáři „src“ se nachází zdrojové kódy aplikace, v podadresáři „classes“ z nich zkompilované *.class* soubory. Podadresář „doc“ obsahuje dokumentaci k aplikaci Gendok. Ta se skládá především z této práce a ze souborů vygenerovaných aplikací *javadoc*.

Dalším přítomným podadresářem je „css“, který je úložištěm pro nadefinované CSS styly. Ty mají za úkol úpravu vzhledu výstupní HTML dokumentace. V případě rozšíření aplikace o nový CSS styl je vyžadováno jeho umístění do tohoto adresáře, neboť právě v něm aplikace očekává přítomnost uživatelem určených stylů.

Podadresář „pix“ obsahuje všechny obrázky (kromě dynamicky generovaných grafů vazeb), které jsou vyžadovány ke správnému zobrazení výstupních dokumentací. Podadresáře adresáře „pix“ mají stejnou strukturu jako umístění XSLT šablon pro vytvoření meziformátu DocBook. Ke každé XSLT šabloně náleží jeden adresář pod adresářem „pix“, ve kterém jsou uloženy obrázky. Tyto obrázky jsou vyžadované pro správné zobrazení sekce DocBooku, vzniklé voláním dané šablony.

Podadresář „xsl“ domovského adresáře aplikace Gendok obsahuje všechny XSL šablony pro XSLT transformace, které aplikace volá. V adresářích s názvy „function“, „object“, „package“, „procedure“, „table“, „type“ a „view“ se nachází jednotlivé šablony pro transformaci formátu Schemagic na formát DocBook. Jména adresářů odpovídají objektům, pro něž dané šablony generují výstup.

Dále jsou v adresáři „xsl“ zahrnuty slovníky pro podporu vícejazyčnosti výstupní dokumentace - v podadresáři „lang“. V „make-dot“ jsou obsaženy XSL šablony, definující tvorbu DOT souborů pro vizualizační nástroj graphviz, který

generuje grafy vazeb mezi objekty. V podadresáři „db-custom“ se nachází soubory přizpůsobení DocBooku, o nichž je více uvedeno v části 1.14.2 tohoto dokumentu.

O významu souborů „db2chunk.xsl“, „db2fo.xsl“ a db2html.xsl“ je pojednáno v kapitole 1.14.3.

Posledním podadresářem domovského adresáře aplikace Gendok je „conf“, v němž je očekávána přítomnost konfiguračního souboru a DTD, definujícího jeho strukturu.

1.25 Přizpůsobení aplikace

Jak je uvedeno v části 1.8, nastavení aplikace je prováděno prostřednictvím příkazové řádky a konfiguračního souboru. Nastavení parametrů příkazové řákdy je nutno provést ve spouštěcím skriptu aplikace („run.bat“ případně „run“). Kromě parametrů popsaných v části 1.23, které se týkají instalace aplikace, je zde možnost určit hodnotu i dalším parametrům. Podívejme se na příklad jejich nastavení (ukázka je ze spouštěcího skriptu „run.bat“).

```
set INPUT_FORMAT=schema
set OUTPUT_FORMAT=rtf
set OUTPUT_DIR=..\results\::FORMAT::\test5
set INPUT_FILE=..\input\loaded_schema.xml
set LOG_LEVEL=ALL
```

Jedná se o proměnné, které se přímo týkají podoby a umístění výstupní dokumentace. Následuje popis jejich významu i přípustných hodnot.

INPUT_FORMAT

Udává název vstupního formátu pro aplikaci Gendok. Povolenými hodnotami tohoto parametru jsou *schema* pro vstup ve formátu Schemagic, *docbook* a *fo*. Při nastavení parametru se vyžaduje se, aby hodnota níže popsaného parametru INPUT_FILE ukazovala na souboru typu INPUT_FORMAT.

OUTPUT_FORMAT

Určuje výstupní formát pro aplikaci Gendok. Přípustnými výstupními formáty jsou *docbook*, *html*, *chunk*, *fo*, *pdf*, *ps* a *rtf*.

OUTPUT_DIR

Jeho hodnotou je cesta k požadovanému výstupnímu adresáři, která navíc může obsahovat některý ze dvou speciálních symbolů ::FORMAT:: a ::TIMESTAMP::. První z nich bude automaticky nahrazen malými písmeny jména požadovaného výstupního formátu, druhý pak časovou známkou ve tvaru YYMMDD-HHmmSS, například 070531-235959 pro vteřinu před půlnocí dne 31. května 2007.

INPUT_FILE

Hodnota tohoto parametru určuje cestu ke vstupnímu souboru aplikace. Formát vstupního souboru by měl odpovídat formátu, specifikovanému hodnotou parametru INPUT_FORMAT.

LOGLEVEL

Udává úroveň logování generátoru. Přípustnými hodnotami jsou řetězce OFF, SEVERE, INFO, WARNING, FINE, FINER, FINEST, ALL. Uvedené hodnoty jsou seřazeny podle množství výpisů, které bude běžící aplikace vypisovat na konzoli – od žádného po největší.

Jako parametr je rovněž možno určit cestu ke konfiguračnímu souboru, který obsahuje podrobnější nastavení pro aplikaci. Díky tomuto parametru je možné, aby si uživatel nadefinoval více konfiguračních souborů podle svých potřeb a následně měnil jen tento parametr - tedy informaci o tom, který z nich má aplikace použít.

```
set CONFIG_FILE=.\conf\myconfig.xml
```

Aplikaci si lze pochopitelně přizpůsobit i změnami v samotném konfiguračním souboru. Tyto změny jsou však až druhořadé – přednost má vždy určení parametru na příkazové řádce před jeho určením v konfiguračním souboru. Přehled parametrů, nastavitelných v konfiguračním souboru, popisuje část 1.12.2 tohoto dokumentu.

1.26 Spuštění aplikace

Aplikace Gendok se spouští spouštěcími skripty – “run.bat” pro operační systém Windows a “run” pro distribuce Linuxu. Vstup aplikace hledá na adrese

definované uvnitř skriptu hodnotou parametru INPUT_FILE, výstup umisťuje do adresáře určeného uvnitř skriptu hodnotou parametru OUTPUT_DIR.

Závěr

Cílem bakalářského projektu bylo navrhnout a implementovat generátor dokumentace databázových schémat. Projekt byl v tomto ohledu úspěšný. Podařilo se vypracovat podrobnou analýzu, sestavit seznam požadavků, vypracovat postup řešení a toto řešení následně úspěšně naimplementovat.

Vzniklá aplikace se stala společně s aplikací Schemagic jednoduchým a sympatickým řešením, jak zdokumentovat databázové schéma.

V budoucnosti by mohl být projekt dále rozšiřován. Pomocí aplikace Schemagic by mělo být snadné doplnění podpory i pro další typy databázových serverů. Samotná generovaná dokumentace k databázovému schématu by mohla v budoucnu obsahovat i jiné druhy grafů než v současnosti implementované, například znázornění hierarchie typů. Dále by bylo vhodné uživateli nabízet SQL kódy, kterými je možné popisované objekty vytvořit v jiné prázdné databázi. Mezi další možná rozšíření patří rovněž přidání dalšího podporovaného vstupního formátu – popisovače schematu pro Schemagic, čímž by bylo možné schéma načíst v rámci jednoho kroku s generováním dokumentace.

Jak ukázaly testy stávajícího generátoru na řadě schémat od jednodušších až po reálný firemní informační systém se stovkami tabulek, pohledů a dalších databázových objektů, generátor je již nyní plně použitelný pro účely, ke kterým byl vytvořen.

Literatura

[1] Nagy, Miroslav: Synchronization of relational DB schemas, Master thesis, Charles University, Prague, 2005

[2] Přispěvatelé Wikipedie: Java. Wikipedie, Otevřená encyklopedie. http://cs.wikipedia.org, 2006

[3] Kosek, Jiří: Podpora češtiny pro FOP, http://www.kosek.cz/sw/fop, 2007

[4] Stayton, Bob: DocBook XSL: The Complete Guide, http://www.sagehill.net/book-description.html, 2005

[5] Kosek, Jiří: DocBook, Stručný úvod do tvorby a zpracování dokumentů, http://www.kosek.cz/xml/db/index.html, 2007

[6] Kosek, Jiří: XSLT v příkladech, http://www.kosek.cz/xml/xslt/index.html, 2004

[7] Emden R. Gansner, Eleftherios Koutsofios, Stephen North: Drawing graphs with dot, 2006

[8] Emden R. Gansner, Drawing graphs with Graphviz, 2007

Příloha A

Obsah kompaktního disku

Přiložený kompaktní disk obsahuje implementovaný generátor dokumentace databázových schémat včetně všech pomocných programů, které ke své činnosti využívá. Dále je na disku k dispozici elektronická verze této práce.

Kořenový adresář disku obsahuje tři podadresáře.

Jednotlive_aplikace

Zde jsou umístěny ZIP archívy jednotlivých aplikací. Je tu k dispozici aplikace Schemagic, sada XSL stylů DocBooku ve verzi 1.72.0, dále aplikace FOP 0.93, Graphviz verze 2.8 pro operační systém Windows a v neposlední řadě aplikace Gendok. Popis instalace těchto jednotlivých programů je popsán v kapitole 0 – Instalace a konfigurace.

Kompletni_instalace

Adresář "Kompletni_instalace" obsahuje jediný ZIP archív se vším, co je potřeba ke zprovoznění generátoru dokumentace. Jedná se o alternativu pro instalaci jednotlivých aplikací, která má uživateli zjednodušit práci. Postup instalace tohoto balíku je popsán v části 1.23.2 tohoto dokumentu.

Bakalarska_prace

V tomto adresáři je umístěna digitální podoba této práce.

Příloha B

Ukázkový konfigurační soubor

Následuje ukázka typického konfiguračního souboru aplikace Gendok. Jsou zde definovány hodnoty základních parametrů aplikace, obsažena je také definice struktury výstupní dokumentace. Podoba konfiguračního souboru je popsána v části 1.12 tohoto dokumentu.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE documentation SYSTEM "./config.dtd">
<documentation>
    <properties>
        <property name="output-directory"
            value="../::FORMAT::/::TIMESTAMP::"/>
        <property name="output-encoding" value="UTF-8"/>
        <property name="output-language" value="cs"/>
        <property name="erase" value="no"/>
        <property name="docbook-xsl-home"
            value="../docbook-xsl-1.72.0"/>
        <property name="css-stylesheet"
            value="./css/::FORMAT::_01.css"/>
        <property name="fop-cfg-file"
            value="../fop-0.93/conf/fop.xconf"/>
        <property name="xslt-procesor"
            value="com.icl.saxon.TransformerFactoryImpl"/>
        <property name="html-pix-format" value="png"/>
        <property name="fo-pix-format" value="png"/>
        <property name="docbook-pix-format" value="svg,png"/>
    </properties>
    <book>
        <chapter title="::L-TABLES::">
            <foreach elem="table" elem-name="::L-TABLE::">
                <section type="table-dependencies"/>
                <section type="table-comments"/>
                <section type="table-warnings"/>
                <section type="table-columns"/>
                <section type="table-picture">
                    <call-plugin name="gendok.plugin.Painter">
                        <param name="dotxslt" value="./xsl/dot.xsl"/>
                        <param name="graphviz-home"
                            value="../graphviz/bin"/>
                    </call-plugin>
                </section>
                <section type="table-constraints"/>
                <section type="table-triggers">
                    <call-plugin name="gendok.plugin.SourcePrinter">
                        <param name="object-family"
                            value="ora_table_trigger"/>
                    </call-plugin>
                </section>
                <section type="table-indexes"/>
            </foreach>
        </chapter>
```

```
        <chapter title="::L-VIEWS::">
            <foreach elem="view" elem-name="::L-VIEW::">
                <section type="view-dependencies"/>
                <section type="view-comments"/>
                <section type="view-columns"/>
                <section type="view-select">
                    <call-plugin name="gendok.plugin.SourcePrinter">
                        <param name="object-family" value="view"/>
                    </call-plugin>
                </section>
            </foreach>
        </chapter>
        <chapter title="::L-PROCEDURES::">
            <foreach elem="procedure" elem-name="::L-PROCEDURE::">
                <section type="procedure-dependencies"/>
                <section type="procedure-source">
                    <call-plugin name="gendok.plugin.SourcePrinter">
                        <param name="object-family" value="procedure"/>
                    </call-plugin>
                </section>
            </foreach>
        </chapter>
    </book>
</documentation>
```