

MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Konverze bibliografických záznamů

DIPLOMOVÁ PRÁCE

David Novák

Brno, jaro 2004

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Miroslav Bartošek, CSc.

Poděkování

Rád bych poděkoval vedoucímu práce panu RNDr. Miroslavu Bartoškovi, CSc. za efektivní vedení, nadhled a příhodné rady. Paní PhDr. Haně Vochozkové za dobrou spolupráci a hlavně za obětavost a trpělivost při zdánlivě nekonečném procesu doladování konverzních pravidel a odstraňování chyb. Také Mgr. Martinu Šárfymu za vytvoření webového rozhraní pro testování konverzního systému. V neposlední řadě své rodině a přítelkyni za jejich pochopení pro mé zaujetí prací na tomto projektu.

Shrnutí

Knihovní katalogy používají různé systémy pro uchovávání bibliografických resp. katalogizačních záznamů. Formáty ukládaných dat v těchto systémech jsou různé a většinou nekompatibilní. Při přenosu záznamů mezi různými knihovními systémy je proto často nutné tyto záznamy konvertovat do jiného formátu. Také, chce-li knihovna začít používat jiný knihovní systém, je třeba celý katalog převést do nového formátu.

Hlavním cílem diplomové práce bylo navržení a implementace samostatného konverzního systému **Tin2marc** pro převod bibliografických záznamů uložených ve formátu **TinLib** (resp. **T-Series**) do formátu MARC21 pro systém **Aleph**. Program musí být využitelný pro konverze záznamů knihoven Masarykovy univerzity při přechodu mezi dvěma zmiňovanými systémy, přičemž musí umožňovat nastavit jednoduchým způsobem konverzní specifika jednotlivých fakultních katalogů.

Konverzní systém byl navržen tak, že sada pravidel popisujících jednotlivé konverzní kroky je zcela oddělena od samotného programu. Program pak aplikuje požadovaná pravidla na vstupní záznamy.

Součástí práce je popis knihovních formátů relevantních pro vytvářený systém, (tedy **T-Series**, MARC21 a **Aleph**), dále obecná část zabývající se problémem výměny a konverze bibliografických záznamů. Samostatná kapitola popisuje formát a způsob zápisu konverzních pravidel v systému **Tin2marc** a následuje popis vlastního programu, který zapsaná pravidla aplikuje na **TinLib** záznamy.

Klíčová slova

TinLib, T-Series, MARC, MARC21, Aleph, konverze, knihovna, katalog, bibliografický záznam, katalogizační záznam.

Obsah

1	Úvod	3
2	Bibliografické formáty	5
2.1	TinLib – T-Series	5
2.2	Formáty MARC	7
2.2.1	Záznam ve formátu MARC	8
2.2.2	Výměnný formát dle ISO 2709	9
2.2.3	Řádkový MARC	9
2.2.4	MARC v XML syntaxi	10
2.3	Aleph	10
3	Konverze bibliografických záznamů	12
3.1	Konverze výměnných formátů	12
3.2	Kódování diakritiky	13
4	Konverzní pravidla	14
4.1	Specifikace problému	14
4.2	Zápis konverzních pravidel	14
4.2.1	XML	15
4.2.2	DTD	16
4.2.3	Regulární výrazy	17
4.3	Formát konverzních pravidel	19
4.3.1	Jedno pravidlo – item	19
4.3.2	Zdrojové pole – TLFIELD	20
4.3.3	Regulární výraz – regexp	21
4.3.4	MARC pole – MARCFIELD	21
4.3.5	Cílový řetězec – targetString	22
4.3.6	Podmínka – condition	23
4.3.7	Posloupnosti pravidel – sequence	24
4.3.8	Poznámky	27
4.4	Příklad konverzního pravidla	27
5	Program Tin2marc	31
5.1	Programovací jazyk	31
5.1.1	Java	31
5.1.2	Objektově orientované programování	32
5.2	Objektový návrh	33
5.3	Průběh výpočtu	37

5.3.1	Logování	38
5.4	Uživatelské rozhraní	38
5.4.1	Webové rozhraní	40
5.5	Požadavky programu a jeho výkon	40
6	Závěr	42
	Literatura	44
	Rejstřík	45
	Přílohy	47

Kapitola 1

Úvod

Současný trend vývoje společnosti je někdy nazýván jako „informační boom“, protože stále zásadnější roli v našich životech hrají informace. Jejich získávání a výměna jsou stále častějšími aktivitami a narůstá také význam informací jakožto obchodního artiklu.

Ve světě s naznačenými prioritami, ve kterém se skladuje stále větší množství údajů, je velice důležitá systematizace ukládání informací a možnost jejich efektivního prohledávání. I přes enormní rozmach elektronických informačních technologií, který nastal v posledních dvaceti letech, stále narůstá i množství publikací vydávaných tiskem. Prostředníkem mezi fyzickými i elektronickými médii a uživateli jsou stále knihovny a jejich katalogy.

Právě rozvoj informačních technologií umožňuje uskutečnit myšlenku opakovaného použití jednou vytvořeného bibliografického záznamu. Knihovní katalogy jsou totiž (až na výjimky) v elektronické formě a výměna záznamů mezi knihovnami by tak neměla být principiálním problémem. Formáty ukládání záznamů v různých knihovních systémech jsou však často navzájem nekompatibilní.

Přenos záznamů mezi různými knihovními systémy proto často vyžaduje konverzi záznamů do nějakého tvaru „srozumitelného“ pro cílový systém. Také, chce-li knihovna začít používat jiný knihovní systém, je třeba celý katalog převést do nového formátu.

Hlavním cílem této práce bylo právě vytvoření samostatného konverzního systému **Tin2marc** pro převod bibliografických záznamů uložených ve formátu **TinLib** (resp. **T-Series**) do formátu MARC21 (resp. MARC21 pro **Aleph**). Program musí být využitelný pro konverze záznamů knihoven Masarykovy univerzity při přechodu mezi dvěma zmiňovanými systémy.

V kapitole 2 tohoto textu jsou popsány knihovní formáty relevantní pro vytvářený systém, tedy **T-Series**, MARC (resp. MARC21) a **Aleph**. Kapitola 3 se zabývá problémem výměny a konverze bibliografických záznamů obecně. V systému **Tin2marc** je oddělen popis pravidel pro převod záznamů od vlastního programu. Ve 4. kapitole je popsán formát a způsob zápisu těchto pravidel a kapitola 5 je o vlastním programu, který zapsaná pravidla aplikuje na **TinLib** záznamy.

Formáty rodiny MARC, jejíž součástí je i MARC21, se za čtyřicet let své existence a vývoje stále jednoznačněji stávají světovým standardem pro ukládání bibliografických záznamů. Od roku 2002 je právě MARC21 formátem, na který by měly přecházet katalogy knihoven v České republice. Více o standardech MARC viz odstavec 2.2.

Kapitola 2

Bibliografické formáty

Jeden z pojmů, které budou v této práci používány velice často, je *bibliografický záznam*. Význam tohoto termínu budeme chápat tak, jak jej definuje norma ČSN ISO 5127-3a (viz [2]). Podle ní je bibliografický záznam soubor prvků bibliografického popisu a záhlaví dokumentu, potřebný pro zařazení do katalogu nebo bibliografie. *Katalogizační záznam* je pak soubor prvků zahrnujících bibliografický záznam a signaturu přiřazenou podle pravidel dané organizace.

Rozvoj výpočetní techniky umožnil automatizaci zpracování bibliografických resp. katalogizačních záznamů. Automatizace přináší jednak výrazné zefektivnění uchovávání knihovních katalogů a vyhledávání v nich a také možnost znovupoužití jednou vytvořeného záznamů dalšími knihovnami. Bezproblémové výměny záznamů vyžadují samozřejmě sjednocení knihovnami používaných formátů dat.

Za zhruba čtyřicetiletou historii vývoje automatizovaných nástrojů pro správu bibliografických záznamů však vzniklo velké množství formátů, které jsou v některých případech částečně kompatibilní, ale většinou zcela odlišné. V této kapitole představíme tři formáty resp. knihovní systémy: **T-Series** (**TinLib**), formáty rodiny **MARC** a systém **Aleph**.

2.1 TinLib – T-Series

T-Series je jedním z prvních automatizovaných knihovních systémů používaných v České republice. Tento systém byl původně vyvinut britskou společností **IME** pod názvem **TinLib**. Později jako **T-Series** začal být rozvíjen mezinárodní společností *Electronic Online Systems International (EOSi)*. V České republice je systém rozšiřován od roku 1992 hlavně v akademickém prostředí, jeho distributorem je ÚVT UK Praha. Nyní se v ČR nejčastěji používá **T-Series** ve verzích 305 nebo 307. Budeme se tedy dále zabývat těmito verzemi, i když většina faktů platí i pro jiné verze zmíněných systémů.

Základem **T-Series** je entitně-relační databázový systém **TinMAN** (také od firmy *EOSi*). Přímou na této nízké úrovni jsou implementovány hypertextové techniky. V databázové struktuře se automaticky vytvářejí vazby mezi ucho-

vávanými entitami – autory, tituly, předmětovými skupinami, klíčovými slovy, výpůjčkami atd. Tyto vazby umožňují plynulou navigaci mezi entitami.

T-Series je tvořen vzájemně propojenými a kooperujícími moduly, které lze k systému postupně připojovat, jako např.

- moduly pro automatizaci interních knihovnických činností a služeb (Katalog, Výpůjčky, Akvizice, Správa seriálů, MVS),
- systémové moduly (Administrátor, Konverze dat, Archivace, Zprávy),
- síťové moduly pro různé typy lokálních i rozlehlých sítí (File-server, Terminal-server, Klient-Server),
- moduly pro potřeby čtenářů (OPAC).

Více informací o systému **T-Series** viz např. [1].

Vstupními daty pro vytvářený program **Tin2marc** je „totální“ export z **T-Series**. Tímto pojmem rozumíme export bibliografických záznamů včetně lokálních údajů příslušné knihovny (přírůstkové číslo, signatura, lokální poznámky. . .), tedy katalogizační záznamy. Export je textový soubor, ve kterém jsou jednotlivé záznamy tvořeny řádky tvaru:

```
název_pole:hodnota pole T-Series
```

Názvy polí (klíče) jsou čtyřpísmenné. První písmeno vyjadřuje vždy druh záznamu – např. M pro monografie, S pro periodika (seriály) atp. Zbytek názvu pole reprezentuje význam obsahu pole, např. MTIT pro název (title) nebo MOTI pro ostatní názvy (other titles). Jednotlivé záznamy jsou odděleny ASCII znakem 12 (hexadecimálně 0C).

Zkráceným příkladem vstupních dat je tedy např. následující text (znak 0C je zde značen jako ^L):

```
MTIT:Pán prstenů.\,\ Společenstvo Prstenu
MAUT:Tolkien, J. R. R. (John Ronald Reuel),\\\ 1892-1973
MPUB:Mladá fronta
MPPL:Praha
MPDT:1990
MISB:80-204-0105-9
^LMTIT:Nadace
MRES:Isaac Asimov
MOTI:Foundation (Orig.)
MAUT:Asimov, Isaac,\\\ 1920-
MOTH:Pravcová, Jarmila
MROL:Překladatel \\730\
```

Předmětem konverzí programu **Tin2marc** mají být monografie a seriály. V databázích **T-Series** jsou kromě záznamů, které odpovídají každý vždy jedné monografii (periodiku) i záznamy, které obsahují informace týkající se „abstraktních entit“ jako jsou např. knižní edice nebo záznamy týkající se originálního

názvu (pro dokumenty, které jsou překladem z jiného jazyka). Tyto záznamy jsou také mezi vstupními daty konverzního programu – nekonvertují se jako ostatní záznamy, ale jsou propojovány s ostatními záznamy, protože informace v nich obsažené jsou potřeba při vytváření záznamů cílového formátu.

2.2 Formáty MARC

Projekt MARC (Machine Readable Cataloging) vznikl v roce 1963 ve Spojených státech amerických jako iniciativa washingtonské Library of Congress. Jeho cílem bylo vytvořit široký formát pro katalogizaci a výměnu bibliografických záznamů. Systém se velmi brzo osvědčil a v USA začalo fungovat první efektivní přebírání bibliografických záznamů.

Na základě koncepce MARC formátu začalo vznikat velké množství národních formátů. Takto vznikly např. CAN/MARC, UKMARC, DANMARC, USMARC (který se později stal standardem v mnoha anglicky mluvících zemích) a několik desítek dalších formátů.

V roce 1977 vznikl z iniciativy organizace IFLA formát UNIMARC, jehož cílem bylo stát se prostředníkem mezi množstvím národních formátů. Tento formát se rozšířil a mnohé (hlavně evropské) státy ho přijaly jako svůj národní formát.

V anglo-americkém prostředí byly stále nejvíce používány USMARC a CAN/MARC. Tyto dva formáty se v roce 1997 sjednotily a vytvořily MARC21. Do vzniklého formátu byly zahrnuty i některé pozitivní rysy UNIMARCU (především dostatečná granularita). MARC21 se stal standardem ve většině anglicky mluvících zemích a v současné době se jeví jako perspektivní i pro evropské prostředí.

Všechny formáty vycházející z MARCu mají společné způsoby zápisu dat a některé další rysy. Pro vytvoření konverzního programu se znalostí slovně formulovaných konverzních pravidel, což je předmětem této diplomové práce, není potřeba znát přesné rozdíly mezi jednotlivými MARC formáty. Nebudeme se tedy faktickými rozdíly mezi nimi do hloubky zabývat, ale zaměříme se spíše na základní strukturu a formát zápisu záznamů. Pokud bude potřeba věnovat se vlastnostem specifickým pro konkrétní MARCovský formát, bude to MARC21, protože ten (a jeho upravená verze pro systém Aleph – viz odstavec 2.3) jsou cílovými formáty konverzního programu **Tin2marc**.

V České republice byla od počátku 90. let minulého století vytvářena česká lokalizace formátu UNIMARC. V roce 2002 ale padlo rozhodnutí o postupném přechodu na MARC21.

2.2.1 Záznam ve formátu MARC

Abychom mohli při popisu logické struktury MARCu používat příklady, uvedme několik poznámek o zápisu tohoto formátu (podrobně viz [5]). MARCovský záznam je textový soubor prokládaný řídicími ASCII znaky (z rozmezí 0–31). Tyto znaky i některé standardně zobrazitelné znaky mají specifický význam z hlediska formátu MARC. Existuje více konvencí pro zobrazování těchto znaků; my se budeme držet následujících:

\$ – znak \$ budeme používat jako oddělovač podpolí (místo znaku hex 1F) – viz dále; např. „podpole a“ budeme značit \$a,

| – znak | bude reprezentovat tzv. výplňový znak (hex 7C),

– znak # bude používán místo znaku mezery (hex 20) v některých speciálních případech a tehdy, když by mohl být význam mezery víceznačný.

Podívejme se nyní blíže na strukturu MARCovského záznamu. Na následujícím příkladu části záznamu v tzv. *řádkovém* MARCu (viz odstavec 2.2.3) vysvětlíme principy formátu MARC.

```
#####nam#a22#####7a#4500
003CZ-BrMU
008|||||s1990####|||r|||||||d
020 ## $a8020401059
040 ## $bcze
100 1# $aTolkien, J. R. R. $q(John Ronald Reuel), $d1892-1973.
245 10 $aPán prstenů. $pSpolečenstvo Prstenu.
260 ## $aPraha : $bMladá fronta, $c1990.
```

Každý záznam je logicky rozdělen do tzv. *polí* (*field*). Existuje např. pole pro informace o autorovi, pro název díla atp. Každému poli je přiřazeno trojciferné číslo (v některých formátech i jiný řetězec délky 3), tzv. *tag*. V záznamu může být i více polí se stejným tagem. Každý řádek z příkladu (kromě prvního) je jedním polem.

Pole jsou rozdělena do tzv. *podpolí* (*subfield*). Jednotlivá podpole jsou oddělena speciálním znakem „\$“ a za ním následuje jednoznačný identifikátor podpole. Např. pole 245 (informace o názvu) obsahuje podpole \$a a \$p:

```
245 10 $aPán prstenů. $pSpolečenstvo Prstenu.
```

Všechna pole kromě polí 001–009 mohou obsahovat jeden či dva tzv. *indikátory* (*indicator*). Jsou to čísla od 0 do 9 a zapisují se mezi tag pole a jeho obsah. Pokud je místo čísla na místě indikátoru mezera (#), znamená to, že indikátor není použit. V předchozím příkladu pole 245 má první indikátor hodnotu 1 a druhý indikátor 0.

Prvních 24 znaků každého záznamu tvoří tzv. *návěští* (*leader*), ve kterém jsou obsaženy informace o struktuře a délce záznamu, bibliografické úrovni záznamu, jeho úplnosti apod. Pozice v návěští se číslují 0–23.

2.2.2 Výměnný formát dle ISO 2709

Formát MARC definuje pouze strukturu záznamu, konkrétní forma zápisu strukturovaného záznamu může být různá. Jeden ze způsobů zápisu je popsán normou **ISO 2709 Formát pro výměnu informací** a byl přijat téměř všemi výrobci knihovnických systémů i tvůrci formátů (viz [3]).

Podle této normy musí bibliografický záznam mít tuto strukturu: návěští, adresář, pole s údaji a oddělovač záznamů. Adresář, který následuje po již zmíněném návěští, se skládá z položek korespondujících s jednotlivými poli záznamu. Každá 12-ti znaková položka adresáře se skládá ze tří znaků označujících tag, následují 4 číselné znaky znamenající počet znaků pole a pěticeferné číslo označující počáteční pozici pole v rámci sekce s údaji.

Po adresáři následuje sekce s poli. Každé se skládá z indikátorů, samotných dat a oddělovače polí. Stejně jako jsme zavedli konvenci pro zobrazování znaku oddělovače podpole jako \$, budeme používat pro zobrazení oddělovačů polí a záznamů (hex 1E a 1D) znaky @ a %. V celém souboru se záznamy nejsou žádné znaky nového řádku (v následujícím příkladu je řádek zalomen vždy po 60-ti znacích):

```
00332nam a22001217a 4500FMT00030000000300080000400800410001
30200015000550400008000711000061000802450040001422600034001
83@BK@CZ-BrMU@|||||s1990 |||||||r|||||||||||||||d@ $a
8020401059@ $bcze@1 $aTolkien, J. R. R.$q(John Ronald Reue
l), $d1892-1973.$4aut@10$aPán prstenů.$pSpolečenstvo Prstenu
.@ $aPrah :$bMladá fronta,$c1990.@
```

2.2.3 Řádkový MARC

Záznam ve výměnném formátu dle ISO 2709 je vytvořen pro strojové zpracování, není však dobře čitelný pro člověka. Proto se často používá (někdy i pro výměnu dat) tzv. řádková struktura MARCovského záznamu. Neexistuje žádná závazná norma popisující tuto strukturu, ale pravidla mohou vypadat např. takto:

Každý záznam začíná na novém řádku. Každé pole začíná na novém řádku a má následující strukturu:

pozice	1-3	4	5-6	7	8-?
	tag	mezera	indikátory	mezera	text pole

Návěští může být značeno jako pole 000, tedy jeden záznam může vypadat např. takto:

```
000 #####nam#a22#####7a#4500
003 CZ-BrMU
008 |||||s1990####|||r|||||||||||||||d
020 ## $a8020401059
```

```

040 ## $bcze
100 1# $aTolkien, J. R. R. $q(John Ronald Reuel), $d1892-1973.
245 10 $aPán prstenů. $pSpolečenstvo Prstenu.
260 ## $aPraha : $bMladá fronta, $c1990.

```

2.2.4 MARC v XML syntaxi

Velice moderním trendem v oblasti výměny dat je používání jazyka XML (více o XML viz část 4.2.1). Jedním z projektů zabývajících se zápisem MARCu pomocí XML je projekt sdružení Open Source Software for Libraries, v jehož rámci vznikl konvertor MARC.pm. XML syntaxe, která je výstupem tohoto programu, vychází přímo ze struktury MARCu:

```

<?xml vesion="1.0" encoding="UTF-8">
<marc>
  <record>
    <field type="000">      nam a22      7a 4500</field>
    <field type="003">CZ-BrMU</field>
    <field type="008">|||||s1990   |||||||r|||||||||||||||d</field>
    <field type="020" il=" " i2=" ">
      <subfield type="a">8020401059</subfield>
    </field>
    <field type="040" il=" " i2=" ">
      <subfield type="b">cze</subfield>
    </field>
    <field type="100" il="1" i2=" ">
      <subfield type="a">Tolkien, J. R. R.</subfield>
      <subfield type="q">(John Ronald Reuel),</subfield>
      <subfield type="d">1892-1973.</subfield>
    </field>
    <field type="245" il="1" i2="0">
      <subfield type="a">Pán prstenů.</subfield>
      <subfield type="p">Společenstvo Prstenu.</subfield>
    </field>
    <field type="260" il=" " i2=" ">
      <subfield type="a">Praha :</subfield>
      <subfield type="b">Mladá fronta,</subfield>
      <subfield type="c">1990.</subfield>
    </field>
  </record>
</marc>

```

2.3 Aleph

Aleph je integrovaný systém pro kompletní správu knihoven od izraelské firmy *Ex Libris*. Pro ukládání dat používá databázi **Oracle**. Je používán stále více

knihovny na celém světě a od roku 2003 je k dispozici i na Masarykově univerzitě.

System **Aleph** podporuje několik MARCovských i jiných formátů pro ukládání, import a export bibliografických záznamů. Jedním z nich je i mírně upravená verze řádkového MARC21, popsaného v části 2.2.3. Ekvivalentní zápis k příkladu popsanému v sekci 2.2.3 zapsaný v Alephovském MARCu vypadá takto:

```
000000333 LDR      L ^^^^^^nam^a22^^^^^7a^4500
000000333 FMT      L BK
000000333 003      L CZ-BrMU
000000333 008      L ||||||s1990^^^^^|||||||r|||||||d
000000333 020      L $$a8020401059
000000333 040      L $$bcze
000000333 1001     L $$aTolkien, J. R. R.$$q(John Ronald Reuel),$d1892-1973.
000000333 24510    L $$aPán prstenů.$pSpolečenstvo Prstenu.
000000333 260      L $$aPraha :$$bMladá fronta,$c1990.
```

Následující vlastnosti odlišují tento zápis od standardního řádkového MARCu:

- 9-ti místné číslo na začátku každého řádku (pole) je systémové číslo záznamu (tzv. SYSNO),
- mezery se místo znaku # označují znakem ^; místo prázdného indikátoru vždy přímo znak mezery „ “,
- před znaky pro indikátory není mezera; za nimi je ještě sekvence „ L “,
- jako oddělovač polí se používá dvojice znaků \$\$,
- umožňuje vytvářet i pole s nečíselnými tagy,
- návěští je zapisováno jako pole s tagem LDR,
- obsahuje některá nová pole (např. FMT).

Tento formát je také jedním z možných výstupů vytvářeného konverzního programu.

Kapitola 3

Konverze bibliografických záznamů

Jak bylo diskutováno v kapitole 2, existuje velké množství formátů pro ukládání bibliografických záznamů. Chceme-li použít záznamy uložené v nějakém knihovním systému mimo něj, musíme je převést do formátu vhodného pro výměnu. Této funkci se říká *export*. Pokud naopak chceme do systému vložit záznamy vytvořené jinde, musíme provést tzv. *import* dat v některém ze systémem podporovaných importních formátů.

Ale záznamy vyexportované z jednoho systému většinou nelze ihned použít pro import do jiného systému právě z důvodu rozdílného formátu dat. Pokud jsou součástí exportních a importních funkcí daných systémů konverze záznamů do/z nějakého společného výměnného formátu, je výměna bez problémů. Pokud ne, je potřeba použít pro tyto konverze samostatný program.

3.1 Konverze výměnných formátů

Převod záznamu z jednoho formátu do jiného není triviální záležitostí. Jednak jsou informace v různých formátech jinak rozloženy a musí být při konverzích vždy v záznamu vyhledány a přeskupeny. Dále může dojít při konverzích ke ztrátě informace, pokud ji cílový formát neobsahuje nebo naopak požadovaná informace chybí ve vstupním formátu.

Podrobně analyzuje Stoklasová varianty konverzí v [8] následujícím způsobem:

- údaje se nachází ve stejné formě, jsou stejně strukturovány a při konverzi se případně pouze přejmenují názvy polí a podpolí,
- údaje mají stejnou formu, ale různou míru detailnosti (v jednom směru převodu se může ztratit informace),
- údaj chybí, ale lze jej odvodit z jiného údaje,
- údaj sice existuje, ale má jinou hodnotu, kterou je nutno zaměnit za jinou odvoditelnou (v případě jiných kódovníků rolí apod.),
- údaj chybí a nelze ho odvodit.

V [4] rozšiřuje Žabičková toto rozlišení ještě o tyto možnosti:

- údaje ze dvou polí je nutno spojit do jednoho,
- údaj z jednoho pole je nutné rozdělit (což se ne vždy podaří).

Pokud se stane, že ve zdrojovém formátu chybí nějaký údaj, který je povinný pro formát cílový, je nutné ho buď doplnit konstantou nebo vytvořit heuristiku, která tento údaj doplní na základě různých charakteristik. Takový mechanismus pak nemůže být zcela bezchybný.

3.2 Kódování diakritiky

Neshoda na úrovni bibliografického formátů nemusí být jediná, kterou je třeba při výměně dat řešit. Typickým problémem, se kterým se potýká výpočetní technika v neanglicky mluvícím prostředí již od dob zavedení osmibitového kódování znaků, je kódování znaků s diakritickými znaménky. Pro češtinu, stejně jako pro většinu ostatních jazyků kromě angličtiny, existuje několik kódování diakritiky, které jsou vzájemně nekompatibilní nebo částečně kompatibilní.

Pro konverzi těchto kódování existuje celá řada programů. Některé programy dokáží kódování dat rozpoznat samy a konverzi provedou automaticky. Je nutné vyhnout se ztrátě informace při převodu znaků, které vstupní kódová sada podporuje a výstupní ne. Konvertory poté znak převedou buď na znak bez diakritiky nebo znak vypustí nebo ho převedou na nesmyslný znak.

Při konverzi formátů dat pomocí samostatného programu je možné použít některý z existujících konvertorů diakritiky nebo kódování převést uvnitř programu.

Kapitola 4

Konverzní pravidla

4.1 Specifikace problému

Hlavním cílem této diplomové práce bylo navržení a implementace samostatného konverzního systému **Tin2marc**, který převádí bibliografické záznamy uložené ve formátu **TinLib** (resp. **T-Series**) do formátu MARC21 (resp. MARC21 pro **Aleph**). Program musí být možné použít pro převod záznamů knihoven Masarykovy univerzity při přechodu mezi dvěma zmiňovanými systémy, přičemž musí umožňovat nastavit jednoduchým způsobem konverzní specifika jednotlivých fakultních katalogů.

Zformování vlastních konverzních pravidel nebylo součástí této práce, ale autorovi byl poskytnut jejich popis ve formě konverzních tabulek a strukturovaného textu. Tato pravidla vytvořila paní PhDr. Hana Vochozková ve spolupráci s dalšími odborníky z MU. Přílohy 1, 2 a 3 obsahují všechny tyto popisy.

Základním požadavkem na **Tin2marc** byla dostatečná flexibilita a robustnost programu při změně vlastních konverzních pravidel. Pravidla byla jednak v průběhu práce na konverzním systému teprve vytvářena, často upravována a „laděna“ a také se mírně liší pro jednotlivé knihovny podle přesného formátu vstupních záznamů. V této situaci je vhodné, aby byla pravidla oddělena od vlastního programu (jádra systému).

Úkolem autora bylo tedy vytvořit formát konverzních pravidel dostatečně obecný na to, aby obsáhl všechna specifikovaná pravidla. Dále vytvořit mechanismus zápisu těchto pravidel tak, aby mohla být tvořena člověkem a zpracována strojově. Nakonec vytvořit program, který načte sadu konverzních pravidel a pomocí nich provede konverzi **TinLib** záznamů do formátu MARC21.

4.2 Zápis konverzních pravidel

Jak bylo zdůvodněno v části 4.1, konverzní pravidla jsou oddělena od programu samotného a měla by být zapsána v takovém tvaru, aby je mohl vytvářet člověk, modifikovat je a jednoduše se vyznat v jejich struktuře, ale aby byla pravidla také dobře zpracovatelná strojově (načtena konverzním programem).

Při takovém zadání se přímo nabízí použít pro zápis pravidel jazyk XML a vytvořit si tak vlastní značkovací jazyk. Standard XML je určen pro strukturo-

vaný zápis jakýchkoli dat textové povahy, dává prakticky neomezenou možnost pro vytvoření vlastního formátu pro zápis údajů a existuje velké množství nástrojů pro různé operace s XML daty.

4.2.1 XML

Abychom mohli popsat navržený formát konverzních pravidel, je potřeba se nejdříve seznámit s principy jazyka XML (eXtensible Markup Language). Standard XML je spravován konsorciem *World Wide Web Consortium (W3C* – viz <http://w3c.org>) a vychází z rozsáhlého jazyka SGML. XML definuje obecnou syntaxi značkovacích jazyků pro popis jakýchkoli více či méně strukturovaných dat. Konkrétní značky si uživatel může nadefinovat sám takovým způsobem, aby co nejlépe odpovídaly konkrétní struktuře dat. Uvedme si příklad XML struktury popisující jeden záznam ze zpěvníku:

```
<pisen id="id_dokud">
  <nazev>Dokud se zpívá ještě se neumřelo</nazev>
  <autor>Jaromír Nohavica</autor>
  <album>Darmoděj</album>
  <rok>1985</rok>
</pisen>
```

XML dokument je složen z jednotlivých *elementů*, každý element je vymezen počáteční a koncovou *značkou (tagem)*, např. `<nazev>Milionář</nazev>`. Každý element může obsahovat další elementy a/nebo textové řetězce (*textové uzly*). Prázdný element lze zkráceně zapsat jako `<element />`.

Element může obsahovat také *atributy*: `<pisen jazyk="cz" />`. Pro zápis speciálních znaků jako „<“ nebo „>“ se používají tzv. *entity*, tedy speciální řetězce začínající „&“ a končící znakem „;“, např. `<` nebo `>`. Kromě předdefinovaných entit lze definovat i své vlastní.

XML dokument musí začínat hlavičkou, např.

```
<?xml version="1.0" encoding="windows-1250" ?>
```

za kterou může následovat deklarace typu dokumentu (viz 4.2.2), a poté následuje jeden element na nejvyšší úrovni (tzv. kořenový element).

Má-li aplikace zpracovat XML dokument, může načíst data jako prostý textový dokument a analyzovat XML strukturu. Výhodnější a jednodušší je ale použít některé z aplikačních rozhraní definovaných za tímto účelem. Tato rozhraní jsou implementována množstvím nástrojů a knihoven.

Jedním z nich je DOM (Document Object Model). Toto rozhraní vytvoří z XML dokumentu strom jeho elementů. Kořenovým uzlem stromu je kořenový element dokumentu a dceřinými uzly každého elementu jsou jeho podelementy resp. textové uzly (jejich pořadí se zachovává). Listy jsou tedy textové uzly nebo prázdné elementy. Atributy uložené u příslušných elementů lze chápat také jako speciální listové uzly. Touto strukturou lze efektivně procházet a manipulovat s ní.

Druhým standardním rozhraním je SAX (Simple API for XML). Toto je rozhraní založené na zprávách – při analýze dokumentu je aplikaci vysílán sled událostí. Typickými událostmi jsou „začátek elementu: element + atributy“, „textový uzel: řetězec“ nebo „konec elementu: element“. Je na zpracovávající aplikaci, aby tyto události obsloužila.

Existují samozřejmě i další aplikační rozhraní pro XML dokumenty, ale tato dvě jsou nejrozšířenější a reprezentují dva základní přístupy k problému. Stromová rozhraní mají výhodu jednoduššího přístupu aplikace k dokumentu a neomezené možnosti procházení dokumentu. Jsou ale nepoužitelné pro velmi rozsáhlé XML dokumenty kvůli své náročnosti na paměť a na dobu vytváření stromu v paměti. Naopak rozhraní založená na událostech jsou velice rychlá a paměťově nenáročná.

4.2.2 DTD

Jak již bylo zmíněno, jednou z nejdůležitějších vlastností XML je volnost uživatele při definování elementů, atributů a jejich zanořování, tedy určení struktury dokumentu tak, aby co nejlépe vyhovovala aplikaci. Existuje několik standardů pro definování struktury XML dokumentů. Tím nejstarším, relativně nejslabším, ale pro svou jednoduchost oblíbeným jazykem je *DTD* (Data Type Definition). Narozdíl od novějších a výkonnějších standardů (např. *XML Schema*) je DTD podporováno prakticky všemi nástroji pro práci s XML.

Pro definici formátu konverzních pravidel bylo použito právě DTD, seznamme se tedy s jeho principy. DTD definice může být buď přímo součástí XML souboru nebo častěji tvoří samostatný soubor. Interní definice nebo odkaz na externí definici typu dokumentu musí být mezi XML hlavičkou a začátkem kořenového elementu:

```
<!DOCTYPE zpevnik SYSTEM "zpevnik.dtd">
```

nebo

```
<!DOCTYPE zpevnik [ DTD definice ]>
```

V první řadě může DTD definovat elementy a povolený obsah jejich těla. Do závorky za název elementu se zapisují dceřiné elementy oddělené čárkou, které mohou mít následující modifikátory: ? (element není povinný), + (jeden či více elementů) nebo * (žádný, jeden nebo více elementů).

```
<!ELEMENT pisen "(nazev, autor+, album?, rok?)">
```

Tato definice říká, že element *pisen* musí obsahovat element *nazev*, jeden nebo více elementů *autor* a může obsahovat elementy *album* a/nebo *rok*. Pořadí elementů musí zůstat zachováno. Pokud místo čárky použijeme znak | (např. (autor | lidova)), znamená to, „právě jeden z uvedených elementů“. Pokud element má obsahovat jen textový uzel, zapíšeme to v DTD takto:

```
<!ELEMENT pisen "(#PCDATA)">
```

Dále můžeme v DTD definovat atributy jednotlivých elementů pomocí následující syntaxe:

```
<!ATTLIST pisen
  id CDATA #REQUIRED
  jazyk CDATA #IMPLIED>
```

Dle této definice má element `pisen` jeden povinný atribut `id` a jeden nepovinný atribut `jazyk` oba typu `CDATA`, tedy prostý text. V DTD můžeme definovat i uživatelské entity. Lze je brát jako makra, jejichž výskyty `&entita;` v XML souboru se rozvinou podle druhé části definice:

```
<!ENTITY oblíbenyAutor "Jaromír Nohavica">
```

Vložení znaku `%` před název entity v její definici vznikne tzv. *parametrická entita*, kterou lze použít pouze v DTD souboru (v ostatních definicích):

```
<!ENTITY % zpevnik_type "(nazev, rok, pisen*)">
<!ELEMENT zpevnik %zpevnik_type;>
```

Pokud XML soubor obsahuje interní nebo externí DTD definici, měla by danému DTD jeho struktura odpovídat. Pak říkáme, že je XML dokument *validní*. Pro ověřování validity dokumentu existují nástroje (parsery). Některé operace nad XML daty přímo vyžadují přítomnost DTD (nebo jiné definice formátu) a validitu dokumentu proti jeho DTD.

4.2.3 Regulární výrazy

Jak víme z části 2.1, pole **TinLibu** nejsou explicitně dělena na podpole, ale obsah pole může být významově oddělen speciálními sekvencemi znaků (např. „.\\\"“). Při konverzích je velice často potřeba specifikovat část zdrojového pole (např. „část pole od posledního výskytu řetězce , : ‘ až do konce“).

Pro zápis a vyhodnocení takových podmínek jsou vhodné *regulární výrazy*. Regulární výraz je řetězec znaků, z nichž některé mají speciální význam a celkově tak vytváří vzor. Následně je možné testovat, jestli konkrétní řetězec znaků odpovídá danému vzoru (*matching*). Pokud ano, řetězec se naváže na daný regulární výraz. Po tomto navázání (např. obsahu pole **TinLibu**) je možné se odkazovat na jisté části řetězce (definované regulárním výrazem).

Existuje několik konkrétních syntaxí zápisu regulárních výrazů. Při definicích konverzních pravidel používáme syntaxi z jazyka *Perl*, protože poskytuje širší možnosti než většina ostatních.

Fakt, že se má v regulárním výrazu vyskytovat na daném místě konkrétní znak, se vyjádří znakem samotným (speciálnímu znaku musí předcházet znak „\\“). Chceme-li popsat povolenou skupinu znaků, jsou k dispozici např. následující deskriptory:

- „.“ vyjadřuje jakýkoli znak,

- „[abc]“ vyjadřuje jakýkoli znak a , b nebo c ,
- „[a-z]“ vyjadřuje jakýkoli znak od a do z včetně,
- „[^abc]“ vyjadřuje jakýkoli znak kromě a , b nebo c ,
- „\d“ vyjadřuje jakýkoli znak číslice, atp.

Tedy regulární výraz

Jan[_]Novák...[1-9]

odpovídá např. řetězci „Jan Novák : 7“ nebo také „Jan_NovákXYZ3“. Největší sílu dávají regulárním výrazům operátory opakování, které jsou aplikovány na předcházející atomický výraz:

- „?“ znamená, že předcházející výraz může a nemusí v řetězci vyskytovat,
- „*“ znamená prázdný řetězec nebo jakýkoli počet opakování předcházejícího výrazu,
- „+“ znamená jakýkoli počet opakování předcházejícího výrazu.

Tedy následující výraz popisuje řetězce začínající znakem „a“, poté jakýkoli řetězec libovolné délky následovaný znakem „b“ a opět libovolným řetězcem, který má na konci jeden nebo více znaků „c“:

a.*b.*c+

Operátory opakování jsou tzv. *hladové*, tzn. že se vždy naváží na co nejdelší řetězec. Tuto vlastnost lze u operátorů „*“ a „+“ lokálně změnit modifikátorem „?“ . Máme-li řetězec „abab“ a dva regulární výrazy „a.*b“ a „a.*?b“, tak první z nich se naváže na celý řetěz „abab“ a druhý jen na začátek „ab“.

Uzavřeme-li část regulárního výrazu do kulatých závorek, můžeme se po navázání výrazu na konkrétní řetězec odkazovat na část řetězce odpovídající části výrazu v závorkách. Odkazujeme se pomocí výrazu „\$n“, což znamená obsah n -té závorky regulárního výrazu. Například po navázání výrazu „(.*)(.*)“ na řetězec „Jan Novák“ se můžeme pomocí „\$1“ odkázat na „Jan“ a pomocí „\$2“ na „Novák“.

Znak „^“ v regulárním výrazu znamená začátek řetězce a znak „\$“ konec řetězce, resp. konec řádku. Tedy i přes výskyt modifikátoru „?“ se regulární výraz „^a.*?b\$“ pro řetězec „abab“ naváže na celé „abab“.

Možnosti regulárních výrazů v Perlu jsou ještě širší, ale vyjadřovací schopnosti uvedené syntaxe budou dostačovat pro většinu konverzních pravidel.

4.3 Formát konverzních pravidel

Důležitou součástí diplomové práce bylo vytvoření obecné podoby konverzních pravidel takové, aby mohla být všechna potřebná pravidla v této podobě vyjádřena. Protože na začátku práce neměl autor všechny konverzní tabulky a slovní popisy pravidel k dispozici, možnosti formátu pravidel byly v průběhu implementace pravidel rozšiřovány.

Jak jsme již zmínili, pravidla jsou zapisována v XML, jehož struktura je definována pomocí DTD. Popišme nyní tuto strukturu.

4.3.1 Jedno pravidlo – `item`

Implementovaná konverzní pravidla jsou často poměrně složitá, ale základem je téměř vždy nalezení nějaké přesně definované části pole **TinLibu**, jeho případná transformace a uložení do podpole nějakého MARCOvského pole. Právě operace tohoto typu mohou být definovány pomocí jednoho konverzního pravidla `item`. Příklad takového pravidla je

```
<item>
  <sourceField type="TLF_THIS"/>
  <sourceRegexp>^(.*)$</sourceRegexp>
  <targetField type="MF_THIS"/>
  <targetString>&#x7e;a$1.</targetString>
</item>
```

Element `sourceField` specifikuje zdrojové **TinLib** pole tohoto pravidla (viz 4.3.2), `sourceRegexp` obsahuje regulární výraz pro zdrojové pole (viz 4.2.3 a 4.3.3). Pokud zdrojové pole neexistuje nebo regulární výraz nelze spojit s jeho hodnotou, toto pravidlo se nepoužije. Dále element `targetField` určuje cílové pole vytvářeného MARC záznamu (viz 4.3.4) a `targetString` definuje formátovací řetězec, který se do cílového pole uloží (viz regulární výrazy v odstavci 4.2.3 a konkrétní popis elementu v 4.3.5).

Atomické pravidlo `item` může také obsahovat podmínku `condition` a pak se pravidlo provede pouze pokud je tato podmínka splněna.

```
<item>
  <condition type="C_REGEX">
    <TLField type="TLF_THIS"/>
    <regexp>.* : .*</regexp>
  </condition>
  (...)
</item>
```

Více o podmínkách v části 4.3.6. Často je při složitějších transformacích obsahu pole výhodné uložit modifikovaný obsah zpět do zdrojového pole. Toto lze v pravidle definovat zařazením elementu `backStoreString`, který má stejný formát jako `targetString`. Řetězec v tomto elementu přepíše hodnotu

zdrojového pole ze `sourceField`. Pokud je v pravidle `backStoreString`, nemusí se v pravidle vyskytovat elementy `targetField` a `targetString`.

V některých pravidlech se do MARCu ukládá řetězec nezávisle na polích **TinLibu**. Proto v pravidle nemusí nutně být ani elementy `sourceField` a `sourceString`. Celkově tedy DTD definice pro `item` vypadá takto:

```
<!ELEMENT item "(condition?,sourceField?,sourceRegexp?,
  targetField?,targetString?,backStoreString?)">
```

Povolené kombinace použitých elementů není možné zachytit pomocí DTD, proto jsou tyto podmínky přímo součástí programu **Tin2marc**.

Jednotlivá pravidla `item` jsou řazena do posloupností logicky příbuzných pravidel `sequence` – viz část 4.3.7.

4.3.2 Zdrojové pole – `TLField`

Přesně specifikovat pole **TinLibu**, se kterým se v daném pravidle pracuje, nemusí být v některých případech triviální. Kromě jednoduchého výběru pole konvertovaného záznamu podle klíče pole (např. `MTIT` pro pole *Název*) je někdy nutné vybrat právě to pole, které vyhovuje nějakému regulárnímu výrazu, nebo iterovat přes všechna pole s daným klíčem.

Při zapisování klíče pole nezáleží na jeho prvním písmenu. Stejná pravidla lze totiž často použít pro konverze více typů záznamů (např. monografie i seriály) a protože názvy ekvivalentních polí různých typů záznamů se liší vždy pouze v prvním písmenu, toto písmeno se ignoruje (viz 2.1).

Poměrně často se při konverzi nějakého pole jednoho záznamu používají pole ze záznamu jiného – z tzv. *propojeného záznamu*, což je záznam, jehož pole `XTIT` má stejnou hodnotu jako právě konvertované pole. Často je také velmi výhodné moci přímo ve specifikaci pole definovat, že se bude pracovat jen s jeho jistou částí (opět pomocí regulárního výrazu).

Skupina pravidel často pracuje s jedním vybraným polem a je tedy výhodné si jeho hodnotu uložit do zvláštního pole a využívat jej v celé sekvenci pravidel. Toto pole budeme nazývat *kontextové pole* (`contextTLField`). XML element specifikující pole **TinLibu** má atribut `type`, který může mít čtyři hodnoty – `TLF_THIS` pro kontextové pole, `TLF_TMP` pro pomocné pole, `TLF_THIS_REC` pro pole konvertovaného záznamu nebo `TLF_JOIN_REC` pro pole z propojeného záznamu (přes aktuální hodnotu kontextového pole). O tom jak se nastavují hodnoty kontextového a pomocného pole viz 4.3.7. U posledních dvou typů se klíč pole specifikuje v podelementu `key`:

```
<sourceField type="TLF_JOIN_REC">
  <key>MACC</key>
</sourceField>
```

Pokud je přítomen element `condRegexp` použije se takové pole s daným klíčem, které navíc vyhovuje regulárnímu výrazu v tomto elementu (pokud takové

pole existuje). V tomto elementu (stejně jako ve všech ostatních, které obsahují regulární výraz) se může vyskytovat element `TLField` (více viz 4.3.3).

```
<sourceField type="TLF_THIS_REC">
  <key>MOTX</key>
  <condRegexp>^<TLField type="TLF_THIS"/>; .*</condRegexp>
</sourceField>
```

Pokud je výhodné použít ze zdrojového pole jen část, je možné přidat elementy `regexp` a `matchString`. Ty specifikují regulární výraz, na který se hodnota pole naváže, a řetězec, který určí použitou část pole pomocí výrazů „\$n“ (viz část 4.2.3):

```
<sourceField type="TLF_THIS">
  <regexp>^.* : (.*)$</regexp>
  <matchString>$1.</matchString>
</sourceField>
```

Specifikovat pole **TinLibu** je třeba nejen v elementu `sourceField`, ale i na dalších místech. Proto je typ tohoto elementu v DTD definován pomocí parametrické entity `TLField` a tato entita je použita v definicích všech příbuzných elementů:

```
<!ENTITY % TLField "(key?,condRegexp?,regexp?,matchString?)">
<!ENTITY % TLFieldAtts "type CDATA #REQUIRED">
<!ELEMENT sourceField %TLField;>
<!ATTLIST TLField %TLFieldAtts;>
```

4.3.3 Regulární výraz – regexp

Elementy `regexp`, `condRegexp` a `sourceRegexp` jsou stejného typu a uplatní se na několika místech konverzních pravidel. Obsahují regulární výrazy podle syntaxe Perlu popsané v části 4.2.3. Navíc tyto elementy mohou obsahovat i podelementy `TLField`. Aktuální hodnota **TinLib** pole, specifikovaného pomocí `TLField`, je vložena na příslušné místo do regulárního výrazu. Pokud vkládaný text obsahuje nějaké speciální znaky (viz 4.2.3), je jim předřazen znak `\`, aby nebyly interpretovány jako funkční znaky regulárních výrazů.

```
<!ENTITY % RE "(#PCDATA | TLField)*">
```

Tato DTD syntaxe znamená, že v elementu s tímto typem se mohou volně mísit textové uzly s elementy `TLField`. Parametrická entita `RE` je použita v definicích elementů `regexp`, `condRegexp` a `sourceRegexp`.

4.3.4 MARC pole – MARCField

Často je nutné v pravidlech specifikovat pole `MARCu`, jeho podpole, případně přesnou pozici v návěští (viz odstavec o `MARCu` 2.2.1). Je to potřeba např.

pomocí elementu `targetField` v rámci definice jednoho pravidla (viz 4.3.1) nebo pomocí elementu `MARCFIELD` v podmínce `condition` (viz 4.3.6).

Z analogických důvodů, jako jsme v odstavci 4.3.2 zavedli pojem kontextového **TinLib** pole, existuje i *kontextové* MARC pole. Ve skupině pravidel, které vytváří jedno MARC pole, by totiž nebylo možné se na něj vždy odkazovat jen pomocí jeho tagu, neboť v záznamu může být více polí s jedním tagem. Proto se vždy na začátku řetězce příbuzných pravidel nastaví kontextové MARC pole a pravidla se na něj pak mohou odvolávat jednoduše.

Elementy `MARCFIELD` a `targetField` mají atribut `type`, který může mít tři hodnoty – `MF_THIS` pro kontextové pole, `MF_FIELD` pro jakékoli tagem specifikované pole nebo `MF_LEADER` pro návěští vytvářeného záznamu. Typ `MF_FIELD` musí obsahovat podelement `tag`, specifikující o které pole se jedná.

```
<targetField type="MF_FIELD">
  <tag>245</tag>
</targetField>
```

Pro pole typu `MF_THIS` a `MF_FIELD` je možné pomocí atributu `target` s povolenými hodnotami `indicator1` nebo `indicator2` stanovit, že se jedná o první resp. druhý indikátor pole.

```
<targetField type="MF_THIS" target="indicator1"/>
```

Pro návěští záznamu (`type="MF_LEADER"`) upřesňuje atribut `position` pozici v návěští – decimální číslo 0–23 (viz odstavec 2.2.1 o formátu MARC).

V odstavci 4.3.5 o elementu `targetString` jsou podrobné informace o tom, jak specifikovat podpole, do kterého se bude vkládat výsledek pravidla. Nyní řekněme jen to, že v konverzním pravidle může být také určeno, že se výsledek pravidla má vložit do vytvářeného pole „za poslední podpole \$x“. Proto element `targetField` může mít atribut `afterLastSubfield`, který upřesňuje toto podpole.

Element `MARCFIELD` může mít navíc podelement `subfield` určující podpole (viz `condition` v odstavci 4.3.6).

4.3.5 Cílový řetězec – `targetString`

V elementu `targetString` je formátovací řetězec určující výstup konverzního pravidla, který se uloží do vytvářeného MARC záznamu. `targetString` může obsahovat výrazy „\$n“, které se nahradí hodnotou *n*-té závorky regulárního výrazu z elementu `sourceRegex` (viz odstavec 4.3.1 Jedno pravidlo – `item` a odstavec 4.2.3 popisující regulární výrazy):

```
<targetString>$1 a $2.</targetString>
```

Obsah elementu `targetString` se tedy po navázání na regulární výraz uloží do MARC pole specifikovaného elementem `targetField`. Pokud chceme pomocí daného pravidla vytvořit v MARC poli podpole (jedno nebo

více) stačí na příslušná místa v `targetString` vložit separátory podpolí (viz odstavec 2.2.1 o MARCu). Tradiční znak „\$“ je již použit pro hodnoty závorek regulárního výrazu, a proto se pro separátor podpolí používá znak vlnka „~“, pro zvýraznění často zapisovaný pomocí znakové entity `~` (ASCII znak 7E hexadecimálně).

```
<targetString>&#x7e;a$1 :&#x7e;b$2</targetString>
```

Tento příklad uloží do podpole \$a hodnotu první závorky regulárního výrazu, oddělí podpole pomocí „ :“ a do podpole \$b uloží hodnotu druhé závorky. Element má také volitelný atribut pomocí kterého se vyjádří, že se nemá do MARCu uložit řetězec samotný ale jen jeho délka:

```
<targetString storeLength="true">$1</targetString>
```

4.3.6 Podmínka – condition

Podmínka `condition` může být prvním elementem jednoho pravidla `item` nebo posloupnosti pravidel `sequence`. Podmínka je vždy nejdříve ověřena a pravidlo nebo posloupnost se provede pouze pokud je splněna. Základní podmínka může testovat buď pole zdrojového záznamu **TinLibu** nebo pole či podpole vytvářeného MARC záznamu. Prvním podelementem tedy může být buď `TLField` popsany v 4.3.2 nebo `MARCField` popsany v 4.3.4.

```
<condition type="C_EXISTENCE">
  <MARCField type="MF_FIELD">
    <tag>245</tag>
    <subfield>h</subfield>
  </MARCField>
</condition>
```

Typ podmínky, určený pomocí atributu `type`, specifikuje, jestli bude podmínka testovat jen existenci objektu nebo jestli bude testovat jeho hodnotu pomocí regulárního výrazu (`type="C_REGEX"`):

```
<condition type="C_REGEX">
  <TLField type="TLF_THIS_REC">
    <key>MACC</key>
  </TLField>
  <regexp>.*\.\\\\\\\\\ .*</regexp>
</condition>
```

Tyto atomické podmínky mohou být skládány pomocí logické konjunkce a/nebo disjunkce. Podmínky s atributem `type="C_AND"` nebo `type="C_OR"`, obsahující další podelementy `condition`, vyjadřují právě tyto logické operátory. Tyto podmínky jsou vyhodnocovány „líně“, tzn. že pro disjunkci se už nevyhodnocují podmínky následující za první splněnou podmínkou, a naopak konjunkce podmínek se vyhodnocuje jen do první nesplněné podmínky.

```
<condition type="C_AND" negation="true">
  <condition type="C_OR">
    <condition type="C_REGEX" > (...) </condition>
    <condition type="C_REGEX" > (...) </condition>
  </condition>
  <condition type="C_EXISTENCE" > (...) </condition>
</condition>
```

V tomto příkladu si všimněme i atributu `negation="true"`, který vyjadřuje logickou negaci podmínky. V pravidlech může být také podmínka, která požaduje např. existenci n polí zdrojového záznamu. Element `condition` má tedy volitelný parametr `minOccurrence` vyjadřující minimální počet polí (podpolí), pro které musí být podmínka splněna.

```
<condition type="C_REGEX" negation="true" minOccurrence="4">
  <TLField type="TLF_THIS_REC">
    <key>MAUT</key>
  </TLField>
  <regex>.* : .*</regex>
</condition>
```

4.3.7 Posloupnosti pravidel – sequence

Implementovaná konverzní pravidla je často potřeba spojovat do logických celků podobných programovacím konstrukcím IF–THEN–ELSE, SWITCH, WHILE nebo FOR. Použití těchto mechanismů i další možnosti poskytuje seškupování pravidel pomocí `sequence`.

Typ elementu `sequence` má následující DTD definici:

```
<!ELEMENT sequence "(condition?, setContextTLField?,
setTmpTLField?, setContextMARCField?, (item|sequence)*)">
```

Prvním elementem v posloupnosti může být podmínka `condition`. Jejím splněním je podmíněno další vyhodnocování `sequence` (odstavec 4.3.6). Poté mohou následovat elementy `setContextTLField` a `setTmpTLField` nastavující kontextové a/nebo pomocné **TinLib** pole (viz odstavec 4.3.2). Tyto elementy jsou typu `TLField` (také viz 4.3.2).

```
<sequence type="S_ALL">
  <condition type="C_REGEX" > (...) </condition>
  <setContextTLField type="TLF_JOIN_REC">
    <key>MCOF</key>
  </setContextTLField>
  <setTmpTLField type="TLF_THIS"/>
  (...)
</sequence>
```

V tomto příkladu se, pokud je splněna podmínka, nastaví do kontextového i do pomocného pole (`TLF_THIS` a `TLF_TMP`) obsah pole `MCOF` z propoje-

ného záznamu. Chceme-li po provedení sekvence pravidel vrátit do kontextového a/nebo pomocného pole hodnoty, které tam byly před jejich přenastavením, stačí v sekvenci použít atributy `restoreTLContextAfter="true"` resp. `restoreTmpContextAfter="true"`.

Pomocí `setContextMARCField` se nastaví kontextové MARC pole (viz 4.3.4). Tento element je následujícího typu:

```
<!ELEMENT setContextMARCField "(condition?, tag, elseTag?)">
<!ATTLIST setContextMARCField useExistingField CDATA #IMPLIED>
```

Většinou `setContextMARCField` obsahuje pouze povinný podelement `tag` specifikující MARC pole, které se má stát kontextovým. Obsahuje-li podmínku a `elseTag`, použije se `elseTag`, jestliže podmínka není splněna. Pokud element obsahuje atribut `useExistingField="true"`, preferuje se použití již existujícího MARC pole vytvářeného záznamu (vždy to poslední vytvořené). Bez použití atributu je v záznamu vytvořeno nové MARC pole vždy.

Po této úvodní části následuje v sekvence posloupnost jednotlivých pravidel `item` a/nebo posloupností sekvence (dohromady budou označovány jako pravidla). Mechanismus jejich vyhodnocování závisí na typu posloupnosti, který je určen pomocí atributu `type`. Pokud je posloupnost typu `S_ALL`, provedou se všechna jeho pravidla. Taková posloupnost odpovídá bloku příkazů v klasickém programování.

Dalším typem posloupnosti je `type="S_FIRST"`. V tomto případě se provede pouze první pravidlo posloupnosti, jehož podmínka je splněna (resp. které není uvozené podmínkou). To odpovídá programovací konstrukci `SWITCH`.

Sekvence typu `type="S_ALL_REPEAT"` a `type="S_FIRST_REPEAT"` se po provedení všech podpravidel (resp. prvního vyhovujícího) začnou vyhodnocovat znovu. Taková posloupnost musí mít vstupní podmínku `condition`. Pokud podmínka není splněna, posloupnost se již neprovádí. Je na tvůrci pravidel, aby zajistil, že cyklus opakující tuto sekvenci podmínek skončí (podmínka nebude platit stále). Tyto typy posloupností odpovídají programovací konstrukci `WHILE-DO`.

```
<sequence type="S_ALL_REPEAT">
  <condition type="C_REGEX">
    <TLField type="TLF_THIS"/>
    <regexp>.* : .*</regexp>
  </condition>
  <item>
    <sourceField type="TLF_THIS"/>
    <sourceRegexp>(.*):(.*)</sourceRegexp>
    <backStoreString>$1 -- $2</backStoreString>
  </item>
</sequence>
```

V tomto příkladu se všechny řetězce „ : “ v kontextovém **TinLib** poli nahradí za „ -- “.

Mnoho konverzních pravidel je třeba aplikovat na všechna zdrojová **TinLib** pole s daným klíčem (např. všechna pole MACC). Toho lze dosáhnout použitím atributu `repeatableTLField="true"` elementu `sequence` (pak `sequence` musí obsahovat podelement `setContextTLField`). Daná sekvence se provede pro všechna pole specifikovaná v `setContextTLField`.

Často je potřeba použít skupinu konverzních pravidel na více místech konverzí. Bylo by neefektivní taková pravidla psát více než jednou nebo je v rámci souboru s pravidly kopírovat. Proto každá posloupnost pravidel může být „zavolána“ jako procedura v klasickém programování. Atributem `id` se sekvenci přiřadí jednoznačný identifikátor, pomocí kterého se na sekvenci lze odkazovat (jméno procedury).

Pro opakované použití již definované sekvence pravidel slouží typ `type="S_CALL"`. Atributem `call="id_sekvence"` se určí volaná sekvence. Pokud má volající sekvence podelementy `setContextTLField`, `setTmpTLField` a/nebo `setContextMARCFIELD`, pak se tyto elementy vloží do volané sekvence (existující se nahradí). To odpovídá v jistém smyslu předávání parametrů proceduře.

```
<sequence type="S_CALL" call="MTIT">
  <setContextTLField type="TLF_THIS_REC">
    <key>MOTI</key>
    <condRegexp>.* \ (Souběž\.\.)$</condRegexp>
  </setContextTLField>
  <setContextMARCFIELD>
    <tag>246</tag>
  </setContextMARCFIELD>
</sequence>
```

Je-li ve volající sekvenci podmínka, otestuje se ještě před vlastním provedením volání. Podmínka ve volané posloupnosti se netestuje.

Při implementaci daných konverzních pravidel se autor setkal s pravidly, jejichž charakter se zcela vymykal vytvářené koncepci. Příkladem může být např. „výběr takového pole MEDT, které obsahuje nejnovější datum“. Nebo pravidlo říkající, že „vždy po odříznutí jistého řetězce ze začátku pole je třeba první písmeno zbytku pole zvětšit (na verzálku)“.

Rozšiřovat formát pravidel tak, aby v něm bylo možné zapsat i takto různorodá pravidla by nebylo efektivní. Při každém výrazně odlišném pravidle by bylo nutno přidat nějaký jednoúčelový konstrukt a zamýšlená koncepce by se zákonitě začala vytrácet. Na druhou stranu implementovat tato konkrétní pravidla na úrovni programu **Tin2marc** je samozřejmě možné. Pravidla jsou tedy přímo naprogramována a provedení těchto částí kódu se aktivuje sekvencí typu `type="special"`. Atributem `special="klicove_slovo"` se specifikuje konkrétní speciální pravidlo:

```
<sequence type="S_SPECIAL" special="specialToUpper"/>
```

Toto pravidlo např. změní první písmeno kontextového pole na verzálku.

4.3.8 Poznámky

Při vytváření pravidel a testovacím běhu konverzního programu může být výhodné přesně sledovat, která pravidla byla použita a v jakém pořadí. Režim výpisu použitých pravidel může uživatel jednoduše zapnout (viz odstavec 5.4). Pravidla jsou ve výpisu identifikována buď pomocí volitelného atributu name (pro elementy item i sequence), pro sekvence také podle id nebo type:

```
sequence with name '245'
  sequence with id 'MTIT_indicator2'
    rule item with name 'set to 0'
  sequence of type 'S_ALL'
    rule item
```

4.4 Příklad konverzního pravidla

Uvedme si nyní příklad sady konverzních pravidel pro pole MISB (pole obsahující ISBN). V tabulce 4.1 je slovní popis, který byl zadáním pro vytvoření pravidla (viz příloha 1). Každý řádek tabulky sice popisuje v jistém smyslu jedno pravidlo (většinou týkající se jednoho podpole MARCu), ale téměř vždy je třeba dodat ještě čistě slovní popis.

Tinlib					MARC21						
pole	podpole	interpunk. předchází podpole	interpunk. ukončuje podpole	interpunk. na konci pole	pole	O N	1. ind	2. ind	interpunk. předchází podpole	pod-pole	O N
MISB					020	O	#	#			
	Číslo ISBN									\$a	N
	Vysvětlivka	_()									
	Dostupnost	_:_							_:	\$c	N
	Chybné číslo			_(chybné) _(chyb.)						\$z	O

Každý výskyt pole MISB se převádí do samostatného pole 020 s výjimkou čísel, která mají sufix „ (chybné) “ nebo „ (chyb.) “ – ta se připojí jako \$z do předchozího pole 020. Do podpole \$z se přenáší pouze číslo, sufix „ (chybné) “ nebo „ (chyb.) “ se nepřenáší (počet mezer před sufixem je různý). Spojovníky, které jsou součástí čísla se odstraní
Pozor interpunkce „ : “ může být také součástí podpole Vysvětlivka tj. nachází se v () – v tom případě se k ní při konverzi nepřihlíží – pozor na případy typu „ (*) : * (*) “

Tabulka 4.1: popis pravidla pro konverzi pole MISB

Protože **TinLib** nemá nijak standardně vyznačovaná podpole (tak jako MARC), rozeznávají se části pole pouze podle speciálních sekvencí znaků, které tyto části předchází resp. následují (viz 3. a 4. sloupec tabulky). Dále má-li se nějaké pole konvertovat jiným způsobem než ostatní, pozná se to často podle jeho sufixu (speciální interpunkce na konci pole – 5. sloupec). Znakem _ v tabulce je vždy myšlena mezera.

7. a 12. sloupec s návěštím „ON“ určuje, jestli je pole resp. podpole MARCu, popisované na daném řádku, opakovatelné nebo jedinečné (v rámci záznamu resp. v rámci pole). 8. a 9. sloupec určují hodnoty indikátorů vytvářeného pole. Významy ostatních sloupců jsou poměrně zřejmé z jejich návěstí.

Následuje konverzní pravidlo vytvořené na základě této tabulky. Pravidlo je přesně převzaté ze souboru `monografie.xml` (příloha 4). Za tímto příkladem následuje slovní rozbor jeho struktury a významu.

```
<!-- prevod poli MISB -->
<sequence type="S_FIRST" repeatableTLField="true">
  <setContextTLField type="TLF_THIS_REC">
    <key>MISB</key>
  </setContextTLField>

  <!-- pokud se jedna o chybné ISBN -->
  <sequence type="S_ALL">
    <condition type="C_REGEX">
      <TLField type="TLF_THIS"/>
      <regex>^.* \((?:chybné|chyb\.)\)\)$</regex>
    </condition>
    <setContextMARCField useExistingField="true">
      <tag>020</tag>
    </setContextMARCField>
    <!-- odstranit spojovniky z ISBN -->
    <sequence type="S_ALL_REPEAT" id="odstranSpojovniky">
      <condition type="C_REGEX">
        <TLField type="TLF_THIS"/>
        <regex>[^\ ]*-*</regex>
      </condition>
      <item>
        <sourceField type="TLF_THIS"/>
        <sourceRegex>^([^\ ]*)-(.*)$</sourceRegex>
        <backStoreString>$1$2</backStoreString>
      </item>
    </sequence> <!-- konec odstranovani spojovniku -->
    <!-- ulozeni do podpole $z posledniho vytvoreneho pole 020 -->
    <item>
      <sourceField type="TLF_THIS"/>
      <sourceRegex>^(.*?) ? ?\((?:chybné|chyb\.)\)\)$</sourceRegex>
      <targetField type="MF_THIS"/>
      <targetString>&#x7e;z$1</targetString>
    </item>
  </sequence>

  <!-- ostatni (spravna) ISBN -->
  <sequence type="S_ALL">
    <setContextMARCField>
      <tag>020</tag>
```

```

</setContextMARCField>
<!-- prevedeni VSECH sekvenci < : > na podpole $c -->
<sequence type="S_FIRST_REPEAT">
  <condition type="C_REGEX">
    <TLField type="TLF_THIS"/>
    <regex>^.* : .*$/regex>
  </condition>
  <item>
    <sourceField type="TLF_THIS"/>
    <sourceRegex>^(.*) : (.*)$/sourceRegex>
    <backStoreString>$1 :&#x7e;c$2</backStoreString>
  </item>
</sequence>
<!-- odstraneni znacek podpoli $c pokud jsou uprostred zavorky
      (poznanky) - lepe to asi nejde -->
<sequence type="S_FIRST_REPEAT">
  <condition type="C_REGEX">
    <TLField type="TLF_THIS"/>
    <regex>^.*\([^\\]*? :&#x7e;c.*?\).*$/regex>
  </condition>
  <item>
    <sourceField type="TLF_THIS"/>
    <sourceRegex>^(.*\([^\\]*? :)&#x7e;c(.*)$/sourceRegex>
    <backStoreString>$1 $2</backStoreString>
  </item>
</sequence>
<!-- odstraneni vsech spojovniku "-" z ISBN -->
<sequence type="S_CALL" call="odstranSpojovniky"/>
<!-- umistení znacky $a na zacatek a ulozeni do MARCu -->
<item>
  <sourceField type="TLF_THIS"/>
  <sourceRegex>^(.*)$/sourceRegex>
  <targetField type="MF_THIS"/>
  <targetString>&#x7e;a$1</targetString>
</item>
</sequence> <!-- konec casti pro prevod spravnych ISBN -->
</sequence> <!-- konec prevodu poli MISB -->

```

Popišme si nyní strukturu tohoto pravidla. Ze zadání plyne, že zdrojové pole MISB se bude konvertovat dvěma různými způsoby podle toho, jestli pole obsahuje sufix `_` (chybné) (resp. `_` (chyb.)) nebo ne. Kořenová sekvence pravidla je proto typu `type="S_FIRST"` a obsahuje dvě podsekvence, z nichž se provede vždy právě jedna (viz 4.3.7). Díky atributu `repeatableTLField` se pravidlo aplikuje postupně na všechna pole určená pomocí `setContextTLField`. Celé pravidlo se nyní už provádí nad kontextovým **TinLib** polem nastaveným tímto elementem.

První podsekvence má podmínku, která pomocí regulárního výrazu za-

jistí použití sekvence pouze pro chybná ISBN. Taková pole se mají připojit do předchozího pole 020, což je zařízeno elementem `setContextMARCField` s atributem `useExistingField` (viz 4.3.7).

Než se obsah pole uloží do MARCu, musí se odstranit spojovníky „-“ z čísla, což má na starost sekvence s `id="odstranSpojovniky"`. Je typu `S_ALL_REPEAT`, provádí se tedy v cyklu, dokud platí její vstupní podmínka. Ta kontroluje, jestli je přímo v čísle (tedy do první mezery) ještě nějaký spojovník. Pokud ano, pomocí jednoduchého pravidla se spojovník odstraní a modifikovaná hodnota se uloží zpět do kontextového pole.

Následující pravidlo již uloží upravené chybné ISBN do podpole `z` kontextového MARC pole. Do MARCu se neukládají popsané sufisy. Syntaxe `(?:chybné|chyb\.)` v regulárních výrazech znamená chybné nebo chyb..

Nyní následuje sekvence pro ostatní (tedy ne chybná) pole MISB. Pro ně se vytvoří vždy nové pole 020 jako kontextové MARC pole. Opět se nejdříve modifikuje řetězec vstupního pole a teprve pak se uloží do MARCu. Všechny sekvence `_:_` znamenají začátek podpole `z` v MARCu, kromě těch, které jsou v závorkách.

Nejdříve se všechny `_:_` převedou na značky `~c` a pak se `~c`, které by eventuálně byly v závorce, převedou zase zpět na `_:_`. Tento zdánlivě krkolomný postup se jeví být jednak nejkratším z hlediska délky zápisu pravidel, ale také zdaleka nejrychlejším z hlediska doby vyhodnocování pravidel, protože sekvence `_:_` se v závorce objevuje velice zřídka.

Poté se opět odstraní spojovníky z čísla na začátku pole (zavolá se již vytvořená posloupnost `odstranSpojovniky`) a modifikované pole se uloží do MARCu.

Takto by mohla vypadat zdrojová pole **TinLibu** a výstup jejich konverzí pomocí těchto pravidel:

```
MISB:0-7043-3831-9 (pbk) : P2.95
MISB:0-8078-4312-1 (pbk. : alk. paper)
MISB:0-7043-2824-0 (chybné)
```

```
020 ## $a0704338319 (pbk) :$cP2.95
020 ## $a0807843121 (pbk. : alk. paper)$z0704328240
```

Toto pravidlo je oproti většině ostatních velice jednoduché a přímočaré. V přílohách 1 a 3 jsou slovní popisy všech konverzních pravidel pro monografie a příloha 4 (soubor `monografie.xml`) obsahuje konverzní pravidla, která vznikla z tohoto popisu.

Uvedený příklad se skládá celkem ze sedmi sekvencí a pěti atomických pravidel. Celý soubor pravidel pro monografie má více než 350 sekvencí a 550 jednotlivých pravidel. Kolekce vytvořená pro konverzi seriálů je o něco menší a značná část těchto pravidel se shoduje pro oba typy záznamů.

Kapitola 5

Program Tin2marc

Oddíl 4.3 se zabýval vytvořeným formátem konverzních pravidel. Program **Tin2marc** lze chápat jako interpret pravidel v uvedeném formátu, který realizuje popsane konverze na požadovaných datech.

5.1 Programovací jazyk

Na první pohled by se mohlo zdát, že konverze formátu bibliografického záznamu bude znamenat provedení několika textových substitucí pomocí regulárních výrazů a drobnější úpravy typu přejmenování polí. Pro takový úkol by byl jistě vhodný např. nějaký skriptovací jazyk.

Konkrétně problém konverzí **T-Series** záznamů do MARC21 je ale mnohem komplexnější. V odstavci 4.3 popsána pravidla poskytují poměrně široké možnosti jak definovat konverzní operace nad vstupními daty a např. ze souboru `monografie.xml` (příloha 4) je zřejmé, že všechny tyto možnosti jsou plně využity.

Pro interpretaci těchto pravidel je vhodnější použít standardní objektový jazyk. Pro každou složku pravidel (viz odstavce 4.3.1 až 4.3.7) se vytvoří objektová třída, která bude zapouzdřovat celou funkčnost této složky. Po načtení konkrétního souboru s pravidly budou tato pravidla rozložena do objektové struktury reprezentující funkčnost celého konverzního systému (viz 5.2).

Jedním z vhodným jazyků pro tento úkol je *Java*. Tento jazyk navíc standardně obsahuje komplexní nástroje pro práci s XML daty (např. načtení XML jako DOM – viz 4.2.1). Nezanedbatelnou výhodou Javy je běh programu v prostředí tzv. Java Virtual Machine, díky němuž se při vytváření a kompilaci programu nemusí stanovit cílová platforma, na které program poběží (viz 5.1.1).

5.1.1 Java

Jazyk Java založila v roce 1995 společnost *Sun Microsystems*. Je to objektově orientovaný programovací jazyk, syntaxí podobný jazyku C++. Je však o něco jednodušší, rozdílly jsou např. v tom, že se programátor nemusí starat o správu paměti, kromě proměnných několika základních typů jsou všechny proměnné ukazateli na objekty a jsou automaticky dereferencovány. Dále Java neumožňuje

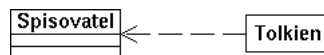
vícenásobnou dědičnost, šablony nebo přetěžování standardních operátorů. Více o jazyce Java např. viz [6].

Javu lze chápat také jako platformu v tom smyslu, že (stejně jako operační systémy) knihovny Javy poskytují programátorům aplikací ucelené API (*Application Programmers Interface*, rozhraní pro aplikační programátory). Toto rozhraní je stejné pro všechny operační systémy a jeho jednotlivá volání jsou za běhu programu překládána pro aktuální platformu. Prostředí, ve kterém Java programy běží a které emuluje platformovou nezávislost, se nazývá *virtuální stroj* – Java Virtual Machine.

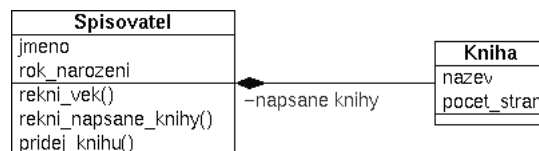
5.1.2 Objektově orientované programování

Java je čistě objektový programovací jazyk a v této kapitole budeme často používat terminologii *Objektově orientovaného programování* (OOP). Proto zopakujeme základní pojmy OOP a sjednotíme používanou terminologii a značení. OOP diagramy v této kapitole budou používat notaci UML (Unified Modeling Language – podrobně o UML viz např. [7]).

V prostředí OOP označuje *objekt* (*object*) abstraktní entitu, která má nějaké vlastnosti (*stav*), nějaké definované chování a také identitu. OOP objekt bývá v programu často obrazem nějakého objektu reálného světa. Skupina objektů, které mají stejnou strukturu a chování se nazývá *třída* (*class*). Třídou může být např. `Spisovatel` a objektem této třídy (také říkáme instancí) je např. J. R. R. Tolkien.



Vlastnostem evidovaným v rámci objektu budeme říkat *atributy* (*attribut*) (neboli členské proměnné); třída `Spisovatel` může mít např. atributy `jmeno`, `datum_narozeni`, `napsane_knihy` atp. Operacím, které lze realizovat s jednotlivými objekty (a které jsou definované vždy v rámci třídy), říkáme *metody* (*method*). Pro třídu `Spisovatel` by to mohly být např. „řekni kolik jsi napsal knih“ (`rekni_napsane_knihy()`), „přidej novou knihu“ (`pridej_knihu()`) nebo „řekni svůj věk“ (`rekni_vek()`).

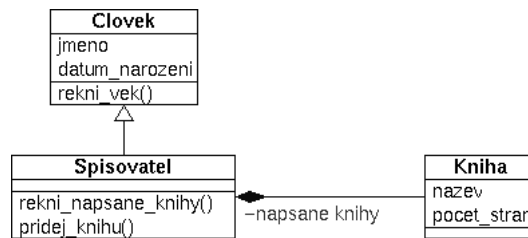


Seznam napsaných knih zde není jednoduchý atribut, ale je to kolekce objektů typu `Kniha`.

Pokud chceme explicitně popsat atribut nebo metodu nějaké třídy nebo instance, můžeme použít zápis `objekt.atribut` resp. `třída.metoda()`, např. tedy `Spisovatel.jmeno`, `Tolkien.rekni_vek()` atp.

Samotné atributy objektu jsou v jistém smyslu před „okolním světem“ ukryté a lze k nim přistupovat pouze prostřednictvím metod objektu. Např. se nemůžeme podívat přímo na datum narození spisovatele, ale můžeme se zeptat na jeho věk. Tomuto principu se říká *zapouzdření tříd* (*encapsulation*).

Dalším důležitým pojmem OOP je *dědičnost tříd* (*inheritance*). Třída objektů může dědit od jiné třídy všechny její atributy a metody a dále je rozšířit. Zůstaneme-li u příkladu třídy `Spisovatel`, tato třída by mohla být zděděna ze třídy `Člověk` a atributy jako `jmeno` nebo `datum_narozeni` by byly definované již ve třídě `Člověk`. Pouze atributy a metody týkající se knih atp. by byly přidány ve třídě `Spisovatel`.



Oblast OOP je poměrně rozsáhlá a stále se vyvíjí, ale zmíněné pojmy budou postačovat k popsání objektové struktury vytvářeného konverzního systému.

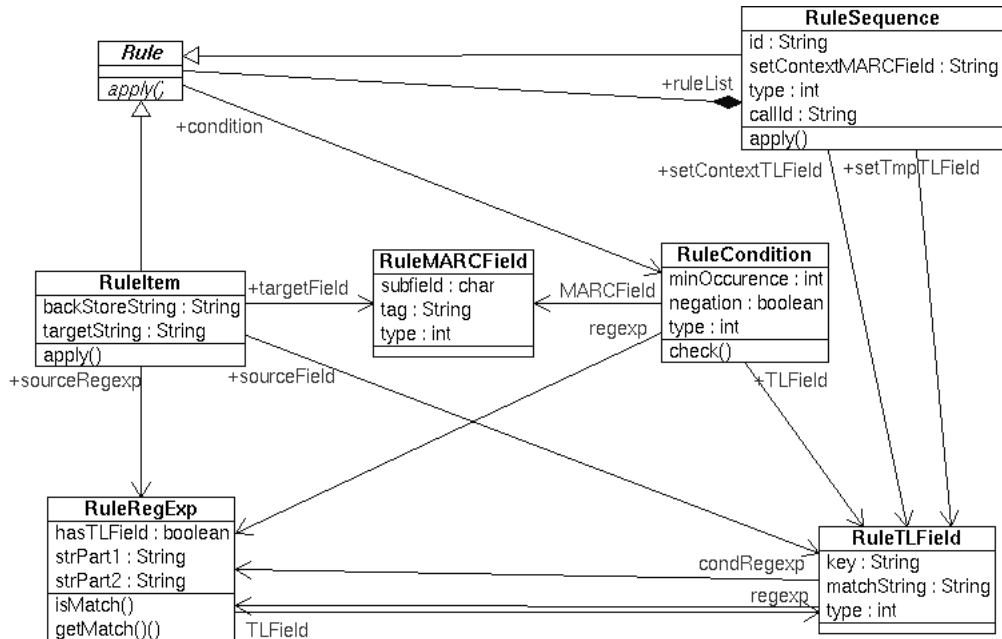
5.2 Objektový návrh

Při návrhu struktury programu pro objektový programovací jazyk je stěžejním úkolem nalezení tříd a jejich vztahů – tzv. *diagram tříd* (*class diagram*). Velká část tříd v systému často koresponduje s „reálnými“ objekty z aplikační oblasti.

K nalezeným třídám se hledají jejich atributy a metody a poté vztahy mezi třídami. Základní vztahy jsou dědičnost (předek – potomek), asociace (třídy jsou propojené, jeden objekt používá služby jiného) a agregace (vztah mezi celkem a jeho částmi).

Objektový diagram vytvořeného konverzního systému bychom mohli rozdělit na dvě logické části. První je skupina tříd, které reprezentují položky konverzních pravidel, které byly popsány v odstavci 4.3. Názvy těchto tříd mají prefix `Rule`. Na obrázku 5.1 je diagram těchto tříd.

Jak je vidět z diagramu i z popisu pravidel, jednotlivé elementy pravidel jsou navzájem silně provázané. Popišme si nyní tyto třídy a jejich vazby.



Obrázek 5.1: : diagram tříd konverzních pravidel

Třída Rule

Třída Rule je tzv. abstraktní předek pro třídy RuleItem (odpovídá XML elementu item) a RuleSequence (element sequence). Abstraktní třída znamená, že nelze vytvářet její instance. To koresponduje i s intuitivním výkladem – můžeme říci *pravidlo* a myslet tím jednotlivá pravidla a posloupnosti pravidel dohromady, ale vytvořit můžeme vždy právě jednu z těchto variant.

Elementy item i sequence mohou mít vstupní podmínku, a proto je třída Rule asociována se třídou RuleCondition (viz odstavce 4.3.1 a 4.3.7).

Třída RuleItem

Již víme, že tato třída reprezentuje jedno konverzní pravidlo (XML element item). Všechny na diagramu naznačené asociace a atributy odpovídají povoleným podelementům pravidla (včetně jejich názvů). Např. tedy element sourceField je typu RuleTLField tak, jak je popsáno v odstavci 4.3.1.

Třída RuleSequence

Tato třída zapouzdřuje jeden element sequence. Všechny jeho atributy opět odpovídají povoleným podelementům nebo atributům elementu sequence

tak, jak byly popsány v části 4.3.7. Všimněme si především atributu `ruleList`, který ukazuje na posloupnost objektů typu `Rule`, resp. potomků této třídy.

Třída `RuleCondition`

`RuleCondition` popisuje XML element pro podmínku `condition`. Z popisu tohoto elementu v odstavci 4.3.6 plyne, že podmínka může buď testovat **TinLib** pole (asociace na třídu `RuleTLField`) nebo MARC pole (`RuleMARCFIELD`). Může mít také podelement s regulárním výrazem (`RuleRegExp`).

Třída `RuleMARCFIELD`

Elementy typu `MARCFIELD` (viz 4.3.4) nemají příliš složitou strukturu a neobsahují žádné podelementy, pro které je definována samostatná třída. Proto jsou všechny atributy třídy `RuleMARCFIELD` základních typů.

Třída `RuleTLField`

XML elementy popsány v odstavci 4.3.2 popisují **TinLib** pole nebo jeho část. Asociace se třídou `RuleRegExp` odpovídají podelementům `condRegexp` a `regexp`.

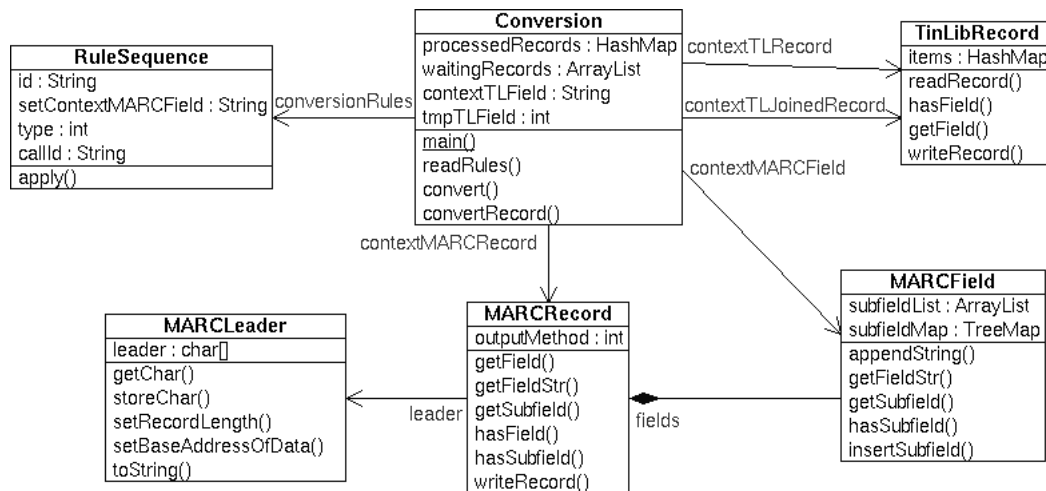
Třída `RuleRegExp`

Poslední třídou této části diagramu tříd je `RuleRegExp`. Pro regulární výrazy byla vytvořena speciální třída z toho důvodu, že XML element s reg. výrazem může i mít podelement `TLField`. Obsah tímto elementem specifikovaného pole se samozřejmě mění v závislosti na konvertovaném záznamu (viz odstavec 4.3.3).

Kromě těchto tříd pro popis konverzních pravidel, existuje v **Tin2marc** další skupina tříd, které zajišťují samotný běh konverzního programu, reprezentují záznam **TinLibu**, MARC záznam atp. Na obrázku 5.2 je diagram těchto tříd a jejich propojení s třídami pro konverzní pravidla.

Třída `TinLibRecord`

Jak název napovídá, objekty této třídy reprezentují jeden konkrétní záznam **TinLibu**. Tělem této třídy je `items` – seznam všech polí záznamu dostupných přes klíč pole (např. „MTIT“) – viz kapitola 2.1 o **TinLibu**. Záznam lze načíst ze souboru pomocí metody `readRecord()` a zapsat na výstup pomocí `writeRecord()`. Vstupní i výstupní formát je textový export popsáný v 2.1.



Obrázek 5.2: : diagram ostatních tříd

Třída MARCRecord

Tato třída popisuje jeden MARC záznam, je tedy v programu používána pro shromažďování výstupních dat konverze jednoho záznamu. Po dokončení konverze jsou tato data vytisknuta na výstup. Protože pole MARC záznamu jsou dále dělena na podpole (viz 2.2.1), je pro každé pole vytvořen objektu typu MARCField a tyto objekty jsou v MARCRecord uloženy v asociativním poli `fields` podle tagu daného pole.

Návěští záznamu je obsahem objektu typu MARCLeader. Kromě metod pro manipulaci s obsahem polí a návěští je ve třídě MARCRecord definována metoda `writeRecord()`, která zapíše MARC záznam na výstup. Možnými výstupními formáty jsou výměnný formát podle ISO 2709, řádkový MARC21 nebo Aleph MARC21 (odstavce 2.2.2, 2.2.3 a 2.3).

Třída MARCField

Objekty třídy MARCField popisují pole MARC záznamu. Protože podpole mohou být v poli v různém pořadí, má třída jednak atribut `subfieldMap` (podpole jsou zde přístupná pomocí identifikátoru podpole) a jednak `subfieldList`, což je seznam určující pořadí podpolí v poli. Dále třída obsahuje atributy určující indikátory. Přirozené jsou metody, které vkládají data do pole nebo naopak vyhodnocují dotazy na obsah jednotlivých podpolí. Metoda `getFieldStr()` sestaví řetězec tohoto pole tak, jak se zapíše na výstup (opět podle typu výstupního formátu).

Třída MARCLoader

Tato jednoduchá třída obsahuje informace o návěští MARC záznamu. Kromě přirozených metod pro manipulaci se znaky na jednotlivých pozicích návěští obsahuje třída jen několik speciálních metod a také `getFieldStr()` vytvářející řetězec pro výstup.

Třída Conversion

V každé Java aplikaci musí existovat třída mající statickou metodu `main()`, která je vstupním bodem programu. V programu **Tin2marc** je to právě třída `Conversion`. Tato třída zajišťuje běh konverzního programu. Průběh výpočtu programu a s ním i metody třídy `Conversion` jsou popsány v odstavci 5.3, nyní si popíšeme pouze některé její atributy.

Atributy `contextTLRecord` a `contextMARCRecord` ukazují na aktuální konvertovaný **TinLib** záznam a na vznikající MARC záznam. Atribut `contextTLJoinedRecord` ukazuje na aktuální propojený záznam (viz 4.3.2), `contextTLField` a `contextMARCField` na kontextové **TinLib** resp. MARC pole (viz 4.3.2 a 4.3.4).

5.3 Průběh výpočtu

Jak již bylo naznačeno, `Conversion` je třída, která zajišťuje celý průběh konverzí. V její metodě `main()`, která je prováděna jako první, se nejdříve zpracují parametry předané programu a zkontroluje se jejich validita (parametry jsou popsány v odstavci 5.4). Pokud nejsou platné, běh programu se ukončí.

Pokud jsou parametry v pořádku, je mezi nimi i soubor s XML popisem pravidel. XML data jsou načtena jako DOM strom (viz 4.2.1) a pomocí něj se vytvoří struktura objektů, která odpovídá těmto pravidlům a tvoří jejich „obraz v paměti“ (viz diagram tříd na obr. 5.1). Tato pravidla jsou přístupná pomocí atributu `Conversion.conversionRules`.

Pokud mají načítaná pravidla správný formát, je zavolána metoda `Conversion.convert()`. Ta vždy načte jeden **TinLib** záznam ze vstupního souboru do `contextTLRecord` a aplikuje na něj konverzní pravidla – zavolá se metoda `conversionRules.apply()`. Pokud je vše v pořádku, vytiskne se na výstup obsah `contextMARCRecord`. V každém případě se do seznamu `processedRecords` uloží obsah pole MTIT tohoto záznamu (MTIT je jednoznačným identifikátorem každého záznamu).

V odstavci 4.3.2 byl zmíněn fakt, že při konverzi záznamu je často potřeba obsah pole z jiného záznamu. Tento propojený záznam je určen tak, že obsah jeho pole MTIT se shoduje s obsahem právě konvertovaného pole. Mohou nastat dvě možnosti:

1. Pokud byl propojený záznam již zpracován, jeho MTIT se najde v poli `processedRecords`, záznam je vyhledán ve vstupním souboru, načten a hodnota požadovaného pole vyhledána.
2. Pokud propojený záznam ještě nebyl zkonvertován, nemůže být konverze aktuálního záznamu dokončena, a proto je přidán do pole `waitingRecords`.

Po dokončení „řádných“ konverzí se znovu konvertují záznamy ze seznamu `waitingRecords`. Nyní už jsou k dispozici data všech záznamů ze vstupního souboru. Pokud se stane, že požadovaný propojený záznam ve vstupním souboru vůbec není, konverze aktuálního záznamu se přeruší, záznam se uloží do souboru se nezkonvertovanými záznamy a informace o jeho nedokončené konverzi je uložena do logu (viz 5.3.1).

Vlastní konverze jednotlivých záznamů probíhají přesně podle mechanismů, které jsou popsány v části o formátu konverzních pravidel (4.3). Každé položce pravidel (každému XML elementu) odpovídá objekt příslušné třídy (třídy s prefixem `Rule`). Tyto třídy v sobě obsahují požadovanou funkcionalitu. Do konkrétních implementačních detailů těchto tříd již nebudeme zabíhat.

5.3.1 Logování

Program **Tin2marc** při svém běhu generuje zprávy o své činnosti, které jsou vypisovány přímo na obrazovku a/nebo ukládány na disk. Této činnosti se říká logování. Pro tuto činnost byl použit standardní balík Javy `java.util.logging`.

Tento balík umožňuje programátorovi generovat zprávy různé závažnosti (*fine*, *info*, *warning* atp.). Všechny tyto zprávy jsou směrovány do jednoho virtuálního logu. Odtud mohou být zachytávány a posílány na několik výstupních bodů a lze je přitom filtrovat podle jejich závažnosti (např. jen zprávy *warning* a závažnější).

Prvním z výstupních logovacích bodů programu **Tin2marc** je obrazovka, na kterou se vypisují pouze závažné zprávy, např. vstup/výstupní chyby nebo informace o záznamech jejichž konverze nebyla dokončena. Druhým je soubor `log/YYYY/MM/DDdHHhMMmSSs.log`, do kterého se ukládají všechny vygenerované zprávy. Zprávy obsahují následující informace: datum, čas, třídu a metodu, které zprávu vygenerovali, závažnost zprávy a vlastní text:

```
5.5.2004 21:07:38 cz.muni.fi.tin2marc.Conversion convertRecord
FINE: 123. record - "I, Robot" was successfully converted
```

5.4 Uživatelské rozhraní

Program **Tin2marc** je tvořen balíkem Javy `cz.muni.fi.tin2marc`. Jako Java aplikace je šířen ve formě Java Archivu (JAR) a je spouštěn z příkazové řádky

skriptem **tin2marc.sh** (pro operační systémy UNIXového typu) nebo dávkou **tin2marc.bat** (pro systémy Windows).

Pro běh programu musí být v systému nainstalováno Java Runtime Environment verze 1.4 a vyšší (viz 5.1.1), které je možné zdarma stáhnout z internetu na adrese <http://java.sun.com/downloads>.

Program nemá grafické uživatelské rozhraní (GUI). Chování programu je řízeno pozičními parametry a různými volbami. Když se spustí skript bez parametrů (nebo s nevyhovujícími parametry), vypíše se na obrazovku informace o všech parametrech a možných volbách:

```
Usage: tin2marc [-options] <rules> <inputFile> [<outputFile>]
```

options are:

```
-r                human readable version of output
-a                Aleph version of output
-S<TL_field>    specify the TinLib field to read SYSNO from
-s<sysno>        specify the first SYSNO (to generate SYSNO)
                  SYSNO read from record has priority over generating
-t                print info about used rules to output
-i<encoding>    encoding of the input file
-o<encoding>    encoding of the output file
                  <encoding> = [UTF8 | CP1250 | ISO-8859-2]
                  default I/O encoding is UTF8

-u<filename>    file to print the unconverted records to
                  default is "log/YYYY/MM/DDdHHhMMmSSs.exp"
```

other params:

```
<rules>          XML file with conversion rules
<inputFile>     TinLib export
<outputFile>    output MARC21 file
                  if <outputFile> not specified, using std output
```

Povinnými parametry programu jsou pochopitelně `<rules>` – cesta ke XML souboru s konverzními pravidly a `<inputFile>` – vstupní soubor s exportem z **T-Series**. Pokud není použit parametr `<outputFile>`, jsou zkonvertované záznamy tisknuty na standardní výstup programu (implicitně na konzoli).

Implicitní výstupní formát je MARC21 ve výměnném formátu dle normy ISO 2709 (viz 2.2.2). Volbami `-r` resp. `-a` je nastaven typ výstupu na řádkový MARC resp. Aleph MARC21. Program Aleph eviduje pro každý záznam tzv. systémové číslo (SYSNO). Toto číslo je možné pro každý importovaný záznam vygenerovat, ale **Tin2marc** ho umožňuje také převzít z uživatelem specifikovaného vstupního pole **T-Series**. Volbou `-S<TL_field>` se určí toto pole a volba `-s` stanoví počáteční SYSNO pro generování systémových čísel. Převzetí čísla ze zdrojového záznamu má vždy přednost.

V odstavci 4.3.8 byla zmíněna možnost sledování použitých pravidel při konverzích jednotlivých záznamů. Toto chování je vyvoláno použitím volby

-t. Informace o použitých pravidel jsou vypisovány přímo do výstupního souboru. Volbami -i resp. -o nastaví uživatel kódování češtiny vstupních resp. výstupních dat, přičemž vybírá ze tří možností – UTF8 (Unicode), CP1250 (MS Windows) nebo ISO-8859-2. Pokud není kódování specifikováno, předpokládá se UTF8.

Poslední možnou uživatelskou volbou je nastavení jména souboru, do kterého se budou zapisovat nezkonvertované záznamy **T-Series**. Pokud není toto jméno specifikováno, použije se stejná cesta jako pro soubor, do kterého se ukládají programové logy (viz odstavec 5.3.1), pouze jeho koncovka je jiná: `log/YYYY/MM/DDdHHhMMmSSs.exp`.

5.4.1 Webové rozhraní

V Kapitole 4, která popisuje formát konverzních pravidel, byl zmíněn fakt, že popisy pravidel se v průběhu vývoje programu měnily. Kontrolu správné implementace pravidel a jejich modifikací nemohl provádět autor sám, protože není knihovníkem a chyby implementace často vznikaly drobnými nedorozuměními mezi autorem a tvůrci popisu pravidel.

Pro zjednodušení testování korektnosti implementace pravidel vytvořil Mgr. Martin Šárfa z Ústavu výpočetní techniky MU webové rozhraní pro **Tin2marc**. Tento systém umožňuje mj. vyhledávat v exportovaných datech z různých knihoven konkrétní záznamy podle hodnot jejich polí, zavolá pak **Tin2marc** s příslušnými parametry a výstup programu uživateli zobrazí. Dále umožňuje vypsát hodnoty konkrétního pole všech zkonvertovaných záznamů a má i další funkce podporující kontrolu správnosti konverzí.

Další výhodou existence tohoto webového systému je jednoduchá správa verzí programu – stačí program aktualizovat na jednom místě a zainteresované subjekty mají k této verzi okamžitě přístup. Rozhraní je přístupné autorizovaným uživatelům na adrese `http://aleph.grr.ics.muni.cz`.

Součástí zadání diplomové práce je věta, že vytvořený konverzní systém by měl obsahovat i nástroje podporující vývoj a testování konverzních pravidel. Popsané webové rozhraní spolu s možnostmi samotného **Tin2marc** plní funkci tohoto nástroje. Webové rozhraní nebylo z důvodu enormní časové náročnosti vývoje konverzního systému vytvořeno autorem.

5.5 Požadavky programu a jeho výkon

Pokud je nějaký program určen pro práci s větším množstvím dat, pak je, kromě funkčnosti, důležitým ukazatelem i jeho výkon. Otázkou, která přímo souvisí s výkonem, je hardwarové vybavení, na kterém program běží a také požadavky na tento hardware. Pomocí **Tin2marc** mají být konvertovány bibliografické zá-

znamy knihoven Masarykovy univerzity, tedy i stovky tisíc záznamu pro jednu knihovnu.

Na začátku každého běhu si **Tin2marc** do paměti nahraje konverzní pravidla a v průběhu konverzí udržuje v paměti vždy maximálně dva záznamy **T-Series** a jeden záznam MARC21. Uchovává pouze pole MTIT všech zkonvertovaných záznamů (viz 5.3). Požadavky na paměť jsou tedy i pro statisíce záznamů ve vstupním souboru řádově maximálně desítky MB.

Na běžném stroji s procesorem na frekvenci okolo 2 GHz konvertuje **Tin2marc** cca 70 záznamů za vteřinu. Za hodinu tedy asi 250 000 záznamů. Vzhledem k množství a složitosti konverzních pravidel (jejich XML popis má přes 6 500 řádků) byl těmito údaji autor příjemně překvapen.

Kapitola 6

Závěr

Cílem této diplomové práce bylo navržení a implementace samostatného konverzního systému **Tin2marc** pro převod bibliografických záznamů z formátu **T-Series** do formátu MARC21 (resp. MARC21 pro **Aleph**). Program vychází ze slovního popisu konverzních pravidel vytvořených pro přechod knihoven Masarykovy univerzity mezi těmito dvěma systémy. Program musí také umožňovat nastavit jednoduchým způsobem konverzní specifika jednotlivých fakultních katalogů.

Konverzní systém byl navržen tak, že sada pravidel je zcela oddělena od samotného programu. Program pak aplikuje požadovaná pravidla na vstupní záznamy. Práce se tedy skládala z několika relativně samostatných úkolů:

- Navrhnout obecný formát pravidel pro konverze **TinLib** záznamů do MARCu a vytvořit mechanismus pro jejich zápis.
- Implementovat dodaná, slovně popsaná pravidla ve vytvořeném formátu.
- Navrhnout a implementovat samotný program **Tin2marc** jako interpret libovolné sady správně zapsaných pravidel.
- Realizovat úpravy implementovaných konverzních pravidel podle pozměňovaných požadavků, testovat správnost implementace a opravovat takto nalezené chyby.
- Vypracovat text diplomové práce.

Jednotlivé fáze realizace těchto úkolů se však časově prolínaly. Navrhovaný formát konverzních pravidel byl rozšiřován a obohacován v průběhu implementace konkrétních pravidel. Také interpret těchto pravidel – program **Tin2marc** – byl vyvíjen současně s úpravami vytvářeného formátu.

Fáze testování a úprav implementovaných pravidel je uvedena samostatně hlavně z toho důvodu, že odstranění co největšího počtu chyb a „doladění aplikace“ je nutnou podmínkou vývoje systému, který má být používán v praxi. Proto byla tato fáze velice časově náročná a pracná.

V době, kdy je psán tento text (květen 2004), byl **Tin2marc** použit pro konverzi katalogů monografií a seriálů pro knihovny Filozofické fakulty, Fakulty

informatiky a Fakulty sportovních studií MU a připravovaly se konverze katalogů dalších knihoven. Zkonvertované záznamy ve formátu MARC21 byly importovány do systému **Aleph**, na který tyto knihovny zcela přešly. Pro některé knihovny se budou kromě monografií a seriálů konvertovat i katalogy článků.

O vytvořený konverzní systém projevil zájem také Ústav výpočetní techniky Univerzity Karlovy v Praze, který je výhradním distributorem systémů **Aleph** i **T-Series** pro ČR a SR. Jejich využití **Tin2marc** je zatím ve fázi testování vhodnosti systému pro Ústavem požadované účely.

Pravidla vytvořená podle slovního popisu pro konverze monografií a seriálů jsou přiložena pouze v elektronické podobě (viz přílohy 4 a 5), protože každý ze souborů `monografie.xml` a `serialy.xml` má přibližně 6 500 řádků. Zdrojový kód aplikace **Tin2marc** má téměř 5 000 řádků, je tedy také pouze na přiloženém CD (příloha 7).

Pro používání **Tin2marc** pro konverze dalších katalogů je vždy potřeba upravit pravidla podle odlišností konkrétních zdrojových dat. Tyto úpravy znamenají často pouze např. nastavit několik textových konstant specifických pro daný knihovní katalog. Pokud nevznikne potřeba zapsání nějakého pravidla, jehož formát se zcela vymyká doposud uvažovaným tvarům pravidel, nejsou nutné zásahy do zdrojového kódu vlastního programu.

Literatura

- [1] *Návoděda klienta systému T-Series.*
- [2] ČSN ISO 5127-3a. *Dokumentace a informace.* Český normalizační úřad, Praha, 1993.
- [3] ČSN ISO 2709. *Informace a dokumentace – Formát pro výměnu informací.* Český normalizační úřad, Praha, 1998.
- [4] Petra Žabičková. *Technické aspekty výměny bibliografických záznamů.* Sdružení knihoven ČR, Brno, 2000.
- [5] Betty Furrie. *Understanding MARC – Bibliographic.* Cataloging Distribution Service, Library of Congress, Washington, D. C., 1998.
- [6] Florian Hawlitzek. *Java 2, příručka programátora.* Grada, Praha, 2002.
- [7] Joseph Schmuller. *Myslíme v jazyku UML, knihovna programátora.* Grada, Praha, 2001.
- [8] Bohdana Stoklasová. Úskalí směnitelnosti. In *Automatizace knihovnických procesů – IV*, pages 27–38. Dům techniky Ústí, Ústí nad Labem, 1993.
- [9] Bohdana Stoklasová. Vývoj katalogizačních pravidel v České republice ve 20. století aneb marné vzdorování zahraničním vlivům. *Národní knihovna. Knihovnická revue*, (2):55–62, 1999.

Rejstřík

- Aleph, 10
- bibliografický záznam, 5
- diakritika, 14
- DTD, 17
 - parametrická entita, 18
- export, 13
- import, 13
- Java, 33
 - Java Virtual Machine, 33, 34
 - virtuální stroj, 34
- katalogizační záznam, 5
- Library of Congress, 7
- MARC, 7
 - CAN/MARC, 7
 - DANMARC, 7
 - field, 8
 - indicator, 8
 - indikátor, 8
 - ISO 2709, 9
 - leader, 8
 - návěšť, 8
 - podpole, 8
 - pole, 8
 - subfield, 8
 - tag, 8
 - UKMARC, 7
 - UNIMARC, 7
 - USMARC, 7
 - XML, 10
 - řádkový, 9
 - MARC.pm, 10
 - MARC21, 7
 - match, 18
- OOP, 34
 - atribut, 34
 - dědičnost, 35
 - metoda, 34
 - objekt, 34
 - třída, 34
 - zapouzdření, 35
- pravidla
 - MARCField, 22
 - TLField, 21
 - backStoreString, 20, 23
 - condRegexp, 22
 - condition, 20, 24, 25
 - item, 20
 - regexp, 22
 - sequence, 25
 - setContextMARCField, 25
 - setContextTLField, 25
 - setTmpTLField, 25
 - sourceField, 20
 - sourceRegexp, 20, 22
 - targetField, 20, 22
 - targetString, 20, 23
- regulární výraz, 18
- T-Series, 5
- Tin2marc, 6

TinLib, 5
TinMan, 5

UML, 34

XML, 10, 16
 atribut, 16
 DOM, 16

DTD, 17
element, 16
entita, 16
tag, 16
textový uzel, 16
validní dokument, 18
XML Schema, 17
značka, 16

Přílohy

Popis souborů a adresářů, které obsahuje přiložené CD:

1. `popisy_pravidel/monografie` – soubory s popisy konverzních pravidel pro monografie,
2. `popisy_pravidel/serialy` – soubory s popisy konverzních pravidel pro seriály,
3. `popisy_pravidel/kodovniky` – soubory s kodovníky pro konverze,
4. `pravidla/monografie.xml` – soubor s konverzními pravidly pro monografie,
5. `pravidla/serialy.xml` – soubor s konverzními pravidly pro seriály,
6. `pravidla/rules.dtd` – soubor s DTD definicí formátu konverzních pravidel,
7. `src/cz/muni/fi/tin2marc` – adresář se zdrojovým textem programu **Tin2marc**,
8. `doc/index.html` – programátorská dokumentace **Tin2marc** ve formátu Java Doc (HTML),
9. `text/konverze.tex`, `text/konverze.bib`, `text/konverze.pdf` – soubory textu diplomové práce,
10. `tin2marc` – vývojový adresář programu **Tin2marc**,
11. `tin2marc-040518` – adresář s aktuální verzí distribuce systému **Tin2marc**.