Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

Výuková aplikace pro gramatiku němčiny

Plzeň, 2008 Rostislav Staněk

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 16.6.2008 Rostislav Staněk

Abstrakt

The teaching utility for german grammar is a complex application which provides interactive learning of foreign languages. The utility is devided into three parts: Data layers containing data, Administration application allowing to enter data in the database and User application enabling final data display and target user usage. The tool can be used either for exercising the grammar by students themselves or through the use of interactive board. The main focus during the developement of the tool was put on the universality of use, especialy adaptibility for another languages.

Obsah

1 Úvod

Moderní gramatika němčiny verze 2.0 by měl být univerzální výukový nástroj pro výuku cizích jazyků. Jeho těžištěm by potom měla být široká škála cvičení umožňující uživateli procvičit konkrétní látku jednak z hlediska různé úrovně obtížnosti, jednak z hlediska konkrétní látky. Základní požadavky na aplikaci z hlediska funkčnosti pro uživatele jsou následující:

- Hierarchické rozčlenění látky umožňující snadnou orientaci uživatele
- Možnost volby úrovně obtížnosti na základě Společného Evropského Referenčního Rámce (SERR)
- 8 typů cvičení (podrobnější popis viz příloha) odpovídající standardním typům používaným u mnoha zkoušek:
  - Cvičení ve formě doplňování tabulky
  - Cvičení ve formě doplňování textu do mezer
  - Cvičení ve formě doplňování textu do mezer z nabídky
  - Cvičení ve formě označení jednoho slova z několika
  - Cvičení ve formě označení jednoho řádku z několika
  - Cvičení ve formě řazení textu z částí
  - Cvičení ve formě porozumění čtenému textu
  - Cvičení ve formě porozumění mluvenému slovu

Základní požadavky na aplikaci z hlediska nakladatelství jsou:

- Ovlivnitelná hierarchie jednotlivých kapitol, podkapitol apod.
- Snadné a přehledné plnění jednotlivých typů cvičení
- Možnost snadné převeditelnosti software pro výklad jiného jazyka
- Snadná změna jazyka uživatelského rozhraní
- Použití standardních technologií jak pro aplikaci, tak pro způsob uložení dat

Plnění dat provede nakladatelství a nemusí být v rámci bakalářské práce ještě kompletní, aplikace však bude obsahovat data umožňující předvést plnou funkčnost aplikace.

Práce by měla u nakladatelství vyjít do konce roku 2008.

2 Teoretická část

V principu jsou na aplikaci kladeny požadavky ze dvou hledisek:

- Z hlediska plnění a správy dat, které provede nakladatelství
- Z hlediska interakce s uživatelem, která bude umožňovat uživateli provádět všechny výše uvedené činnosti

Jeví se tedy jako výhodné, celý problém rozdělit do dvou aplikací - *aplikace redakční* a *aplikace uživatelské*; obě potom budou pracovat nad společnými daty - tzv. *datovou vrstvou*.

2.1 Datová vrstva

Při návrhu datové vrstvy se v zásadě jeví 3 možnosti uložení dat:

- Uložení pomocí proprietárních formátů
- Uložení pomocí obyčejných textových souborů v proprietárním tvaru
- Uložení do databáze
- Uložení pomocí XML-technologie

2.1.1 Proprietární formáty

Vzhledem k požadavku užití standardních technologií uložení dat a především vzhledem k množství nástrojů pro práci s databázemi a XML-strukturami lze obě dvě první možnosti zavrhnout. Nevýhodou takové volby ovšem zůstává nutnost převodu dat z *Moderní gramatiky němčiny verze 1.0*, která používá uložení do obyčejných textových souborů za užití nestandardních značek a návěští. Výhody zbylých dvou možností ovšem převáží tuto nevýhodu.

2.1.2 Databáze

Databáze přináší oproti proprietárním formátům mnoho výhod. Hlavní potom je asi dobrá strukturovanost dat a v neposlední řadě také možnost užití velmi efektivního příkazu *SELECT* jazyka *SQL* umožňujícího rychlé třídění a vyhledávání v datech. Uložení dat do tabulek potom dobře odpovídá nejen hierarchické struktuře gramtických celků, ale i uložení jednotlivých typů cvičení. Nevýhodou potom je poměrně velká administrace nástroje a také nutnost předchozí instalace databáze na počítači uživatele.

2.1.3 XML-technologie

XML-technologie oproti tomu nabízí kompromis mezi oběma předchozími možnostmi - je strukturovaná a umožňuje tak ukládat data oproti proprietárním formátům přehledně. Dále dobře vyhovuje stromovému uspořádání gramatických celků a struktuře gramatických cvičení. Oproti datbázi potom nabízí menší administraci a použitelnost bez nutnosti instalovat speciální software. Ani velikost problému a množství nebrání použití tohoto způsobu. Další výhodou je potom možnost poměrně jednoduché validace dat. Podrobnější popis této technologie viz níže.

2.2 Redakční a uživatelská aplikace

Pro vlastní program je potom rozhodující volba programovacího jazyka. Přestože je dnes možno považovat z určitých hledisek jazyk *Borland Delphi 7.0* oproti např. *Javě*, která by přinášela také výhodu platformové nezávislosti, za zastaralý, je možné v něm vyvinout poměrně snadno aplikaci pro Windows, přičemž bohatství knihoven tohoto jazyka je pro rozsah tohoto projektu postačující. Důležitým faktorem je také předchozí verze produktu, která byla v tomto jazyce vyvinuta; přestože verze 2.0 vzniká jako zcela nový projekt a nejedná se tedy o modifikaci předchozího kódu, dojde k převzetí mnoha prvků - například části speciálně pro tuto aplikaci vyvinutých komponent umožňujících snadno realizovat mj. cvičení typu doplňování do mezer nebo typu doplňování do mezer s výběrem možností. Popis těchto komponent a využití principů *Object Pascalu* je uvedeno níže v této kapitole.

Redakční aplikace potom vzniká zcela nově bez návaznosti na minulou verzi produktu. Z důvodu případné existence společných tříd např. definujících autorská práva nebo jednotky definující jako konstanty názvy různých tagů, byl i zde zvolen stejný vývojový nástroj.

2.2.1 Objektový model Delphi

Obejktový model Delphi (nebo též *Object Pascal*) nabízí poměrně širokou škálu možností. V mnohém se tyto možnosti podobají jiným programovacím jazykům (např. *Javě* nebo *C#*), často se ale také jedná o unikátní principy. Chtěl bych tady pouze nastínit některé z vlastností, které byly použity při vývoji této aplikace.

Instituce „vlastnění“:

Delphi, stejně tak jak jako většina moderních jazyků, definují *objektové*

třídy. Základní objektovou třídou (podobně jako v Javě *main*) je *TApplication*, která sama o sobě obsahuje velké množství metod. Zvláštností Delphi, které je také v programu užito je ale instituce *vlastnictví* u „vyšších komponent“ (konkrétně se jedná o dědice třídy *TControl* - tedy především o grafické komponenty; obecně potom o všechny, které mají schopnost obsluhovat události). Při vytváření takové třídy je potom třeba uvádět v konstruktoru jako parametr vlastníka (*AOwner*), který musí být dědicem třídy *TWinControl*, nejčastěji právě pak *TApplication* nebo *TForm*. Tomuto vlastníkovi se potom uloží odkaz na vytvářenou komponentu. Povinností vlastníka je v případě jeho uvolnění uvolnit i všechny komponenty, které vlastní.

Právě tento mechanizmus přináší velkou výhodu především pro grafické aplikace s uživatelským rozhraním, kde je velké množství komponent. Dochází tak k stromovému vytváření grafických prvků, které zajišťuje správné uvolnění v případě potřeby (nejčastěji výjimky). Zásadním rozdílem oproti jiným programovacím jazykům je u Delph ale to, že „vlastník“ nemusí být nutně tím, kdo komponentu vytvořil a kde je na ní uložen odkaz - nabízí se tedy dvojí mechanizmus (a jištění) správného uvolnění komponent - buď standardně, tedy postupně přes odkazy na konkrétní, nejčastěji grafické, prvky, a nebo (v případě chyby) k hromadnému uvolnění pomocí vlastníka (např. celého panelu s ovládacími prvky) - je tak zaručeno, že nebudou ve formuláři zůstávat přebytečné prvky.

Této možnosti je využito při vytváření různých cvičení (podrobněji viz níže) - v případě chyby dojde k uvolnění celého bloku komponent a žádné přebytečné, které by případě bránily vytváření dalšího cvičení tak nezůstanou na formuláři.

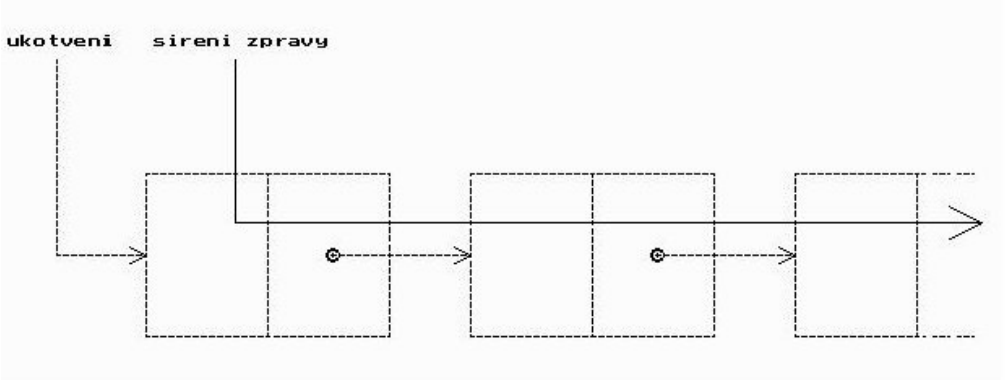
Zde bych ještě rád upozornil na metodu Free, která uvolňuje prvek v případě, že existuje, v opačném běží program bez výjimky dál.

Princip spojitých seznamů:

V programu je často použit tzv. „princip spojitých seznamů“. Pro vytvoření např. cvičení, kde by měl existovat text s mezerami, do kterých by uživatel vyplňoval koncovky, je třeba poměrně velké množství komponent (na každou mezeru jedna a na každý text mezi mezerami také minimálně jedna (leží-li obě mezery na stejném řádku)) Nabízí se potom otázka, jak tyto komponenty vytvářet, ukládat a jak je popř hromadně ovládat. Jednou možností by bylo, uložit je do pole, které by bylo obaleno třídou s příslušnými metodami, které by vždy pole procházely a prováděly případné akce. Druhou, v tomto programu zvolenou možností potom je použití spojitého seznamu, kde bude každý prvek obsahovat odkaz na následující prvek stejného typu. Všechny tyto prvky potom budou odděděny od některé z grafických

tříd (např. *TLabel*). Vlastní šíření určité zprávy potom může probíhat tak, že bude určitá metoda vyvolána u prvního prvku seznamu a ta ji automaticky vyvolá u následovníka (pokud existuje). Dojde tak k postupnému šíření signálu skrz celý řetězec (viz Obrázek 1).

Delphi také nabízejí možnost vytvářet tzv. *property*, neboli *vlastnosti*. Jedná



Obrázek 1: Princip šíření zprávy spojitým seznamem

se o typy, které fungují podobně jako proměnné s tím rozdílem, že jsou definovány 2 metody - jedna pro čtení a druhá pro zápis (*setter* a *getter*). Tyto metody bývají ovšem před cílovým uživatelem většinou skryty (jsou definovány jako *private*), ovládají nicméně přístup k vlastnostem, které se ale pro uživatele chovají jako obyčejné proměnné. Lze také použít pouze jednu z přístupových metod a docílit tak proměnných, do kterých „lze pouze zapisovat“ nebo ze kterých „lze pouze číst“.

Výhoda vlastností (*property*) je v případě spojitých seznamů především ta, že setter nebo getter lze (v případě, že je definován jako *virtual*) předefinovat. Vytvoříme-li tedy spojitý seznam např. z třídy *TLabel* a předefinejume metodu *Show*, lze standartním přiřazením do vlastnosti *Visible* docílit zobrazení všechprvků seznamu. Nabízí se tedy způsob, jak jednoduše ovládat větší „shluky“ komponent bez toho, abychom museli definovat zvláštní metody.

Příkladem takto definované třídy může být:

```
TL = class(TLabel)  //pro spojeny retezec Text u
  public
Next: TL;
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;

    procedure Show;
```

```
    procedure Hide;
end;
```

Také lze využít toho, že podobně definované komponenty jsou samy schopny komunikovat s okolím. Např. v případě cvičení, kde by měl uživatel umisťovat pomocí myši text z nabídky do některé z mezer, lze definovat mezery a pohyblivé texty jako 2 zvláštní spojité seznamy, které ovšem mají na sebe navzájem odkaz. Při pohybu potom může komponenta s textem sama rozpoznat, která mezera je nejblíž a do které ho tedy nejspíš chtěl uživatel přemístit. (Všechny takto definované seznamy v této aplikace lze najít v jednotce *UzivatelskaAplikace_U_Konstanty.pas* na přiloženém CD.)

2.2.2 XML-technologie a Delphi

XML technologie je v Delphách od verze 1.6 podporována. Za tímto účelem existuje komponenta *TXMLDocument*, která umožňuje načíst XML-strukturu. Delphi obsahují pouze DOM-parser; v případě potřeby užití SAX-parseru, je třeba použít speciálních tříd,které ovšem nejsou produktem firmy *Borland*. Komponenta *TXMLDocument* po načtení dat a vytvoření odpovídající struktury obsahuje ukazatel na třídu *DocumentElement*, která reprezentuje kořenový element XML-souboru. Dále je možné se strukturou pohybovat pomocí rozhraní *IXMLNode* a *IXMLNodeList*. První z nich reprezentuje jeden element struktury. Hodnotu obsahuje v proměnné *NodeValue*, atributy potom v poli *Attributes*, kde indexem pole je samotný název atributu. Metodou *ChildNodes* (vrací *IXMLNodeList*) lze získat potomky. Celkově tedy práce s XML dokumentem probíhá v Delphách poměrně jednoduše a s minimální administrativou, což byl také důvod k volbě tohoto programovacího jazyka. Dále je možná validace oproti *XSD* nebo *DTD*.

3 Realizační část

Vzhledem k rozdělení programu do více aplikací je třeba, aby celá aplikace tvořila kompaktní celek. Bylo proto zvoleno jednoznačné rozdělení, ve kterém jsou data (a jejich validační soubory) vždy na jednom místě v stromové struktuře dat a ostatní aplikace k nim budou pouze přistupovat. Dále bude existovat složka *Konstanty*, která bude obsahovat soubory, které budou obsahovat konstanty (popř. celé třídy) pro všechny aplikace shodné, jako např. informaci o autorských právech, konstanty s názvy XML-tagů apod.

3.1 Datová vrstva

Po zvážení všech možností byla zvolena XML-technologie s následujícím uspořádáním dat:
Data lze rozdělit na 2 základní skupiny:

- Data měněná *uživatelskou aplikací*:
  Zde bude pouze 1 soubor - *konfigurace.xml*. Jedná se o data o užvatelích, kteří tento program užívají. Tento soubor zde existuje na přání Nakladatelství a měl by být více použit v případě úspěšnosti produktu a vývoje další verze. Mohl by potom obsahovat např. různá přístupová práva (pro studenty, uživatele, apod.), statistiky úspěšností a jiné podobné informace. Nyní obsahuje pouze statickou informaci, že se jedná o obecného uživatele a má následující obsah:

```
<?xml version="1.0" encoding="windows-1250" standalone="no"?>
    <konfigurace
        xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
        xs:noNamespaceSchemaLocation="konfigurace.xsd">
    <uzivatel jmeno="Uživatel" />
</konfigurace>
```

- Data měněná *redakční aplikací*:
  Jedná se o data, která budou vytvořena pouze jednou nakladatelstvím v *redakční aplikaci* a uživatel nebude mít možnost tato data měnit. Dále bude existovat soubor *konfigurace.xml*, ve kterém bude nadefinována stromová struktura gramatických celků. V Moderní gramtice verze 1.0 se osvědčilo rozdělení do 3 úrovní:

  - *Skupina*:
    Jedná se o nejhrubší členění jednotlivých cvičení. Příkladem takové skupiny mohou být např. *Gramatické celky podle slovních druhů* nebo *Slovní zásoba*.

– *Podskupina*:
  Jedná se o členění jednotlivých skupin. Do skupiny *Gramatické celky podle slovních druhů* tak například mohou patřit *Podstatná jména*, *Přídavná jména* apod.

– *Gramatický celek*:
  Zde již bude konkrétní gramatický celek, který bude obsahovat jednotlivá cvičení. Příkladem u *Podstatných jmen* může být *Slabé skloňování*, *Zeměpisná jména* apod.

  Termín *gramatikcý celek* byl převzat z *verze 1.0* a je v důsledku rozšíření záběru procvičované látky používán i pro negramatické celky, jako např. *Čtení s porozuměním*, vždy se jím však rozumí nejnižší úroveň hierarchie gramatických celků.

Cvičení budou moci být pouze u nejnižší úrovně - u *Gramatického celku*. Část souboru *konfigurace.xml* tedy může vypadat následovně:

```
<skupina id="0" jmeno="Obecné tematické okruhy">
        <podskupina id="0" jmeno="Vyjadřování podmětu">
                <celek id="0" jmeno="Neosobní a všeobecný podměť" />
        </podskupina>
        <podskupina id="1" jmeno="Souvětí podřadná">
                <celek id="0" jmeno="Časové věty" />
                <celek id="1" jmeno="Podmínkové věty" />
                <celek id="2" jmeno="Příčinné věty" />
                <celek id="3" jmeno="Účelové věty" />
        </podskupina>
</skupina>
```

Každá hierarchická úroveň obsahuje kromě svého názvu také *id*, které je vždy pouze relativní, tedy vztažené k nadřazené úrovni. S tím je spojená především výhoda snadnější modifikovatelnosti struktury bez nutnosti přečíslovávání velkého množství prvků. Dále lze složením všech *id* nadřazených úrovní jednoznačně identifikovat nejen každý gramatický celek, ale i každou skupinu a podskupinu.

Takto složená *id* (jednotlivé části jsou oddělené znakem „_“) potom určuje *id* u souborů, ve kterých jsou uložená cvičení. Každý soubor ve tvaru *cviceni_id.xml* tedy odpovídá jednomu gramatickému celku (příkladem takového souboru může být soubor *cv_0_1_2.xml*, který by vzhledem k předchozí ukázce kódu konfiguračního souboru obsahoval cvičení ke gramatickému celku *Příčinné věty*). Tento soubor existuje ke každému gramatickému celku.

Ke každému cvičení typu *porozumění textu* dále ještě patří soubor *poslech_id.mp3*, kde *no* je *id* konkrétního cvičení. Struktura tohoto čísla je popsána níže, zde je však důležité, že dané cvičení identifikuje jednoznačně, a to i v rámci všech gramatických celků. Typ souboru *.mp3* byl zvolen z důvodu úspory místa na uživatelově disku.

3.1.1 Vnitřní struktura souborů cvičení

Vnitřní struktura cvičení, tedy souborů ve tvaru *cv_id.xml*, pokračuje v hierarchii gramatických celků. Na nejvyšší úrovni se jednotlivá cvičení člení podle úrovní obtížnosti a teprve pod každou úrovní obtížnosti jsou jednotlivá cvičení. Část souboru tohoto typu by tedy mohla vypadat následovně:

```
<uroven obtiznost="A2">
        <cviceniMezirkove id="0">
                ...
        </cviceniMezirkove>
        <cviceniMezirkove id="1">
                ...
        </cviceniMezirkove>
</uroven>
<uroven obtiznost="B2">
        <cviceniMezirkoveSVyberemMoznosti id="0">
                ...
        </cviceniMezirkoveSVyberemMoznosti>
</uroven>
```

Na této ukázce je zřejmý také systém indexace jednotlivých cvičení, která jsou indexována vždy relativně k nadřazené úrovni obtížnosti, a to bez ohledu na svůj typ. Tento systém umožňuje jednoznačně indexovat cvičení nejen v rámci jednotlivé úrovně obtížnosti, ale v rámci všech cvičení. Tento index potom vznikne složením indexu gramatického celku, úrovně obtížnosti a relativního indexu cvičení vzhledem k ní. Oddělovačem je opět podtržítko. Příkladem takového indexu cvičení může tedy být např. *0_1_2_A2_1*, kde *0_1_2* určuje jednoznačně gramatický celek, *A2* úroveň obtížnosti a číslo *1* potom relativní index cvičení vztažený v tomto případě k úrovni obtížnosti *A2*.

Důvodem, proč k rozlišení jednotlivých typů cvičení nebyl užit atribut, ale rozdílný název tagu, je možnost snadné validace rozdílné struktury obsahu cvičení právě v závislosti na jeho typu pomocí *schémového souboru*. Ten je přiložen ke každému typu .xml souboru a umožňuje tak mj. snadnou externí validaci.

Dále byla pomocí nich vygenerována podrobná dokumentace k jednotlivým typům .xml souborů; ta je přiložena v elektronické podobě. Za zmínku ještě stojí, že schéma souborů cvičení je složeno z více částí tak, že ke každému typu cvičení existuje samostatný schémový soubor, z nichž je potom složeno výsledné schéma. Celkem existuje 8 typů cvičení s následující strukturou:

Všechna cvičení

- Obsahují jako první tag (*popisCviceni*) popis cvičení, tedy text, který uživateli vysvětlí, co má se zadaným cvičením dělat.

Cvičení typu tabulka

- Je uvozeno tagem *caviceniTabulkove*.
- Každá řádka tabulky je uzavřena do tagu *polozka*. V každé položce je potom zadání (*zadani*), tedy obsah levého sloupce tabulky a správné řešení (*reseni*). Řešní potom může být tří typů - *standartni*, *dvojnasobne* a *viceSpravnych* - ty odpovídají po řadě případům kdy (podrobnější význam následujících možností viz příloha - *Typy cvičení*):
  - Jednomu zadání odpovídá pouze jedno správné řešení.
  - Jednomu zadání odpovídá pouze jedno správné řešní, které se ale skládá ze dvou částí.
  - Jednomu zadání odpovídá více správných řešení, přičemž se každé z nich skládá pouze z jedné části.

  Přestože zatím redakční aplikace neumožňuje přiřadit k různým řádkům tabulky různý typ řešení a přestože cvičení převedená z minulé verze aplikace také definují typ řešení vždy pro celé cvičení, byl tento atribut přiřazen ke každému řešení. Důvodem je případný budoucí požadavek Nakladatelství na tuto možnost, popř. požadavek na rozšíření o další typy, u kterých by byla tato možnost potřeba.

  Každé řešní se potom skládá z částí (*cast*), přičemž může být v závislosti na typu řešní tato část pouze jedna, ale může jich být i více; v rámci návrhu datové vrstvy není toto množství u řešní, kde je více správných možností, omezeno.

  Část cvičení typu tabulka může tedy vypadat následovně:

```
<caviceniTabulkove id="0">
        <popisZadani>
```

```
                Napište pomocné sloveso a příčestí
                minulé ve správném tvaru ("habe / gehabt").
        </popisZadani>
        <polozka>
                <zadani>haben (ich)</zadani>
                <reseni atributReseni="dvojnasobne">
                        <cast>habe</cast>
                        <cast>gehabt</cast>
                </reseni>
        </polozka>
        <polozka>
        <zadani>sein (Sie)</zadani>
                <reseni atributReseni="dvojnasobne">
                        <cast>sind</cast>
                        <cast>gewesen</cast>
                </reseni>
        </polozka>

        ...

</cviceniTabulkove>
```

Cvičení typu doplňování do mezer

- Je uvozeno tagem *cviceniMezirkove*.
- Každé cvičení typu *doplňování do mezer* se skládá z textu (*text*), mezer pro doplnění (*polozka*) a z odřádkování (*novaRadka*). Tyto prvky mohou být v libovolném pořadí za sebou kombinovány a vytvářejí tak celé cvičení. Jednotlivé mezery v sobě uchovávají ještě řešení v tagu *cast*, kterých i zde může být více - znamená to ovšem, že je více možných správných řešení; řešení, které by se skládalo z více částí zde nemá smysl a v případě potřeby je možné požadovat vyplnění více mezer bezprostředně za sebou.

  Část cvičení typu doplňování do mezer tedy může vypadat následovně:

```
<cviceniMezirkove id="1">
        <popisZadani>
                Doplňte sloveso haben nebo sein
                ve správném tvaru ("haben").
```

```
        </popisZadani>
        <text>1. Endlich</text>
        <polozka>
                <cast>haben</cast>
        </polozka>
        <text>wir den Weg gefunden.</text>
        <novaRadka />
        <text>2. Wir</text>
        <polozka>
                <cast>haben</cast>
        </polozka>
        <text>uns schnell nach Hause beeilt.</text>
        <novaRadka />
        <text>3. Wann</text>
        <polozka>
                <cast>seid</cast>
        </polozka>
        <text>ihr abgereist?</text>
        <novaRadka />

        ...

</cviceniMezirkove>
```

V praxi by se potom jednalo o tento text:

```
Doplňte sloveso haben nebo sein ve správném tvaru
("haben").

Zadání:
1. Endlich ___ wir den Weg gefunden.
2. Wir ___ uns schnell nach Hause beeilt.
3. Wann ___ ihr abgereist?
...

Správné(á) řešení:
1. Endlich haben wir den Weg gefunden.
2. Wir haben uns schnell nach Hause beeilt.
3. Wann seid ihr abgereist?
...
```

Cvičení typu doplňování do mezer s výběrem možností

- Je uvozeno tagem *viceniMezirkoveSVyberemMoznosti*.
- Stejně tak jako cvičení typu *doplňování do mezer* se skládá tento typ cvičení z textu (*text*), mezer pro doplnění (*polozka*) a z odřádkování (*novaRadka*). Tyto prvky mohou být v libovolném pořadí za sebou kombinovány a vytvářejí tak celé cvičení. Na rozdíl od cvičení typu *doplňování do mezer* ale definice mezer neobsahuje řešení. Důvodem jsou případy, kdy může existovat více správných řešení - v tomto případě je ale třeba brát v potaz, že se nejedná o lokální záležitost vzhledem ke konkrétnímu řešení, ale o globální proházení jednotlivých slov; těchto variací určených pořadím potom může existovat více. Řešení je tedy uloženo zvlášť (*reseni*) a tag *polozka* je tak prázdný a důležitý pouze svým výskytem.

  V rámci řešní je potom každé ze správných řešní uloženo v tagu *cast*, který dále obsahuje jednotlivé položky (*polozka*) v tom pořadí, v jakém odpovídají mezerám.

  Za řešením ještě existuje oddíl *slovaNavic*, ve kterém jsou uloženy možnosti, které budou uživateli nabídnuty „navíc“; ty slouží k tomu, aby nebylo možné cvičení řešit logicky, tedy např. výběrem z posledních dvou možností, které zbyly.

  Příkladem části cvičení tohoto typu může být:

```
<cviceniMezirkoveSVyberemMoznosti id="0">
        <popisZadani>Doplňte vhodnou předložku.</popisZadani>
        <zadani>
                <text>1. Die Ware wird</text>
                <polozka />
                <text>einen Scheck ausgegeben.</text>
                <novaRadka />
                <text>
                        2. Wir brauchen Ihre Nachricht
                        spätestens
                </text>
                <polozka />
                <text>nächsten Montag.</text>
                <novaRadka />

                ...

        </zadani>
        <reseni>
                <cast>
                        <polozka>gegen</polozka>
                        <polozka>bis</polozka>

                        ...

                </cast>
        </reseni>
        <slovaNavic>
                <slovo>entlang</slovo>
        </slovaNavic>
</cviceniMezirkoveSVyberemMoznosti>
```

Ekvivalentem by potom bylo:

```
Doplňte vhodnou předložku.

Zadání:
1. Die Ware wird ___ einen Scheck ausgegeben.
2. Wir brauchen Ihre Nachricht spätestens ___
        nächsten Montag.
...

Nabízené možnosti:
bis, ..., gegen, ... entlang

Správné(á) řešení:
1. Die Ware wird gegen einen Scheck ausgegeben.
2. Wir brauchen Ihre Nachricht spätestens bis
        nächsten Montag.
```

Cvičení typu označení slova

- Je uvozeno tagem *cviceniOznaceniSlova*.
- Po popisu zadání (*popisZadani*) následuje v tomto typu cvičení slovo (resp. sousloví), které má být označeno, tedy slovo, které má uživatel vybrat; nachází se v tagu *spravneSlovo*. Ostatní slova, mezi které bude zamícháno, jsou potom uložena v tagu *slovo*.

  Příkladem části cvičení tohoto typu může být:

```
<cviceniOznaceniSlova id="0">
        <popisZadani>Které slovo se nehodí?</popisZadani>
        <spravneSlovo>der Anorak</spravneSlovo>
        <slovo>der Bademantel</slovo>
        <slovo>die Pantoffeln</slovo>
        <slovo>das Sommerkleid</slovo>
        <slovo>der Bikini</slovo>

        ...

</cviceniOznaceniSlova>
```

Cvičení typu označení řádku

- Je uvozeno tagem *cviceniOznaceniRadku*.
- Toto cvičení je velmi podobné cvičení typu *označení slova*. Rozdíl zatím není patrný ani v *uživatelské aplikaci*, je ale možné, že v budoucích verzích budou slova zobrazována po řádcích (tzn. postupně na jeden řádek, po jeho zaplnění na další apod.), zatímco u cvičení typu *označení řádku* budou zobrazeny jednotlivé položky každá na jeden řádek.

  Správný řádek (*spravnyRadek*) i zde předchází řádkům ostatním (*radek*).

  Část cvičení tohoto typu může vypadat následovně:

```
<cviceniOznaceniRadku id="0">
        <popisZadani>
                Označte řádek, který se nehodí.
        </popisZadani>
        <spravnyRadek>
                Was ist dein Freund von Beruf?
        </spravnyRadek>
        <radek>Wann ist die nächste Führung?</radek>
        <radek>Wie viel kostet der Eintritt?</radek>
        <radek>Wie sind die Öffnungszeiten?</radek>

        ...

</cviceniOznaceniRadku>
```

Cvičení typu řazení textu

- Je uvozeno tagem *cviceniRazeniTextu*.
- Jedná se o cvičení, ve kterém je cílem složení textu z několika částí. Po popisu zadání tedy následují jednotlivé části (*text*), které, aby umožňovaly uživateli odřádkovat (a vytvářet tak např. odstavce), obsahují každou řádku uloženou zvlášť (*radek*). Texty jsou uložené ve výsledné požadovaném pořadí - jejich náhodné „zamíchání“ provede tedy již uživatelská aplikace.

  Část cvičení tohoto typu může tedy vypadat takto:

```
<cviceniRazeniTextu id="1">
        <popisZadani>
                Seřad'te následující text tak, aby dával smysl.
        </popisZadani>
        <text>
                <radek>Hallo,</radek>
                <radek>ich bin Peter und ich bin 25</radek>
        </text>
        <text>
                <radek>Jahre alt. Ich komme aus</radek>
        </text>
        <text>
                <radek>
                        Dresden, aber jetzt studiere
                        ich an der
                </radek>
        </text>

        ...

</cviceniRazeniTextu>
```

Zde sice každá část textu obsahuje jeden nebo dva řádky, obecně jich ale může být neomezené množství a je tedy možné vytvořit i cvičení, u nichž bude cílem z jednotlivých delších odstavců sestavit výsledný text.

Cvičení typu čtení s porozuměním

- Je uvozeno tagem *cviceniCteniSPorozumenim*.
- Jedná se o cvičení, u nichž bude uložen text, který bude uživateli zobrazen. Uložení tohoto textu bude provedeno podobně, jako uložení jedné z částí textu u předchozího typu - v obalovacím tagu (*text*) budou dále uloženy jednotlivé řádky (*radek*), které budou i zde umožňovat ovlivnit odřádkovávání a vytvářet tak např. odstavce.

  Tvrzení potom budou uložena každé v tagu *teze*, který bude obsahovat atribut (*reseni*) určující, zda je tvrzení pravdivé (*pravda*), nepravdivé (*nepravda*), nebo zda toto nelze na základě textu rozhodnout - tedy zda tato teze nebyla v textu zmíněna (*nebyloZmineno*).

  Cvičení tohoto typu potom může vypadat následovně:

```
<cviceniCteniSPorozumenim id="1">
        <popisZadani>
                Přečtěte si následující text a na jeho
                základě rozhodněte, zda jsou
                předložená tvrzení pravdivá,
                nepravdivá, nebo nebyla v textu
                zmíněna.
        </popisZadani>
        <text>

                ...

                <radek>
                        Das Königliche Arbeitszimmer
                        ist im romanischen Stil gehalten,
                        entsprechend der Wartburg, die
                        als Vorbild zum Bau des Schlosses
                        Neuschwanstein diente. Sämtliche
                        Holzarbeiten ...
                </radek>

                ...

        </text>

                ...

        <teze reseni="pravda">
                Wartburg diente als Vorbild zum
                Neuschwansteinbau.
        </teze>

                ...

</cviceniRazeniTextu>
```

Datová vrstva (a validační soubor) neurčuje, v jakém pořadí budou jednotlivé teze uloženy, v praxi ovšem bude redakční aplikace vyžadovat nejprve zadání tezí, které jsou pravdivé, potom tezí, které jsou nepravdivé a nakonec těch, které nebyly v textu zmíněny; v tomto pořadí také budou uloženy do .xml-souboru. Uživatelská aplikace bude tedy muset

tyto teze před zobrazením nejprve „zamíchat“.

Cvičení typu poslech s porozuměním

- Je uvozeno tagem *cviceniPoslechSPorozumenim*.
- Toto cvičení je, alespoň co se způsobu uložení týká, prakticky stejné jako cvičení typu *čtení s porozuměním*. I cvičení tohoto typu tedy bude obsahovat text (*text*) členěný na řádky (*radek*) a teze (*teze*), v jejichž atributu (*reseni*) bude uloženo, zda jsou pravdivé (*pravda*), nepravdivé (*nepravda*), nebo zda nebyly v textu zmíněny (*nebyloZmineno*). I zde sice nezáleží na pořadí uložených tezí, ale vzhledem k naprogramování redakční aplikace je třeba, aby tyto teze uživatelská aplikace „zamíchala“.

  Rozdíl bude pouze v tom, že k cvičení tohoto typu bude existovat nahrávka, která bude uložena v speciálním souboru; ten bude jednoznačně identifikován *id* tohoto cvičení (podrobnější popis indexace a celkové struktury datové vrstvy viz výše). Uživatel si nahrávku poslechne a na jejím základě rozhodne o zadaných tezích. Až poté, co bude cvičení opraveno, bude moci shlédnout psanou podobu textu.

3.2 Redakční aplikace

Hlavní otázkou při tvorbě *redakční aplikace* je, jak široký záběr činností by měla obsahovat a také, zda se bude jednat ve výsledku o aplikaci jedinou nebo aplikací více. V principu lze rozdělit redakční činnosti na 2 skupiny:

- Na činnosti zasahující do hierarchické struktury gramatických celků.
- Na činnosti zasahující do vlastních dat.

V případě první skupiny se jedná tedy pouze o modifikaci souboru *konfigurace.xml*. Po poradě s redakcí nakladatelství a ze zkušeností z minulé verze, kde byly jednotlivé gramatické celky uloženy jako konstanty v jedné z tříd aplikace a aplikace se tedy pro jejich změnu musela vždy znovu překládat, byla zvolena kompromisní možnost. Hierarchie gramatických celků je sice flexibilně uložena v souboru *konfigurace.xml* a *redakční* i *uživatelská aplikace* ji bude vždy při spuštění znovu načítat, měnit ji ale z těchto aplikací možné nebude. Důvodem je malá pravděpodobnost této změny (alespoň v případě, že se bude jednat o stále stejný jazyk). V případě potřeby pro němčinu (velmi málo pravděpodobné), nebo v případě převádění aplikace pro výuku jiného jazyka, lze tuto poměrně jednoduchou strukturu v souboru ručně změnit a

aplikace ji budou respektovat. Dalším důvodem pro toto rozhodnutí je fakt, že rozvržení gramatických celků je třeba (i z hlediska nakladatelství) udělat před vlastním plněním dat. Při převodu pro výuku jiného cizího jazyka potom dojde nejprve k smazání předchozích dat (která by neměla pro jiný jazyk význam), změně tohoto souboru a teprve potom znovuplnění daty novými.

V případě druhé supiny, tedy zásahu do vlastních dat, se potom jedná o činnosti, které redakční aplikace musí vykonávat minimálně z následujících tří hledisek:

- Z hlediska přidávání jednotlivých cvičení
- Z hlediska jejich editace
- Z hlediska jejich mazání

Dále je třeba, aby redakční aplikace byla na jedné straně uživatelsky přátelská a ne zbytečně složitá (tzn. aby nebylo těžké ji ovládat i člověkem bez rozsáhlejších znalostí počítače) a aby byla na druhé straně „robustním“ a „inteligentním“ nástrojem udržujícím v každém případě konzistentnost dat, a aby také zabránila jejich případné ztrátě ať již v důsledku neodborného zacházení s touto aplikací nebo v důsledku pádu systému.

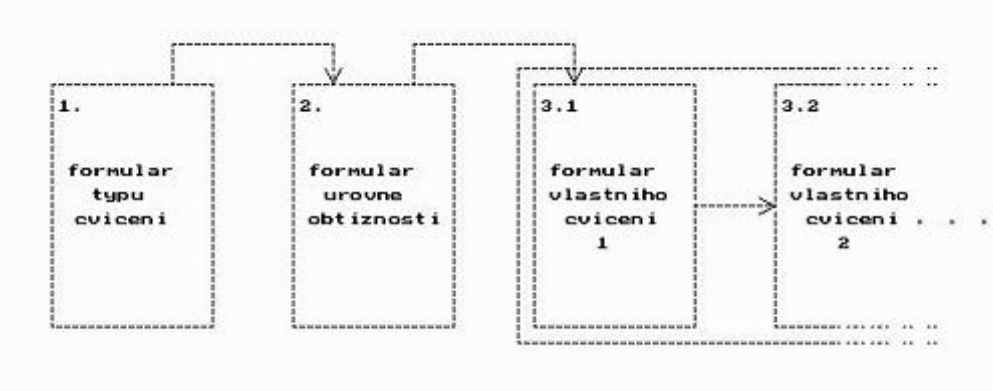
Z uživatelského hlediska je tedy jedním z hlavních cílů jednoduchost ovládání a snadná orientace v aplikaci. Z tohoto důvodu bylo také zvoleno znázornění struktury gramatických celků pomocí stromu (komponenta *TTreeView*). Zde jsou potom dvě možnosti jak to udělat - buď se bude celá struktura od *skupin* až po jednotlivá *cvičení* nacházet v jednom stromě, nebo lze využít dělení datové vrstvy na *soubor konfigurační* a *soubory jednotlivých cvičení*. Druhý přístup s sebou přináší velké množství výhod. Nespornou výhodou této možnosti je také možnost načítat jednotlivé .xml-struktury cvičení (*cv_id.xml*) až v případě potřeby.

Logickým členěním je tedy vznik jedné stromové struktury pro hierarchii gramatických celků a druhé pro konkrétní gramatický celek. První z nich by se potom načetla pouze jednou při spuštění aplikace (která do ní nemůže zasahovat) a druhá by se načítala až po kliknutí na některý z nabídnutých celků, popř. zůstala prázdná v případě, že by uživatel kliknul na *skupinu* nebo *podskupinu*. S tím by také souvisela případná aktivita nebo neaktivita tlačítka pro *přídávání cvičení*. V případě, že by uživatel zvolil konkrétní celek, a i konkrétní cvičení, stala by se aktivními i tlačítka *editace* a *mazání*.

Redakční aplikace z hlediska konkrétního cvičení

U konkrétních cvičení, jak již bylo popsáno v předchozím odstavci, dojde nejprve k volbě gramatického celku a teprve poté bude možno cvičení

přidávat. Zde je třeba zdůraznit, že uživatel nebude mít možnost přidávat úroveň obtížnosti, neboť tu bude volit až při vytváření cvičení - v případě, že již byla požadovaná úroveň vytvořena, dojde k přidání tohoto cvičení do ní, v opačném případě dojde k jejímu vytvoření. Tento přístup je především jednodušší pro uživatelské ovládání, neboť uživatel nebude muset kontrolovat, která úroveň již existuje a do které tedy cvičení může přidat. Z důvodu uživatelské vstřícnosti byl dále zvolen princip „průvodce“ (*wizard*), který uživatele provede vytvořením jednotlivého cvičení. Schéma tohoto „průvodce“ je znázorněno na Obrázku 2.
Ze schématu je zřejmé, že při vytváření musí uživatel projít třemi základními



Obrázek 2: Schéma vytváření nového cvičení

částmi:

- Volbou typu cvičení
- Volbou úrovně obtížnosti
- Vytvořením vlastního cvičení

Třetí část potom může sestávat ještě z několika dalších. V principu se nabízejí 3 možnosti, jak tento problém v *Borland Delphi 7.0* realizovat:

- Vytvořením pouze jednoho formuláře, na kterém by byly všechny potřebné komponenty (*radioboxy*, *nadpisy* apod). Aktuální stav, tedy to, v které části se právě proces vytváření cvičení nachází, by byl potom určen viditelností (vlastnost *Visible*) a dalšími vlastnostmi (*properties*) jednotlivých komponent.

- Vytvořením sice pouze jednoho formuláře, který by ale obsahoval více záložek (komponenty *TPageControl* a *TTabSheet*). V rámci těch by se potom dalo přepínat buď kliknutím pouze na záložku v horní části obrazovky nebo klikáním na tlačítko *Další*, které by uživatele jednotlivými záložkami „provedlo“.

- Vytvoření formuláře (resp. dialogu) pro každou tuto část. Přepínání mezi nimi by potom bylo řízeno klikáním na tlačítko *Další* popř. tlačítko *Zpět*.

Nakonec byla zvolena poslední možnost.

První možnost s sebou nese riziko vzniku „chaosu“ v nejednoznačném použití komponent a dálě ztěžuje možnou implementaci editace, neboť by vždy při přepnutí (což u této možnosti znamená pouze změnu vlastností jednotlivých komponent) docházelo ke změně jejich účelu. Vždy by se tak musela data nově do komponent z XML-struktury načítat.

Druhá možnost potom sice částečně řeší problémy předchozího řešení, ty ale přetrvávají u třetí části - tedy u plnění vlastního cvičení, neboť tato záložka (*TPageConstrol*) by potom musela obsahovat komponenty pro všechny typy cvičení nebo některé z nich užívat pro více typů zároveň.

Jako nejrozumnější se tak jeví třetí možnost, která navíc s sebou přináší možnost s výhodou užít *dědičnosti*. Jednotlivé formuláře obsahují mnoho společných prvků - všechny musí mít tlačítko *Další* (resp. *Dokončit*), tlačítko *Zpět* a tlačítko *Cancel*. Další podobnost potom lze vysledovat i u plnění vlastního cvičení, protože každé musí obsahovat alespoň v jednom ze svých kroků mj. *Popis zadání*.

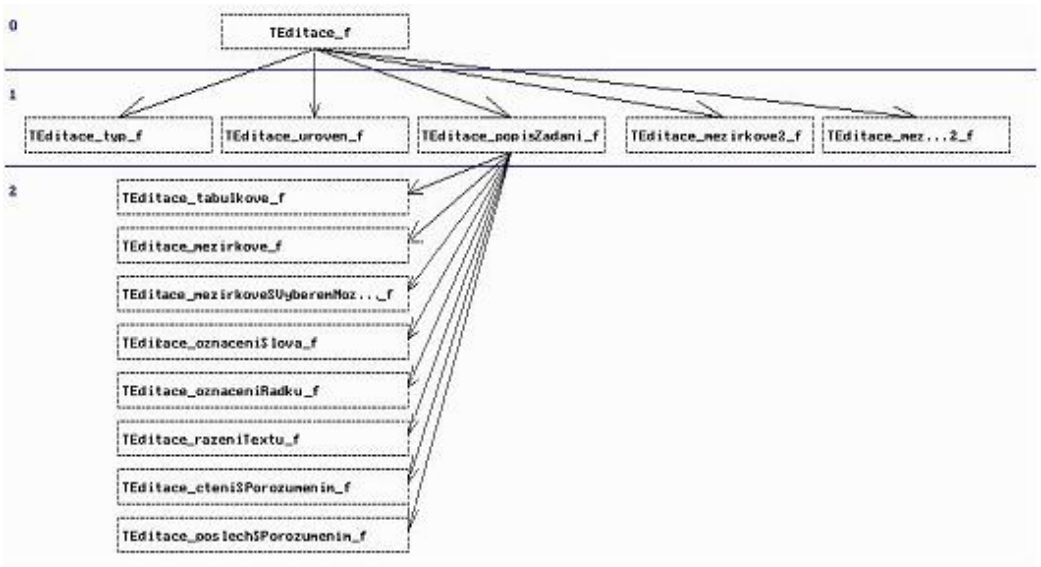
Bude tedy existovat jeden formulář (*TEditace_f*) jako základní, od něhož budou všechny ostatní odděděny. Sám bude obsahovat pouze výše zmiňovaná 3 tlačítka a 2 virtuální metody (*inicializuj* a *vycistiFormular*), které je třeba v potomcích překrýt; první z nich se vždy spust automaticky při vytváření formuláře. Potomek nejprve zavolá zděděnou metodu (příkaz *inherited inicializuj;*), která provede inicializaci zděděných prvků a poté provede sám inicializaci prvků v tomto potomkovi přidaných - tento postup zaručuje fungující inicializaci i při vícenásobném dědění. Obdobně bude fungovat druhá metoda s tím rozdílem, že se bude jednat o „vyčištění“ všech komponent, tedy uvedení do výchozího stavu pro plnění. Tato metoda bude např. volána vždy před vytvářením nového cvičení pro každý z formulářů, které při tomto procesu budou figurovat. Tato metoda je také automaticky volána již při vytváření, neboť její volání je zároveň posledním příkazem metody *inicializuj* definované v rodičovské třídě *TEditace_f*. Dále tento „prarodič“ obsahuje obsluhu tlačítka *Cancel*, která je pro všechny formuláře společná a zajišťuje,

že při zobrazení (metoda *Show*) tohoto (nebo zděděného) formuláře dojde k jeho vykreslení do středu obrazovky, což má za následek, že bude průvodce působit kompaktně a uživatel nepozná, že je složen z více formulářů.

Od této třídy jsou přímo zděděny formuláře pro volbu typu cvičení (*TEditace_typ_f*) a pro volbu úrovně obtížnosti (*TEditace_Uroven_f*).

Dalším přímo zděděným formulářem, který ale opět bude sloužit pouze jako rodič, je *TEditace_popisZadani_f*. Ten oproti *TEditace_f* přidává nadpis a okno pro vyplnění popisu zadání. Od tohoto formuláře jsou potom odděděny ve třetí části první formuláře všech typů cvičení (*TEditace_tabulkove_f*, *TEditace_mezirkove_f*, *TEditace_mezirkoveSVyberemMoznosti_f*, *TEditace_oznaceniSlova_f*, *TEditace_oznaceniRadku_f*, *TEditace_razeniTextu_f*, *TEditace_cteniSPorozumenim_f* a *TEditace_poslechSPorozumenim_f*)

Schéma znázorňující tyto závislosti je vidět na Obrázku 3.



Obrázek 3: Princip dědičnosti formulářů pro přidávání cvičení

Vlastní realizace plnění tedy probíhá tak, že je zobrazen první formulář (výběr typu cvičení) a kliknutím na talčítko *Další* se zobrazí formulář zajišťující výběr úrovně obtížnosti. Zde, po dalším kliknutí na *Další*, dojde v závislosti na zvoleném typu k zobrazení dalšího správného formuláře a takto probíhá řada až do konce. Podstatou tohoto systému ovšem je, že vždy existuje možnost se vracet, neboť je pro každý úkon speciální formulář; tedy není třeba data hned ukládat, ale je možné je uložit až najednou při kliknutí na

tlačítko *Dokončit*. Také rozhodování o zobrazeném typu formuláře po výběru úrovně obtížnosti je flexibilní - k rozhodnutí dojde v závislosti na aktuálně vybraném *radioButtonu* ve fromuláři s výběrem typu cvičení - to také umožňuje uživateli vrátit se zpět, změnit typ cvičení a ten začít vyplňovat; v případě, že by se chtěl opět vrátit k původnímu schématu, nepřijde díky oddělenosti formulářů k jednotlivým typům cvičení o data, která již naplnil - ta zůstanou v tomto formuláři zachována až do vyvolání metody *vycistiFormular*, která je volána vždy před vytvářením nového cvičení.

Při ukládání dat do XML-struktury bylo potom dbáno především na to, aby jednak data zůstala v každém případě konzistentní a to i v případě, že by došlo kdykoliv k pádu systému, ale také na to, aby uživatel v takovém případě přišel pouze o minimum vykonané práce. Celý systém tedy funguje na principu zasílání zpráv hlavní třídě (*RedakcniAplikace_U_f*), která jediná zpravuje celou XML-strukturu. Při vytváření nového cvičení dojde tedy po kliknutí na tlačítko *Dokončit* k volání požadavku hlavní třídě na vytvoření nového cvičení. Parametrem této metody je potom úroveň obtížnosti a typ cvičení v podobě názvu jeho tagu. (Gramatický celek není třeba předávat, neboť kliknout na tlačítko *Přidat cvičení* lze pouze tehdy, byl-li již nějaký gramatický celek v stromové struktuře vybrán; ve chvíli, kdy uživatel začne vytvářet nové cvičení (nebo staré editovat), dojde automaticky k zablokování hlavního formuláře a jeho parametry tak zůstanou nezměněné.) Návratovou hodnotou je potom ukazatel na nově vytvořený objekt cvičení (typu *IXMLNode*), který již je správně zařazen do XML-struktury, ke změně na disku ale ještě nedochází. Zde je třeba také podotknout, že tento princip umožňuje určovat z hlavní třídy, jestli dojde k vytvoření cvičení nového nebo k přepsání starého novými daty v případě editace. Teprve v případě, že metoda dotvoří cvičení v pořádku (tzn. nedojde k chybě a data uživatelem zadaná jsou validní), dojde k vyvolání uložení na disk. V každém případě však je volána událost hlavní třídy obnovující stromové struktury, která proběhne v případě úspěšně vytvořeného cvičení již nad novou uloženou XML-strukturou, v opačném případě se tak obnoví původní validní struktura bez „zbytků“ nedotvořeného cvičení.

Je zde také vidět, že k ukládání dochází automaticky, vždy jednorázově ve chvíli, kdy je zřejmé, že ukládaná struktura je validní - poškození dat ze strany redakční aplikace v případě pádu aplikace nebo systému je tímto eliminováno na minimum.

Další výhodou tohoto systému práce s daty je editace - v případě kliknutí na tlačítko *Editace* se tak může stát v podstatě totéž, jako při vytváření nového cvičení, čehož je v aplikaci využito. Jediným rozdílem je, že místo vyčištění formulářů dojde k jejich naplnění načtenými daty - uživatel potom data může modifikovat a při kliknutí na *Dokončit* dojde k vytvoření zcela

nového cvičení (metoda svázaná s tímto tlačítkem není pro tento případ nijak modifikována). Opět dojde k vyvolání metody žádající o přidělení nového cvičení v podobě *IXMLNode*. Tato metoda (*getCviceniNode*) ovšem vrátí ukazatel na kořenový uzel starého cvičení, který je pouze „vyčištěný“ od původních dat. Dojde tak k vytvoření nového cvičení na místě původního. Tento přístup opět přináší výhodu udržení konzistence dat, neboť k zásahu (jednorázovému v podobě přepsání ukazatele) dojde pouze v případě, že bylo po modifikaci vytvořeno cvičení, které je validní. Mechanizmus ochrany popsaný u vytváření tak zůstává platný i zde.

Mazání je potom záležitostí použití metody *removeChild*.

3.2.1 Obecné poznámky k redakční aplikaci

Vzhledem k možnému požadavku Nakladatelství vytvořit uživatelské rozhraní v jiném jazyce, načítá aplikace všechny zobrazované texty ze souboru *RedakcniAplikace_U_Language_CZ.txt*, který je při překladu importován do jednotky *RedakcniAplikace_U_Konstanty*. Ke změně jazyka uživatelského rozhraní tedy stačí pouze přeložit řetězce na pravých stranách definic konstant a aplikaci znovu přeložit. Podobným způsobem jsou do aplikace importovány definice konstant, které budou společné i pro *uživatelskou aplikaci*. Jedná se o soubory *Xml.txt*, *Copyright.txt* a *Konstanty.txt* v adresáři *Konstanty*. První z nich obsahuje definici všech názvů tagů a jejich atributů, druhý potom informaci o autorských právech. Ve třetím jsou uloženy jiné společné konstanty.

3.3 Uživatelská aplikace

Přestože se *uživatelská aplikace* týká podobného problému jako *aplikace redakční*, jsou na ní kladeny jiné požadavky. Zde na prvním místě stojí uživatelská přátelskost a rychlost. To, co bylo možné u *redakční aplikace* oželet v rychlosti, díky čemuž byla důsledně zajišťována konzistentnost dat, u *uživatelské aplikaci* možné není. Zde je ovšem situace jednodušší především v té zkutečnosti, že se data z datové vrstvy pouze čtou. S tímto modelem lze počítat i do budoucna pro vyšší verze, neboť i v nich by mělo docházet maximálně k zápisu do jediného souboru *konfigurace.xml*, který ovšem z hlediska aplikace nepatří k nejdůležitějším a u něhož by hrozila maximálně ztráta uživatelských statistik. Této skutečnosti se v *uživatelské aplikaci* využívá tím, že již neprobíhá žádná (nebo pouze minimální) kontrola validity dat. Nehrozí zde ztráta informací, mohlo by se pouze stát (a to jedině v případě, že by uživatel nainstalované soubory poškodil nebo smazal), že bude třeba aplikaci přeinstalovat.

Když tedy vyjdeme z předpokladu, že jsou vstupní data validní, naskýtá se opět i zde základní otázka, jak hierarchii gramatických celků členit. Pominemeli možnost více různých formulářů a vyjdeme-li z předpokladu, že bude vše umístěno na „jediné ploše“, jak vzhledem k plánovanému použití na interaktivních tabulích požadovalo Nakladatelství, máme i zde, podobně jako u *rudakční aplikace*, několik možností:

- Celou strukturu včetně jednotlivých úrovní a koncových cvičení zobrazovat v jediné stromové. struktuře.
- Snížit hloubku stromové struktury vyjmutím některých voleb ven (např. volby úrovně pomocí *RadioButtonů*).
- Mít podobně jako u redakční aplikace stromy 2 - jeden pro volbu gramatického celku a druhý pro výběr úrovně obtížnosti a vlastního cvičení.

První možnost s sebou přináší především úsporu místa na ploše, neboť, jak bylo uvedeno výše, je třeba, aby se na jediný formulář (bez rollerů) vešly všechny komponenty. Nevýhodou této možnosti je ovšem jednak velká doba spouštění aplikace (nutno načíst všechny xml-soubory), jednak menší uživatelská přehlednost, ale především hrozba pádu aplikace v případě, že by data svým rozsahem převýšila paměťové možnosti počítače. Tato nevýhoda by se sice nemusela projevit hned (nyní mají data (která ale ještě nejsou kompletní) 10MB), ale především po dokončení plnění daty, popř. ve vyšších verzích, do nichž se počítá s postupným přidáváním dat.

Druhá možnost neřeší v minulém odstavci nastíněné problémy, navíc s sebou přináší větší nároky na prostor na ploše a její řešení přehlednosti je přinejmenším spekulativní.

Jako nejlepší se pak jeví třetí z možností. Ta sice není, co se prostoru na ploše týká, nejúspornější, tuto nevýhodu lze ale vyvážit užitím svislého rolleru - dojde tak k faktickému „rozpůlení“ plochy, kterou by jinak zabíral jeden velký strom. Tato možnost řeší tako problémy s velikostí dat. A protože gramatických celků je v němčině definováno cca 60, a pro jiné jazyky by byl tento počet nejspíš podobný, lze tak pracovat s 60 * objemnějšími daty. Nevýhodou je potom pouze možná prodleva při načítání vždy nového xml-souboru pro jiný gramatický celek. Předpokládané užití je ovšem takové, že uživatel bude dělat cvičení nejprve k jednomu celku (zde by k prodlevám nedocházelo), a teprve potom k jinému. Také objem dat (do budoucna max cca 200 MB - po vydělení 60-ti 3,3 MB na soubor) není tak velký, aby tato zpoždění byla neúnosná.

Dojde zde tedy k stejnému (vizuálnímu i programovému) řešení jako u *redakční aplikace*. Při startu aplikace se načte jednorázově pouze konfigurační

soubor s gramatickými celky (*konfigurace.xml* v adresáři *Data*), a teprve při kliknutí na konkrétní gramatický celek dojde k načtení příslušného souboru a zobrazení struktury úrovní a všech cvičení.

Při kliknutí na konkrétní cvičení je potom třeba toto cvičení vykreslit. Celý mechanizmus vykreslování funguje následovně:

Vždy, když se klikne na aktuální cvičení v příslušném stromě, dojde k uložení reference na uzel tohoto cvičení v DOM-struktuře do proměnné *aktualniCviceniXMLNode*. Potom na základě *if ... else if ... else if...* – konstrukcí dojde k vybrání příslušného typu cvičení. Protože se vše nachází v jediném formuláři, bylo by podle běžné sémantiky Delph logické vše umístit do jediné třídy, neboť přístup z jiných tříd na tento formulář by byl někdy obtížný (sémantika Delph to nepředpokládá). V této aplikaci bylo zvloleno kompromisní řešení importů. Pro každý typ cvičení existuje zvláštní soubor (např. *CviceniTabulkove.txt*); v tomto souboru jsou uloženy vždy 2 metody - jedna pro vykreslení (v tomto případě *tvorCviceniTabulkove*) a jedna pro kontrolu (v tomto případě *zkontrolujCviceniTabulkove*). Tyto soubory jsou pomocí direktivy překladače *$I* naimportovány do hlavní třídy. Jedná se tedy sice formálně o jedinou třídu, z hlediska programátora jsou ovšem tyto metody odděleny. Dále je zde využita možnost vhnízďování procedur a funkcí, která je speciofická pro jazyky Pascalského typu - každá z těchto metod si může definovat podpůrné procedury a funkce, které patří pouze jí a s nimiž tak tvoří jedinou „entitu“ - všechny nástroje potřebné pro vytvoření (resp. zkontrolování) určitého typu cvičení jsou tak zapouzdřeny v jediné entitě.

Metody pro tvoření vrací typ *boolean*. V případě neúspěchu (např. z důvodu nenainstalovaných ovladačů zvukové karty) pak dojde k „hromadnému uvolnění“ popsanému v *Teoretické části*.

Kontrola cvičení pak probíhá po kliknutí na tlačítko v dolní části obrazovky. I zde dojde k *if... else if ...* – sekvenci, která vyvolá příslušnou metodu pro kontrolu. Tyto metody se samy starají o „změnu vzhledu“ svých komponent spojenou s kontrolou.

Dále zde existuje metoda *rusCviceni*, která vyvolá uvolnění všech prvků všech typů cviční. Tato metoda je volána vždy, když uživatel klikne do některého stromu (přesněji řečeno, pouze v případě, že neklikne na cvičení, ve kterém se právě nachází). Po tomto uvolnění se znovu zobrazí úvodní obrazovka.

Součástí uživatelské aplikace je také jednotka *UzivatelskaAplikace_U_Konstanty*, která obsahuje kromě konstant (jako např. odstupu textu od kraje panelu, apod.) také všechny komponenty (zřetězebné seznamy) potřebné pro jednotlivá cvičení. Tyto komponenty jsou podrobněji popsány v *Teoretické části*.

Poslední součástí uživatelské aplikace je „okénko“ s „typem uživatele“,které obsahuje jedinou položku „Uživatel“ - ta je načtena při startu z konfiguračního

souboru *konfigurace.xml*. Význam této součásti aplikace je pouze pro budoucí verze a je zde na výslovné přání Nakladatelství.

Za zmínku ještě stojí, že stejně tak, jako u *redakční aplikace*, je i zde zcela oddělena jazyková podstata aplikace - všechny zobrazené texty jsou uloženy v konstantách ve speciláním souboru *UzivatelskaAplikace_U_Language_CZ.txt*. Převod uživatelského rozhraní do jiného jazyka (např. do celoněmecké verze) je pak pouze otázkou změny tchto konstant a znovupřeložení programu.

3.4 Data a jejich testování

K oběma aplikacím existují reprezentativní data, která obsahují všechny typy cviční. S těmito daty byly aplikace testovány na různých počítačích a různých verzích systému (*Windows XP* a *Windows ME*). Aplikace dále byly testovány i na počítačích s minimálním systémem. Datová vrstva je nyní naplněna cca z 2/3. Naplnění zbytku provede Nakladatelství do konce roku 2008.

4 Závěr

V rámci bakalářské práce byl proveden kompletní návrh datové vrstvy aplikace *Moderní gramatika němčiny verze 2.0*, na jehož kvalitu byl kladen největší důraz z důvodu možného použití dat i v jiných aplikacích (např. generování testů pro učitele přes internet, apod.). Toto možné použití i jinými programtory je také důvodem k nejpodrobnějšímu popisu právě této vrstvy a také k vytvoření pokud možno co nejkvalitnějších validačních *XSD* souborů, které kontrolují mimo počty i obsahy jednotlivých atributů a snaží se nechávat při zachování pro použití důležité volnosti strikntí pravidla, která by zamezila nesprávnému a nekoncepčnímu užití. Důsledně byly v *XSD* také užívány dokumentační tagy pro všechny elementy, atributy i datové typy, což umožnilo vygenerovat podrobnmou dokumentaci (viz přiložené CD). V rámci vytváření datové vrstvy potom došlo ke kompletnímu převedení všech cvičení z *verze 1.0*.

Dále byla kompletně vytvořena *Redakční aplikace*. Zde se jedná především o robustní nástroj umožňující snadné a uživatelsky přátelské plnění datové vrstvy. Přestože byla důsledná kontrola validity vstupních dat a častý zápis přímo na disk někdy na úkor rychlosti aplikace, věřím, že se jednalo o správnou volbu. Nemělo by tedy dojít k poškození datové vrstvy (ať již její konzistence nebo obsahu). V rámci redakční aplikace nebyly implementovány 2 typy cvičení, které již byly z knihy plně využity ve verzi *1.0* a došlo tak k jejich převodu. Do budoucna, v případě dalšího jiného použití datové vrstvy, ovšem bude nutné tuto funkčnost dodat.

Uživatelská aplikace, která je pouze jednou z možných použití předchozích částí potom podle mého názoru zcela vyhovuje zadanému problému. Nejedná se o aplikaci příliš rozsáhlou (z hlediska instalace a nároků na systém), ale především o kompaktní aplikaci umožňující přehlednou formou procvičovat německou gramatiku jednak na PC studentů, ale také přehledně ve škole na interaktivních tabulích, pro které je mj. určena. Tomu je také přizpůsoben vzhled, kde je vše „na jedné ploše“ a vše je tedy přímo dostupné bez složitého procházení mnoha formuláři. Další možné rozšíření se nabízí především v oblasti správy uživatelů, u kterých by mohly být vedeny statistiky úspěšnosti, popř. jiné informace (přístupová práva k mazání těchto statistik, poznámky k jednotlivým cvičením, apod.).

Přes všechny nedostatky, které určitě tento produkt má, a které se nejspíše projeví až jeho používáním v praxi, věřím, že se jedná o přívětivý a systematický nástroj, který nabízí další možnosti ve vzdělávání a který, jak pevně věřím, pomůže zacelit další z mezer na trhu se vzděláním a dopomůže tím i mnoha uživatelům k lepšímu pochopení německé gramatiky.

A Použitá literatura

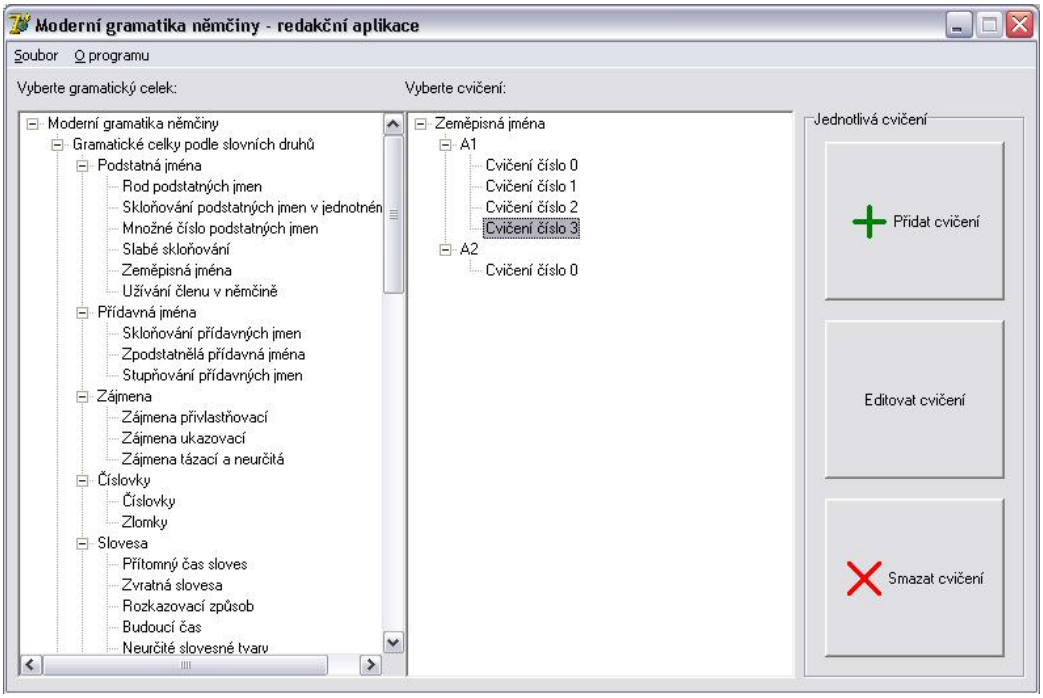
- Borland Software Corporation: *Object Pascal Language Guide*. Borland Software Corporation 2001
- Ray Lischner: *Delphi v kostce*. Praha, Computer Press 2000
- Košek Jiří: *www.kosek.cz* (xml-technologie)
- Berglová Eva a kol.: *Moderní gramatika němčiny*. Plzeň, Nakladatelství Fraus 2003

B Uživatelský manuál

B.1 Redakční aplikace

Redakční aplikaci lze spustit kliknutím na ikonu s jejím názvem v *redakční verzi* programu (*uživatelská verze* tuto ikonu neobsahuje. Po spuštění se zobrazí okno, které je zobrazeno na Obrázku 4

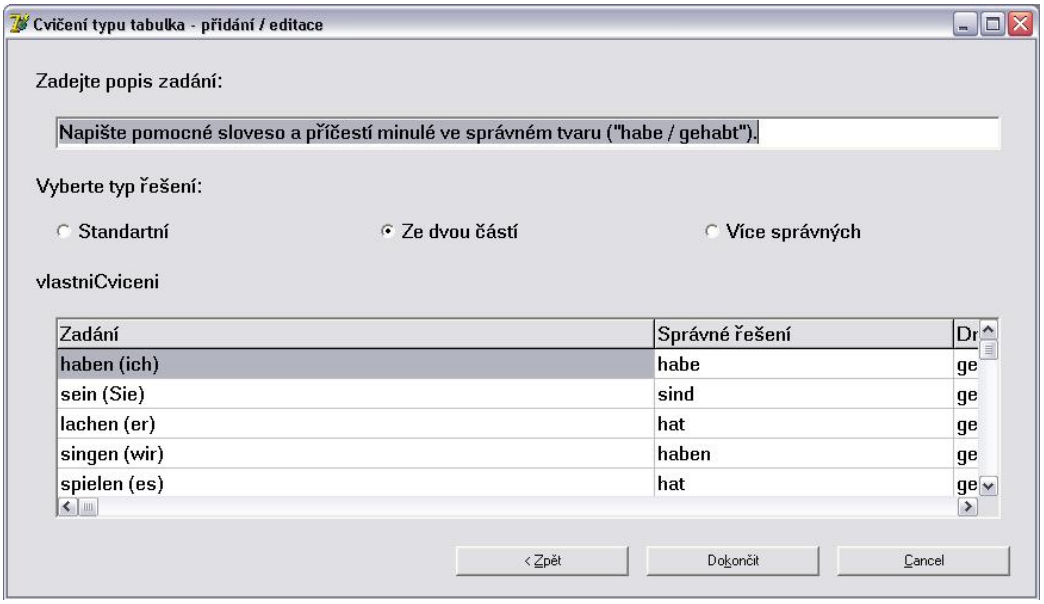
Zde lze „rozklikáním“ vybrat konkrétní gramatický celek a poté i konkrétní



Obrázek 4: Redakční aplikace - základní formulář

cviční (existuje-li již nějaké pro tento celek). Nyní lze cvičení přidávat kliknutím na tlačítko *Přidat cvičení*. Po výběru jeho typu (seznam typů cvičení viz příloha *Typy cvičení*) a jeho úrovně obtížnosti (podle Společného Evropského Referenčního Rámce SERR) lze naplnit jeho vlastní obsah. Význam jednotlivých položek je zřejmý ze specifikace jednotlivých typů v již zmíněné příloze. Tento formulář (již po částečném naplnění) je vidět na Obrázku 5.

Cvičení je možné také mazat a editovat. Nejprve je třeba toto cvičení v hlavním formuláři vybrat a následně kliknout na tlačítko *Editovat cvičení* nebo *Mazat cvičení*. Po kliknutí na *Mazat cvičení* bude cvičení nenávratně smazáno.



Obrázek 5: Redakční aplikace - formulář pro vytváření cvičení typu „tabulka“

Provedené změny (ať již vytvoření nového cvičení nebo smazání nebo editace starého) není třeba ukládat. Tato operace probíhá automaticky.

Pro změnu struktury gramatických celků je třeba kontaktovat autora (`stanekr@students.zcu.cz`).

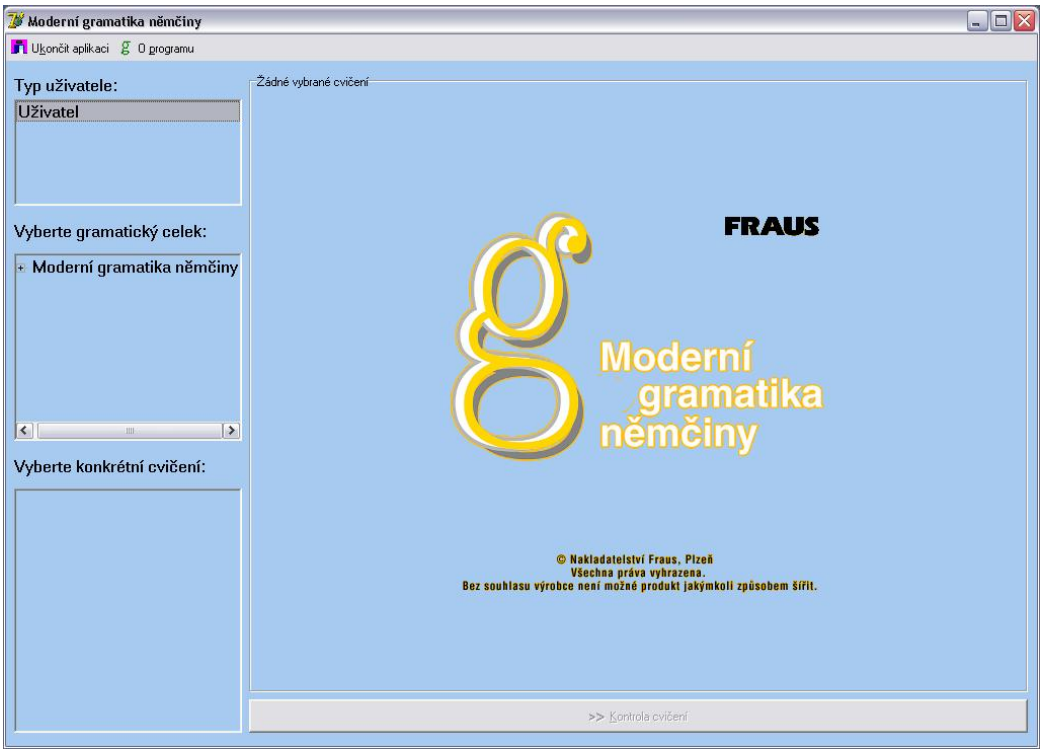
B.2 Uživatelská aplikace

Uživatelskou aplikaci lze spustit kliknutím na ikonu s jejím názvem v *uživatelské verzi* programu. Po spuštění se zobrazí okno, které lze vidět na Obrázku 6.

V levé části obrazovky lze potom po rozbalení položky *Moderní gramatika němčiny* vybrat postupně konkrétní gramatický celek. Po kliknutí na něj lze níže vybrat konkrétní úroveň obtížnosti a konkrétní cvičení. Nyní již stačí cvičení podle instrukcí v horní části obrazovky vyřešit. Až budete mít cvičení vyřešeno, stačí kliknout na tlačítko *Kontrola cvičení*. V případě, že není vidět správné řešní (např. u cvičení,kde se dpolňovalo řešení do mezer v textu), je třeba podržet myš nad touto mezírkou a počkat, až se řešení zobrazí v bublinkové nápovědě.

Aplikaci lze ukončit kliknutím na tlačítko *Ukončit aplikaci*.

Další ukázky z uživatelské aplikace *Moderní Gramatika němčiny* lze spatřit na Obrázcích 7 - 10 v závěru přílohy (*Další obrázky aplikací*).



Obrázek 6: Uživatelská aplikace - základní formulář

C Typy cvičení

- Cvičení typu *tabulka*:
  - Jedná se o cvičení založená na principu tabulky, kde v levém sloupci je zadání a do pravého má uživatel psát řešení. Příkladem takového cvičení může být doplňování členů k podstatným jménům, tvoření množného čísla apod. ...

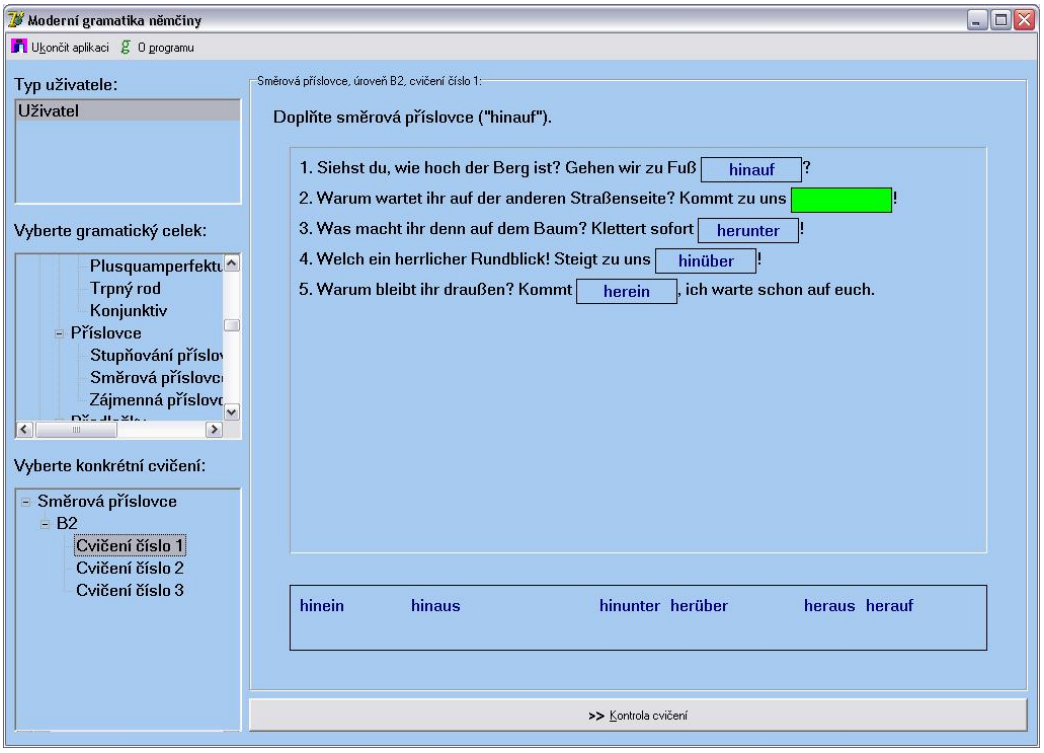
    V principu jsou potom možné 3 typy řešení:

    * Standardní řešení - jednomu zadání jednoznačně odpovídá jedno řešení. Příkladem může být určování členů
    * Dvojnásobné řešení - jednomu zadání odpovídá sice jedno řešení, které se ale skládá ze dvou částí, u nichž nutně nezáleží na pořadí. Příkladem může být určování příčestí minulého a pomocného slovesa pro tvorbu perfekta.
    * Více správných řešení - jednomu zadání odpovídá více správných řešení. Každé se však skládá pouze z jedné části (je unitární). Příkladem může být doplňování vhodné předložky (často je možno doplnit do věty různé předložky a tím změnit její smysl).
- Cvičení typu *doplňování do mezer*:
  - Jedná se především o cvičení založená na principu souvislého textu, ve kterém jsou vynechána některá slova, popř. jejich části, které má uživatel doplnit. Příkladem může být text, ve kterém je třeba doplnit správný člen ve správném tvaru. Tento typ cvičení lze ovšem s výhodou užít i k tvoření jednotlivých vět, ve kterých je třeba něco doplnit. Jednotlivé věty potom mohou být číslovány a každá může být na samostatném řádku. I zde je možné, že k jedné mezeře existuje více správných řešení.
- Cvičení typu *doplňování do mezer s výběrem možností* (popř. řazení věty):
  - Jedná se o podobná cvičení jako cvičení typu *doplňování do mezer* s tím rozdílem, že uživatel požadovaná slova (popř. jejich části) vybírá z několika možností. Dalším použitím tohoto cvičení jsou potom cvičení typu *řazení věty*, ve kterých má uživatel za úkol seřadit z uvedených slov větu nebo její část. Zde se také může vyskytnout případ, že je možné nabízená slova přiřadit mezerám v různém pořadí, popř. různě sestavit větu. Dále je třeba, aby existovala v nabídce slova navíc, která nepatří do žádné z mezer - cílem je, aby uživatel slova přiřazoval skutečně na základě gra-

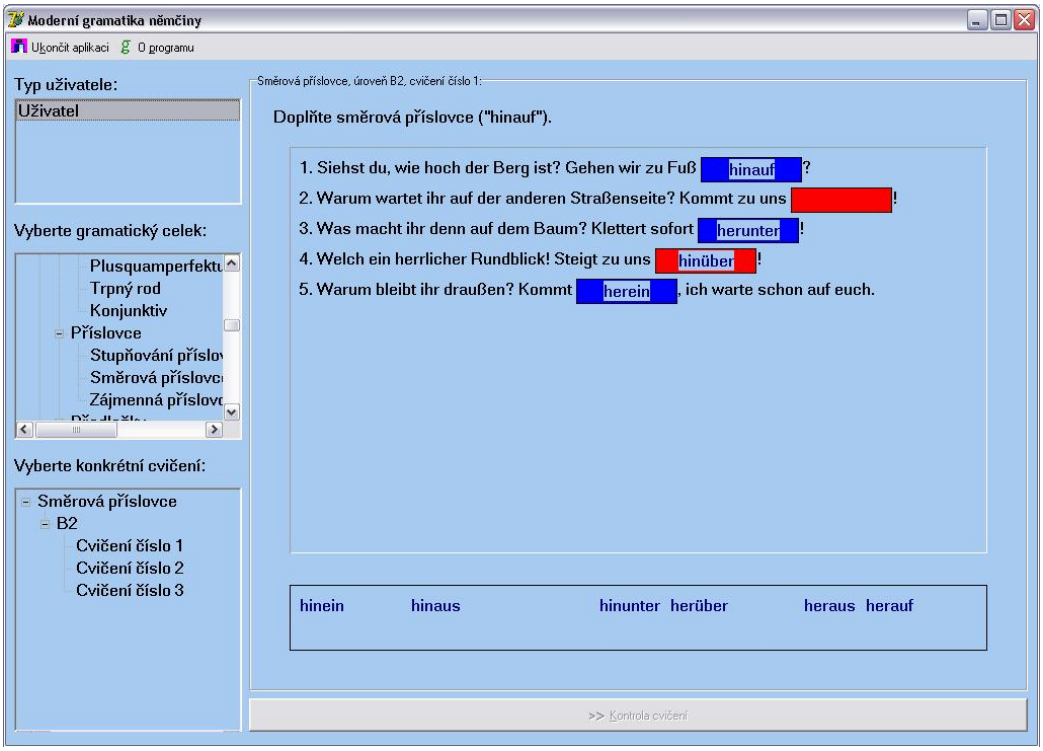
matické znalosti a ne na základě logické úvahy - nepoužíval např. „vylučovací metodu“.

- Cvičení typu *označení slova*:
  - Jedná se o cvičení, kde bude mít uživatel vždy u každé skupiny vybrat podle v popisu zadání určeného kritéria jedno slovo (resp. sousloví), které označí. Tato cvičení mohou být využita např. ke škrtnutí nehodícího se slova.
- Cvičení typu *označení řádku*:
  - Jedná se o cvičení, ve kterých bude uživatel moci označit řádek podle zadaného kritéria. Příkladem užití může být výběr gramaticky správné věty z několika, popř. výběr věty nejvhodnější. Oproti předchozímu typu cvičení předpokládá tento typ delší sousloví (popř. celé věty) - tomu také bude podřízeno výsledné zobrazení v uživatelské aplikaci
- Cvičení typu *řazení textu*:
  - Jedná se o cvičení, ve kterých bude uživatel z částí textu vytvářet jejich řazením celek. Jednotlivé části mohou být buď jednotlivé odstavce, nebo také naprosto nepravidelně rozvržené části. V rámci jednotlivých částí bude respektováno případné odřádkovávání.
- Cvičení typu *čtení s porozuměním*:
  - Jedná se o cvičení, ve kterých si uživatel přečte text a na základě jeho obsahu bude rozhodovat o zadaných tezích, zda jsou pravdivé, nepravdivé, nebo nebyly v textu zmíněny.
- Cvičení typu *poslech s porozuměním*:
  - Jedná se o cvičení, ve kterých uživatel uslyší text a na základě jeho poslechu bude rozhodovat o zadaných tezích, zda jsou pravdivé, nepravdivé, nebo nebyly v textu zmíněny. Po vyhodnocení bude mít uživatel možnost si prohlédnout poslouchaný text.

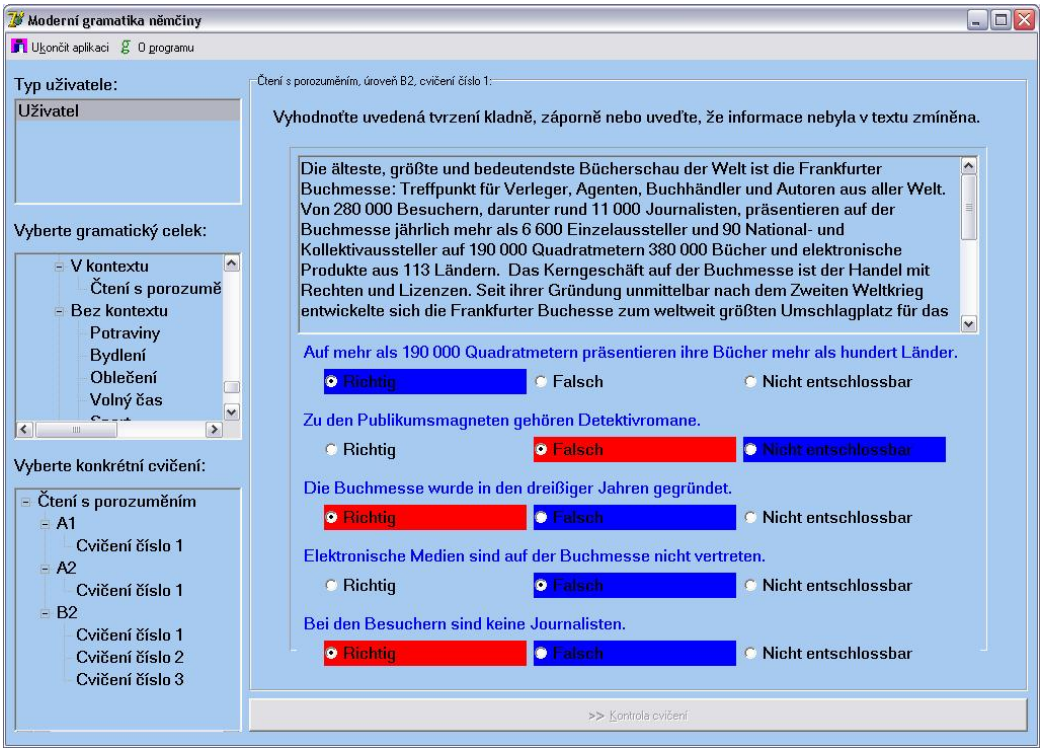
D Další obrázky aplikací



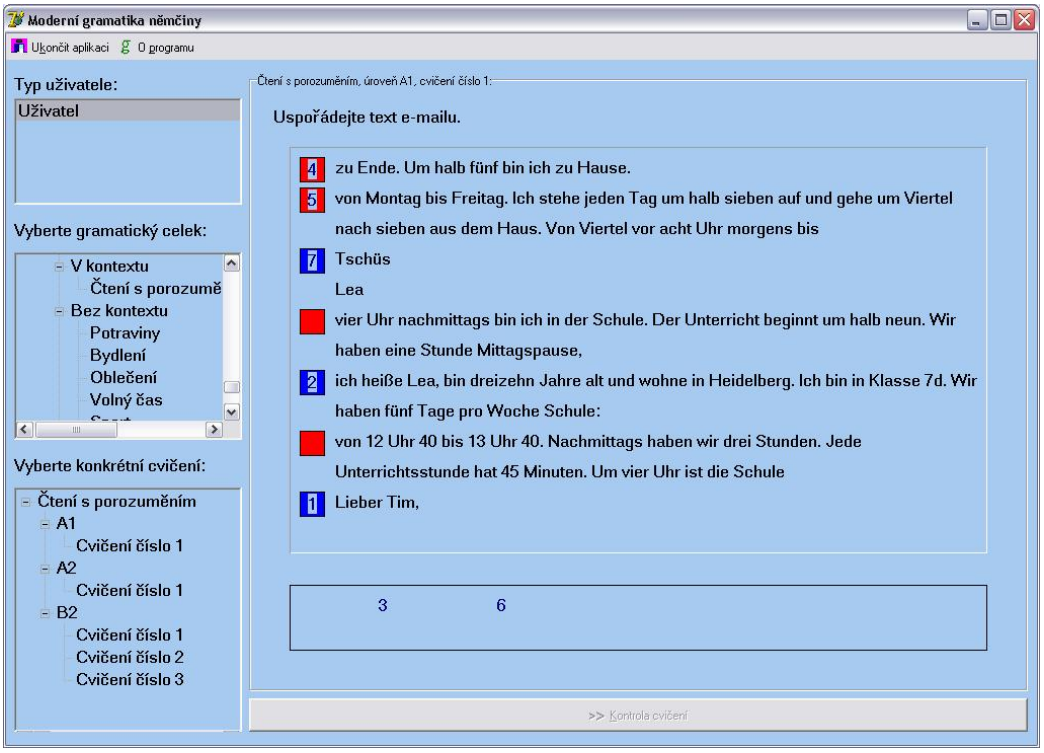
Obrázek 7: Uživatelská aplikace - cvičení typu *Mezírkové s výběrem možností*



Obrázek 8: Uživatelská aplikace - cvičení typu *Mezírkové s výběrem možností* po kontrole



Obrázek 9: Uživatelská aplikace - cvičení typu *Čtení s porozuměním*



Obrázek 10: Uživatelská aplikace - cvičení typu *Řazení textu*