VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

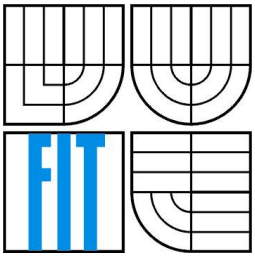
JAZYKOVÉ VERZE WEBU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE BC. ONDŘEJ LAGA
AUTHOR

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

JAZYKOVÉ VERZE WEBU
LANGUAGE VERSION OF WEB

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. ONDŘEJ LAGA
AUTHOR

VEDOUCÍ PRÁCE — ING. JAROMÍR MARUŠINEC, PH.D., MBA
SUPERVISOR

BRNO 2008

Abstrakt

Tato práce se zabývá problematikou vícejazyčných webových aplikací. Dokument popisuje některá obecná řešení při návrhu takovýchto aplikací, především se však zaměřuje na Informační systém VUT a jeho rozšíření o správu překladů. Text obsahuje strukturní charakteristiku tohoto systému a nástroje používané při jeho vývoji, zejména však definuje požadavky vývojářů a překladatelů na systém, popisuje a hodnotí nové řešení jazykových verzí a v závěru se zamýšlí nad možnostmi případných rozšíření.

Klíčová slova

Jazykové verze, internacionalizace, syntaktická analýza, PHP, systém pro správu verzí, IS VUT.

Abstract

This thesis concerns a dilemma of multi-lingual web applications. The document describes some general solutions while suggesting such applications, however first of all it is aimed for information system VUT and its enlargement for translation administration. The text contains structural description of this system and instruments used during its development, but especially it defines system requirements of programming engineers and translators, describes and evaluates new language versions solution and there are possibilities of contingent extensions considered at the conclusion of my thesis.

Keywords

Language version, internationalization, syntactic analysis, PHP, version control system, IS VUT.

Citace

Laga Ondřej: Jazykové verze webu. Brno, 2008, diplomová práce, FIT VUT v Brně.

Jazykové verze webu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jaromíra Marušince, Ph.D, MBA. Další informace včetně odborných konzultací mi poskytl Ing. Michal Jurosz. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Ondřej Laga
5. 5. 2008

Poděkování

V této sekci bych chtěl poděkovat vedoucímu diplomové práce Ing. Jaromíru Marušincovi, Ph.D., MBA za umožnění zpracování tohoto projektu, jakož i panu Ing. Michalu Juroszovi za odborné vedení a zájem, kterým usměrňoval moji práci.

Obsah

1 Úvod

V moderní době globalizace, kdy dochází k dynamickému vzrůstu obchodu a sbližování kultur, se internet stává jedním z největších nástrojů přiblížení, nezbytným prostředníkem pro sdělování informací. Komunikace však vyžaduje společný, srozumitelný jazyk. Na internet jsou tak kladeny nové požadavky, které reflektují tento stav multikulturní společnosti.

Tento jev je možné pozorovat také z ekonomického hlediska. Velké firmy investují nemalé peníze ve snaze přiblížit se svým klientům. V oblasti webových stránek se jedná například o přehledný a čistý design, snadnou navigaci a přehlednost. Nejde však pouze o pohodlí. Jestliže smyslem internetových stránek (tím spíše informačních systémů, kde je informace na prvním místě) je přenos informace, sdělení, pak nesrozumitelností obsahu ztrácí svůj smysl. Jazyk je tedy velmi důležitým prvkem a jazyková podpora je v oblasti IS většinou nezbytnou součástí, především u větších projektů je potřeba s ní počítat již při návrhu.

V následujícím textu se zaměřujeme na jazykové verze IS VUT (Informační systém Vysokého učení technického v Brně). Současný způsob vytváření vícejazyčných textů se stává nevyhovujícím a vyžaduje inovaci. Z pohledu vývojářů a překladatelů vzniká potřeba celý proces překladů zefektivnit, zjednodušit a přizpůsobit novým požadavkům.

Následující kapitola se zamýšlí nad obecnou problematikou tématu a definuje obecné požadavky na vícejazyčný systém. Kapitola 3 popisuje nejdůležitější nástroje využívané při tvorbě IS VUT, je zde charakterizován použitý programovací jazyk, databáze a systém pro správu verzí. V další kapitole jsou vyjmenovány základní rysy celého informačního systému, detailní popis databáze, struktura serverů atd. Následující kapitoly se již zabývají samotnými překlady. Postupně popisují současný stav systému v oblasti jazykových verzí, návrh nového řešení, implementaci a testování. V závěru této práce pak můžeme najít zhodnocení výsledků a možnosti budoucího rozšíření.

2 Obecná problematika vícejazyčných webových aplikací

Tato kapitola definuje základní požadavky na vícejazyčné webové aplikace a ukazuje vybraná řešení nejčastějších problémů.

2.1 Základní požadavky

2.1.1 Oddělení obsahu stránek od zobrazení

Jedná se o moderní trend hojně používaný například ve spojení XML (Extensible Markup Language) a XSL (Extensible Stylesheet Language). XML zajišťuje univerzální formát pro přenos informací, XSL definuje způsob zobrazení. Toto rozdělení je možné aplikovat i na dvojici XHTML (Extensible HyperText Markup Language) a CSS (Cascading Style Sheets). Pokud bychom sledovali vývoj webových stránek v průběhu desetiletí, uvidíme posun od původního HTML (HyperText Markup Language), kde značky zároveň definují obsah i zobrazení, k XHTML a CSS, kde jsou tyto role rozděleny. Tento nový způsob je výhodný hned z několika důvodů:

- informace stejného druhu se soustředí na jednom místě
- jeden styl zobrazení je možné přiřadit k několika informacím (textům)
- jednomu textu lze přiřadit více stylů zobrazení

Ovšem oddělení obsahu a zobrazení je možné aplikovat také na samotný text. Nejedná se už o barvu a velikost textu, nýbrž o oddělení na jiné úrovni, oddělení významu (smyslu textu) od slovního vyjádření. Význam textu pak bývá reprezentován nějakým unikátním klíčem vyjadřujícím obsahové sdělení, které spolu s konkrétními větami v daném jazyce vytváří požadované rozdělení rolí. V praxi je to většinou realizováno pomocí relační tabulky, která může mít např. takovou podobu:

klic	**text_en**	**text_cz**
uvitani	Welcome	Vítejte

Tabulka č. 1 – Příklad relační tabulky s překlady

Uložení překladů do databáze poskytuje vysoký komfort pro správu textů, vyhledávání, sledování změn či nastavení přístupových práv. Alternativním řešením může být uložení textů do souboru. Tento způsob sice nenabízí tak vysoký komfort jako databáze, avšak vyhledávání může být značně rychlejší, zvláště při použití hashování. Uchovávání textů v souboru se často používá u menších systémů či u vícejazyčných desktopových aplikací.

U extrémně zatížených systémů je nutné hledat jiná optimální řešení, vyhovující právě specifickým požadavkům daného systému. Některé požadavky dokonce mohou být protichůdné (např. podmínka správného fungování určitých částí webu s jazykovými verzemi i při výpadku databáze). Tímto způsobem vznikají jakási hybridní řešení. Jedním z nich může být právě kombinace předchozích dvou způsobů. Data jsou uložena v relačních tabulkách. Po jejich změně se automaticky vygenerují soubory obsahující asociativní pole s překlady. Toto řešení tedy spojuje výhody obou předchozích možností. Vytvoření takového automatického systému ovšem vyžaduje značné úsilí.

V rámci IS VUT se v současnosti používá uložení překladů do souboru. To však vede k narůstající velikosti souboru, nepřehlednosti a obtížné správě. Toto řešení je pro stávající rozsah systému již nevyhovující. Na druhou stranu jde o rychlé a poměrně transparentní řešení.

2.1.2 Správnost dat

U aplikací obecně, především však u informačních systémů, požadujeme, aby uchovávaná data byla konzistentní a zároveň nedocházelo ke zbytečné duplicitě informací. Tyto dvě vlastnosti spolu souvisí, neboť u duplicitních dat narůstá riziko nekonzistence. Protože jsou data uchovávána především v databázi, je možné těmto problémům předcházet správným návrhem databáze a použitím kontrolních mechanismů (např. triggerů), které sledují správnost dat (atomicitu operací a transakční zpracování zajišťuje databáze). Kromě toho je vhodné data testovat již na úrovni aplikační, především při vstupu z formulářů, a zabránit tak některým útokům či chybám způsobeným uživateli. Některé z těchto mechanismů jsou v IS VUT již implementovány.

2.1.3 Správnost zobrazení

Vícejazyčné aplikace se zpočátku potýkaly s problémem správného zobrazení znaků. Rozmanitost národních abeced vedla k vytváření několika různých kódování pro každou abecedu, následkem čehož se národní znaky často zobrazovaly chybně. Například český jazyk používal nejméně 5 různě kódovaných tabulek (kódování bratří Kamenických, PC Latin 2, Windows 1250, ISO Latin 2 atd.). Docházelo tak k problémům při spolupráci aplikací a při přenosech dat mezi programy.

S postupující globalizací tedy narůstá požadavek na určité sjednocení, vytvoření normy. Tímto způsobem vzniká univerzální kódování Unicode[1], které bez ohledu na program či platformu jednoznačně identifikuje každý znak v rámci celého světa. Použití této normy však s sebou nese určité nevýhody. Máme sice zajištěno správné zobrazení jakéhokoliv symbolu, avšak větší rozsah znaků

znamená větší množství dat, takže místo v paměti, které texty zabírají, narůstá mnohem rychleji. Některé starší programy navíc kódování Unicode nepodporují. Nicméně tato norma se používá stále více a především v oblasti vícejazyčných webových aplikací je její použití nezbytné.

Unicode používá pro uchování každého znaku 16 bitů. Ačkoliv se jedná o univerzální kódování, které pokrývá 38 885 znaků národních abeced[2] a postačuje více než dostatečně pro běžnou komunikaci, přináší také několik nevýhod. Mezi nimi bychom jmenovali především mnohem větší paměťovou náročnost ukládaného textu oproti předchozím kódováním (tento aspekt je způsoben univerzálností kódování) a z toho plynoucí časově náročnější zpracování textu.

Pro mnohé jazyky platí, že národní znaky tvoří pouze menšinu znaků a pro velká kvanta textu postačuje pouze ASCII (American Standard Code for Information Interchange). Pro takové texty je zcela zbytečné použít dva bajty na uložení každého písmene. Postupem času tedy vzniklo několik různých jiných zápisů Unicode, které sice pracují s omezenou abecedou národních znaků, avšak ve většině případů je toto řešení naprosto postačující. Jedním z nejpoužívanějších je UTF-8 (UCS Transformation Format) s osmibitovým kódováním znaků, který je také použit pro uchovávání textů v databázi informačního systému VUT.

2.1.4 Výkonnost a rozšiřitelnost systému

U rozsáhlých informačních systémů se často setkáváme s problémem poklesu výkonnosti, čímž je myšlena například nižší rychlost zpracování požadavku. Důvodem zpomalení většinou bývá narůstající složitost, nepřehlednost či nesprávný počáteční návrh.

V rámci plánování je potřeba také zvážit rozložení zátěže jednotlivých serverů a používání vyrovnávacích (cache) pamětí. Klasicky se snažíme minimalizovat množství požadavků, čímž se urychlí obsluha. Zároveň chceme rozložit zátěž rovnoměrně mezi jednotlivé servery tak, aby nedocházelo k přetížení a výpadkům. Tyto problémy je možné řešit nejenom na úrovni hardwarové a konfigurační, ale také v rámci samotné aplikace.

V oblasti rozdělení výkonu jsou webové servery IS VUT nakonfigurovány způsobem, který umožňuje rovnoměrné rozdělení požadavků mezi několik serverů (podrobnosti viz kapitola Informační systém VUT).

Další často podceňovanou a přitom velmi důležitou otázkou je možnost pozdějšího rozšíření systému. Jestliže od počátku stavíme aplikaci jako uzavřenou s tím, že ji nelze dále rozvíjet, výrazně se zkracuje její životní cyklus. Její možnosti jsou omezeny. A protože se s postupem času objevují stále nové požadavky na systém, ať už se jedná o zvýšení rychlosti či rozvoj nových možností a aplikace není schopná tento vývoj respektovat, její relativní výkon klesá a uživatel o ni ztrácí zájem. Z tohoto důvodu je potřeba investovat do budoucího rozvoje systému. Rozšiřitelnost se tak stává jedním ze základních požadavků.

2.1.5 Jednoduchost a transparentnost použití

Dalším důležitým předpokladem úspěšného IS je zachování jednoduchosti a s tím souvisejícího transparentního použití. To znamená nejen fakt, že se i méně zkušení uživatelé musí v systému vyznat (důraz na přehlednost stránek a celkovou logickou strukturu z pohledu uživatele), ale také v rámci vývoje musí být systém přehledný a transparentní. Zde se jedná především o čitelnost zdrojových kódů, použití knihoven či celkový styl programování. Pro psaní zdrojových kódů bývají někdy vydávána určitá doporučení, která sjednocují formu programování v rámci systému (podrobnosti viz [20]).

V rámci čitelnosti lze také porovnávat jednotlivé metodiky programování, například objektově-orientované aplikace jsou obecně mnohem přehlednější než klasické funkcionální. Zde hraje významnou roli abstrakce, která dovoluje používat objekty, aniž bychom znali jejich složitou implementaci, čímž dochází k značnému zjednodušení.

Kromě výše jmenovaných technik je také důležité zachovat vnitřní logické uspořádání aplikace (např. uspořádání souborů a knihoven podle účelu a použití). Za podstatné považujeme také vytvoření kvalitní dokumentace, která nejen vysvětluje smysl jednotlivých částí systému, ale také uchovává souvislosti mezi jednotlivými prvky a je významným pomocníkem při pozdějším rozšiřování.

3 Nástroje používané při vývoji systému

3.1 Jazyk PHP

PHP (rekurzivní zkratka PHP: HyperText Preprocessor, původně Personal Home Page) je dynamický programovací jazyk pro tvorbu webových aplikací. Programování stránek pomocí PHP[3] je velmi oblíbené pro svou jednoduchost a rozšířenost u poskytovatelů, provázanost s webovým serverem Apache a snadnou komunikaci s nejznámějšími databázemi (MySQL, PostgreSQL a řada dalších). Samotný jazyk je multiplatformní, což přináší značnou výhodu nezávislosti na hardwaru a operačním systému. Velice často se objevuje ve spojení s webovým serverem Apache, jedním z nejpoužívanějších serverů vůbec. Je to otevřený projekt s rozsáhlou podporou a dokumentací. Na internetu je možné najít mnoho příkladů či hotových aplikací a možnost využití PHP poskytuje většina webhostingových společností. Mezi nejčastější aplikace psané v PHP patří například internetové obchody, diskusní fóra, redakční systémy, poštovní klienti atd. Mezi nesporné výhody tohoto jazyka patří také jednoduchost a rychlost. U projektů, kde je kladen velký důraz na bezpečnost nebo přehlednost, se často používají alternativní řešení (např. Java Server Pages).

Jazyk PHP také poskytuje komfort objektově-orientovaného programování. Tato moderní technika představuje při tvorbě komplexních webových aplikací značný přínos. Ztrácí se tak obtíže s používáním globálních a lokálních proměnných a můžeme využít všech výhod OOP (zapouzdření, dědičnost, abstrakce, polymorfismus). Kromě toho dochází k zefektivnění a zpřehlednění kódu, což je u rozsáhlých systémů velice pozitivní. Navzdory těmto výhodám se často přistupuje ke klasickému stylu programování, kde dochází k problematickému míchání HTML kódu s aplikační logikou. Důvodem může být jednoduchost použití či vyšší rychlost. U velkých projektů však narůstá nepřehlednost celého systému, což naopak zpomaluje vývoj a vede k častějším chybám. Z dlouhodobého hlediska je tedy lepší využít objektově-orientovaný přístup a návrhový vzor Model-view-controller (MVC – softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídící logiku do tří nezávislých komponent tak, že modifikace některé z nich má minimální vliv na ostatní [4]).

Aktuální verze PHP 5, která se používá také v IS VUT obsahuje kromě jiného nové jádro Zend II, snadnější práci s XML a strukturovanou správu chyb pomocí třídy Exception. Zvláště poslední jmenované je silným prostředkem při hledání a ošetřování chyb. V oblasti OOP zaznamenala tato verze opravdu velký posun ke klasickým objektovým jazykům. Nabízí využití mechanizmů jako autoloading, statické metody, abstraktní třídy, návrhové vzory, typová kontrola atd. (podrobnosti viz [5]). Je tedy vhodné využít těchto prostředků k budování tak rozsáhlého systému jako je IS VUT.

3.2 Databáze

3.2.1 Obecný popis

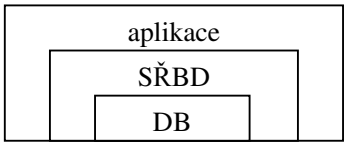
Databáze je důležitou součástí informačních systémů a webových aplikací všeobecně. Existují sice webové stránky bez ní, většinou se však jedná o statické stránky bez nutnosti uchovávat nějaká perzistentní data. Naopak vytvoření IS bez databáze je téměř nemyslitelné, i když do jisté míry je možné její funkčnost nahradit například uchováním dat v souborech.

Použitím databáze získáváme vysoký komfort práce s daty, máme možnost je uchovávat, vyhledávat, měnit, porovnávat, vytvářet složité statistiky apod. Výhodou je také deklarativní standardizovaný jazyk SQL (Structured Query Language) používaný pro práci s daty v relačních databázích. Další podstatnou předností je samotný databázový server, který mimo jiné zajišťuje správu uživatelů a integritu dat. Podle uvedených vlastností je tedy vhodné použít DB nejen jako skladiště informací obsahového charakteru (např. jméno, rodné číslo, datum narození, číslo konta), ale v tomto případě také pro uchování jazykových verzí daného webu.

Při práci s databází je potřeba dbát na zvýšenou bezpečnost systému. Je nutné ošetřit uživatelské vstupy i chování celé aplikace tak, aby nevznikaly nekonzistentní záznamy nebo bezpečnostní rizika.

3.2.2 Struktura a rozdělení databází

Každý databázový systém se skládá ze samotné DB (databáze), která obsahuje data a SŘBD (systém řízení báze dat), což je programová vrstva řešící operace nad DB s cílem odstínit uživatele od technických detailů. Tyto dvě části obvykle zastřešuje ještě třetí vrstva, kterou tvoří aplikační rozhraní mezi uživatelem a databází. DBS umožňuje rychlý a paralelní přístup více uživatelů s možností nastavení práv.



Obrázek č. 1: Struktura databázového systému

Databázové systémy často rozdělujeme na:

- **předrelační** – hierarchické a síťové
- **relační** – data jsou organizována v tabulkách, operacemi vznikají nové tabulky, většinou obsahují velké množství jednoduchých dat
- **postrelační** – objektově-orientované, deduktivní, multimediální, atd. často obsahují mapování na relační SŘBD

3.2.3 Srovnání databází

V současné době nelze obecně říct, který databázový server je nejlepší. Každý z nich se odlišuje svými specifickými vlastnostmi a kde jeden vítězí v oblasti ceny, jiný se vyznačuje například vyšším výkonem. Pro malé či nízkorozpočtové projekty se většinou používají nekomerční databáze jako MySQL či PostgreSQL. Výhoda MySQL je především v rychlosti a jednoduchosti. V oblasti dynamického webu je tato databáze značně populární, avšak pro nasazení do rozsáhlých a komplexních systému je nevhodná (např. ještě nedávno chyběla podpora pro transakce a triggery). PostgreSQL je zdarma distribuovaná kvalitní databáze s open-source licencí. Vyniká stabilitou, rychlostí, dobrou podporou a kvalitní integrací pokročilých technologií (např. cachování indexů). V oblasti výkonu je srovnatelná například s komerční MsSQL.

Databáze Oracle je komerční produkt často využívaný pro rozsáhlé systémy. Důvodem může být jak vysoký výkon, tak i kvalitní podpora, bezpečnost a vyspělé funkce pro správu DB (nástroje pro diagnostiku a ladění výkonu či vynikající prostředky pro zálohování a obnovení dat). Kromě výše jmenovaných existuje také celá řada jiných softwarových produktů, přičemž jejich použití závisí především na požadavcích uživatele.

Samotné srovnávání databází se často provádí v rámci návrhu aplikace. Zkoumají se vybrané vlastnosti jako výkonnost, stabilita či možnosti zotavení. Již v okamžiku návrhu DB schématu je možné prakticky vyzkoušet vhodnost zvoleného systému. Náklady na několikadenní verifikaci se však ve většině případů bohatě vyplatí - odladí se DB schéma, získají se testovací data a minimalizují se změny. Důkladným prozkoumáním možností a hranic databáze tedy můžeme předejít některým budoucím problémům.

3.3 Systém pro správu verzí

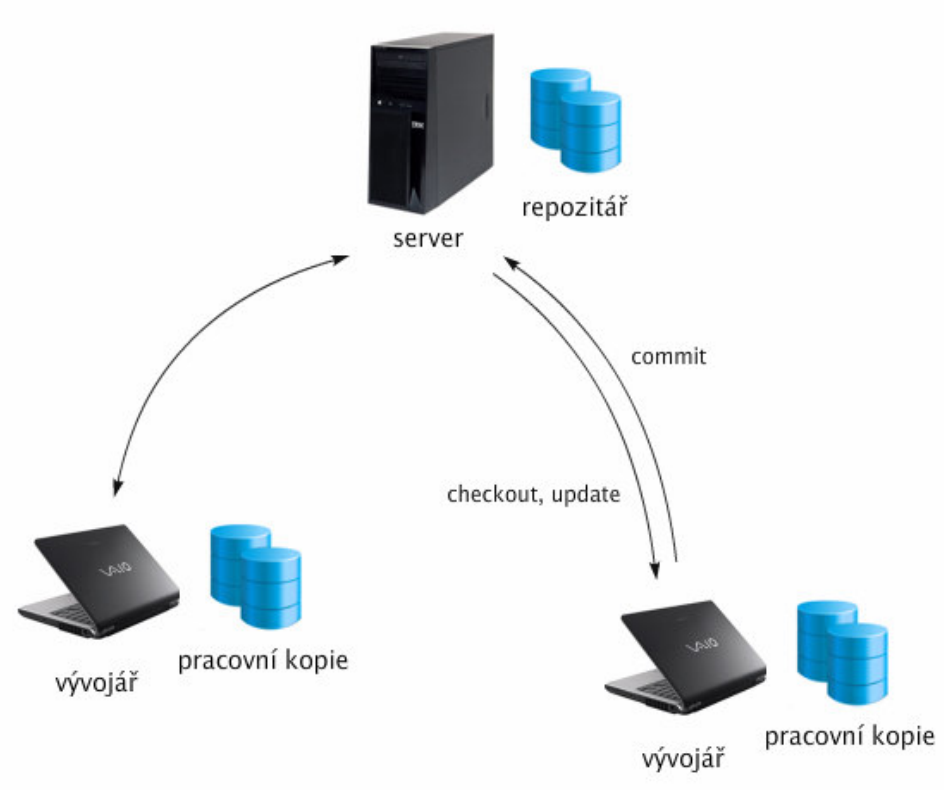
Pro správu a verzování zdrojových kódů se v rámci vývoje IS VUT používá open-source aplikace Subversion[6] (SVN). Jedná se o nástupce staršího CVS (Concurrent Versions System), z něhož jsou použity některé principy, zároveň však odstraňuje některé jeho nedostatky. SVN je vyvíjeno firmou CollabNet, Inc. a šířeno pod licencí, která umožňuje jeho bezplatné komerční použití. K dispozici

jsou zdrojové kódy a rozsáhlá anglická dokumentace. Systém je navíc implementován pro různé platformy, takže vývoj je možné provádět zároveň z různých operačních systémů (Windows, Linux, MacOS, FreeBSD, Solaris aj.).

Jedná se o software, který pomáhá programátorům udržovat vývoj přehledný. Díky tomu, že zaznamenává jakékoliv změny v projektu, stává se velmi užitečným pomocníkem při řešení konfliktů různých verzí a v případě potřeby umožňuje návrat v historii zpět ke stabilní verzi. SVN je vhodné použít především tam, kde se na vývoji podílí více programátorů, avšak stále častěji je používán také jednotlivci, kteří poznali výhody verzování.

Systém je postaven na architektuře klient-server. Stranu serveru tvoří tzv. repozitář, který zde vystupuje ve funkci centrálního úložiště. K němu lze přistupovat pomocí klienta několika způsoby- od klasické příkazové řádky, přes webové rozhraní, až po nástroje integrované do uživatelského rozhraní operačního systému (TortoiseSVN[7]).

Jednou z výhod Subversion je udržování pouze nezbytných dat na serveru. Přestože si pomocí klienta můžeme prohlížet jednotlivé soubory, nejedná se o souborový systém v pravém slova smyslu. Ukládají se pouze informace popisující rozdíly mezi verzemi, nikoliv každá verze systému zvlášť. Nároky na diskový prostor jsou tak překvapivě malé.



Obrázek č.2: Pracovní cyklus SVN [8]

Stručný popis práce s SVN: Programátor si nainstaluje klienta a příkazem *checkout* si stáhne aktuální verzi z repozitáře – tím se u něj vytvoří tzv. pracovní kopie. Provede změny ve zdrojových kódech a příkazem *commit* tyto změny promítne zpět do repozitáře. Po této aktualizaci se zvýší číslo revize (verze) o 1. Kromě toho provádí také *update*, což je aktualizace pracovní kopie. Tato operace je nutná především kvůli spolupráci s ostatními vývojáři.

Síla této aplikace spočívá v tom, že několik programátorů může zároveň upravovat stejný soubor, aniž by došlo k chybě. Vývojář může příkazem *diff* okamžitě vypsat rozdíly mezi pracovní kopií a repozitářem, takže okamžitě zjistí, jaké změny v souboru provedli ostatní (pokud již provedli příkaz *commit*). Proto je doporučováno provádět *commit* co nejčastěji a v ucelených balících.

Výše popsané vlastnosti SVN značně usnadňují vývoj softwaru, obzvláště pokud na projektu pracuje více programátorů, zároveň vzniká automatická záloha systému v každém okamžiku vývoje. Přitom nemusí jít pouze o správu zdrojových kódů – pomocí systému Subversion lze s úspěchem verzovat soubory s jakýmkoliv obsahem.

4 Systém IS VUT

4.1 Základní popis

IS VUT je rozsáhlý systém skládající se z několika aplikací: *Apollo, Web, SAP* a dalších, přičemž každá z těchto částí obsahuje několik modulů[9]. Pro uchovávání velkého množství dat slouží několik databází, které se také používají k různým účelům (podrobnosti viz kapitola 4.4).

4.1.1 Důležité části systému

Web – aplikace IS VUT s webovým rozhraním. Obsahuje moduly *Portal, StudIS, Teacher* atd.
Apollo – aplikační rozhraní IS VUT. Po instalaci klientské aplikace (3,8 MB) umožňuje přihlášení do systému a následné prohlížení či správu systému. Jednotlivé moduly slouží především pracovníkům VUT. Jejich účelem je například snadná komunikace v rámci systému, správa osobních informací, projektů a publikací, podpora přijímacího řízení a výzkumu, informace o jednotlivých fakultách atd. V některých případech se používá i pro studijní účely.
SAP – systém pro ekonomii a řízení. Poskytuje pouze omezený počet uživatelů, z nichž každý se platí. Z finančních důvodů tedy není možné poskytnout jej všem zaměstnancům VUT
Pasportizace GT Facility – jedná se o získání spolehlivých aktuálních údajů o plošném a dispozičním uspořádání všech podlaží budov, které jsou majetkem VUT v Brně.

Z hlediska diplomové práce se budeme dále podrobněji věnovat pouze oblasti webu. Jak již bylo zmíněno, vývoj probíhá v jazyce PHP s využitím verzovacího systému Subversion. Zdrojové kódy všech částí webu a všechny jejich verze jsou uloženy v jednom centrálním repozitáři. Z něj se pak aktualizují jednotlivé instance webových aplikací. Postup vývoje: vývojáři ukládají pomocí Subversion změny do repozitáře. S každou změnou vzroste číslo verze o 1. Některá verze je ve fázi testování a jedna z už otestovaných verzí běží v ostrém provozu.

4.2 Instance web aplikací

verze	databáze	url	revize	správce	přístup
ostra	CDBX	https://www.vutbr.cz/	A	webaři	všichni
test	CISD	https://ent.ro.vutbr.cz/	B	webaři	všichni
vyvoj	CISD	https://ent.ro.vutbr.cz/vyvoj/	X	webaři	IP
ostratest	CDBX	https://ostratest.ro.vutbr.cz/_base/www_base/	X	webaři	IP
student	CISD	https://3wtest.ro.vutbr.cz/.../	C	studenti	IP

Tabulka č.2 – Přehled jednotlivých instancí aplikace s přidělením práv[10]

Popis jednotlivých instancí:

Ostra **–** produkční instance, která běží na webovém clusteru, je dostupná pro návštěvníky a uživatele. Zdrojové kódy se synchronizují z adresáře na vývojovém serveru každé dvě minuty. Připravuje se uložení na sdíleném disku.

Test – verze určená pro testování. Zde by se měly všechny (nebo alespoň větší změny) otestovat, než se dostanou do ostrého provozu.

Vyvoj – několik instancí sloužících pro potřeby vývojářů. Společná instance *vyvoj/all* je udržována beze změn a může být používána pro rychlé úpravy.

Ostratest – instance má stejné zdrojové kódy jako *vyvoj/all*, avšak připojuje se k databázi CDBX (viz kapitola 4.4.2)

Student – instance jednotlivých studentů a externích spolupracovníků

Vývoj aplikace a ostrý provoz samozřejmě probíhá na různých databázích, aby nedošlo k poškození důležitých dat. Původně instance ostratest sloužila k testování aplikace na ostrých datech, v současné době se příliš nevyužívá, protože každou neděli dochází k pravidelné synchronizaci CISD podle CDBX (viz kapitola 4.4.2), takže testovací databáze je v podstatě shodná s ostrou verzí. Instance student je využívána studenty k realizaci některých projektů či diplomových prací.

Webové rozhraní informačního systému je vyvíjeno v jazyce PHP. Pro jednotlivé stránky se používá kódování Windows-1250, což je znaková sada používaná systémem Microsoft Windows pro reprezentaci textů ve středoevropských jazycích používajících latinku - např. pro češtinu, slovenštinu, polštinu, maďarštinu, rumunštinu a další. U ostatních jazyků se mohou objevovat problémy se zobrazením národních znaků. Toto řešení je pro vícejazyčné aplikace nevhodné, proto se plánuje přechod na kódování UTF-8, na němž už interně běží databáze.

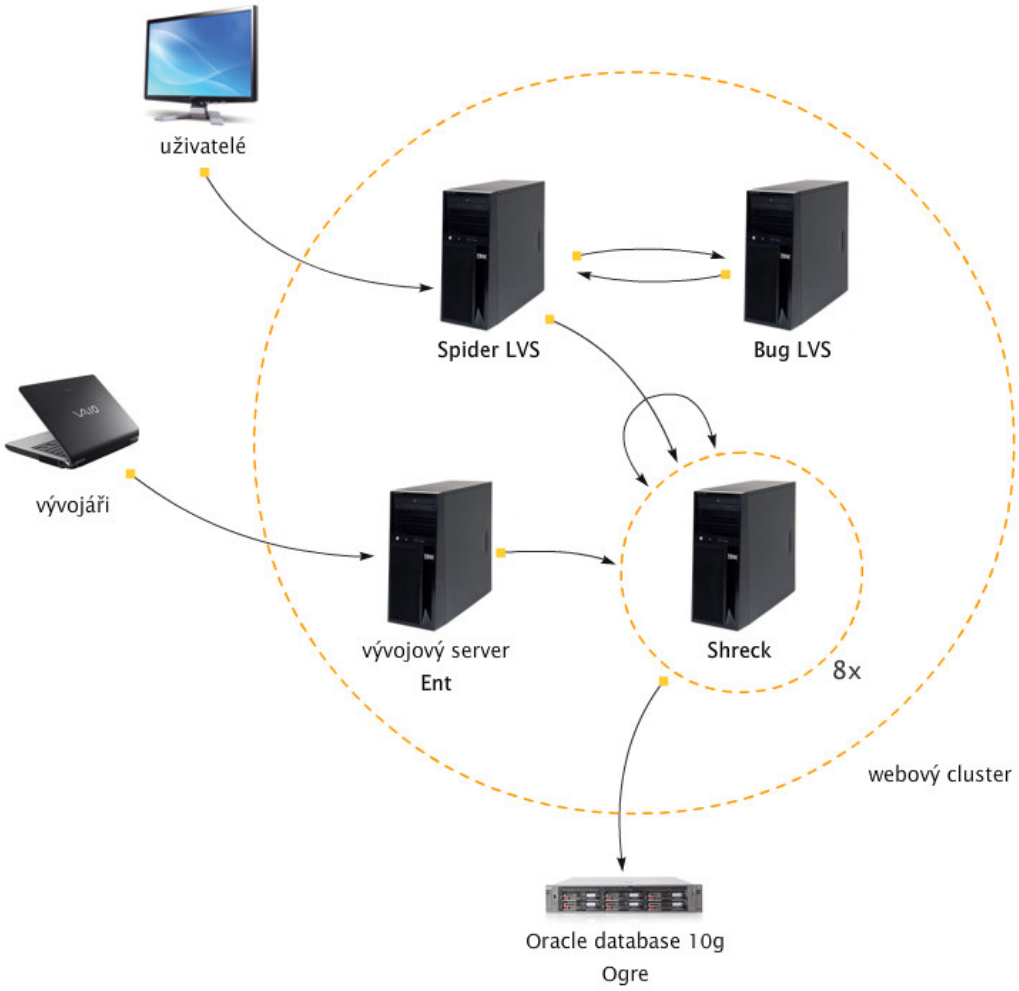
4.3 Struktura serverů

Na informační systém VUT jsou kladeny vysoké výkonnostní nároky, které jsou při použití jediného serveru prakticky nesplnitelné (bylo by například možné použít superpočítač, ovšem jeho cena bývá příliš vysoká). Proto se pro obsluhu požadavků používá tzv. webový cluster (viz obrázek č.3). Jedná se vlastně o několik serverů, z nichž každý hraje důležitou roli. Pomyslnou vstupní bránu tvoří tzv. Linux Virtual Server (LVS), který využívá linuxový modul pro rozdělování IP paketů *ipvs*. Tato služba běží na serverech Spider a Bug, které se vzájemně zálohují, přičemž jeden z nich je vždy aktivním LVS routerem a druhý záložním. Router pak rozděluje výkon jednotlivým serverům Shreck1 až Shreck8 podle nastavených vah, přičemž každou váhu ovlivňuje výkonnost a dostupnost serveru. Na každém serveru Shreck je spuštěn webový server Apache, na němž je k dispozici celý portál, takže při výpadku jednoho či několika z nich je celý IS uživatelům nadále k dispozici.

Z pohledu vývojářů je důležitý server Ent fungující jako hlavní vývojový server. Zde je umístěn centrální repozitář, takže pomocí služby SVN je možné nahrávat či získávat jednotlivé vývojové instance. Programátoři přitom nemají přímý přístup na servery Shreck, takže neohrožují ostrý provoz systému. Teprve po důkladném otestování jsou zdrojové texty dané vývojové instance replikovány na servery Shreck.

Použití clusteru tedy umožňuje více zatížit výkonnější servery či naopak snížit zatížení nebo zajistit vyřazení nedostupných serverů. Tím se podstatně urychluje odezva uživateli a zvyšuje odolnost proti výpadkům, což je důležité především v krizových situacích (např. registrace předmětů). Nejslabším článkem tak zůstává DB server, avšak plánuje se taktéž přechod na tzv. Real Application Cluster (RAC).

S rostoucími požadavky na výkon je samozřejmě nutné také investovat do hardwaru. Tabulka č. 3 ukazuje aktuální konfiguraci nasazených serverů.



Obrázek č.3: Struktura webového clusteru

Název	**Použití**	**Počet procesorů**	**CPU**	**Paměť**
Ogre	DB server	4	3,66 GHz	26 GB
Shreck1-4	Webový server	8	1,86 GHz	4 GB
Shreck5-8	Webový server	2	3,40 GHz	3 GB

Tabulka č. 3: Přehled aktuální konfigurace serverů

4.4 Databáze IS VUT

4.4.1 Oracle Database

V rámci IS VUT se používá komerční software Oracle *Database 10g Enterprise with Partitioning option*[11]. V současnosti můžeme mluvit o nejvýkonnějším databázovém serveru s velice pokročilými možnostmi zpracování dat, vysokým výkonem a důrazem na bezpečnost. Důvěryhodnost implementace bezpečnostních mechanismů dokazuje i mnoho nezávislých ohodnocení, která databázový server Oracle opakovaně úspěšně absolvoval. Podporuje nejen standardní relační dotazovací jazyk SQL (Structured Query Language) podle normy SQL92, ale také proprietární firemní rozšíření Oracle (např. pro hierarchické dotazy), imperativní programovací jazyk PL/SQL rozšiřující možnosti vlastního SQL (v tomto jazyce je možné tvořit procedury, uživatelské funkce, programové balíky a triggery), dále podporuje objektové databáze a databáze uložené v hierarchickém modelu dat (XML databáze, jazyk XSQL). Systém VUT využívá klasické relační schéma.

Tento výkonný nástroj byl zvolen především pro svou rychlost a spolehlivost, což je při uchovávaném objemu dat velmi důležité. Při řešení problémů je navíc možné využít podpory od firmy Oracle. Podle [12] umožňuje zakoupená verze rozdělovat data v tabulkách podle různých kritérií na menší části. Například rozdělení tabulky elektronického indexu, která obsahuje asi 1 milión záznamů podle akademických let, umožňuje zrychlit dotazy v řádu stovek procent, protože se budou vyhledávat data v zadaném akademickém roce a ne mezi celým miliónem záznamů v rámci celé tabulky. Další výhodou jsou bezesporu paralelní dotazy, kdy lze jedním dotazem vyhledat patřičná data 1 až x-krát rychleji, kde x je index paralelizace. Z mnoha dalších kladných vlastností bychom nakonec jmenovali také možnost připojení k databázi v režimu *shared*, díky němuž lze podstatnou část prováděného kódu sdílet mezi několika uživateli. Tímto vzniká podstatná úspora výpočetních prostředků. Sdílení má ale také několik nevýhod (např. vyšší zátěž CPU), protikladem však může být menší počet procesů.

4.4.2 Popis jednotlivých databází

Pro účely vývoje, testování a ostrého provozu byly zřízeno několik databází. Na oddělení CVIS (Centrum výpočetních a informačních služeb) je provozováno několik kopií centrálního datového skladu. Z hlediska diplomové práce považujeme za nejdůležitější tyto tři:

- **CDBX** – obsahuje ostrá data. Tato databáze slouží k přímému provozu informačního systému VUT. V době nízkého provozu (nejčastěji v noci) dochází pomocí vlastními silami vytvořeného systému k pravidelnému zálohování.
- **CISD** – jedná se o testovací databázi, která vzniká pravidelnou obnovou ze zálohy ostré databáze CDBX, obsahují tedy stejná data. Obnovu lze provádět v libovolném čase a libovolně často (ovšem s rozumnou periodou a nikoliv zbytečně). Kromě automatického vytváření je samozřejmě možné vytvořit kopii i manuálně. Podrobnosti o aktualizaci CISD viz [13].
- **CISB** – tato databáze je určena pro vývoj. Slouží studentům a pracovníkům k realizaci projektů či vývoji nových částí systému, proto může mít jinou strukturu a obsahuje jiná data než předchozí jmenované. Z těchto důvodů se CISB obnovuje pouze několikrát za rok. Při obnovení zároveň dochází ke změně a promíchání osobních dat, aby se zamezilo jejich zneužití.

Využití databází v procesu vývoje: zdrojové kódy jsou vytvořeny na vývojovém serveru a vyzkoušeny na CISB. Poté začíná, bez ohrožení ostrého provozu, testování na reálných datech CISD. Po úspěšném testování je nová verze systému připravena k nasazení do reálného provozu. Zdrojové kódy jsou duplikovány na servery a aplikace začíná fungovat na databázi CDBX.

Každá databáze je rozdělena do několika logických celků pomocí schémat, přičemž každé z nich sdružuje data z určité oblasti systému. Jednotlivá schémata jsou dále dělena na úroveň tabulek obsahujících vlastní data.

Příklad existujících schémat:

- **APOLLO** – obsahuje data pro aplikační rozhraní IS VUT, uživatelská nastavení atd.
- **BRUTISADM** – rozsáhlé schéma, v němž byla původně data z informačního systému Brutis. Postupně se ale rozšiřovalo a dnes obsahuje například osobní informace, kontakty, informace o organizačních jednotkách VUT či data týkající se vědy a výzkumu.
- **LOGGER** – zde jsou uloženy informace o změnách nebo mazání záznamů v důležitých tabulkách. Kromě změněných sloupců zde můžeme najít také čas a autora změny, což umožňuje zpětné dohledání.
- **ST01** – další rozsáhlé schéma uchovávající informace týkající se studia. Jedná se například o studijní programy a obory, individuální plány, data týkající se elektronických indexů či studijní výsledky.
- **DOKUMENTACE** – obsahuje data týkající se dokumentace centrální databáze.

4.4.3 Konvence pojmenování DB objektů

Pro snadnější přehlednost se pojmenování DB objektů (tabulky, klíče, indexy, triggery atd.) řídí dohodnutými konvencemi[14]. Příklad nejdůležitějších pravidel:

Primární klíče:
Tvoří se prefixem PK a názvem tabulky.
Příklad: PK_NAZEV_TABULKY

Cizí klíče:
Tvoří se prefixem FK, názvem referované tabulky a názvem tabulky referující v tomto pořadí. V případě potřeby provázání dvou tabulek více než jedním cizím klíčem je rozlišujeme připojením vhodného postfixu charakterizujícího klíč.
Příklad: FK_NAZEV_REFEROVANÉ_TABULKY_NAZEV_REFERUJÍCÍ_TABULKY

Indexy:
Tvoří se prefixy IX, IXUF, UQ, názvem tabulky a výčtu sloupců užitých v indexu. Prefixy mají následující význam: IX = index, IXUF = unikátní funkční index (neexistuje unikátní constraint), UQ = unikátní index, unikátní constraint.
Příklad: IX_NAZEV_TABULKY_VYCET_SLOUPCU

Sekvence:
Tvoří se prefixem SQ a názvem tabulky (resp. s názvy tabulek oddělených podtržítky), která sekvenci využívá. Jako název sekvence je možné použít i název sloupce, který využívá sekvenci a je přítomný ve všech tabulkách využívajících tuto sekvenci.
Příklad: SQ_NAZEV_TABULKY

Triggery:
Tvoří se prefixem TG, názvem triggeru. Je možné název doplnit postfixy složenými z písmen A, B, I, U, D, mající tento význam: A = after, B = before, I = insert, U = update, D = delete. V případě že je třeba rozlišit, zda-li jde o trigger pro řádek nebo pro tabulku, připojujeme postfix R nebo T.
Příklad: TG_NAZEV_TRIGGERU

Pohledy:
Tvoří se prefixem V a názvem pohledu. Název pohledu by měl intuitivně vypovídat o jeho obsahu, případně oblasti využití.
Příklad: V_NAZEV TABULKY

4.4.4 Použití databáze

Pro samotnou práci s databází byla vytvořena třída *ClassOracle*, která s využitím knihovny ADOdb[15] zprostředkovává připojení a manipulaci s DB. Knihovna vytváří abstrakci databáze a mimo jiné umožňuje zpracování transakcí, zpracování chyb, logování aj. Samotná třída pak pouze upravuje použití knihovny a přidává některé funkce. Obzvláště důležitá je možnost bindování proměnných v dotazech, což nejenom brání útokům typu SQL injection, ale také podstatně urychluje zpracování dotazů.

```
$sql = "
    select ....
           ob.obor_id,
           ob.nazev{$lang_suffix} as program_nazev,
           ....
      from st01.rocnik ro,
           st01.stupen sp,
           st01.obor ob,
           ....
     where ro.rocnik_id = :p_rocnik_id
       and ro.status in (0, 9)
       and sp.stupen_id = ro.stupen_id
       and sp.status in (0, 9)
       and ob.obor_id = sp.obor_id
       and ob.status in (0, 9)
       ....
";
$ba = array( 'p_rocnik_id' = $rocnik_id, );

if ( ! $rs = run_sql($sql,$ba) ) return false;
if ( $rs->EOF ) return array();
return $rs->fields;
```

Kód č.1: Ukázka práce s databází, bindování proměnných

Pokud je SQL dotaz úspěšně proveden, získáváme objekt třídy ADORecordSet (v příkladu proměnná $rs). Tento objekt obsahuje celou řadu funkcí pro získávání jednotlivých záznamů či celých skupin záznamů (např. FetchRow, FetchObject, GetArray, MoveNext, MoveFirst, FieldCount atd.). Za zmínku stojí také možnost přímého generování HTML tabulky.

5 Současný a požadovaný stav překladů

5.1 Aktuální stav překladů

V současné době je veškerá práce s překlady manuálně náročná a zbytečně složitá. Programátoři během vývoje vytvářejí zdrojové kódy, které obsahují nepřeložené texty. Tyto texty buď přímo odesílají překladatelce emailem nebo je uchovávají v souboru a odešlou je hromadně. Překladatelka vytvoří překlady do příslušného jazyka a odešle je zpátky, případně se zeptá na upřesňující informace. Nové texty jsou vloženy do souboru *func_tr_data.php*. Překladatelka počítá, kolik textů přeložila, a žádá o odměnu.

Přeložené texty jsou kvůli snadnosti hledání a zachování konzistence uloženy v jednom souboru. V rozsáhlém systému je však potřeba značné množství textů, takže rychle narůstá velikost souboru, což v době velké zátěže IS způsobuje problémy a pokles výkonu.

5.1.1 Datový soubor

Jak již bylo zmíněno, přeložené texty jsou uchovávány v souboru *func_tr_data.php*. Jedná se o klasický PHP skript, jehož cílem je vytvořit globální proměnnou *$tr_mem* pro snadný přístup k překladům. Tento soubor tedy obsahuje asociativní pole překladů, kde klíčem (identifikátorem) je krátký text bez diakritiky a mezer a hodnotou je pole obsahující jednotlivé jazykové verze.

```
$tr_mem['search_courses_title'] = array(
      'Vyhledávání předmětů',
      'Search of courses'
);
$tr_mem['schedule_teacher'] = array(
      'Rozvrh pro vyučujícího',
      'Teacher´s study schedule'
);
```

Kód č.2: Ukázka datového souboru

V současné době má soubor zhruba 2400 záznamů. Vzhledem k tomu že klíč slouží jako identifikátor, je nevhodné, aby jej při takovém množství vytvářeli programátoři ručně. V několika případech je dokonce u odlišného textu stejný klíč, čímž nesplňuje svou původní úlohu identifikátoru (k této situaci došlo spojením několika souborů). Tento problém lze částečně vyřešit podmíněným použitím překladu v závislosti na požadovaném kontextu aplikace, avšak je vhodné jej vyřešit dříve, než se konflikty rozrostou.

5.1.2 Funkce pro překlad

Samotný překlad zajišťují tři funkce ze souboru *func_tr.php*. Každá z nich má jiný účel, ale pro všechny platí, že získávají data z překladového souboru, přesněji řečeno z globální proměnné *$tr_mem*.

```
function insert_label( $id_label, $visible_en=1, $ret=0 )
function get_lang( $tr_mem_key, $in_lang=null )
function tr( $cs, $lang=null, $force_ucfirst_en=0 )
```

Kód č.3: Funkce pro překlad

Parametrem každé z těchto funkcí je řetězec, který slouží jako klíč pro vyhledání překladu (jedná se o proměnné *$id_label, $tr_mem_key, $cs*). Protože jsou tyto parametry ještě před vyhledáním vyhodnocovány, je možné na ně aplikovat stejné funkce jako pro práci s řetězci, tedy například:

```
tr("Bylo zaregistrováno ".$pocet." předmětů. ");  ≈
tr("Bylo zaregistrováno 5 předmětů");
```

Kód č.4: Ukázka parametrických textů

Problém je ovšem v tom, že tímto způsobem není možné texty s parametry zpracovávat, protože bychom museli mít všechny použitelné varianty překladu, které by se lišily např. pouze číslem, které navíc není potřeba překládat.

Tato situace byla vyřešena pomocí php funkce *sprintf(format, arg1, arg2, arg++),* která umožňuje tisk formátovaného textu do proměnné (podrobnosti viz [16]).

Znak	**Typ**	**Formát**
%	žádný	znak %
b	celé číslo	binární číslo
c	celé číslo	znak s touto ASCII hodnotou
d	celé číslo	desítkové číslo
f	reálné číslo	číslo s desetinnou tečkou
o	celé číslo	osmičkové číslo
s	řetězec	řetězec

x	celé číslo	šestnáctkové číslo
X	celé číslo	šestnáctkové číslo

Tabulka č.4: Přehled formátovacích znaků funkce sprintf()

U neparametrických překladů je tedy možné přímo použít funkce pro překlad, u parametrických se využívají v kombinaci s funkcí *sprintf()*.

```
$str = sprintf( tr( 'Předměty zapsané v akad. roce %d.'), $rok);
echo ($str);
```

Pokud má uživatel zvolen zobrazení stránek v angličtině, zobrazí uvedený kód následující text:

```
Courses registered in academic year 2008.
```

Kód č.5: Příklad použití formátovaného překladu

Ačkoliv je tento způsob poměrně výhodný, skrývá v sobě jistá omezení, mezi něž patří například nemožnost vytvořit jazykové verze konkrétního textu s odlišným pořadím parametrů.

5.1.3 Problémy stávajícího řešení

Mezi hlavní problémy aktuálního řešení patří:

- nejednoznačnost (klíč není jednoznačným identifikátorem)
- nepřehlednost (z krátkého klíče nelze poznat o jaký text jde)
- nelze zjistit reference (není možné jednoduše a rychle poznat, kde všude se text používá a jestli je možné jej odstranit)
- nárůst velikosti datového souboru (vysoké nároky na dobu kompilace a paměť)
- zbytečné načítání všech textů při procházení části systému
- neustálé přeposílání emailů a kopírování a z toho pramenící neochota k práci s překlady, ruční kopírování textů navíc umožňuje vznik dalších chyb
- nelze poznat, co přeložil překladatel
- obtížná správa překladů

Naopak výhodou stávajícího řešení je fakt, že reflektuje zachování různých překladů pro různé vývojové verze, neboť soubor s překlady je součástí systému a podléhá verzování.

5.2 Požadavky vývojářů

- automatizovat procházení zdrojových kódů a hledání řetězců určených k překladu
- vytvořit administrační rozhraní pro správu překladů
- logování změn
- usnadnit komunikaci s překladatelkami
- vyřešit nepříjemně narůstající velikost souboru s překlady, případně jej rozdělit do menších celků, zjednodušit orientaci v textech
- nové řešení musí respektovat různé verze překladů pro různé vývojové verze IS
- minimalizovat duplicitu překladů, umožnit přidání dodatečných informací (např. autor překladu, datum, poznámky…)
- optimalizace výkonu, snížit zatížení serveru
- bezpečnost aplikace, ošetření rizikových událostí

5.3 Požadavky překladatelů

- vytvoření administračního rozhraní, které bude umožňovat správu překladů, případně náhled starších verzí konkrétního překladu
- dodatečné informace o překládaném textu
- počítadlo přeložených textů
- výpočet odměny
- podpora komunikace s vývojáři

6 Návrh řešení

Tato kapitola se zabývá teoretickým popisem řešení pro jednotlivé požadavky. Poslední podkapitola poskytuje ucelený pohled na navrhovaný systém překladů.

6.1 Rozpracování požadavků

6.1.1 Uchovávání textů

Cíl: Správa textů, snadnější orientace, redukce duplicitních informací, verze překladů, přidání dodatečných informací.

Z výše uvedených požadavků vyplývá, že nejvhodnějším řešením je jak uchovávání textů v databázi, tak i vytváření překladových souborů pro rychlejší načítání. Tímto krokem vyřešíme hned několik problémů najednou. Texty, které se dříve ukládaly do jediného souboru a způsobovaly nepříjemné zpomalení a nepřehlednost, budeme nyní spravovat pomocí databáze. Kromě samotných překladů můžeme snadno uchovávat dodatečné informace, aniž by je musel vývojář zapisovat do komentářů zdrojového kódu. Díky vhodné struktuře tabulek můžeme navíc zjistit v kterých verzích a souborech se daný překlad vyskytuje, kdo a kdy jej vytvořil či změnil atd. Pokud by tyto informace byly podle stávajícího řešení uloženy v souboru, jeho velikost by dále několikanásobně vzrostla, přičemž by narůstala také duplicita některých informací. Otázka duplicity tak závisí na správném návrhu DB, avšak například pomocí cizích klíčů ji můžeme redukovat na minimum. Převod překladů do databáze představuje výrazný krok vpřed v oblasti možností správy, pohodlí i výkonu, může však mít zároveň negativní dopad v podobě narůstající složitosti či výskytu nových chyb.

6.1.2 Vyhledávání

Cíl: Automatické hledání řetězců určených k překladu

Použitím systému Subversion pro správu verzí můžeme detekovat změny v souboru a reagovat na ně. Celá automatizace tedy bude fungovat takto: programátor změní soubor a pomocí SVN jej uloží do repozitáře. Detekují se změny a spustí se skript (tzv. parser), který v upraveném souboru vyhledá překlady a aktualizuje databázi. Tento parser bude pomocí konečného automatu nebo regulárních výrazů procházet zdrojový kód a hledat řetězce určené k překladu. Skript musí správně reagovat na jednotlivé změny souboru, tedy např. pokud je soubor smazán, zrušit všechny jeho texty či naopak při změně aktualizovat pouze změněné texty apod.

Tím, že se některé operace stanou bezobslužné a řeší je počítač, objevují se nové problémy, o kterých zatím rozhodovali programátoři. Zde je potřeba dořešit několik otázek, z nichž pro představu zmíníme tyto: Pokud se po každé aktualizaci vyhledají změny v překladech, jakým způsobem zajistit, aby se texty z různých souborů v databázi neukládaly duplicitně? Jak uchovávat informace o tom, ke které verzi řetězec patří? Jak vyřešit situaci, kdy v novější verzi aplikace řetězec chybí (smazáním bychom porušili konzistenci starší verze)?

6.1.3 Administrační rozhraní

Cíl: Usnadnění práce překladatelkám a vývojářům

Administrační rozhraní bude sloužit k vytváření a prohlížení překladů. U této části systému je důležité ošetřit souběžnost událostí (např. pokud se současně generují soubory s překlady a zároveň překladatelka provádí změny v databázi). Jiným problémem může být také následující situace: pokud by chtěla například překladatelka uložit rozpracované texty tak, aby se s nimi nepracovalo jako s přeloženými. Odpovědí na tyto otázky může být uchovávání dočasných informací v pomocné tabulce, odkud se budou již „naostro“ uložená data mazat.

Dalším problémem může být skutečnost prolínání rolí vývojáře a překladatele. Je vhodné, aby mezi překladatele byli zařazeni všichni vývojáři a tímto jim byl umožněn přístup k vytvořeným textům, zároveň by však měla existovat určitá omezení, aby všichni programátoři neměli přístup k editaci překladů. Nakonec tedy bylo zvoleno řešení, kdy všichni vývojáři jsou zároveň překladateli, avšak každý překladatel má nastaven příznak určující právo k zápisu.

Součástí aplikace by měl být i formulář pro rychlejší komunikaci překladatelů s programátory. Přihlašování bude po konzultaci řešeno centrálně pomocí již fungující správy uživatelů v rámci IS VUT.

6.1.4 Logování

Cíl: Uchovávání změn překladů

Každá změna překladů, ať už ji provede kdokoliv, se bude ukládat buď v souboru nebo v databázi. Kromě samotného popisu změny by se měly uchovávat také informace o času změny a osobě, která ji provedla. V souvislosti s logováním změn v překladech pomocí administračního rozhraní je také potřeba uchovávat informace o parsování zdrojových kódů a generování překladových souborů.

Tento požadavek sice přímo nesouvisí s funkčností systému, je však důležitý z hlediska bezpečnosti aplikace a zpětného hledání chyb. Jedná se především o logování změn v rámci aplikace, v databázi jsou změny ošetřeny pomocí triggerů.

6.1.5 Zvýšení výkonu

Cíl: Urychlení vývoje i práce s překlady

Již samotné použití databáze vede k většímu komfortu a zvýšení výkonu. Toto řešení by však při vyšším provozu zbytečně příliš zatěžovalo DB server a zpomalovalo provoz. Bylo tedy rozhodnuto o vytvoření generátoru, který po změně jazykových verzí v databázi vytvoří několik souborů s překlady, pro každou část systému jeden. Tímto způsobem dojde ke značnému urychlení překladu. V současném systému se sice pro získání jazykových verzí nepoužívá DB vůbec (texty jsou uloženy pouze v jediném souboru *func_tr_data.php*), avšak pokud chceme využívat všech jejích výhod, je potřeba minimalizovat její použití na nezbytnou úroveň tak, aby zůstal zachován výkon. Podle návrhu by se mělo s překlady v databázi pracovat pouze při změně verze nebo při aktualizaci jazykových verzí pomocí administračního rozhraní.

Nové řešení také znamená zvýšení výkonu pro programátory a překladatele, neboť jim podstatnou měrou usnadní práci s texty a tím urychlí celkový vývoj systému.

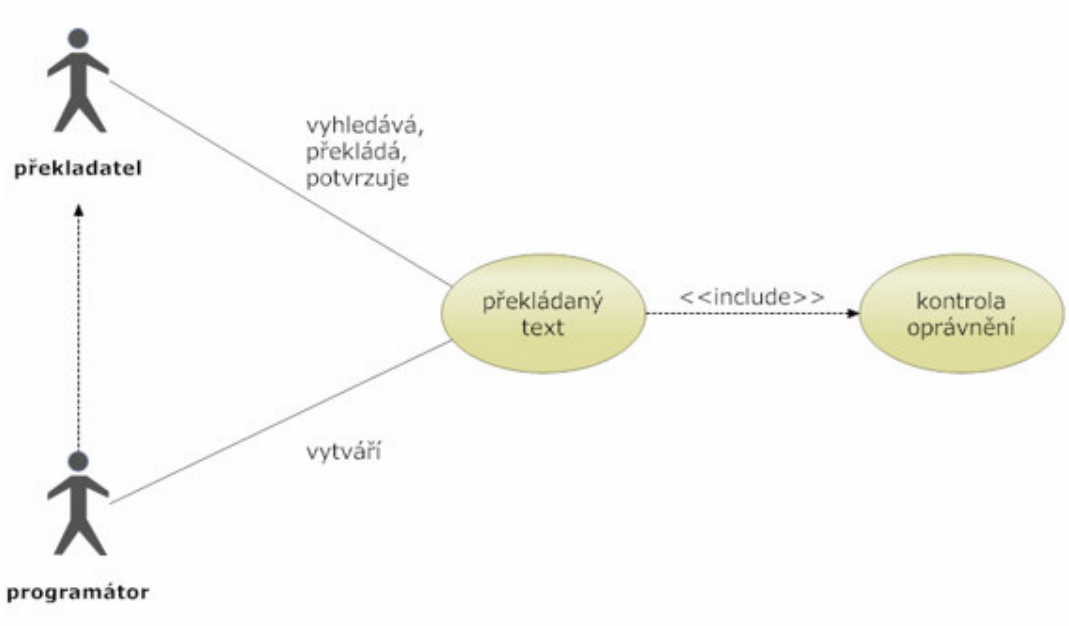
6.2 Analýza funkčnosti

6.2.1.1 Use case diagram

Diagram případů použití modeluje dynamické (funkční) části systému z pohledu uživatele. Je určen především k definici chování systému, aniž by detailně odhaloval jeho vnitřní strukturu.

6.2.1.2 Stručný popis

V procesu překladu textů figurují dvě uživatelské role: programátor a překladatel, přičemž role překladatel obsahuje i programátory (aby mohli jednoduché texty překládat sami). Programátoři vytvářejí zdrojové kódy a tím vznikají texty určené k překladu. Ty jsou pomocí skriptů přeneseny do databáze. Po přihlášení do administračního prostředí je možné texty prohlížet a vytvářet k nim jazykové verze. Tyto přeložené texty se ukládají do pomocné tabulky a teprve po potvrzení jsou přeneseny mezi ostrá data. Potvrzení může provádět pouze kvalifikovaný překladatel. Tímto je tedy zajištěno, že programátoři sice mohou jednotlivé texty překládat, ale vždy je zkontroluje překladatel.



Obrázek č. 4: Use case diagram

6.2.1.3 Specifikace případů použití

Vytváření překládaných textů

Účastníci: programátor

Vstupní podmínky: programátor s oprávněním pro commit vytvoří zdrojový kód obsahující texty určené k překladu

Tok událostí:

1. Zdrojové kódy s texty jsou příkazem commit promítnuty do centrálního repozitáře
2. Spustí se parser, který vyhledá texty a uloží je do databáze

Výstupní podmínky: texty určené k překladu jsou uloženy v databázi

Vyhledání textu

Účastníci: programátor nebo překladatel

Vstupní podmínky: texty jsou uloženy v databázi a uživatel má přístup do administračního rozhraní

Tok událostí:

1. Uživatel se přihlásí do administračního rozhraní
2. Vybírá si, zda chce prohlížet přeložené nebo nepřeložené texty
3. Vybírá si konkrétní text
4. Prohlíží si detailní informace

Výstupní podmínky: vyhledaný text

Překlad textů

Účastníci: programátor nebo překladatel

Vstupní podmínky: texty jsou uloženy v databázi a uživatel má přístup do administračního rozhraní

Tok událostí:

1. Uživatel se přihlásí do administračního rozhraní
2. Vyhledá si text, který chce přeložit či upravit
3. Doplňuje nebo upravuje jazykové verze
4. Stiskne tlačítko Uložit

Výstupní podmínky: přeložený text je uložen v dočasné tabulce *preklady_text_temp*

Potvrzení textu

Účastníci: programátor nebo překladatel

Vstupní podmínky: text je správně přeložen, uživatel má přístup do administračního rozhraní a navíc má oprávnění k potvrzení překladu

Tok událostí:

1. Uživatel se přihlásí do administračního rozhraní
2. Z přeložených textů si vybírá ten, který chce potvrdit
3. Kontroluje správnost překladu
4. Stiskne tlačítko Potvrdit

Výstupní podmínky: přeložený text je potvrzen a uložen do tabulky *preklady_text*

6.3 Návrh funkčních částí

Podle vstupních požadavků a následné analýzy byl systém překladů rozdělen do několika funkčních bloků, které se starají o jednotlivé úlohy:

6.3.1 Parser

Zajišťuje počáteční vyhledání textů ve zdrojových kódech a jejich správné uložení do databáze. Spouští se při akci *commit* (promítnutí změn do centrálního repozitáře). Samotné parsování funguje na principu konečného automatu zastoupeného ekvivalentním regulárním výrazem, přičemž cílem je správně vyhledat a zpracovat první parametr překladových funkcí (podrobnosti viz kapitola Implementace). Při ukládání do databáze je potřeba dát pozor na správné kódování textu a uložení speciálních znaků (např.<,>,“,’,\n atd.).

6.3.2 Generátor

Úkolem generátoru je získání překladových textů z databáze a vytvoření příslušných překladových souborů. Protože pro každou část (aplikaci) IS VUT je potřeba jeden soubor, a každá instance systému je určitou vývojovou verzí popsanou číslem revize, je k vytvoření jednoho překladového souboru potřeba znát číslo revize a aplikaci, pro kterou bude generován.

6.3.3 Administrační rozhraní

Jak již bylo dříve popsáno, tato část slouží vývojářům a překladatelům ke správě překladů a mají k ní přístup pouze oprávněné osoby. Protože je vhodné zařadit toto rozhraní do celého IS VUT a využít například centrální přihlašování do systému, je potřeba v rámci přihlašování také ověřit, zda má osoba oprávnění k administraci překladů a pokud ano, reagovat na to například zobrazením nové položky v menu. Rozhraní by mělo být navíc navrženo tak, aby odlišovalo osoby s oprávněním potvrzení překladů a poskytovalo jim širší možnosti.

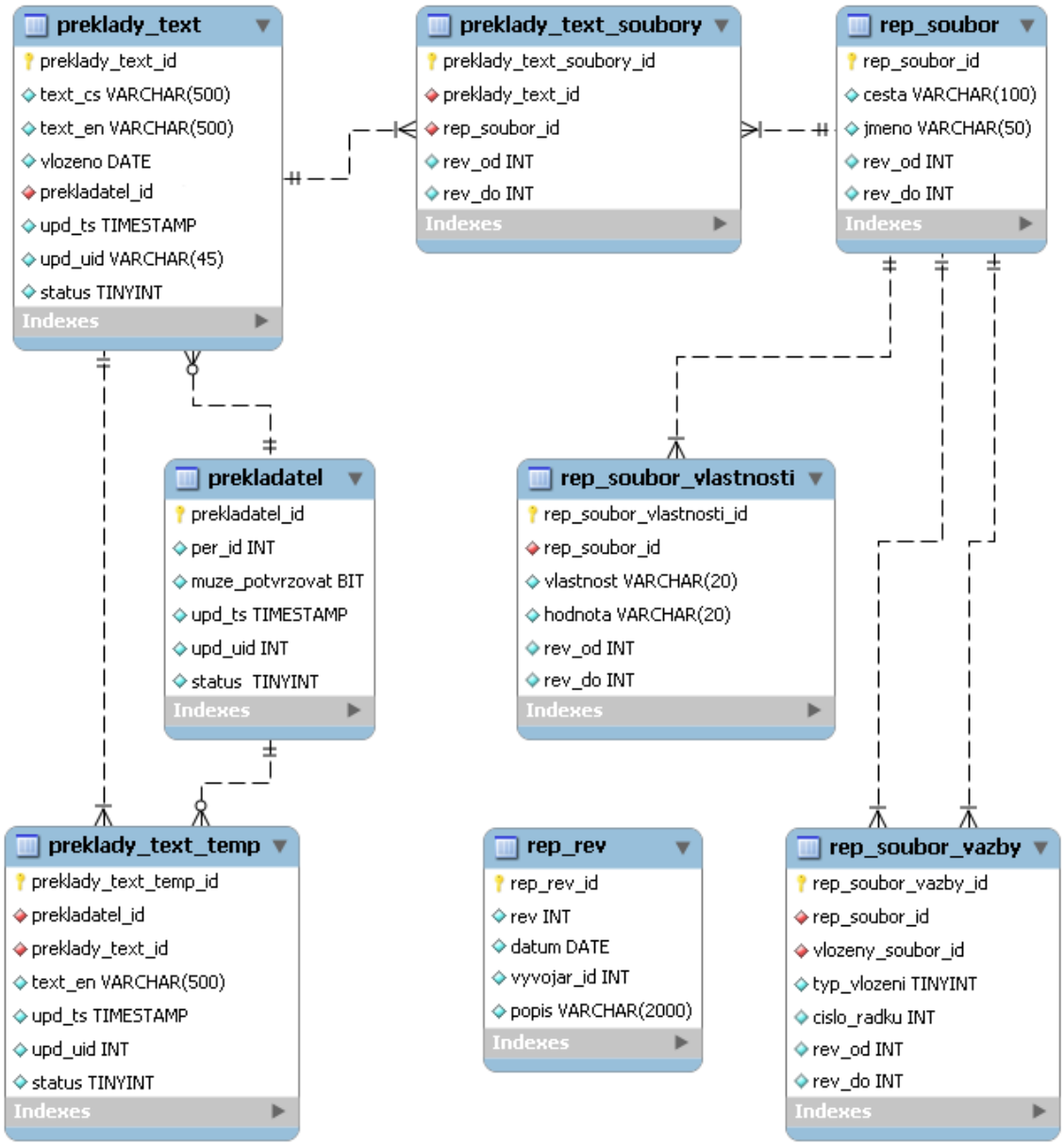
6.3.4 Logger

Tato část se stará o logování důležitých událostí (např. parsování, změny překladů, výjimky, varování atp.). Měla by zajišťovat ukládání informací do souboru, případně do DB. Slouží nejen ke kontrole správného chodu ostatních částí, ale v případě chyby do značné míry usnadňuje lokalizaci chyb a tím i jejich odstranění.

6.4 Návrh databáze

Pro uložení jazykových verzí se všemi dodatečnými informacemi bylo potřeba přidat do databáze několik tabulek. Při jejich návrhu byly zachovány doporučené konvence[14] (týká se především pojmenování tabulek a sloupců).

6.4.1 Entity Relationship Diagram



Obrázek č.5: Entity Relationship Diagram (ERD)

Obrázek č.5 ukazuje ER diagram, který slouží k modelování perzistentních dat a vztahů mezi nimi. Entitní množiny *preklady_text*, *preklady_text-soubory*, *preklady_text_temp* a *prekladatel* uchovávají informace o jazykových verzích, ostatní znázorněné se týkají především informací o jednotlivých revizích systému, které však s překlady úzce souvisí. Například sloupce *rev_od* a *rev_do* tabulky *preklady_text_soubory* obsahují rozmezí revizí, v nichž byl konkrétní text obsažen v určitém souboru, přičemž každá z těchto revizí je detailně popsána v tabulce *rep_rev*. Pokud sloupec *rev_do* v některé tabulce obsahuje hodnotu null, znamená to platnost informace po současnou revizi.

6.4.2 Detailní popis tabulek

Většina tabulek podle zavedených konvencí obsahuje následující 3 sloupce, proto jsou popsány nejprve samostatně:

upd_uid – osobní číslo uživatele, který záznam naposledy změnil
upd_ts – čas poslední změny
status – informace o platnosti dat. Může nabývat čtyř hodnot:

0 – záznam je platný, byl vytvořen synchronizací ze starého systému
9 – záznam je platný, jedná se o nově vytvořený záznam
1 – záznam je neplatný, z nějaké důvodu by měl ještě zůstat v DB
-1 – záznam je neplatný, určený k smazání

6.4.2.1 Tabulka preklady_text

Tato tabulka je pro systém překladů klíčová. Jsou zde uloženy všechny české texty (pokud už byly přeloženy tak také jejich jazykové verze). Kromě toho obsahuje také některé dodatečné informace jako datum vytvoření, čas poslední změny atd. Sloupec *prekladatel_id* může obsahovat hodnotu null, protože do této tabulky jsou záznamy vkládány při parsování zdrojových kódů, kdy ještě zdaleka neznáme osobu překladatele.

atribut	**typ**	**popis**
preklady_text_id	number(10)	primární klíč
text_cs	varchar2(500)	česká verze textu
text_en	varchar2(500)	anglická verze textu
vlozeno	date	datum vytvoření záznamu
prekladatel_id	number(10)	cizí klíč, id prekladatele
upd_uid, upd_ts, status		

6.4.2.2 Tabulka preklady_text_temp

Slouží k dočasnému uložení překladů. Veškeré cizojazyčné texty jsou nejprve uloženy sem a teprve po potvrzení překladatelky jsou přesunuty do tabulky *preklady_text*.

atribut	**typ**	**popis**
preklady_text_temp_id	number(10)	primární klíč
prekladatel_id	number(10)	cizí klíč, id prekladatele
preklady_text_id	number(10)	cizí klíč, id textu
text_en	varchar2(500)	anglická verze textu
upd_uid, upd_ts, status		

6.4.2.3 Tabulka prekladatel

Obsahuje seznam překladatelů a jejich práv. Sloupec *muze_potvrzovat* určuje, zda má přihlášená osoba právo k potvrzení překladu, čímž dojde k přesunu uloženého textu z tabulky *preklady_text_temp* do tabulky *preklady_text*.

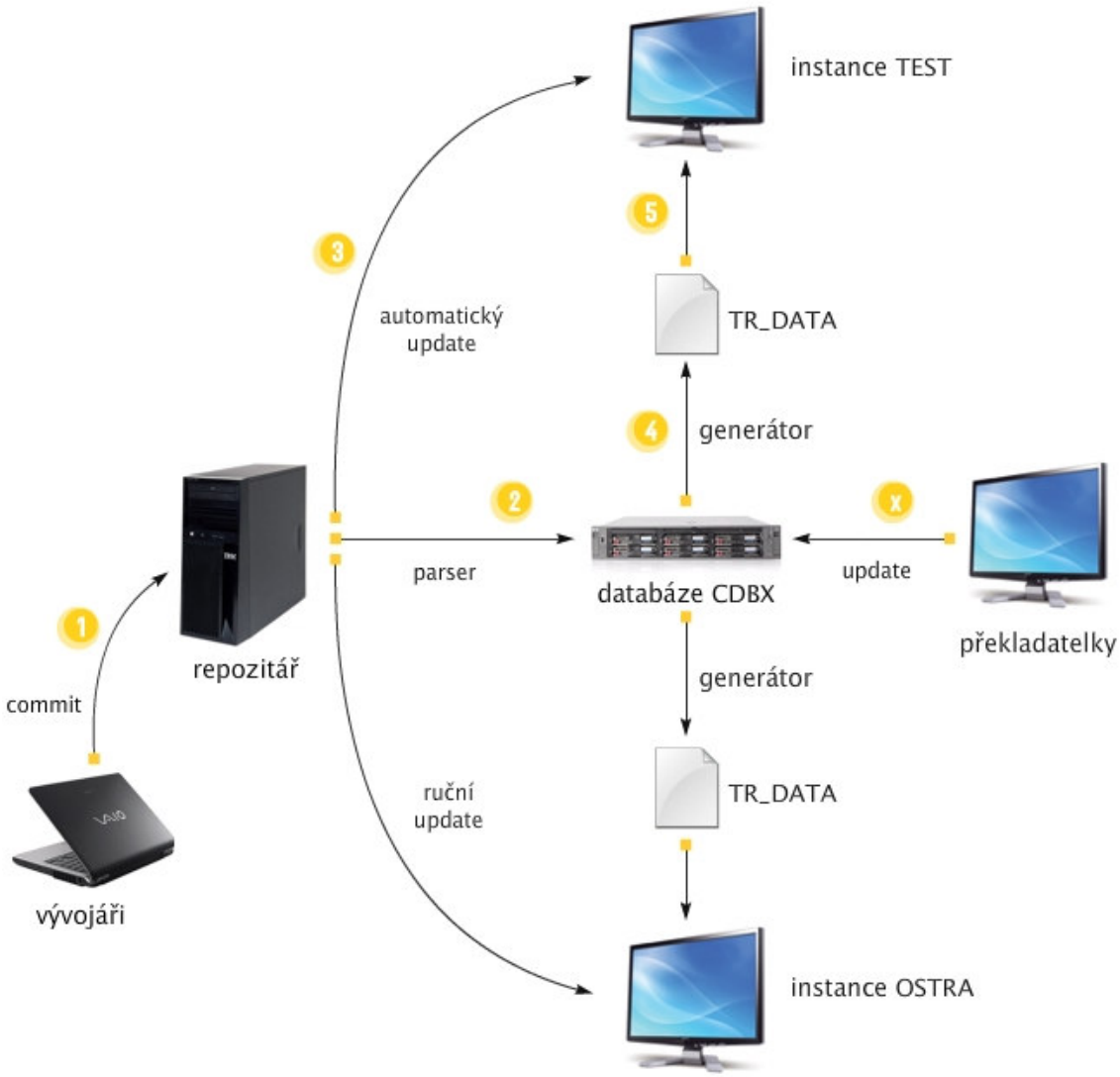
atribut	**typ**	**popis**
prekladatel_id	number(10)	primární klíč
per_id	number(10)	cizí klíč do brutisadm.person
muze_potvrzovat	number(1)	oprávnění k potvrzení
upd_uid, upd_ts, status		

6.4.2.4 Tabulka preklady_text_soubory

Tato tabulka slouží k uchování vazeb textů na soubory v repozitáři. Můžeme snadno zjistit, jaké texty a v jaké revizi daný soubor obsahuje či například všechny texty v aktuální revizi. Protože pomocí ostatních tabulek v databázi můžeme například zjistit, které soubory obsahuje daná aplikace, umožňuje nám to generovat soubory s překlady pro konkrétní instanci systému či pouze pro jeho část.

atribut	**typ**	**popis**
preklady_text_soubory_id	number(10)	primární klíč
preklady_text_id	number(10)	cizí klíč, id textu
rep_soubor_id	number(10)	cizí klíč, id souboru
rev_od	number(10)	platný od revize
rev_do	number(10)	platný po revizi (včetně)

6.5 Celkový popis řešení jazykových verzí



Obrázek č.6: Diagram návrhu řešení

Celý proces automatizace překladů tedy bude fungovat takto: Vývojáři mění lokálně (na svém počítači) zdrojové kódy. Příkazem *svn commit*(1) dojde k přenesení změn do centrálního repozitáře a zvýšení revize. Pomocí tzv. háčku je spuštěn parser(2), který ve změněných souborech dané revize vyhledá texty určené k překladu a aktualizuje podle nich databázi. Po dokončení činnosti parseru se spustí automatický update(3), který z repozitáře aktualizuje instanci *test*. Jakmile je update ukončen, aktivuje se generátor, který pro danou revizi vytvoří několik souborů s překlady pro vybrané části systému (4). Tyto soubory pak instance *test* používá(5) pro naplnění globální proměnné *$tr_mem* a

zobrazení překladů. Body 3, 4, 5 probíhají podobně i pro instanci *ostra* s tím rozdílem, že jsou spouštěny pouze po aktualizaci této instance (nikoliv automaticky po každém commitu) a zároveň není potřeba parsovat změněné soubory. Instance *ostra* má totiž vždy nižší číslo revize, je tedy vývojově „opožděná“ a veškeré změny již jsou v databázi zaneseny pomocí tzv. post-commit háčku. Po ručním update je tedy pouze potřeba vygenerovat příslušné překladové soubory pro novější verzi instance *ostra*.

Překladatelé (a částečně i vývojáři) překládají pomocí administračního rozhraní texty (x). Jazykové verze se nejprve ukládají do pomocné tabulky a teprve po potvrzení, které mohou provést pouze oprávnění uživatelé, jsou přeneseny do tabulky překladů.

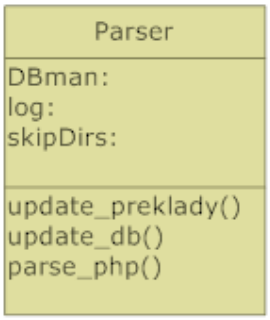
7 Implementace

Tato kapitola se zaměřuje na konkrétní popis jednotlivých částí navrženého systému pro automatizovaný překlad. V závěru jsou také popsány některé komplikace, které bylo nutné vyřešit během vývoje.

7.1 Parser

Protože PHP 5 umožňuje pokročilé metody práce s objekty, byla tato část implementována jako třída *Parser* a umístěna v souboru *class_parser.php*. Jedná se vlastně o syntaktický analyzátor, který prohledává zdrojové kódy a hledá v nich výskyt překladových funkcí. Protože různorodost těchto funkcí a především nesrovnalosti s použitím klíčů způsobovaly značné problémy, byla po konzultaci odsouhlasena změna jejich použití. Jedná se především o odstranění všech výskytů funkcí *get_lang()* a *insert_label()* a naopak použití funkce *tr()*. Tato změna by měla odstranit duplicitu klíčů a zjednodušit používání překladů. Nevýhodou tohoto řešení je, že klíčem pro vyhledávání je samotný text, který může dosahovat značné délky, avšak ve srovnání s nevýhodami použití klíčů se jedná o drobnost. Pro uživatele (programátory a překladatele) to znamená zjednodušení práce, navíc u odstraněných funkcí odpadá riziko s použitím nevhodných klíčů, neboť klíčem je samotný překládaný text.

Zpracování formátovaných textů probíhá stejně jako u předchozího řešení (viz kap. 5.1.2) s využitím funkce *sprintf()*.



Obrázek č.7: Třída Parser

7.1.1 Popis třídy

Proměnné:

DBMan – privátní proměnná obsahující objekt třídy DBManager (viz dále). Pomocí tohoto objektu je realizována veškerá činnost s databází. Alternativním řešením by bylo všechny funkce, které

parser využívá z DBManageru, přesunout do třídy Parser. Od toho však bylo upuštěno z důvodu pravděpodobné sdílené funkčnosti (např. práce nad stejnou oblastí dat).

log – privátní proměnná obsahující objekt třídy Logger. Tento objekt se používá k záznamu důležitých událostí během činnosti analyzátoru.

skipDirs – privátní proměnná. Jedná se o pole obsahující cesty k adresářům nebo souborům, které se mají při parsování přeskočit (např. adresáře, které obsahují pouze obrázky nebo cizí knihovny)

Metody:

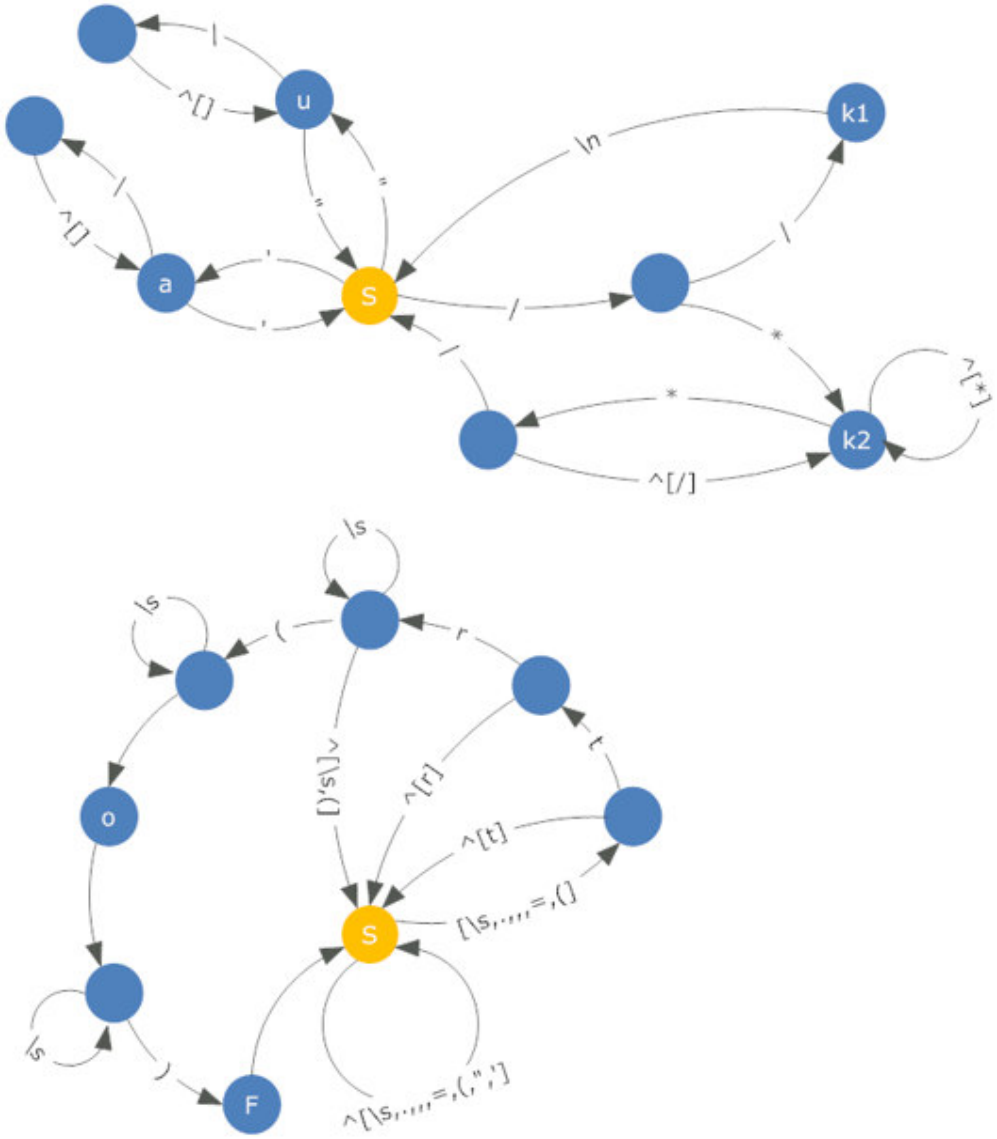
update_preklady() – veřejná metoda, která je spouštěcím mechanizmem pro syntaktickou analýzu. Při svém volání vyžaduje parametry název, umístění a obsah parsovaného souboru a také číslo revize. Hlavním úkolem této funkce je zjistit, zda má být soubor vůbec zpracován (zda není mezi zakázanými) a také zjistit jeho id, se kterým se dále pracuje. Pokud je nalezen v DB, je volána metoda update_db(), která se stará o samotné zpracování.

update_db() – jde o privátní metodu, která je volána s parametry obsah souboru, jeho id a číslo revize. Tato funkce provádí na základě obsahu souboru aktualizaci databáze. Nejprve tedy získá z DB seznam textů, které soubor obsahoval v předchozí verzi. Pak pomocí funkce parse_php() získá seznam textů v aktuální verzi. Na základě srovnání těchto dvou seznamů je možné zjistit, které překlady jsou nové či naopak které ze souboru zmizely.

parse_php() – tato privátní metoda tvoří jádro parseru. Obsahuje regulární výraz, díky němuž jsou ze zdrojových kódů extrahovány vhodné řetězce, které jsou dále zpracovány. Pro vyhledávání výrazu je použita PHP funkce *preg_match_all()* (u nadměrně velkých souborů může působit problémy).

7.1.2 Popis procesu

Proces zpracování jedné revize tedy probíhá takto: po příkazu *commit* je spuštěn skript, který pro každý změněný soubor v revizi volá parser. Změnou souboru je myšleno jeho přidání, smazání, přesun nebo změna obsahu. Pro každý soubor je tedy volána veřejná funkce *update_preklady()*, které jsou předány potřebné parametry. Voláním dalších funkcí dochází ke zpracování souboru, přičemž důležité události jsou ukládány do logu. Předpokladem úspěšného parsování je validní PHP kód. Obrázek č.8 ukazuje konečný automat ekvivalentní regulárnímu výrazu v parseru. Z uvedeného diagramu vyplývá, že syntaktický analyzátor funkci *tr()* nevyhledává uvnitř uvozovek, apostrofů a komentářů. Pro větší přehlednost je automat rozdělen na dvě část a zjednodušen.



Obrázek č.8: Konečný automat

Vysvětlivky:

S – počáteční stav

F – koncový stav (úspěšné nalezení funkce *tr()*)

o – obsah hledané funkce *tr()*

[x,y,z] – množina znaků obsahující x, y, z

^[x,y,z] – jakýkoliv znak kromě x, y, z

^[] – jakýkoliv znak

\s – bílý znak (mezera, tabulátor, atd.)

\n – konec řádku

7.2 Generátor

Také generátor byl implementován jako třída. Jeho cílem je pro každou aplikaci vytvořit soubor s překlady (PHP skript s globální proměnnou *$tr_mem*, která obsahuje pole jazykových verzí používaných v dané aplikaci).

Proces generování začíná voláním funkce *generate_app(rev_od, rev_do, cesta_k_souboru)*, kde význam parametrů je postupně určení rozsahu prohledávaných revizí a cesta k souboru s překlady, který teprve bude vytvořen. Aplikace je určována podle prvního adresáře z cesty. Samotné získání textů je poměrně jednoduchý proces. Nejprve probíhá výběr souborů relevantních pro danou aplikaci a rozsah revizí. Tento výběr se provádí na základě adresářové struktury (např. všechny soubory aplikace *Studis* lze najít v adresáři s*tudis*). Máme-li k dispozici takovýto seznam souborů, je již snadné z tabulek *preklady_text_soubory* a *preklady_text* vybrat ke každému souboru příslušné texty. Oba výběry musí probíhat s ohledem na požadovaný rozsah revizí a pracovat pouze s platnými daty.

Tímto způsobem vznikne pro každou aplikaci jeden překladový soubor a jeden pro společnou část *_base*. Důsledkem rozdělení je pak vyšší rychlost zpracování požadavků. Zvolené řešení předpokládá, že v každé aplikaci jsou používány pouze soubory v adresáři aplikace nebo ve společném adresáři *_base*.

Proměnné:

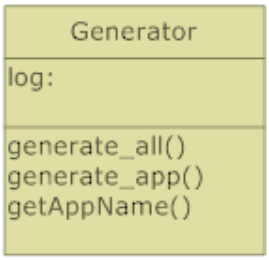
log – obsahuje objekt třídy Logger, který slouží k zaznamenání důležitých událostí během procesu generování

Metody:

generate_app() – veřejná metoda sloužící ke generování souboru s překlady pro jednu aplikaci

generate_all() – metoda se v současnosti nevyužívá, je možno použít pro vytvoření jediného souboru s překlady obsahujícího všechny texty v daném rozsahu revizí

getAppName() – veřejná metoda, je určena k získání názvu aplikace podle adresářové struktury (cesty k souboru)



Obrázek č.9: Třída Generátor

7.3 DBManager

Tato třída obsahuje metody pro práci s databází, které jsou využívány třídami *Parser* a *Generator* (mohou je využít i jiné třídy). Alternativním řešením by bylo umístit metody přímo do jednotlivých tříd, avšak tento způsob umožňuje přehledný přístup k databázi na jednom místě.

Relační data jsou získávána podle doporučených konvencí s pomocí třídy *ClassOracle* a knihovny ADOdb (viz kapitola 4.4.3). Jednotlivé metody není nutné detailně popisovat.

7.4 Logger

Tato třída byla implementována na základě požadavku o zaznamenání změn při vytváření a administraci jazykových verzí a generování překladových souborů. Její hlavní výhoda spočívá v abstrakci a sjednocení práce s logováním, není tedy nutné tyto metody vytvářet v ostatních třídách.

Proměnné:

logFile – privátní proměnná obsahující název souboru, do něhož se budou ukládat zprávy

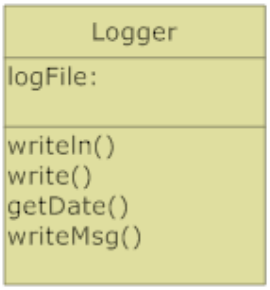
Metody:

writeln() – veřejná metoda zajišťující zápis zprávy ve tvaru [datum a čas] [zpráva] na nový řádek

write() – veřejná metoda, která zapisuje pouze text zprávy bez odřádkování

getDate() – veřejná metoda, která vrací formátovaný datum a čas

writeMsg() – privátní metoda která je využívána metodami write() a writeln(), zajišťuje fyzický zápis do souboru



Obrázek č.10: Třída Logger

7.5 Administrační rozhraní

Aplikace pro překladatele byla na rozdíl od ostatních objektově-orientovaných částí implementována pomocí klasického funkcionálního programování. Jedná se o modul webové aplikace *Portal*, jehož úkolem je především usnadnění práce překladatelům a vývojářům.

7.5.1 Popis jednotlivých souborů

Modul se skládá ze 3 souborů, které jsou umístěny ve složce *WWW/Portal3/app/p_preklady/* a podílí se na zobrazení jediné stránky (v podstatě se jedná o návrhový vzor Model-View-Controller). Toto rozdělení reflektuje snahu o oddělení funkční a nefunkční (statické) části skriptu.

Preklady.php – základní soubor, do něhož jsou vloženy ostatní dva. Stará se o získání dat a uživatelských vstupů, obsahuje většinu programového kódu. Kromě zpracování vstupních/výstupních informací také ověřuje práva uživatelů k zobrazení modulu a potvrzení překladů.
Preklady_templ.php – soubor se stará především o zobrazení výsledků, z velké části obsahuje statický HTML kód.
Preklady_func.php – soubor slouží jako knihovna funkcí, především pro manipulaci s databází. Funkce nebyly zařazeny do třídy DBManager z důvodu zachování konvencí.

7.5.2 Popis prostředí

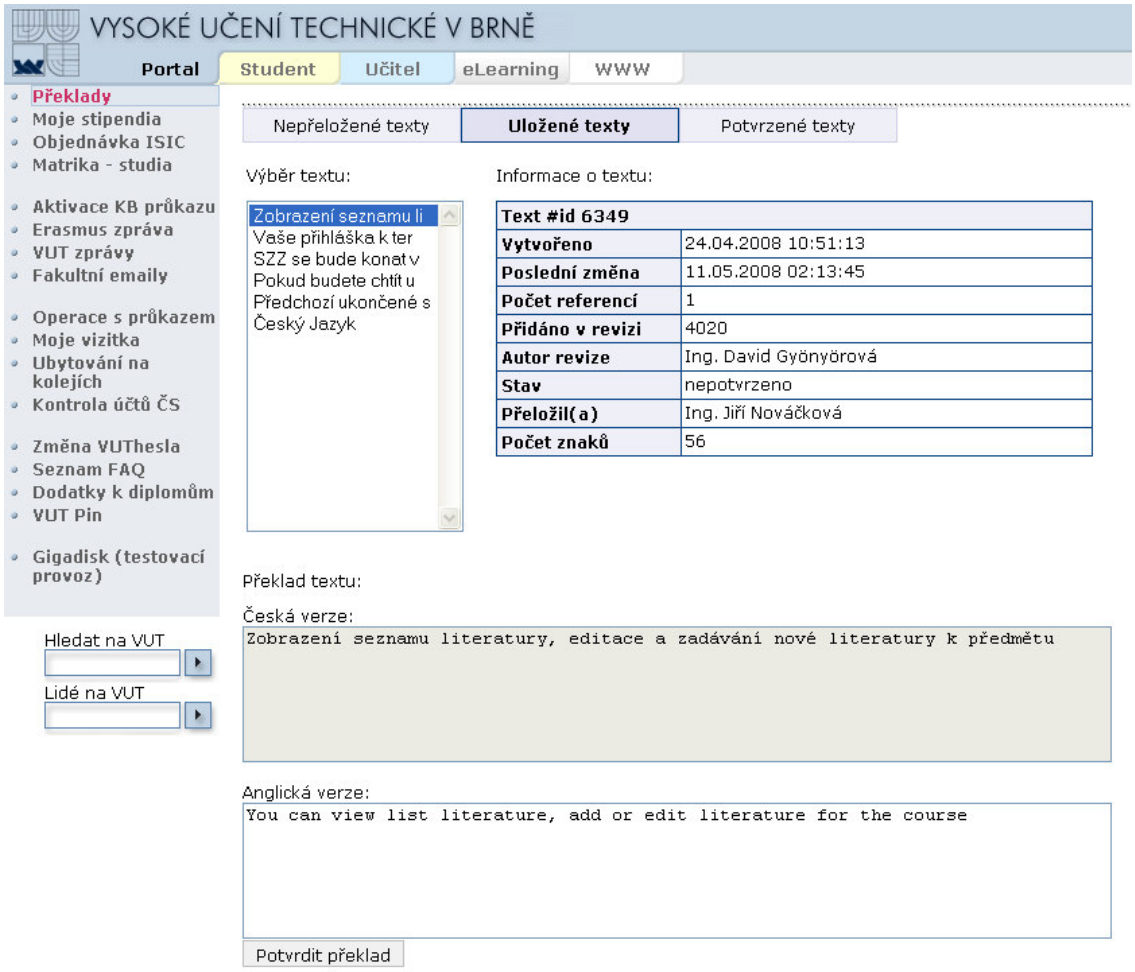
Jak vyplývá z výše popsaného, modul byl integrován do stávajícího informačního systému. Po vyplnění uživatelského jména a hesla na centrální přihlašovací stránce je uživatel přesměrován na úvodní stránku systému. Pokud má daný uživatel právo na zobrazení tohoto modulu, objeví se v záložce *Portal* odkaz *Překlady* (viz obrázek č.11).

Po otevření modulu *Překlady* se zobrazí stránka obsahující 3 záložky („Nepřeložené texty“, „Uložené texty“, „Potvrzené texty“). To odpovídá situaci, že se každý text určený k překladu může nacházet ve třech stavech: nepřeložený, uložený a potvrzený. Stav není v databázi přímo uchováván, je však možné jej určit podle tabulek *preklady_text* a *preklady_text_temp*.

V každé záložce se tedy objeví seznam textů daného typu, přičemž je možné si prohlížet jejich jazykové verze a další podrobné informace či přejít do následujícího stavu.

Zobrazení některých detailních informací o textu může být značně problematické (každý text se může vyskytovat v několika souborech a zároveň jednotlivé změny v každém souboru mohou dělat různí vývojáři).

Správa textů na úrovni přidávání a mazání jednotlivých textů není možná. Při vkládání nového textu by chyběla vazba na soubor, odstranění textu je taktéž nemožné kvůli zachování starších revizí.



Obrázek č.11: Aplikace pro překladatele

Vysvětlení zobrazovaných informací:

Vytvořeno – časový údaj o prvním výskytu textu ve zdrojových kódech během vývoje (čas vložení do DB pomocí parseru po akci *commit*)

Poslední změna – časový údaj o poslední změně daného textu (pro uložené texty se týká změny v tabulce *preklady_text_temp*)

Počet referencí – počet souborů, které někdy během vývoje obsahovaly daný text. Jedná se o orientační údaj popisující četnost využití daného textu

Přidáno v revizi – číslo revize, kdy se text naposledy nově objevil v nějakém souboru

Autor revize – vývojář, který provedl *commit* výše popsané revize, autor české verze textu

Stav – stav textu (nepřeložený, uložený, potvrzený)

Přeložil – autor překladu

Počet znaků – obsahuje počet znaků anglické verze (bez mezer)

8 Integrace a testování

Systém pro práci s jazykovými verzemi byl úspěšně integrován do IS VUT a nyní se nachází ve fázi testování. Před samotným nasazením do ostrého provozu je potřeba provést některé změny současného systému. Kromě toho by bylo vhodné doplnit do databáze texty z dosavadních překladových souborů, aby se zabránilo zbytečnému překladu již přeložených textů.

V rámci vývoje a testování jsme se setkali s těmito problémy:

- logování – problém s nastavením oprávnění, nedovolený zápis do souboru. Řešení: zapisovat pouze do souboru, který je umístěn v adresáři určeném pro logování
- parser – během testování byla zjištěna chyba (PHP bug[17]). Při použití funkce *preg_match_all()* u nadměrně velkých souborů může dojít k výpadku, čímž je přerušen proces parsování. Řešení: neparsovat příliš velké soubory či využít parseru jazyka PHP.
- parser – problémy se správným uložením národních a speciálních znaků. Řešení: správná konfigurace serveru a databáze a využití PHP funkce *addslashes()*
- parser – problémy s odlišným zpracováním textů v uvozovkách a apostrofech. Řešení: simulace vyhodnocení řetězců
- duplicita klíčů – ruční vytváření klíčů v sobě skrývalo hrozbu duplicity. Řešení: automatizací překladů je toto riziko odstraněno (nemohou existovat dva různé texty se stejným klíčem)
- přizpůsobení současného systému novým požadavkům vyžaduje některé výrazné úpravy, jako např. odstranění všech výskytů funkcí *insert_label()* a *get_lang()* a jejich náhrada pomocí funkce *tr()*. Všechny výrazné úpravy byly diskutovány a měl by je provést některý z vývojářů IS VUT
- databáze – každý text by se měl překládat pouze 1x, proto budou všechny texty uloženy v ostré databázi CDBX. To ovšem způsobuje možnost šíření chybného překladu do reálného provozu IS. Řešení: z tohoto důvodu má proces překladu několik stavů a text se může objevit v ostré verzi až po finálním potvrzení překladatele.
- generátor – vytvoření překladového souboru pro každou aplikaci s sebou přináší problém vložených souborů. U konkrétní aplikace sice známe soubory, které do ní patří, avšak každý z nich může obsahovat několik vložených souborů, které není snadné pokrýt. Řešení: většina aplikací splňuje předpoklad samostatnosti, tzn. nevyužívá soubory jiné aplikace kromě části *_base*, která obsahuje sdílené knihovny. IS VUT lze upravit tak, aby všechny aplikace byly samostatné. Pak stačí vygenerovat jeden soubor pro část *_base* a jeden pro konkrétní aplikaci.

9 Možnosti rozšíření

Tato kapitola popisuje možnosti budoucího rozšíření či vylepšení jednotlivých komponent.

9.1 Parser

Budoucí rozšíření parseru spočívá především v lepší syntaktické analýze. Elegantním řešením by bylo použít PHP parser, který umožňuje transformovat zdrojový kód na seznam tokenů[18]. Tímto způsobem by se vyřešili problémy s velikostí souborů, zpracováním komentářů, uvozovek, atd.

```
$token = token_get_all('<?php echo; ?>');
```

Výsledkem uvedeného kódu je pole:

```
array(
      array(T_OPEN_TAG, '<?php'),
      array(T_ECHO, 'echo'),
      ';',
      array(T_CLOSE_TAG, '?>')
);
```

Kód č. 6: Příklad odlišné syntaktické analýzy

9.2 Generátor

Vytváření souborů s překlady pro každou aplikaci se vždy odvíjí od aktuálního seznamu souborů dané aplikace. V navrženém systému je seznam získán prostým porovnáním adresářové struktury (např. soubory aplikace Studis jsou v adresáři Studis). Protože se však v některých oblastech systému může programová struktura od té adresářové lišit, bylo by správnější provádět generování jiným způsobem.

Jedním z možných způsobů může být získání seznamu souborů příslušných k určité aplikaci (databáze tyto informace obsahuje) a zároveň ke každému souboru zjistit soubory na něm závislé (vložené pomocí include, require atd.). Z těchto částečných seznamů poté vytvořit jediný seznam obsahující všechny soubory týkající se dané aplikace. Ačkoliv je tento způsob mnohem komplikovanější, z pohledu logiky se zdá být správnější.

Alternativou k tomuto řešení je ponechat rozdělování souborů podle adresářů, avšak v tom případě je potřeba systém přizpůsobit tak, aby každá aplikace byla samostatná (využívala pouze soubory vlastní nebo ze sdílených knihoven).

Důležitým prvkem možného vylepšení je také četnost vytváření souborů. Generátor vytváří jeden soubor pro jednu aplikaci. Je vhodné, aby skript využívající služeb generátoru rozhodoval o generování pouze v opodstatněných případech (např. pokud se změní jeden soubor v aplikaci *Teacher*, není potřeba vytvářet překladové soubory pro všechny aplikace).

Jako další oblast poskytující prostor budoucím změnách bych zmínil formát vytvořených souborů. Z důvodu kompatibility zůstal zachován původní formát *$tr_mem[klíč] = [text_cs, text_en]*. Protože však nový návrh počítá s tím, že klíčem je přímo český text (čímž se použití značně zjednodušuje), není potřeba jej používat zároveň jako klíč i hodnotu. Drobná změna některých funkcí umožní generovat soubory podle nového formátu *$tr_mem[text_cs] = [text_en]*.

9.3 Administrační rozhraní

Aplikace pro překladatele nabízí základní funkčnost pro práci s překlady. V rámci budoucích rozšíření je vhodné ještě více usnadnit práci překladatelů a vývojářů. Jedná se především o lepší správu textů, detailnější informace (např. zobrazení všech souborů, revizí a autorů, v nichž se daný text vyskytuje).

Dalším nedostatkem implementovaného systému a příležitostí k vylepšení je počítadlo znaků. Aplikace sice ukazuje počet znaků každého překladu, nezaznamenává však celkový počet pro daného překladatele. Je sice otázkou, do jaké míry je vhodné tuto informaci perzistentně uchovávat či aktuálně počítat, avšak určitě by z hlediska překladatele bylo vhodné zobrazovat nějaké souhrnné či statistické informace o přeložených textech.

Přestože v současném návrhu překladatel zná identitu autora textu, aplikace prozatím nepodporuje jejich přímou komunikaci. Dalším možným vylepšením tedy může být komunikace emailem nebo pomocí tzv. VUT zpráv (pro tento účel vhodnější).

Pro řešení případných problémů a konzultaci s autorem doporučuji rozšířit aplikaci také o vyhledávání textů (např. podle stavu, české nebo anglické verze, autora, souboru, modulu, revize atd.). Aktuální implementace poskytuje opravdu základní možnosti a texty jsou roztříděny pouze podle stavu. Při větším počtu se tak může stát výběr nepřehledným a vyhledávání vhodným prostředkem k úpravě překladů.

Dalším námětem pro budoucí vývoj může být také zobrazení historie změn překladů. Pokud by bylo nutné text později upravovat, vývojář i překladatel by mohl mít přehled o předchozích verzích textu. Kromě toho by mohlo být užitečné rozšířit dodatečné informace o možnost poznámek k jednotlivým textům.

V rámci tohoto modulu je seznam překladů získáván z databáze při každém zobrazení stránky. Ačkoliv je doba načítání stránky srovnatelná se zobrazením ostatních modulů, bylo by vhodné

získávat data z DB pouze v případě nutnosti, v ostatních případech je uchovávat v session, což může mít pozitivní vliv na dobu potřebnou k zobrazení stránky a snížení zátěže DB serveru.

Mezi možné oblasti rozšíření bychom mohli zařadit také využití slovníků. Propojením s některou webovou aplikací specializovanou na překlad by bylo možné proces vytváření jazykových verzí ještě více zjednodušit. Překladatelům by se pak zobrazovala nabídka automatického překladu a jejich úlohou by bylo tyto texty pouze kontrolovat či doplňovat. Minimálně propojení se slovníkem se zdá být vhodným směrem dalšího vývoje.

Na závěr bychom měli zmínit také lepší ošetření událostí. Jedná se například o situace, kdy překladatel upravuje jazykové verze a zároveň je potřeba generovat nový překladový soubor. To znamená, že pokud je upravovaný text obsažen v některé z nasazených instancí systému, je potřeba po úpravě vygenerovat nový překladový soubor. Při návrhu aplikace byly tyto problémy uváženy, avšak samotná obsluha události nebyla dosud implementována a je ponechána pro případná rozšíření.

10 Zhodnocení

10.1 Splnění jednotlivých požadavků

Podle zadání byl navržen a implementován systém pro správu překladů. Tento systém byl dále integrován do IS VUT, nyní se nachází ve stavu testování. Plnému nasazení do ostrého provozu brání především nutné úpravy IS VUT, jejichž cílem je příprava na nasazení automatizace překladů (např. zmiňované osamostatnění jednotlivých aplikací či odstranění zastaralých funkcí). Tyto úpravy může provést pouze programátor IS VUT.

Přestože navržené řešení není optimální a poskytuje mnoho námětů pro pozdější rozpracování, ve většině případů splňuje definované požadavky vývojářů i překladatelů. Jedná se především o:

- automatizaci překladů a vyhledávání textů – zajišťuje třída Parser
- administrační rozhraní pro správu překladů – vytvořen modul pro překladatele
- logování změn – zajišťuje třída Logger
- řešení vzrůstající velikosti překladového souboru – třída Generator vytváří soubor pro každou aplikaci zvlášť
- verzování systému – všechny vytvořené komponenty jsou schopné pracovat s revizemi
- minimalizaci duplicity překladů – navržený systém vylučuje duplicitu
- dodatečné informace – překladatel i autor textu mají dostatek informací a vzájemně znají svou identitu. Přímá možnost komunikace nebyla implementována a je otázkou budoucího rozšíření.
- optimalizaci výkonu - je zajištěna např. rozdělením překladových souborů
- bezpečnost – z uživatelského hlediska je systém chráněn pomocí centrálního přihlašování a ověřování práv. Ošetření rizikových událostí závisí především na nadřazených skriptech, které využívají parser, generátor atd.
- počítadlo přeložených textů – funguje pouze částečně, jeho dokončení a výpočet odměn jsou předmětem dalšího vývoje
- možnost přidání dalších jazyků – nebylo přímo implementováno, avšak navržený systém to umožňuje (v blízké budoucnosti se nepočítá s rozšířením o další jazyky)

10.2 Srovnání v kontextu podobných témat

Tato diplomová práce částečně navazuje na práci Ing. Milana Pavlíčka (Systém pro správu jazykových verzí VUT[19]). Oba texty se věnují stejné problematice, základní myšlenka inovace je velmi podobná, avšak zcela se liší ve způsobu zpracování.

Předchozí práce obsahuje několik závažných nedostatků, kvůli nimž bylo požadováno vytvoření nového návrhu. Jedná se především o to, že předcházející řešení nerespektuje správu verzí, což do značné míry ovlivňuje celý způsob řešení problému. Pokud by došlo k integraci a nasazení zmíněného návrhu do provozu, snadno by mohlo dojít k situaci, kdy by vygenerované texty sice odpovídaly jedné instanci (např. vývojové), pro jinou instanci (např. ostrou) by však texty byly nekompletní.

Mezi další odlišnosti je možné zařadit také použití klíčů, rozdílná implementace jednotlivých komponent, způsob integrace apod.

11 Závěr

Cílem diplomové práce bylo navrhnout systém pro správu jazykových verzí IS VUT a inovovat tak nevyhovující současné řešení. Ve shodě se zadáním tedy byly zformulovány jak obecné požadavky na vícejazyčný systém, tak i konkrétní požadavky vývojářů a překladatelů, které reflektují nevýhody stávajícího řešení. S ohledem na tyto požadavky byla navržena a implementována aplikace, která proces překladů do značné míry automatizuje. V současné době je aplikace ve fázi testování a připravuje se její nasazení do reálného provozu.

Během vývoje jsem se detailně seznámil nejen s informačním systémem, především s částí Portal, ale s i prostředky používanými pro jeho vývoj. Návrh a implementace byly prováděny v kontextu odborných konzultací i zkušeností z dřívějších, převážně komerčních projektů.

Samotná práce jako celek navazuje na návrh Ing. Milana Pavlíčka, obojí vychází ze stejného teoretického základu, avšak diametrálně se odlišují zpracováním.

Tento projekt nabízí významný posun v otázce jazykových verzí IS VUT. Z hlediska vývojářů a překladatelů dochází k značnému zjednodušení a zefektivnění práce s překlady. Možnosti rozšíření, popsané v kapitole 9, naznačují směr dalšího vývoje tohoto systému.

12 Literatura

[1] *Wikipedie: Otevřená encyklopedie: Unicode* [online]. c2007 [cit. 2007-12-27]. Dostupný z WWW: <http://cs.wikipedia.org/w/index.php?title=Unicode&oldid=2106087>

[2] BOJAR, Ondřej, et al. *Unicode - cesta z chaosu kódování znaků* [online]. [cit. 2008-04-12]. Dostupný z WWW: <http://www.cestina.cz/kodovani/unicode.html>.

[3] JELÍNEK, Lukáš, et al. *Manuál PHP* [online]. 2007 [cit. 2007-12-23]. Dostupný z WWW: <http://cz.php.net/manual/cs/>

[4] *Wikipedie: Otevřená encyklopedie: Model-view-controller* [online]. c2008 [cit. 2008-04-27]. Dostupný z WWW: <http://cs.wikipedia.org/w/index.php?title=Model-view-controller &oldid=2506429>.

[5] *Objektově orientované programování (OOP) v PHP* [online]. 1999 [cit. 2008-03-19]. Dostupný z WWW: <http://php.interval.cz/objektove-orientovane-programovani-oop-v-php/>.

[6] *Subversion* [online]. 2006 [cit. 2007-12-20]. Dostupný z WWW: <http://subversion.tigris.org/>.

[7] *TortoiseSVN* [online]. c2006 [cit. 2008-04-15]. Dostupný z WWW: <http://tortoisesvn.tigris.org/>.

[8] VAŠÍČEK, Zdeněk. *Stručný úvod do SVN* [online]. 2006 [cit. 2007-12-20]. Dostupný z WWW: <http://merlin.fit.vutbr.cz/FITkit/?pg=navody&cl=20060209svn>.

[9] *Centrum výpočetních a informačních služeb* [online]. 2006 [cit. 2008-04-08]. Dostupný z WWW: <http://www.vutbr.cz/index.php?page=is&wapp=webcis&parent=1 &tail=1?=0>.

[10] *CVIS Wiki:Instance web aplikací* [online]. 2007 [cit. 2007-12-29]. Dostupný z WWW: <https://ent.ro.vutbr.cz/ostra/cvis/mediawiki/index.php? title=Instance_web_aplikac%C3%AD&oldid=4102>.

[11] *Databáze Oracle* [online]. 2005 [cit. 2007-12-20]. Dostupný z WWW: <http://www.oracle.com/global/cz/database/index.html>.

[12] *CVIS Wiki:Zápisy FEEC 2007 - technická zpráva* [online]. 2007 [cit. 2008-04-16]. Dostupný z WWW: < https://wiki.ro.vutbr.cz/w/index.php?title=Z%C3 %A1pisy_FEEC_2007_-_technick%C3%A1_zpr%C3%A1va&oldid=5543>.

[13] *CVIS Wiki: Aktualizace CISD* [online]. 2007 [cit. 2008-04-15]. Dostupný z WWW: <https://wiki.ro.vutbr.cz/w/index.php?title=Aktualizace_CISD &oldid=3566>.

[14] *CVIS Wiki: Konvence pojmenování DB objektů* [online]. 2007 [cit. 2008-04-18]. Dostupný z WWW: <https://wiki.ro.vutbr.cz/w/index.php?title=Konvence_pojmenov %C3%A1n%C3%AD_DB_objekt%C5%AF&oldid=2734>.

[15] LIM, John. *ADOdb Database Abstraction Library for PHP* [online]. c2000 [cit. 2008-04-13]. Dostupný z WWW: <http://adodb.sourceforge.net/>.

[16] PONKRÁC, Miloslav. *PHP – 49. díl – formátování čísel a řetězců* [online]. 2005 [cit. 2008-04-19]. Dostupný z WWW: <http://www.zive.cz/PHPPerl/PHP--49-dil--formatovani-cisel-a-retezcu/sc-71-sr-1-a-122164/default.aspx>.

[17] *PHP Bugs: Segmentation Fault with preg_match_all* [online]. 2001 [cit. 2008-04-18]. Dostupný z WWW: <http://bugs.php.net/bug.php?id=40909>.

[18] *List of Parser Tokens* [online]. c2001 [cit. 2008-04-21]. Dostupný z WWW: <http://cz2.php.net/manual/en/tokens.php>.

[19] PAVLÍČEK, Milan. *Systém pro správu jazykových verzí portálu VUT*. [s.l.], 2007. 58 s. FIT VUTBR. Vedoucí diplomové práce Jaromír Marušinec. Dostupný z WWW: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=5755>.

[20] *CVIS Wiki: Coding standards* [online]. 2007 [cit. 2008-04-18]. Dostupný z WWW: <https://wiki.ro.vutbr.cz/w/index.php?title=Coding_standards &oldid=6782>.