VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

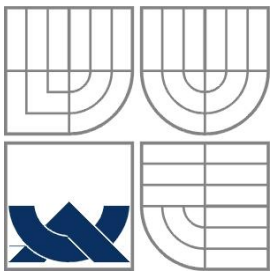
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SERVEROVÁ ČÁST SYSTÉMU PRO SPRÁVU PROJEKTOVÉ DOKUMENTACE

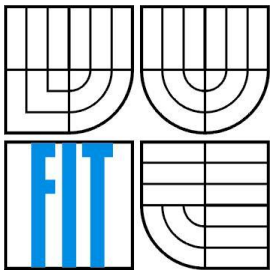
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. RADEK ČERNOBILA
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

SERVEROVÁ ČÁST SYSTÉMU PRO SPRÁVU PROJEKTOVÉ DOKUMENTACE

SERVER PART OF THE PROJECT DOCUMENTATION MANAGEMENT SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Radek Černobila
AUTHOR

VEDOUCÍ PRÁCE — Ing. Zbyněk Křivka, Ph.D.
SUPERVISOR

BRNO 2008

Abstrakt

Cílem této práce je návrh a implementace revizního systému, schopného spravovat celý vývojový proces projektu. Ukázat alternativy implementace jeho jednotlivých částí. Důraz je kladen zejména na perzistentní uložení dat a architekturu systému. Výsledkem by měl být program schopný prezentovat hlavní řešené problémy, jako je větvení projektových větví a distributivní datového úložiště.

Klíčová slova

Repositář, verze, distribuovaný souborový systém, revize, databáze, regulární gramatika, stromová struktura, kooperace

Abstract

The goal of this work is proposal and implementation of the revision system that can control whole development process of a project. It shows alternatives of the implementation of its parts. The main is a persistent data storage and the system architecture. The expected result is a application that is able to present basic solved problems as are project branching and distributed data storage.

Keywords

Repository, version, distributed file system, revision, database, regular grammer, tree structure, cooperation

Citace

ČERNOBILA Radek: *Serverová část systému pro správu projektové dokumentace*. Brno, 2007-2008, diplomová práce, FIT VUT v Brně.

Serverová část systému pro správu projektové dokumentace

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Zbyňka Křivky, Ph.D.
Další informace mi poskytl doc. Ing. Jaroslav Zendulka, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Radek Černobila
15. 5. 2008

Poděkování

Tímto děkuji svému vedoucímu za kvalitní koordinaci práce, která zdárně směřovala k vytyčenému cíli. V neposlední řadě pak Ondřeji Býmovi za skvělou spolupráci při návrhu vlastního systému.

Obsah

1 Úvod

V softwarovém i jiném průmyslu se vedle konkrétních řešených problémů spadajících do daného oboru musí spravovat i dokumentace a jiné elektronické materiály. Právě správa projektové dokumentace je tím problémem, který tvoří průnik nad různě zaměřenými projekty. Všechny tyto projekty mají společné to, že dokumentace je vytvářena postupně, v celém vývojovém procesu projektu a během svého vytváření je často modifikována. Jelikož projektová dokumentace je velmi důležitým projektovým prvkem, jsou na její správu a bezpečné uložení kladeny vysoké nároky. Zvláště v softwarovém a multimediálním průmyslu může dokumentace a výsledky práce dosahovat velkých objemů dat, která je nutné velmi efektivně spravovat a udržovat v přehledné formě.

Často se pro tyto systémy používá označení verzovací nebo též revizní systémy. Jejich primárním úkolem je zajistit bezpečné uložení všech projektových elektronických dokumentů, ale i vytvářet chronologický přehled nad jejich vývojem. Nedílnou součástí je i podpora týmové spolupráce, která se odráží ve víceuživatelském přístupu a dále pak podpora strukturovanosti vývojového týmu.

Tato práce ve své teoretické části shrnuje možné alternativy vývoje a architektury takového systému. Popisuje celý vývoj revizního systému, od počáteční myšlenky až po koncové zhotovení. Práce popisuje zásadní problémy, se kterými se musí systém pro správu revizí vyrovnat, navrhuje jejich možné řešení, s ohledem na již existující systémy a snaží se o vytvoření průniku mezi různými přístupy ke stejné problematice. Nedílnou součástí práce je i podrobné navržení systému pro správu revizí a jeho implementace. Tato část se zaměřuje zejména na použité algoritmy, případně abstraktní datové typy, které jsou k vyhotovení systému použity. Hlavně v oblasti nejdůležitější pro tyto systémy, kterou je perzistentní uložení dat, která jsou systémem spravována.

2 Obecný pohled

Systémy pro správu revizí jsou systémy, které umožňují spravování celého životního cyklu fyzických souborů s možností přístupu k jejich historickým variantám. Tento požadavek je stěžejní pro takové systémy a je alfou a omegou při jejich implementaci. Pohled na historický obsah vyvíjeného souboru hraje významnou roli při vývoji projektu ve všech odvětvích, ve kterých jsou systémy pro správu revizí nasazovány. Za všechny možné lze uvést projekty plynoucí z oblasti softwarového inženýrství, kde se systémy pro správu revizí starají o samotný kód vyvíjené aplikace, o doprovodné dokumentace atd. Z tohoto příkladu je patrné, že systémy pro správu revizí jsou určeny zejména pro správu souborů obsahujících textový obsah srozumitelný člověku.

Revize, se kterými tyto systémy pracují jsou ve skutečnosti změny provedené na daném souboru. Úkolem systému pro správu revizí je tyto změny hlídat a uživateli poskytovat ucelený pohled na vývoj jednotlivých souborů a zároveň umožnit jednoduše porovnávat obsahy souborů. Tímto zabezpečují, aby měl kdykoli přehled o konkrétních změnách, které byly se souborem provedeny.

V praxi ovšem jednoduché verzování souborů nestačí. U softwarových projektů je naprosto běžnou praxí, že projekt je složen z velkého množství souborů, které spolu logicky souvisí. V tomto případě není na místě, aby systém pro správu revizí vnímal změny pouze na úrovni souborů, ale je nutné, aby se zaměřil i na větší objemy změn pokrývající velké množství souborů. Dílčí změny na sebe logicky navazují. Tímto krokem se posouváme z úrovně revize fyzického souboru, na úroveň logickou. Zatímco v první revizní úrovni se systém pro správu revizí musí starat o fyzické uložení souborů, jejich změn atd., na vyšší úrovni abstrakce již pracuje pouze s metadaty. Metadaty se rozumí logické svázání změn nejnižší úrovně do celku logické změny.

Jestliže, se na celý projekt podíváme jako na logickou strukturu, která se v průběhu svého života mění, pak je patrné, že ve svém životním cyklu prochází určitými stavy. Jakákoli změna projektu s sebou zároveň nese změnu stavu celého projektu. Změna stavu projektu je způsobena vždy logickou revizí projektu, tedy revizí, která najednou reviduje libovolné množství souborů. V případě, že by tento hierarchický mechanismus nebyl zaveden a nový stav projektu by byl v relaci s revizí fyzického souboru, pak by tento mechanismus vedl k tomu, že by existovalo tolik stavů projektu, kolik bylo revizí souborů za celou dobu vývoje projektu. Z poslední věty je cítit, že tento přístup by rozhodně správný nebyl a uživatel takového systému by byl nespokojen nad ztrácející se přehledností vývoje, kterou mu systém měl přinést.

2.1 Revize textových i netextových dat

Revizní systémy jsou navrženy zejména pro práci s daty srozumitelnými. Jedná se tedy převážně o textová data, jako jsou zdrojové kódy, případně jiné textové řetězce. Revizní systém by měl svému uživateli umožnit porovnávání dvou souborů dat a tím získat přehled o změnách, které byly s daty provedeny. Výsledkem porovnávání je vždy rozdílový soubor, který popisuje změny mezi oběma soubory. Při porovnávání je nutné vždy určit, jeden soubor jako výchozí a druhý, který bude s výchozím souborem porovnáván. Rozdílový soubor je množina po sobě jdoucích bloků, které popisují rozdíly mezi soubory. Typy bloků, které může obsahovat jsou tyto:

- **Nezměněná data** – jedná se o blok dat, který se v obou souborech nachází v identické podobě.
- **Nová data** – reprezentuje blok dat, který není obsažen ve výchozím souboru, ale je obsažen v porovnávaném souboru.
- **Změněná data** – reprezentuje blok dat, který je shodný v obou souborech, ovšem nacházejí se v něm drobné odchylky (změna názvu proměnné, konstanty, atd.).
- **Odstraněná data** – reprezentuje blok dat, který je součástí výchozího souboru, ale není obsažen v porovnávaném souboru.

V případě porovnávání netextových dat je situace mnohem složitější, protože není možné na nejnižší úrovni do dat zavést sémantiku, která by data logicky členila. U textových dat velmi napomáhají konce řádků, či jiné separátory, které u binárních dat chybí. Aby porovnávání binárních dat uživateli přineslo užitek, je nutné, aby proces porovnávání byl schopen porozumět sémantice takto uložených dat. Protože formát ukládaných dat si každý výrobce softwaru definuje sám a většinou nebývá standardizovaný, je nutné pro každý typ dat mít speciální porovnávací mechanismus.

2.2 Základní operace se systémem

Každý systém pro správu revizí musí svému uživateli poskytnout minimální podmnožinu příkazů, které jsou nezbytné pro to, aby se systém dal nazvat systémem pro správu revizí. Těmito příkazy jsou:

1. **CheckIn** – tato operace slouží k registraci změn souborů uložených v systému mezi soubory lokálními na uživatelově PC. Registrace změn se provádí podle schématu logických změn, tudíž při jednom provedení tohoto příkazu se do systému nahrávají všechny soubory, na kterých byla provedena revize a tyto revize vzájemně spjaty jako jedna logická revize projektu [2]. Provedení tohoto příkazu mění stav projektu. Uživatel nemusí sám, z možné velké záplavy souborů, vybírat ty které změnil, ale udělá to za něj uživatelské rozhraní revizního systému, které bude pracovat pouze se soubory, které byly lokálně revidovány.

2. **CheckOut** – tato operace umožňuje uživateli stažení poslední revize projektu do svého lokálního úložiště a bez dohledu revizního systému na něm pracovat a provádět změny [2]. Tímto mechanismem se zařídí, aby všichni členové týmu pracujících na projektu obdrželi vždy poslední konfigurací projektu.
3. **Fork** – tato operace slouží k rozdělení vývojových větví. Této operaci bude věnována samostatná kapitola, protože se jedná o dosti složitý mechanismus [1].
4. **Merge** – je opakem operace *fork* a naopak slouží ke slučování vývojových větví do jedné [1].

2.3 Princip verzování

Jak již bylo nastíněno, projekt při svém vývoji nabývá mnoha stavů. Pro potřeby uživatele je nutné čitelné odlišení jednotlivých stavů tak, aby se v nich mohl bez problémů orientovat. Těmto označením jednotlivých stavů se říká **verze**, která je reprezentována čísly oddělenými tečkami (např. 1.2.3.4). Čísla verzí jsou známa zejména ze softwarových produktů, kde se nejčastěji používají pro reprezentování verze finálního výrobku [2]. Finální výsledek projektu je často označován pouze dvojicí čísel, např.:

1.2

Číslice 1 – udává majoritní verzi produktu. Toto číslo se zpravidla mění při velkých změnách prováděných na produktu, majících zásadní vliv na jeho architekturu, případně podstatně mění jeho funkčnost.

Číslice 2 – udává minoritní verzi produktu. Toto číslo se využívá při menších změnách, zásadně neovlivňujících funkčnost produktu. Zpravidla se jedná o změny jako jsou, přidání nové funkčnosti, opravy chyb atd.

Tento dvou-číselný model je nejhojněji využíván u finálních produktů projektů vyvíjejících generický software a toto číslování má pouze marketingový charakter. Při vývoji softwarového projektu je dvou-číselné označení nedostačující. Při vývoji softwaru se uplatňují další kroky vývoje, které potřebují dodatečné interní verzování. Mezi takovéto kroky patří zejména testování, při kterém se testerům podává jedno spustitelné sestavení cílového produktu, které je podrobeno testům a všechny chyby hlášené k danému sestavení jsou předmětem revizí. Často se používá model čtyř-číselný, např.:

1.2.3.4

V tomto modelu mají první dvě čísla naprosto stejný význam jako v předchozím. Navíc přibyla další dvě čísla s tímto významem:

Číslice 3 – číslo sestavení. Toto číslo udává verzi spustitelného kódu, který byl z kódu projektu sestaven.

Číslice 4 – číslo revize. Toto číslo popisuje onen stav projektu, o kterém bylo hovořeno v obecném pohledu.

Jak je vidět, tento model již nemá pouze marketingový charakter, ale popisuje stav projektu komfortnějším způsobem. Přesto při reálném vývoji softwaru za použití systému pro správu revizí je i toto číslování nedostatečné. Ona nedostatečnost plyne z podpory revizních systémů pro operaci rozvětvení vývoje, kde každá vývojová větev je verzována separátně a je kladen důraz na to, aby bylo možné z čísla verze větve vyčíst, z jakých větví vychází.

3 Principy správy revizí

Tato kapitola se zabývá vnitřními principy systémů pro správu revizí a zároveň popisuje architekturu systému, který je v této práci navržen a implementován.

Základem každého revizního systému je úložiště fyzických souborů a úložiště metadat. Následně pak obsahuje postupy, které se uplatňují při operacích s revizním systémem.

3.1 Repositář

Jako repositář je v terminologii revizních systémů chápána struktura popisující celý spravovaný projekt. Tato struktura obsahuje fyzické soubory, které korespondují se soubory projektu, a metadata. Metadaty je obecně myšlena struktura, nebo spíše popis fyzických dat. V případě revizních systémů se jedná o popis organizace fyzických souborů do logických celků [7].

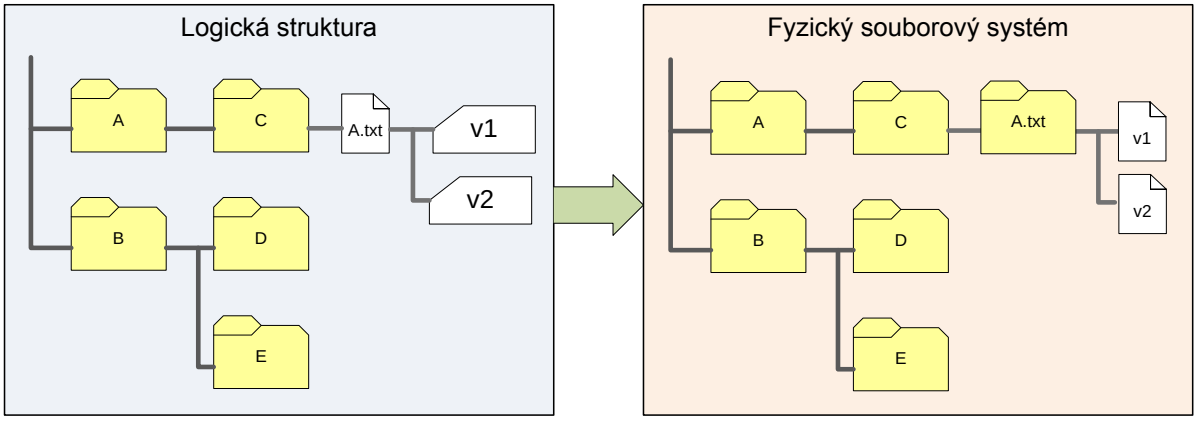
Všechny základní operace definované v kapitole 2.2, revizní systém provádí nad repositářem.

3.1.1 Úložiště fyzických souborů

Uložení fyzických souborů v systému pro správu revizí je základním kamenem celého systému, není však nijak technologicky náročné. V zásadě je možné využít čtyř různých způsobů jak fyzické soubory ukládat. Všechny tyto způsoby musí počítat s jedním zásadním rozdílem proti klasickému souborovému systému a tím je existence více instancí jednoho fyzického souboru [7].

3.1.1.1 Obraz v souborovém systému

Prvním možným řešením tohoto problému může být vytvoření identické stromové struktury, jakou má projekt ukládaný lokálně. V tomto přístupu, jako i v ostatních, je nutné se vyrovnat s faktem existence více instancí jednoho fyzického souboru. Řešením tohoto problému je mapování souborů na složky fyzického souborového systému. Soubory fyzického souborového systému budou tvořit jednotlivé verze daného souboru. Obrázek 1 demonstruje mapování logické struktury repositáře na fyzický souborový systém.



Obrázek 1 - Mapování logické struktury repositáře na fyzický souborový systém

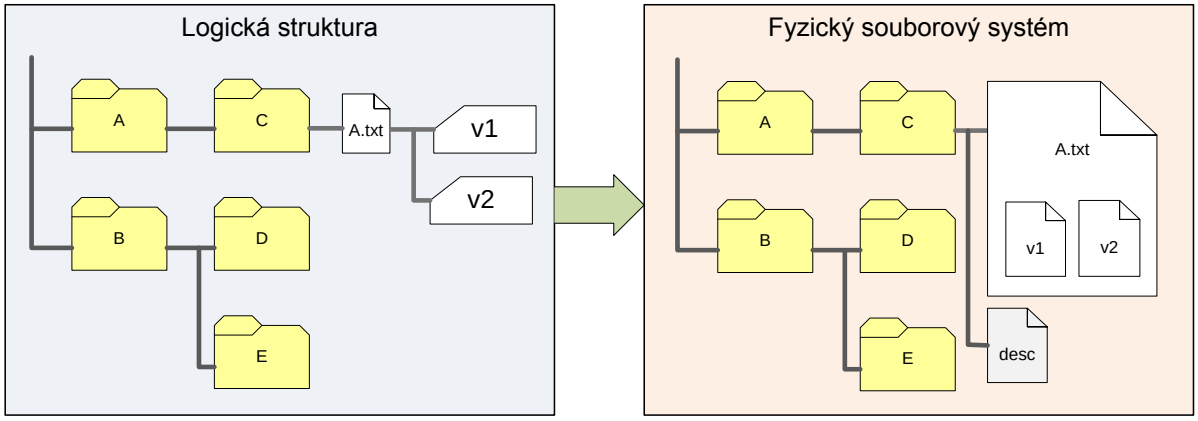
Obrázek 1 znázorňuje situaci, kde logický soubor *A.txt* obsahuje dvě verze. Tento stav je potřeba perzistentně uložit do souborového systému. Mapování tuto situaci řeší tak, že soubor *A.txt* je fyzicky nahrazen složkou se stejným názvem a obě verze daného logického souboru jsou uloženy jako fyzické soubory do této složky. Tímto postupem je zachována nadřazenost souboru *A.txt* nad jeho verzemi.

Tento postup je prakticky použitelný pouze u projektů, u kterých se nepředpokládají masivní změny logické struktury, kde by tato operace byla časově velmi náročná, s ohledem na pomalý přístup k diskovým kapacitám.

Naproti této nevýhodě má jednu podstatnou výhodu, kterou je zachování celé struktury projektu fyzicky a tím i jednodušší obnovení projektu po havárii, která způsobila ztrátu metadat.

3.1.1.2 Skládání verzí souboru

Tento přístup je založen na snaze minimalizovat počet fyzických souborů ukládaných v hostitelském souborovém systému. Výsledkem tohoto snažení je ukládání všech verzí souboru do jednoho fyzického souboru. Ve výsledném souboru jsou všechny jeho verze uloženy za sebou. Jedinou informaci, kterou systém pro správu revizí potřebuje znát je přesné umístění dat každé verze v takto vytvořeném souboru. Problém je dobře řešitelný jedním přidaným souborem, který popisuje vnitřní strukturu souboru a rozsahy fyzického uložení jednotlivých verzí. Obrázek 2 popisuje jak může mapování prakticky vypadat.



Obrázek 2 - Skládání verzí do jednoho fyzického souboru

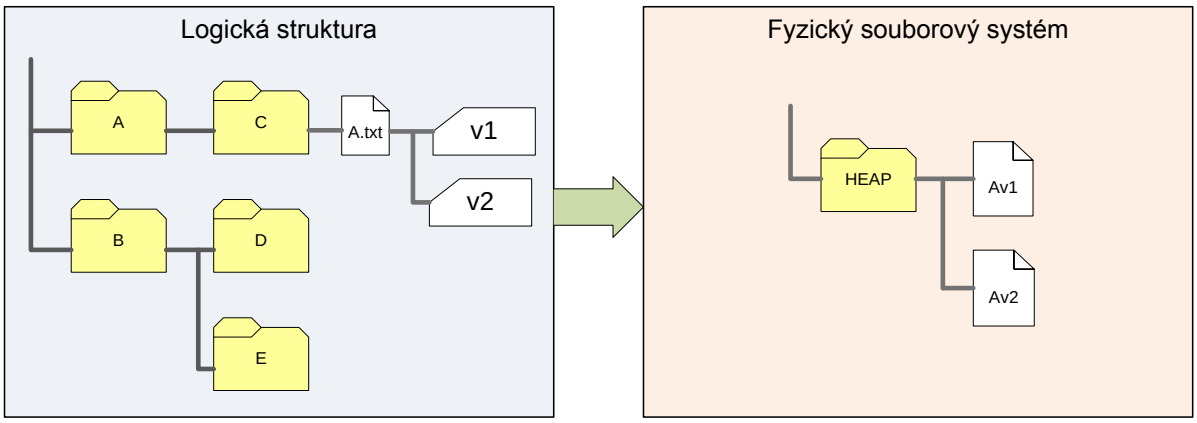
Obrázek 2 znázorňuje situaci, kde logický soubor *A.txt* je fyzicky mapován na stejnojmenný fyzický soubor, který ve svém těle obsahuje všechny své verze. Soubor *desc* je tím stínovým souborem, který slouží pouze k popisu vnitřní struktury souboru *A.txt* a udržuje reviznímu systému informaci o tom, jak jsou jednotlivé verze fyzicky uloženy.

Tento postup velmi zásadně zvyšuje lidskou čitelnost fyzického uložení celé struktury, protože je naprosto identická s logickou. Proti předchozímu přístupu je tento o poznání efektivnější co se týče změn logické struktury. Podstatnou výhodou může být i fakt, že při existenci dostatečných informací uvnitř popisujícího souboru je možné obnovit strukturu metadat v případě jejich ztráty způsobené havárií. Tato vlastnost s sebou ovšem nese nutnost duplicity této informace jak v metadatech, tak v popisujících souborech.

Zásadním omezením je nepříjemné chování při odstranění některé nechtěně vložené verze souboru, při níž by docházelo k fragmentaci fyzického souboru, což by vedlo k nešetrnému zacházení s diskovým prostorem, obzvláště u rozsáhlejších souborů. Dalším omezením je maximální velikost fyzického souboru, kterou dovoluje souborový systém uložit.

3.1.1.3 Mapování na haldu

Tento přístup vychází z existence jednoho velkého datového úložiště všech fyzických souborů nazvaným *halda* (heap). Do tohoto úložiště jsou ukládány všechny fyzické soubory, které tvoří verze jednotlivých logických souborů. Protože počet fyzicky ukládaných souborů bude roven součtu všech verzí napříč všemi logickými soubory, tak je patrné, že úložiště může obsahovat obrovské množství souborů. Systém pro správu revizí musí zajistit to, aby žádný z fyzicky ukládaných souborů nekolidoval s jiným. Tento problém se řeší zavedením speciálního číslování fyzicky ukládaných souborů a informace o vazbě mezi verzí logického souboru a jemu odpovídajícího fyzického souboru je uložena v metadatech.



Obrázek 3 - Mapování logických souborů na haldu

Obrázek 3 popisuje, jak je tento způsob prakticky proveditelný. Fyzické úložiště je tvořeno pouze jednou složkou, která obsahuje všechny ukládané verze logických souborů. Je patrné, že verze 1 souboru *A.txt* je mapována na fyzický soubor *Av1* a podobně je tomu i u druhé verze tohoto souboru. S čím se musí systém pro správu revizí vyrovnat, je fakt, že v celém projektu může existovat více souborů se stejným názvem v různých podsložkách. Tudíž musí obratným způsobem vytvářet jména fyzických souborů tak, aby nedocházelo ke kolizím.

Hlavním cílem tohoto přístupu je zbavení se nutnosti udržovat logickou strukturu projektu na fyzickém úložišti a tím snížení režie při práci se systémem obzvláště v případech, které se strukturou projektu manipulují. V případě ukládání velkého množství malých souborů s velkou frekvencí jejich změn může být tento postup značně neefektivní, protože souborové systémy jsou konstruovány tak, že alokovaný prostor pro soubory zarovnávají na určitou hodnotu. V případě zarovnání na 4kB a maximální velikostí ukládaných souborů 1kB má tento způsob redundanci 75%, což je naprosto nevyhovující. Tedy v tomto případě je lepší variantou uložení do jednoho fyzického souboru popsaného v kapitole 3.1.1.2.

3.1.1.4 Využití systému řízení báze dat

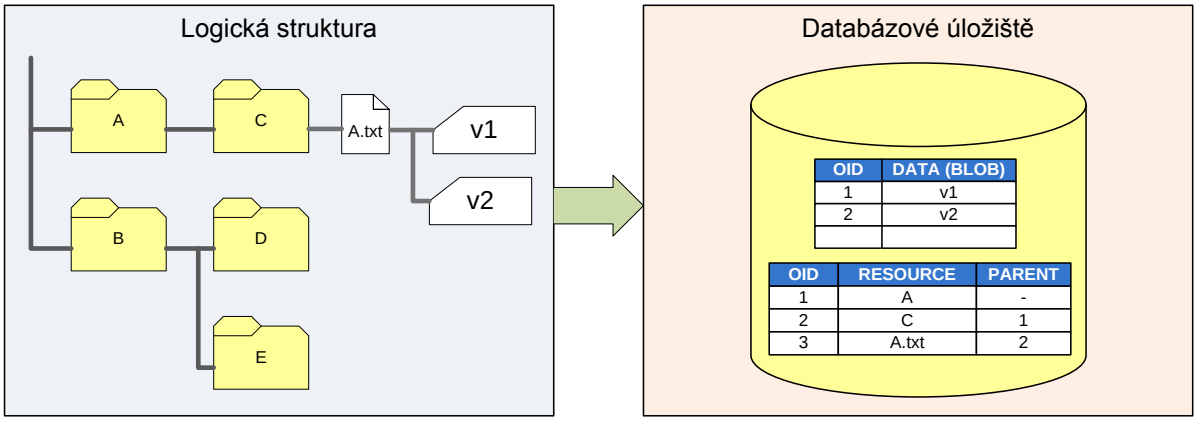
Posledním zmíněným schématem, které je k tomuto tématu použitelné je ukládání fyzických souborů pomocí systémů řízení báze dat. Tímto je fyzické úložiště naprosto oproštěno od nutnosti vlastní implementace logiky ukládání fyzických souborů a zároveň systém řízení báze dat ukládá data dostatečně efektivně bez nadbytečné redundance cílového prostoru. Většina dnes používaných systémů řízení báze dat bez problémů zvládá ukládat rozsáhlá data v binární podobě do tzv. BLOB objektů (Binary Large OBject), což přináší značné výhody z hlediska snadnosti přístupu k fyzickým souborům.

V případě sloučení obou datových struktur a to fyzické a logické do systému řízení báze dat, lze vazby mezi nimi implementovat na úrovni schématu databáze, čímž celá struktura dostává

ucelený formát z hlediska abstraktního pohledu. Nezanedbatelná je i možnost jednoduchého zálohování všech dat, při které postačí přímočaré vytvoření zálohy celé databáze, z čehož plyne, že i obnova celého systému v případě havárie není o nic více složitější.

Velkou přidanou hodnotu tohoto přístupu přináší možnost dokonalého oddělení výpočetní části systému pro správu revizí a jeho fyzického úložiště. Rozdělení vytváří dva samostatné výpočetní uzly. Tímto jsme schopni dosáhnout toho, že oba uzly revizního systémů mohou běžet na různých strojích a nejsme vázáni místem spuštění výpočetní části systému. Toto rozdělení má též podstatný vliv na způsob zabezpečení celého systému, kde kritickým bodem je úložiště, které v sobě obsahuje celý repositář a výkonná část je vždy lehce nahraditelná.

Obrázek 4 schématicky znázorňuje, jakým způsobem jsou data v revizním systému uložena za podpory systému řízení báze dat.



Obrázek 4 - Uložení souborů za podpory SŘBD

3.1.2 Úložiště logické struktury

Způsob a forma uložení popisu projektového stromu a jeho všech možných variant, tím jsou myšleny možné paralelní vývojové větve, je zásadní pro rychlou práci. Projektový strom je nejčastěji využívanou komponentou celého systému. Existuje více způsobů jak lze strukturu projektu perzistentně ukládat, nicméně každý ze způsobů koresponduje se způsobem fyzického uložení souborů. Tentokrát se však nejedná o způsoby mapování na haldu, či obrazem v adresářové struktuře, nýbrž o způsob ukládání změn v jednotlivých souborech a následné vytváření nových variant (verzí) souborů.

První variantou je ukládání celých souborů, tak jak reálně existují. Tato varianta je velmi příjemná z důvodu jednoduché a rychlé přístupnosti libovolné verze kteréhokoli souboru, bez zásadních požadavků na výkonnost systému. Stromová struktura se v tomto případě rozšíří o jednu úroveň, kterou jsou verze fyzických souborů a ty tvoří listy celého stromu. Pak jediným úkolem systému je pamatovat si poslední číslo verze souboru [2].

Druhou variantou je ukládání diferenčních souborů. Tyto soubory se vytváří při každé revizní operaci se stromem, a to jenom u souborů, které byly při vývoji změněny. Při operaci získání souboru v požadované verzi z revizního systému, je nutné na základě diferenčních souborů provést seskládání souborů, kde výsledek odpovídá souboru v dané verzi [1]. Tato operace je ovšem značně náročná, a to hlavně v případech, kdy počet změn je obrovský. Na příkladu je uvedeno získání verze 6 modelového souboru. Získání je provedeno složením všech změn, které byly na tomto souboru provedeny od doby jeho vytvoření do doby aktuálnosti požadované verze.

$$C_1 \mid C_2 \mid C_3 \mid C_4 \mid C_5 \mid C_6 \rightarrow V_6$$

Tento mechanismus se dá jednoduše urychlit, a to vytvářením mezistupňů, kdy se například po každé desáté revizi vytvoří jeden souhrnný rozdílový soubor. Tím se podstatně sníží počet kroků, které je k získání výsledného souboru nutné provést.

$$C_1 \mid C_2 \mid C_3 \mid C_4 \mid C_5 \rightarrow \boldsymbol{CC_1}$$

$$\boldsymbol{CC_1} \mid C_6 \rightarrow V_6$$

Příklad demonstruje praktické využití tohoto mechanismu, kde hranicí pro vytváření souhrnných revizních souborů je pět základních revizí. Tedy po páté provedené revizi byl vytvořen soubor $\boldsymbol{CC_1}$, který v sobě kloubí všechny předešlé diferenční soubory. Později byla do systému zavedena další revize, která způsobila vytvoření diferenčního souboru $\boldsymbol{C_6}$. V případě, že uživatel nyní požádá o poslední verzi souboru, bude tento seskládán jen z diferenčních souborů $\boldsymbol{CC_1}$ a $\boldsymbol{C_6}$.

3.2 Logická struktura repositáře

Předchozí kapitoly pojednávaly spíše o vnitřním fungování revizního systému, do kterého uživatel systému nevidí. Na repositář se uživatel může dívat z různých úhlů pohledu. V prvé řadě jej zajímá pohled na strukturu projektu. Jedná se tedy o adresářovou strukturu, která obsahuje verzovatelné objekty, tedy soubory. Prakticky uživatel nepožaduje pouhé zobrazení nejčerstvější podoby, ale chce mít i možnost nahlížet do historie projektu. Pohled na repositář je tedy velmi silně spjat s časovým údajem, který uživatele v pohledu na repositář posouvá do historie.

Dalším aspektem při pohledu na strukturu projektu je větev, ve které se uživatel nachází. Vývoj projektu může být za svého života paralelizovaný do vývojových větví, kde každá z nich může vypadat podstatně jinak (blíže je tato problematika popsána v kapitole 3.3).

Další pohled, který uživatele zajímá je struktura provádění revizí a změny jednotlivých verzí. S touto problematikou úzce souvisí granularita záznamu revizí.

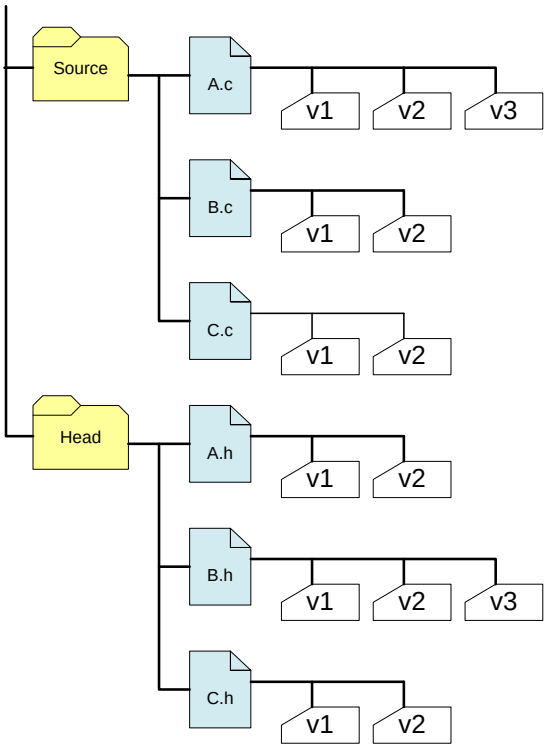
- Nejjemnějším rozlišením je sledování revizí u konkrétního souboru. Zde si uživatel dokáže udělat obrázek o tom, jak se daný soubor vyvíjel. Při tomto pohledu se velmi silně uplatňuje porovnávání jednotlivých verzí souborů, aby bylo možné identifikovat změny, které revizi způsobily. Dalším aspektem je kontrola oprávněnosti a správnosti revize.

- Mezi revizemi souborů většinou existuje vazba, která jim dává logičtější charakter. V praxi se velmi často zásah do projektu neobejde bez úpravy většího množství souborů než je pouze jeden. I tento fakt musí být v revizním systému zaveden. Tedy jedna revize projektu může postihovat velké množství souborů a je tedy složena z libovolného množství revizí souborů.
- Revize projektu mění stav projektu a verze projektu je odvozena od počtu jeho revizí. Revizní systém uživateli poskytuje ještě jeden aparát, který pohled na vývoj projektu více zjednodušuje. Tímto aparátem je možnost ručního vkládání milníků, které označují významné fáze v práci na projektu. Prakticky se vložení milníku projeví jako změna čísla verze v jedné z majoritnějších složek.

Z popisu pohledů na projekt je patrné, že je revizním systémem chápán jako stavový prostor, popisující projekt v jeho vývojových fázích. Přechody mezi stavy jsou ohodnoceny revizemi projektu (jeden přechod = jedna revize).

3.2.1 Projektový pohled

Projektový pohled je znázorněn na Obrázek 5 a demonstruje fyzické vzezření projektového stromu. Toto je jediný pohled, který obyčejného uživatele revizního systému zajímá, protože potřebuje pouze získat poslední verzi projektu na kterém má provádět práci, případně porovnat starší verzi souboru s novější pro ujištění správnosti jeho následné práce [2].

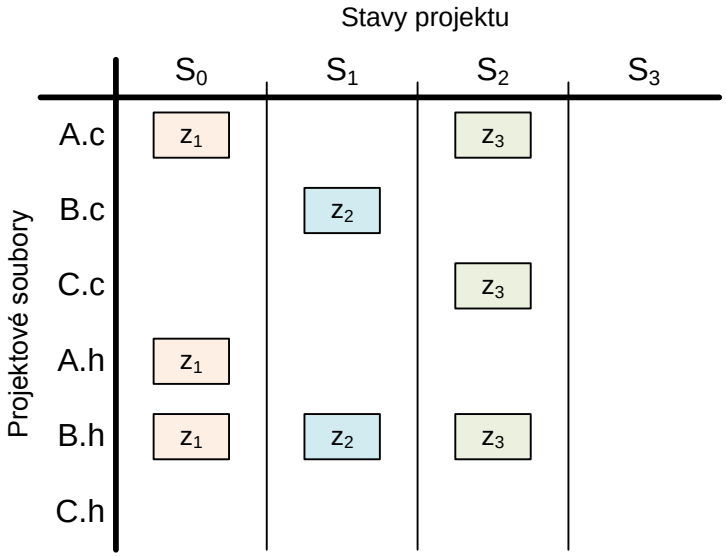


Obrázek 5 - Stromová struktura projektu

3.2.2 Pohled revizí

Druhý pohled demonstrovaný na následujícím obrázku vykresluje pohled na vývoj stavů projektu a na příčiny, které změnu stavu způsobily [1].

Na horizontální ose jsou vyneseny stavy projektu, ve kterých se během svého životního cyklu nacházel. Vertikální osa znázorňuje fyzické soubory projektu. Na průsečících stavu a fyzického souboru jsou evidovány revize souboru. Zároveň všechny revize souborů jsou označeny stejně, jako jedna revize projektu v daném stavu.



Obrázek 6 - Schéma revizí projektu

Tabulka popisující změny stavu projektu se dá formálněji zapsat prostřednictvím gramatiky typu 3, tedy gramatiky tvořící regulární jazyk .

$$G = (N, \sum, P, S, F), \text{ kde}$$

N reprezentuje stavy projektu, ve kterých se může nacházet

$\sum$ množina revizí projektu

P množina přechodových pravidel, přechod reprezentuje revizi výchozího stavu projektu

S výchozí stav

F množina koncových stavů

Gramatika popisující příklad demonstrovaný na Obrázek 6 se dá zapsat následovně.

$N = \{S_0, S_1, S_2, S_3\}$

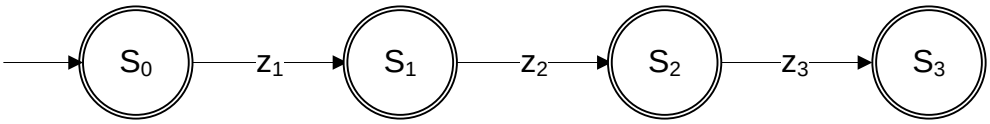
$\sum = \{z_1, z_2, z_3\}$

$P = \{S_0 \rightarrow z_1S_1, S_1 \rightarrow z_2S_2, S_2 \rightarrow z_3S_3\}$

$S = S_0$

$F = \{S_0, S_1, S_2, S_3\}$

Z takto definované gramatiky je možné sestrojit konečný automat.



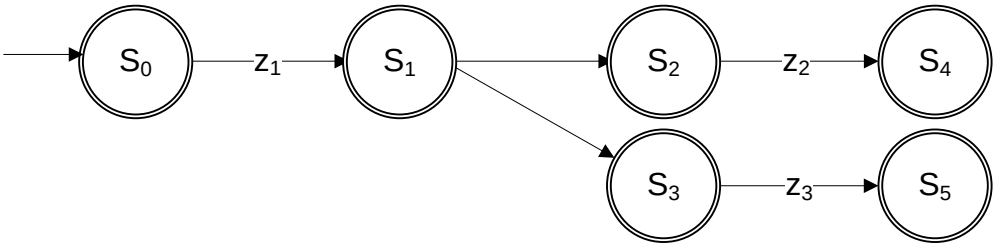
Obrázek 7 - Automat popisující stavový prostor verzí projektu

3.3 Rozdělení vývojových větví

Rozdělení vývojových větví je mechanismus, kterým revizní systém simuluje rozdělení vývoje projektu do nezávislých pracovních skupin, pracujících na odlišných částech projektu [7]. Pracovní skupiny nejsou při své práci v kolizi. Tento prostředek napomáhá v procesu paralelizování práce na projektu a tím urychlení jeho vývoje.

Praktický obraz poskytne pohled na vývoj softwarového projektu, který se již nachází v nějakém funkčním stavu, nicméně se na něm nadále pracuje. V průběhu vznikne požadavek na optimalizaci určité části projektu. Kdyby tým lidí, kteří tuto optimalizaci provádí pracovali na stejném kódu jako tým, který provádí vývoj nových funkcionalit, docházelo by k situacím, kdy chyby v prováděné optimalizaci mají za následek chybné fungování nově vytvářeného kódu. Z tohoto důvodu je nutné, aby vývojový tým pracující na nových funkcionalitách pracoval stále s odladěnou verzí a práce optimalizačního týmu jej nijak neomezovala. Tento problém řeší možnost větvení repositáře, kde uživatel vidí pouze svou větev projektu a může pracovat pouze se soubory ve své větvi.

Po provedení větvení projektu z příkladu z kapitoly 3.2.2 by stavový prostor projektu vypadal následovně (demonstrace bude pro názornost nadále prováděna na stavovém diagramu).



Obrázek 8 - Stavový prostor po rozdělení projektu

Takovému stromu potom odpovídá gramatika:

$N = \{S_0, S_1, S_2, S_3, S_4, S_5\}$

$\sum = \{z_1, z_2, z_3\}$

$P = \{S_0 \rightarrow z_1 S_1,\ S_1 \rightarrow S_2,\ S_1 \rightarrow S_3,\ S_2 \rightarrow z_2 S_4,\ S_3 \rightarrow z_3 S_5\}$

$S = S_0$

$F = \{S_0, S_1, S_2, S_3, S_4, S_5\}$

Popsaná gramatika zavádí nový typ pravidla, dále označovaného jako **ε**, které vytváří přechod stav→stav, bez aplikace revize mezi oběma stavy. Toto pravidlo vytváří pouze referenci na výchozí stav ve stavu koncovém.

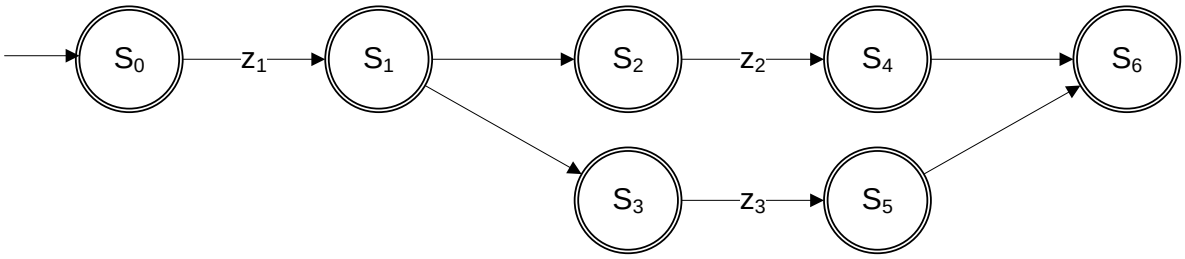
Rozdělení větví je provedeno zavedením dvou **ε** přepisovacích pravidel (např. $S_1 \rightarrow S_2$). Tedy existence dvou **ε** přepisovacích pravidel vycházejících z jednoho stavu popisuje proces rozdělení vývojové větve.

3.4 Slučování větví

Slučování větví je opačným procesem k procesu rozdělení. Tedy slučuje do sebe dvě vývojové větve a vytváří z nich větev novou [7]. Slučování má za následek, že uživatelé v obou větvích se taktéž spojí a po provedení operace spojení vidí všichni stejnou větev projektu.

Prakticky se dá tento proces demonstrovat na příkladu z předchozí kapitoly. Tým, který pracoval na optimalizaci již funkční části projektu svou práci dokončil a optimalizovaná část může být použita v hlavní vývojové větvi. Provede se sloučení obou větví, kde výsledkem je jedna vývojová větev. Tým pracující na vývoji nových funkcionalit při své práci ani nemusí tuto změnu zaregistrovat, snad jen si může povšimnout zvýšení výkonu výsledné aplikace v případě, že optimalizace byla úspěšná.

Tento proces se dá opět prezentovat na stavovém diagramu vycházejícího z diagramu z předchozí kapitoly.



Obrázek 9 - Stavový prostor po sloučení větví projektu

Gramatika odpovídající takovému stromu je následující:

$N = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6\}$

$\sum = \{z_1, z_2, z_3\}$

$P = \{S_0 \rightarrow z_1 S_1,\ S_1 \rightarrow S_2,\ S_1 \rightarrow S_3,\ S_2 \rightarrow z_2 S_4,\ S_3 \rightarrow z_3 S_5,\ S_4 \rightarrow S_6,\ S_5 \rightarrow S_6\}$

$S = S_0$

$F = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6\}$

Potom existence dvou **ε** přepisovacích pravidel směřujících do jednoho stavu popisují proces slučování vývojových větví. V tomto případě **ε** pravidlo nevytváří pouze odkaz na předchozí stavy, jako v případě rozdělování, ale vytváří nový stav, který je kombinací obou slučovaných.

Při operaci slučování může docházet ke kolizím, které jsou zapříčiněny tím, že byly provedeny změny téhož souboru v obou větvích. Tento proces může být částečně automatizovaný a ponechaný na stroji. Jenže ve valné většině případů dochází ke kolizím, kde by stroj musel rozumět sémantice problému, který má slučovat. V těchto případech by sloučení mohlo být chybné, proto jsou kolize často řešeny lidským faktorem.

Příkladem strojově řešitelného kolizního stavu je situace, kdy byly do souboru implementovány nové funkce. Opačným případem je situace, kdy v obou větvích byla změněna stejná funkce a jednoduché sloučení tedy možné není.

4 Návrh systému

Tato kapitola shrnuje možné alternativy architektury revizního systému, zejména z pohledu uložení fyzické části repositáře. Dále popisuje ucelenou podobu systému a způsob komunikace mezi jeho jednotlivými částmi.

4.1 Požadavky na revizní systém

Revizní systémy je možné přirovnat k jistému druhu souborového systému, který je přes definované rozhraní poskytován uživateli. Ovšem možnosti práce s tímto souborovým systémem jsou hodně omezené vzhledem k operacím, které musí provádět. Revizní systém je nucen ve svém nitru velmi pečlivě uchovávat informace o struktuře dat, která spravuje a následně i samotná fyzická data.

S ohledem na charakter dat a následky možných chyb, které by jejich vznik způsobil, jsou na revizní systém kladeny vysoké požadavky co se týče bezpečnosti uložení obou částí repositáře. Dále je kladen velký důraz na dostupnost systému, který má význam hlavně v případě víceuživatelského přístupu, kde skupina uživatelů, která systém využívá je velmi početná a se systémem pracuje intenzivně.

4.1.1 Bezpečnostní požadavky

Z hlediska bezpečnosti je nejdůležitějším argumentem možnost zničení repositáře a tím i celého projektu. Díky rozdělení repositáře na dvě části, a to na část logické struktury a část fyzického úložiště, je možné problém bezpečnosti rozdělit na dva podproblémy.

Prvním z nich je ochrana logické struktury repositáře, protože v případě poškození této komponenty se vytrácí celý pohled na projekt a v případě nestrukturovaného uložení dat ve fyzickém úložišti může být výskyt chyby v této části systému neopravitelný. Tudíž tato část systému má největší kritičnost co se bezpečnosti a nepostradatelnosti týče.

Druhým kritickým bodem je fyzické úložiště. Toto úložiště nemusí obsahovat informaci o struktuře projektu a tudíž je jeho kritičnost nižší. Nižší kritičnost je způsobena faktem, že výskyt malé chyby (prakticky chybějící soubor) nemusí významnou měrou narušit chod celého systému, jako by tomu bylo v předchozím případě. Je zřejmé, že na velké množství souborů se uživatel systému nikdy nedotáže, případně se k nim ani nebude chtít nikdy vracet. Tato vlastnost sice lehce snižuje kritičnost této části systému, nicméně je taktéž velmi nepostradatelnou.

Rozdíly v hodnocení kritičnosti jsou svázány zejména se schopností vyrovnání se s chybovým stavem jednotlivých částí systému v důsledku chybějící informace, případně fyzického selhání hardwaru a v míře nepostradatelnosti informace, kterou daná část systému uchovává.

4.1.2 Požadavky dostupnosti

Pod pojmem dostupnosti je v tomto případě vnímána zejména rychlost odezvy systému na zpracovávání požadavků plynoucích z možností práce s repositářem, které jsou ve své podstatě složité transakční operace nad celým systémem. Problém dostupnosti je možné opět rozdělit na dvě kategorie, a to na rychlost odezvy při práci s logickou strukturou repositáře a na rychlost získání konkrétních fyzických souborů.

Prvním problémem, který byl nastíněn je rychlost při práci s logickou strukturou repositáře, které je stejně jako v případě bezpečnostních požadavků kladen vyšší důraz. Vyšší priorita je způsobena hlavně faktem velkých výpočetních nároků z důvodu zpracovávání složitých transakcí, které po každé provedené operaci zachovávají strukturu repositáře v konzistentním stavu. Od této části systému se očekává, že bude výpočetně nejzatíženější, z důvodu častých dotazů na strukturu repositáře a případné provádění jeho změn.

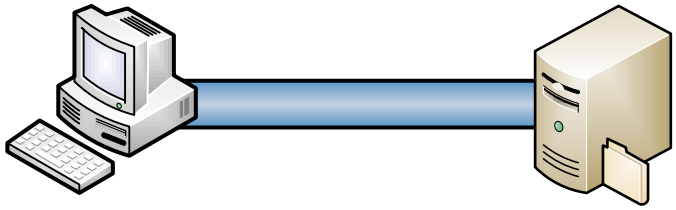
Druhým úzkým hrdlem je práce s fyzickými soubory. Tato část systému nevyžaduje velký výpočetní výkon, ale rozumnou datovou propustnost kanálu, přes který se data k uživateli přenášejí. S ohledem na teoretickou možnost, že revizní systém bude uchovávat například nějakou formu multimediálních dat, kde se předpokládají velké objemy přenesených dat, je nutné, aby tyto přenosy neměly zásadní vliv na rychlost odezvy výpočetní části systému.

4.1.3 Přenos fyzických souborů

Vstupem a výstupem operací revizního systému jsou soubory dat. Dá se předpokládat, že systém během svého provozu bude přenášet velké množství souborů, proto je nutné, aby byl tento proces kvalitně optimalizovaný. Kvalita zvládnutí tohoto procesu je určena mírou zatížení výpočetní části systému a rychlostí přenosu souborů mezi uživatelem a systémem. Nabízí se několik variant, jak proces přenosu souborů implementovat. Všechny varianty vynikají svými specifickými vlastnostmi a každá z nich najde své uplatnění.

4.1.3.1 Centrální uložení s řízeným přenosem

Na Obrázek 10 je znázorněno schéma řešení tohoto přístupu. Všechny fyzické soubory spravované systémem jsou uloženy přímo na řídícím stroji. Výpočetní jednotka je taktéž nucena obstarávat přenosy souborů.

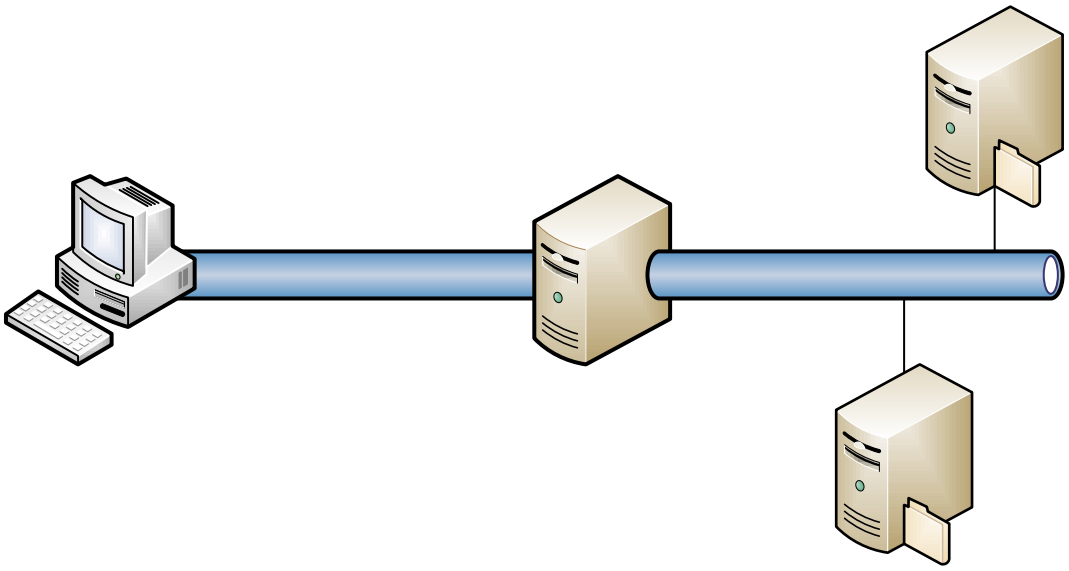


Obrázek 10 - Centrální uložení s řízeným přístupem

Toto řešení je akceptovatelné pro menší projekty, kde práce s revizním systémem není tak intenzivní a řízení přenosu souborů nebude mít neblahý dopad výkonnost systému.

4.1.3.2 Distribuované uložení s řízeným přenosem

Obrázek 11 demonstruje zavedení distribuovaného uložení souborů mezi více síťových míst [3]. Distribuce je zde chápána na úrovni celých projektů. V případě spravování velkého množství projektů revizním systémem je vhodné, aby distribuce rovnoměrně pokrývala jejich přístupové nároky. Proti předchozímu přístupu má tento výhodu v tom, že přenos souborů nezatěžuje diskové úložiště výpočetní části systému, která jej potřebuje k ukládání logické struktury repositáře. Přenos souborů je ovšem stále plně řízen výpočetní částí systému. Čtení souborů se však děje pouze na síťovém rozhraní a nezatěžuje tak disk výpočetní části.

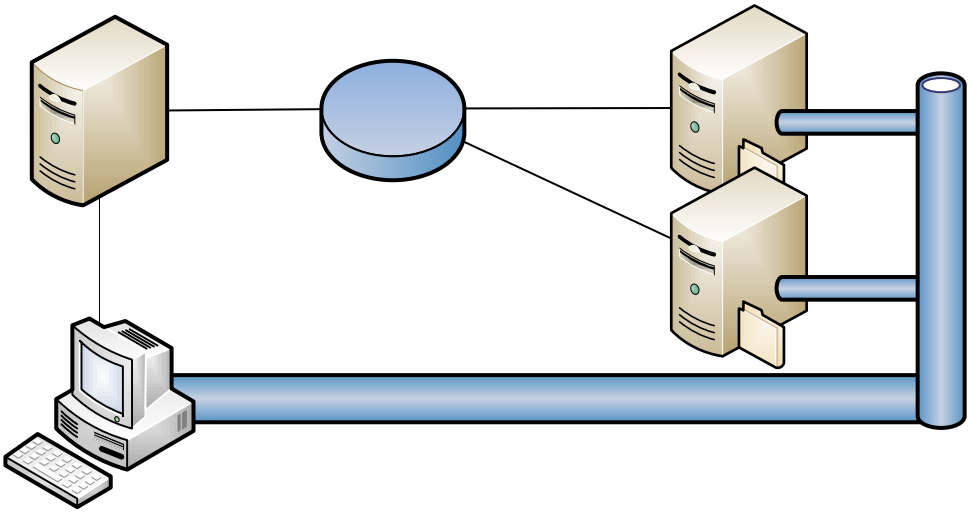


Obrázek 11 - Distribuované uložení s řízeným přenosem

Toto řešení se ovšem, stejně jako první, potýká s problémem limitované rychlosti síťového rozhraní. Tedy distribuce fyzického úložiště nemusí od jistého okamžiku přinést žádné zlepšení, díky nemožnosti přenést všechna připravená data.

4.1.3.3 Distribuované uložení s distribuovaným přenosem

Poslední schéma ponechává distribuci souborového úložiště a zároveň se snaží odstranit úzké hrdlo v přenosu souborů, kterým bylo síťové rozhraní. Výpočetní část systému již slouží pouze jako centrální koordinátor přenosů a přenosy souborů se realizují přímo mezi klientem a fyzickým úložištěm. V tomto řešení se přes výpočetní část systému data vůbec nepřenášejí a není tedy těmito procesy vůbec zpomalováno.



Obrázek 12 - Distribuované uložení s distribuovaným přenosem

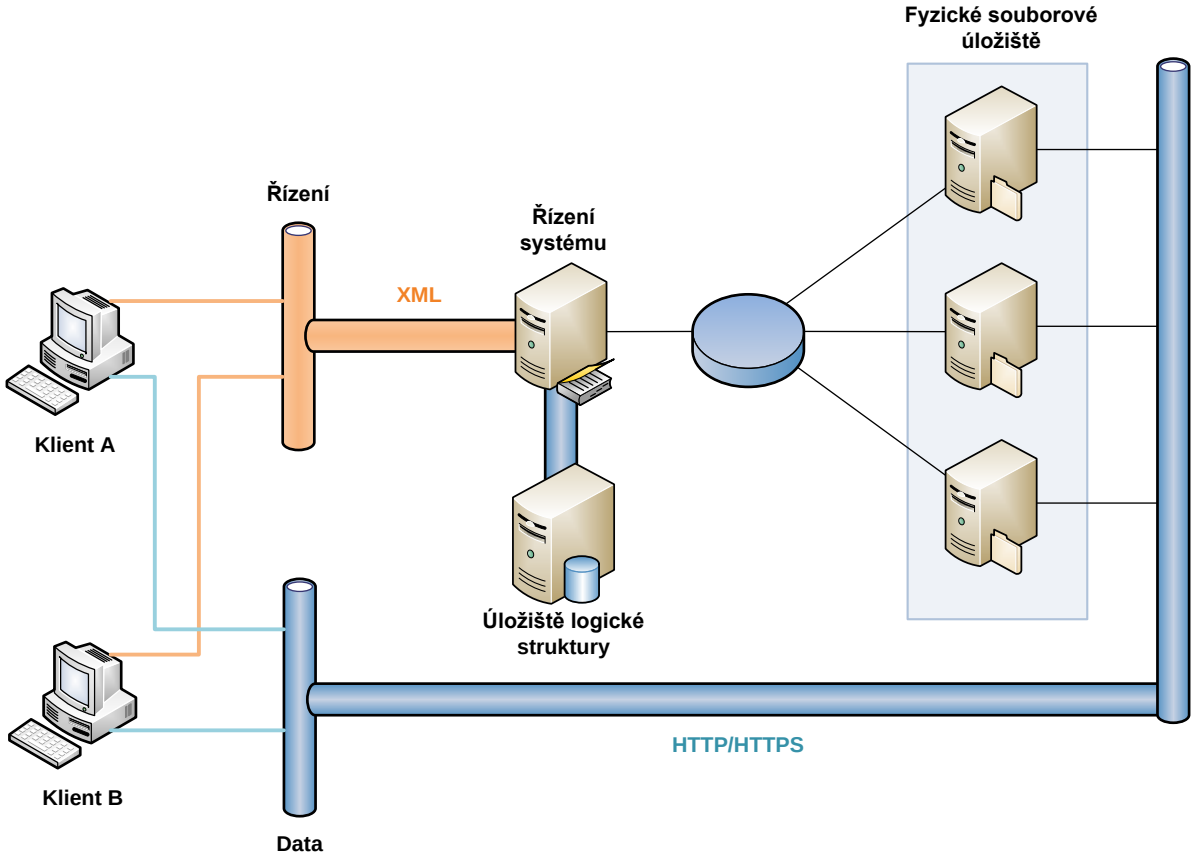
Výhoda tohoto řešení se výrazně projeví zejména u správy velkého množství rozsáhlých projektů, náročných na přenosové kapacity. Teoreticky se dá předpokládat, že datová propustnost vzroste tolikrát, kolikrát více fyzických úložišť systém bude obstarávat. To platí pouze za předpokladu, že každé úložiště je umístěno na vlastní nesdílené lince (na jednom PC využívá disk a síťové připojení pouze jeden souborový server).

4.2 Schéma kooperace

Na Obrázek 13 je znázorněno schéma navrženého, a v rámci této práce též částečně implementovaného, revizního systému.

Celý systém se sestává ze tří hlavních komponent, jimiž jsou :

- **Úložiště logické struktury** – v této části systému jsou uloženy kompletní informace o struktuře repositáře a poskytuje nutné operace pro práci s ním.
- **Fyzické souborové úložiště** – je jistým způsobem distribuovaným souborovým systémem, ovšem ne plně autonomním, nýbrž řízeným.
- **Řízení systému** – zajišťuje zpracovávání a tvorbu odpovědí klientům. Dále obstarává řízení fyzického úložiště a stará se o koordinaci se strukturou repositáře. V neposlední řadě je článkem, který v sobě implementuje algoritmy na sestavení stromu projektu, řízení přístupu ve víceuživatelském prostředí a hlídání transakčního zpracování uživatelských požadavků.



Obrázek 13 - Schéma kooperace

Je patrné, že systém zásadně odděluje práci s logickou strukturou repositáře a s fyzickými soubory tím, že vytváří klientovi dvě cesty, kterými data skrz systém procházejí. První cestou je cesta řízení, přes kterou klient systému zadává příkazy a požadavky na provedení operace. Stejnou cestou získává

zpět odpověď systému na stav vykonání operace. V případě datových toků, které v sobě zahrnují přenos fyzických souborů je využito druhé cesty, která vytváří přímou spojnici mezi klientem a fyzickým úložištěm. V tomto případě se řízení zapojuje pouze jako koordinátor komunikace [3] a přenos fyzického souboru jej nezatěžuje větší měrou. Je zde využito stejného principu, známého pod označením DMA (Direct Memory Access), použitého při řízení datových toků mezi jednotlivými komponentami počítače bez asistence procesoru.

Takto provedená implementace počítá s faktem, že řídící uzel systému a datové úložiště jsou v jedné síti. V případě jiného umístění souborových serverů je nutné využít síťových technologií (NAT, tunelové spojení) k zajištění tohoto přímého kanálu.

4.2.1 Úložiště logické struktury

Úložiště logické struktury repositáře je tou částí systému, kde jsou uloženy struktury popisující vazby mezi jednotlivými objekty projektu, čímž jsou myšleny jejich typy, vlastnosti, zanoření a podobně. Od této části systému se očekává, že bude obstarávat kompletní správu logické části repositáře tak, aby se řídící část systému nemusela starat o, pro ni nepodstatné, nízko-úrovňové operace při manipulaci s repositářem. Cílem tohoto oddělení od řídící části je zmenšení množiny příkazů, kterými bude řídící část s repositářem manipulovat.

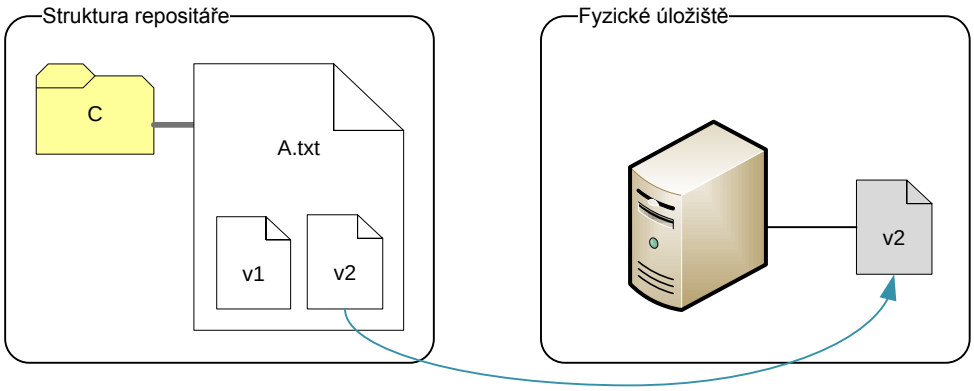
Typickým příkladem je zavedení nové změny projektu, která v sobě zahrnuje mnoho změn fyzických souborů. Jedním příkazem řídící systém požaduje po správci logické struktury zavedení všech změn, ten se již postará o vše co je zapotřebí, aby tato operace proběhla. Výsledkem jsou dva možné stavy a to buď „ANO“ nebo „NE“ podle úspěšnosti operace, která se provádí jako jedna transakce.

V revizním systému, implementovaném v rámci diplomové práce, je tato část logiky implementována pomocí systému řízení báze dat. Tedy všechny nutné informace, které revizní systém ke své práci potřebuje jsou uloženy v databázových tabulkách a aplikační logika na úrovni správy struktury repositáře je prováděna taktéž na databázovém stroji s využitím programovatelnosti databázového stroje jazykem PL/SQL (Procedural Language/Structured Query Language).

Toto řešení přesunuje podstatnou část výpočetních operací do jiného výpočetního uzlu, kde není problémem, aby tento uzel byl fyzicky realizovatelný na odlišném hardwarovém vybavení, čímž se podstatně zvýší rychlost odezvy celého systému.

4.2.2 Fyzické souborové úložiště

Z podstaty reprezentace repositáře, jako projektového stromu a neexistence pevné vazby směřující od fyzické reprezentace verze souboru směrem k logickému objektu, není revizní systém žádným způsobem vázán na způsobu uložení fyzických souborů. Prakticky situace může vypadat jako na Obrázek 14.



Obrázek 14 - Vazba mezi logickým a fyzickým úložištěm

V takovém případě je nutné, aby u každého uzlu repositáře, reprezentujícího fyzický soubor byla uložena informace o umístění souboru. Tímto velmi jednoduchým mechanismem se otevírá brána k dalšímu možnému snížení rychlosti odezvy na revizní systém. Toto vylepšení spočívá v jednoduchém mechanismu, kdy je možné jednotlivé spravované projekty umístit fyzicky na jiný hardware a tím snížit požadavky na propustnost linky, přes kterou by soubory měly procházet.

Jak již bylo nastíněno ve schématu kooperace, každá jednotka fyzického úložiště poskytuje pouze operace pro zápis a čtení souborů v ní uložených. Celý tento proces se děje přímo s klientem, nikoli s řízením revizního systému. Řízení dává fyzickému úložišti pouze příkazy, které konkrétnímu klientovi otevřou cestu k tomu, aby mohl zkopírování nebo nahrání souboru provést. Na zbytek komunikace již řídící systém nedohlíží a dále jen čeká na zprávy o dokončení nebo selhání operací.

V návrhu systému je počítáno s protokolem HTTP, který bude obstarávat výměnu fyzických souborů a samotné řízení bude klientovi pouze poskytovat URL adresu s jemu přístupným souborem.

Díky této vlastnosti byl jako model pro uložení fyzických dat zvolen typ popsaný v kapitole 3.1.1.3, a to mapování všech souborů na haldu.

4.2.3 Řízení systému

Řídící jednotka slouží jako hlavní koordinátor mezi všemi částmi systému a je vstupní branou pro klienta při práci se systémem. Klientovi poskytuje rozhraní pro práci se systémem. Příkazy klient do systému zadává přes řídící komunikační kanál a to formou strukturovaných dokumentů XML. Řídící jednotka se pak stará o zdárné zpracování požadavku, kde během své činnosti koordinuje všechny části systému k jejímu úspěšnému dokončení. Hlavní pracovní činnosti této jednotky jsou:

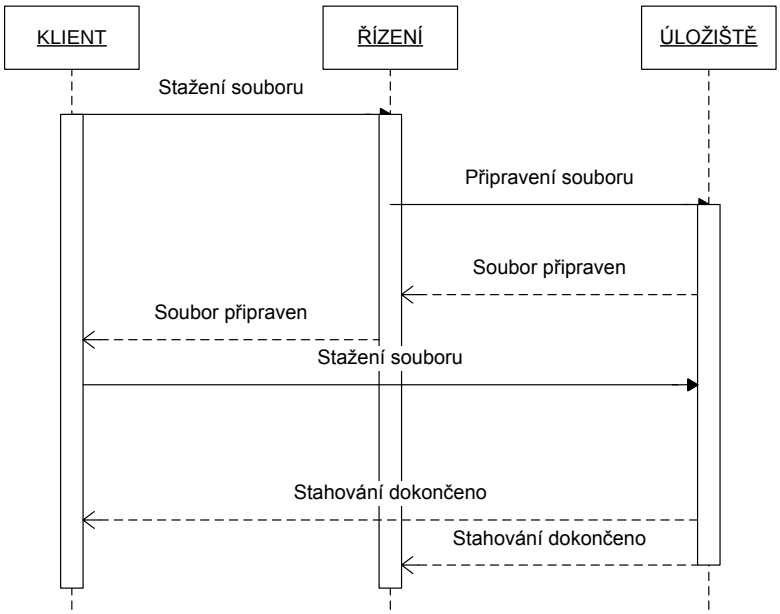
- Rozpoznání typu přijaté zprávy.
- Vytvoření požadavků směřujících k logickému uložení repositáře, tedy do vrstvy implementované na databázovém stroji.
- Koordinace přenosu souborů mezi klientem a fyzickým úložištěm (vytvoření komunikačního kanálu pro přenos).
- Zpracování požadavků jako jednu transakci. S tím související vytváření žurnálovacích záznamů o průběhu provádění transakce.
- Logika práce s repositářovým stromem a jeho efektivní sestavení s ohledem na konkrétního uživatele a jeho pracovní větev projektu.

4.2.3.1 Koordinace přenosu souborů

Jednou z hlavních funkčností řídící jednotky je koordinace přenosu souborů mezi fyzickým úložištěm a klientem. Aby bylo možné vyhnout se prostému přeposílání dat mezi úložištěm a klientem, což by značně zatěžovalo řídící jednotku, je využito principu stejného jako při metodách DMA. Řízení tedy pouze vytvoří komunikační kanál a klient se soubory pracuje přímo na fyzickém úložišti.

Schéma průběhu stažení jednoho souboru je nastíněno na Obrázek 15, který na diagramu sekvence prezentuje kooperaci řízení při přenosu souboru.

1. Na počátku celého procesu klient zašle řídící jednotce zprávu o potřebě stažení souboru.
2. Řídící jednotka vyhodnotí jeho žádost a zároveň určí konkrétní fyzický soubor, který bude přenášen. Fyzickému úložišti, ve kterém se soubor nachází, dá příkaz k otevření kanálu pro daného klienta na soubor, který požaduje.
3. Fyzické úložiště dá řízení najevo, že je vše v pořádku a je připraveno na přenos souboru. Stejnou zprávou informuje i řídící jednotka klienta s tím, že do zprávy přidává informaci o URL adrese, kde je soubor připraven.
4. Klient se připojí na získanou URL adresu a stáhne si připravený soubor.
5. Po dokončení stahování je o tomto stavu informováno řízení, které na to reaguje.



Obrázek 15 - Diagram spolupráce při stahování souboru

V případě jakýchkoli chybových stavů v přenosu souboru je o tomto opět informováno řízení, které se postará o navrácení k původnímu stavu, vzhledem k nemožnosti dokončit požadovanou transakci. Toto opatření má zásadní oporu v případech, kdy jsou do systému zaváděny změny projektu, které postihují velké množství souborů. V tomto případě je změna projektu chápána jako celek a je nutné, aby se provedla buď celá nebo vůbec.

Z předchozího se dá usoudit, že fyzické úložiště je implementováno jako autonomní služba nad protokolem HTTP, kde přístup ke spravovaným objektům je řízen z vnějšku, a to z řídícího modulu systému, který rozhoduje o realizovatelných přenosech.

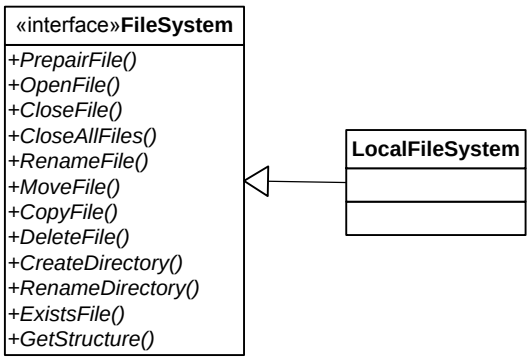
5 Implementace fyzického úložiště

Tato kapitola se zaměřuje na reálnou implementaci úložiště fyzických souborů systému a popisuje použité modely a rozhraní. Zejména se zaměřuje na popis implementace vzdáleného datového úložiště.

5.1 Schéma fyzického uložení

V implementovaném revizním systému bylo použito schéma uložení souborů, které bylo popsáno v kapitole 3.1.1.3. Jedná se tedy o formu mapování souborů na haldu. Všechny soubory, které revizní systém fyzicky spravuje jsou uloženy v jednom adresáři souborového systému, proto je nutné, aby žádné dva soubory neměli společný název. Jelikož se fyzické úložiště musí chovat jako samostatná autonomní jednotka a zároveň v metadatech musí být soubor taktéž jednoznačně definován, není možné, aby fyzické úložiště samo definovalo název (adresu) fyzického souboru. Tento problém je řešen v metadatech, která určují a jednoznačně definují názvy (adresy) jednotlivých souborů v rámci celého revizního systému.

Pro účely abstraktní správy fyzického úložiště byla vytvořena třída, která má za úkol oddělit nízko úrovňovou práci se souborovým systémem operačního systému a přizpůsobit ji požadavkům kladeným na revizní systém.



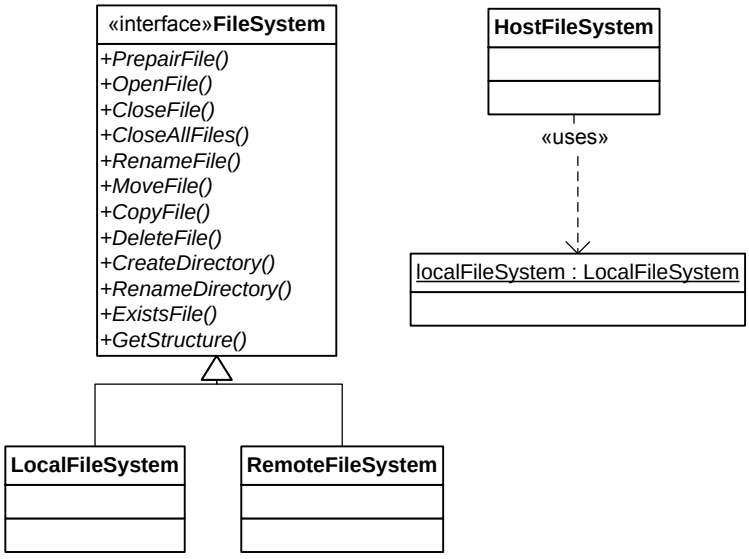
Obrázek 16 - Model třídy pro správu souborového systému

Na Obrázek 16 je znázorněno schéma třídy, která tuto funkčnost obstarává. Třída *LocalFileSystem* je implementací rozhraní *FileSystem*, které definuje všechny nutné operace pro práci se souborovým systémem. Takto definované fyzické úložiště se mapuje na souborový systém operačního systému. V tomto případě by situace mohla být taková, že instance třídy *LocalFileSystem* by se mapovala na složku C:\DATA. Následně pak všechny složky a soubory v něm uložené se tváří, jako by se nacházely v kořenovém adresáři.

Zajímavou vlastností tohoto řešení je metoda *PrepairFile*, kterou je nutné použít před každou operací otevření souboru. Tato metoda vrací jedinečný deskriptor souboru, který se klientovi na pozadí připraví a přiřadí se mu požadovaný soubor. Operace předpřipravení je zde implementována z důvodů obecného použití při asynchronní práci s úložištěm, kdy jeden proces soubor otevře pro práci a jiný s ním pracuje.

5.2 Vzdálený přístup k fyzickému úložišti

Obrázek 17 popisuje kompletní model, který je použitý při realizaci vzdáleného souborového úložiště. Základní strukturou je opět *LocalFileSystem*, kde její implementace obstarává nízko úrovňovou práci. Pro účely vzdáleného přístupu je implementována její synovská třída *RemoteFileSystem*, která má naprosto stejné rozhraní, ovšem nepracuje s lokálním úložištěm, ale tvoří převodní vrstvu ke vzdálenému úložišti.



Obrázek 17 - Model vzdáleného fyzického úložiště

Hostitelskou jednotkou pro *RemoteFileSystem* je implementace třídy *HostFileSystem*, což je vzdálená služba, která ve svém nitru ukrývá objekt lokálního souborového systému. Obě třídy *RemoteFileSystem* a *HostFileSystem* pouze vytváří komunikační kanál k objektu lokálního souborového systému ukrytého na hostující straně. Komunikace probíhá tak, že při volání metody na straně *RemoteFileSystem* jsou její parametry zabaleny do dokumentu XML. Tento dokument je přenesen hostující straně, která z něj získá potřebné parametry a typ metody, který je volán. Následně pak zavolá požadovanou metodu na objektu lokálního souborového systému.

Komunikace mezi oběma stranami je vystavěna na protokolu HTTP, kde příkazy se přenáší formou XML dokumentů. Čtení a zápis vzdálených souborů se děje stejnou cestou, ovšem již ne

v XML formátu, ale přenáší se čistá data souboru. Vzdálený souborový systém uživateli ve svém používání klade jednu překážku, kterou je nemožnost se v souboru libovolně pohybovat. Tato vlastnost je způsobena použitým protokolem HTTP, kdy je klientovi při práci se souborem otevřen datový proud, pouze pro zápis či čtení v dopředném směru.

6 Implementace komunikace

Kapitola popisuje všechny komunikační kanály, které implementovaný revizní systém využívá. Jedná se o komunikaci mezi klientem a revizním systémem a dále pak mezi jednotlivými částmi revizního systému v jeho nitru.

6.1 Klientská komunikace

Na Obrázek 13, popisujícího schéma kooperace jednotlivých částí systému a klienta a systému jako celku, je vidět, že komunikace s klientem není až tak přímočará. Klient je nucen se systémem komunikovat dvěma na sobě nezávislými kanály. Jedním z kanálů je kanál pro přenos dotazů a příkazů, případně přenos strukturovaných dat. V tomto případě se jedná o práci s metadaty, nikoli o práci s fyzicky uloženými soubory. Přenos fyzických souborů je pak zprostředkováván druhým komunikačním kanálem, přes který se přenášejí konkrétní data zpracovávaných souborů.

6.1.1 Příkazový kanál

Tímto kanálem klient ovládá chod systému a pracuje s metadaty, která spravuje. V příkazovém kanále data putují ve strukturovaném formátu ve formě XML dokumentu. Příkazový kanál je opět postaven na protokolu HTTP.

Aby obě komunikující strany hovořily naprosto stejným jazykem, je zapotřebí, aby popis protokolu byl na obou stranách stejný. Protože se data v kanále přenášejí ve strukturované formě, dají se jednotlivé příkazy popsat jim odpovídající třídou, která reflektuje strukturu příkazu přímo v kódu. Tímto mechanismem se systém tváří, jako by si mezi sebou posílal jednotlivé pracovní objekty.

Tento mechanismus je implementován pomocí XML serializace. Na jedné straně je objekt příkazu serializován do XML formátu a komunikačním kanálem odeslán druhé straně. Druhá strana provede opačný postup, tedy deserializaci a z příchozích XML dat zrekonstruuje zasílaný objekt. Tento objekt pak definuje příkaz na straně serveru, který je zde vykonán a výsledek operace je stejnou cestou odeslán klientovi.

6.1.2 Datový kanál

Datový kanál pracuje opět nad HTTP protokolem, ovšem v tomto případě se přenáší již konkrétní obsahy fyzických souborů. Tento kanál klientovi poskytuje možnost nahrání, případně stažení souboru ze systému. Princip je naprosto stejný, jako je tomu u webových prohlížečů, kde se soubor adresuje pomocí jeho URL adresy. Klient má dvě možnosti, buď požaduje zápis, případně čtení souboru na zadané adrese.

6.2 Vnitřní komunikace

Zbývající částí komunikace je komunikace mezi jednotlivými částmi systému v jeho nitru. Tato komunikace se dá opět rozdělit na dvě části. První částí je komunikace s fyzickým úložištěm souborů, která spočívá jenom v jeho řízení a nepřenáší se žádná data souborů. Druhou částí je komunikace mezi úložištěm metadat, které zároveň tvoří výkonnostní uzel systému.

6.2.1 Komunikace s fyzickým úložištěm

Komunikace s fyzickým úložištěm probíhá opět nad protokolem HTTP ve strukturovaných XML dokumentech. Řídící jednotka celého systému využívá z fyzického úložiště hlavně příkaz *PrepairFile* (z popisu v kapitole 5.1), který klientům požadujícím práci se souborem otevírá komunikační kanál.

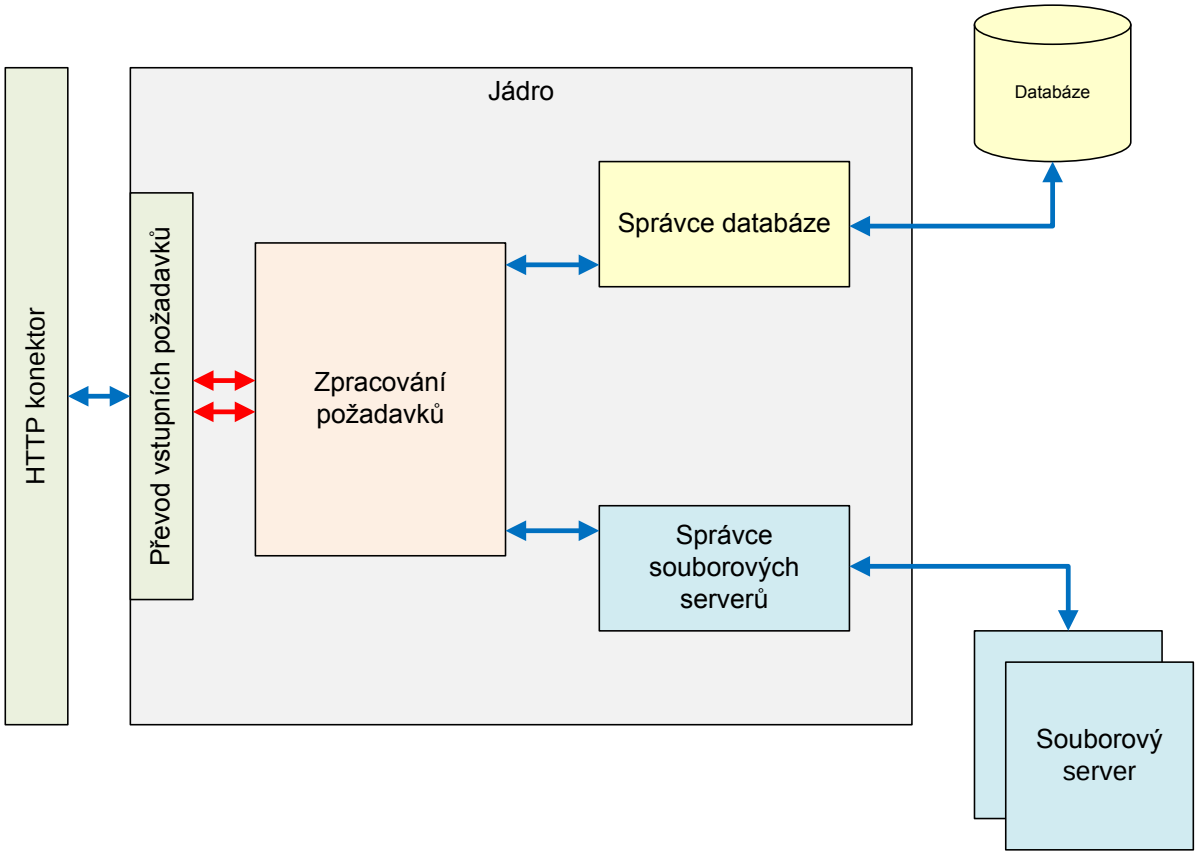
6.2.2 Komunikace s úložištěm metadat

Jelikož je úložiště metadat implementováno na databázovém serveru MS SQL 2005, je komunikace řešena konektorem k této databázové platformě. Celá logika je zde implementována za pomoci uložených procedur a klient s databází nemůže pracovat jinak, než prostřednictvím těchto procedur. Komunikace je tudíž odkázána na volání uložené procedury s přenosem parametrů a zpětnou vazbou v podobě návratové hodnoty procedury.

7 Implementace řídící části systému

Řídící část revizního systému má za úkol koordinovat všechny procesy, které plynou z podstaty správy repositáře. Zastává funkci prostředníka a dá se říci, že je převodní vrstvou mezi úložištěm metadat a klientem. Zároveň koordinuje přenosy fyzických souborů směrem ke klientovi a naopak. V neposlední řadě v sobě implementuje některé algoritmy logiky správy revizí, které by bylo velmi obtížné na databázovém prostředí naimplementovat. Hlavním představitelem je algoritmus pro rozdílové porovnávání *n*-árních stromů, který vytvoří seznam rozdílových záznamů. O této problematice podrobně pojednává kapitola 8.2.

7.1 Schéma architektury

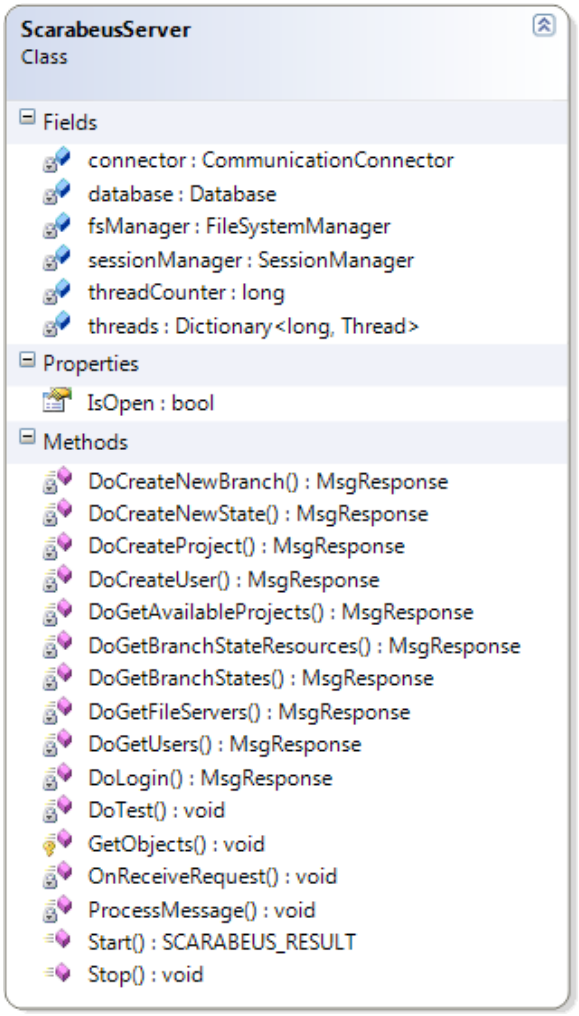


Obrázek 18 - Schéma architektury systému

Obrázek 18 popisuje architekturu celého systému a návaznosti mezi jeho jednotlivými částmi. Šipky definující datové toky mezi jednotlivými částmi systému zároveň naznačují prováděcí větvení procesů. Větvení do prováděcích vláken se děje pouze mezi jednotkami pro příjem vstupních požadavků a jednotkou, která tyto požadavky zpracovává. Tedy společným krokem je pouze přijetí

příchozího kontextu na síťové rozhraní a samotné rozpoznání zprávy a její zpracování je již prováděno ve vlastním prováděcím vlákně tak, aby nebyly odmítány požadavky ostatních klientů.

7.1.1 Jádro systému



Obrázek 19 - Schéma jádra

Schéma třídy znázorněné na Obrázek 19 představuje vlastní prováděcí jádro celého systému. Vytvořená instance této třídy obsahuje konektor pro příjem vnějších požadavků klientů (v třídě je definován jako objekt *connector*).

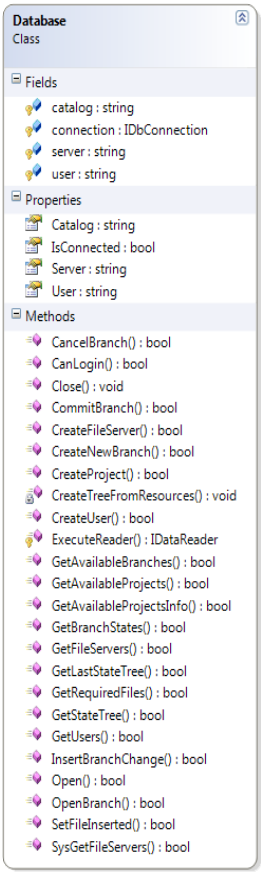
Dále pak obstarává nativní připojení k databázovému serveru, které je implementováno uvnitř třídy *Database* a provádí překlad argumentů volaných tříd do podoby argumentů uložených procedur na databázovém serveru.

Velmi podstatnou část tvoří správce souborového systému *fsManager*, který obstarává komunikaci se všemi připojenými fyzickými úložišti. Tento prvek zachytává zpětnou vazbu od fyzických úložišť po dokončení klientské operace (nahrání nebo stažení souboru).

Díky víceuživatelskému přístupu není možné požadavky zpracovávat sekvenčně, proto všechny najednou vykonávané úkony běží v jádře paralelně. Jádro se tedy musí dokázat vyrovnat se situací, kdy klient požádá o zastavení činnosti systému a očekává, že i při vysokém zatížení neskončí v nekonzistentním stavu, díky rozpracované činnosti prováděcích vláken. Pro ochranu před tímto možným stavem si jádro dokáže hlídat všechny spuštěné procesy a při ukončování systému postupně dokončit všechnu rozdělanou práci.

Z pohledu na třídu jádra je vidět množství metod začínajících písmeny *Do*. Tyto metody slouží ke konkrétnímu spouštění příkazů přicházejících z klientské strany. Při příchodu klientského požadavku je rozpoznám příslušný požadovaný příkaz a zavolána mu odpovídající metoda. Vykonávací metoda běží na úrovni jádra a má tudíž kompletní přístup ke všem rozhraním, které jí jádro poskytuje (databáze, souborové systémy).

7.1.2 Správa databáze



Obrázek 20 - Schéma databázového správce

Na Obrázek 20 je znázorněno schéma třídy, která tvoří obálku nad celým databázovým úložištěm. Tato třída ve svém veřejném rozhraní implementuje metody, které přesně korespondují s uloženými

procedurami na straně databázového serveru. Tato třída tedy slouží pouze jako převodní most a umožňuje vykonávat požadované operace v jiném výpočetním uzlu.

7.2 Správa uživatelů

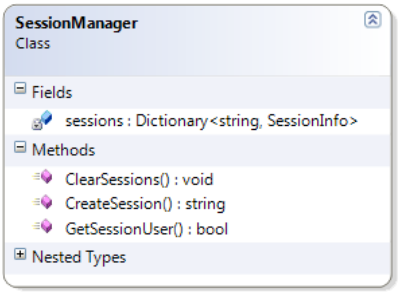
Správa uživatelů v systému je založena na modelu uživatelských práv a oprávnění. Právy se rozumí povolené/zamítnuté operace ve vztahu uživatel-systém. V případě implementovaného revizního systému se jedná o práva pro:

- **Správu uživatelů** – uživatel může vytvářet nové uživatele, měnit jejich vlastnosti apod.
- **Správu projektů** – uživatel může vytvářet projekty, měnit jejich vlastnosti a přiřazovat k projektům oprávněné uživatele
- **Správu souborových serverů** – umožňuje do systému přidávat nové připojení k souborovým serverům

Pod pojmem oprávnění je zde vnímán vztah uživatel-část systému. V případě implementovaného revizního systému se jedná hlavně o objekt projektové větve, ke které je uživatel přiřazen.

Všechna práva a oprávnění jsou hlídána na nejnižší úrovni, a to přímo na databázovém serveru, který na základě práv a oprávnění požadované operace provádí. Ovšem od jádra systému očekává, že obdrží informaci o jménu uživatele, který o vykonání operace žádá.

Jelikož je HTTP bez stavový protokol, musí jádro systému udržovat informaci o již přihlášených uživatelích. K tomuto účelu slouží implementace třídy *SessionManager*, která udržuje informace o všech vytvořených připojeních a jejich vlastnících.



Obrázek 21 - Schéma správce uživatelů

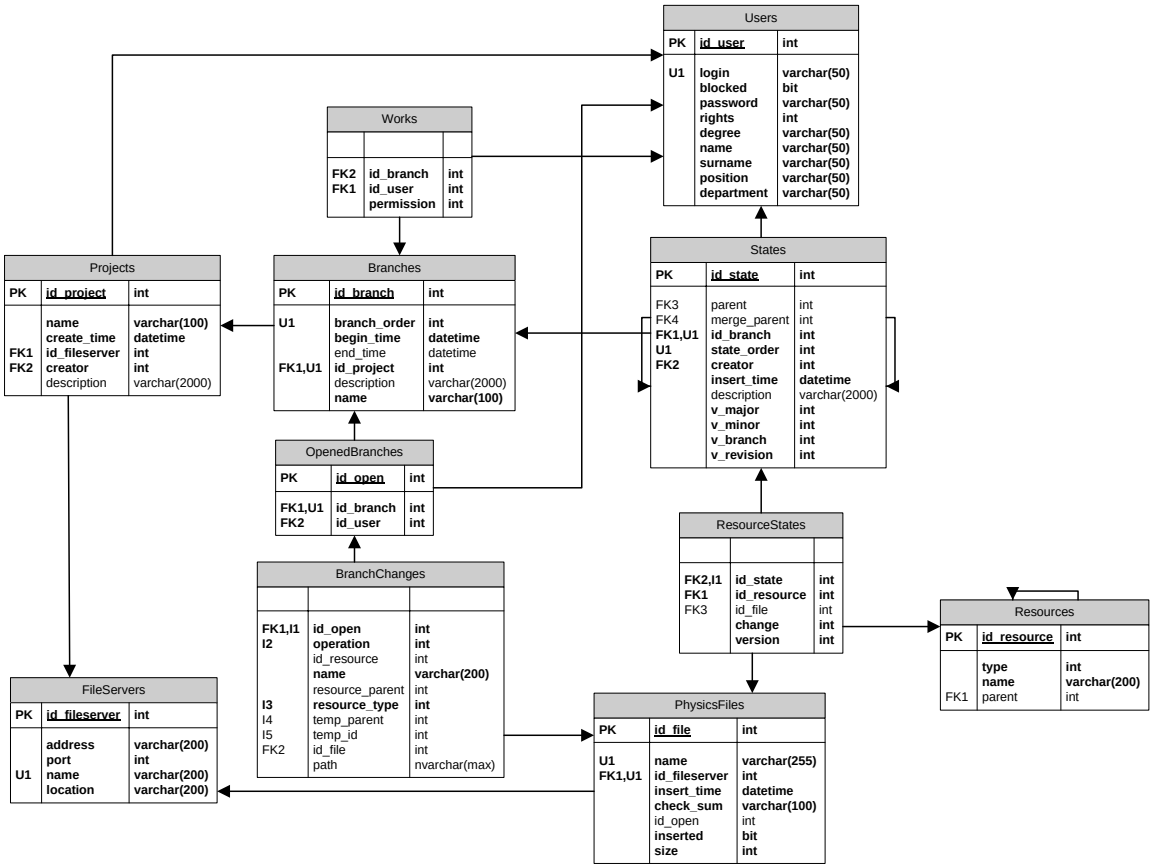
Při připojení uživatele k systému je zapotřebí, aby vlastnil systémem vygenerované a jedinečné číslo sezení. Toto číslo sezení je vygenerováno a uživateli poskytnuto vždy při operaci přihlášení do systému a dále komunikuje pouze prostřednictvím tohoto čísla. Při příchodu jakýchkoli příkazů bez platného čísla sezení systém okamžitě takovou relaci ukončuje.

8 Implementace uložení metadat

Jak již bylo v několika předchozích kapitolách nastíněno, metadata jsou v implementovaném revizním systému uložena v relační databázi. Aplikační logika, která nad relačními daty pracuje je taktéž uložena na databázovém serveru v podobě uložených procedur. Uložené procedury vytváří rozhraní pro práci s daty pro zbývající část systému. Žádná část systému nemůže k datům přistupovat přímo (prostřednictvím SQL dotazů), ale pouze prostřednictvím těchto procedur. Tímto je do velké míry zajištěna kontrolovatelnost operací uživatelů a je docíleno toho, že celé logické úložiště se v systému chová jako jedna uzavřená černá skříňka.

8.1 Schéma uložení metadat

Stavebním kamenem repositářové struktury je objekt projektu. Projekt je podle teorie dále možné členit na vývojové větve, které jsou obrazem stavu projektu v jeho jednotlivých vývojových fázích. Každá větev ve vývoji projektu tvoří lineárně svázaný seznam stavů větve, ve kterých se projekt během vývoje nacházel. Stavem větve je tedy rozuměna stromová struktura projektu, ve které se nachází všechny soubory a složky. Stav větve neobsahuje kompletní informace o podobě projektového stromu, nýbrž pouze změny, kterých projektový strom dostál od poslední revize. Při vytváření obrazu stavu projektového stromu je tedy nutné vycházet pouze ze změn a na základě nich sestavit odpovídající stromovou strukturu. Projektová stromová struktura obsahuje obecně zdroje, kterými v jemnějším pohledu mohou být soubory či složky. Zatímco složky tvoří pouze uzly a člení projekt do stromu, tak soubory tvoří listy tohoto stromu. Soubory se ovšem během vývoje projektu mohou často měnit a výsledkem je, že každý soubor se může nacházet ve více vývojových stavech. Tyto stavy opět tvoří chronologicky seřazený seznam. Stav každého souboru pak vytváří odkaz na jeden fyzický soubor uložený v repositáři. Tedy repositář obsahuje tolik fyzických souborů, kolik činí součet všech stavů všech zdrojů typu soubor.



Obrázek 22 - Schéma uložení metadat

Tabulky relací mají ve schématu databáze následující význam:

- **Projects –** obsahuje informace o všech projektech, které se v systému nacházejí
- **Branches –** vývojové větve projektů
- **States –** stavy jednotlivých větví
- **ResourceStates –** informace o změnách provedených odpovídajícím stavu
- **Resources –** zdroje vyskytující se v projektech
- **PhysicsFiles –** informace o uložení fyzických souborů v repositáři
- **FileServers –** seznam dostupných souborových serverů, ve kterých jsou uloženy fyzické soubory
- **User –** uživatelé, kteří mohou k systému přistupovat
- **Works –** přiděluje uživatelům oprávnění pro práci s vývojovou větví
- **OpenedBranches –** seznam otevřených vývojových větví, u kterých uživatel zažádal o operaci *commit*
- **BranchChanges –** seznam změn, které se mají provést s otevřenou větví s cílem vytvoření nového stavu větve

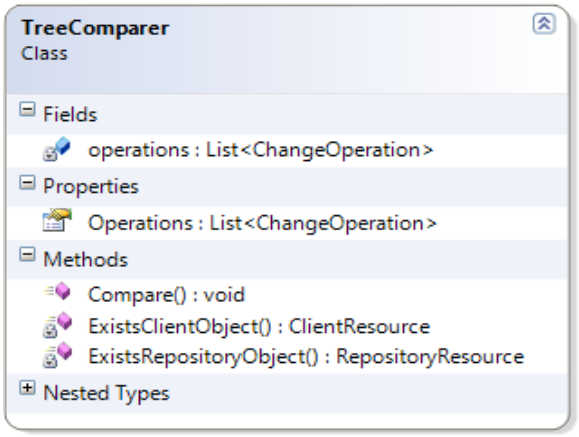
8.2 Algoritmus pro porovnávání stromů

Při vytváření nového stavu projektové větve je stěžejním problémem získat informaci o tom, jakých změn dostál původní strom vůči nově vkládanému stromu.

Mezi jednotlivými stavy větví je zaveden vztah dědění, ovšem ne pouze s klasickou operací přidávání, jak je tomu u modelu spousty objektových programovacích jazyků, ale je zde při dědění mezi stavy povolena i operace změny a odebrání.

V úložišti metadat jsou uloženy pouze změny, které vznikly mezi oběma stavy. Kompletní podoba stromu se pak rekonstruuje pouze z těchto změnových informací. Tento algoritmus je popsán v kapitole 0.

Pro účely vytvoření změnových záznamů slouží implementace třídy *TreeComparer*. Algoritmus, v ní implementovaný, porovnává souborový strom získaný z klientské kopie, který v projektové větvi vytváří nový stav, vůči poslednímu vloženému stavu v projektové větvi. Výsledkem operace porovnání je seznam změn, kterých dostál poslední stav větve do vkládaného stavu.



Obrázek 23 - Třída pro porovnávání stromů

Algoritmus porovnává vždy dvě sobě odpovídající si složky proti sobě a vyhledává v nich změny. Porovnávání probíhá postupně celým projektovým stromem, kdy se zanořuje do všech existujících složek na obou stranách. Vždy nastane jedna z možných situací, a to buď existuje složka na straně klientského stromu, nebo na straně repositáře, případně existují obě.

V případě, že existuje složka na straně repositáře a ne na straně klientské, pak všechny existující objekty v repositářovém stromě, které se nacházejí v podstromu, kde kořenem je odpovídající repositářová složka, musí být odstraněny.

Druhý případ je naprosto opačný. Existuje složka na straně klientského stromu a neexistuje na straně repositáře. V tomto případě musí být do repositářového stromu přidány všechny položky z podstromu, kde kořenem je odpovídající klientská složka.

Posledním případem je situace, kdy existují obě vzájemně si odpovídající složky. V tomto případě se sekvenčně porovnávají všechny objekty. Při existenci adresáře na jedné či druhé straně se provede zanožení do této složky. Složky tvoří dělící uzly celého stromu. V případě souborů se při existenci na obou stranách porovnávají hash kódy obou souborů a v případě jejich neshody, se inkrementuje číslo verze souboru v repositáři. V případě neexistence na jedné z porovnávaných stran se soubor buď odebere nebo přidá do repositářového stromu.

8.2.1 Popis rozdílových záznamů

Algoritmus vytváří seznam změnových záznamů. Jedná se o záznamy, které říkají, jaké změny se mají provést s repositářovým stromem, aby bylo docíleno přesné podoby klientského stromu. Každý záznam obsahuje následující položky:

- **Operace** – typ operace, ADD nebo REMOVE.
- **ID objektu** – odpovídá již existujícímu objektu na straně repositáře. Číslo je vyplněno v případě přidávání nové verze souboru, nebo odstranění existujícího.
- **Název** – název objektu.
- **ID rodiče** – odpovídá již existujícímu rodičovskému objektu na straně repositáře.
- **Typ objektu** – FILE nebo FOLDER.
- **TempID rodiče** – dočasné identifikační číslo rodiče. Toto číslo neodpovídá žádnému již existujícímu objektu na straně repositáře a uplatňuje se u nově přidávaných objektů, které jsou samy zanořeny v neexistujícím objektu.
- **TempID objektu –** dočasné identifikační číslo objektu. Uplatňuje se u všech nově přidávaných objektů, které nemají svůj obraz v již existujících repositářových objektech.

	Operace	ID obj.	Název	ID rod.	Typ	TempID rod.	TempID obj.
1	ADD	číslo	xxx	číslo	FILE	-	-
2	REMOVE	číslo	xxx	číslo	FILE	-	-
3	REMOVE	číslo	xxx	číslo/-	FOLDER	-	-
4	ADD	-	xxx	číslo/-	FILE	číslo/-	číslo
5	ADD	-	xxx	číslo/-	FOLDER	číslo/-	číslo

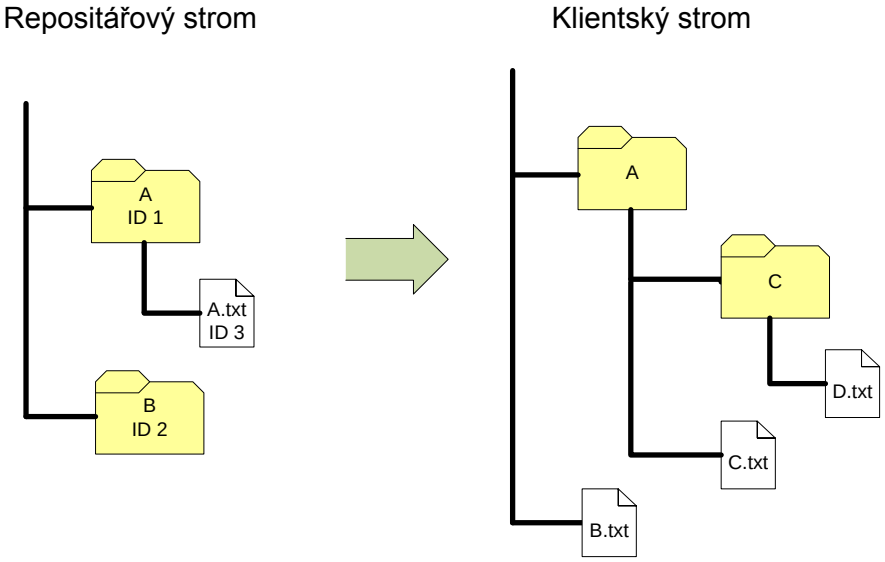
Tabulka 1 - Typy rozdílových záznamů

V Tabulka 1 je seznam všech možných kombinací rozdílových záznamů, kde jejich význam je následující:

1. Situace, kdy soubor existuje v obou porovnávaných složkách a zároveň je klientský soubor odlišný od dříve uloženého v repositáři. Jedná o situaci inkrementace verze již existujícího souboru.
2. Situace, kdy soubor existuje na straně repositáře, ale chybí na klientské straně. Záznam tedy soubor z repositáře odstraňuje.
3. Odstranění složky z repositáře. Složka existuje na straně repositáře, ale neexistuje na straně klienta. Položka *ID rod.* Může nabývat dvou hodnot, a to buď bude vyplněno ID rodičovské složky, kde se odstraňovaná nacházela, nebo bude prázdná a tudíž rodičovskou složkou je kořenová složka.
4. Popisuje přidání nového souboru, který nemá svůj ekvivalent v repositáři. Položka *ID obj.* není vyplněna, protože neexistuje repositářový ekvivalent. Položka *ID rod.* Může být buď konkrétní identifikační číslo existujícího repositářového objektu, nebo může být nevyplněno. V případě, že je nevyplněno, mohou nastat dvě situace, které závisí na položce *TempID rod.*. Jestliže je tato položka nevyplněna, pak je to známka toho, že se přidávaný soubor nachází v kořenové složce. V opačném případě je přidávaný soubor zanořen ve složce, která taktéž v repositáři neexistuje a pak *TempID rod.* odpovídá *Temp ID obj.* odpovídajícího rodiče.
5. Poslední možná situace je naprosto shodná jako předchozí, pouze pracuje s objektem složky.

8.2.2 Příklad vytváření rozdílových záznamů

Na Obrázek 24 je znázorněna možná situace, kdy algoritmus musí vytvořit seznam změnových záznamů, které převedou repositářový strom do podoby klientského stromu.



Obrázek 24 - Příklad převodu stromů

Výsledný seznam bude vypadat následovně:

Operace	ID obj.	Název	ID rod.	Typ	TempID rod.	TempID obj.
REMOVE	2	B	-	FOLDER	-	-
ADD	-	B.txt	-	FILE	-	1
REMOVE	3	A.txt	1	FILE	-	-
ADD	-	C	1	FOLDER	-	2
ADD	-	C.txt	1	FILE	-	3
ADD	-	D.txt	-	FILE	2	4

Tabulka 2 - Příklad použití rozdílových záznamů

V Tabulka 2 je vidět seznam záznamů, které vznikly při porovnávání obou stromů. Následně pak systém musí při vkládání nového stavu repositářového stromu do pracovní databáze doplnit skutečná identifikační čísla u nově přidávaných objektů. K tomu, aby měl k dispozici informaci o vazbách mezi objekty poslouží vazby definované dočasnými identifikačními čísly. Podrobně je tento postup popsán v kapitole 8.4.

8.3 Algoritmus pro sestavení projektového stromu

Algoritmus pro sestavení projektové stromu je plně implementován na databázovém serveru a využívá se při každém požadavku vedoucímu k získání stromové struktury projektu v požadovaném stavu projektové větve.

Algoritmus při své práci využívá tří databázových tabulek, a to *States*, *ResourceStates* a *Resources*. Algoritmus je parametrizován identifikačním číslem větve a číslem stavu v této větvi. Výsledkem je pak seznam zdrojů, které odpovídají obsahu projektového stromu v požadovaném stavu větve. Práce algoritmu je následující:

1. Vytvoř **seznam zdrojů** ve výsledném stromu
2. **Načti** seřazený **seznam stavů** požadované větve, konče požadovaným stavem
3. Je-li seznam stavů prázdný, pokračuj krokem 13
4. Vezmi stav s nejnižším číslem
5. Načti **seznam změn** v tomto stavu
6. Je-li seznam změn prázdný, pokračuj krokem 12
7. Vezmi první **změnu**
8. Jedná-li se o přidání složky **přidej** zdroj **do** výsledného **seznamu zdrojů**
9. Jedná-li se o přidání souboru, zkontroluj, zda-li již soubor ve výsledném seznamu existuje
    a. Existuje-li, **nahraď** jeho **číslo verze** číslem obsaženým ve změně
    b. Neexistuje-li, **přidej** jej **do** výsledného **seznamu zdrojů**
10. Jedná-li se o smazaní zdroje, **odstraň** jej z výsledného **seznamu zdrojů**
11. **Odstraň změnu** ze seznamu načtených a pokračuj krokem 6
12. **Odstraň stav** se seznamu stavů a pokračuj krokem 3
13. Jako **výsledek** vrať seznam zdrojů

Algoritmus tedy postupně zpracovává všechny změny, kterých dosáhly jednotlivé stavy. Prochází všechny stavy od nejnižšího, až po požadovaný a zaznamenává zdroje do výsledné tabulky zdrojů. Po dokončení algoritmu výsledek obsahuje i vazby otec-syn, na základě kterých je možné v aplikaci sestavit odpovídající stromovou reprezentaci.

8.4 Implementace operace *commit* nového stavu projektu

Operace *commit* představuje akci, která se provádí při zavádění změn projektové větve. Tato operace v projektové větvi vytváří nový stav, který je následovníkem posledního stavu revidované větve. Operace *commit* musí zabezpečit, aby vytvoření nového stavu bylo atomické a celý repositář se neměnil postupně, nýbrž najednou.

V implementovaném revizním systému tato operace představuje jednu transakci, která pro svou práci vyžaduje všechny prvky systému a není odkázána pouze na databázové úložiště, kde by její implementace byla jednoduchá. Nicméně v databázovém úložišti probíhá zásadní změna, a to změna metadat popisujících strukturu repositáře. Zavedení změn v databázi je však možné provést až v době, kdy proběhly všechny potřebné akce s fyzickými soubory. Postup zavedení nového stavu je následující:

1. Klient systému odešle požadavek o vytvoření nového stavu vybrané projektové větve. Součástí požadavku je i struktura projektového stromu v klientském lokálním úložišti. Na klientské straně je nutné pro každý fyzický soubor vytvořit kontrolní součet, který je součástí informací o fyzickém souboru.
2. Systém požádá databázové úložiště o otevření transakce pro práci s vybranou projektovou větví. Databázové úložiště transakci otevře tak, že vytvoří nový záznam v tabulce *OpenedBranches*, který je svázán s požadovanou větví. V případě, že požadovaná větev je již otevřena, je tento požadavek zamítnut.
3. Systém načte strukturu projektového stromu posledního stavu požadované větve, na který bude nový stav navazovat, využívá přitom algoritmu popsaného v kapitole 8.3. Provede výpočet rozdílových záznamů mezi stromovou strukturou obdrženou od klienta a načtenou.
4. Všechny takto vytvořené změnové záznamy jsou iteračně nahrány do databázového úložiště, konkrétně do tabulky *BranchChanges*, kde každý záznam je svázán s otevřenou transakcí nad měněnou větví.
5. Po zavedení informací o všech změnách do databáze, je již možno získat informaci o souborech, které klient musí do systému nakopírovat ze svého lokálního úložiště. Tento seznam souborů se načte z databázového úložiště. Pro všechny požadované soubory se na odpovídajícím souborovém serveru otevře komunikační kanál. Klientovi je následně zaslán seznam se všemi potřebnými soubory, kde informace o souboru obsahuje cestu v jeho lokálním úložišti a přesnou URL adresu, kde má být tento soubor nahrán.
6. Dále systém čeká, až klient do souborového systému nahraje všechny požadované soubory. Při zavedení informací o požadovaných souborech do databáze, má každý soubor nastaven

výchozí atribut *inserted* v tabulce *PhysicsFiles* na hodnotu *false*. V době, kdy klient soubor úspěšně do souborového systému nahraje je ihned o tomto stavu informováno jádro řízení, které konkrétnímu souboru změní atribut *inserted* na *true*. Při každé takové operaci je kontrolováno, zda jsou již ve fyzickém úložišti uloženy všechny požadované soubory běžící transakce. Je-li tomu tak je spuštěna operace vytvoření nového stavu na databázovém úložišti.

7. Do tohoto okamžiku transakce probíhala jako kooperace mezi všemi částmi systému, ale nyní je již celé provádění zakončeno pouze na databázovém úložišti zavedením všech změn starého stavu a vytvořením nového. Celá následující akce probíhá, jako jedna databázová transakce.
8. Vytvoří se nový stav projektové větve, přidáním záznamu do tabulky *States*. Nový stav je následníkem posledního vloženého.
9. Do tabulky *ResourceStates* se zavedou všechny záznamy, které odpovídají operaci REMOVE existujícího zdroje v projektovém stromu.
10. Přidají se všechny záznamy, které odpovídají přidání nové verze již existujícího souboru.
11. Nyní je nutné vytvořit všechny nové složky. Složky se musí vytvářet od kořenové, protože žádná z nich nemá ještě v metadatech své pevné místo a při vytváření zdroje je nutné vkládat vazbu na již existujícího rodiče. Algoritmus tedy pracuje s dočasnými vazbami reprezentovanými atributy *temp_parent* a *temp_id* v tabulce *BranchChanges*. Při vytvoření nové složky je okamžitě načteno jeho skutečné identifikační číslo a to je u všech podřízených zdrojů vyplněno v atributu *resource_parent*. Při vytváření synovského zdroje se již systém nemusí orientovat pomocí dočasných identifikačních čísel, ale má již k dispozici konkrétní zdroj.
12. V tomto kroku se již vytvoří zbývající soubory, které jsou v projektové stromu nové. Jelikož rodičem souboru může být pouze složka, jsou v tomto okamžiku všechny složky již vytvořeny a každý nově vytvářený soubor má již k dispozici informaci o identifikačním čísle rodičovské složky. Vytváření nových souborů je tedy přímočaré.
13. Nyní jsou již všechny změny projektové větve úspěšně zavedeny v tabulce *ResourceStates* a je vytvořen nový stav. Dále je třeba odstranit všechny záznamy o otevřené transakci z tabulek *BranchChanges* a *OpenedBranches*. Po provedení odstranění se zavedou všechny změny provedené v transakci a tímto je celá práce dokončena a nový stav je již přístupný klientům ke čtení, případně větev je připravena pro další možné změny.

9 Testování výkonnosti

Výkonnost implementovaného systému byla porovnávána proti reviznímu systému SVN ve verzi 1.4.6. Tento revizní systém byl vybrán z důvodu jeho velmi časté praktické implementaci při správě projektové dokumentace. Při výkonnostních testech byla snaha o co nejpodobnější nastavení obou systémů tak, aby se při testu porovnávaly ekvivalentní operace. Testy se zaměřují na práci s velkými soubory, velkým množstvím malých souborů a na konkurentní práci dvou klientů.

Při testování se měřila celková doba zpracovávání operace. K měření uplynulého času byla použita utilita pro sledování stavu PC (Reliability and Performance Monitor), která je součásti operačního systému Windows Vista. Při měření bylo sledováno vytížení procesoru a vstupně/výstupní operace při práci s diskem. Nástroj umožňuje měřit zmíněné vlastnosti přímo pro vybraný proces, proto byl tento nástroj připojen na všechny běžící procesy, které jsou při provádění operací aktivní (databázový server, atd.).

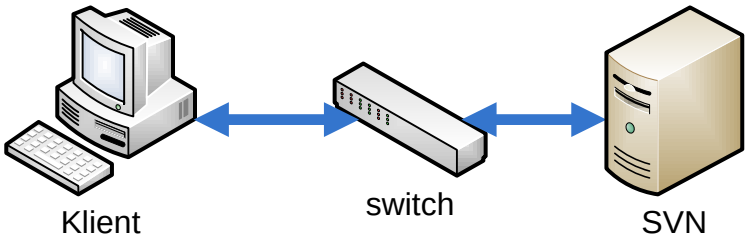
Při všech testech bylo spojení mezi počítači realizováno po 100 Mb/s rychlé síti.

9.1 Jednouživatelské prostředí

Tyto testy se zaměřují na testování výkonnosti v jednouživatelském prostředí s přístupem pouze k jednomu projektu.

9.1.1 Zapojení

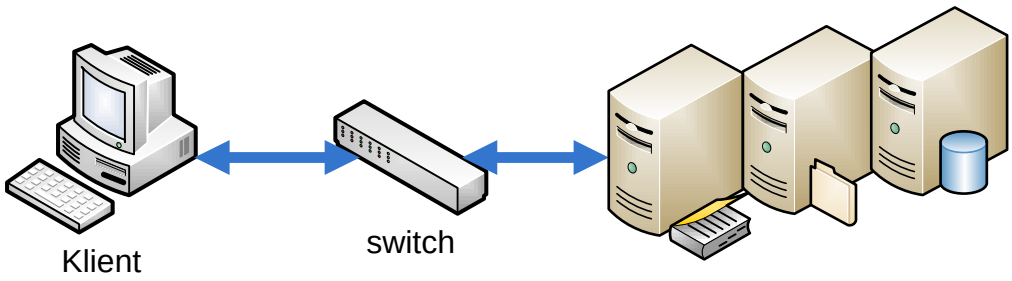
Tyto testy spočívají v porovnání výkonnosti při běžné práci, kdy se systémem pracuje pouze jeden klient. Obrázek 25 znázorňuje, v jakém prostředí bylo testování provedeno. Klient a repositář SVN jsou umístěny na různých počítačích, propojených sítí.



Obrázek 25 - Schéma zapojení při testování systému SVN

Zapojení s implementovaným revizním systémem bylo provedeno stejným způsobem. Jelikož testovaný systém se skládá ze tří částí, byly všechny umístěny na jednom fyzickém počítači, tak aby

bylo využíváno stejných zdrojů jako v případě zapojení s revizní systémem SVN. Obrázek 26 popisuje testované zapojení.



Obrázek 26 - Zapojení testovaného revizního systému

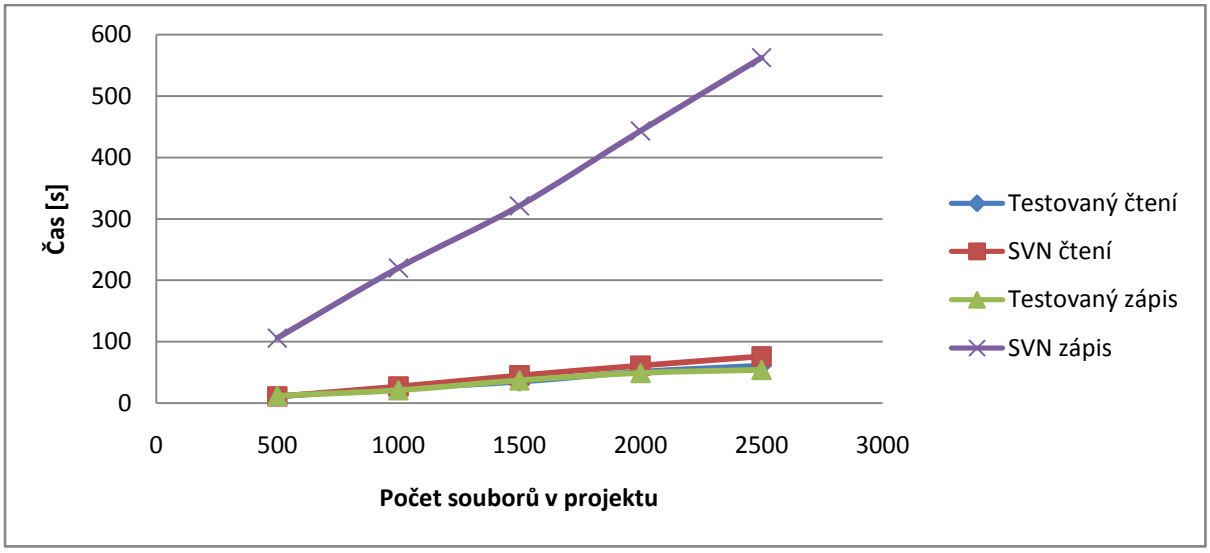
9.1.2 Výsledky

	Soubory		Čtení [s]		Zápis [s]	
	Počet souborů	Průměrná velikost [kB]	Testovaný	SVN	Testovaný	SVN
A_1	500	3,6	11	11	12	106
A_2	1000	3,6	24	27	21	220
A_3	1500	3,6	35	45	37	321
A_4	2000	3,6	51	61	49	443
A_5	2500	3,6	61	76	54	562
B_1	50	9010	58	434	57	4176
B_2	100	9010	103	837	97	8060
B_3	150	9010	175	1234	155	11891
B_4	200	9010	215	1630	202	15700
B_5	250	9010	288	2021	271	19474

Tabulka 3 - Výsledky jednouživatelských testů

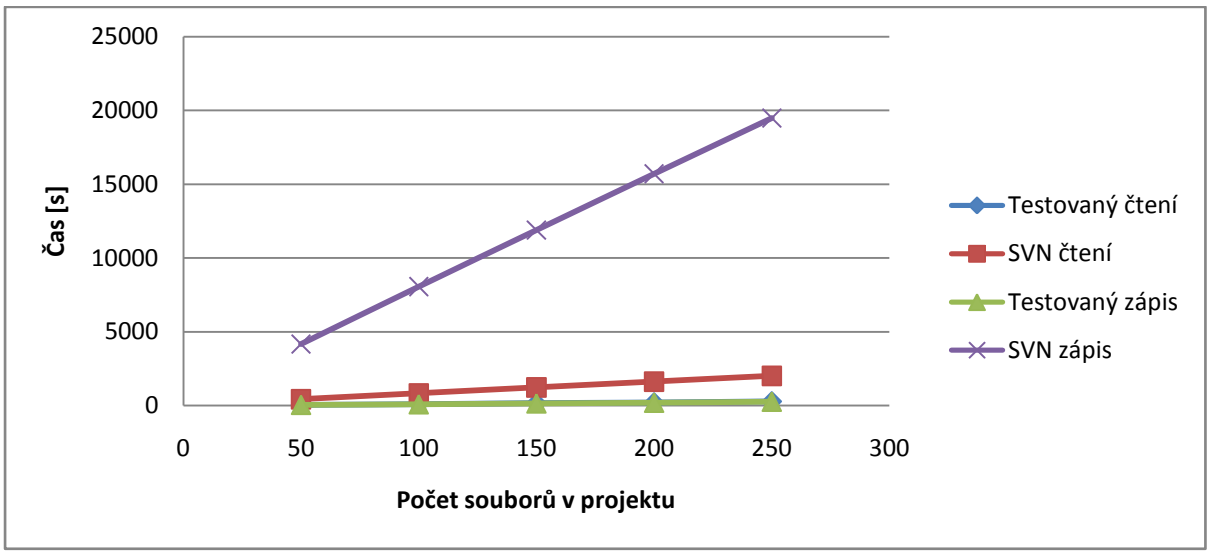
Prvním testovaným případem bylo porovnání propustnosti systému při práci s velkým množstvím malých souborů. V Tabulka 3 jsou tyto varianty označeny jako $A_{1\text{-}5}$. Tento test by měl ve velké míře připomínat reálné softwarové projekty, ve kterých soubory nedosahují závratných velikostí, ale jejich počet může být vysoký.

V druhém případě se jednalo o test práce s relativně velkými soubory, v Tabulka 3 označeny jako $B_{1\text{-}5}$. Tento test by měl navozovat situaci při práci s multimediálními daty, například obrázky.



Obrázek 27 - Graf výkonnosti při schématu A

Graf na Obrázek 27 vykresluje výslednou situaci, ze které je patrné, že operace čtení je v případě obou testovaných systémů rovnocenná. Opačným případem je operace zápisu, kdy revizní systém SVN potřebuje značně větší dobu na vykonání stejné operace. Toto chování je způsobeno tím, že testovaný revizní systém s vkládanými soubory neprovádí žádné další operace a soubory jsou jenom přeneseny přes síť.



Obrázek 28 - Graf výkonnosti při schématu B

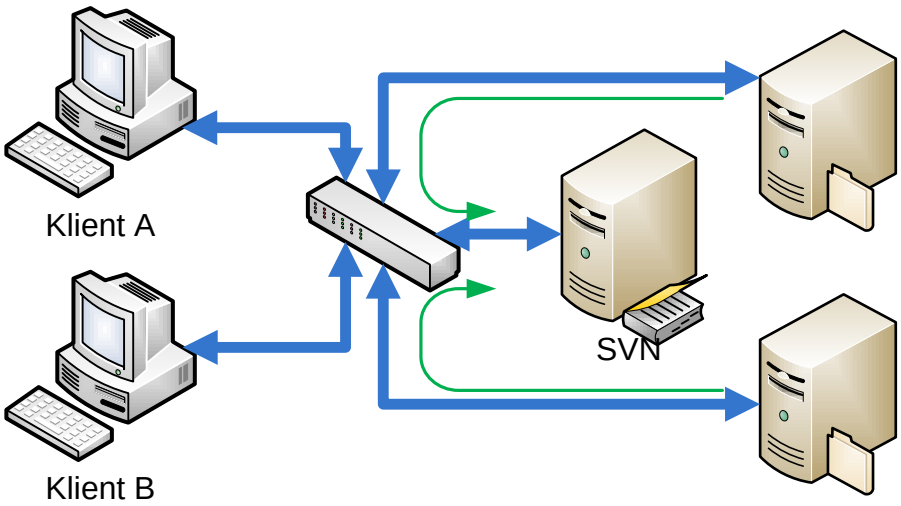
Graf na Obrázek 28 vykresluje situaci při práci s menším množstvím větších souborů. Výsledek je téměř stejný jako v případě schémat $A_{1\text{-}5}$. Zápisové operace budou u systému SVN vždy výrazně pomalejší, neboť SVN vytváří rozdílové soubory, kdežto testovací systém nikoli a jedinou režií je přenos souborů.

9.2 Víceuživatelské prostředí

Tyto testy jsou zaměřeny na konkurenční práci dvou klientů vůči jednomu reviznímu systému. Každý klient pracuje se svým projektem. Pro jednoduchost mají oba projekty stejný obsah, aby výtěžnost každého projektu byla stejná. Každý projekt je uložen v jednom fyzickém úložišti.

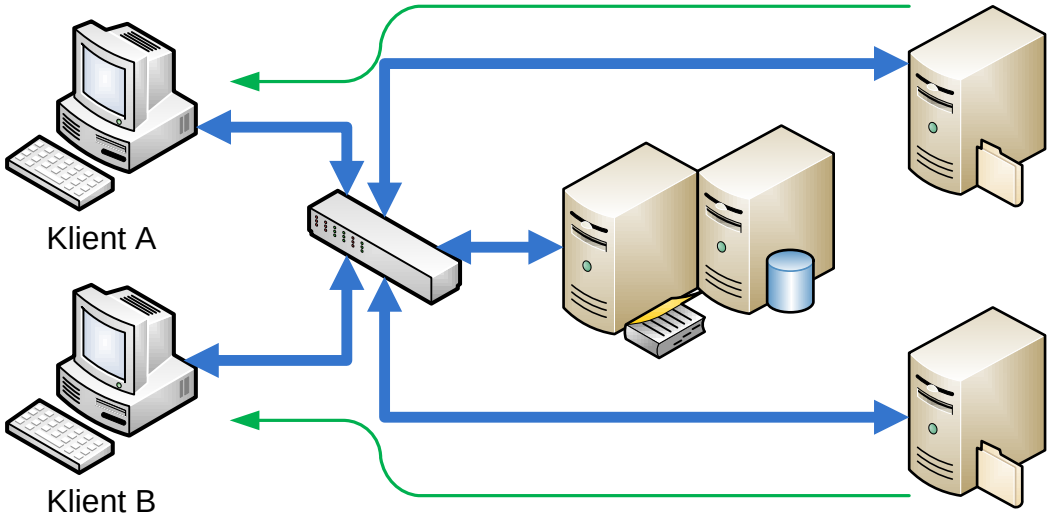
9.2.1 Zapojení

V prvním případě je k dispozici jeden SVN server, který spravuje dva nezávislé projekty. K SVN serveru se připojuje z jednoho bodu, ale projekty jsou rozmístěny na vzdálených úložištích. Toto vzdálené úložiště je vytvořeno pomocí síťového sdílení, kde se vytvoří disková jednotka, která je mapována na složku, která je sdílena vzdáleným počítačem. Na takto vytvořené disky se vloží repositáře testovaných projektů. Situaci popisuje zapojení znázorněné na Obrázek 29.



Obrázek 29 - Distribuované zapojení SVN

V druhém případě je zapojení stejné. Vlastností testovaného systému je fakt, že se data z fyzického úložiště nepřenáší k uživateli přes centrální výpočetní uzel, ale přímou cestou. Naproti tomu u systému SVN budou všechna data zpracovávána serverem a až poté odeslána klientovi. V tomto případě je patrné, že z SVN budou všechna data přenášena pouze jednou linkou, kdežto v případě testovaného systému dvěma nezávislými linkami, znázorněno na Obrázek 30. Dále pak systém SVN bude větší měrou zatěžovat centrální uzel, který se musí o přeposílání dat postarat.

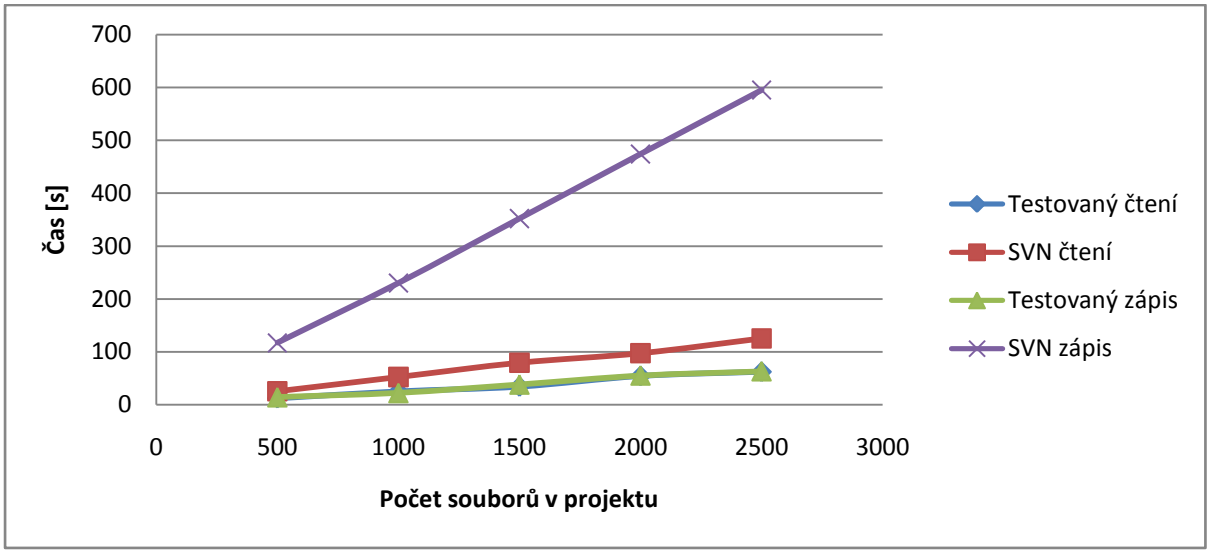


Obrázek 30 - Distribuované zapojení testovaného systému

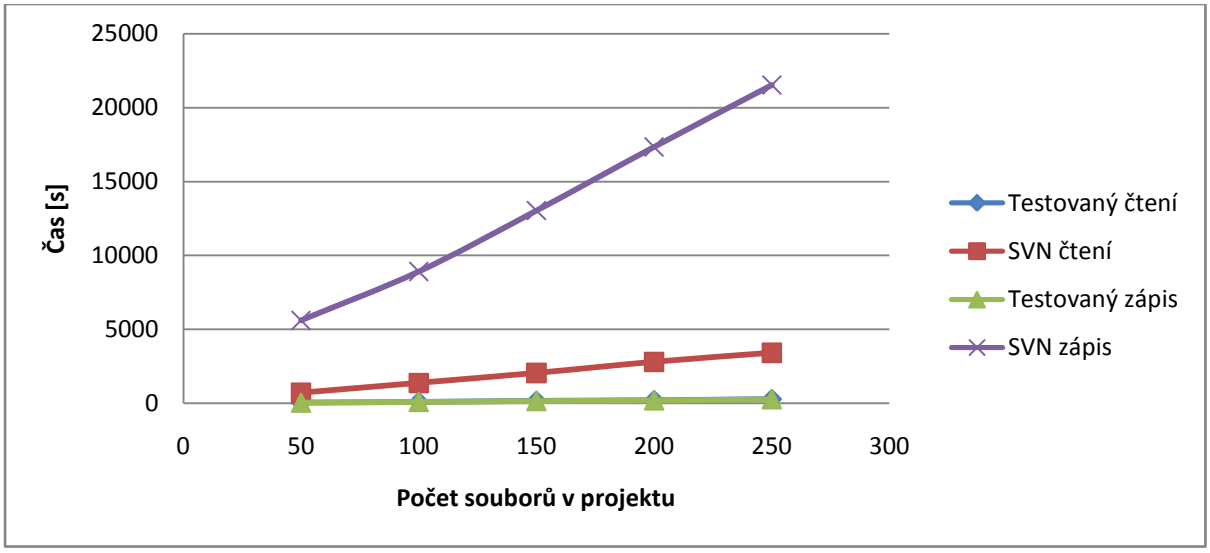
9.2.2 Výsledky

	Soubory		Čtení [s]		Zápis [s]	
	Počet souborů	Průměrná velikost [kB]	Testovaný	SVN	Testovaný	SVN
A1	500	3,6	12	25	14	117
A2	1000	3,6	25	52	22	230
A3	1500	3,6	34	79	38	352
A4	2000	3,6	54	97	55	474
A5	2500	3,6	62	125	63	595
B1	50	9010	57	714	52	5612
B2	100	9010	104	1381	97	8912
B3	150	9010	176	2054	157	13032
B4	200	9010	218	2802	201	17341
B5	250	9010	294	3419	276	21528

Tabulka 4 - Výsledky víceuživatelských testů



Obrázek 31 - Graf výkonnosti při schématu A



Obrázek 32 - Graf výkonnosti při schématu B

Při porovnání výsledků s testy provedenými v kapitole 9.1.2 je vidět, že linearita časové náročnosti obou případů je zachována. V případě revizního systému SVN se doba zpracování zvýšila přibližně o 25%, kdežto testovaný systém zůstal na stejných hodnotách. Rovnocenné časy jsou do velké míry způsobeny faktem, že i přes větší velikost souborů je méněkrát otevíráno nové spojení. Při testech se tedy ukázalo, že počet otevíraných spojení v době přenášení projektu má na celkovou propustnost velký vliv.

Při testech se bohužel nepodařilo docílit situace, kdy by testovaný systém byl do takové míry limitován rychlostí síťového spojení. V tomto případě by se dal předpokládat strmější vzestup křivky, nicméně stále by měl být lineární.

10 Závěr

Práce shrnuje veškerou teorii, ze které bylo čerpáno při návrhu a implementaci výsledného revizního systému. V teorii bylo nastíněno několik variant, které bylo možno při implementaci realizovat. Tyto varianty byly následně porovnány z hlediska výkonnosti a datové propustnosti systému. Ze všech možných variant byla vybrána ta, která by měla vykazovat nejlepší výsledky z hlediska datové propustnosti a zároveň umožňovala snadnou rozšiřitelnost velikosti datového prostoru a kapacity datové linky.

Navržený systém byl následně implementován. Implementace byla provedena na platformě .NET, která sloužila jako hostitelský middleware pro řídící jádro revizního systému. Na této platformě byla implementována i část týkající se fyzického uložení dat, tedy souborový server. Jelikož se implementace řízení metadat repositáře realizovala na databázovém serveru MSSQL Server 2005, byla tato část implementována v jazyce Transact-SQL formou uložených procedur. Toto řešení přineslo podstatné zvýšení výkonnosti při operacích s metadaty, díky nízkému počtu otevíraných připojení k databázovému úložišti a přenášení pouze konečných výsledků algoritmů.

Při implementaci byl kladen velký důraz na abstrakčnost jednotlivých částí systému a striktní vymezení jejich působnosti. S tím souvisí i vytvoření přehledného a přesně definovaného rozhraní a aplikačních protokolů, přes které části systému mezi sebou komunikují.

Výslednou aplikaci se podařilo implementovat v předpokládaném rozsahu tak, aby bylo možné demonstrovat její funkčnost na příkladech. Hlavním cílem bylo dokončit implementaci do stádia, kdy revizní systém bude schopen nahrávat a stahovat obraz projektu z repositáře a zároveň bude schopen větvit projektový strom.

Následným úkolem by měla být možnost slučování větví, kterou se nepodařilo implementovat, protože je příliš časově náročná. U slučování je velmi nutná kooperace s uživatelem, tudíž celá operace není odkázána pouze na průběh algoritmu. Řešení tohoto problému si vyžádá dobré work-flow zpracování, které bude umět velmi efektivně komunikovat s uživatelem v případě kolizí dokumentů při spojování, kdy je rozhodnutí ponecháno na uživateli.

Zajímavým rozšířením by mohla být pokročilejší implementace uložení fyzických souborů, které by dokázalo řešit rozdělování zátěže mezi jednotlivé uzly. V nynějším stavu je rozdělení řešeno po projektech, kdy je možné každému projektu definovat fyzické datové úložiště a rozhodnutí o vyvážení zátěže je ponecháno na konfiguraci systému a uživatelské znalosti očekávané zátěže na projekt.

Při testování systému se jako nejužší hrdlo ukazuje nahrávání souborů od klienta do repositáře, kdy je otvíráno velké množství spojení. Toto chování je velmi nepříjemné zvláště v případech, kdy je nahráváno velké množství malých souborů. Jelikož se dá předpokládat, že právě tato situace bude nastávat nejčastěji, měl by být tento problém optimalizován. V rámci této práce problém řešen nebyl a

bylo ponecháno pouze jednoduché neinteligentní nahrávání jednotlivých souborů. Řešení by mohla přinést komprimace přenášených souborů do jednoho archivu, který by byl na straně repositáře dekomprimován na dílčí soubory a poté by s nimi bylo nakládáno stejně jako při postupném nahrávání. Tímto by se snížila náročnost na datovou propustnost linky mezi klientem a revizním systémem, ovšem vzrostly by nároky kladené na klienta. Komprese vyžaduje využití výpočetní síly, ale čtení z datového zdroje by zůstalo stejně náročné, protože je možné výsledek komprimace přímo odesílat linkou do revizního systému.

Očekávaným přínosem od této práce mělo být podrobné seznámení se s architekturou a implementací distribuovaného systému. Neméně významná je zkušenost s programováním databázového serveru a snaha o co nejtěsnější přiblížení logiky k datům.

11 Slovníček

Repositář místo, do kterého revizní systém ukládá všechny soubory projektů, o které pečuje. Má podobu souborového systému, ovšem pohled na něj je vázán časem (souborový systém + čas = repositář).

DMA (Direct Memory Access) v počítačích se jedná o technologii řízení přímého přístupu do paměti bez obsluhy procesoru. Jinak též řízení komunikace moderováním.

Halda je obecně nestrukturované datové úložiště. Informace o místě uložení jednotlivých prvků jsou vedeny externě.

Deskriptor jedinečný identifikátor objektu, často v souborových systémech, který identifikuje používaný objekt

Literatura

[1] CONRADI Reidar; WESTFECHTEL Bernhard. Version models for software configuration management. ACM Computing Surveys (CSUR). 1998, vol. 30, is. 2, s. 232-282. Dostupný z WWW: <http://portal.acm.org/citation.cfm?id=280280>.

[2] BAUDIŠ Petr. Výlet do říše verzí [online]. První vydání. Root.cz, 2003-2004 , 10.5.2004 [cit. 2007-12-29]. Dostupný z WWW: <http://www.root.cz/serialy/vylet-do-rise-verzi/>.

[3] HOWARD John H., et al. Scale and performance in a distributed file system. ACM Transactions on Computer Systems (TOCS). 1988, vol. 6, is. 1, s. 51-81. Dostupný z WWW: <http://portal.acm.org/citation.cfm?id=35059>.

[4] Objekty - Návrhové vzory [online]. 2003-2007 , 16. 6. 2005 [cit. 2007-12-29]. Dostupný z WWW: <http://objekty.vse.cz/Objekty/Vzory>.

[5] ČEŠKA Milan, VOJNAR Tomáš, SMRČKA Aleš. Teoretická informatika: Studijní opora. FIT VUT Brno, 2007. 167 s.

[6] BEN-GAN Itzik, SARKAANDROGER WOLTER Dejan. Inside Microsoft SQL Server 2005: T-SQL Programming. [s.l.] : Microsoft Press, 2006. 532 s. ISBN 0735621977.

[7] SINK Eric. Source control [online]. 2001-2007, 26.8.2004 [cit. 2007-12-29]. Dostupný z WWW: <http://www.ericsink.com/scm/index.html>.

12 Příloha 1: instalace a nastavení

12.1 Databázové úložiště

Databázové úložiště je implementováno na platformě Microsoft SQL Serveru 2005, je tedy nutné jej mít nainsalován. K vytvoření pracovní databáze slouží skripty přiložené na CD v adresáři **SQL scripts**. Obsah je následující:

- **create_tables.sql** – slouží k vytvoření uložených procedur, které vytváří pracovní databázi revizního systému.
- **initial.sql** – provede inicializační nastavení databáze do výchozího stavu (vytvoření administrátorského účtu, vytvoření odkazu pro lokální souborový server a vytvoří první testovací projekt).
- **branches.sql** – vytvoří vestavěné procedury v existující databázi pro manipulaci s větvemi projektu.
- **file_server.sql** – vytvoří vestavěné procedury v existující databázi pro správu souborových serverů.
- **project.sql** – vytvoří vestavěné procedury v existující databázi pro správu projektů.
- **states.sql** - vytvoří vestavěné procedury v existující databázi pro manipulaci se stavy větví projektu.
- **user.sql** - vytvoří vestavěné procedury v existující databázi pro správu uživatelů.

12.1.1 Postup instalace

1. Spuštění programu **SQL Management Studio Express.**
2. Otevření scriptu **create_tables.sql** a spuštění. Tímto se ve schématu **master** vytvoří potřebné vestavěné procedury.
3. Pod schématem master spustit příkaz

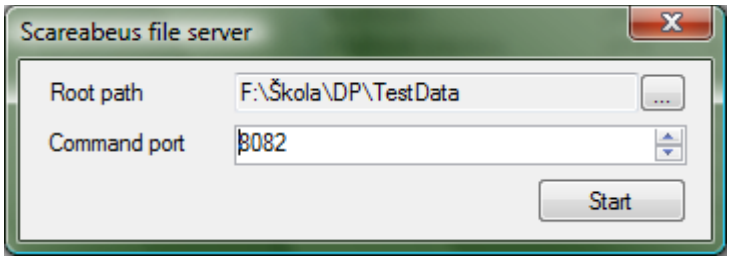
   ```
   EXEC [dbo].[scar_create_db] @name = N'<název databáze>'
   ```

   Místo elementu *<název databáze>* je vložen požadovaný název vytvářené databáze.
4. Postupně spustit všechny skripty **branches.sql**, **file_server.sql**, **project.sql**, **states.sql**, **user.sql**. Všechny skripty musí být spuštěny pod schématem nově vytvořené databáze. Všechny takto vytvářené uložené procedury musí být součástí vytvořené databáze.
5. Spuštění skriptu **initial.sql** opět pod schématem nově vytvořené databáze.

12.2 Souborový server

Spustitelná implementace souborového serveru je umístěna na přiloženém CD ve složce **FileServer**. Složka obsahuje samotný spustitelný soubor **FileServer.exe** a dále knihovnu **FileTransfer.dll**, která obsahuje implementaci logiky souborového serveru. Samotný program pak slouží pouze jako uživatelské rozhraní pro obsluhu programového rozhraní knihovny.

Souborový server stačí pouze spustit bez jakékoli předchozí konfigurace. Program musí být spuštěn pod účtem s administrátorskými oprávněními, a to z důvodu, že využívá komponentu HttpListener a ta může být provozována pouze s těmito oprávněními. Po spuštění se zobrazí dialog pro nastavení parametrů souborového serveru.



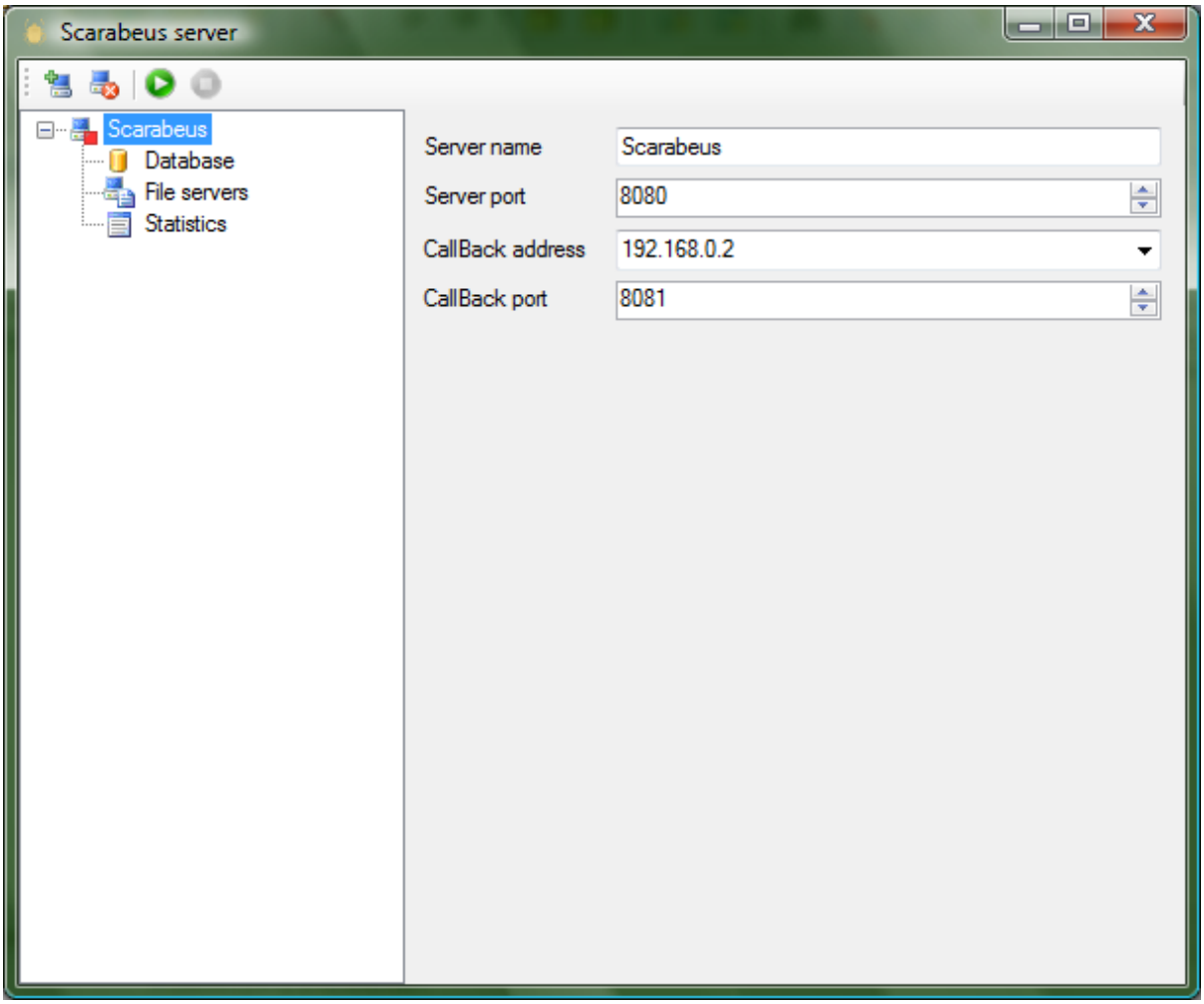
Obrázek 33 - Dialog nastavení souborového serveru

Nastavují se pouze dva parametry, a to kořenová cesta (Root path) a port pro příjem příkazů (Command port). Kořenová cesta určuje složku na lokálním datovém úložišti, na kterou se mapuje fyzická část repositářové struktury. Revizní systém může najednou obsluhovat libovolné množství takto provozovaných souborových serverů. Parametr příkazového portu určuje číslo portu, který bude využit k naslouchání příkazů zasílaných od řízení revizního systému a zároveň slouží k přijímání a odesílání souborů.

12.3 Řídící část

Řídící část je možné, stejně jako souborový server, pouze spustit. Na CD je umístěna ve složce **Control**. Složka obsahuje tyto soubory:

- **Control.exe** – obsahuje samotnou implementaci řídící části a zároveň tvoří uživatelské rozhraní pro administraci revizního systému.
- **FileTransfer.dll** – tato knihovna je shodná se stejnojmennou knihovnou ze složky souborového serveru. V této knihovně je obsažena jak serverová tak klientská část komunikace potřebné pro fyzické úložiště dat.
- **Servers.xml** – obsahuje nastavení definovaných serverů. Tento soubor není nutné ručně editovat, protože je automaticky generován z nastavení provedeném v uživatelského rozhraní.



Obrázek 34 - Uživatelské rozhraní řídící části

V řídící části je možné nastavit libovolný počet instancí revizního systému, které mohou na jedno stroji běžet paralelně. U každé instance revizního systému je nutné zadat následující parametry:

- **Název serveru** – slouží pouze jako identifikace a na funkčnost nemá vliv.
- **Serverový port** – je číslo, na který jsou přijímány všechny požadavky zasílané od klientů.
- **Návratová adresa** – slouží pro zasílání zpráv od souborových serverů do řízení (po nahrání souboru na souborový server). Ze seznamu všech dostupných adres je možné vybrat pouze jednu. Seznam obsahuje všechny adresy, které hostitelský počítač na všech síťových rozhraních vlastní. Vybraná adresa musí být taková, která je součástí stejné sítě jako je umístění souborového serveru.
- **Návratový port** – doplňuje vybranou adresu o specifikaci portu.

Pro každou instanci musí být nakonfigurováno připojení k databázovému serveru, kde je nutné zadat adresu počítače, na kterém databázový server běží, název databáze, uživatelské jméno a heslo.

13 Příloha 2: přiložené CD

Součástí práce je přiložený CD disk, který obsahuje tyto adresáře:

- **Control** – přeložený a spustitelný program řídící části revizního systému
- **FileServer** – přeložený a spustitelný souborový server
- **Sources** – zdrojové kódy pro řídící část revizního systému a pro souborový server. Zdrojové kódy jsou přiloženy formou projektu pro Microsoft Visual Studio 2008.
- **SQL scripts** – SQL skripty pro vytvoření databáze a vložení uložených procedur.
- **Text** – obsahuje zdrojový text této práce ve formátech PDF, DOC, DOCX a XPS.