VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE WEBDAV ROZHRANÍ DOKUMENTOVÉHO SKLADU IS FIT

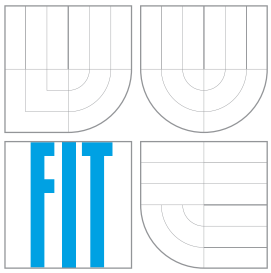
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ JELÍNEK

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

IMPLEMENTACE WEBDAV ROZHRANÍ DOKUMENTOVÉHO SKLADU IS FIT

WEBDAV INTERFACE FOR IS FIT DOCUMENT REPOSITORY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ JELÍNEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR LAMPA

BRNO 2008

Abstrakt

Cílem této diplomové práce je implementace WebDAV rozhraní dokumentového skladu IS FIT v jazyce PHP. Navazuje na semestrální projekt, který se zabýval studiem protokolu WebDAV a volně dostupného WebDAV serveru. Práce pojednává o protokolu WebDAV a jeho významu, zabývá se také souvisejícími technologiemi: HTTP, XML, PHP a MySQL. Dále popisuje studovaný WebDAV server, dokumentový sklad IS FIT, návrh a implementaci jeho WebDAV rozhraní. V poslední části pak popisuje spolupráci vytvořeného rozhraní s WebDAV klienty, shrnuje a hodnotí dosažené výsledky.

Klíčová slova

WebDAV, HTTP, XML, PHP, IS FIT, dokumentový sklad, MySQL

Abstract

This Master's thesis aim is implementation of WebDAV interface for IS FIT document repository in PHP language. It concurs to term project, that has dealt with protocol WebDAV and open source WebDAV server. Thesis discuss protocol WebDAV and it's meaning and related technologies: HTTP, XML, PHP and MySQL. Then it describes studied WebDAV server, IS FIT document repository, design and implementation of it's WebDAV interface. Final part describes cooperation with WebDAV clients and gives a summary and evaluation of achieved results.

Keywords

WebDAV, HTTP, XML, PHP, IS FIT, document repository, MySQL

Citace

Tomáš Jelínek: Implementace WebDAV rozhraní dokumentového skladu IS FIT, diplomová práce, Brno, FIT VUT v Brně, 2008

Implementace WebDAV rozhraní dokumentového skladu IS FIT

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Petra Lampy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Jelínek
19. května 2008

Poděkování

Děkuji vedoucímu Ing. Petru Lampovi za poskytnuté rady a odbornou pomoc.

Obsah

Kapitola 1

Úvod

Tato diplomová práce se zabývá implementací WebDAV (Web-based Distributed Authoring and Versioning) rozhraní dokumentového skladu IS FIT. Cílem je umožnit přístup k dokumentovému skladu protokolem WebDAV a zrychlit a zefektivnit tak práci s ním.

Text je členěn do kapitol seřazených způsobem odpovídajícím postupu při studiu a implementaci. Nejprve jsou stručně popsány výhody protokolu WebDAV vzhledem k jiným možnostem vzdálené správy souborů. Následuje kapitola popisující technologie související s WebDAVem. Představuje protokol HTTP (HyperText Transfer Protocol), neboť WebDAV je jeho rozšířením. Dále jazyk XML (eXtensible Markup Language) jako prostředek přenosu strukturovaných dat. Druhá část kapitoly se zabývá jazyky PHP (PHP Hypertext Preprocesor) a SQL (Structured Query Language) a databázovým systémem MySQL, které tvoří implementační prostředí.

Čtvrtá kapitola se věnuje protokolu WebDAV, popisuje jeho principy, vlastnosti, možnosti a funkce. Další kapitola pak studuje jednu z existujících volně dostupných implementací WebDAV serveru v jazyce PHP. Následuje představení dokumentového skladu, jeho vývoje a popis principů činnosti jednotlivých verzí.

Počínaje sedmou kapitolou je již pozornost věnována vytvářenému WebDAV rozhraní. Nejprve jsou uvedeny základní principy fungování rozhraní jako výsledky fáze analýzy a návrhu rozhraní. Následuje detailní popis implementace. Dále jsou uvedeny výsledky testování hotového produktu některými rozšířenými WebDAV klienty. Následuje popis možného rozšíření a doplnění dalších funkcí rozhraní. Závěrečná kapitola shrnuje a hodnotí dosažené výsledky.

Práce čerpá z řady materiálů. Jsou to především RFC dokumenty a specifikace protokolu HTTP [3], jazyka XML [2] a protokolu WebDAV [4]. Byla využita také encyklopedie Wikipedia, konkrétně články [7], [8], [9], [10] a [11]. Ke studiu dokumentového skladu posloužily [5] a [6]. Při implementaci pak [1].

Tato diplomová práce navazuje na dříve vypracovaný semestrální projekt. V jeho rámci jsem prostudoval protokoly HTTP a WebDAV, jazyk XML a jednu z existujících implementací WebDAV serveru. Získané znalosti jsem uplatnil v této diplomové práci.

Kapitola 2

WebDAV jako prostředek vzdálené správy souborů

WebDAV je rozšíření protokolu HTTP/1.1 sloužící pro vzdálenou správu souborů na webových serverech. Umožňuje přenášet soubory směrem z klienta na server a naopak, vytvářet a rušit adresáře a soubory, kopírovat a přesouvat je v rámci serveru i mezi servery, pracovat s metadaty k nim přidruženými, podporuje zamykání.

V současné době existuje více protokolů a možností, jak řešit tyto uvedené činnosti. Lze použít například protokol FTP (File Transfer Protocol) a jeho varianty nebo SFTP (SSH File Transfer Protocol). Jinou možností je sdílení souborů, jak je známo z prostředí MS Windows, prostřednictvím protokolu SMB (Server Message Block) nebo jeho svobodné implementace Samba. Nevýhodou tohoto přístupu může být nutnost použití jednoho protokolu pro přenos souborů na server a druhého pro přenos souborů ke klientům, pokud se jedná o soubory na webovém serveru, ke kterým klienti přistupují HTTP protokolem.

Další možností je použití protokolu HTTP a serverových skriptů. V tomto případě uživatel na straně klienta provádí veškerou práci skrze webový prohlížeč. Skripty na serveru tvoří mezivrstvu mezi klientem a spravovanými daty na serveru: generují uživatelské rozhraní pro klienta a podle jeho pokynů pracují s daty na serveru. Takto dosáhneme použití pouze jednoho protokolu, nevýhodou je ale svým způsobem nepřímé uživatelské rozhraní, které činí práci poněkud nepohodlnou a zdlouhavou.

WebDAV řeší nedostatky obou výše zmíněných přístupů. Protože rozšíření protokolu HTTP je provedeno přidáním nových typů zpráv a nikoliv zásahy do základních principů protokolu, můžeme při určitém zjednodušení říci, že nebylo třeba použití dalšího protokolu. WebDAV servery lze připojit jako síťové disky a se soubory na nich pak pracovat s komfortem srovnatelným s prací se soubory na lokálních discích.

Kapitola 3

Použité technologie a prostředí

Před studiem samotného protokolu WebDAV je vhodné se seznámit s technologiemi, které tento protokol používá. Jedná se zejména o protokol HTTP, jehož je WebDAV rozšířením. Dále pak jazyk XML, který při komunikaci slouží pro zasílání některých parametrů upřesňujících význam zpráv. Dokumentový sklad je implementován v jazyce PHP a využívá databázi MySQL. I jim je tedy věnováno po jedné podkapitole.

3.1 Protokol HTTP

HTTP (HyperText Transfer Protocol) je komunikační protokol náležející do aplikační vrstvy. Jeho původním účelem bylo umožnit přenos hypertextových dokumentů. Stal se jedním z nejpoužívanějších protokolů poslední doby. Díky rozšíření MIME jím lze přenášet libovolné soubory. Protokol je vyvíjen organizacemi W3C (World Wide Web Consortium) a IETF (Internet Engineering Task Force). Aktuální verze HTTP/1.1 je popsána ve specifikaci [3], která posloužila jako hlavní zdroj informací pro tuto kapitolu. Druhým zdrojem byl článek na Wikipedii [7]. Specifikace je velmi rozsáhlá, proto jsou zde uvedeny pouze základní principy a pozornost je věnována částem vztahujícím se k protokolu WebDAV.

Jedná se o textový protokol typu požadavek – odpověď (request – response). Klient naváže spojení se serverem a odešle svůj požadavek. Server požadavek zpracuje a zašle klientovi odpověď. Poté je spojení uzavřeno. Každý požadavek je nezávislý na předchozí komunikaci, HTTP je tedy bezestavový protokol. Tato vlastnost zůstává zachována i ve verzi 1.1, která umožňuje jedním spojením přenést několik požadavků a odpovědí. Protože však ve webových aplikacích je třeba stav uchovávat, byly vytvořeny mechanismy, které to umožňují (sessions, cookies). Mezi klientem a serverem se mohou nacházet prostředníci: proxy servery, brány a tunely. Jako přenosový protokol je obvykle využíván TCP/IP, kde protokolu HTTP je přidělen port 80. Šifrovaná varianta HTTPS využívá port 443. Mohou však být použity i jiné porty. Ani použití protokolu TCP/IP není podmínkou, jediným požadavkem na přenosový protokol je spolehlivý přenos.

V protokolu HTTP se rozlišují dva typy zpráv, jsou to *požadavky* (zasílané klientem na server) a *odpovědi* (zasílané klientovi v reakci na požadavek). Oba typy zpráv mají stejnou strukturu:

- úvodní řádek – určuje a popisuje požadavek nebo odpověď
- žádná nebo několik hlaviček, každá na samostatném řádku, na pořadí hlaviček nezáleží
- prázdný řádek

- tělo zprávy – v závislosti na typu požadavku nebo odpovědi nemusí být ve zprávě obsaženo

Hlavičky jsou doplňující informace, které upřesňují požadavek nebo odpověď. Jsou tvořeny *názvem hlavičky* následovaným *dvojtečkou* a *daty hlavičky*. Dělí se na:

- obecné hlavičky (datum, řízení cache, uzavření spojení, ...)
- hlavičky v požadavku (určení serveru, schopnosti klienta, autentizace, ...)
- hlavičky v odpovědi (přesměrování, požadavek na autentizaci, ...)
- hlavičky popisující entitu předávanou v těle zprávy (kódování, délka, jazyk, ...)

Požadavek obsahuje v úvodním řádku *metodu* (příkaz) definující požadovanou akci, *URI* (Uniform Resource Identifier) určující zdroj, nad kterým bude akce provedena, a *označení protokolu* včetně verze. HTTP protokol definuje osm metod, přičemž ponechává možnost přidávat metody další:

`OPTIONS` Umožňuje klientovi zjistit schopnosti serveru.

`GET` Požaduje zaslání zdroje určeného předaným URI. Je to pravděpodobně nejpoužívanější metoda.

`HEAD` Totožné s `GET`, ale server zasílá pouze hlavičky (metadata) a nikoliv tělo zprávy (data).

`POST` Zaslání dat z klienta na server ke zpracování. Používá se například pro odesílání formulářů. Data jsou přenášena v těle zprávy. Přesná funkce této metody je určena serverem a závislá na URI. URI zde označuje zdroj, který bude data zpracovávat.

`PUT` Uložení dat na server na zadané URI. Na rozdíl od metody `POST` zde URI označuje místo, kam budou odeslaná data uložena. Pokud takové URI neexistuje, bude vytvořeno. Pokud existuje, pak jeho obsah bude nahrazen zaslanými daty.

`DELETE` Smaže zdroj určený v URI.

`TRACE` Vytváří zpětnou smyčku. V těle odpovědi je předán původní požadavek. Umožňuje klientovi zjistit, co server přijal.

`CONNECT` Používá se pro vytvoření tunelu z proxy serveru.

Úvodní řádek v odpovědi obsahuje *označení protokolu* včetně verze, trojciferný *stavový kód* a slovní *popis výsledku* dotazu. Je definováno pět skupin výsledků dotazu a odpovídajících stavových kódů:

1xx informační – požadavek přijat, zpracovává se

2xx úspěch – požadovaná akce byla úspěšně provedena

3xx přesměrování – aby byl požadavek dokončen, je třeba vykonat další akci

4xx chyba klienta – požadavek je syntakticky nesprávný nebo ho nelze splnit

5xx chyba serveru – server nedokázal splnit požadavek

3.2 Jazyk XML

Zdrojem informací pro tuto kapitolu byl článek na Wikipedii [11] a specifikace jazyka [2].

XML (eXtensible Markup Language) je obecný značkovací jazyk. Jeho hlavním účelem je umožnit sdílení strukturovaných dat mezi různými systémy. Slovo „rozšiřitelný“ (extensible) v jeho názvu znamená, že dovoluje uživatelům definovat si vlastní tagy. XML vychází ze staršího jazyka SGML (Standard Generalized Markup Language). Byl vyvinut a standardizován konsorciem W3C.

Dokument v jazyce XML se skládá z *elementů*, jejichž začátek je označen *počátečním tagem* a konec *ukončovacím tagem*. Mezi těmito tagy se nachází *obsah* elementu. Existují také elementy bez obsahu, ty lze zapisovat jedním *tagem označujícím prázdný element*. Každý element je nějakého typu, který je dán jeho názvem uvedeným v tagu. Názvy mohou být zařazeny do různých jmenných prostorů. Elementy mohou být parametrizovány *atributy* tvořenými dvojicí *název atributu* a *hodnota atributu*. Atributy se zapisují do počátečního tagu nebo tagu prázdného elementu. Jednotlivé elementy se mohou do sebe zanořovat. Na začátku XML dokumentu před všemi elementy může být uvedena *XML deklarace*, která označuje verzi jazyka a použité kódování. Za ní může následovat definice XML schématu.

Správnost XML dokumentu se posuzuje ze dvou hledisek: zda je dokument *správně strukturován (well-formed)* a zda je *validní*. Dokument je well-formed, pokud odpovídá XML syntaxi. Nejpodstatnější pravidla, která musí XML dokument splňovat, aby byl well-formed:

- V dokumentu se vyskytuje právě jeden kořenový element, který obsahuje všechny ostatní elementy.
- Neprázdné elementy jsou ohraničeny počátečním a ukončovacím tagem. V názvech elementů se rozlišují velká a malá písmena.
- Prázdné elementy mohou být označeny tagem pro prázdný element, což je ekvivalentní použití počátečního a ukončovacího tagu bez obsahu mezi nimi.
- Elementy se mohou vnořovat, ale nesmí se křížit nebo překrývat. To znamená, že každý nekořenový element musí být celý obsažen v nadřazeném elementu.
- Hodnoty atributů musí být uzavřeny v uvozovkách, buďto jednoduchých nebo dvojitých. Kombinace obou druhů uvozovek je nepřípustná, lze však použít jedny uvnitř druhých.

Dokumenty, které nejsou well-formed, jsou celé prohlášeny za chybné, nejsou dále zpracovávány a výsledky dosavadního zpracování jsou zahozeny.

Validní XML dokumenty jsou takové, které jsou well-formed a navíc vyhovují dalším pravidlům. Ta jsou specifikována v dokumentech *XSD (XML Schema Definition)* nebo *DTD (Document Type Definition)*. Jedná se v podstatě o stanovení jmen, pořadí a zanořování elementů a jmen a povolených hodnot atributů. Tyto dokumenty tedy obsahují popis dat neboli *metadata*. DTD je starším způsobem definice XML schématu. Je zděděný z jazyka SGML, nelze v něm tudíž zachytit některé vlastnosti XML, například jmenné prostory. Následníkem DTD je XSD, což je dokument v jazyce XML. Použití jednoho jazyka pro popis dat i metadat je výhodné, neboť pro zpracování obojího pak lze do určité míry použít stejné nástroje. Protože je XSD tvořené na míru XML, umožňuje zachytit všechny jeho vlastnosti.

Jazyk XML definuje pouze syntaxi. Uživatel si tak může libovolně zvolit strukturu dokumentu a sémantiku jednotlivých jeho části. To je jednou ze základních vlastností tohoto jazyka. Díky tomu lze vytvořit nástroje pracující se strukturou XML nezávisle na sémantice, tedy se všemi XML dokumenty stejným způsobem. To přináší programátorům a dalším uživatelům určitý komfort, neboť se při práci s XML dokumenty mohou soustředit pouze na sémantiku a starost o syntaxi přenechat těmto nástrojům.

Některá rozšíření XML:

XPath (XML Path Language) je jazyk umožňující adresovat libovolnou část XML dokumentu, například data a jména elementů a atributů, komentáře.

XInclude umožňuje do těla XML dokumentu vložit jiný XML dokument.

XQuery je dotazovací jazyk sémanticky podobný SQL.

XML Signature definuje syntaxi pro elektronický podpis.

XML Encryption definuje šifrování obsahu XML elementů.

XPointer je systém pro adresování komponent internetových médií založených na XML.

XML dokumenty lze zpracovávat několika různými přístupy:

SAX API (Simple API for XML) je rozhraní, které umožňuje sériový přístup k datům v dokumentu. SAX parser postupně čte XML dokument a volá uživatelem definované funkce, čímž oznamuje nalezení elementů, atributů a dalších částí. Toto řešení je poměrně jednoduché a efektivní. Nevýhodou je nemožnost náhodného přístupu k částem dokumentu a nutnost uchovávat si informace o aktuální pozici v jeho struktuře. Parser pracující na tomto principu je běžně k dispozici v PHP.

DOM API (Document Object Model) vytváří objektovou reprezentaci XML dokumentu ve formě stromu. Poskytuje prostředky pro navigaci po jeho uzlech, pro přidávání, mazání a modifikaci uzlů a podstromů. Je doplňkem SAX principu, umožňuje libovolný přístup k částem dokumentu za cenu vyšší paměťové náročnosti a pomalejšího zpracování. Pro PHP je k dispozici i tento typ parseru.

Transformace na základě popisu. Používají se jazyky XPath a XQuery zmíněné výše, dále pak XSLT a XSL-FO. XSLT (Extensible Stylesheet Language Transformations) je deklarativní jazyk s XML syntaxí sloužící pro popis transformace struktury XML dokumentu. XSL-FO (Extensible Stylesheet Language Formatting Objects) je deklarativní jazyk taktéž s XML syntaxí, slouží pro převod XML dokumentu do formátů pro zobrazení jako například PDF. Taktéž tento přístup ke zpracování XML, konkrétně XSLT, lze v PHP použít.

3.3 PHP

PHP (PHP Hypertext Preprocesor, původně Personal Home Page) je multiplatformní interpretovaný skriptovací jazyk. Byl původně vytvořen jako prostředí pro programování dynamických webových stránek, později pak celých webových aplikací. Syntaxí se podobá jazykům C, Perl a Java. Umožňuje práci s množstvím databázových systémů, například

MySQL, PostgreSQL, Oracle, Microsoft SQL Server, Firebird. Často se nasazuje v kombinaci s HTTP serverem Apache a databází MySQL na serverech s operačním systémem Linux. Toto řešení se označuje zkratkou LAMP.

Historie vývoje jazyka je čerpána z [9]. Autor jazyka Rasmus Lerdorf začal s jeho vývojem v roce 1994. Impulzem pro zahájení práce byla myšlenka vytvoření nástroje, kterým by sledoval návštěvnost a provoz na svých webových stránkách. Skripty v jazyce Perl, které za tím účelem napsal, tuto funkci splňovaly, avšak příliš zatěžovaly server. Přepsal je tedy do jazyka C a nabídl ostatním uživatelům pod názvem Personal Home Page Tools. Za nějaký čas se objevili uživatelé, kteří přispěli k vývoji několika svými rozšířeními. Prostředí PHP bylo doplněno o možnost práce s databázemi a zpracování formulářů na webových stránkách. Tato verze označená PHP/FI 2 (Form Interpreter) se již začala šířit do světa.

V roce 1997 Zeev Suraski a Andi Gutmans přepsali parser a položili základy přelomové verze 3. Ta byla vydána v roce 1998 a přinesla mnohá vylepšení. Jednalo se především o multiplatformnost, možnost práce s databázovými systémy a s cookies. Původní význam zkratky již neodpovídal rozsahu systému a byl proto nahrazen názvem PHP Hypertext Preprocesor. Verze 4 byla postavena na novém jádru Zend 1.0, což se projevilo podstatným zrychlením. Přišla také s celou řadou nových funkcí, jednou z nejvýznamnějších byla možnost práce se sessions. Ukončení vývoje této verze bylo oznámeno na konec roku 2007, definitivní ukončení podpory se předpokládá v polovině roku 2008.

V současné době se používá verze 5, která byla vydána v červenci 2004. Má opět nové jádro Zend Engine II. Mezi nejdůležitější změny patří vylepšení podpory objektově orientovaného programování, zavedení výjimek, iterátorů, podpory SOAP (protokol pro výměnu zpráv založených na XML počítačovou sítí) a usnadnění práce s databázovými systémy. Souběžně je vyvíjena verze 6, která odstraní některé zastaralé nepoužívané vlastnosti a naopak doplní potřebné vlastnosti nové, například nativní podporu kódování Unicode.

Pro PHP bylo vytvořeno množství doplňujících knihoven a rozšíření. Kromě práce s různými databázemi přidávají funkce pro práci s komprimovanými soubory, obrázky, PDF soubory, XML dokumenty. Jiné knihovny umožňují komunikaci s FTP, IRC, e-mailovými a dalšími aplikačními servery.

Typicky se v PHP vytvářejí dynamické webové stránky a webové aplikace. PHP kód se může začleňovat přímo do HTML dokumentů, což se používá v případě krátkých jednoduchých skriptů. U větších projektů se obvykle používá druhý přístup, kdy v souborech převládá PHP kód. Skripty realizující funkčnost webové aplikace jsou odděleny od skriptů řešících HTML výstup. Spolu s využitím technik objektového programování je tak výsledná aplikace lépe udržovatelná. Pokud klient požádá o soubor obsahující PHP skripty, jsou tyto skripty zpracovány serverem a klientovi je odeslán jejich výstup. Ten obyčejně nabývá podoby HTML dokumentů, výjimkou však nejsou výstupy ve formě obrázků, XML dokumentů a dalších formátů.

V dnešní době je prostředí PHP značně univerzální a umožňuje vytvářet i jiné typy aplikací. Pokud zůstaneme v oblasti internetu, lze v PHP programovat servery a klienty implementující protokoly aplikační úrovně. Kromě protokolu WebDAV, kterým se zabývá tato práce, se může jednat například o protokol IRC, novější Jabber nebo široce používaný FTP. Protože PHP poskytuje i rozhraní příkazové řádky, lze ho použít také k tvorbě desktopových aplikací, démonů, skriptů pro parsování logů a tak dále. S využitím GUI knihoven GTK+ a QT je možné v PHP vytvořit multiplatformní GUI aplikace.

3.4 MySQL

Jazyk SQL (Structured Query Language) je standardizovaný jazyk pro práci s relačními databázovými systémy. Syntakticky se blíží přirozenému jazyku, konkrétně angličtině. Umožňuje pracovat s vlastními uloženými daty, databázovým schématem a používá se i pro řízení přístupu k databázovým objektům.

Historie jazyka se podle [10] začíná psát v 70. letech. Tehdy byl firmou IBM vyvinut databázový systém s názvem System R, který byl založen na práci E. F. Codda popisující relační model databázového systému. Pro práci s tímto systémem byl vytvořen jazyk SEQUEL (Structured English Query Language), zkratka byla později změněna na SQL. V roce 1979 představila společnost Relational Software, Inc. (nyní Oracle Corporation) první komerčně dostupnou implementaci SQL – Oracle V2.

Brzy se začaly objevovat další relační databázové systémy používající jazyk SQL. Ten se tak stal de facto standardem. V roce 1986 se stal ANSI standardem, o rok později ISO standardem. Tato varianta jazyka je proto označována jako SQL-86 nebo SQL-87. Časem se objevily různé nedostatky, byla proto vytvořena nová verze SQL-92 označovaná také jako SQL2, která je odstraňovala. Obsahovala nové prvky související především s integritou dat. SQL3 vydaný v roce 1999 přinesl další zlepšení jako rekurzivní dotazy, triggery a podporu regulárních výrazů. Zavedl také příkazy ovlivňující tok řízení (control-of-flow), procedurální příkazy a některé prvky objektového přístupu. Následující verze SQL:2003 a SQL:2006 zavádějí a rozšiřují práci s XML.

Přestože je jazyk SQL standardizovaný, většina databázových systémů k němu přidává vlastní prvky a naopak některé části standardu nepodporuje. Vznikla tak řada SQL dialektů, což omezuje přenositelnost.

Příkazy jazyka SQL se dělí do několika kategorií:

- Příkazy pro manipulaci s daty (DML – Data Manipulation Language) umožňují čtení, zápis, úpravu a mazání dat.
- Příkazy pro definici dat (DDL – Data Definition Language) se používají pro tvorbu, úpravu a mazaní tabulek, indexů, pohledů a dalších objektů. Definuje se jimi struktura databáze.
- Příkazy pro řízení dat (DCL – Data Control Language) zahrnují příkazy řízení přístupu a nastavování přístupových práv a oprávnění uživatelů.
- Příkazy pro řízení transakcí (TCC – Transaction Control Commands) umožňují zahájit, potvrdit a stornovat databázové transakce. Někdy se řadí do DCL.
- Ostatní příkazy jako například ty ovlivňující tok řízení.

MySQL je podle [8] vícevláknový víceuživatelský multiplatformní relační databázový systém. Implementuje jazyk SQL, přesněji vlastní dialog tohoto jazyka. Je vlastněn a sponzorován švédskou společností MySQL AB, která vlastní copyright na většinu jeho kódu. MySQL AB založili v roce 1995 David Axmark, Allan Larsson a Michael „Monty“ Widenius, kteří jsou hlavními autory MySQL.

Díky snadné dostupnosti (je k dispozici pod komerční placenou licencí i pod bezplatnou licencí GPL), multiplatformnosti a dobrému výkonu se MySQL těší značné oblibě. Je nasazován zejména jako databázový systém využívaný webovými aplikacemi, často těmi napsanými v jazyce PHP. Příkazy v jazyce SQL se z PHP zasílají MySQL serveru voláním funkcí příslušné PHP knihovny.

Kapitola 4

WebDAV

Protokol WebDAV (Web-based Distributed Authoring and Versioning) je rozšíření protokolu HTTP/1.1 sloužící pro vzdálenou správu souborů na webových serverech. Je popsán ve specifikaci [4], která posloužila jako zdroj informací pro tuto kapitolu. Rozšíření je realizováno přidáním *metod*, *hlaviček* a *stavových kódů*, což je plně v souladu se specifikací protokolu HTTP, která taková rozšíření přímo umožňuje. Při práci se zdroji je podporováno *zamykání*, práce s *vlastnostmi (metadaty)*, *jmennými prostory* a *kolekcemi* zdrojů.

V protokolu HTTP je celý požadavek specifikován v hlavičce zprávy, žádné tělo ve zprávě není. Výjimku tvoří pouze požadavky metod `POST` a `PUT`, kde tělo obsahuje data odesílaného formuláře nebo souboru. Toto tělo však z hlediska protokolu význam požadavku neovlivňuje. V protokolu WebDAV je ale často třeba předávat požadavky složitější a přenos jejich parametrů v hlavičce by byl komplikovaný. Řešením bylo přidání těla v jazyce XML, které obsahuje některé parametry požadavků a odpovědí, zpravidla ty předem neznámé délky nebo ty potenciálně zobrazitelné člověku. Jazyk XML byl zvolen proto, že je rozšiřitelný a používá kódování ISO 10646 (Unicode), což umožňuje mezinárodní použití. WebDAV je jedním z prvních protokolů používajících XML.

Následující podkapitoly popisují principy protokolu WebDAV (vlastnosti, kolekce, zamykání) a rozšíření protokolu HTTP (metody, hlavičky, stavové kódy). Lze z nich získat základní avšak ucelenou představu o možnostech a funkci protokolu. Jednotlivé XML elementy těl zpráv a jejich gramatika popsána není, neboť elementů je oproti vlastnostem poměrně mnoho a slouží většinou pro strukturovaný zápis vlastností. Jejich popis by tedy přinesl minimum nových podstatných informací, zájemci ho mohou nalézt v [4].

4.1 Vlastnosti a jejich datový model

V terminologii WebDAVu jsou metadata popisující zdroje označována termínem *vlastnosti (properties)*. Skládají se ze *jména* a *hodnoty*. Vlastnosti se dělí do dvou skupin:

Živé vlastnosti (live properties) jsou takové, jejichž syntaxe a sémantika je vynucována serverem. Jsou to vlastnosti pouze pro čtení, klient jejich hodnoty nemůže měnit, udržuje je server. Mohou být udržovány také klientem, ale server kontroluje jejich syntaxi a sémantiku. Příkladem je vlastnost `getcontentlength` udávající délku (datovou velikost) zdroje na serveru.

Mrtvé vlastnosti (dead properties) server pouze uchovává. Jejich syntaxe, sémantika a konzistence je věcí klienta.

Vlastnosti existují v omezené míře v HTTP hlavičkách. Nicméně vlastností je třeba přenášet velké množství a jejich umístění pouze do hlaviček by bylo neefektivní. Přenášejí se tedy v těle zpráv v XML struktuře. Využívá se výhod jazyka XML: je sebepopisující, flexibilní, podporuje různá kódování (znakové sady), zajišťuje snadnou rozšiřitelnost a zpětnou kompatibilitu (starší implementace ignorují jim neznámé nově přidané elementy).

Jména vlastností jsou unikátní identifikátory přiřazující vlastnostem definovanou syntaxi a sémantiku. Klienti se tak mohou spoléhat na to, že daná vlastnost se chová konzistentně na všech serverech a v souvislosti se všemi zdroji. Jmenný prostor vlastností je plochý.

Specifikace WebDAV popisuje následující vlastnosti:

`creationdate` datum a čas vytvoření zdroje

`displayname` popis zdroje vhodný pro zobrazení člověku

`getcontentlanguage` jazyk zdroje – odpovídá hlavičce `Content-Language`

`getcontentlength` délka (datová velikost) zdroje – odpovídá hlavičce `Content-Length`

`getcontenttype` typ obsahu zdroje – odpovídá hlavičce `Content-Type`

`getetag` odpovídá hlavičce `ETag`

`getlastmodified` datum a čas poslední změny zdroje (nebo jeho vlastností) – odpovídá hlavičce `Last-Modified`

`lockdiscovery` popis aktuálně existujících zámků pro daný zdroj

`resourcetype` povaha zdroje

`source` obsahuje cestu k nezpracovanému zdrojovému kódu zdroje (užitečné například při práci se serverovými skripty)

`supportedlock` popis platných kombinací rozsahu (sdílený, výlučný) a typu (čtení, zápis) zámku pro daný zdroj

4.2 Kolekce

Kolekce (collection) je nový typ zdroje modelující objekty typu kolekce (jako jsou například adresáře v souborovém systému). Systém pojmenování kolekcí je konzistentní s URL, jednotlivé položky hierarchie jsou odděleny znakem / . Kolekce obsahuje seznam členských URI a je popsána množinou vlastností. Může obsahovat také tělo, které lze získat metodou `GET` aplikovanou na kolekci.

Pro vytváření kolekcí byla zavedena metoda `MKCOL`. Použití metod `PUT` a `POST` se jevilo jako nevhodné. Význam metody `PUT` je v případě kolekcí nejednoznačný a nejasný. Při použití metody `POST` by vznikly problémy při řízení přístupu a odlišení vytváření kolekce od ostatních použití metody.

4.3 Zamykání

Řízení souběžného přístupu je vyřešeno pomocí zamykání. Cílem je zabránit stavu, kdy dva klienti upravují současně tentýž zdroj, což vede k nekonzistenci. Zámky mají dva parametry: *rozsah* zámku (výlučný, sdílený) a *typ* přístupu (čtení, zápis).

Výlučný (exclusive) zámek povoluje daný typ přístupu pouze jednomu klientovi, a to tomu, který uzamčení provedl.

Sdílený (shared) zámek může získat více klientů současně. Neomezuje přístup ke zdroji, ale informuje, že se zdrojem pracuje někdo jiný.

Zamykání nemusí být implementováno ani na serveru ani na klientovi. Podle podpory zamykání se implementace rozdělují do dvou *tříd shody (Compliance Classes)*: třída 1 zamykání nepodporuje, kdežto třída 2 ano. Je možné, že později vzniknou i další třídy, které se budou odlišovat jinými vlastnostmi, v současné době existují pouze tyto dvě. Pokud konkrétní implementace zamykání podporuje, musí umět zpracovávat všechny prvky protokolu WebDAV, které se zamykáním souvisí. Může ale podporovat libovolnou kombinaci výlučných a sdílených zámků pro libovolné typy přístupu. Tato flexibilita je vynucena různými typy používaných datových úložišť na serveru (souborový systém, databáze a další), která mají různou podporu pro zamykání.

Pokud je zamykání implementováno, je třeba řešit některé typické související problémy:

- klient musí být schopen zjistit, jaké zámky server podporuje
- klient musí být schopen zjistit, jaké zámky existují a kdo je vlastní
- může se stát, že klient zámek nikdy neodemkne (například následkem chyby klienta, přerušení spojení)

WebDAV obsahuje popis mechanismů řešících všechny tyto problémy. Aby se zabránilo trvalému zamčení nějakého zdroje, klient může při požadavku o zámek stanovit časový interval, po jehož uplynutí je zámek automaticky uvolněn. Tento interval ale nestanovuje dobu, po kterou bude zámek určitě existovat. Zámek může být uvolněn i kdykoliv dříve například zásahem správce.

Jednotlivé zámky jsou identifikovány *tokeny*, které jsou reprezentovány unikátními URI. Znalost identifikátoru zámku nezajišťuje klientovi přístup k danému zdroji, protože tyto identifikátory jsou zjistitelné všemi klienty (všichni musí být schopni zjistit, jaké zámky existují). Zamykání je založeno na autentizaci na straně serveru a nikoliv na utajování identifikátorů zámků.

Pro práci se zámky slouží metody `LOCK` a `UNLOCK`, hlavičky `Lock-Token` a `Timeout`, stavový kód `423 Locked`, vlastnosti `lockdiscovery` a `supportedlock` a jim odpovídající XML elementy. Jejich popis je uveden v příslušných podkapitolách.

4.4 Metody

Protokol WebDAV přidává nové metody a upravuje některé stávající převzaté z protokolu HTTP. Všechny tyto metody jsou povinné s výjimkou `LOCK` a `UNLOCK`, implementace bez nich odpovídá třídě shody 1. Implementace jedné z metod `LOCK` nebo `UNLOCK` je podmíněna implementací druhé z těchto metod. Pak se jedná o třídu shody 2.

`PROPFIND` Slouží ke zjištění vlastností zdroje. Pokud je zdrojem kolekce, vrátí server vlastnosti této kolekce a jejích členů (zdrojů v kolekci). Slouží tedy mimo jiné pro zjišťování obsahu kolekcí. Klient může požádat o všechny nebo některé konkrétní vlastnosti, případně o seznam vlastností definovaných na daném zdroji.

`PROPPATCH` Umožňuje přidávat, měnit a odebírat vlastnosti zdroje.

`MKCOL` Vytvoří novou kolekci. Spolu s vytvořením kolekce je možné také určit její vlastnosti, vytvořit její členy a určit jejich vlastnosti.

`GET a HEAD` Význam těchto metod je definován protokolem HTTP a v protokolu WebDAV je zachován. Význam aplikace metody `GET` na kolekce není přesně stanoven. Server má vrátit tělo kolekce, což může být index, obsah kolekce ve formě HTML dokumentu nebo cokoliv jiného.

`POST` Také tato metoda je převzata z HTTP beze změn. Protože význam této metody je definován serverem, nelze jej změnit ani pro kolekce.

`DELETE` Smaže zdroj. Pokud se jedná o kolekci, bude smazána včetně všech svých členů. Pokud nelze některý člen kolekce smazat, zůstanou nesmazány všechny jemu nadřazené kolekce, aby byla zachována konzistence hierarchie zdrojů.

`PUT` I tato metoda je převzata z protokolu HTTP. Slouží pro změnu těla zdroje. Vlastnosti zdroje nejsou ovlivněny (výjimkou jsou vlastnosti jako datová velikost a podobně). Pro vytváření kolekcí se používá metoda `MKCOL`.

`COPY` Kopíruje zdroje i celé kolekce včetně jejich vlastností v rámci jednoho serveru i mezi více servery. U kolekcí lze zvolit, zda se mají kopírovat včetně svých členů. Je také možné zvolit chování v případě, kdy cíl již existuje: buď se před kopírováním smaže, nebo se kopírování neprovede.

`MOVE` Přesun zdrojů a kolekcí. Detaily jsou podobné jako u `COPY`, ale nelze přesunout kolekci bez jejích členů.

`LOCK` Vytvoří nebo obnoví na zdroji zámek zvoleného rozsahu (sdílený, výlučný) pro zvolený přístup (čtení, zápis). Klient však musí předpokládat, že zámek může být kdykoliv zrušen například zásahem správce systému. Zámek ovlivňuje celý zdroj, to znamená jeho tělo, vlastnosti a u kolekcí i možnost přidávat nebo odebírat členy. U kolekcí lze zamknout pouze kolekci jako takovou nebo kolekci včetně všech jejích členů. Pokud je zdroj dostupný pod více URI, projeví se uzamčení na všech, neboť zámek se vztahuje na zdroj a nikoliv na URI.

`UNLOCK` Odstranění zámku na daném zdroji.

4.5 Hlavičky

Specifikace protokolu WebDAV popisuje následující hlavičky:

`DAV` Informuje klienta o tom, že daný zdroj podporuje WebDAV a v jaké třídě shody. Posílá se v odpovědi na metodu `OPTIONS`.

`Depth` Určuje, do jaké úrovně zanoření v hierarchii zdrojů bude metoda provedena. Buď lze pracovat pouze s určenou kolekcí (hodnota `0`), s kolekcí a jejími přímými členy (hodnota `1`) nebo s kolekcí a všemi jejími členy (hodnota `infinity`).

`Destination` Obsahuje cílové URI pro kopírování a přesun metodami `COPY` a `MOVE`.

`If` Určuje podmínky pro jednotlivá URI, za kterých je požadovaná metoda provedena.

`Lock-Token` Obsahuje identifikátor zámku. Používá se v požadavku metody `UNLOCK` pro zrušení zámku nebo v odpovědi na metodu `LOCK` jako předání identifikátoru vytvořeného zámku.

`Overwrite` Určuje, zda server má nebo nemá přepsat existující cíl při provádění metod `COPY` a `MOVE`.

`Status-URI` Může být odeslána spolu se stavovým kódem `102 Processing`. Obsahuje dvojice URI a příslušný stavový kód. Informuje klienta o výsledku metody pro jednotlivé zdroje.

`Timeout` Klient může tuto hlavičku odeslat spolu s požadavkem o zámek (metoda `LOCK`). Hodnota hlavičky udává požadovanou délku trvání zámku. Server tuto hodnotu nemusí dodržet ani respektovat.

4.6 Stavové kódy

K existujícím stavovým kódům protokolu HTTP jsou přidány tyto:

`102 Processing` Server přijal požadavek, ale zatím jeho zpracování nedokončil. Používá se tehdy, pokud je předpoklad, že splnění požadavku zabere delší čas (specifikace [4] doporučuje hodnotu 20 vteřin). Zabraňuje tak odpojení klienta z důvodu timeoutu (vypršení spojení). Po dokončení zpracování požadavku musí server poslat konečný stavový kód.

`207 Multi-Status` Označuje, že stavy jsou přenášeny v XML struktuře v těle zprávy. Umožňuje informovat o více operacích s různými výsledky.

`422 Unprocessable Entity` Server rozpoznal typ entity, syntaxe požadavku je správná, ale požadavek nelze zpracovat. Nastává například tehdy, pokud je XML struktura požadavku well-formed, ale sémanticky nedává smysl.

`423 Locked` Informuje o uzamčení zdroje.

`424 Failed Dependency` Informuje o neprovedení operace z důvodu selhání jiné akce, na které požadovaná operace závisí.

`507 Insufficient Storage` Nedostatek volného prostoru v datovém úložišti zabránil provedení operace.

Rozdělení stavových kódů do skupin podle první číslice zůstalo zachováno.

Kapitola 5

Existující implementace WebDAV serveru v PHP

K prostudování byla vybrána implementace dostupná v rámci projektu PEAR. Jedná se o balíček HTTP_WebDAV_Server, jehož domovská stránka se nachází na adrese http://pear.php.net/package/HTTP_WebDAV_Server. Podle informací z dokumentace k balíčku se jedná o implementaci plně vyhovující specifikaci protokolu WebDAV podle [4]. Pro její testování byl používán také nástroj litmus (http://www.webdav.org/neon/litmus/).

Autory balíčku jsou Hartmut Holzgraefe a Christian Stocker. První verze 0.9 byla uveřejněna v únoru 2003. Zatím poslední verze 1.0.0 RC4 pochází z listopadu 2006. Celkový počet stažení všech verzí překročil 25 000. Balíček je tedy poměrně hojně využíván, což poskytlo jeho vývojářům dostatečnou zpětnou vazbu. To je vidět i na seznamu nahlášených chyb, z nichž většina se v poslední verzi již nevyskytuje. Zkušenosti autorů z vývoje tohoto balíčku budou při implementaci vlastního serveru velmi užitečné.

5.1 Bázová třída

Hlavní částí balíčku je bázová třída `HTTP_WebDAV_Server`. Tvoří framework umožňující implementovat vlastní WebDAV server. Jedná se o abstraktní třídu, která musí být rozšířena přidáním metod pracujících s datovým úložištěm. Bázová třída je na datovém úložišti zcela nezávislá. Jejím úkolem je naopak zapouzdřit protokol WebDAV. Zpracovává a vytváří hlavičky a XML struktury a snaží se řešit známé problémy některých běžně používaných klientů. Samotným protokolem HTTP se nezabývá, neboť jeho zpracování zajišťuje webový server, na kterém skript běží.

Třída implementuje všechny metody protokolu WebDAV. Umožňuje tedy vytvořit server ve třídě shody 2. Záleží však na tom, jaké metody budou jejím uživatelem skutečně implementovány v rozšiřující třídě. Při běhu jsou automaticky zjištěny existující a neexistující funkce zpracovávající požadavky daných metod a podle toho jsou volány. Klientovi jsou odesílány příslušné hlavičky informující o schopnostech serveru.

5.2 Parsování XML

Těla požadavků v jazyce XML jsou zpracovávána pomocnými třídami. Jejich úkolem je zapouzdřit zpracování XML struktury a vytvořit z ní pole, které je poté dále zpracováváno. Každá ze tříd provádí parsování těla požadavku jedné metody a to `PROPFIND`, `PROPPATCH`

a `LOCK`. Všechny třídy pracují na stejném principu. Vlastní parsování se provádí s využitím XML parseru dostupného v prostředí PHP. Jedná se o SAX parser, jak byl popsán v kapitole 3.2 o XML.

5.3 Rozšiřující třída

Jako ukázka použití bázové třídy a také z důvodu umožnění jejího testování je v balíčku obsažena také rozšiřující třída `HTTP_WebDAV_Server_Filesystem`. Doplňuje bázovou třídu o práci s datovým úložištěm a tvoří tak spolu s ní provozovatelný WebDAV server. Obsahuje funkce implementující zpracování metod protokolu WebDAV na úrovni dat. Z bázové třídy jsou také volány funkce pro autentizaci a zjištění zámků k danému zdroji. Autentizace je však neřešena (funkce vrací vždy hodnotu `true` označující úspěšnou autentizaci) a spoléhá se na autentizaci v rámci HTTP protokolu. Existence funkce a její volání ale umožňuje používat vlastní autentizaci.

Jako datové úložiště pro zdroje a kolekce slouží souborový systém serveru. Zdroje jsou v něm přirozeně reprezentovány soubory a kolekce adresáři. Informace o zámcích a některé vlastnosti zdrojů se ukládají do dvou tabulek v databázi MySQL.

5.4 Další vlastnosti

Přestože je tato verze označována jako vyhovující specifikaci WebDAV [4], některé funkce nejsou ještě plně implementovány. Například v odpovědi na požadavek `PROPFIND` aplikovaný na kolekci nejsou uvedeny vlastnosti členů kolekcí obsažených v dotazované kolekci, přestože to klient požadoval uvedením hlavičky `Depth` s hodnotou `infinity`. Neimplementované části však nejsou běžně klienty využívány a nebrání tedy provozu serveru.

Jsou poskytovány zámky sdílené i výlučné, avšak pouze pro zápis. To je v souladu se specifikací, neboť ta umožňuje realizovat libovolnou kombinaci zámků. Zamykání kolekcí včetně rekurzivního zamykání jejich členů zatím podporováno není.

Kapitola 6

Dokumentový sklad IS FIT

Historie vývoje skladu je popsána v [5] a [6], odkud tato kapitola čerpá. Dokumentový sklad je na FIT provozován již od roku 2002. Během své existence byl několikrát přepsán a z nástroje pro ukládání textů diplomových prací se stal univerzálním úložištěm dokumentů poskytujícím pokročilé funkce jako určování majitelství dokumentů a nastavování přístupových práv, ukládání metadat a verzí dokumentů.

6.1 Verze 1

První verze dokumentového skladu sloužila pro ukládání textů diplomových prací a ročníkových projektů ve formátech PDF a PostScript. Oproti následným verzím byla poměrně jednoduchá. Pojmenování souborů a jejich zařazování do adresářů odpovídalo roku odevzdání práce a jejímu druhu podle vzoru `rok/BP/id.pdf`. Přístupová práva byla pevně nastavena tak, že každý student mohl číst a ukládat svou práci, ostatní uživatelé měli přístup ke všem pracem, avšak pouze pro čtení. Sklad však přes svou jednoduchost splňoval požadavky na něj kladené: umožňoval archivaci dokumentů, ke kterým se nepředpokládal častý přístup.

6.2 Verze 2

Druhá verze vznikla s cílem poskytnout fakultě stromově strukturovaný dokumentový sklad pro ukládání dokumentů obecně různého druhu a to včetně jejich metadat. Zatímco první verze se dá považovat za jakéhosi předchůdce, zde byl již realizován plně funkční dokumentový sklad. Je tedy zřejmé, že se od předchozí verze podstatně lišil. Bylo třeba vytvořit systém pro určování majitelství dokumentů a nastavování přístupových práv k nim. Mezi další požadavky patřila možnost uchovávání předchozích verzí dokumentů. Oproti první verzi bylo třeba vytvořit komplexní webové rozhraní pro správu skladu a jeho obsahu. Současně musela nová verze poskytovat dříve zavedené úložiště pro diplomové práce a také měla sloužit pro odevzdávání projektů v jednotlivých předmětech. Z toho vyplynula potřeba poměrně těsné vazby na stávající IS, aby studentem odevzdané soubory byly ukládány do správných adresářů, a také proto, aby se využila metadata, která již v systému existují jako například popis zadání diplomových prací.

Úložištěm vlastních dat dokumentů byl zvolen souborový systém, zatímco metadata dokumentů jsou ukládána v databázi. Ukládání všech dat přímo do databáze by při předpokládaném počtu dokumentů ve skladu zvětšilo její velikost na tolik, že by to zkomplikovalo

zálohování a případnou obnovu databáze a pravděpodobně i snížilo její odezvu.

Požadavky na organizaci a funkčnost skladu byly poměrně pestré a realizace takového skladu jako jednoho celku by byla těžkopádná. Bylo proto vytvořeno několik druhů skladu pro uchovávání různých dokumentů: pro odevzdávání projektů do předmětů, pro odevzdávání diplomových prací, pro ukládání výukových materiálů k předmětům, pro ukládání dokumentů do obecné adresářové struktury s nastavitelnými právy, aj. Některé vlastnosti měly být všem těmto druhům společné, například jednotné uživatelské rozhraní. Naopak systém pojmenování dokumentů a struktura skladu, přístupová práva a vazba na tabulky v IS je pro každý druh jiná.

Pro implementaci takového systému bylo s výhodou využito objektově orientovaného programování. Vlastnosti společné pro všechny typy skladu obsahovala bázová třída, od které byly odvozeny třídy implementující specifickou funkčnost pro jednotlivé mutace skladu. Všechny druhy skladů byly integrovány do jednoho stromu tak, že na první úrovni stromu byl určen typ skladu. Jednotlivé třídy poskytovaly kromě metod pro zjišťování cesty k souborům, kontrolu přístupových práv a metod pro vlastní práci se soubory jako upload, download, mazání a podobně také další metody, které jsou z hlediska principu funkce skladu zajímavější. Sem patří například metoda pro zjišťování atributů, které jsou v daném typu skladu pro dokumenty definovány, nebo metoda pro čtení obsahu adresáře, kde v jejím výstupu jsou ke každému souboru přiřazeny všechny jeho atributy. Díky těmto metodám lze uživatelské rozhraní zobrazovat univerzální metodou nezávisle na typu skladu, který si uživatel právě prohlíží.

Problémem této verze skladu byla koncepce, kdy dokument je reprezentován jednou třídou, což se ukázalo jako příliš omezující. Především parsování a validace cesty muselo být v každé třídě implementováno znovu, což zvyšovalo pravděpodobnost vzniku chyby, která by mohla být poměrně nebezpečná. V některých případech se ukázalo výhodné moci přecházet mezi mutacemi skladu v hierarchii libovolně, neomezovat se definicí typu skladu již na první úrovni stromu, ale umožnit tuto volbu i kdekoliv níže. Problematická byla také realizace operací se dvěma soubory současně.

6.3 Verze 3

Zatím poslední verze skladu vznikla s cílem odstranění omezení předchozí verze. Do bázové třídy dokumentu bylo implementováno více funkčnosti, aby se zjednodušil návrh odvozených tříd. Bylo přepsáno parsování a validace cesty s využitím principů podobných těm, které se používají v operačních systémech pro práci se souborovými systémy – cesta není zpracovávána jako celek ale po jednotlivých složkách, což je klíčovou změnou. Dále bylo odděleno uživatelské rozhraní od popisu dokumentu.

Sklad je tedy v této verzi tvořen následujícími třídami:

`doc_folder` představuje komponentu cesty

`doc_file` reprezentuje dokument, obsahuje pole objektů `doc_folder`

`doc_repos` zabaluje uživatelské rozhraní

Třída `doc_folder` je podobně jako u předchozí verze bázová třída, od které jsou odvozovány třídy jednotlivých typů skladů:

- odevzdané projekty

- soubory k předmětům
- strukturovaný adresář s právy a metadaty (obecný sklad)
- diplomové práce
- soubory k produktům, prototypům, vzorkům
- soubory k projektům, grantům
- soubory k publikacím
- FTP klient

Při parsování cesty k dokumentu se postupuje takto:

1. Vytvoří se objekt třídy `doc_file` reprezentující soubor dokumentu a spustí se parsování cesty.
2. Vytvoří se objekt třídy `doc_folder` představující kořen stromu dokumentového skladu a reference na něj se uloží do pole v objektu `doc_file`.
3. Vytvoří se pole obsahující v každé své položce jeden úsek cesty. Úseky jsou v cestě odděleny znakem / , jak je běžné.
4. Pro každý úsek cesty se pomocí objektu reprezentujícího předchozí úsek ověří, zda právě zpracovávaný úsek existuje, je platný a zda má uživatel práva ke vstupu do něj. Poté se vytvoří objekt třídy `doc_folder` nebo od ní odvozené, který reprezentuje tento úsek, a reference na něj se uloží jako další prvek do pole v `doc_file`.

Výsledkem změn je logičtější struktura skladu a snadnější práce s ním. Bylo umožněno v některých operacích pracovat přímo s daným souborem, zatímco jindy s jemu nadřazeným adresářem (parsování cesty se zastaví před poslední položkou cesty). Tím se zjednodušilo provádění některých operací, na druhou stranu je ovšem činnost skladu obtížněji pochopitelná. Významným důsledkem změn je možnost libovolně přecházet z jedné mutace skladu do druhé, takže každá položka v cestě může patřit do jiného typu skladu. V předchozí verzi byl typ skladu dán první položkou cesty a byl dále neměnný.

Díky tomu, že prohlížeč skladu generuje běžné HTML dokumenty s odkazy na soubory uložené ve skladu, lze pro vytvoření indexu pro fulltextové vyhledávání použít libovolný nástroj pro indexaci webového prostoru. Bylo však třeba vyřešit problémy, které se v běžném webovém prostoru nevyskytují. Indexace by měla být omezena jen na část skladu (není třeba indexovat například odevzdané projekty k předmětům). Musí se řešit i otázka práv, aby se díky vyhledávání uživatel nedozvěděl o existenci nebo obsahu jemu nepřístupných dokumentů. Volba padla na nástroj Mnogosearch, což je open source engine ukládající index do MySQL databáze. Podporuje dokumenty s více jazykovými verzemi, ohýbání slov a má další vhodné vlastnosti.

Použití částí dokumentového skladu je zabudováno do různých částí IS FIT s omezením na konkrétní podstrom skladu. Například studenti mají ze stránky předmětu přímý přístup k části dokumentového skladu obsahující soubory k tomuto předmětu jako jsou studijní opory, slidy k přednáškám a další, na jiném místě odevzdávají své projekty.

Kapitola 7

Návrh WebDAV rozhraní

Po prostudování struktury a funkce skladu byl zpracován detailní návrh WebDAV rozhraní. V této kapitole jsou však uvedeny pouze základní principy, detailům, které popisují spíše už samotnou implementaci, se bude věnovat kapitola následující.

Poznámka: V následujícím textu může mít slovo „metoda“ dva významy: označuje jak příkaz protokolu HTTP nebo WebDAV tak operace nebo funkce objektu z pohledu objektově orientovaného programování. Proto se bude pro jednoznačnost nadále „metodou“ rozumět příkaz protokolu a pro operace objektů bude použito slovo „funkce“.

7.1 Rozlišení WebDAV požadavku

Nejprve bylo třeba vyřešit otázku, jak rozlišovat, zda se ke skladu přistupuje WebDAV klientem nebo webovým prohlížečem za pomoci existujícího HTML rozhraní. Nabízela se dvě řešení: na základě použité adresy nebo na základě metody protokolu. V prvním případě by existovalo jedno URL pro přístup webovým prohlížečem a druhé pro přístup WebDAV klientem. Každému URL by odpovídal jeden PHP skript, který by buďto pracoval se skladem dosavadním způsobem nebo by zpracovával WebDAV požadavky a teprve na jejich základě by prováděl ve skladu další operace. Výhodou tohoto řešení by bylo zcela jasné rozlišení, zda se o WebDAV přístup jedná nebo ne. Nevýhodou pak nutnost vytvářet další skripty pro přístup ke skladu, které by patrně sloužily pouze k nastavení pár proměnných nebo vytvoření objektu pro práci s protokolem WebDAV.

V druhém případě, tedy rozlišení metodou protokolu, by se v existujícím PHP skriptu pro přístup ke skladu zjistila klientem použitá metoda a podle ní by se buď prováděl kód zpracovávající WebDAV požadavky nebo ne. Přitom je třeba, aby zpracování metod `GET`, `HEAD` a `POST` nebylo nijak ovlivněno, protože jsou používány také při přístupu přes HTML rozhraní. Tyto metody je třeba zpracovávat vždy stejným způsobem tak, aby odpovídaly jak požadavkům webového prohlížeče tak WebDAV klienta. Protože protokol WebDAV prakticky nijak nemění význam těchto metod [4], lze ponechat jejich obsluhu tak, jak je. Naopak metody, které jsou do protokolu HTTP nově přidány v rámci WebDAV rozšíření, stávající sklad nijak nezpracovává a proto nebude jeho funkce nijak ovlivněna, budouli se obsluhovat v novém WebDAV rozhraní. Existují však metody, které byly zavedeny v HTTP protokolu, ale jejichž význam nebo zpracování na straně serveru bylo WebDAVem do určité míry ovlivněno: `DELETE`, `OPTIONS` a `PUT`. V případě `OPTIONS` server musí v odpovědi uvést, že podporuje přístup protokolem WebDAV. Webový prohlížeč bude tuto informaci prostě ignorovat, není tedy problém posílat ji vždy. Zbylé dvě metody nejsou v dnešní době

prakticky vůbec používány a nebude činit komplikace, budou-li zpracovány vždy WebDAV rozhraním.

Výhodou rozlišování metodou je jednoduchost, není nutné vytvářet zvláštní URL pro přístup ke skladu. Proto bylo zvoleno toto řešení. To, že v případě požadavků typu `GET` a `HEAD` se bude sklad vždy chovat tak, jako by je odeslal webový prohlížeč a nikoliv WebDAV klient, není na závadu.

7.2 Integrace rozhraní do skladu

Dále bylo třeba navrhnout způsob realizace WebDAV rozhraní a jeho připojení ke stávajícímu kódu skladu. Přitom výsledné řešení by mělo zasahovat do původního kódu skladu co nejméně. Dalším cílem bylo co možná největší využití již existujícího kódu. Pro studium a další práci byly k dispozici pouze zdrojové kódy základních tříd skladu `doc_repos`, `doc_file`, `doc_folder` a z ní odvozené `doc_folder_dr`. Zvláště zásahů do `doc_folder_dr` bylo třeba se v maximální možné míře vyvarovat, aby integrace WebDAV rozhraní do ostatních typů skladů, která již nebyla úkolem v rámci této diplomové práce, byla co nejsnazší. Nabízelo se několik různých řešení, jejichž principy jsou zde nastíněny včetně jejich výhod i nevýhod a důvodů, které vedly k použití jednoho z nich.

Jednou z možností bylo vytvoření nové třídy `doc_repos` zabalující celý sklad, případně třídy od ní odvozené. V ní by byly nově implementovány její funkce `doc_oper` a `doc_show`. První jmenovaná zpracovává požadavky z HTML formulářů a ve spolupráci s funkcemi dalších objektů provádí operace jako parsování cesty, vytváření adresářů, upload a download souborů. Druhá funkce se primárně stará o sestavení HTML dokumentu představujícího uživatelské rozhraní skladu. Mimo to jsou z ní však také volány operace pracující s dokumenty ve skladu: kopírování, přesun, mazání a změna metadat. Kód obou funkcí je rozsáhlý, obsahuje mnoho různých kontrol a v podstatě řídí činnost celého skladu. Nově vytvořená třída by musela obsahovat kód zpracovávající WebDAV požadavky a na jejich základě by pak prováděla ve skladu činnosti podobně, jako to dělají zmíněné funkce stávající třídy `doc_repos`. Ve výsledku by bylo třeba prakticky celý kód původní třídy znovu napsat, což by při jeho rozsáhlosti mohl být poněkud problém. Navíc by se kód provádějící v podstatě tutéž činnost vyskytoval na dvou různých místech ve dvou různých podobách, což by značně zkomplikovalo případné budoucí změny. Výhodou by bylo velice těsné propojení kódu skladu s kódem pro zpracování WebDAV požadavků.

Jinou alternativou tohoto řešení je využití stávající třídy `doc_repos` a připsání nového kódu přímo do ní. Při běhu skriptu by se prováděl převážně původní kód, v případě potřeby by se provedl kus kódu řešící problematiku WebDAV protokolu a poté by se provádění vrátilo zpět do kódu původního. Tím se tato alternativa liší od předchozí možnosti, kde tok řízení skriptu by se odvíjel od toho, jaká metoda WebDAV protokolu se zpracovává a podle toho by se prováděly kusy kódu pracující se skladem. Výhoda těsného provázání skladu a WebDAV protokolu zůstává, kód by existoval pouze jedenkrát. Bylo by však třeba značně do kódu skladu zasahovat, což by při jeho složitosti mohlo způsobit chyby v jeho činnosti a navíc to odporuje požadavkům na snadnou udržovatelnost kódu.

Naprosto jiným přístupem je postavení kódu zpracovávajícího WebDAV protokol zcela mimo stávající sklad. Rozhraní by bylo tvořeno samostatným objektem, který by zaobaloval celý protokol a sklad od něj odstínil. Z pohledu tohoto objektu by se celý dokumentový sklad jevil jako černá skříňka, které se předá nějaký vstup a očekává se od ní výstup. Do činnosti skladu by nebylo nijak výrazně zasahováno, nebylo by to možné ani potřebné. Jakým způsobem však toto realizovat, když sklad považuje za vstupy data z HTML for-

mulářů, které pochopitelně u WebDAVu neexistují? Lze využít vlastnosti jazyka PHP, kde data odeslaná z formulářů jsou dostupná v polích `$_POST` případně `$_GET` a `$_REQUEST`, s kterými se pracuje stejně jako s každým jiným polem. Je možné v běžícím skriptu vytvořit tato pole podobně, jako by se vytvořila automaticky při přijetí formuláře. Nic tedy nebrání tomu, aby objekt WebDAV rozhraní uložením vhodných hodnot do těchto polí simuloval příjem odeslaného HTML formuláře, pak předal řízení skladu, který z nich přečte požadované hodnoty a bude pracovat stejně, jako by dostal požadavek odeslaný ze svého vlastního HTML rozhraní. Po skončení běhu skladu se opět dostane ke slovu WebDAV rozhraní, zpracuje výsledky získané ze skladu a zašle klientovi odpověď.

Výhodou tohoto řešení je maximální nezávislost skladu a WebDAV rozhraní. Jediné změny, které se budou muset provádět na více místech, by se týkaly změn HTML formulářů případně datových struktur skladu čtených rozhraním. Nevýhodou je naopak nulová kontrola nad činností skladu, což ovšem není nijak závažným nedostatkem. Větším problémem je, že sklad je navržen jako uzavřený celek, který přímo nevrací žádná data. WebDAV rozhraní však potřebuje být informováno nejen o výsledku operace (úspěch/neúspěch), ale vyžaduje přístup i k dalším datům, například obsahu adresářů, vlastnostem souborů a podobně. Taktéž simulaci formulářů nelze provést do úplných detailů, například v případě uploadu souborů, kdy se ve skladu používá funkce `move_uploaded_file`. Je tedy třeba zasahovat i do stávajícího kódu skladu, ale pouze v minimální míře. Většinou se bude jednat o krátké úseky kódu řešící tyto popsané nesrovnalosti.

Jiným problémem tohoto řešení je zpracovávání chyb. Sklad interně rozpoznává nastalé chyby většinou pouze booleovskými příznaky a jejich popis ukládá do textových řetězců. To je výborné řešení, pokud se chyby přímo vypisují uživateli. Při použití WebDAV protokolu se ale chyby označují pouze číselnými kódy, viz kapitola 4.6. Řešením je přidat do skladu proměnné pro uchovávání těchto kódů a v případě vzniku chyby do nich uložit příslušnou číselnou hodnotu. Posledním problémem je stávající HTML výstup skladu, který WebDAV klienti nedokáží zpracovat a nesmí jim být posílán. Naopak při použití webového prohlížeče být odeslán musí. Obalení každého úseku kódu generujícího výstup podmínkou by bylo velmi pracné. Sklad naštěstí používá bufferování výstupu, což znamená, že po dobu běhu skriptu se výstup ukládá do bufferu a na klienta je odeslán až po skončení běhu. Voláním funkce `ob_clean` lze buffer zahodit, což se bude provádět v rozhraní vždy před odesláním výstupu WebDAV klientovi.

S přihlédnutím ke všem výhodám a nevýhodám uvedených přístupů a náročnosti jejich realizace bylo vybráno poslední popsané řešení.

Kapitola 8

Implementace WebDAV rozhraní

Předchozí kapitola se věnovala obecným principům funkce vytvářeného WebDAV rozhraní a jeho zakomponování do dokumentového skladu. Tato kapitola již detailně popisuje výslednou implementaci včetně integrace do skladu, vytvořených objektů, jejich funkcí a řešení obsluhy jednotlivých metod protokolu WebDAV. Popisuje a hodnotí také nutné zásahy do dokumentového skladu a snaží se podat i popis zásahů nutných pro zprovoznění WebDAV funkčnosti i pro jiné typy skladu než je `doc_folder_dr`.

8.1 Připojení ke stávající implementaci skladu

Podle výsledků úvah z předchozí kapitoly má vytvořený objekt WebDAV rozhraní převádět příkazy WebDAV protokolu na pole simulující data přijatá z HTML formulářů. Následně se má spustit dokumentový sklad, který tato pole zpracuje. Je potřeba vyřešit, jakým způsobem bude sklad spouštěn a jak bude předávat objektu WebDAV rozhraní informace o výsledcích své činnosti.

Jak bude dále popsáno v částech týkajících se implementace metod `PROPFIND` (8.4.2) a `PROPPATCH` (8.4.7), je třeba, aby bylo možné sklad volat opakovaně v rámci zpracování jednoho požadavku. Přitom je žádoucí, aby sklad pracoval nezávisle na svých předchozích běžích, ale aby se informace získané z jednotlivých běhů uchovávaly. Naopak při obsluze metody `OPTIONS` se nemusí sklad volat vůbec. Objekt rozhraní tedy musí být pro zpracování jednoho požadavku vytvořen vždy právě jednou, zatímco objekt skladu nemusí být vytvořen vůbec nebo bude vytvářen opakovaně pro každý jeho běh znovu.

Připojení rozhraní ke skladu je nejsnáze pochopitelné ze zdrojových kódů doplněných popisem. Kódy jsou pro větší přehlednost zkráceny vynecháním řádků nedůležitých pro vysvětlení principu činnosti. Původně byl sklad používán takto:

```
$docs_params = array( ... );
$repos = new doc_repos($pathinfo, $self, $path_type, $docs_params);
$ok = $repos->doc_oper($vars, &$file);
is_header();
if(!$ok) $repos->docs_print_error();
else $repos->docs_show(&$file, $vars);
is_footer();
```

Konstruktoru objektu `doc_repos` se předává postupně: cesta v rámci skladu, adresa běžícího skriptu, typ cesty a pole parametrů ovlivňující funkčnost skladu.

Nově je sklad volán takto:

```
$do_webdav = (($_SERVER['REQUEST_METHOD'] == 'PROPFIND') || ... );
$do_loop = true; // without webdav make one loop
$method = ''; // nonsense without webdav
if($do_webdav) { // init webdav and process request
    require_once('docs_webdav.php');
    $webdav = new doc_webdav($self);
    $webdav->do_request1(); // process webdav request, part 1
    $do_loop = $webdav->do_loop;
    $method = $webdav->method; }
while($do_loop)
{
    $repos = new doc_repos($pathinfo, $self, $path_type, $docs_params,
                           $do_webdav, $method);
    $ok = $repos->doc_oper($vars, &$file);
    if($do_webdav)
    {
        $webdav->set_repos(&$repos); // webdav needs to read from repos
        $webdav->set_repos_file(&$file);
        $webdav->do_request2(); // process webdav request, part 2
        if($webdav->skip_part_3) {
            $do_loop = $webdav->do_loop;
            continue; }
    }
    is_header();
    if(!$ok) $repos->docs_print_error();
    else $repos->docs_show(&$file, $vars);
    is_footer();
    $do_loop = false; // if not using webdav, make only one loop
    if($do_webdav) {
        $webdav->do_request3(); // process webdav request, part 3
        $do_loop = $webdav->do_loop; }
}
```

Nejprve se na základě metody rozhodne, zda se bude WebDAV rozhraní používat nebo ne. Booleovský výraz obsahuje sadu porovnání pro jednotlivé metody. Kromě metod zavedených ve WebDAVu se rozhraním zpracovávají i HTTP metody `OPTIONS`, `PUT` a `DELETE`. Dále se nastaví proměnná `$do_loop` tak, aby se provedlo jedno volání skladu. To pro případ, že by se jednalo o požadavek `GET`, `HEAD` nebo `POST`, který se zpracovává pouze skladem. Následně se vytvoří objekt WebDAV rozhraní a zavolá se jeho funkce pro obecné zpracování požadavků (bude popsána dále v 8.3). Výsledkem je nastavené pole simulující přijetí HTML formuláře a hodnoty proměnných určujících opakované spouštění skladu.

Poté se v cyklu `while`, který je ale pro většinu WebDAV metod proveden pouze jedenkrát, vytvoří objekt dokumentového skladu a zavolá se jeho funkce pro zpracování požadavku `doc_oper`. Po jejím ukončení se činnost předá objektu WebDAV rozhraní voláním jeho funkce pro druhou část zpracování požadavku, která čte z objektu skladu výsledek jeho činnosti. Zde může dojít k ukončení běhu skriptu, pokud byl již požadavek splněn. V opačném případě se v rámci provádění funkce `do_request2` nastavily nové vstupy pro sklad a sklad se opakovaně spouští s volitelným prováděním funkce `doc_show` tak dlouho, dokud objekt WebDAV rozhraní nedokončí zpracování požadavku.

8.2 Objekty parserů

Než přistoupíme k popisu třídy `doc_webdav`, která tvoří WebDAV rozhraní skladu, je vhodné se zmínit o pomocných třídách pro parsování XML těl požadavků. Obě tyto třídy, `propfind_parser` a `proppatch_parser`, jsou si v principu velmi podobné a liší se jen tím, že je každá vytvořena pro zpracování jiných dat. Jak jejich název napovídá, jedna slouží pro parsování těla požadavku `PROPFIND`, zatímco druhá parsuje tělo požadavku `PROPPATCH`.

Úkolem těchto tříd je převedení strukturovaných XML dat do pole hodnot a několika proměnných, s kterými se dále pracuje v objektu `doc_webdav`. Využívají XML parser na principu SAX (viz kapitola 3.2) dostupný v prostředí jazyka PHP. Při zpracování XML dat se nespoléhá na to, že klienti dodržují specifikaci a data budou správná, ale kontroluje se přítomnost a zanoření všech elementů podle DTD pro daná XML, jak jsou uvedena v [4].

8.3 Objekt rozhraní

Pomineme-li XML parsery a funkci `put_finish` zmíněnou v kapitole 8.4.4, je celé WebDAV rozhraní implementováno v jediné třídě `doc_webdav`. Ta kromě funkcí týkajících se přímo zpracování jednotlivých požadavků obsahuje i několik funkcí obecných, které budou nyní popsány.

`__construct($script)` Konstruktor. Zálohuje pole `$_SERVER` obsahující informace související s během PHP skriptu a HTTP protokolem, registruje funkci pro zpracování chyb `webdav_err_handler` (je popsána v kapitole 8.5) a uloží do proměnné v objektu adresu běžícího skriptu.

`debug_log($msg)` Vypisuje předanou ladící zprávu nebo obsah pole do souboru včetně data a času. Logování lze vypnout změnou hodnoty proměnné `$debug` objektu `doc_webdav`. Proměnnou `$debug_file` se nastavuje cesta k souboru pro zápis zpráv.

`get_status($status)` Převede číslo stavového kódu na řetězec, který lze odeslat klientovi funkcí `header`. Současně provede zápis do logu o odeslání tohoto kódu. Texty odpovídající stavovým kódům jsou uloženy v poli `$statuses` v objektu `doc_webdav`. Pole neobsahuje všechny kódy ale jen ty skutečně používané.

`set_repos(&$repos)` Ukládá do objektu `doc_webdav` referenci na objekt dokumentového skladu `doc_repos`.

`set_repos_file(&$file)` Podobně jako předchozí funkce, ale ukládá se reference na objekt `doc_file` reprezentující soubor nebo adresář.

`split_path($path)` Odstraní z předané cesty ukončující lomítko. Následně cestu rozdělí podle nově posledního lomítka a části cesty vrátí v dvouprvkovém poli.

`send_errors` Kontroluje, zda při běhu skladu nastala chyba. Čte kódy chyb z objektů `doc_repos` a `doc_file`. Pokud k nějaké chybě došlo, provede o tom zápis do logu, odešle klientovi příslušný stavový kód a ukončí běh skriptu.

`my_die($stat, $msg = '')` Zapíše do logu předanou zprávu, vyprázdní výstupní buffer, odešle klientovi předaný stavový kód a ukončí běh skriptu.

`get_file_id($name)` Vytvoří identifikátor souboru daného názvu. Používá se při simuluaci výběrů souborů v HTML formuláři pomocí checkboxů. Výpočet id musí být stejný jako ve funkci `doc_show` objektu `doc_repos`, kde se formulář generuje a hodnoty identifikátorů se porovnávají.

`do_request1` Obecné zpracování WebDAV požadavku. Tato funkce musí být volána před vytvořením objektu třídy `doc_repos`. Provádí operace společné pro požadavky všech WebDAV metod. Následně vyhledá v objektu `doc_webdav` funkci obsluhující příslušnou metodu a zavolá ji. Pokud funkce neexistuje, znamená to, že zpracování metody není implementováno, o čemž informuje klienta. Díky tomuto mechanismu lze pohodlně přidávat obsluhu dalších metod pouze vytvořením příslušné funkce. V této funkci by měly být zpracovávány hlavičky `If`, to však nebylo implementováno. Na funkci rozhraní to nemá vliv, protože současní běžně používaní klienti tyto hlavičky neodesílají.

`do_request2` Obdobně jako předchozí, ale používá se po dokončení funkce `doc_oper` objektu `doc_repos`. Volá `send_errors` a příslušnou funkci obsluhy požadavku.

`do_request3` Volá se po dokončení funkce `doc_show` objektu `doc_repos`. Volání těchto tří funkcí je ukázáno v kapitole 8.1.

8.4 Zpracování jednotlivých metod

Následuje popis funkcí implementujících obsluhu jednotlivých metod protokolů WebDAV a HTTP. Funkce mají jednotné názvy tvaru `do_metoda`, `do_metoda2` a `do_metoda3`, aby se daly jednotným způsobem volat z funkcí `do_requestx`. U metod, jejichž zpracování je složitější nebo je sklad volán vícekrát, je zvlášť popsán princip činnosti. Jsou uvedena i omezení a neimplementované části zpracování metod a z toho plynoucí důsledky.

8.4.1 `OPTIONS`

`do_options` Vyhledá v objektu `doc_webdav` funkce obsluhující jednotlivé metody protokolu, sestaví hlavičky informující klienta o implementovaných metodách a odešle je.

Podle [3] by se odpověď měla vztahovat ke zdroji určeném v požadavku, tedy odpověď na požadavek `OPTIONS` by se mohla pro různé zdroje lišit. Pro jednoduchost je zpracování metody implementováno tak, že na adresu zdroje nebere ohled a pro každý zdroj dovoluje všechny metody, které WebDAV rozhraní poskytuje. Toto zjednodušení nemá na funkci rozhraní, klientů ani serveru jako celku žádný významný vliv.

8.4.2 `PROPFIND`

Metoda `PROPFIND` slouží ke zjišťování obsahu kolekcí (adresářů) a vlastností zdrojů. Jedná se tedy o základní funkci umožňující vůbec procházet adresářovou strukturou. Dokumentový sklad umožňuje zjistit obsah jednoho daného adresáře včetně vlastností jeho podadresářů a souborů.

Mohou nastat tyto situace:

- Klient chce zjistit vlastnosti souboru. Potom je třeba zjistit obsah adresáře, ve kterém je soubor uložen, a z výsledku vybrat pouze informace týkající se daného souboru.

- Klient chce zjistit vlastnosti a případně i obsah adresáře. Opět se ze skladu získá obsah nadřazeného adresáře, aby se získaly vlastnosti adresáře žádaného. Poté je třeba znovu zavolat sklad, aby vrátil výpis obsahu adresáře. Pakliže o to klient požádal, volá se sklad v cyklu, aby se zjistil obsah všech podadresářů.

- Klient chce zjistit vlastnosti a případně i obsah kořenového adresáře. Protože neexistuje adresář nadřazený kořenovému, jehož obsah by sklad mohl načíst a vrátit, vygeneruje vlastnosti kořenového adresáře WebDAV rozhraní. Sklad poté zjišťuje pouze obsah kořenového adresáře a jeho podadresářů.

Pro realizaci obsluhy metody `PROPFIND` musí být sklad volán v cyklu, což je ukázáno v kapitole 8.1. Přitom je potřeba uchovávat již načtené vlastnosti souborů nebo adresářů a také seznam adresářů, které teprve majít být skladem načteny. Pro tyto účely se používají následující proměnné objektu `doc_webdav`:

`$propfind_targets` je pole obsahující adresáře k prohledání.

`$propfind_actual` obsahuje index do pole `$propfind_targets` ukazující na právě zpracovávaný adresář.

`$propfind_results` je dvourozměrné pole obsahující vlastnosti souborů a adresářů získané ze skladu. Indexy pole jsou tvořeny úplnou cestou k souboru nebo adresáři, položky jsou pole vlastností.

Obsluhu požadavku provádějí následující funkce:

`do_propfind` Přečte a zkontroluje hodnotu hlavičky `Depth`. Provede parsování a kontrolu XML těla požadavku pomocí objektu `propfind_parser`. Nastaví první adresář k prohledávání na adresář nadřazený požadovanému souboru/adresáři posle hodnoty proměnné `$pathinfo`. Pokud je prohledávaným adresářem kořen, vygeneruje jeho vlastnosti a uloží je do pole výsledků. Pokud se navíc nepožaduje zjištění obsahu kořenového adresáře, volá funkci pro odeslání výsledků `propfind_send`. V opačném případě nastaví kořenový adresář jako první adresář k prohledávání. Do globální proměnné `$pathinfo`, která udává cestu v rámci skladu, uloží cestu k prvnímu prohledávanému adresáři a ukončí se. Tím dojde k vykonání funkce skladu `doc_oper`, čímž se obsah adresáře načte.

`do_propfind2` Tato funkce čte z objektu `doc_file` informace o obsahu adresářů a zároveň nastavuje další adresáře k přečtení. Rozlišuje, zda jde o její první volání nebo ne.

Při prvním volání se jedná o zjištění vlastností požadovaného souboru nebo adresáře z výpisu adresáře nadřazeného. Z objektu `doc_repos` je získáno pole s výpisem adresáře a hledá se v něm požadovaná položka. Pokud se najde a je to soubor, uloží se jeho vlastnosti do pole `$propfind_results` a zavolá se funkce `propfind_send`. Když se jedná o adresář a klient nepožádal o výpis jeho obsahu, pak je také po uložení jeho vlastností volána funkce `propfind_send`. V případě, že je potřeba načíst obsah adresáře, pak se jeho vlastnosti uloží, jeho jméno se přidá do pole `$propfind_targets`, nastaví se do proměnné `$pathinfo` a inkrementuje se `$propfind_actual`, takže v dalším běhu skladu po ukončení funkce `do_propfind2` bude obsah načten.

Jestliže byla funkce volána podruhé nebo opakovaně, jedná se o procházení adresářovou strukturou v cyklu. V tom případě se vlastnosti každé položky načteného

adresáře uloží do `$propfind_results`. Pokud se má prohledávat celý adresářový podstrom (hlavička `Depth` má hodnotu `infinity`), přidávají se všechny nalezené podadresáře do pole `$propfind_targets`. Po zpracování všech položek adresáře se kontroluje, zda zbývají další adresáře k načtení. Pokud ne, volá se funkce `propfind_send`. V opačném případě se nastaví do proměnné `$pathinfo` další adresář, inkrementuje se `$propfind_actual` a cyklus pokračuje.

Během provádění cyklu čtení adresářů se vůbec nevolá funkce `doc_show` objektu `doc_repos`, protože není k ničemu potřebná.

`propfind_send` Odesílá klientovi XML dokument obsahující výsledek volání metody – načtené vlastnosti souborů a adresářů. Obsahuje dva zanořené cykly: vnější jde přes všechny položky pole `$propfind_results` reprezentující zdroje, vnitřní pak zpracuje všechny jejich vlastnosti. Ve vnitřním cyklu se sleduje, které vlastnosti byly nalezeny, a po jeho doběhnutí se odešle seznam vlastností klientem požadovaných avšak nenalezených. Poté se pokračuje odesíláním informací o dalším souboru/adresáři. Po zpracování celého pole se běh skriptu ukončí. Indexy položek pole vlastností se porovnávají s řetězcovými konstantami, které je třeba udržovat konzistentní se skladem.

V odpovědi na metodu `PROPFIND` se vyskytují jednak vlastnosti definované v RFC WebDAVu [4], jednak některé vlastnosti definované skladem. Všechny vlastnosti jsou dodané skladem, WebDAV rozhraní žádné další nevytváří. Zde je jejich výčet:

`displayname` popis zdroje vhodný pro zobrazení člověku

`resourcetype` povaha zdroje – adresář nebo obecné označení typu souboru (obrázek, textový soubor, dokument MS Office apod.)

`getcontenttype` typ obsahu zdroje – odpovídá hlavičce `Content-Type`

`getlastmodified` datum a čas poslední změny zdroje (nebo jeho vlastností) – odpovídá hlavičce `Last-Modified`

`getcontentlength` délka (datová velikost) zdroje – odpovídá hlavičce `Content-Length`

`creationdate` datum a čas vytvoření zdroje

`name` český popisek

`name_e` anglický popisek

`info` popis, komentář

`versions` příznak ukládání verzí (pouze u souborů)

V informacích o adresáři sklad vrací místo velikosti v bytech počet jeho členů. Pokud se tato hodnota odešle jak vlastnost `getcontentlength`, nejsou někteří klienti schopni odpověď zpracovat a obsah adresáře nezobrazí. Proto je třeba při odesílání velikosti rozlišovat soubory a adresáře a u adresářů velikost neodesílat.

Zpracování metody `PROPFIND` plně odpovídá specifikaci [4]. Nejsou implementovány části tykající se práce se zámky, které jsou označeny jako nepovinné.

8.4.3 MKCOL

`do_mkcol` Vstupem je globální proměnná `$pathinfo` obsahující plné jméno vytvářeného adresáře (tedy včetně cesty). Tento řetězec se funkcí `split_path` rozdělí na jméno vytvářeného adresáře a cestu k adresáři nadřazenému. Do `$pathinfo` se zapíše cesta k nadřazenému adresáři, jméno vytvářeného adresáře se uloží do pole `$_POST`. Nastaví se další položky pole `$_POST`, aby se simulovalo přijetí formuláře pro vytvoření adresáře a funkce končí.

`do_mkcol2` Odešle informaci o úspěšném vytvoření adresáře, případné chyby byly odeslány již dříve funkcí `send_errors`.

Není implementováno zpracování nepovinného těla požadavku. Podle specifikace [4] má sloužit k nastavování vlastností adresáře a vytváření jeho prvků a jejich vlastností. Jeho formát ale není popsán. Testovaní klienti tělo neposílají.

8.4.4 PUT

Upload souborů se ve skladu řeší odesláním formuláře metodou `POST`. Následně se pracuje s polem `$_FILES`, které obsahuje informace o nahrávaných souborech, uložení souboru do příslušného adresáře provádí funkce `move_uploaded_file`. WebDAV rozhraní dokáže naplněním polí `$_POST` a `$_FILES` simulovat odeslání formuláře, ale nelze provést simulaci takovou, aby správně pracovala i funkce `move_uploaded_file`. Řešením je nahrávaný soubor uložit do dočasného souboru a ve skladu pak místo `move_uploaded_file` podmíněně volat funkci `copy`. Před ukončením skriptu se dočasný soubor smaže. Sklad umožňuje upload více souborů jedním odesláním formuláře. Tento mechanismus by bylo možné využít při uploadu více souborů metodou `PUT`, avšak testovaní klienti odesílají každý soubor ve zvláštním požadavku.

`do_put` Rozdělí cestu k nahrávanému souboru z proměnné `$pathinfo` na jeho vlastní jméno a cestu k mu nadřazenému adresáři. Zkontroluje obsah hlaviček `Content-*`, jak vyžaduje specifikace [3]. Zaregistruje funkci `put_finish`, aby byla volána po ukončení běhu skriptu. Obsah nahrávaného souboru uloží do souboru v adresáři `/tmp/`. Nastaví hodnoty polí `$_POST` a `$_FILES`, do proměnné `$pathinfo` zapíše cestu k nadřazenému adresáři a ukončí se.

`do_put2` Odešle informaci o úspěšném nahrání souboru, případné chyby byly odeslány již dříve funkcí `send_errors`. Podle specifikace [3] by se mělo rozlišovat vytvoření nového souboru a přepis existujícího. Protože ale sklad neposkytuje prostředek, jak toto detekovat, oznámí se vždy vytvoření nového souboru, což specifikaci vyhovuje.

`put_finish` Tato funkce stojí mimo objekt `doc_webdav`. Jejím úkolem je po jakémkoliv ukončení skriptu, pokud se zpracovávala metoda `PUT`, smazat vytvořený dočasný soubor.

Není implementováno zpracování hlaviček `Content-Encoding` (umožňuje přenos komprimovaných dat), `Content-MD5` (obsahuje hash přenášených dat), `Content-Range` (upload části souboru) a `Content-Language` (jazyk). Poslední uvedená není považována za kritickou a je ignorována. Ostatní tři způsobí v souladu se specifikací [3] ukončení zpracování požadavku, což v důsledku znamená neuložení souboru na server. Testovaní klienti však tyto hlavičky neodesílají a tak není funkčnost dotčena. Není podporováno ani nahrávání více souborů v jednom požadavku (multipart upload), který klienti také nepoužívají.

8.4.5 DELETE

`do_delete` Zkontroluje, zda je hlavička `Depth` nastavena na hodnotu `infinity`, jak vyžaduje specifikace [4]. Rozloží cestu k mazanému zdroji přečtenou z proměnné `$pathinfo` na jeho název a cestu k mu nadřazenému adresáři voláním funkce `split_path`. Pomocí funkce `get_file_id` zjistí identifikátor souboru a naplní pole `$_POST` a `$_REQUEST` a proměnnou `$pathinfo`.

`do_delete2` Nedělá nic, existuje jen z důvodu zachování jednotných názvů a způsobu volání funkcí.

`do_delete3` Odesílá stavový kód informující o úspěšném provedení operace.

8.4.6 COPY a MOVE

Zpracování těchto dvou metod se až na detaily nijak neliší. Jsou tedy obě obsluhovány funkcemi `do_copymove`x. Funkce `do_copy`x a `do_move`x a obsahují pouze volání těchto nových funkcí.

`do_copymove` Načte hlavičky `Depth` a `Overwrite` a zkontroluje jejich hodnoty. Dále načte hlavičku `Destination` určující cílovou URL kopírování nebo přesunu a zkontroluje, zda ukazuje do prostoru dokumentového skladu. Rozdělí zdrojovou a cílovou cestu voláním funkce `split_path`. Do proměnné `$pathinfo` uloží cestu k nadřazenému zdrojovému adresáři, nastaví pole `$_POST` a `$_REQUEST` a ukončí se.

`do_copymove2` Nedělá nic.

`do_copymove3` Informuje klienta o úspěšném dokončení operace. Podobně jako u `PUT` by se mělo rozlišovat, zda došlo nebo nedošlo k přepsání existujících souborů. Sklad ale neumožňuje soubory kopírováním nebo přesouváním přepisovat, takže se odesílá vždy tatáž informace.

Není implementováno zpracování těla požadavku, kde lze stanovit, které vlastnosti zdrojů se mají přenést do nového umístění. Sklad tuto funkčnost neposkytuje a klienti tělo neodesílají.

8.4.7 PROPPATCH

Při implementaci obsluhy této metody bylo třeba vyřešit dva problémy:

- Všechny vlastnosti se nastavují jedním formulářem. Pro správnou funkci rozhraní a simulování tohoto formuláře je třeba nejprve zjistit stávající hodnoty vlastností, aby mohly být do formuláře vyplněny a nedošlo tak k jejich nežádoucí změně na prázdné hodnoty, pokud klient mění pouze jednu vlastnost.
- WebDAV dovoluje přiřazovat zdrojům libovolné vlastnosti, zatímco ve skladu jsou vlastnosti pevně definovány. Je třeba kontrolovat měněné vlastnosti a v případě nepovolených vlastností operaci ukončit a informovat klienta o chybě.

`do_proppatch` Nejprve se ověří, že klient chce měnit vlastnosti souboru nebo adresáře uvnitř nějakého druhu skladu. Pomocí objektu `proppatch_parser` se načte a zkontroluje tělo požadavku, čímž se zjistí, které vlastnosti chce klient měnit a jaké hodnoty

chce do nich zapsat. Každá vlastnost je potom porovnána s polem povolených vlastností `$proppatch_allowed` obsaženým v objektu `doc_webdav`. Je-li nalezena nepodporovaná vlastnost, volá se funkce `proppatch_send`, která odešle klientovi chybovou zprávu. V opačném případě se nastaví proměnná `$pathinfo` tak, aby sklad přečetl obsah adresáře, ve kterém se nachází zdroj, jehož vlastnosti mají být měněny.

`do_proppatch2` Rozlišuje se, zda se tato funkce provádí v první nebo druhé fázi zpracování požadavku. První fází se rozumí načtení informací o zdroji a tím stávajících hodnot jeho vlastností. Ve druhé fázi se provádí změna hodnot vlastností. Fáze je určena hodnotou proměnné `$proppatch_phase`.

První fáze se podobá čtení obsahu adresáře při zpracování metody `PROPFIND`. V poli získaném z objektu `doc_file` se vyhledá zpracovávaný soubor nebo adresář a jeho vlastnosti se zkopírují do pomocného pole. Znovu se zkontroluje, které vlastnosti chce klient nastavovat, protože soubory a adresáře mají definovány různé množiny vlastností. Tuto kontrolu nebylo možno provést v předchozím kroku, protože nebylo známo, zda se jedná o soubor nebo adresář. Pole `$_POST` a `$_REQUEST` se naplní tak, aby simulovalo nezměněný formulář. Následně se do těchto polí zapíší nové hodnoty měněných vlastností. Nakonec se nastaví druhá fáze zpracování a povolí se cyklus volání skladu, avšak s vynecháním volání funkce `doc_show`, která je zatím nepotřebná.

Ve druhé fázi tato funkce pouze zabrání dalšímu spuštění skladu v cyklu a povolí volání funkce `doc_show`, která ve skladu provede změnu vlastností zdroje.

`do_proppatch3` Volá funkci `proppatch_send`.

`proppatch_send` Sestaví a odešle klientovi XML dokument informující o výsledku operace. Prochází v cyklu pole obsahující seznam měněných vlastností a pro každou vytvoří příslušné XML elementy.

Lze nastavovat tyto vlastnosti:

`name` český popisek

`name_e` anglický popisek

`info` popis, komentář

`versions` příznak ukládání verzí (pouze u souborů)

Hodnoty všech vlastností, které lze metodou `PROPPATCH` nastavovat, jsou zjistitelné metodou `PROPFIND`.

8.4.8 `LOCK` a `UNLOCK`

Tyto metody jsou podle [4] nepovinné a klienti, o kterých se předpokládá, že budou používáni pro přístup ke skladu, je nepoužívají. Nebyly tedy implementovány. Důkladný návrh jejich realizace je uveden v kapitole 10.1.

8.5 Provedené změny v dokumentovém skladu

Implementace WebDAV rozhraní si vyžádala změny v kódu dokumentového skladu, zde je jejich přehled uspořádaný podle souborů a tříd:

- Nový způsob volání skladu, je popsán v kapitole 8.1.
- Vytvoření funkce `webdav_err_handler` v souboru `common.php`. Slouží jako obsluha chyb, které nastávají při běhu skriptu. Jejím úkolem je zapisovat chyby do souboru a v případě výskytu závažné chyby ukončit běh skriptu a odeslat klientovi stavový kód informující o chybě serveru. Důvodem pro vytvoření této funkce byla potřeba odstranění chybových výpisů ze standardního výstupu, který se odesílá klientovi.
- Do všech tříd dokumentového skladu byla přidána proměnná `$error_code` pro uložení stavového kódu popisujícího chybu vzniklou při provádění operací ve skladu. Současně bylo do všech míst, kde se nastavují ostatní proměnné indikující chybu, doplněno přiřazení příslušné hodnoty do této proměnné.
- Ve třídě `doc_repos` byly provedeny následující úpravy:
  - Byly vytvořeny proměnné `$do_webdav` a `$method`. První z nich je booleovská proměnná sloužící k rozlišení, zda se zpracovává WebDAV požadavek nebo se má provádět původní kód skladu obsluhující požadavek z HTML rozhraní. Druhá obsahuje jméno zpracovávané metody. Současně byl upraven také konstruktor, aby přijímal a nastavil hodnoty těchto proměnných.
  - Ve funkci `do_oper`:
    * Po vytvoření objektu `doc_file` se podmíněně volá jeho nově vytvořená funkce `doc_check_webdav`, která zkontroluje, zda všechny vytvořené objekty `doc_folder_typ` reprezentující cestu ke zpracovávanému souboru nebo adresáři jsou upraveny tak, aby podporovaly zpracování WebDAV požadavků. Pokud tomu tak není, uloží se chybový kód a běh skladu se ukončí. Chyba je následně detekována v rozhraní a odeslána klientovi.
    * Po volání funkce `doc_do_oper` objektu `doc_file`, která provádí vytváření nových adresářů, se v případě zpracování WebDAV požadavku ukončí běh skladu a pokračuje se prováděním kódu rozhraní, kde se odešle kód popisující výsledek operace. Původně se běh skriptu ukončil a došlo k přesměrování na novou stránku.
    * Při zjišťování, zda se má zpracovávat upload souboru, je upravena podmínka volající funkci `is_uploaded_file`. Funkce zodpovědné za uložení nahrávaného souboru se tak provedou i v případě zpracovávání metody `PUT`, kdy nelze upload odsimulovat do posledních detailů a výsledek volání funkce `is_uploaded_file` by zapříčinil neprovedení uploadu.
    * Po dokončení uploadu ukončení běhu skladu místo přesměrování podobně jako při vytváření adresářů.
    * Odeslání stavového kódu `404 Not Found` v případě, kdy požadovaný soubor nebo adresář neexistuje. Dříve se pouze vypsalo chybové hlášení, které člověku stačí. WebDAV klienti však obdrželi kód `200 OK` a pracovali tedy stejně, jako by soubor existoval, což způsobilo nemožnost provádění některých operací.

  - Ve funkci `doc_show` ukončení běhu skladu a návrat do WebDAV rozhraní po provedení mazání, kopírování a přesunu.

- Ve třídě `doc_file` byla vytvořena funkce `doc_check_webdav`. Zjišťuje, zda všechny objekty `doc_folder`, které byly vytvořeny při parsování cesty k souboru/adresáři, byly upraveny pro podporu WebDAVu. Rozhodování se provádí na základě existence a hodnoty proměnné `$webdav_sup_`třída v objektu `doc_folder_`typ. Například pokud byla třída `doc_folder_dr` upravena, obsahuje proměnnou `$webdav_sup_doc_folder_dr` nastavenou na hodnotu `true`. Zavedení těchto názvů proměnných bylo nutné kvůli dědičnosti, kdy by neupravená třída obsahovala proměnnou z bázové třídy a kontrola by nefungovala.

- Třída `doc_folder` byla upravena takto:

  - Vytvořena proměnná `$webdav_sup_doc_folder` oznamující podporu WebDAVu.
  - Funkce `doc_do_oper` byla upravena tak, aby při pokusu o vytvoření adresáře metodou `MKCOL` vracela chybu. Původně se chyba nevracela, protože HTML rozhraní neumožňuje na této úrovni adresáře vytvářet.

- Ve třídě `doc_folder_dr` byly provedeny tyto úpravy:

  - Vytvořena proměnná `$webdav_sup_doc_folder_dr` identifikující podporu protokolu WebDAV.
  - Ve funkci `doc_get_dir` je kromě původních vlastností zdrojů vracen navíc mimetyp (WebDAV vlastnost `getcontenttype`), čas modifikace souboru ve formátu unix timestamp pro snadnější zpracování (`getlastmodified`) a také komentář (`info`).
  - Ve funkci `doc_upload` je místo funkce `move_uploaded_file` v případě uploadu přes WebDAV metodu `PUT` volána funkce `copy`.
  - Funkce `doc_formattr`, která mění vlastnosti souborů a adresářů, byla upravena tak, aby při provádění metody `PROPPATCH` nebyl ovlivněn stav zámku. Tato úprava byla nutná proto, že zámek se nastavuje přes jeden formulář spolu s ostatními vlastnostmi a nesmí být metodou `PROPPATCH` ovlivněn.

Pokud má být zavedena podpora WebDAVu do dalších typů skladů, musí se příslušné třídy odvozené od `doc_folder` upravit v duchu následujících bodů:

- Vytvoření proměnné `$error_code` a na vhodných místech nastavení její hodnoty, aby popisovala vzniklou chybu. Tento krok není vyloženě nutný, protože WebDAV rozhraní rozpozná chybu i ze stávající proměnné `$error`, ale klientovi pak odešle obecný stavový kód `500 Internal Server Error`. Tato úprava bude patrně představovat většinu zásahů do kódu třídy.

- Provedení úprav ekvivalentních popsaným úpravám třídy `doc_folder_dr`.

- Provedení úprav specifických pro danou třídu. Tyto zde nemohu být popsány, protože zdrojové kódy všech tříd skladu nebyly při řešení práce k dispozici.

Výčet změn a zásahů do původního kódu skladu není zrovna krátký. Na druhou stranu provedení většiny těchto změn znamenalo připsání jen několika málo řádků kódu obalených

jednou podmínkou. Největší objem změn představuje nastavování a předávání stavových kódů popisujících chyby. Také požadavky na změny, které je nutné provést pro umožnění přístupu protokolem WebDAV k dalším typům skladu, nejsou nijak rozsáhlé. Dá se tedy říci, že požadavek minima zásahů do kódu skladu byl splněn.

8.6 Omezení funkcí protokolu WebDAV

Dokumentový sklad obecně poskytuje stejnou funkčnost jako protokol WebDAV. Přesto však s ním není zcela kompatibilní v tom smyslu, že určité detaily při zpracování WebDAV požadavků nebylo možno implementovat, protože sklad danou funkčnost nenabízí. Některé z těchto nesrovnalostí řeší WebDAV rozhraní, jako například rekurzivní výpis adresářového stromu opakovaným voláním skladu. Většinu však vyřešit možné nebylo a ty jsou popsány v následujícím výčtu. Je možné, že některé body platí pouze pro sklad typu dr a v některé další verzi skladu budou odstraněny, naopak pro sklady jiných typů zde příslušný bod může chybět.

- V kořenovém adresáři skladu nelze provádět žádné operace mimo výpisu jeho obsahu. Podobné omezení bude pravděpodobně existovat i v jiných adresářích podle typu prohlíženého skladu.
- Nelze mazat neprázdné adresáře.
- Se soubory, u nichž se ukládají verze, nelze provádět některé operace.
- Kopírování a přesouvání lze provádět pouze v rámci skladu a ne mezi různými WebDAV servery. Toto je záměrné omezení implementované na úrovni WebDAV rozhraní.
- Při zpracování metod `COPY` a `MOVE` se ignorují hlavičky `Depth` a `Overwrite`. První z nich je platná pouze při kopírování a určuje, zda se má kopírovat pouze samotný adresář nebo i jeho obsah. Druhá hlavička nastavuje chování serveru v případě, že cílový adresář nebo soubor již existuje. Lze zvolit, zda se v takovém případě má operace provést, což znamená smazání existujících souborů a adresářů před kopírováním nebo přesunem, nebo se má zpracování požadavku zastavit. Sklad kopírování pouze adresářů bez obsahu neumožňuje a přepis existujících souborů nedovoluje nastavovat.
- Dalším omezením při kopírování je nemožnost vybrat vlastnosti souborů, které se budou kopírovat spolu s daty. Přenášejí se vždy všechny vlastnosti. Z tohoto důvodu není v rozhraní implementováno zpracování těl požadavků metod `COPY` a `MOVE`.
- Nelze kopírovat adresáře.
- Při kopírování a přesouvání není možné skladu předat cílové jméno souboru nebo adresáře. Například nelze soubor `/dr/adresar_1/soubor_A.txt` kopírovat nebo přesunout do `/dr/adresar_2/soubor_B.txt`. Podobně nelze soubory ani adresáře přejmenovávat, například `/dr/adresar_1/soubor_A.txt` na `/dr/adresar_1/soubor_B.txt`.
- Metodou `PROPPATCH` lze nastavovat pouze definované vlastnosti. Podle specifikace [4] lze zdrojům přiřazovat vlastnosti libovolných názvů.
- Není podporován částečný upload souboru. Tento případ je detekován a zpracován v rozhraní, požadavek je odmítnut.

Důsledky uvedených omezení nejsou nijak kritické. V nejhorším případě sklad nastaví chybový kód a operace se neprovede. WebDAV rozhraní pak odešle tento kód klientovi. Výsledek se podobá situaci, kdy chce uživatel provést operaci, pro kterou nemá dostatečná práva.

Kapitola 9

Testování klienty

Pro testování vytvářeného rozhraní bylo použito několik různých nástrojů a klientů. Jejich popis je uveden v následující podkapitole. Dále je popsána metodika testování a nakonec jsou shrnuty dosažené výsledky.

9.1 Popis klientů

9.1.1 Telnet

Telnet samozřejmě není WebDAV klientem, je zde uveden spíše pro úplnost. Byl používán zvláště v raných fázích vývoje, protože umožňuje přímo zadávat data, která se odešlou serveru. Je tedy nutné vypisovat celé zprávy protokolu, což je z hlediska praktického použití nemyslitelné, avšak při ladění nedocenitelné. Lze tak předávat serveru přesně definované požadavky testující každou jeho konkrétní vlastnost a funkci.

9.1.2 Cadaver

Cadaver je WebDAV klient pro unixové operační systémy pracující v příkazové řádce. Implementuje všechny metody protokolu včetně nepovinného zamykání. Běžní uživatelé ho patrně používat nebudou, neboť jsou zvyklí na grafické prostředí. Jeho výhodou je především to, že není omezen grafickým rozhraním a podporuje všechny funkce protokolu WebDAV, a to i ty, které dále popsaní klienti nenabízí. Při testování implementace těchto metod byl tedy jediným použitelným klientem. Například jako jediný z použitých klientů zobrazuje všechny vlastnosti zdrojů. Další jeho výhodou je, že pracuje na poměrně nízké úrovni. Uživatel zadává textové příkazy, které více méně odpovídají jednotlivým metodám protokolu. Nedochází tak k odesílání více požadavků za sebou bez vědomí uživatele. Při neúspěchu operace nevypisuje vlastní chybové hlášení, které v některých případech může být poněkud matoucí, ale stavový kód přijatý od serveru. Pro testování a ladění WebDAV rozhraní se ukázal být dobrou volbou.

9.1.3 Klient integrovaný v MS Windows

Operační systémy MS Windows obsahují integrovaného WebDAV klienta. Předpokládá se, že tento bude nejčastěji používaným pro přístup k dokumentovému skladu. Integrace je provedena do té míry, že obsah WebDAV serveru je zobrazován v okně průzkumníka podobně jako obsah místních adresářů.

Pro testování byl použit klient ze systému MS Windows XP SP1. Aby bylo možné pracovat se soubory na serveru, je třeba vytvořit nové připojení. To se provádí v ovládacím panelu *Místa v síti*. Vytvoříme nové místo, zadáme URL skladu včetně protokolu `http://` a PHP skriptu skladu, libovolný název, přihlašovací údaje a potvrdíme. V *Místech v síti* se vytvoří nová ikona odkazující na server. Pokud na ni poklepeme, otevře se okno zobrazující obsah serveru a můžeme začít pracovat s jeho obsahem.

Klient nabízí pouze základní funkce, volitelné zamykání neimplementuje. Neumožňuje měnit vlastnosti zdrojů, dokonce je ani všechny nevypisuje. Lze zjistit pouze název, velikost, mimetyp a čas vytvoření a poslední změny souboru nebo adresáře. Pro běžné použití spočívající v přenosu souborů je ale vhodný a snadno použitelný. Umožňuje provádět operace hromadně s více zdroji naráz, což je realizováno odesláním jednoho požadavku pro každý zdroj.

9.1.4 Konqueror

Konqueror je webový prohlížeč a správce souborů integrovaný do desktopového prostředí unixových operačních systémů KDE. Mimo jiné se dá používat i jako WebDAV klient. Z pohledu uživatele se nijak výrazně neliší od klienta zabudovaného v MS Windows. Nabízí stejnou funkčnost a i na úrovni protokolu se chová přibližně stejně.

9.1.5 Litmus

Litmus je nástroj pro testování WebDAV serverů. Připojí se k serveru jako běžný klient a provádí řadu operací s cílem zjistit, zda server pracuje ve shodě se specifikací protokolu [4]. Vzhledem ke specifické architektuře dokumentového skladu, jeho pevně dané adresářové struktuře a dalším vlastnostem nakonec nebyl pro testování rozhraní použit.

9.2 Popis testování

Testování probíhalo po celou dobu implementace rozhraní. Byl používán zejména klient cadaver, v některých případech program telnet. Teprve až byly implementovány všechny WebDAV metody, byl použit klient z MS Windows a Konqueror.

Testování spočívalo v zasílání příkazů na server, odchytávání síťové komunikace nástrojem Wireshark a sledování ladících výpisů vytvářených WebDAV rozhraním. Každým klientem byly otestovány všechny jím proveditelné operace v různých adresářích skladu.

9.3 Výsledky a zhodnocení

S klientem cadaver bylo dosaženo plné funkčnosti, to znamená vše, co bylo ve WebDAV rozhraní skladu implementováno, funguje. Při použití druhých dvou klientů je také spolupráce se serverem výborná, i když některé funkce klienti nenabízejí. Hromadné operace se soubory a adresáři byly shledány funkčními. Přesto se objevilo několik nedostatků:

- Klient v MS Windows chybně zpracovává časové zóny – čas je o hodinu posunutý. V Konqueroru toto funguje správně. Cadaver zobrazuje čas tak, jak jej zaslal server, a jeho formát nemění, takže vliv časových zón se neprojeví.
- Klient v MS Windows i Konqueror si pamatují obsah adresářů. Například uživatel otevře adresář A a smaže v něm nějaké soubory, poté přejde do jiného adresáře, pak

se opět vrátí do adresáře A a uvidí v něm smazané soubory. Je nutné znovu načíst obsah adresáře například klávesou F5. Toto chování přetrvalo i když WebDAV rozhraní odesílalo hlavičky `Expires` s datem v minulosti a `Cache-control` s hodnotami `must-revalidate`, `no-cache` a `max-age=1`, což by mělo podle [3] zabraňovat ukládání do cache.

- Při požadavek o vytvoření adresáře klient z MS Windows nejprve vytvoří adresář pojmenovaný `New Folder` a následně ho přejmenuje na uživatelem zadaný název. Protože sklad neumí přejmenovávat adresáře, operace se nezdaří. Nový adresář požadovaného jména lze vytvořit jeho zkopírováním z jiného umístění, přičemž kopírovaný adresář musí obsahovat alespoň jeden soubor.
- V Konqueroru nelze na server nahrávat soubory. Jedná se o chybu klienta, protože upload se nezdařil ani na jiný server.
- Změna vlastností zdrojů byla ověřena pouze klientem cadaver, ostatní klienti touto funkcí nedisponují.

WebDAV rozhraní bylo shledáno funkčním a je použitelné spolu s popsanými klienty.

Kapitola 10

Možnosti dalšího vývoje

Vytvořené WebDAV rozhraní poskytuje takovou funkčnost, že je bez problémů použitelné pro práci ve spojení s dříve uvedenými klienty. Některé pokročilé, volitelné nebo zřídka používané funkce však implementovány nebyly.

Předpokládaným směrem dalšího vývoje je úprava zbývajících podtypů dokumentového skladu tak, aby k nim bylo možné přistupovat přes protokol WebDAV za pomoci vytvořeného rozhraní. To je však již nad rámec této diplomové práce.

10.1 Zamykání

Práce se zámky je nepovinnou součástí protokolu WebDAV [4]. V rámci vývoje rozhraní byl proveden detailní návrh řešení této problematiky, ale k implementaci nakonec nedošlo. Výsledky návrhu jsou zde uvedeny a mohou být použity, pokud v budoucnu vznikne potřeba zamykání implementovat.

10.1.1 Zamykání z pohledu skladu a WebDAVu

Dokumentový sklad poskytuje možnost zamykat soubory výlučným zámkem pro čtení. Znamená to, že zamčený soubor mohou číst všichni uživatelé, kteří k němu mají přístupová práva, ale ostatní operace s ním provádět nemohou. Adresáře zamykat nelze. To je celkem v souladu se specifikací protokolu WebDAV [4]. Jediným problémem je nemožnost zamknout adresář. Nejjednoduším způsobem řešení tohoto nedostatku je předat skladu příkaz k zamčení, sklad zamčení neprovede a oznámí chybu. Pokus o zamčení adresáře tedy nebude mít žádný účinek a klientovi bude vrácen příslušný stavový kód oznamující neúspěch operace. Pokud by někdy zamykání adresářů bylo do skladu implementováno, nebude třeba do WebDAV rozhraní nijak zasahovat.

Každému zámku je ve WebDAVu přiřazen unikátní identifikátor nazývaný *lock-token*. Každý požadavek na práci se zámkem musí tímto identifikátorem označit požadovaný zámek. Na serveru je tedy třeba uchovávat informace o tom, který identifikátor odpovídá kterému zámku na kterém souboru. Běžně by řešením bylo ukládání lock-tokenů do databáze. V našem případě by ale bylo potřeba upravit existující databázové schéma, což není žádoucí. Proto bylo hledáno jiné řešení, které by splnilo požadavky na unikátnost identifikátorů ale zároveň by nevyžadovalo databázi. Asi jedinou možností je generování lock-tokenů z takových údajů, které zaručí jejich neměnnost v době existence zámku, zároveň ale budou poskytovány různé identifikátory pro zámky na různých souborech vytvořené

různými uživateli. Přirozeně se tedy nabízí generovat identifikátory z jména uživatele a cesty k zamykanému souboru, což splňuje všechny zmíněné požadavky.

10.1.2 Metoda `LOCK`

Implementace zamykání vyžaduje vytvoření funkcí obsluhujících požadavky typu `LOCK` a `UNLOCK` a upravení funkcí zpracovávajících metodu `PROPFIND`. Tyto změny budou popsány v následujícím textu podobně jako v kapitole 8.4.

Zamykání souborů se v HTML rozhraní provádí přes formulář, který slouží ke změně vlastností souborů. Tento je již simulován funkcemi obsluhujícími požadavky `PROPPATCH`, čehož lze s výhodou využít – kód není třeba znovu psát, pouze dojde k jeho drobným úpravám. Následující přehled popisuje princip činnosti nových funkcí a úpravy těch stávajících. Navíc bude třeba vytvořit objekt `lock_parser` pro parsování XML těla požadavku `LOCK`.

`get_lock_token` Funkce generující identifikátor zámku. Jejím vstupem bude login uživatele, který zámek vytváří, a jméno zamykaného souboru včetně cesty. Tyto údaje budou spojeny v jeden řetězec, například `cesta/soubor/login`, ze kterého bude vypočítán hash. Ten pak bude naformátován tak, aby odpovídal opaquelocktoken URI, což je podle specifikace [4] doporučený identifikátor zámku.

`do_lock` Musí rozlišit, zda se jedná o požadavek vytvoření zámku nebo jeho obnovení. Při vytvoření zpráva obsahuje XML tělo a neobsahuje hlavičku `If` s identifikátorem zámku, pokud se jedná o obnovu, je tomu naopak. Provede se tedy načtení a kontrola XML těla pomocí objektu `lock_parser` případně získání identifikátoru zámku a kontrola jeho formátu. Poté se zavolá funkce `do_proppatch`.

`do_proppatch` Podle proměnné `$method` objektu `doc_webdav` funkce pozná, že má zpracovávat metodu `LOCK`. Nebude tedy parsovat XML tělo požadavku a provádět kontroly, zda se klient nepokouší nastavit nedovolenou vlastnost. Zajistí pouze načtení informací o zamykaném souboru.

`do_lock2` Může případně kontrolovat, zda se klient nepokouší zamknout adresář, ale není to nutné. Tuto kontrolu není možné provést dříve, protože až nyní jsou k dispozici informace o obsahu skladu. Zavolá funkci `do_proppatch2`.

Pokud jde o obnovení již existujícího zámku, vygeneruje lock-token pro daný soubor a uživatele a porovná jej s klientem dodaným. Při neshodě oznámí klientovi chybu a ukončí běh skriptu.

Jedná-li se o druhé volání funkce v rámci zpracování požadavku, neprovádí nic kromě volání `do_proppatch2`, která nastaví proměnné ovlivňující cyklus volání skladu.

`do_proppatch2` Kontroly nastavovaných vlastností budou opět vynechány. Při nastavování hodnot do polí `$_POST` a `$_REQUEST` bude přidána položka označující požadavek vytvoření zámku. Ve druhé fázi nastaví proměnné cyklu volání skladu.

`do_lock3` Zavolá funkci `lock_send`.

`lock_send` Sestaví XML dokument popisující výsledek operace a odešle ho klientovi. Poté ukončí běh skriptu.

Princip obsluhy metody `LOCK` je podobný principu obsluhy metody `PROPPATCH`. Cyklus volání skladu je proveden celkem dvakrát. V první fázi `do_lock` načte a zkontroluje požadavek, `do_proppatch` způsobí načtení informací o souboru ze skladu včetně vlastníka případného zámku. Funkce `do_lock2` volá `do_proppatch2`, která nasimuluje přijetí formuláře nastavujícího zámek. Nastává druhá fáze, kdy je znovu vytvořen objekt skladu, `do_lock2` nedělá nic a `do_proppatch2` zabrání další iteraci cyklu. Následně dokumentový sklad provede uzamčení. Zbývá informovat klienta o výsledku operace, což provede funkce `lock_send` volaná z `do_lock3`.

10.1.3 Metoda `UNLOCK`

Zpracování metody `UNLOCK` je velmi podobné obsluze `LOCK`, takže není důvod uvádět popis principu a postačí popis funkcí.

`do_unlock` Z hlavičky `Lock-Token` přečte identifikátor zámku a zkontroluje jeho formát. Zavolá funkci `do_proppatch`.

`do_proppatch` Podle proměnné `$method` objektu `doc_webdav` funkce pozná, že obsluhuje metodu `UNLOCK`. Opět nebude parsovat XML tělo požadavku a provádět kontroly vlastností a zajistí pouze načtení informací o souboru.

`do_unlock2` V první fázi ověří oprávněnost požadavku zrušení zámku porovnáním dodaného a vygenerovaného lock-tokenu. Při neshodě oznámí klientovi chybu a ukončí běh skriptu. Ve druhé fázi pouze volá `do_proppatch2`.

`do_proppatch2` V první fázi vynechá kontroly vlastností a do polí `$_POST` a `$_REQUEST` nepřidá položku pro zámek, čímž simuluje nezaškrtnutí příslušného políčka formuláře a požaduje tak zrušení zámku. Ve druhé fázi zabrání další iteraci cyklu volání skladu.

`do_unlock3` Informuje klienta o výsledku operace příslušným stavovým kódem.

10.1.4 Metoda `PROPFIND`

Je nutné také doplnit odpověď na požadavek `PROPFIND` o informace o existujících zámcích a zavést podporu zpracování XML elementů, kterými klient požaduje zaslání těchto informací. Konkrétně je třeba provést následující změny:

- V objektu `propfind_parser` je nutné upravit funkci `start_tag`, aby rozeznávala tagy `lockdiscovery` a `supportedlock` v elementu `prop` a nastavovala proměnné indikující jejich přítomnost. Dále je vhodné upravit konstruktor v místě, kde se kontrolují příznaky přítomnosti jednotlivých elementů, aby kontrola zahrnovala i tyto nové elementy.
- Funkce `propfind_send` objektu `doc_repos` musí být upravena tak, aby:
  - při požadavku na zjištění všech vlastností zdroje, což je indikováno XML elementem `allprop` a proměnnou objektu `propfind_parser` stejného jména, vracela pro všechny zdroje v elementu `prop` element `supportedlock` s obsahem informujícím o možnosti vytvoření výlučného zámku pro zápis;
  - při požadavku na zjištění podporovaných zámků indikovaným XML elementem a proměnnou `supportedlock` vracela v `prop` pouze `supportedlock` se stejným obsahem;

– při požadavku na zjištění existujících zámků indikovaným `lockdiscovery` vracela v `prop` element `lockdiscovery` obsahující v elementu `activelock` informace o existujícím zámku včetně jeho rozsahu, typu a identifikátoru.

Přesný popis zmiňovaných XML elementů a další potřebné informace lze nalézt ve specifikaci WebDAV protokolu [4].

10.2 Další neimplementované funkce

Následující seznam shrnuje ostatní funkce, které nebyly v rozhraní implementovány. Jedná se o funkce, které klienti většinou nepoužívají, takže jejich nepřítomnost neovlivňuje celkovou funkčnost. Většina jich není implementována například ani v serveru studovaném v kapitole 5. Výčet se zaměřuje pouze na WebDAV rozhraní a neobsahuje položky, jejichž realizace vyžaduje zásah do samotného dokumentového skladu.

- Parsování, kontrola a zpracování `If` hlaviček. Vhodným místem pro implementaci je funkce `do_request`.
- Zpracování těla požadavku `MKCOL`.
- Zpracování hlaviček `Content-Encoding` (umožňuje například přenos dat komprimovaných algoritmem gzip) a `Content-MD5` (slouží pro kontrolu integrity přeneseného souboru pomocí hashe) v obsluze požadavku `PUT`.
- Zavedení možnosti tzv. multipart uploadu v metodě `PUT`, tedy nahrání více souborů jedním požadavkem.
- Rozšíření metody `PROPPATCH` tak, aby umožňovala nastavovat všechny vlastnosti souborů a adresářů včetně přístupových práv.

Kapitola 11

Závěr

11.1 Dosažené výsledky

Cílem této diplomové práce bylo vytvoření WebDAV rozhraní dokumentového skladu IS FIT. Jeho prostřednictvím se umožní přístup ke skladu přímo z prostředí operačního systému například použitím klienta integrovaného v MS Windows nebo v některých desktopových prostředích unixových operačních systémů. Vytvořené rozhraní poskytuje veškerou funkčnost potřebnou k jeho použití s těmito klienty.

Díky důkladnému detailnímu návrhu byla následná implementace provedena bez větších problémů. Ačkoliv bylo nezbytné zasahovat do původního kódu dokumentového skladu, nejedná se o zásahy velkého rozsahu, které by významně měnily funkci skladu. V rámci řešení této práce byl pro použití s WebDAV rozhraním upraven pouze jeden podtyp skladu, avšak úprava zbývajících podtypů nebude představovat nepřijatelně velký rozsah prací.

Lze tedy říci, že byly splněny všechny požadavky kladené na výsledný produkt. Přesto je možné pokračovat v jeho dalším rozvoji doplněním dosud neimplementovaných méně podstatných funkcí.

11.2 Přínos práce

Během řešení této práce jsem získal množství nových zkušeností a znalostí. Podrobně jsem se seznámil s principy fungování protokolu HTTP a detailně pak s jeho rozšířením WebDAV. Studováním zdrojových kódů dokumentového skladu a WebDAV serveru jsem získal nové náhledy na možnosti řešení problémů v oblasti objektově orientovaného programování internetových aplikací.

Literatura

[1] ACHOUR, M.; aj.: *PHP Manual*. The PHP Documentation Group, © 1997–2007 The PHP Documentation Group, [online], rev. 9 December 2007, [cit. 2008-04-12], anglicky.
URL http://www.php.net/manual/en/

[2] BRAY, T.; PAOLI, J.; aj.: *Extensible Markup Language (XML) 1.0 (Fourth Edition) W3C Recommendation*. [online], rev. 16. srpna 2006.
URL http://www.w3.org/TR/2006/REC-xml-20060816/

[3] BRAY, T.; PAOLI, J.; aj.: *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*. [online], rev. 16. srpna 2006.
URL http://tools.ietf.org/html/rfc2616

[4] GOLAND, Y.; WHITEHEAD, E.; aj.: *RFC 2518: HTTP Extensions for Distributed Authoring – WEBDAV*. [online], rev. únor 1999.
URL http://www.webdav.org/specs/rfc2518.html

[5] LAMPA, P.: Dokumentový sklad. [online], slidy k prezentaci.
URL http://www.fit.vutbr.cz/research/pubs/TR/2007/sem_uifs/s070416slidy1.pdf

[6] LAMPA, P.: Dokumentový sklad. [online], zvukový záznam prezentace.
URL http://www.fit.vutbr.cz/research/pubs/TR/2007/sem_uifs/s070416zaznam1.mp3

[7] Hypertext Transfer Protocol. [online], internetová encyklopedie Wikipedia.
URL http://en.wikipedia.org/wiki/HTTP

[8] MySQL. [online], internetová encyklopedie Wikipedia.
URL http://en.wikipedia.org/wiki/MySQL

[9] PHP. [online], internetová encyklopedie Wikipedia.
URL http://en.wikipedia.org/wiki/PHP

[10] SQL. [online], internetová encyklopedie Wikipedia.
URL http://en.wikipedia.org/wiki/SQL

[11] XML. [online], internetová encyklopedie Wikipedia.
URL http://en.wikipedia.org/wiki/XML