VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

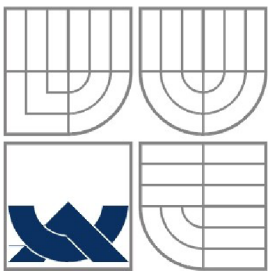
WEBOVÁ APLIKACE PRO VYHLEDÁVÁNÍ V ON-LINE KATALOZÍCH KNIHOVEN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

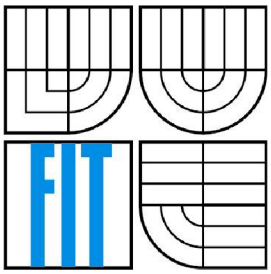
AUTOR PRÁCE
AUTHOR

ONDŘEJ POLÁCH

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ APLIKACE PRO VYHLEDÁVÁNÍ V ON-LINE KATALOZÍCH KNIHOVEN

A WEB-BASED APPLICATION TO SEARCH ONLINE LIBRARY CATALOGUES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ POLÁCH

VEDOUCÍ PRÁCE
SUPERVISOR

Mgr. MAREK RYCHLÝ, Ph.D.

BRNO 2010

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2009/2010

Zadání bakalářské práce

Řešitel: **Polách Ondřej**
Obor: Informační technologie
Téma: **Webová aplikace pro vyhledání v on-line katalozích knihovnen**
A Web-Based Application to Search Online Library Catalogues
Kategorie: Web

Pokyny:

1. Seznamte se a porovnejte různé formáty elektronických katalogů knihoven a protokoly pro on-line přístup k těmto katalogům (Z39.50, SRU/SRW, atp.).
2. Navrhněte webovou aplikaci pro vyhledání v on-line katalozích knihoven přes uvedené protokoly. Kromě zadávání dotazů a interpretace odpovědí by měla aplikace umožnit systémovou i uživatelskou konfiguraci zdrojů (knihoven) a uživatelské nastavení automaticky opakovaných dotazů s možností odeslání výsledků na email.
3. Implementujte navrženou aplikaci. Snažte se využít dočasné úložiště "cache" pro optimalizaci procesu vyhodnocení dotazů.
4. Výsledek otestujte, zhodnoťte a navrhněte další rozšíření projektu.

Literatura:

- Z39.50 Implementors Group ZIG-CZ. *Z39.50 zdroje tuzemské*. [http://old.stk.cz/ZIG/zdroje_cz.html]
- *Information Retrieval (Z39.50): Application Service Definition and Protocol Specification*. ANSI/NISO Z39.50-2003. [http://www.loc.gov/z3950/agency/Z39-50-2003.pdf]
- *SRU: Search/Retrieval via URL -- SRU, CQL and ZeeRex (Standards, Library of Congress)*. [http://www.loc.gov/standards/sru/]
- OMG. *Unified Modeling Language (UML)*. 2007. [http://www.omg.org/technology/documents/formal/uml.htm]

Při obhajobě semestrální části projektu je požadováno:

- Bod 1 a částečně bod 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Rychlý Marek, Mgr.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2009
Datum odevzdání: 19. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Internet je od dob svého počátku zdrojem nejrůznějších informací. Jejich vyhledávání bylo standardizováno s příchodem protokolu Z39.50, který se v dnešní době stále vyvíjí a slouží také jako základní kámen pro moderní protokoly informačního vyhledávání. Skupina protokolů SRU/SRW je příkladem nových protokolů, které jsou orientované na servisně orientovanou architekturu. Cílem této bakalářské práce je navrhnout a implementovat jednotnou webovou informační bránu sloužící pro vyhledávání v různých katalozích knihoven z jediného místa. Vývojová platforma bude .NET a Mono. Největší výzvou je použití WCF služeb spolu s klientskou částí systému postavenou na technologii Silverlight a v neposlední řadě je to řešení problémů vyplývajících z dosavadní neúplné kompatibility technologie Mono a .NET, hlavně co se týče dvou zmíněných komponent. Obě budou hostovány webovou aplikací ASP.NET. Hlavním požadavkem je přenositelnost řešení na různé platformy. Pokud se podaří dosáhnout souladu mezi Mono a .NET, bude možno systém provozovat na všech platformách, které jsou podporovány technologií Mono.

Abstract

The Internet is since its inception, source of various informations. Their retrieve was standardized with the advent of Z39.50 protocol, which today is still evolving and it also serves as the cornerstone for modern retrieve information protocols. Protocols group SRU/SRW is example of new protocols which are oriented into service-oriented architecture. The aim of this bachelor's thesis is to design and implement a unified web-based information gateway for retrieve informations from various library catalogues from single location. Development platform will be .NET and Mono. The biggest challenge is to use WCF services with system's client-side build on Silverlight technology, and last, but not least, is the solve problems arising from the current incomplete compatibility between Mono and .NET, especially with regard to these two components. Both will be host in ASP.NET web application. The main requirement is portability solution on various platforms. If manage to achieve compatibility between Mono and .NET, application can run on all platforms that are supported by Mono technology.

Klíčová slova

Získávání informací, Z39.50, Microsoft .NET, Mono, Silverlight, Moonlight, WCF služby

Keywords

Information retrieve, Z39.50, Microsoft .NET, Mono, Silverlight, Moonlight, WCF services

Citace

Ondřej Polách: Webová aplikace pro vyhledávání v on-line katalozích knihoven, bakalářská práce, Brno, FIT VUT v Brně, 2010.

Webová aplikace pro vyhledávání v on-line katalozích knihoven

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Mgr. Marka Rychlého, Ph.D.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Ondřej Polách
24. dubna 2010

Poděkování

Rád bych mnohokrát poděkoval své rodině za jejich velkou podporu a toleranci.

Také velmi děkuji mému vedoucímu práce, panu Mgr. Marku Rychlému, Ph. D., za jeho užitečné rady a shovívavý přístup k mému řešení.

Obsah

1 Úvod

Tato práce se zabývá získáváním informací z katalogů knihoven. Výsledný produkt bude webová brána, která zastřešuje vyhledávání v různých knihovnách. Celý systém ponese název WASOLIC, což je zkratka anglického názvu bakalářské práce. Řešení je postaveno na platformě .NET a jí přidružené platformě Mono pro přenositelnost. Základními pilíři jsou WCF (Windows communication foundation) služby a technologie Silverlight. Obě jsou postaveny na platformě .NET. WCF služby mají na platformě Mono zatím implementovánu jen základní část a Silverlight je zastoupen technologií Moonlight. Jak WCF služby, tak i Silverlight aplikace budou hostovány na webové aplikaci ASP.NET, která je v Mono podporována téměř bez výhrad. O samotné získávání informací z katalogů knihoven se bude starat Z-klient, který bude postaven na balíčku nástrojů YAZ a ZOOM.NET.

V první části této práce se budu obecně zabývat získáváním informací z informačních systémů. Poté srovnám platformy .NET a Mono z obecného i mého konkrétního hlediska. Zbývající části práce odrážejí vývojový cyklus aplikace. Od analýzy požadavků, přes návrh a implementaci, až po možné nasazení aplikace.

2 Získávání dat z informačních systémů

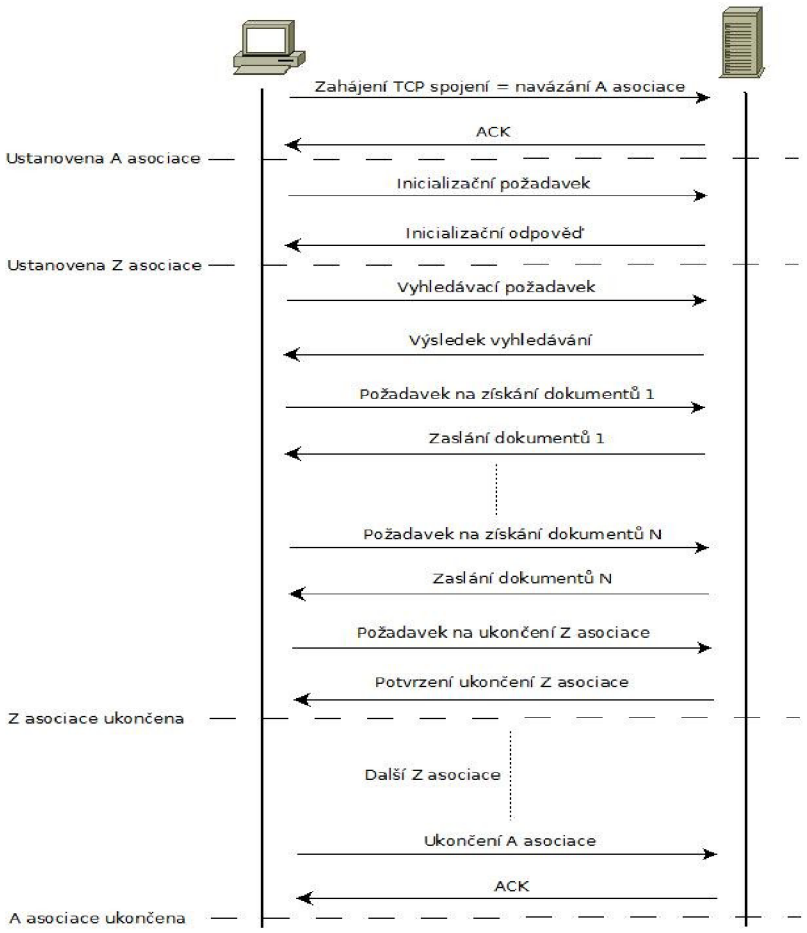
2.1 Historie a budoucí směr

Vznik protokolu Z39.50 se datuje na rok 1970, kdy se tři významné organizace pokusily o zavedení standardizovaných prostředků pro vyhledávání informací v jejich bázích dat. Těmito organizacemi byli *Library of Congress*, *Online Computer Library Center* a *Research Libraries Information Network* působící ve Spojených Státech Amerických. Záměrem bylo vytvořit sdílený katalog využívající národní bibliografickou databázi. Od počátku se účastníci na projektu podíleli jak na specifikaci protokolu, tak i na jeho implementaci. S postupem času se ale zaměřili pouze na implementaci a samotná specifikace protokolu byla přesunuta pod záštitu organizace *National information Standards Organization*. Tato zveřejnila první standard v roce 1988, který nesl název „*American National Standard Z39.50, Informational Retrieval Service Definition and Protocol Specification for Library Applications*“. Ke konci osmdesátých let se objevila množina jiných standardů. Mezi nimi i protokol s označením „*Search and Retrieve*“ (SR), který byl téměř identický jako Z39.50 a byl standardizován pod označením ISO 10162/10163. Vznikla potřeba vytvořit novou verzi protokolu Z39.50 kompatibilní s uvedeným ISO standardem. Pod vedením organizace *Z39.50 Maintenance Agency* a skupinou implementátorů *Z39.50 Implementators Group* byla v roce 1992 vydána nová verze protokolu označená *Z39.50-1992* nebo též jako verze 2. Bylo vytvořeno množství systémů klient-server komunikujících mezi sebou pomocí protokolu TCP/IP a protokol Z39.50 si tak udělal své místo v knihovnické obci. Vývoj protokolu stále pokračoval a v roce 1995 byla vydána další, již třetí verze protokolu, označená *Z39.50-1995*, která byla roku 1997 uznána jako ISO standard s označením ISO 23950 [1].

V současné době se pracuje na dalším vývoji protokolu Z39.50, hlavně za účelem propojení s dalšími standardy a technologiemi. Protokol SRU (Search and Retrieve via URL) používá REST-ful služby, které mají komunikaci založenou na URL. Protokol SRW (Search and Retrieve via Web) je postaven na SOAP-ful službách, které využívají XML zprávy protokolu SOAP. Celá tato skupina protokolů je tedy zaměřená na servisně orientovanou architekturu (SOA) [2].

2.2 Protokol Z39.50

Z39.50 řídí komunikaci mezi původcem (klientem) a cílem (serverem) při vyhledávání a získávání informací z bází dat. Spojení je navazováno původcem a ukončit jej může jak původce, tak cíl. Komunikace je založena na mechanismu požadavek-odpověď a je znázorněna na obrázku Obr. 1.



Obr. 1: Komunikace protokolu Z39.50.

Původce a cíl spolu komunikují skrze Z39.50 asociaci (Z-asociace) jako součást aplikační asociace (A-asociace). V rámci jedné A-asociace může probíhat několik posloupných Z-asociací. V rámci jedné Z-asociace může probíhat více po sobě jdoucích nebo paralelních operací. Samotné vyhledávání je zahájeno vyhledávacím požadavkem od původce, na který cíl reaguje spuštěním vyhledávací operace a zasláním výsledků vyhledávání zpět původci, které mimo jiné obsahují počet nalezených záznamů. Poté, co původce přijme výsledek vyhledávání, smí požádat cíl o dané záznamy. Tyto záznamy smí žádat postupně v rámci několika požadavků-odpovědí (1 - N). Zprávy protokolu, nebo-li datové jednotky, jsou kódovány pomocí BER (Basic Encoding Rules), které používají formát TLV (Type-Lenght-Value), což znamená posloupnost dat takovou, že první je typ následovaný délkou a poté samotnou hodnotou [3].

2.2.1 Profily protokolu

Protokol Z39.50 je velmi obsáhlý a jeho kompletní podpora v implementaci je obtížná. Proto vznikly různé profily, které definují podmnožinu funkcí původního protokolu. V České republice je nejvíce

používán profil Bath2-2. Existují i další, jako například ATS-1, ONE-2 nebo Z Texas. Bath2-2 profil rozděluje požadavky na funkčnost do 4 skupin (A – D). Každá z těchto skupin má 2 až 3 úrovně shody, které se liší požadovanými atributy dotazu, formátem záznamů atd. [4].

A) Pro základní bibliografické vyhledávání a získávání záznamů s primárním důrazem na knihovní systémy.
B) Pro bibliografické vyhledávání exemplářů a získávání záznamů.
C) Pro mezioborové vyhledávání a získávání záznamů.
D) Pro vyhledávání a získávání autoritativních záznamů v on-line knihovních katalozích.

2.2.2 Služby protokolu

Protokol Z39.50 obsahuje několik služeb. Profil vymezuje hlavně služby pro inicializaci, vyhledávání, získávání a skenování. Protokol ale obsahuje i služby pro odstraňování záznamů, řazení, rozšířené služby ad. Každá služba je součástí výměny zpráv mezi původcem a cílem a má na cíli odpovídající operaci. Následuje krátký popis služeb vymezených profilem [3].

- **Init**
  Požadavek na zahájení Z-asociace.
- **Search**
  Požadavek na vyhledávání. Specifikuje se dotaz, formát záznamů, databáze ad.
  Dotaz i formát mohou být různého typu.
- **Present**
  Požadavek na získání nalezených záznamů. Záznamy jsou segmentovány dle
  1. (segmentačního) nebo 2. (fragmentačního) stupně. Cíl si v rámci Z-asociace uchovává množinu výsledků, kterou je připraven přenést na požádání původci. Ten si ji může vyžádat pomocí jednoho nebo více požadavků.
- **Scan**
  Slouží k prohledávání rejstříků.

2.2.3 Formáty záznamů

Záznamy mohou mít různé formáty. Hlavně se jedná o formáty rodiny MARC (Machine-readable cataloging) jako MARC21 nebo UNIMARC, dále SUTRS (Simple unstructured record syntax), XML a další [3].

2.2.4 Dotazy

Protokol Z39.50 nejčastěji používá dotaz ve formě obrácené polské notace – RPN. Dotaz používá atributivní sadu Bib-1. Tato atributivní sada obsahuje 6 typů atributů, kde každý má několik užití. Celý výčet kombinací se nachází v [3].

2.3 Protokoly SRU/SRW

SRU/SRW jsou protokoly založené na webových službách. Slouží pro dotazování a vyhledávání v databázích a indexech na internetu a pro získávání výsledků [2].

Protokol SRU obvykle kóduje požadavek do adresy URL. Jedná se tedy o REST-ful službu. Každá dvojice jméno/hodnota dotazového řetězce specifikuje vstupní parametry pro server, který je zpracuje a výsledek vrací jako proud dat XML. Hlavní výhodou SRU je to, že je založen na protokolu HTTP a může tak používat metody POST, GET atd. [2].

Protokol SRW pracuje obdobně, ale dvojice jméno/hodnota není kódovaná do URL, ale je vložena do obálky XML dokumentu protokolu SOAP. Jedná se o SOAP-ful službu. Server vrací výsledek vyhledávání ve formě SOAP zprávy ve formátu XML. Oproti SRU, nemusí SRW používat jako transportní protokol pouze HTTP, ale může použít i jiné [2].

REST-ful služby jsou v porovnání se SOAP-ful službami jednodušší na implementaci, naproti tomu SOAP-ful může poskytovat větší robustnost.

2.3.1 Služby protokolu

Protokoly SRU/SRW podporují 3 základní operace:

- **Explain**
  Získání informací o cíli. Výsledek je navrácen ve formě XML proudu.
- **Search/Retrieve**
  Odeslání požadavku na cíl, který jej zpracuje a vrací výsledek vyhledávání ve formě XML proudu.
- **Scan**
  Vyhledávání v rejstřících indexů a databází [2].

2.3.2 Dotazy

Protokoly SRU/SRW používají dotazovací jazyk CQL (Contextual Query Language - verze 1.2). Jedná se o formální jazyk pro dotazování na informace v systémech jako bibliografické katalogy, webové indexy a databáze. Syntaxe a BNF se nacházejí v [5].

2.4 Z-brány

Z uživatelského hlediska, pokud chce uživatel vyhledávat pomocí protokolu Z39.50 informace v bázích dat, má na výběr ze dvou typů klientů. Buď klasický pevný klient, který se instaluje na klientský počítač nebo klient dostupný přes webové rozhraní. Druhá možnost používá protokol HTTP a uživatel pro vyhledávání používá webový prohlížeč, který komunikuje s tzv. Z-bránou. Tato brána je vlastně Z-klient, který zpracovává požadavky uživatelů přicházející přes webové rozhraní protokolu HTTP.

Systém WASOLIC bude implementován jako Z-brána, která ale nebude postavena na HTTP protokolu, ale na SOAP (WCF).

3 Technologie Microsoft .NET a Mono

Technologie Microsoft .NET je softwarová platforma pro vývoj aplikací v prostředí operačních systémů firmy Microsoft. Základním pilířem je .NET Framework. Tato technologie není přímo přenositelná na jiné platformy, proto vznikla skupina, která má za úkol vytvářet přenositelnou softwarovou platformu, která implementuje .NET Framework. Tato platforma, založená na otevřeném kódu, nese název Mono. Implementace je založena na standardech pro jazyk C# a CLR (Common Language Runtime). Hlavními komponentami platformy Mono jsou překladač jazyka C# a běhové prostředí, které implementuje CLI (Common Language Infrastructure) a poskytuje překladač JIT (Just-In-Time), správce paměti, systém podpory vláken ad. Mono je přenositelné na většinu dnešních platforem od Linuxu přes BSD a Windows, až po Mac OS X a Sun Solaris. Ke dni 27.04.2010 existovala vydaná verze 2.6.4, která je z velké části kompatibilní s verzí .NET Frameworku 3.5 [6].

Řešení mé bakalářské práce je postaveno na technologiích ADO.NET, ASP.NET, Silverlight, WCF a jím odpovídajícím protějškům na platformě Mono.

ADO.NET se stará o přístup k datovým zdrojům. ASP.NET je platforma pro vývoj webových aplikací založených na webových formulářích. Obě tyto komponenty jsou v aktuální verzi platformy Mono v režimu maximální kompatibility. Technologie Silverlight je na platformě Mono implementována projektem Moonlight. Aktuální stabilní verze Moonlight 2.0 je kompatibilní z velké části s verzí 2.0 a z části také s verzí 3.0 technologie Silverlight. WCF služby jsou v aktuální stabilní verzi podporovány jen v základní podobě a nejsou úplně doladěny [6, 7].

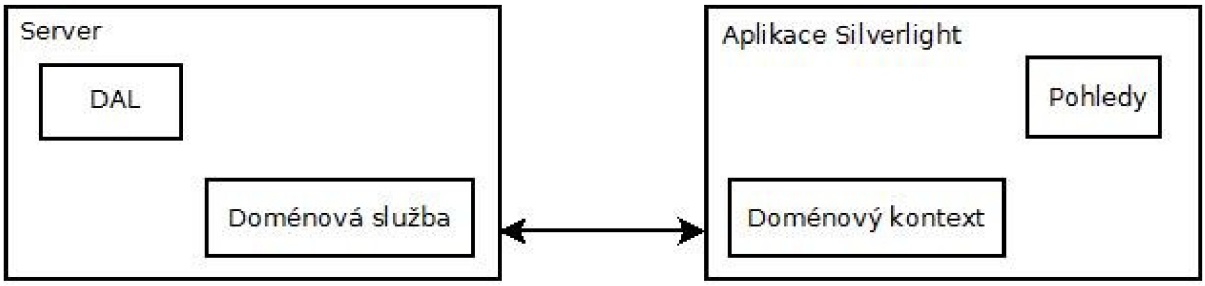
3.1 Porovnání ASP.NET a Silverlight

V této podkapitole se zaměřím na srovnání technologií ASP.NET a Silverlight při vývoji webových aplikací..

Při tvorbě webových aplikací pomocí ASP.NET se vychází ze znalosti komunikace klient-server nebo požadavek-odpověď. Webové aplikace jsou tvořeny formuláři, které obsahují jednoduchý obsah. Od textových popisků, přes textová pole, až po tabulky. Princip komunikace je takový, že server generuje obsah formuláře na základě požadavku klienta, kterým je webový prohlížeč. Formulář je generován ve značkovacím jazyce HTML. Takový formulář je klientovi poslán pomocí protokolu HTTP. Webový prohlížeč poté provede zobrazení HTML dokumentu. Ve výchozím stavu je při každém požadavku generován celý formulář znovu. Toto může být někdy na obtíž. Pro odstranění tohoto „problému“ se mohou použít klientské skripty spolu s klientským voláním. Takto je mezi klientem a serverem vytvořen kanál, kterým spolu komunikují na pozadí. Hlavním průkopníkem tohoto přístupu je skriptovací jazyk Javascript.

ASP.NET stránky jsou plně objektově orientované. Protože je ASP.NET postaveno na .NET Frameworku, může využívat jím poskytnuté rozhraní, metody, třídy aj. Lze tak například použít ADO.NET pro přístup k datovému zdroji apod. Webová aplikace je centrálně konfigurovatelná přes konfigurační soubor a běží na webovém serveru.

S použitím technologie Silverlight je přístup k vytváření webových aplikací poměrně jiný. Vychází to ze skutečnosti, že aplikace Silverlight neběží na serveru samotném, jako je tomu u aplikace ASP.NET. Webový server aplikaci Silverlight pouze hostuje a to s použitím ASP.NET stránky. Aplikace Silverlight běží na klientské straně, tzn. spotřebovává zdroje klientského počítače. Při prvním spuštění aplikace se tato stáhne na klientský počítač a od této doby běží v prostředí webového prohlížeče. Aby mohla aplikace Silverlight běžet v prohlížeči, je nutné doinstalovat Silverlight Plug-In. V prvním vydání Silverlight 1.0 se jednalo o technologii podporující web bohatý na grafiku a multimédia, s postupem času a příchodem WCF RIA (Rich Intenet Application) služeb se ale stále více hodí i pro obchodní aplikace, které například potřebují přístup k datových zdrojům apod. Stavebním kamenem je značkovací jazyk XAML (Extensible Application Markup Language), který podporuje deklarativní programování uživatelského prostředí. Poprvé byl použit na platformě WPF (Windows Presentation Foundation) a lze s ním vytvářet bohaté uživatelské prostředí webové aplikace. Jelikož aplikace Silverlight běží na klientovi, je obtížné představit si, jak získává například data z databázového zdroje, který se nachází v lepším případě na hostujícím serveru. Zde přicházejí na scénu WCF služby. Právě díky nim můžeme na hostovaném serveru zpřístupnit službu, která může například komunikovat s databází a výsledky vracet aplikaci Silverlight. Pro zjednodušení práce byli vyvinuty zmíněné WCF RIA služby, které za použití doménové služby a doménového kontextu dávají možnost aplikaci Silverlight používat datové zdroje na serveru, autentizaci a autorizaci, případně jiné služby potřebné pro naplnění logiky aplikace. Obr. 2 znázorňuje schéma WCF RIA služeb. Zkratka DAL označuje vrstvu pro přístup k datům. Pohledy znázorňují uživatelské prostředí. Jak z názvu vyplývá, WCF RIA služby komunikují pomocí WCF služeb. Doménová služba je ve skutečnosti WCF služba a doménový kontext je konzument této služby.



Obr. 2: Schéma WCF RIA služeb.

Bohužel WCF RIA služby nejsou na platformě Mono implementovány. Je proto třeba vytvořit zjednodušenou obdobu doménové služby a kontextu. Tím pádem se musí obejít neúplná kompatibilita WCF služeb na Mono, což je velká výzva.

4 Analýza

Cílem práce je vytvořit webovou aplikaci, která má sloužit pro vyhledávání informací v knihovnách. Základním stavebním kamenem aplikace je protokol pro vyhledávání a získávání záznamů v bázích dat. Jelikož je toto téma známé již řadu let, je velmi pravděpodobné, že již existuje množina fungujících řešení. V podkapitole 4.1 popíši ty z nich, které ve své práci použiji. Dále se v této kapitole zaměřím na analýzu požadavků a stručně popíši vývojové prostředí projektu.

4.1 Hledání existujících řešení

I když se má jednat o webovou aplikaci, bude potřeba implementovat samostatný modul, který bude řídit vyhledávání a získávání dat pomocí protokolu Z39.50 nebo skupiny protokolů SRU/SRW. Existuje několik softwarových implementací, které tyto protokoly implementují. Já jsem si ve své práci vybral balíček nástrojů YAZ firmy *Index Data Aps* [8].

4.1.1 Balíček nástrojů YAZ a ZOOM.NET

YAZ je volný programovací balík podporující vývoj Z39.50/SRU/SRW klientů a serverů. Protokol Z39.50 je podporován ve verzi Z39.50-2003 (verze 3) a SRU/SRW ve verzi 1.1, a to jak na klientovi, tak i na serveru. Já ve svém řešení použiji pouze klientskou část.

YAZ je napsán v jazyce C. Existují i vazby na jiné jazyky jako je C++, PHP aj. Pro usnadnění vývoje klienta vystavuje aplikační rozhraní nazvané ZOOM. Jedná se o soubor funkcí, které dovolují jednoduchou implementaci klienta [8].

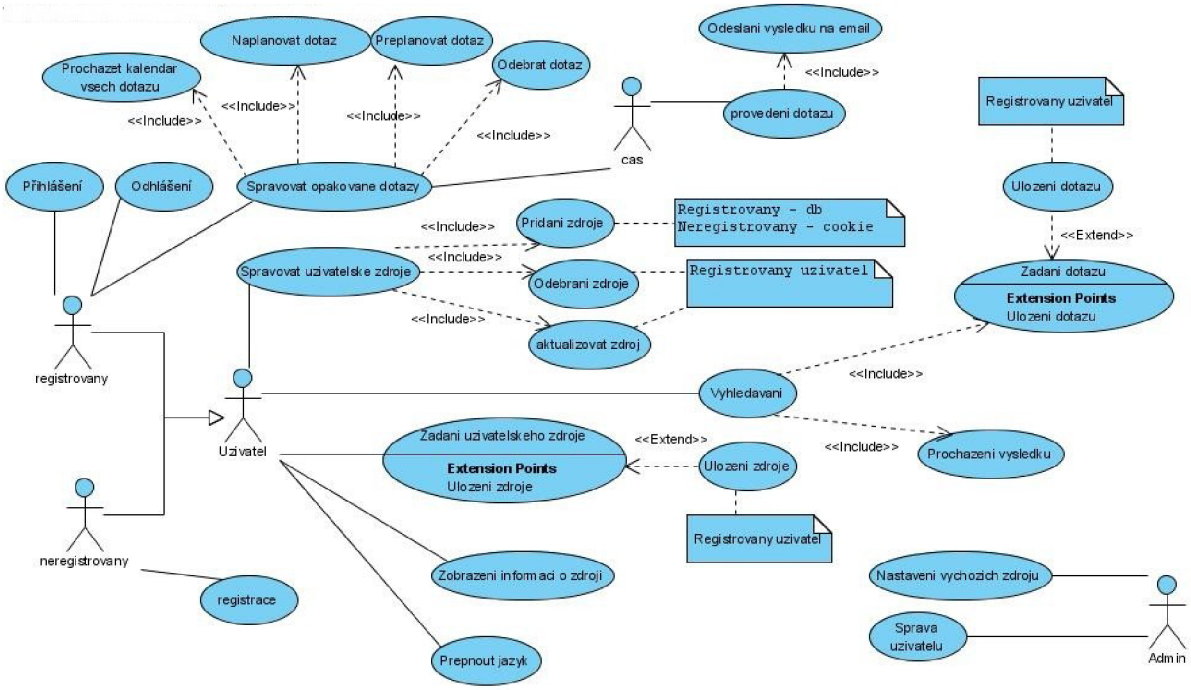
Pro použití YAZ v prostředí .NET je nutná portace na jazyky jako C# nebo Visual Basic. Firma *Index Data Aps* bohužel neposkytuje žádnou portaci na jazyk C#. Existuje ale otevřené řešení nazvané ZOOM.NET, které bylo napsáno v roce 2006 a aktuální verze je 1.0.2400. Od té doby se s řešením bohužel nic nedělalo, ale pro mé potřeby je dostačující. Řešení se nachází na webovém portálu *Source Forge* a autory jsou pánové *Rob Styles, Eric C. Willman* a *Marc Cromme* [9]. ZOOM.NET využívá interoperabilní aplikační rozhraní .NET Frameworku pro volání funkcí původní YAZ knihovny napsané v jazyce C. Deklaruje základní třídy a rozhraní pro implementování jednoduchého klienta Z39.50/SRU/SRW protokolu.

Pro dosažení přenositelnosti kódu bude nutný menší zásah do zdrojových kódů ZOOM.NET knihovny. Bližší popis se nachází v kapitole 6.

4.2 Požadavky na systém

Na systém je kladeno několik kritérií, které jsou rozděleny do kategorií uživatelských a systémových. Analýza požadavků byla provedena z diagramu případu užití.

4.2.1 Diagram případů užití



Obr.3: Diagram případů užití.

4.2.2 Uživatelské požadavky

Z pohledu uživatele jsou definovány následující požadavky:

- Uživatelé mohou bez přihlášení vyhledávat a získávat informace z výchozích zdrojů. Vyhledávání musí být rozděleno na základní a na pokročilé. Obě se liší počtem možných kontextů a počtem souběžných vyhledávání. Základní vyhledávání bude mít možnost vyhledávat pouze v jednom zdroji v daný čas.
- Po přihlášení mají možnost konfigurovat si vlastní zdroje, ukládat a plánovat opakované dotazy. Výsledky opakovaných dotazů jsou jim odeslány na emailovou adresu.
- Výchozí zdroje spravuje administrátor systému.
- Uživatelské prostředí by mělo být jednoduché a přístupné přes webový prohlížeč.
- Uživatelé musí mít jednoduchý a bezpečný přístup ke svým zdrojům a dotazům.
- Mělo by být implementováno administrátorské rozhraní alespoň pro správu zdrojů.
- Vyhledávání a získávání informací musí být co nejefektivněji řešeno.

4.2.3 Systémové požadavky

Z pohledu systému jsou definovány následující požadavky:

- Systém musí obsahovat správu uživatelských účtů.
- Většina dat systému bude uložena v relační databázi. Jedná se o uživatelské účty, dotazy, zdroje a úložiště dočasné paměti.
- Pro definici a přístup k databázi musí být použit standard SQL (Structured Query Language) pro možnost přenesení databáze mezi různými databázovými stroji.
- Je nutná přenositelnost systému mezi různými platformami operačních systémů.
- Uživatelská data musí být chráněna před zneužitím.
- Musí být použita dočasná paměť pro zefektivnění vyhledávání.
- Je nutné úložiště plánovače dotazů ve formě dokumentu XML.

4.2.4 Vývojové prostředí

Tato podkapitola obsahuje stručný popis vývojového prostředí použitého při vytváření systému WASOLIC.

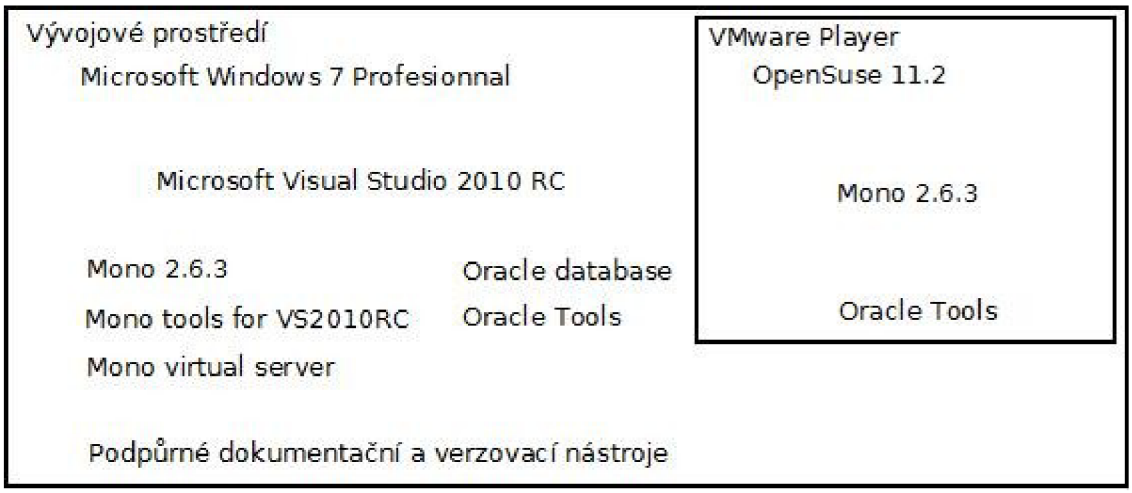
Hlavním vývojovým nástrojem bylo použito Microsoft Visual Studio 2010 RC, které běželo na operačním systému Microsoft Windows 7. Pro dodržení maximální kompatibility s platformou Mono byl použit .NET Framework ve verzi 3.5.

V první etapě vývoje bylo použito Mono ve verzi 2.6.3. Bohužel v této verzi se objevili potíže s WCF službami způsobené nejspíše nedostatkem v implementaci těchto služeb ve verzi 2.6.3. Pokoušel jsem se nainstalovat SVN verzi Mono, kde by možná mohli být problémy odstraněny. Kompilace Mono ovšem i po několika pokusech selhala a proto jsem zůstal u verze 2.6.3, kterou jsem později nahradil nově vydanou verzí 2.6.4. Ta ovšem také nenapravila problémy s WCF službami.

V první etapě vývoje datové vrstvy jsem používal Oracle server na vývojovém stroji. Později jsem kvůli výkonnostním problémům přešel na server Berta, který se nachází v síti Fakulty informačních technologií v Brně. Pro ladění databáze jsem využíval nástroj SQL Developer od firmy Oracle.

Pro ladění aplikace jsem díky povaze projektu zvolil dvě prostředí. Samotné Microsoft Windows a virtuální stroj s linuxovou distribucí OpenSuse 11.2, kde byli nainstalované nástroje pro ladění aplikací Mono. Pro přístup k těmto nástrojům přímo z Visual Studia 2010 byli použity Mono tools pro Visual Studio, které dovolily ladit aplikaci Mono přímo z prostředí studia, i když aplikace běžela na virtuálním operačním systému OpenSuse. Jako host virtuálního stroje byl použit Vmware Player.

Celý projekt byl řízen pomocí prostředků dostupných na webovém portálu Assembla. Byli využity hlavně verzovací nástroje a poznámkové bloky ve formě wiki nebo tikety. Adresa umístění projektu se nachází v [10]. Obrázek Obr. 4 zobrazuje stav vývojového prostředí v první etapě vývoje projektu WASOLIC.



Obr. 4: Stav vývojového prostředí v první etapě.

5 Návrh

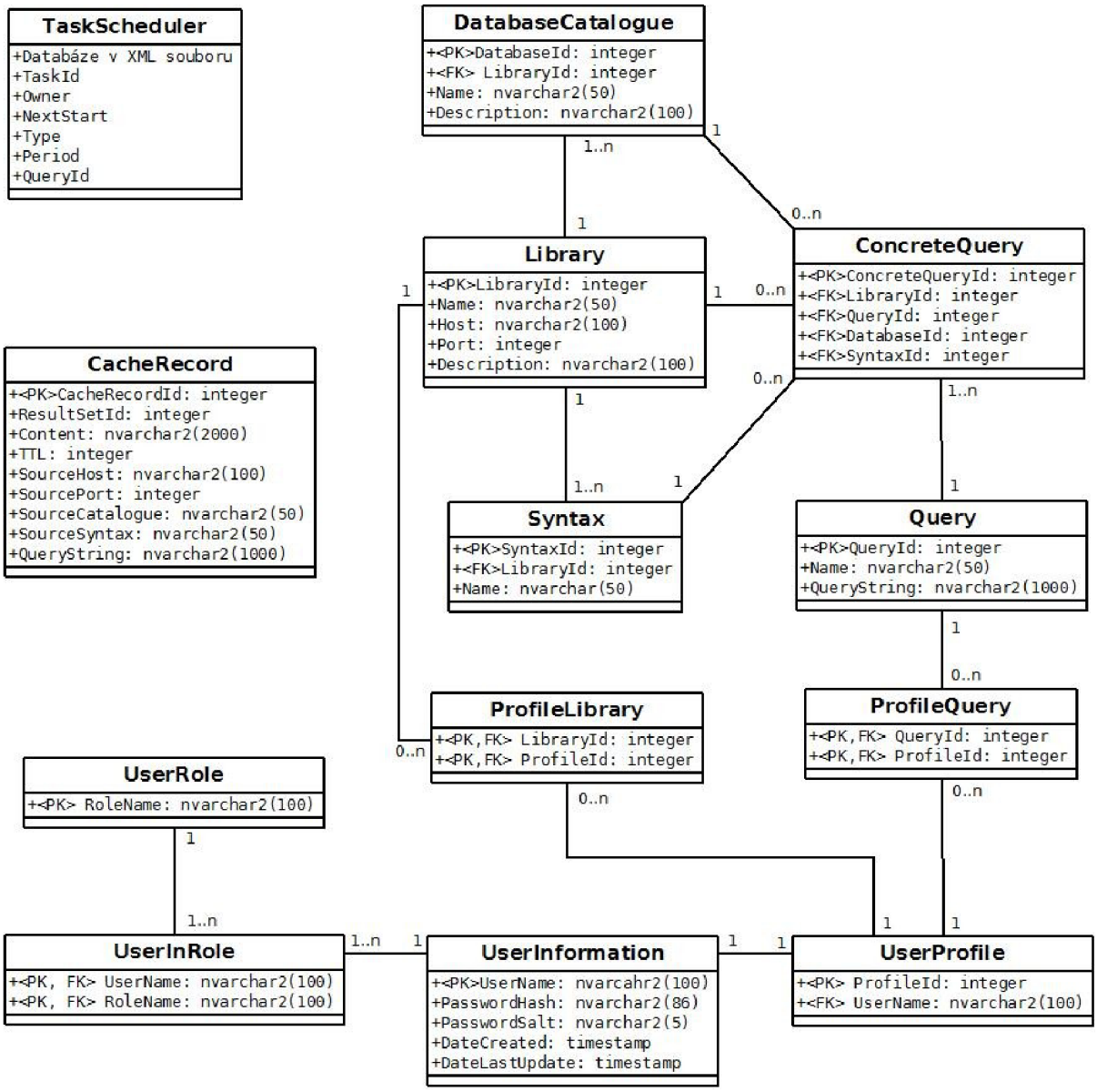
Tato kapitola je zaměřena na návrh řešení. V předchozích kapitolách byly již naznačeny některé střípky řešení a v této kapitole budou dále rozvedeny a budou navrhnuty další.

5.1 Architektura systému

Jednou z nejdůležitějších etap vývoje softwarového produktu je navržení architektury systému. Jak jsem naznačil v úvodu, bude systém postaven na architektuře technologie Silverlight s využitím WCF služeb, které mají podporovat komunikaci serveru a klientské aplikace Silverlight. Návrh architektury jsem rozdělil do několika logických celků.

5.1.1 Datový model

V analýze byly zjištěny data, která má systém uchovávat. Patří mezi ně uživatelské účty, profily, zdroje, dotazy a úložiště plánovače a dočasné paměti. Datový model databázových tabulek je popsán ER (Entity Relationship) diagramem na Obr.5. Entita *TaskScheduler* není tabulkou, ale ukázka obsahu záznamu v XML úložišti plánovače dotazů. Tabulka dočasné paměti (*CacheRecord*) nemůže být ve vztahu s tabulkami zdrojů a dotazů, protože by tak byli dané záznamy z této paměti dostupné pouze uživateli, který vlastní daný zdroj. To je nežádoucí, protože je potřeba jediná sdílená dočasná paměť pro všechny uživatele. Pro uložení samotného obsahu záznamu je použit datový typ *NVARCHAR2*, který má maximální velikost 2000 B, což není mnoho a záznamy tak musí být ořezány. Z tohoto vyplývá první námět na rozšíření v podobě použití jiného datového typu, například *BLOB*.



Obr.5: Entity relationship diagram.

Platforma .NET obsahuje EF (Entity Framework), který se používá pro datové modelování aplikací. Jeho hlavním cílem je zvýšit abstrakci při práci s daty. Odstiňuje návrháře od práce s datovými tabulkami. Místo toho pracuje s množinou entit, které mapuje na databázové tabulky. Definuje nad nimi operace a poskytuje jednotné rozhraní pro práci s nimi, tzv. datový kontext [11].

EF není dostupný v aktuální verzi Mono 2.6.4 a nelze jej proto použít. Systém WASOLIC bude ale používat základní princip z EF – práce s entitami.

Data budou vystavena pro klientskou aplikaci pomocí doménové služby. Budou se přenášet entity.

Databáze obsahuje pro operace vkládání, mazaní a aktualizování tabulek uložené procedury. Zdrojový kód SQL obsahuje tyto procedury pouze pro Oracle. Soubor SQL je uložen v projektu *WASOLIC.DAL*.

5.1.2 Bezpečnostní model

Systém WASOLIC pracuje s uživatelskými daty a musí tedy zajistit jejich nejvyšší možnou ochranu. Data navíc budou přenášena sítí mezi klientskou a serverovou stranou systému.

Bude použit bezpečnostní model založený na rolích. Každý uživatel bude spadat do některých rolí. V základní verzi bude systém obsahovat dvě role – administrátoři a uživatelé. Uživatelské účty budou chráněny uživatelským jménem a heslem. Uživatelské jméno běžných uživatelů bude jejich emailová adresa, která bude využitelná plánovačem dotazů pro odesílání výsledků. Tyto údaje jsou nejcitlivější data celého systému. Budou přenášena při každém volání vrstvy WCF služeb pro zajištění autorizace přístupu. Proto musejí být maximálně chráněna.

WCF služby implementovány v Mono, verze 2.6.4, neposkytují bezpečnou vazbu WCF služby, a komunikace tak probíhá po otevřeném kanálu. Uživatelské jméno nebude přenášeno ve formě hash, protože to sebou nese nároky na databázi, které ale nejsou vyváženy klady. Dokud nebude zabezpečen kanál komunikace pomocí kryptografie, nevyplatí se chránit uživatelské jméno. Heslo ovšem bude přenášeno jako hash a v databázi bude uloženo v této formě spolu se „solí“, použité při jejím vytváření. Jakmile se uživatel poprvé zaregistruje, vytvoří se hash hesla a toto se uloží do databáze. Od této doby, kdykoliv se uživatel přihlásí nebo bude volat WCF službu jako přihlášený uživatel, heslo bude přenášeno ve formě hash.

Jak jsem naznačil v předchozích odstavcích, při každém volání WCF služeb se bude provádět autorizace, zda je uživatel, který danou službu volá, tím za koho se vydává a má právo danou operaci služby volat. Za tímto účelem bude použit token, který zapouzdřuje identitu uživatele. Identita bude obsahovat uživatelské jméno, heslo a jiné potřebné údaje. Tento token je předáván jako první parametr každé operaci WCF služby, jak je vidět v ukázce Kód 1.

```
ReturnValue ServiceOperation(Token _token, type _param1, type _paramN);
```

Kód 1: Signatura operace WCF služby s bezpečnostním tokenem.

Existuje tady riziko, že pokud nebude kanál bezpečný, což s Mono 2.6.4 platí, tak může útočník odposlechnout celý token a jednoduše se vydávat za legitimního uživatele tím, že přes vlastního klienta služby pošle daný token a může tak získat uživatelská data. Nehledě na to, že přímo z tokenu odhalí emailovou adresu uživatele. Tento typ útoku je znám jako útok přehráváním.

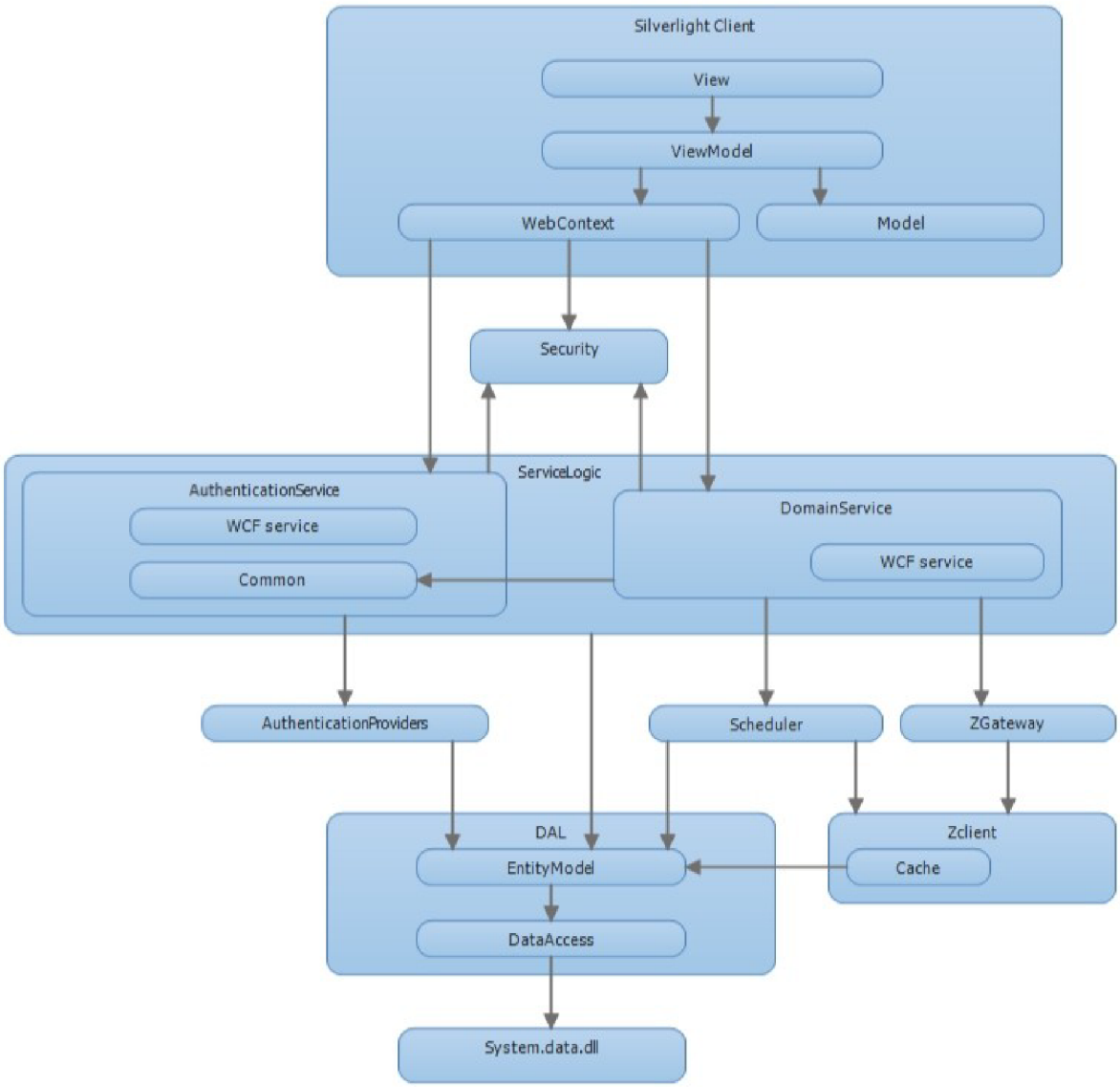
Možným zvýšením ochrany proti tomuto útoku by bylo přidání náhodně vygenerovaného čísla (tzv. Nonce) pro každé sezení (od přihlášení do odhlášení). Pokud by byla množina generovaných čísel periodická a útočník zjistil periodu, ochrana by byla rázem pryč. Navíc pokud by útočník odposlechl token v rámci daného sezení, mohl by ho v ní použít stejně jako jsem zmínil výše. Pokud by chtěl útočník odposlechnout heslo za účelem jeho použití přímo v aplikaci, tak toto je mu znesnadněno, protože je heslo ve formě hash. Dokud nebude kanál zabezpečen, existuje vyšší riziko nabourání systému ukradením bezpečnostního tokenu.

Pokud shrnu otázku bezpečnosti, tak je systém zabezpečen proti laickým uživatelům, kteří by se mohli maximálně pokusit uhádnout heslo jiného uživatele, popřípadě jej odposlechnout. Díky hash hesla je toto obtížné. Pokud se jedná o pokročilejšího útočníka, který si může napsat svého klienta WCF služby a odposlechnout celý token, tak proti takovému útočníkovi je systém velmi zranitelný. Pozvednutí bezpečnosti je možné pomocí zabezpečení kanálu, což není zatím možné.

5.1.3 Vrstvový model

V této části bude popsán vrstvový model, který odráží kompletní architekturu. V předchozích podkapitolách architektury jsem probral důležité architektonické části systému. V této podkapitole ještě všechno rekapituluji. Obr. 6 ukazuje celkovou architekturu systému pomocí vrstvového modelu.

O přístup k datům se stará vrstva *DAL* (*Data Access Layer*), která poskytuje vyšším vrstvám model entit. Vrstva WCF služeb (*ServiceLogic*) obsahuje dvě základní komponenty. Doménovou službu sloužící k práci s daty uživatelů, jako zdroje a dotazy ad. A dále autentizační službu, která zajišťuje bezpečnostní model systému WASOLIC. Přes tuto službu se uživatelé registrují, přihlašují/odhlašují do/ze systému. Využívá k tomu autentizační vrstvu poskytovatelů členství, rolí a profilů (*AuthenticationProviders*), která komunikuje s datovým kontextem modelu entit z DAL. Logika systému WASOLIC, co se týče vyhledávání, je postavena na vrstvách *Zclient* a *Zgateway*. *Zclient* zapouzdřuje Z-klienta, který používá ZOOM.NET pro vyhledávání a získávání záznamů. *Zgateway* je mezivrstva mezi doménovou službou a Z-klientem. Jedná se o Z-bránu. Právě přes tuto vrstvu probíhají požadavky na vyhledávání a získávání dat od klientské strany systému (Silverlight). Z-brána tyto požadavky předává vrstvě Z-klienta. Z-klient v sobě zapouzdřuje samostatnou vrstvu dočasné paměti *Cache*. Na klientské straně, která je postavena na technologii Silverlight se vyskytuje webový kontext (*WebContext*), který je konzument dvou zmíněných WCF služeb. Klient dále používá vývojový vzor Model-View-ViewModel (MVVM), který se skládá z pohledů (*View*), modelů pohledů (*ViewModel*) a modelu (*Model*). Pokud potřebuje model pohledu komunikovat se serverem, používá zmíněný webový kontext. Model obsahuje datové kontexty pro práci se zdroji a dotazy a další výhodná rozšíření. Vrstva Security je další komponentou zajišťující bezpečnostní model. Je používána oběmi stranami komunikace.

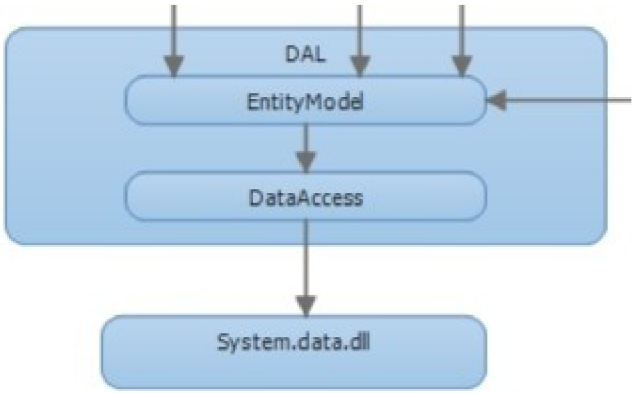


Obr. 6: Architektura systému WASOLIC.

5.2 Server

5.2.1 Datová vrstva

Datová vrstva je navržena tak, aby bylo možné dosáhnout přenositelnosti mezi různými databázovými stroji na úrovni kódu. Datová vrstva je ukázána na obrázku Obr. 7.



Obr. 7: Datová vrstva.

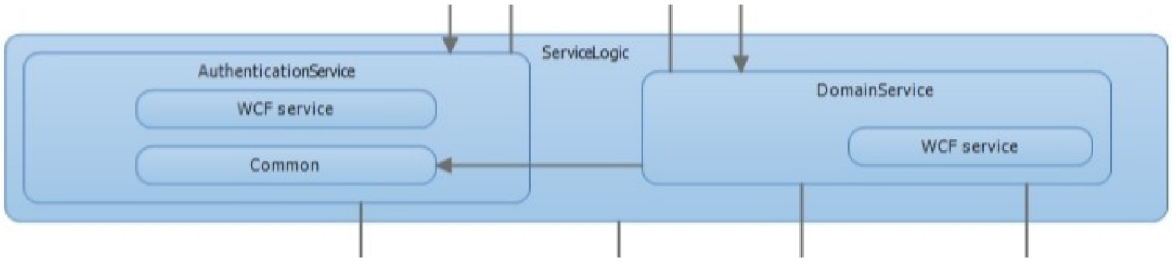
DAL (Data Access Layer) se skládá ze dvou podvrstev – *DataAccess* a *EntityModel.* Podvrstva *DataAccess* používá přímo ADO.NET pro přístup k databázi (znázorněno pomocí vazby na *System.Data.dll*). Pracuje s tabulkami a nepoužívá žádného specifického poskytovatele dat, aby bylo možné dosáhnout nezávislosti na datovém zdroji. Místo toho využívá návrhový vzor *Factory*, který ADO.NET podporuje. Specifický poskytovatel se pomocí továrních metod vytvoří až za běhu podle nastavení datové vrstvy v konfiguračním souboru, kde je uvedeno mimo jiné název databázového poskytovatele, prefixy parametrů dotazů SQL pro danou databázi, připojovací řetězec apod. Pokud by byly použity specifické vlastnosti konkrétního databázového stroje, které například nejsou standardizovány, tak tento přístup fungovat nebude. Systém WASOLIC ale počítá pouze se standardem SQL a může si dovolit takový přístup. Podvrstva *EntityModel* přistupuje k databázi pomocí nižší vrstvy *DataAccess*. Transformuje datové tabulky na entity, které poskytuje ostatním částem systému. Definuje entity a datový kontext obsahující metody pro práci s těmito entitami. Entity jsou definovány jako datové kontrakty, aby byli připravené pro WCF služby.

Jako databázový stroj bude použit Oracle. Pokud by se databáze přenášela na jiný databázový stroj, například na MySql, měla by se jen změnit konfigurace systému pomocí konfiguračního souboru a vše by mělo fungovat jak má.

5.2.2 Vrstva WCF služeb

Serverová strana vystavuje množinu operací pomocí vrstvy WCF služeb. Budou obsaženy dvě služby: autentizační a doménová. Autentizační bude sloužit pro autentizaci uživatelů a doménová bude vystavovat jejich data jako zdroje, úlohy, dotazy a také rozhraní Z-brány a plánovače dotazů. Doménová služba bude používat část metod autentizační služby pro zajištění autorizace přístupu při vyvolávání operací služby. Tyto metody bude volat lokálně a nikoli přes rozhraní služby. Kvůli tomu

budou tyto metody umístěny ve třídě v jiném jmenném prostoru (*Common*). Jedná se hlavně o validaci uživatele. Tato vrstva je ukázána v detailu na obrázku Obr. 8.



Obr. 8: Vrstva WCF služeb.

5.2.3 Vrstva plánovače dotazů

Systém WASOLIC musí být schopen ukládat úlohy uživatelů, které uchovávají informace o opakovaných dotazech. Tyto úlohy musí být někde uloženy. Nejvhodnějším úložištěm se zdá být XML soubor. Úlohy budou řazeny podle data zahájení.

Tento soubor bude přístupný přes vrstvu DAL. Lze do něj zapisovat, číst, upravovat a mazat úlohy. Tato vrstva musí obsahovat práci s vlákny. Více podrobností v kapitole implementace.

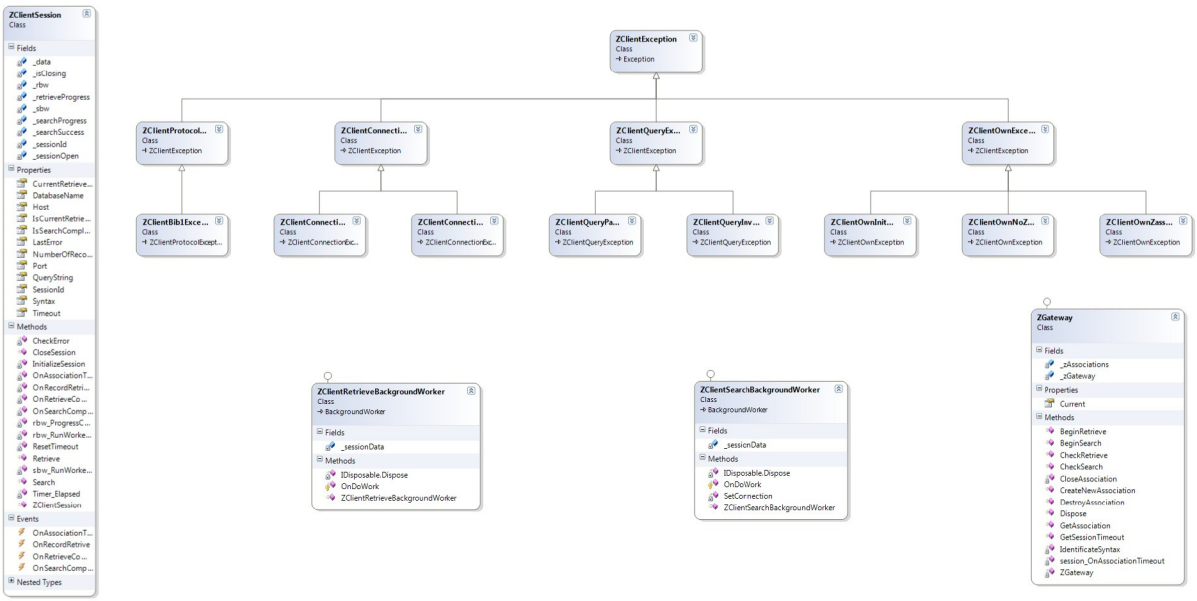
Úložiště plánovače je popsáno schématem XML. Na obrázku Obr. 9 se nachází jeho zkrácená verze.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="TaskSchedulerSchema">
 <xs:element name="TaskSchedulerSchema">
  <xs:complexType>
   <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="TaskScheduler">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="TaskId" type="xs:string" minOccurs="1" />
       <xs:element name="Owner" type="xs:string" minOccurs="1" />
       <xs:element name="NextStart" type="xs:dateTime" minOccurs="1" />
       <xs:element name="Type"type="xs:string" minOccurs="1" />
       <xs:element name="Period" type="xs:int" minOccurs="1" />
       <xs:element name="Query"type="xs:int" minOccurs="1" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:choice>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Obr.9: Schéma XML úložiště plánovače dotazů.

5.2.4 Vrstva Z-brány a Z-klienta

Logika pro vyhledávání a získávání záznamů se celá nachází na straně serveru, což odpovídá definici Z-brány. Systém WASOLIC používá takové složení vrstev, aby bylo adaptovatelné pro použití s doménovou službou. Jelikož má systém podporovat paralelní vyhledávání, musí být Z-klient patřičně připraven. Lze použít režim událostí z balíčku YAZ, který se mi ale zdál náročný. Proto jsem navrhl řešení, které je založeno na vláknech. Vrstva Z-klienta obsahuje hlavní třídu vyhledávacího sezení, která zapouzdřuje informace potřebné pro vyhledávání a také výsledky vyhledávání. Operace vyhledávání a získávání jsou delegovány na pracovní vlákna a lze tak relativně snadno dosáhnout možnosti paralelních sezení. Z-klient bude používán dvěmi komponentami, a to pracovními vlákny plánovačem dotazů a Z-bránou. Každá komponenta pochází z jiného prostředí. Plánovač je součástí spojeného prostředí a Z-brána je součástí rozpojeného prostředí. Pro oba typy prostředí musí mít třída sezení Z-klienta rozdílné rozhraní. Vlákna plánovače budou využívat rozhraní založené na událostech a Z-brána bude používat rozhraní založené na vlastnostech. Diagram hlavních tříd Z-klienta a Z-brány, včetně vyjímek Z-klienta, které jsou mapovány na ZOOM.NET vyjímky, se nachází na obrázku Obr.10.



Obr.10: Diagram základních tříd Z-klienta a Z-brány.

Ve třídě *ZClientSession* jsou vidět události, které jsou používány pracovními vlákny plánovače. Jedná se o události informující o ukončení vyhledávání nebo získávání nebo přijetí záznamu. Oproti tomu vlastnosti jako *IsSearchCompleted* nebo *IsRetrieveCompleted* jsou použity v rozpojeném prostředí z-bránou. Třída *ZGateway* zapouzdřuje metody, které jsou volány z doménové služby v rámci rozpojeného prostředí. Jedná se o metody vytváření sezení (Z-brána si udržuje seznam

vytvořených sezení), rušení sezení, zahájení vyhledávání nebo získávání, a metody pro periodické kontrolování vlastností.

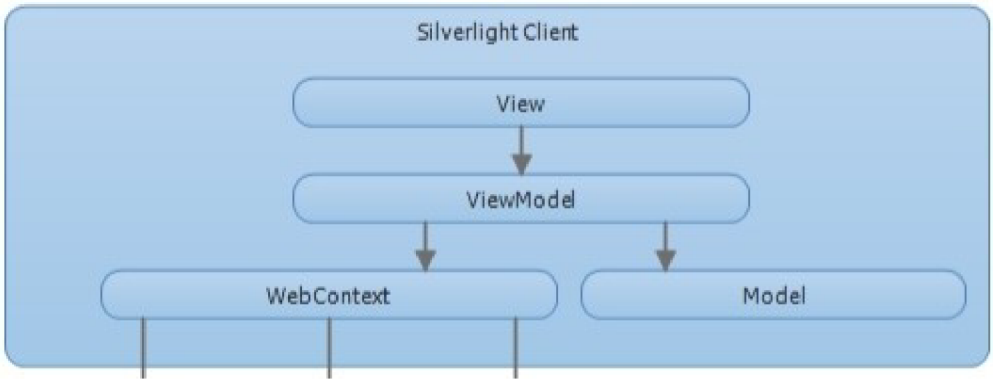
5.2.5 Sdílená vrstva bezpečnosti

Klientská i serverová strana musí mít k dispozici metody pro výpočet hash hesla a s tím spojený výpočet „soli“ pro výpočet hash. Tyto metody budou zapouzdřeny ve třídě *Hash*, která bude součástí knihovny Security. Tato knihovna bude dostupná jak na serverové straně, tak i na klientské. Ve skutečnosti nebude kód sdílený, ale bude duplicitně uložen jak v řešení serveru, tak i v řešení klientu.

5.3 Klient

5.3.1 Model-View-ViewModel

Uživatelské prostředí se bude nacházet na klientské aplikaci Silverlight/Moonlight. Bude použit deklarativní přístup k samotnému vývoji pohledu pomocí jazyka XAML s minimálním kódem na pozadí. Spolu s tímto bude použit návrhový vzor MVVM (Model-View-ViewModel), který podporuje minimální kód v pozadí spolu s maximálním použití vázání ovládacích prvků na prvky z ModelView a schopností testovatelnosti jednotky ModelView bez nutné vazby na View. Obrázek Obr.11 ukazuje vrstvu klientské strany systému WASOLIC.



Obr.11: Vrstva klientské strany systému.

Bude kladen důraz na vytváření znovupoužitelných ovládacích prvků v pohledech, aby mohli být používány ve více částech uživatelského prostředí.

Systém musí být schopen nabídnout efektivní prostředí pro práci se zdroji, dotazy a výsledky vyhledávání a také přívětivé a dostačující možnosti vyhledávání.

Při práci s dotazy a zdroji bude použit přístup, který používá WCF RIA spolu s EF. Jedná se o to, že uživatel má možnost pracovat s daty takovým způsobem, že si sám rozhodne, zda provedené změny uloží do systému nebo naopak neuloží.

Možnost zmíněná v předchozím odstavci bude dosažena pomocí datových kontextů (jak pro zdroje, tak pro dotazy), které si budou uchovávat záznamy podle stavu přidáno, aktualizováno nebo odebráno a až na požádání (stisk tlačítka) tyto změny bude propagovat na server (přes webový kontext). Tyto kontexty odpovídají modelu z probíraného vývojového vzoru MVVM.

Výsledky vyhledávání budou zobrazeny v tabulce, která bude používat stránkování po deseti záznamech. Každé sezení Z-klienta bude přístupné v záložce ovládacího prvku *TabControl*.

5.3.2 Webový kontext

Webový kontext je konzument doménové a autentizační služby, které běží na serveru. Pro každou službu bude obsahovat jednu třídu. Autentizační třída se stará o uživatele aplikace, udržuje uživatelský kontext, což zahrnuje token aplikačního sezení, informace o uživateli (přihlášeném nebo anonymním) a stará se o operace jako registrace, přihlášení nebo odhlášení. Třída pro doménovou službu bude obsahovat operace pro práci se zdroji, dotazy, úlohami a jako speciální část bude obsahovat klienta Z-brány. Klient Z-brány si uchovává informace o probíhajících sezení Z-klienta. Hlavním obsahem každé informace je seznam záznamů, který je používán k zobrazování výsledků v pohledu přes jeho ViewModel.

6 Implementace

Tato kapitola obsahuje implementační detaily systému WASOLIC.

6.1 Datová vrstva

Jak bylo zmíněno v návrhu, datová vrstva bude postavena na platformě ADO.NET. Byl použit přístup nezávislý na datovém zdroji, při dodržení určitých podmínek v implementaci databáze. Kód 2 ukazuje použití návrhového vzoru *Factory* v praxi.

```
/*Vytvoření poskytovatele dat pro konkrétní databázi za běhu*/
DbFactory provider = DbFactories.GetFactory("System.Data.OracleClient");
/*Pomocí poskytovatele vytvářím objekty pro práci s daty*/
DbConnection connection = provider.CreateConnection();
DbCommand cmd = provider.CreateCommand();
DbParameter param = provider.CreateParameter();
/*Ukázka použití parametru SQL dotazu s prefixem pro danou databázi*/
cmd.CommandText = "SELECT * FROM Table WHERE ID=" + paramPrefix + "Id";
param.ParameterName = paramPrefix + "Id";
param.Value = 1;
cmd.Parameters.Add(param);
cmd.Connection = connection;
/*Nyní se může provést příkaz ...*/
```

Kód 2: Ukázka použití abstraktních poskytovatelů na datové vrstvě.

Konfigurační soubor obsahuje informace pro datovou vrstvu jako typ poskytovatele (např. *System.Data.OracleClient*), připojovací řetězec, prefix parametrů a schéma tabulek (pro použití např. na SQLServeru).

Entitní model musí být kvůli problémům s WCF službami na Mono 2.6.4 trochu ořezán. Nemohou být dodrženy veškeré vazby mezi entitami, hlavně vazby cyklické [10]. Ve výsledku by to ale vadit nemělo.

6.2 Bezpečnostní metody a třídy

WASOLIC systém obsahuje knihovnu podpůrné třídy bezpečnosti. Tato třída obsahuje metody pro výpočet „soli“ pro hash hesla a metodu pro vytvoření hash hesla. Pro výpočet hash hesla je použit algoritmus SHA s klíčem dlouhým 256 B.

Bezpečnost se týká také zpracování chyb, které se mohou vyskytnout na všech vrstvách systému. Důležité je zaměřit se zejména na chyby, které vzniknou na serverové straně a klient na ně musí být upozorněn. Aplikace Silverlight nemá podporu pro *Fault Contracts*, které se používají u WCF služeb. *Fault Contracts* spočívají v tom, že chyba je jednoduše zabalena do datového kontraktu a platforma .NET přenáší tuto chybu jako výjimku přes kanál WCF služby, a tato výjimka potom může být zachycena *catch* blokem na klientovi. Systém WASOLIC používá způsob, kdy všechny operace vrací v datové obálce jak samotné návratové hodnoty, tak také údaje o chybě. Kód 3 ukazuje všechny datové kontrakty, které souvisí se zmíněnou obálkou (jedná se o zjednodušující ukázku, názvy tříd se mohou ve zdrojovém kódu lišit, pro vyjádření myšlenky je toto dostačující).

```
/*
    Návratová hodnota operace. Jedná se generický typ, aby mohla
    každá operace vracet svá specifická data
*/
[DataContract]
public class ResultData<T>
{
    [DataMember]
    public T Data { get; set;}
}



/*Zjednodušená třída chyb. Ve skutečnosti obsahuje více vlastností*/
[DataContract]
public class ErrorData
{
    [DataMember]
    public string Message;
}
```

```
/*
    Obálka, která je vráce každou operací WCF služby.
    Kvůli generické třídě výsledku musí být také generická.
*/
[DataContract]
public class OperationReturnWrapper<T>
{
    [DataMember]
    public ResultData<T> Result {get; set;}

    [DataMember]
    public ErrorData Error {get; set;}
}
```

Kód 3: Obálka pro data a chyby vrácená každou operací WCF služby.

Zjistil jsem, že WCF služby implementované doposud na platformě Mono 2.6.4 mají problémy s některými typy návratových hodnot. Jako datové kontrakty nesmí být použity objekty třídy *List* a *Dictionary*. Smí být použity jejich rozhraní jako *IList* a *IDictionary*. Dalším problémem je generický typ. Ten nesmí obsahovat zmíněné třídy ani rozhraní. Pokud se dané třídy zapouzdří do jiné třídy, která je použita pro tvorbu generického objektu, tak služba funguje (ukázka Kód 4). Více o nedostatcích zmíněných WCF služeb v Mono na wiki stránkách projektu WASOLIC [10].

```
/*Tato konstrukce není zatím v Mono možná*/
[OperationContract]
OperationReturnWrapper<IList<Library>> GetLibraries();

/*Musí se použít zapouzdřující třída spolu s rozhraním*/
[DataContract]
public class DataListLibrary
{
    [DataMember]
    public IList<Library> List {get; set;}
}

/*Nyní je možné použít seznam ve WCF operaci*/
OperationContract<DataListLibrary> GetLibraries();
```

Kód 4: Použití seznamu v generickém typu jako datového kontraktu WCF služby pro Mono.

6.3 Vrstva autentizační služby

Vrstva autentizace používá vlastní poskytovatele členství, rolí a profilů pro autentizaci uživatelů. Vystavuje operace pro přihlášení/odhlášení do/ze systému, pro registrování nových uživatelů apod. Operace dodržují model bezpečnosti, popsaný v kapitole 5.1.2. Většina z těchto operací vrací nový bezpečnostní token. Operace *Login* například vrací identitu úspěšně přihlášeného uživatele v tomto tokenu, který je poté používán pro autorizaci volání dalších operací. Používá entity z entitního modelu vrstvy DAL. Jedná se o uživatelskou entitu, role apod.

Poskytovatel členství poskytuje statické metody pro práci s uživatelskými údaji, jako uživatelské jméno, heslo, datum vytvoření apod. Poskytovatel rolí poskytuje statické metody pro práci s rolemi jako přidání/odebrání uživatele do/z role, zjištění, zda se uživatel nachází v dané roli apod. O profily se stará poskytovatel profilů, který obsahuje dvě statické metody. Jednu pro získání uživatelského profilu, druhou pro jeho aktualizaci. WASOLIC obsahuje pouze identifikační číslo profilu uživatele, které je použito v relaci s dotazy, knihovními zdroji. Všichni poskytovatelé přistupují k databázi pomocí modelu entity vrstvy DAL.

Při testování této WCF služby jsem narazil na samé dno možností WCF služeb implementovaných v Mono ve verzi 2.6.4. Problém se objevil při přenášení hash dat. Některé znaky službě vadili a vyvolávaly proto vyjímku. Tento problém musí odstranit vývojáři Mono a věřím, že se jim to v nejbližší vydané verzi povede. Bohužel díky tomuto problému není splněna přenositelnost, alespoň do doby, než bude opravena zmíněná chyba.

6.4 Vrstva doménové služby

Doménová služba vystavuje operace, které slouží pro manipulaci s entitami typu knihovna, dotazy a úlohy. Každá operace doménové služby dodržuje signaturu uvedenou v ukázce Kód 1, tzn., jako první parametr přebírá bezpečnostní token s identitou volajícího uživatele. Pokud operace vyžaduje autorizaci, tak právě tento token je použit pro ověření identity uživatele (je použita metoda validace z autentizačního modulu ze jmenného prostoru *Common*). Pokud nesouhlasí uživatelské jméno a heslo v identitě, operace je zamítnuta.

Operace pro práci se zdroji a dotazy jsou typu vlož, aktualizuj, odstraň a získej. Přímo používají entitní model vrstvy DAL. U těchto operací je vyžadována zmíněná autorizace. Nemělo by se stát, že např. běžný uživatel odstraní výchozí zdroje, které patří administrátorovi.

Dále jsou zde obsaženy operace, sloužící jako rozhraní plánovače dotazů. Operace jsou stejného typu jako u zdrojů a dotazů. Také požadují autorizaci.

Poslední skupinou operací jsou operace tvořící rozhraní Z-brány. Tyto operace jsou specifického typu. Slouží k vytváření nebo odstraňování sezení Z-klienta, dále zahájení vyhledávání nebo získávání výsledků a k periodické kontrole průběhu daného sezení.

6.5 Z-klient

Z-klient je implementován tak, jak jsem popisoval v kapitole návrhu. Hlavní třídou tedy je *ZclientSession,* která uchovává informace o svém sezení a obsahuje metody pro vyhledávání a získávání dat. Samotná činnost vyhledávání a získávání je delegována na pracovní vlákno, které je implementováno pomocí třídy *BackgroundWorker* z .NET frameworku. Nabízelo se použít klasické vlákno třídy *Thread*, ale toto nemá požadované specifika pro tento úkol, kterými jsou možnost sledování průběhu pomocí událostí a zpracování vyjímek přímo v hlavním vláknu aplikace. Implementoval jsem dvě třídy, jednu pro vyhledávání a druhou pro získávání. Před zrušením sezení je čekáno na ukončení případně běžících vláken.

Vyhledávací vlákno volá metodu *Search* z knihovny ZOOM.NET, která vytváří Z-asociaci a podává požadavek na vyhledávání cíli. Search je blokováno dokud nepřijde výsledek vyhledávání. Jakmile se tak stane, předá vlákno informaci o svém ukončení a o výsledku vyhledávání pomocí události, která je zachycena v rámci třídy sezení na hlavním vláknu.

Vlákno pro získávání dat pracuje na stejném principu s tím rozdílem, že po každém přijatém záznamu vyvolává navíc událost, pomocí které předá získaný výsledek. Pracovní metoda má ale větší složitost než vyhledávací vlákno. Kód 5 ukazuje pseudokód této metody.

```
for(každý požadovaný záznam)
{
    // Pro tuto prvni podmínku jsem musel provést úpravu
    // v ZOOM.NET. Viz 6.7
    if(YAZ keš obsahuje požadovaný záznam - v ResultSet)
    {
          získej záznam z result set přes ZOOM.NET.
    }

    else if(zaznam je v keši systému WASOLIC)
    {
          získej záznam z cache systému WASOLIC.
    }
```

```
        else
        {
              // záznam nebyl ještě nikdy získán nebo byl před časem
              // odstraněn z keše WASOLIC
              // Toto volání je blokující, ale ve vláknu to nevadí.
              získej záznam ze serveru přes ZOOM.NET
        }

        // Vyvolej událost o získání jednoho záznamu, čímž informuješ
        // hlavní vlákno.

        // Pokud keš systému WASOLIC neobsahuje získaný záznam,
        // ulož jej do ní.
  }
```

Kód 5: Kód pracovního vlákna pro získávání dat z cíle.

6.6 Z-klient cache

Jak bylo již několikrát zmíněno, Z-klient implementuje svoji vlastní dočasnou paměť, kam ukládá získané záznamy ze všech vyhledávání a získávání dat. Každý záznam je v dočasné paměti uchován dokud mu nevyprší čas TTL (Time To Live). Tento čas lze nastavit v konfiguraci systému. TTL je dekrementováno při každém přístupu k záznamu. Výchozí, testovací, hodnota TTL je 2. Dočasná paměť je samostatná jednotka a běží na pracovním vláknu aplikace (klasické vytváření instancí této třídy při použití). Používá log-soubor, takže pokud v ní dojde k chybě, tak nejsou ovlivněny zbylé části systému a protokol o chybě je zapsán do tohoto souboru. Záznam obsahuje čas, metodu, ve které došlo k chybě a popis vyjímky. Pro přístup k databázi používá entitní model vrstvy DAL.

6.7 Z-brána

Z-brána je implementována podle návrhu. Její operace lze využít k práci se sezeními Z-klient. Uchovává si seznam sezení, se kterými pracuje. Instance Z-brány běží v pracovním vlákně aplikace a existuje přes celý život systému, tzn. je vytvořena při startu celého systému a zrušena na jeho konci. Tohoto je dosaženo použitím návrhového vzoru *Singleton* Při rušení instance musí být zajištěno počkání na ukončení pracovních vláken všech vytvořených sezení Z-klienta. Tohoto je dosaženo prostým počkáním na zrušení sezení, které se samo stará o čekání na případné běžící pracovní vlákna.

6.8 ZOOM.NET

Systém WASOLIC používá ZOOM.NET jako jádro Z-klienta. Tato knihovna zapouzdřuje volání funkcí jazyka C z knihovny YAZ. Je šířen volnou licencí a tak jsem si pro své účely upravil nebo přidal některé metody a třídy. Ve zdrojových kódech jsem na místa, která jsem upravoval, přidal svůj komentář se jménem. Následuje seznam změn v knihovně ZOOM.NET (Zoom.Net.YazSharp):

- *Zoom.Net.YazSharp.dll.config* - přidáno mapování Yaz.dll knihovny na systému Linux a cesta k souboru pro CQLParser.
- *CQLParser.cs* – konvertuje CQL dotaz na PQF. Přidáno mnou.
- *CQLTransformFile.txt* – transformační soubor pro CQLParser.
- *MarcConvertor.cs* – konvertuje záznam MARC na MARC v jiném módu. Přidáno mnou.
- *ResultSet.cs* – řádek 61. Metoda pro podporu dočasné paměti WASOLIC
- *Yaz.cs* – pro přidané třídy CQLParser a MarcConvertor jsem přidal interoperabilní prostředí, které volá C funkce z knihovny YAZ. Začínají na řádcích 408 a 676.

6.9 Plánovač dotazů

Stejně jako Z-brána, také plánovač dotazů má životnost stejnou jako celá aplikace, opět je použit návrhový vzor *Singleton*. Plánovač se skládá ze dvou hlavních celků. První část běží v pracovním vláknu (zmíněný Singleton). Tato část slouží jako rozhraní pro přicházející požadavky z doménové služby. Jsou obsaženy metody pro získání, vložení, aktualizování nebo odstranění úloh. Druhou částí je pracovní vlákno plánovače, které se stará o vybírání úloh z fronty (úložiště XML), delegování jejich provádění a přeplánování dle informací uložených v úloze. Toto vlákno může být spuštěno, zastaveno nebo probuzeno hlavním vláknem. Viz. dále. Samotné provedení úlohy spočívá v provedení vyhledávání a získávání informací s odesláním výsledků na email přes SMTP (Simple Mail Transfer Protocol) klienta. Pracovní vlákno plánovače deleguje provádění úkolů podřízeným pracovním vláknům, které jsou opět odvozeny od *BackgroundWorker* třídy. Tyto pracovní vlákna pracují se Z-klientem a jeho spojeným prostředím (události). Je to samostatná jednotka, takže pokud v ní dojde k chybě, neohlašuje to pracovnímu vláknu plánovače, ale protokol chyby zapíše do log-souboru, stejně jako je tomu u modulu dočasné paměti. Pracovní vlákno je schopno zvládnout i paralelní vyhledávání. Při ukončení pošle pracovnímu vláknu plánovače událost, která je zpracována tak, že objektu ukončeného vlákna je odstraněn z vlastního seznamu pracovních vláken.

Nyní se vraťme k pracovnímu vláknu plánovače, které deleguje provádění úloh, čte čelo fronty atd. Práce s tímto vláknem je založena na metodách *Wait* a *Pulse* objektu *Monitor*. Pracovní vlákno plánovače pracuje v nekonečné smyčce, v níž provádí tyto činnosti. Nejdříve si přečte čelo fronty

úloh, podle času zahájení rozhodne, zda se má provést nyní nebo ne. Pokud se má provést, tak deleguje činnost novému vláknu a opakuje cyklus. Pokud se nemá nyní provést, tak se pomocí metody Wait uspí. Použil jsem menší kličku s vypršením času. Metodě Wait jsem předal čas, který uplynul do další události a po tomto čase se toto pracovní vlákno probudí samo a opět bude opakovat cyklus. Ovšem já potřebuji, aby toto vlákno mohlo být vzbuzeno při změně XML úložiště. Toto probrání zajišťuje hlavní vlákno aplikace při přidání, smazání nebo změně dat v úložišti (frontě). K probrání používá metodu *Puls*. Obě tyto metody pracují nad jedním zámkem, který je uvolněn, pokud pracovní vlákno časovače spí. Pokud je zámek uvolněn (vlákno spí), tak může hlavní vlákno přistupovat k úložišti. Hlavní vlákno smí také pracovní vlákno ukončit, například při ukončení sytému. Toto je možné díky sdílené proměnné, kterou si pracovní vlákno periodicky kontroluje při získávání úlohy z čela fronty. Pokud dá hlavní vlákno pokyn k ukončení, pracovní vlákno počká na svá, případně běžící pracovní vlákna a poté se ukončí. Více se o metodách *Wait* a *Pulse* můžete dozvědět v [13].

6.10 Klientská strana systému

Klientská strana aplikace byla implementována technologií Silverlight s možností portace na Moonlight. Byla použita verze Silverlight 3.0. Implementace se řídí návrhem a používá vzor MVVM. Při implementaci ovládacích prvků byl kladen důraz na znovupoužitelnost. Například prvek pro výběr zdrojů je použit jak ve správě zdrojů, tak také v zadávacích formulářích vyhledávání. Dalším příkladem může být dialog zobrazující výsledky vyhledávání, který je při paralelním vyhledávání použit v každém sezení. Ukázka uživatelského prostředí se nachází v manuálu systému WASOLIC.

6.11 Problémy s kompatibilitou

Jak jsem naznačil v kapitole 6.2 a 6.3, s WCF implementací na platformě Mono ve verzi 2.6.4 jsou zatím problémy. Přes některé jsem se dostal, ale některé jsou už záležitostí dalšího vývoje týmem Mono. Bohužel se mi díky tomuto nepodařilo zatím splnit přenositelnost, ale jakmile vyjde nová verze platformy Mono a budou v ní opraveny nedostatky této rané verze WCF služeb, tak by měla být přenositelnost dodržena. Věřím, že se to vývojářům Mono úspěšně podaří.

7 Testování

Testování probíhalo v celém průběhu vývoje systému. Po každém dokončeném modulu se prováděl jeho test. Při dokončení více spolupracujících modulů se prováděli integrační testy. Právě při testování jsem narazil na chyby v implementaci WCF služeb na platformě Mono. Nebyly vytvořeny žádné regresní testy a testování probíhalo jednorázově a manuálně. Testovací projekt je přítomný ve zdrojových kódech.

7.1 Testy jednotek

Byli testovány následující jednotky:

- Vrstva DAL
- Vrstva poskytovatelů členství, rolí a profilů
- Vrstva Z-klienta
- Plánovač dotazů

7.2 Integrační testy

Byli prováděny následující integrační testy:

- Autentizační služba spolu s poskytovateli
- Doménová služba s datovou vrstvou, plánovačem a Z-bránou
- Klientská strana systému spolu s WCF – hotové řešení

7.3 Problémy současného řešení

Během testování systému jsem narazil na některé chyby, které nejsou opraveny. Mohou se samozřejmě objevit i nové nenalezené chyby. Většinu nalezených chyb jsem ale opravil. Následuje seznam neopravených chyb:

- Při prvním spuštění klientské aplikace po startu systému se objevuje chyba v klientské části systému - „DatabaseCatalogue neobsahuje vlastnost Description“. Stačí odkliknout dialog a aktualizovat prohlížeč, aby se aplikace znovu načetla a vše funguje jak má. Chyba je zřejmě způsobena mou úpravou generovaného kódu proxy klienta WCF služby kvůli validaci. Nelze totiž použít *MetadataType* pro úpravu entit a musí se provádět zásah do generovaného kódu. Je pravděpodobné, že toto je příčina.

- Další a zatím poslední chybou, kterou se mi nepodařilo odlalit je to, že server někdy vrací chybu „*NotFound*“. Tato chyba by také neměla mít velký dopad na běh klientské aplikace. Stačí odklepnout a případně opakovat akci. Tato chyba se objevuje, zdá se, náhodile, ale myslím si, že je způsobena špatnou prací z některými referenčními typy na serveru.
- Přidávám ještě jednu chybu, kterou se mi, zdá se, podařilo odlalit, ale myslím, že na jiném stroji se může opět vyskytnout. Je to práce s vlákny na klientské straně aplikace. Takže pokud se aplikace zasekne, a to hlavně v části procházení výsledků vyhledávání, je pravděpodobné, že za to může pracovní vlákno spravující čas vypršení sezení.

8 Nasazení aplikace

Tato kapitola stručně pojednává o možnostech nasazení systému WASOLIC.

Jelikož bude WASOLIC přenositelný měl by fungovat jak na serveru Apache, tak IIS (bohužel přenositelnost nebyla zatím dodržena. Viz kapitola 6.10). Jako databázoví stroj může být použit „jakýkoliv“, který splňuje SQL standard a má svého poskytovatele na platformě .NET. WASOLIC používá Oracle běžící na serveru Berta na Fakultě informačních technologií v Brně.

Jako linuxová varianta hostování systému se jeví školní server Merlin nebo Eva. Ani na jednom bohužel neběží podpora webového serveru Apache pro Mono (modul mod_mono). Hostingů, které podporují Mono je zatím málo a jedná se o placené služby. Nabízí se možnost hostovat systém na některém IIS hostingu, ovšem u mého stávajícícího poskytovatele je problém s připojením na databázi Oracle, která je v jiné síti. Zatím jedinou možností je použití systému WASOLIC na vývojovém stroji nebo domácím serveru, který zatím není připraven.

Pro shrnutí uvádím důležité komponenty pro hosting:

- Webový server IIS nebo Apache s podporou mod_mono (Mono).
- Databázový stroj, například Oracle.
- Knihovny YAZ, ZOOM.NET a jím přidružené (viz. [8]).
- .NET Framework 3.5 SP1 nebo Mono (vyšší než 2.6.4).

9 Závěr

Systém poskytuje veškerou funkčnost, která byla v zadání požadována. Implementuje správu zdrojů, dočasnou paměť i plánovač opakovaných dotazů. Navíc implementuje možnost paralelního vyhledávání a přidává některé prvky, které zefektivňují práci uživatelů.

Jako možné rozšíření se nabízí použití jiného datového typu pro uložení obsahu záznamu v dočasné paměti. Současný typ dovoluje použít pouze 2000 B a záznamy tak musí být ořezány. Klientské prostředí zobrazuje nalezené záznamy v jejich surovém tvaru, proto dalším možným rozšířením je modul, který konvertuje formáty záznamů na čitelnější tvar.

Přenositelnost se mi nepodařila splnit díky nekompatibilitě WCF služeb mezi platformami .NET a Mono. Celkový přínos práce ale hodnotím kladně, protože jsem měl možnost nahlédnout do větší hloubky použitých technologií.

Literatura

[1] Rubringer, Tomáš. Z39.50. Ikaros [online]. 1999, roč. 3, č. 8 [cit. 27.04.2010]. Dostupný z WWW: <http://www.ikaros.cz/node/1034 >. URN-NBN:cz-ik1034. ISSN 1212-5075.

[2] Morgan, Lease, Eric. An Introduction to the Search/Retrieve URL Service (SRU). 2004 [cit. 27.04.2010].
Dostupný z WWW: <http://www.ariadne.ac.uk/issue40/morgan/intro.html>.

[3] Bartoš, Ivan, Šmilauer, Bohdan. Získávání dat z informačních systémů (Z39.50): Definice aplikačních služeb a specifikace protokolu. Praha, 2002. 192 s.
Dostupné z WWW: <http://old.stk.cz/ZIG/Z39.50.zip>.

[4] Lunau, Carrol, Moen, William, Miller Paul. Bath profil Verze 2. Přel. ZIG – Czechia. 1. vyd. Praha, 2002. 58 s. Přel. z: Bath profil version 2.
Dostupné z WWW: <http://old.stk.cz/ZIG/Bath2.doc>.

[5] CQL: Contextual Query Language (SRU Version 1.2 Specifications) [online]. Page updated 22.08.2008 [cit. 27.04.2010].
Dostupný z WWW: <http://www.loc.gov/standards/sru/specs/cql.html>.

[6] Mono: Start [online]. [cit. 27.04.2010].
Dostupný z WWW: <http://www.mono__project.com/Start>.

[7] Mono: Moonlight [online]. [cit. 27.04.2010].
Dostupný z WWW: <http://www.mono-project.com/Moonlight>.

[8] Index Data: Software, YAZ [online]. [cit. 28.04.2010].
Dostupný z WWW: <http://www.indexdata.com/yaz>.

[9] Source Forge: Zoom.Net, [online]. [cit. 28.04.2010].
Dostupný z WWW: <http://sourceforge.net/projects/zoomdotnet/>.

[10] Assembla: WASOLIC [online]. [cit. 28.04.2010].
Dostupný z WWW: <https://www.assembla.com/wiki/show/wasolic>.

[11] MSDN: The ADO.NET Entity Framework Overview [online]. [cit. 28.04.2010].
Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/aa697427%28VS.80%29.aspx>.

[12] Smith, Josh. MSDN: WPF Apps With The Model-View-ViewModel Design Pattern [online]. [cit. 29.04.2010].
Dostupný z WWW: <http://msdn.microsoft.com/en-us/magazine/dd419663.aspx#id0090006>.

[13] Josepha, Albahari. Vlákna v C#. Přel. Jakub Kottnauer. 81 s. Přel. z: Threading in C#. [cit. 30.04.2010].
Dostupné z WWW: <http://www.albahari.com/threading/threading_czech.pdf>.

Seznam schémat

Seznam zdrojových kódů

Seznam zkratek

SRU	Search and retrieve via URL
SRW	Search and retrieve via Web
WCF	Windows communication foundation
HTTP	Hyper text transport protocol
SOAP	Service-oriented architecture protocol
URL	Uniform Resource Locator
SOA	Service-oriented architecture
ISO	Internacional Standard Organization
BER	Basic Encoding Rules
MARC	Machine-readable cataloging
SUTRS	Simple unstructured record syntax
XML	Extensible markup language
XAML	Extensible Application Markup Language
CQL	Contextual Query Language
BNF	Backus-Naur Form
SQL	Structured Query Language
WASOLIC	A Web-based Application to Search Online Library Catalogues
ER	Entity Relationship
EF	Entity Framework
DAL	Data Access Layer
MVVM	Model-View-ViewModel
TTL	Time To Live
SMTP	Simple Mail Transfer Protocol

Seznam příloh

Dodatek A

Obsah CD

- Soubor wasolic_manual.pdf
- Adresář WASOLIC, který obsahuje řešení bakalářské práce.
- Adresář doc, který obsahuje tuto zprávu ve formátu PDF.