VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

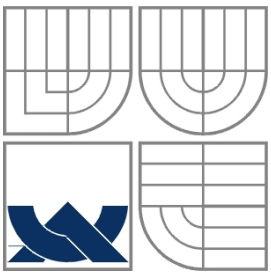
STRATEGICKÝ PORTÁL SYSTÉMU PROCESNÍHO ŘÍZENÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

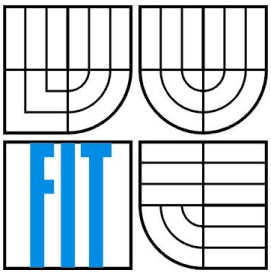
AUTOR PRÁCE
AUTHOR

MARTIN MADĚRA

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

STRATEGICKÝ PORTÁL SYSTÉMU PROCESNÍHO ŘÍZENÍ

STRATEGIC PORTAL OF PROCESS MANAGEMENT SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN MADĚRA

VEDOUCÍ PRÁCE
SUPERVISOR

doc. RNDr. JITKA KRESLÍKOVÁ, CSc.

BRNO 2011

Abstrakt

Univerzální rozšiřitelný webový portál především pro Metastorm Business Process Management System postavený na technologiích Microsoft ASP.Net, Microsoft MVC 2 a komunikující přes webové služby Microsoft WCF.

Abstract

Universal extensible web portal for Metastorm Business Process Management System based on Microsoft technologies: ASP.Net and MVC 2 for portal and WCF web services for communication.

Klíčová slova

Webový portál, Procesní řízení, ASP.Net, MVC, WCF

Keywords

Web portal, Process management, ASP.Net, MVC, WCF

Strategický portál systému procesního řízení

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. Jitky Kreslíkové.
Další informace mi poskytli Tomáš Sotoniak (brněnský ředitel firmy REENGINE CZ a.s.),
Martin Tlolka (vedoucí vývoje firmy REENGINE CZ a.s.) a Jaroslav Malík (procesní architekt firmy REENGINE CZ a.s.).
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Maděra
30.4.2011

Poděkování

Chtěl bych poděkovat vedoucí práce, doc. Kreslíkové, a zaměstnancům firmy REENGINE CZ a.s., jmenovitě Tomáši Sotoniakovi, Martinu Tlolkovi a Jaroslavu Malíkovi za poskytnuté informace, materiály a odborné rady.

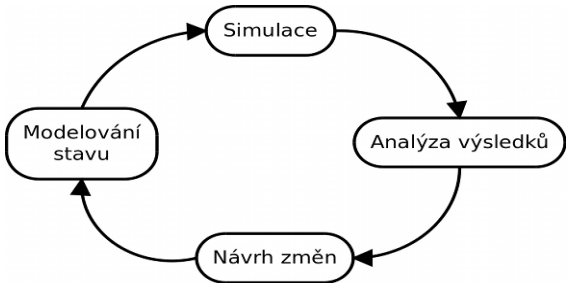
Obsah

1 Úvod

Cílem této bakalářské práce je vznik obecného a znovupoužitelného webového portálu komunikujícího s BPM (Business Process Management) softwarem Metastorm. Tento webový portál (oficiální marketingový název je Strategický portál) by měl být následně používán v zakázkách firmy REENGINE CZ a.s. (dále jen REENGINE). Nejprve bude v této práci definováno, co to vlastně BPM znamená a jaký má pro pracovní tým přínos a dále proč vzniká potřeba vytvářet webové portály pro firemní řešení. Budu se zaobírat zvolenými technologiemi a shrnu jejich klady a zápory. Další významná část práce bude samotný návrh a implementace vybraných částí výsledného Strategického portálu s použitím zvolených technologií. Na závěr bude provedena sumarizace práce a návrh rozšíření. Pasáže, u kterých to bude vhodné, se budu snažit doplnit o názorné obrázky, diagramy a snímky obrazovky. Od případného čtenáře práce je očekávána základní orientace v oblasti návrhu software, objektově orientovaného programování, webových služeb a webových aplikací. V této úvodní kapitole je rozebíráno několik pojmů, které jsou důležité pro tuto práci. Informačním zdrojem pro tuto kapitolu jsou konzultace [10], [11], [12].

1.1 BPM?

BPM je akronym pro Business Process Management, česky procesní řízení. Jedná se o přístup, který se snaží zlepšit a zefektivnit podnikové procesy (nejen výrobní, mohou být jakékoliv) a to za použití počítačové simulace a analýzy výsledků. Běžný životní proces BPM spočívá ve vymodelování stávajících podnikových procesů, jejich simulaci, analýzy výsledků simulace, návrhu změn a jejich simulace, opětovné analýzy a následně případné implementace navrhovaných změn (názorné schema viz Obrázek 1.1). V základu se tedy jedná o *restrukturalizaci fungování podniku na základě výsledků simulace*.



Obrázek 1.1: Cyklus BPM

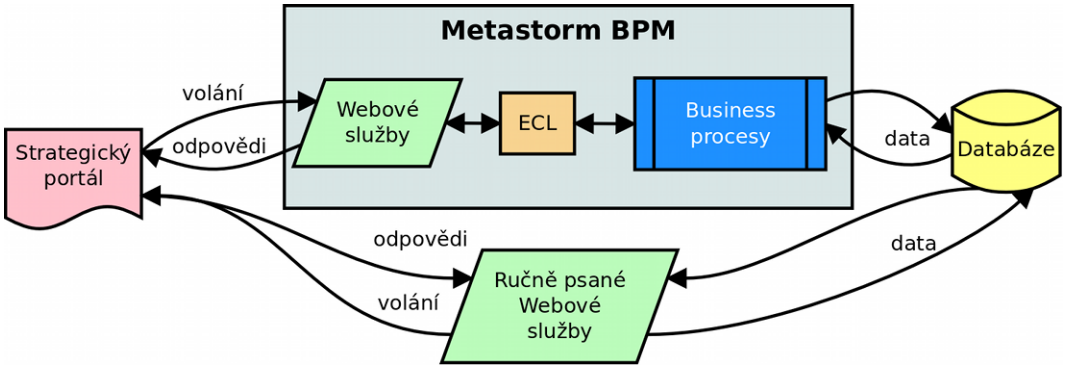
Současné vyspělé softwarové vybavení dokáže mnohem více, než jen simulaci. Například v REENGINE používaný Metastorm BPM software dokáže podnikové procesy nejenom modelovat a simulovat, ale také automatizovat (jedná se o nástroj typu BPMS – BPM System), čehož je využíváno při vytváření podnikových systémů. Metastorm BPM obstará většinu výpočetní logiky nad daty, která jsou uložena v databázi a pomocí jednoduchého a čistě pracovního webového rozhraní je zobrazí, případně umožní jejich ruční manipulaci, pokud taková manipulace je součástí nějakého podnikového procesu. Díky této vlastnosti je možno poměrně rychle a hlavně jednoduchou deklarativní cestou vytvořit např. komplexní systém pro řízení a prodej v multi-levelové

marketingové společnosti, často se opakující (administrační) či finančně významné procesy, tracking procesů, komunikaci mezi více systémy nebo distribuovaný systém testování (např. pro autoškoly či úřady práce). Viz Obrázek 1.2 .

1.2 Strategický portál a BPM?

Bohužel jednoduché a čistě pracovní webové rozhraní, které Metastorm BPM implicitně poskytuje, je v některých případech nedostačující. Jsou to především ty případy, kdy se s výsledným softwarovým řešením setkávají nejen interní zaměstnanci firmy (kteří mají raději jednotu, jednoduchost a přímočarost), ale i zákazníci firem (externí uživatelé). Proto se v takových případech, na přání zákazníků REENGINE, vytváří software, který ponechává většinu výpočetní logiky v Metastorm BPM, ale výsledná data zobrazuje pro zákazníky přijatelným způsobem (vztah portálu a Metastorm BPM jsem zachytil na Obrázek 1.2). Navíc je tímto řešena i platformní závislost (software je možno použít z více operačních systémů (a mobilních telefonů), Metastorm BPM je možno použít pouze z Microsoft Windows) a distribuci (na každém počítači, na kterém běží server Metastorm BPM, musí být Metastorm BPM nainstalovaný a jeho licence není levnou záležitostí).

Vzhledem k tomu, že pokud si situace vyžádá vyřešení výše zmíněných nedostatků Metastorm BPM, znamená to, že řešení firmy REENGINE využívá mnoho lidí a v jistých případech se může jednat i o řády statisíců. Proto, aby administrace řešení byla jednodušší, zvolili v REENGINE technologii webových portálů. V takovém případě lze velmi snadno provádět aktualizace, které lze rychle předávat zákazníkům, navíc není třeba se starat o balíčkování softwaru a jeho distribuci ke koncovým zákazníkům – stačí „jen“ udržovat v provozu a aktualizovat produkční servery (což by, kvůli samotnému Metastorm BPM, bylo stejně potřeba). K rozhodnutí pro webové aplikace vedla i možnost upravovat jejich vzhled pomocí kaskádových stylů, které jsou pro definici vzhledu velmi mocným nástrojem, což je u většiny grafických knihoven pro standardní „těžké“ aplikace nevídané[1]. Uživatelé na takovémto přístupu oceňují snadnost, se kterou mohou řešení používat – nemusí nic instalovat, stačí jim k tomu webový prohlížeč a zákazníci jsou spokojeni, neboť mohou komunikační bráně se svými zákazníky dát slušivý a osobitý vzhled[2] (a v dnešní době bohužel prodává právě vzhled, design, přístupnost a dobrý marketing).



Obrázek 1.2: Vztah Strategického portálu a Metastorm BPM

1 Ačkoliv mám přehled jen o těch známějších, rád bych poznamenal, že např. pomocí QSS (Qt Style Sheet) lze upravovat styly grafického rozhraní vytvořeného za pomoci C++ knihovny Qt. Nicméně je to spíše vzácné. Většinou je potřeba vyvinout mnohem větší úsilí (např. Swing pro Javu) a jinde to moc možné ani není (Microsoft Foundation Class).

2 Jeden z hlavních požadavků na Strategický portál (z případové studie): *„Prostředí pro práci je třeba uživatelsky zpříjemnit a zasadit je do 'firemní grafiky'“* [2].

Oficiální (a marketingový) název pro diskutovaný webový portál, který integruje Metastorm BPM systém do konkrétních webových aplikací vytvářených na míru zákazníkům, je „Strategický portál“. Více informací o Strategickém portálu je možno najít v jeho případové studii (viz Literatura: [2])

1.3 SCRUM

Pro správné vytvoření modelů podnikových procesů, bez kterého by byl celý systém k ničemu, je nutná dobrá komunikace pracovníků REENGINE se zákazníkem. Komunikace se zákazníkem probíhá zpravidla průběžně, ne jen na začátku vývoje, což zajišťuje lepší pochopení dané problematiky ze strany REENGINE a není nutno se opírat jen o formální specifikace „vydolované“ ze zákazníka (který jim stejně nebude rozumět). Tento přístup vyžaduje i jiný model řízení projektů, než je klasický vodopád nebo spirála. V REENGINE mají vyzkoušeno, že z agilních metodologií pro vývoj software v tomto případě nejlépe funguje model SCRUM a posléze byl tento model adaptován (nejen) na všechny projekty a podprojekty včetně vývoje webových portálů. Díky tomu má REENGINE vždy jistotu, že vyvíjí přesně to, co zákazník chce. Zákazník má, díky tomu, že vidí a je mu vysvětlován význam mezivýsledků, více času si uvědomit, jak přesně má vypadat výsledek, za který je ochoten dát své peníze a zda-li se vývoj ubírá správným směrem.

1.3.1 Popis metody SCRUM

Zamýšlení se nad agilními metodologiemi vývoje software je mimo rozsah této práce, proto zde uvedu pouze stručnou charakteristiku této metody.

Na začátku vývoje softwaru je třeba se zákazníkem zformulovat popis výsledku. Tento popis je psán co nejjednodušeji a je prost formalismů, kterým by zákazník nerozuměl. Je důležité udržet tento popis srozumitelný pro obě strany. Po definici výsledku je provedena dekompozice problému a jeho rozčlenění na jednotlivé vývojové běhy (sprint, milestone) ideálně tak, aby tyto běhy trvaly 10 pracovních dnů. V rámci každého běhu jsou definovány jednotlivé úkoly (task, ticket) a každý úkol by měl trvat 1 – 2 MD (man-day). Tyto úkoly se rozdělí mezi jednotlivé pracovníky. Na konci každého běhu je zákazníkovi předveden mezivýsledek („klikatelný“ fungující program), na jehož základě je rozhodnuto o dalším vývoji, zda-li má jít podle plánu dalšího běhu, či se má něco změnit nebo celý běh zahodit. Následně je další postup zákazníkem odsouhlasen.

1.3.2 SCRUM vs. klasické modely vývoje SW

V případě klasického modelu vývoje softwaru je na začátku vytvořena formální specifikace, kterou zákazník odsouhlasí, avšak problém je, že jí dostatečně nemusí rozumět, případně může některé pojmy interpretovat jinak, než analytik, který specifikaci zpracovával. Po několik měsíců probíhá vývoj za zavřenými dveřmi a zákazník má pocit, že se „nic neděje“ a nemá možnost vývoj svého softwaru ovlivnit. Až je mu nakonec s velkou pompou předveden výsledný produkt, může vzniknout situace typu „ale tohle jsem přece nechtěl“.

Takovou situaci si REENGINE nemůže dovolit, neboť je závislý na přesném modelování podnikových procesů v podniku zákazníka. Díky vývojovému modelu SCRUM mají zákazníci kontrolu nad vývojem svého software a tím pádem mohou ihned říci „ale takhle to není“ nebo „takhle

to nechci“. Toto má výhodu v tom, že je možno chybu opravit ihned a tím pádem na konci vývoje odevzdat produkt, který je zákazníkem chápán jako kvalitní (splňuje jeho požadavky, ať už jsou jakékoliv)[3]. Navíc obecně platí, že čím dříve se chyba objeví, tím rychlejší a levnější je její oprava. Stejně tak pokud je zákazník více „vtažen“ do vývoje, ví, koho a za co platí. Zpravidla tedy nemá problém za vývoj zaplatit (v ideálním případě probíhají platby průběžně).

1.4 Unit test

Pojem Unit test (občas do češtiny překládán jako „jednotkový test“) je v poslední době jedním z oblíbených „šumných slov“ (*buzzword*). Jednotkové testování je dnes „v kurzu“ a mnozí manažeři softwarových projektů o něm dost často mluví, ovšem ne vždy je použito v praxi. Důvod je většinou ten, že vytváření testovacího kódu nepřináší žádnou okamžitou viditelnou hodnotu, žádné nové tlačítko, na které by si mohl uživatel kliknout. Naopak, tento kód zůstává před uživateli skryt a při běhu aplikace uživatelé nijak nepoznají, zda-li k projektu byly vytvořeny testy či nikoliv.

Vytváření testů se projeví především v životní fázi údržby softwarového produktu. Každou změnou, která se do výsledného produktu provádí, je potencionálním zdrojem dalších chyb. Jednotkové testy napomáhají odhalovat takováto místa, neboť pokud programátor svým zásahem „vyrobí“ chybu (která je testem podchycena), ihned se o tom dozví a může zjednat nápravu.

1.4.1 Definice

Definovat by se Unit test mohl jako krátký a jednoduchý program (často vytvořen s použitím aplikačních rámců typu SUnit (Smalltalk, Scala[4]), JUnit (Java) či NUnit (Microsoft .Net)), který má za úkol automatizovat testování ucelené části programu. Ideální stav je takový, že jeden modul s jednotkovými testy testuje jeden objekt testovaného programu (jednu metodu v objektu programu testuje několik testovacích případů – minimálně očekávaná data, okrajová data, chybová data) a pokryje veškeré jeho chování. Tomuto ideálu, stejně jako jakémukoliv jinému, je velice těžké se přiblížit. U objektů modelové vrstvy (tedy modely dat, datové či aplikační logiky) není vytvoření jednotkových testů moc náročné, ale pokud máme grafickou aplikaci, je automatizace testování grafického rozhraní (ať už klasického (*thick application*) či webového (*thin application*)) netriviální záležitost a vývojové týmy mají tendenci spoléhat spíše na ruční „proklikání“.

Microsoft Visual Studio v nové (z roku 2010) verzi umožňuje relativně jednoduchou tvorbu a spouštění jednotkových testů, což usnadňuje jejich tvorbu. Pro vybrané části této bakalářské práce (tam, kde to bude mít smysl) budou jednotkové testy vytvořeny.

Plusy a přínosy jednotkových testů jsou především v automatizaci testování programu. Testování je vždy repetitivní a nezáživná činnost, kterou mají tendenci vývojáři odbýt nebo úplně vynechat (navíc u mnoha firem, kde projektový manažer nemá moc velký vhled do problematiky vývoje softwaru, není možno do pracovního výkazu napsat „Vývoj knihovny – 4 hodiny; Testování a ladění – 2 hodiny“). Samozřejmě i v programování platí Murphyho zákony a v každém kódu je alespoň jedna chyba. Pokud se vynechá testování, obvykle se chyby projeví při předávání projektu

3 „*Pokud bude cílem vyvinout nespolehlivý software, pak čím bude tento software méně spolehlivý, tím bude kvalitnější. “* [1], str. 13

4 Knihovny mají stejný název, ale vyvíjí je jiní lidé nezávisle na sobě. Pravděpodobně se bude jednat o náhodnou shodu jmen.

a demonstraci zákazníkovi. Rozsah společenského faux pas, kdy jeden chybějící středník v kódu zapříčiní jeho nefunkčnost, je nedozírný. Stejně tak se během vývoje program mění a vyvíjí a může se stát, že změna jedné části způsobí nefunkčnost jiné, již hotové a odzkoušené části. Toto riziko je vysoké zvláště jedná-li se o rozsáhlejší projekt složený z několika ne zcela dobře oddělených částí. Bohužel pak nejsou raritou otázky vedoucího vývoje typu „Ale tohle jsme změnili už před více než 3 měsíci. Jak se to mohlo stát, že na to ještě nikdo nepřišel? To na to tlačítko ty 3 měsíce nikdo neklikl?“.

Po každé změně programu (a v určitých časových intervalech preventivně) by mělo probíhat testování programu. To by měl dělat buď nezávislý tester (což je poměrně nákladná záležitost) nebo by mělo probíhat automaticky (levnější záležitost, ale často je nutné ještě ruční testování některých případů). Jednotkové testy jsou výhodné i v pozdějších fázích životního cyklu softwaru, neboť pokud se dodatečně dodělává do již hotového a „nasazeného“ programu nějaký rys nebo se v něm dělá změna, zpravidla si již nikdo přesně nepamatuje, co by mohl svým krokem znefunkčnit. Argument, že při dobré vnitřní architektuře programu se dodatečnou změnou nic nemůže pokazit, je zcela zcestný v jakémkoliv prostředí.

Nevýhody tvorby unit testů jsou především v tom, že zpomalí vývoj. V mnoha případech, zvlášť, je-li potřeba vývoj dokončit rychle (např. kvůli konkurenční výhodě nebo kvůli tomu, že odhad časové náročnosti projektu byl příliš optimistický), není čas ani na psaní dokumentace, ani na tvorbu jednotkových testů. V extrémních případech se dokonce vynechává i testování a pilotní provoz [3]. A dokonce i v případech, kdy je na projekt času relativně dost (dobré časové odhady, dostatečně dlouhá doba na testování a časová rezerva na konci vývojového plánu), může být tvorba jednotkových testů vnímána negativně. Před zákazníkem/manažerem jsou vidět pouze aktuální náklady a může si myslet, že se jedná o zbytečnou práci, neboť vytvoření jednotkového testu do programu nepřidá žádný nový viditelný rys. Dost těžko se vysvětluje, že jednotkové testy budou mít přínos především v budoucnu, při údržbě a dalším rozšiřování programu a že se čas investovaný do jejich vytvoření mnohonásobně vrátí při hledání chyb.

1.4.2 Test Driven Development

Testy řízený vývoj (*test driven development*) je již relativně neznámý pojem spadající do kategorie XP (*eXtreme programming, extrémní programování*). Je založen na tom, že nejprve vznikne jednotkový test, ve kterém se definují vstupy, požadované rozhraní objektu a jeho výstupy. Jedná se o zajímavý postup, který zaručuje, že rozhraní objektů bude opravdu pro danou problematiku pohodlně využitelné (při klasickém postupu vývoje je rozhraní objektu zpravidla definováno při/před jeho vytvořením a návrhář má tendenci sklouzávat k vytvoření takového rozhraní objektu, které se snadno implementuje, ale nemyslí na logičnost a pohodlnost jeho používání). Ačkoliv se může testy řízený vývoj zdát na první pohled nesmyslný a nelogický, můžeme zde nalézt paralelu s definicí případů užití (*use case*), z nichž vychází plánování vývoje programu. Zde také nejprve vzniká testovací případ (*test case*) a teprve na jeho základě se navrhuje daná objektová struktura programu. Navíc při této metodice uvidí vývojář svůj výtvor z pohledu toho, kdo jej bude využívat a zamyslí se nad pohodlností použití jeho rozhraní.

Tato metoda vývoje programů je rozšířená především mezi uživateli jazyka Smalltalk. Jeho implementace (minimálně ty známější) poskytují dynamický debugger, ve kterém je možno přímo tvořit programový kód. Pokud se při běhu programu ve Smalltalku stane, že nějaký objekt nerozumí

dané zprávě, zpravidla je programátor vyzván k jeho doplnění a následně může program pokračovat, na rozdíl od většiny ostatních používaných prostředí pro vývoj softwaru, kdy program jednoduše skončí chybou. A toto je případ i prostředí .Net. Bohužel také výpisy stavů objektů jsou v Microsoft Visual Studio nekompletní a není možné v okamžiku přerušení programu volat funkce objektu, což tuto formu vývoje programu dále ztěžuje.

Přestože je tento pojem zmiňován jako doporučení v některých publikacích k ASP.Net (např. v [4]), musím učinit závěr, že toto prostředí je k tomuto stylu vývoje značně nepohodlné.

1.5 Komunikace s Metastorm BPM

Aby bylo možno webový portál napojit na Metastorm BPM, je v REENGINE využívána knihovna Metastorm ECL (Enterprise Component Library). Díky této knihovně, přímo poskytované Metastormem, je možno vytvořit v prostředí Java nebo .Net programy komunikující s Metastorm BPM, případně rozšiřující jeho funkčnost. Stejně tak Metastorm BPM umí sám vytvořit webové služby (konkrétně komponenta „Metastorm Process Activator“) pro přístup a manipulaci s definovanými automatizovanými procesy. Popis těchto knihoven, ač se jedná o zajímavé téma, je bohužel nad rámec této práce. Pro účely práce postačí vědět, že stejně jako stávající portálová řešení i Strategický portál nově vytvářený v rámci práce bude komunikovat s Metastorm BPM pomocí technologie webových služeb, z nichž některé budou přímo generované z Metastorm BPM a některé psané ručně (schema vztahu portálu a Metastorm BPM jsem ilustroval na Obrázek 1.2). Služby vygenerované z Metastorm BPM vyžadují Microsoft IIS (Internet Information Service – (nejen) webový server od Microsoft, integrovaný do Microsoft Windows) a běhové prostředí Microsoft .Net . Pro to, aby oba druhy služeb mohly běžet na stejném serveru a aby se na server nemuselo instalovat zbytečně mnoho věcí navíc, jsou i ručně psané webové služby postavené na technologii Microsoft .Net (pro jejich vývoj je použito jazyka C#). Komunikace s webovými službami probíhá pomocí protokolu SOAP.

1.6 Volba platformy

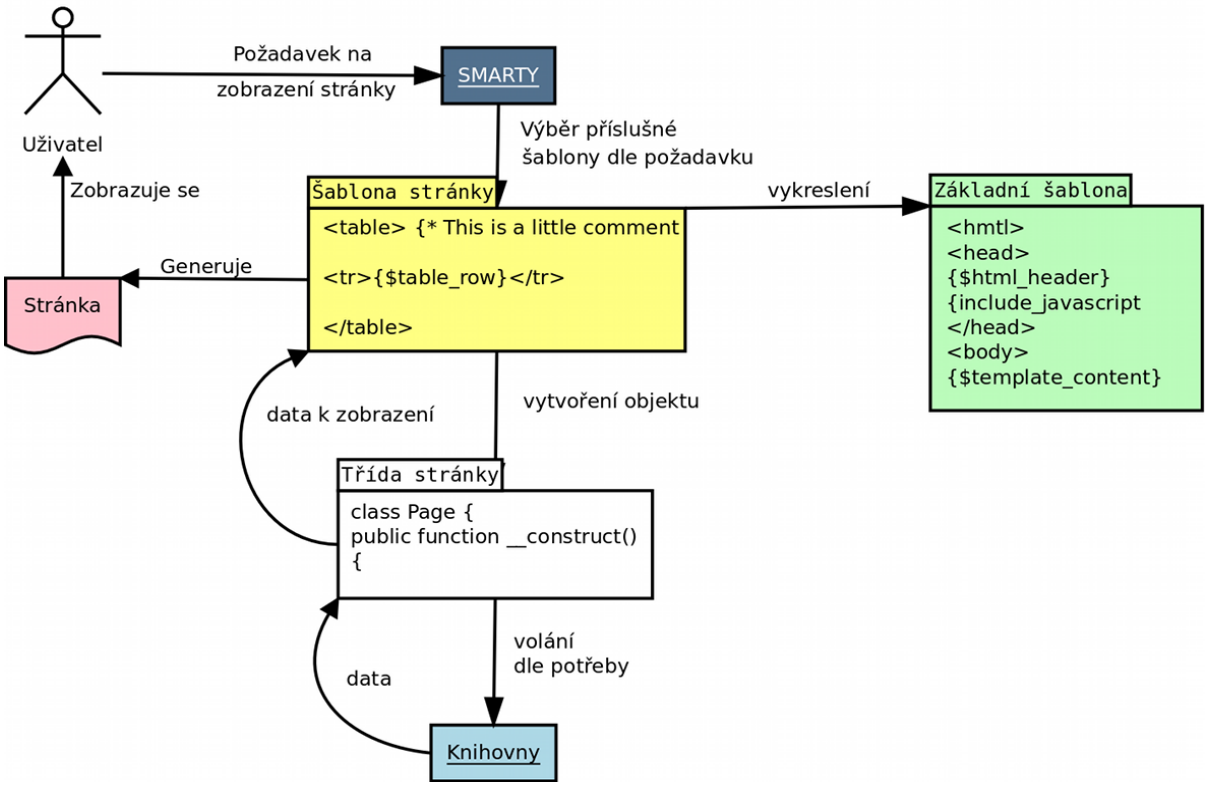
Velmi rozsáhlým tématem při vývoji software je volba platformy, knihoven a programovacího jazyka. Tyto tři věci nelze oddělit, neboť spolu více či méně souvisí. Volbu je nutno provést opravdu pečlivě, neboť špatně zvolená platforma (knihovny nebo programovací jazyk) může v jisté fázi vývoje celý projekt velmi prodražit, případně další vývoj znemožnit. Naopak dobře zvolená platforma umožní dodat kvalitní a udržovatelné řešení rychle a levně.

1.6.1 Historie vývoje Strategického portálu a současný stav

V době, kdy poprvé vznikla potřeba vytvořit webový portál, se jednalo o požadavek zákazníka, který byl více-méně „mimochodem“. Mělo se jednat o projekt, ve kterém měly být postupně spouštěny jen jednotlivé části a měl být do 4 měsíců ukončen, přičemž výkonové požadavky nebyly příliš vysoké. Vzhledem k tomu, že všichni z tehdejšího brněnského týmu REENGINE, který měl tuto zakázku realizovat, znali (pro vývoj webových aplikací) skriptovací jazyk PHP, volba padla právě na něj. (Ve hře byly ještě další možnosti, které by byly z nynějšího hlediska hodnoceny lépe, avšak v kontextu

doby tomu tak nebylo – především díky tomu, že ne všichni členové vývojového týmu měli zkušenosti s Javou (a frameworkem Spring) či Pythonem nebo ASP.Net a fungující portál bylo třeba dodat co nejdříve; „jednotícím“ jazykem, který by znali všichni, bylo pouze PHP.)

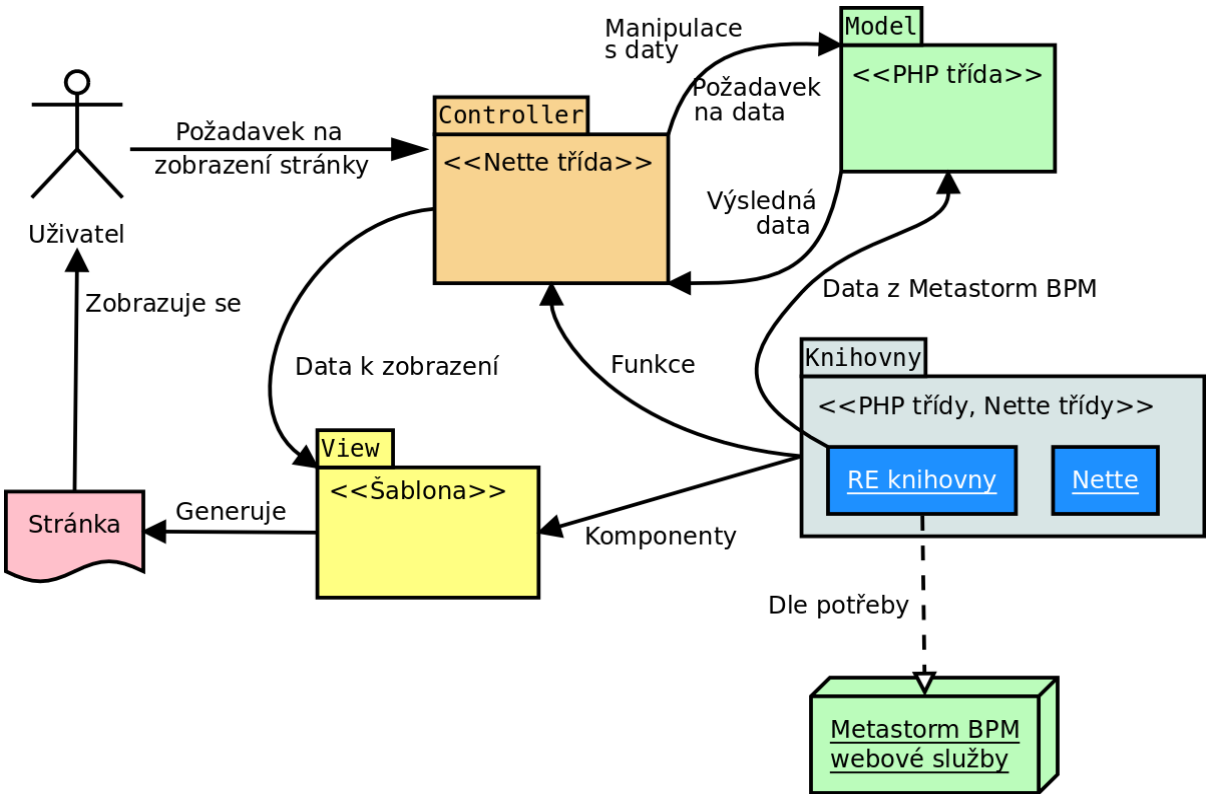
První portál byl tedy napsán v čistém PHP a data, která byla potřeba ukládat na straně portálu (a nebyly součástí podnikových procesů, tudíž nebylo nutné, aby se o ně staral Metastorm BPM), byla ukládána v databázi MySQL (např. místní poznámky, kontextová nápověda, chybová hlášení …), v textových souborech (lokalizace) a XML souborech (nastavení). Vzhledem k obecné „pravdě“, že každý PHP programátor, pokud nechce používat cizí aplikační rámce (*framework*), si vytvoří framework vlastní, byl vytvořen a postupně doplňován i soubor knihoven a samotný portál fungoval na principu řízení šablonami (*template driven*). Tedy vznikl-li požadavek na zobrazení stránky, byla vytvořena instance třídy příslušné k šabloně dané stránky. Šablona systému SMARTY definovala celkový vzhled a rozmístění prvků na stránce a PHP objekt zajišťoval volání funkcí z knihoven dle potřeby, připravoval a posílal data k zobrazení šabloně. Funkce pro získávání a transformaci dat, ať už z databáze nebo z webových služeb Metastorm BPM, byly obsaženy v knihovnách. Základní schema této funkčnosti je na obrázku níže (viz Obrázek 1.3). Takový systém byl jednoduchý a jeho vývoj byl velmi rychlý a zároveň díky jeho návrhu jej bylo možno snadno a rychle rozšiřovat o nové knihovny.



Obrázek 1.3: schema původního portálového jádra

U zakázky, kvůli které portál vznikl, měl zákazník ještě další požadavky. Díky tomu se portál rozrůstal a postupným přidáváním stále nových požadavků ze strany zákazníka se z něj stal projekt na 2 roky. Bylo jasné, že takto navržené jádro portálu pro budoucí projekty podobného rozsahu stačit nemůže – jádro bylo navrhováno pro malé projekty, které se „vytvoří a ukončí“, nikoliv pro portály, které má používat 6.000 lidí každý den a do kterých je postupně přidělávána další a další funkčnost.

Vznikl požadavek na tvorbu nového Strategického portálu, přičemž tentokrát byl v době návrhu kladen důraz na znovupoužitelnost a rozšiřitelnost, méně již na rychlost vývoje jádra. Při analýze požadavků a možností realizace takového návrhu padla volba na objektový návrhový vzor MVC (Model-View-Controller)[5]. Tento návrhový vzor rozděluje program do tří vrstev: vrstva modelu je datový model a stará se o získávání a manipulaci s daty. Vrstva view data zobrazuje a controller se stará o samotné řízení běhu aplikace (reakce na podněty od uživatele). Každá tato vrstva je oddělená od ostatních a má jasně dáno, co má dělat. Díky tomu pokud k projektu přijde vývojový pracovník, který na něm do té doby nepracoval, ví, co se kde děje a co má kde změnit či doplnit. Zvládnutí vývoje produktů dle tohoto vzoru vyžaduje disciplínu a v prvních fázích vývoje může u pracovníků, kteří se s ním dosud nesetkali, vyvolávat dezorientaci. Avšak přenesení se přes prvotní obtíže se následně mnohokrát vyplatí při údržbě či dalším vývoji. Po prozkoumání „nabídky“ aplikačních rámců umožňujících návrh systému dle návrhového vzoru MVC padla volba na český Nette.



Obrázek 1.4: schema portálového jádra postaveného na Nette

Díky Nette velmi rychle vznikla kostra portálu, která má v sobě mnoho prvků architektury MVC (přesněji se jedná o návrhový vzor MVP[6]) a umožňuje používání již hotových zobrazovacích komponent, často s vlastním chováním. Výslednou stránku je možné poskládat z těchto komponent

5 Návrhový vzor MVC byl v roce 1979 popsán norským profesorem Trygvem Reenskaugem z University v Oslu původně pro grafické uživatelské rozhraní a byl poprvé použit v jazyku Smalltalk-80, jedním z průkopníků objektově orientovaného programování. Tento čistě objektový, třídně orientovaný a dynamicky typovaný programovací jazyk je (dle mého mínění) velmi zajímavý i dnes a mnohé jeho dobré vlastnosti se začínají objevovat i v „main-streamových“ jazycích, jako je C#.
Viz: http://en.wikipedia.org/wiki/Model-view-controller
http://en.wikipedia.org/wiki/Trygve_Reenskaug
http://pharo-project.org

6 Model-View-Presenter – návrhový vzor podobný MVC

a docílit tak rychlejšího vývoje. Navíc tyto komponenty nemusí být přímo součástí zdrojových kódů každého portálu, ale mohou být mezi knihovnami, což usnadňuje jejich přenos mezi projekty. Zjednodušené grafické schema tohoto portálu je na obrázku výše (Obrázek 1.4).

Veškerá komunikace s Metastorm BPM je opět ukryta v knihovnách portálového jádra. Díky tomu, že v Nette je možno striktně oddělit knihovny od kódu samotného portálu, je zjednodušena distribuce aktualizací mezi jednotlivými projekty. Přísným dodržováním návrhového vzoru se také zvýšila efektivita práce a snížilo riziko vzniku některých chyb v portálu, stejně jako „temných zákoutí“, které „tak nějak asi“ fungují a nikdo s nimi nechce mít nic společného. Další významnou změnou v této vývojové fázi byla i změna ve vrstvě starající se o ukládání dat portálu. Byla přidána další vrstva, převzatá ze Zend framework, která se stará o přístupy do databáze a generování SQL dotazů. Tento přístup, ačkoliv k původně zamýšlenému dokonalému objektově-relačnímu mapování má daleko, umožnil odstranění SQL dotazů z kódu knihoven pro přístup k datům. Díky tomu je již možno používat různé databázové systémy pro různé portály. Vzhledem k tomu, že Metastorm BPM ukládá svá data do MSSQL databáze, byl zvolen tento systém jako výchozí. Dále se co nejvíce dat potřebných pro běh portálu (a bezvýznamných pro funkční logiku, tedy pro vrstvu v tomto případě představovanou Metastorm BPM) přesunulo do databáze, např. lokalizace.

1.6.2 Platforma pro tuto práci

Nyní je pověstným trnem v oku jazyk PHP a jeho interpret. Jedná se o jazyk, se kterým se díky Nette pracuje sice o mnoho příjemněji, přesto ne dostatečně příjemně[7]. Absence pořádných nástrojů jako profiler a debugger jsou mnohdy překážkou ve vývoji a přestože NetBeans nebo Eclipse jsou velmi dobrými vývojovými prostředími, nad staticky typovanými jazyky zde PHP ztrácí. Dalšími drobnými problémy PHP je nekonzistentnost základní knihovny a i syntaxe a sémantiky jazyka jako takového, na čemž se značnou měrou podepsal jeho vývoj a původní určení. V neposlední řadě je jistou překážkou poměrně nízký výkon a relativně vysoká paměťová náročnost interpretu PHP[8], stejně jako skutečnost, že interpret PHP musí být na produkčním serveru nainstalovaný a stává se tak další věcí, o kterou je nutné se starat.

Nejen z těchto důvodů si členové brněnského vývojového týmu často pohrávali s myšlenkou mít i portál postavený na technologii Microsoft.Net . Změna situace v týmu pomohla k odsouhlasení vývoje portálového jádra postaveného na technologii Mcrosoft.Net s primárním programovacím jazykem C#, čímž se výše zmíněné nedostatky vyřeší. Velkou výhodou platformy .Net je také možnost kombinovat více programovacích jazyků – z tohoto úhlu pohledu je zajímavým jazykem multiparadigmový F#, ve kterém je možno kombinovat objektově-orientovaný přístup s funkcionálním. Funkcionální paradigma je výhodnější pro zpracování dat díky vyšší přehlednosti kódu (např. uzávěry (*closure*) či operátor „roura“, angl. „pipe“) a vrozené jednoduchosti při paralelním zpracování. Databázová technologie zůstane MSSQL, ke které Microsoft Visual Studio poskytuje nástroj pro mapování relačních databázových entit na objekty programovacího jazyka (ORM, *object-relational mapping*). Zda-li využít Microsoftem poskytovaného aplikačního rámce

7 Navíc Nette jako takové je samo o sobě relativně pomalé – pomocí profileru jsme na jedné z konzultací zjistili, že to není jen subjektivní pocit

8 V době dokončování této bakalářské práce byl ve firmě REENGINE úspěšně vyzkoušen Zend server (http://www.zend.com/en/products/server/) – webový aplikační server, který při prvním požadavku na soubor tento soubor přeloží do mezikódu a při dalších požadavcích pracuje již přímo s mezikódem [více viz oficiální dokumentace – odkaz výše]. V případě portálů firmy REENGINE se průměrně snižují paměťové nároky na třetinu a rychlost se zvyšuje průměrně o polovinu. Přes to ale toto zrychlení není v některých specifických případech dostatečné.

Microsoft MVC či se držet „standardního“ Web.Forms, je zatím otázkou a bude rozebíráno dále v této práci v kapitole o návrhu Strategického portálu. Další poměrně silnou stránkou této platformy je jednotkové testování, ovšem zatím není rozhodnuto, zda-li vůbec a pro které části portálu testy vytvářet (konečné rozhodnutí bude pravděpodobně na mně). Z hlediska návrhu by měl nový portál co nejvíce reflektovat stávající portál založený na technologii PHP, aby se vývojovým pracovníkům usnadnil přechod mezi nimi. Vzhledem k rozsáhlosti současného řešení je také nutný souběh obou technologií, neboť vývoj všech částí je časově náročný a stávající projekty se nebudou na nový portál předělávat, pokud to nebude nezbytně nutné.

Další podstatnou výhodou architektury Microsoft.Net je možnost využívat Metastorm ECL knihovnu přímo a ne přes webové služby. Zda-li a v jaké míře bude tato možnost využita, zatím není uzavřenou kapitolou. Jako nejpravděpodobnější možnost se zatím jeví kombinace obou přístupů, neboť v době psaní této práce již nemá smysl vytvářet zcela nový produkt pro Metastorm BPM verze 7, ale nová verze Metastorm BPM 9 (číslovka 8 byla ve verzování produktů Metastorm přeskočena) stále ještě nemá ECL knihovnu zdokumentovanou a připravenou k použití (obsahuje stále ještě množství chyb).

Vytvoření Strategického portálu na této technologii bude mým úkolem v rámci této práce.

1.7 Organizace práce

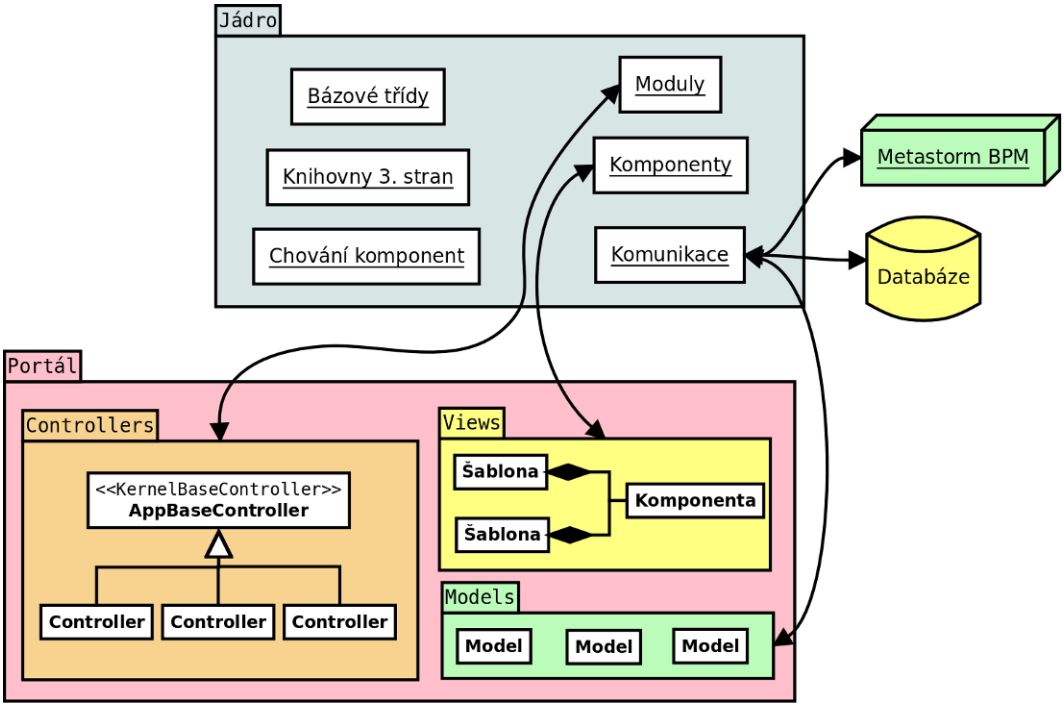
Vývoj nového portálového jádra bude probíhat v několika oddělených fázích, přičemž první dvě budou předmětem této práce. Cílem první fáze bude vytvořit minimalistické modulární jádro, které bude robustní, přehledné a snadno udržovatelné. Jak je možno vytušit z již napsaného, vývojové práce zde budou organizovány podle agilní metodologie vývoje softwaru jménem SCRUM. Následně bude probíhat vývoj rozšiřujících knihoven, z nichž ne všechny budou nezbytné pro vývoj všech projektů. V této fázi bude vývoj probíhat dle aktuálních potřeb. V závěru druhé fáze by měla být k dispozici většina knihoven, které jsou dostupné i v nynějším PHP jádru.

2 Návrh systému

Během návrhu je nejdříve nutno vyřešit otázku, zda-li či nikoliv využít Microsoft MVC framework či knihovny Web.Forms. Tato problematika bude rozebírána hned na začátku. Dále, jak již bylo naznačeno, bude vytváření systému probíhat ve dvou fázích. V této části vývoje je třeba konkrétněji rozdělit, co se ve které fázi bude tvořit a blíže je specifikovat.

2.1 Fáze 1 – minimalistické jádro

Aby mohl být projekt co nejdříve použit k vývoji projektů, je třeba co nejrychleji vybudovat stabilní a robustní jádro, které bude schopno samostatného fungování a které bude v pozdějších fázích vývoje rozšiřováno o nové moduly.



Obrázek 2.1: Rozdělení komponent mezi jádrem a portálem

Požadavky na nové jádro jsou v zásadě stejné jako na stávající. Architektura nemá výrazné nedostatky a není ji tedy třeba měnit. Jádro by mělo obsahovat obecné komponenty, které bude portál používat ve více situacích. Pokud se v rámci projektu vyskytne nějaký konkrétní problém, měl by se (pokud to situace dovolí) řešit na obecnější úrovni tak, aby se dané řešení mohlo následně znovu použít. Ideální případ by byl takový, aby jádro obsahovalo soubor komponent a konkrétní portál by je pouze poskládal dohromady. K tomuto ideálu bych se měl postupným vývojem přiblížit (současné řešení je takové, že většinu problémů lze v portálech řešit pouze použitím již připravených komponent, s minimálním úsilím a minimálním počtem napsaných řádků kódu). Jak je vidět z nákresu (viz Obrázek 2.1), v samotném portálu jsou pouze věci nezbytné pro jeho fungování nebo

specifické pro daný projekt. Většinu znovupoužitelných komponent, veškeré komunikační rozhraní či bázové třídy, je možno nalézt v jádře.

Dobrým výchozím bodem pro nové jádro by mohlo být splnění následujících bodů:

- Navigační nabídka;
- Lokální přihlašování, kontrola přístupu;
- Vytvoření knihovny komponent pro statické články[9] (část obsahu portálů bývá statická);
- Zprovoznění komunikace s databází (souvisí s předchozím bodem);
- Vytvoření modulu pro komunikaci přes webové služby;
- Lokalizace.

2.2 Fáze II – základní knihovny

Po vytvoření jádra a nezbytných knihoven nastoupí fáze, kdy bude třeba vytvořit sadu základních knihoven. Tyto sice nejsou nezbytné pro naplnění podstaty Strategického portálu, ale řeší věci, které se vyskytují téměř v každém projektu.

Fáze II by měla být ukončena splněním následujících bodů:

- Přihlašování do Metastorm BPM a kontrola přístupu na základě jeho rolí;
- Ajax, jQuery a jQueryUI (tuto funkčnost si pravděpodobně vyžádá již implementace článků, proto může být posunuta do fáze 1);
- Možnost zobrazovat a odesílat tabulky a formuláře došlé z Metastorm BPM.

V rámci této bakalářské práce není podmínkou dokončení implementace všech částí Strategického portálu, neboť počet hodin práce bude v řádu stovek. Nicméně projekt bude pokračovat i po obhajobě bakalářské práce a je v plánu firmy (i v rozpočtu) jej dokončit a používat.

9 Pokud je čtenář obeznámen se stávajícími knihovnami v jádře Strategického portálu, tak zde se nebude jednat o „články“ (knihovna Re.Articles), ale o „seskupené články“ (Re.ArticleGroup). Seskupené články rozšiřují možnosti původních článků o možnost zobrazit více článků na jedné stránce v rámci vlastního kontejneru, do které lze články přidávat, mazat a měnit jejich pořadí. Přenesení knihovny pro původní články není nutné, neboť skupina článků mající pouze jeden článek je přípustná.

2.3 MS MVC vs. Web.Forms

Než bude moci být zhotoven konkrétní návrh jádra portálu, je třeba vybrat vhodný framework pro jeho tvorbu. Pro zvolenou platformu, ASP.Net, existují dva, každý s odlišným přístupem. Kromě získávání teoretických informací (především z oficiálních zdrojů, tedy [5]) jsem provedl i některé praktické zkoušky těchto dvou aplikačních rámců. V závěru kapitoly popisuji rozhodnutí a jeho důvod.

2.3.1 Web.Forms

Tento framework byl dostupný pro ASP.Net jako první a částečně vychází z tradic původního ASP. Je zde velmi patrná snaha tvorbu interaktivních webových aplikací co nejvíce připodobnit vývoji desktop aplikací. Každý webový formulář je v grafickém editoru „poskládán“ z ovládacích prvků zhruba odpovídajících nabídce knihovny Windows.Forms a pro každý formulář je definován „code behind“, kód na pozadí. Na toto místo se píše obslužný kód pro různé události, které jsou vyvolány uživatelem u jednotlivých prvků formuláře. Aplikace je stavová, tedy v obslužném kódu jsou přístupné informace o stavu ovládacího prvku a je také možno stav prvku měnit. Taková změna se projeví ihned. Podobně se uchovávají stavy objektů – můžeme se spolehnout na to, že v programových objektech budou po znovu načtení stránky stejná data, jako před jejím načtením. [5]

Díky tomu, že formuláře jsou „poskládány z komponent“, je vývoj relativně rychlý – formuláře se navrhují graficky a není třeba psát ani řádek HTML kódu. Pokud nám nějaká komponenta chybí, lze ji vytvořit. Mnohé komponenty se umí vykreslit pro různé internetové prohlížeče různě, čímž řeší nekompatibilitu MS IE s ostatními prohlížeči. Toto chování ušetří mnoho práce, pokud funguje dobře. Pokud ne, stane se z takových komponent noční můra. Položkou v tomto výčtu, kterou bychom neměli opomenout, by měl být i fakt, že mnoho vývojářů či firem nabízí své hotové komponenty ostatním. Na internetu lze najít mnoho takových komponent, jak placených, tak k volnému použití. Bohužel jejich kvalita dosti kolísá a není-li předem dostupný zdroj kvalitních a přijatelně drahých komponent, může být rychlejším řešením si napsat svoji vlastní, než shánět použitelnou cizí.

Struktura programu používajícího Web.Forms není přesně dána, vývojářům je poskytnuta velká volnost. Toto opět může být výhodou i nevýhodou. Je zde umožněna značná volnost pro tvůrčí a inovativní přístupy k řešení, avšak nepromyšlené tvůrčí řešení se může stát pohromou, je-li potřeba následně hotový software rozvíjet a udržovat. Je sice možné nějakou strukturu definovat interně v rámci vývojového týmu, avšak bylo by nutno zajistit, aby se této struktury všichni striktně drželi a to i ti, kteří do vývoje budou vtaženi později. Na druhou stranu by nebyl problém se zbytečnými omezeními, na které je možné narazit u aplikačních rámců, které se striktně drží nějaké předem dané formální struktury a „ohýbat“ své řešení podle zvolené technologie (neboť každý projekt je jiný a má své specifické požadavky).

Web.Forms:

+ Velmi rychlý vývoj, formuláře se „naklikají“ (RAD);
+ Je stavové;
- Komponenty se občas nezobrazují dobře => občas těžce řešitelný problém;
- Tým musí být disciplinovaný, aby nevznikl paskvil;
? Nevnucuje pevně danou strukturu.

2.3.2 Microsoft MVC framework

Microsoft MVC framework je v mnohých ohledech opakem Web.Forms. Drží se návrhového vzoru Model-View-Controller [5] zhruba tak, jak byl (a je) používán v jazyce smalltalk. Ačkoliv se jedná o poměrně starý a konzervativní návrh, víme, že pro většinu grafických aplikací funguje. Tento návrhový vzor si sice vynucuje od vývojového týmu disciplínu, ale na druhou stranu i pokud vývojový pracovník přijde k vývoji v pozdějších fázích, „ví kam má sáhnout“ a velice rychle se do nového projektu zapracuje. Tímto je umožněno dynamičtější rozdělení pracovních sil mezi jednotlivé projekty v čase, což je při vývoje dle agilních metod velkou výhodou.

Naproti tomu vcelku velkou vrozenou vadou návrhového vzoru je vyšší množství tříd (de facto pro každou „maličkost“ vzniká třída) a vývojový pracovník, který na toto není připraven, se může snadno ztratit a nebude vědět, co se kde děje[10]. Naštěstí dobrá organizace kódu a dobré vývojové prostředí může riziko „ztracení se“ snížit[11]. Navíc současné jádro postavené na Nette framework má podobnou strukturu (odpovídá návrhovému vzoru MVP, který je MVC velmi podobný), což by pro členy našeho týmu znamenalo pohodlnější přechod na novou technologii.

Dalším významným bodem tohoto drobného srovnání by měl být fakt, že při použití MS MVC framework má programátor nad chováním aplikace a nad výsledným generovaným HTML kódem plnou kontrolu, avšak za cenu pomalejšího vývoje, neboť například formuláře je potřeba vytvářet ručně. Naštěstí většina formulářů či tabulek bude odeslána z Metastorm BPM a jejich zobrazování může být automatizováno (v současné době již je toto know-how k dispozici).

V neposlední řadě je třeba zmínit, že v Nette bylo možno zákonitosti MVP vzoru porušit, kdežto Microsoft MVC framework je navržený tak, aby to možné nebylo. Toto může a pravděpodobně bude působit problémy, pokud budeme chtít, stejně jako v současné době, mít v knihovnách zobrazitelné komponenty s vlastním chováním.

Microsoft MVC framework:

+ Konzervativní, lety prověřená struktura (MVC)
+ Vynucuje disciplínu od týmu => organizovaný předem daný pořádek
+ Podobné Nette => snazší přechod
+ Absolutní kontrola nad chováním aplikace
- Struktura je hodně svazující
- Pomalejší vývoj než u Web.Forms
- Sklony k „ravioli kódu“
? Snazší unit testing (není rozhodnuto, v jaké míře bude použit)

10 Tato forma kódu, která se vyznačuje velkým množstvím tříd, se pejorativně nazývá „ravioli code“ a je jistou obdobou špagetového kódu (tento pojem v poslední době proslul díky obecně nízké kvalitě amtérských programátorů dynamických webových stránek).
http://en.wikipedia.org/wiki/Spaghetti_code (definice špagetového, ravioli a lasagnového kódu)
http://www.garfield.com/comics/todayscomic.html

11 Z vlastních zkušeností musím říci, jedno z nejlepších vývojových prostředí, které jsem zatím potkal, bylo to, které je integrováno do dialektu smalltalku Pharo (Squeak). Bohužel smalltalkový image (vč. komerčních a drahých distribucí, např. VisualWorks či VA Smalltalk) je obvykle plný malých tříd a drobných funkcí (a špatné dokumentace). Naštěstí prostředí jazyka C# k tomuto jevu náchylné není.

2.3.3 Shrnutí a rozhodnutí

Po porovnání obou možných aplikačních rámců jsem se (po konzultacích) rozhodl pro MVC framework, který pro potřeby Strategického portálu vyšel v tomto srovnání lépe. Je to především z toho důvodu, že v projektech je pevná struktura, je zde možnost ovlivnit každý detail chování aplikace a je podobnější současnému řešení, což pro bude znamenat snazší přechod. A to i za cenu mírně pomalejšího vývoje (díky tomu, že tabulky a formuláře se většinou posílají z Metastormu BPM, není toto zpomalení natolik razantní) nebo nutnosti překonávat některé překážky (vykreslitelné komponenty v knihovnách).

Zvláště na poslední zmíněný aspekt by bylo ideální použít Web.Forms. Po prozkoumání možností použití obojího zároveň jsme se na konzultaci ve firmě rozhodli tyto dva rámce nemíchat (přestože to možné je), neboť z hlediska formálního návrhu by se nejednalo o čisté řešení a zbytečně by do projektu byla zanášena místa náchylná k chybám.

3 Implementace

Z předchozí kapitole vyplývá, že ta část implementace Strategického portálu, která je předmětem této práce, by měla být rozdělena do dvou fází. V první fázi by měla být implementována základní funkčnost a na jejím konci by měl být k dispozici již použitelný výsledek. Jelikož části webů, pro které je Strategický portál určen, bývají statické, je dobré nyní implementovat onu statickou část, nezávislou na Metastorm BPM.

3.1 Fáze 1

Před tím, než se začne s jakýmikoliv pracemi, je třeba si připravit prostředí pro vývoj. Dále je třeba zprovoznit alespoň minimální příklad (*minimal example*) a komunikaci s databází. Zapomínat by se nemělo ani na logování chyb.

Po těchto nezbytných krocích je možno se pustit do samotného vývoje. Po první fázi by měl portál umět ukládat články do své databáze, lokálně autentizovat a autorizovat uživatele, podporovat lokalizace do více jazyků a mělo by být jasné, jak přesně bude komunikovat s Metastorm BPM, případně by měla být hotova knihovna funkcí, budou-li na portálu třeba.

3.1.1 Příprava prostředí

Jelikož byl pro tento projekt zvolen aplikační rámec Microsoft MVC 2 a programování bude probíhat především v jazyce C#, je cestou nejmenšího odporu zvolit Microsoft Visual Studio 2010 jako integrované vývojové prostředí, Microsoft Internet Information Services 7 jako webový server a Microsoft .Net Framework 4 jako výchozí sadu knihoven. Alternativou by bylo zvolení Mono Develop jako integrovaného vývojového prostředí, Mono Framework jako výchozí sadu knihoven a Apache2 jako výchozí webový server. Ale vzhledem k požadavku, aby Strategický portál využíval jako úložiště dat Microsoft SQL Server, by tato varianta nepřinášela žádné výhody (kterými by mohly být nezávislost na Microsoftu a operačním systému Windows); spíše naopak, je třeba zaručit, že databázový server pro Strategický portál a samotný Strategický portál mohou fungovat na stejném počítači. Jako verzovací systém použiji SVN, neboť se používá i k ostatním projektům ve firmě REENGINE a na stávající verzi portálu. Výhodou bude i možnost využití firemních serverů pro hostování zdrojových kódů tak portálu.

Instalace výše zmíněných programů proběhla vcelku hladce a bylo možno přistoupit k přípravě samotného projektu. Vytvoření projektu ve Visual Studiu je sice záležitost na jedno kliknutí, ale struktura základního projektu v Microsoft MVC Framework není připravena na „skládání“ webu z menších komponent tak, jak bych si představoval. V Microsoft MVC jsou komponenty jen části šablony, nemají vlastní chování. Pátral jsem na Internetu a pokud někdo měl v takovém projektu komponenty s vlastním chováním, jednalo se o komponenty typu Web.Forms (oba dva aplikační rámce, jak Web.Forms tak MVC, je možno libovolně míchat dohromady, jen je třeba dát pozor, aby nevznikl nepřehledný projekt s nedeterministickým chováním, neboť oba rámce jsou v mnoha věcech protichůdné). Mým cílem bylo mít komponentu s vlastním chováním a odpovídající struktuře MVC v samostatné knihovně tak, jak je to možno vidět např. v Nette Framework pro PHP.

U některých částí webu je možno použít rozčlenění do částí (*Web Area*), které nabízí druhá verze Microsoft MVC frameworku. Bohužel takto vytvořená část webu je nedílnou součástí projektu Visual Studia, což bývá v některých případech nežádoucí (např. tehdy, je-li požadována modularita).

Nakonec jsem vytvořil bázovou třídu pro komponenty (`ReComponent`), od které ostatní komponenty dědí. V této třídě je jedna metoda, která ručně vykresluje části pohledů (*partial views*). Samotné komponenty se budou skládat z třídy komponenty (dědící od `ReComponent`), modelu pohledu (*view model*) a šablony. Každá komponenta bude samostatný C# projekt – knihovna tříd (přípona .dll). Tyto komponenty budou zajišťovat většinu funkčnosti portálu – samotné jádro portálu by mělo být minimalistické a neobsahovat zbytečnou funkčnost navíc.

Jako vůbec první komponentu jsem vytvořil komponentu pro zobrazení bleskových zpráv (*flash message*). Mělo by se jednat o krátké zprávy uživatelům, jak informativní, tak chybové. Důležité je, aby se zobrazily v horní části stránky a aby zůstaly viditelné i po přesměrování na novou stránku. Pohledový model bleskové zprávy je součástí bázového pohledového modelu, což zajišťuje, že bleskové zprávy budou součástí každé stránky.

Dále přišla na řadu knihovna pro logování zpráv. Původně jsem uvažoval o použití bezpochyby skvělé knihovny log4j, která má pro Microsoft .Net ekvivalent. Bohužel její konfigurace a začlenění do projektu by stály přibližně stejné úsilí, jako si napsat svoji vlastní (což je bohužel velmi rozšířený nešvar mnoha skvělých věcí v celém prostředí Javy a z Javy pocházejících). Zde byla částečně využita metodika Test driven development, kdy nejprve vznikl unit test a třída pro logování byla vytvářena tak, aby unit test skončil úspěšně. Při logování se rozlišuje 5 druhů zpráv od informativních po kritické chyby a je možné nastavit, které druhy se mají zapisovat do souboru a které nikoliv.

3.1.2 Databáze

Základním předpokladem pro úspěch implementace článků a lokalizací je mít zprovozněnu komunikaci portálu a databáze. Pro tuto problematiku jsem byl zvyklý využívat aplikační rámec `Hibernate`[12], který je dostupný i pro prostředí Microsoft.Net. Bohužel jeho konfigurace je v tomto prostředí netriviální záležitostí. Proto bylo využito rámce Microsoft LINQ to SQL, který má zabudovanou podporu přímo ve Visual Studiu. Vytvoření persistentních tříd je tak mnohem jednodušší a de facto jediná záležitost, kterou je nutno se zabývat, je vytvoření třídy objektu pro „repozitář“, který se bude zapouzdřovat funkce pro vytváření, mazání a synchronizaci persistentních objektů s databází stejně jako dotazování se na ně. Pro každou oblast webu jsem vytvořil vlastní mapování do databáze, neboť ve výsledné klientské instalaci portálu nemusí být všechny oblasti přítomny. Stejně tak při logickém rozčlenění podle oblastí se snáze udržuje jak databáze, tak mapování.

3.1.3 Články

Jako první použitelná součást projektu vznikl modul nazvaný „ArticleGroups“, tedy skupiny článků. Jedná se o sdružení statických článků uložených do databáze portálu a uživatel s právy k jejich administraci může v rámci skupiny článků články přidávat, mazat, upravovat a měnit jejich pořadí.

12 ORM (*object-relational mapping*) aplikační rámec Hibernate je nejznámějším aplikačním rámcem pro prostředí jazyka Java, který řeší persistenci objektů do relačních databází. Je portován i do prostředí Microsoft.Net, ale bohužel se zde ztrácí některé z jeho skvělých vlastností, které ztěžují jeho použití.
http://www.hibernate.org/

Celá skupina článků je zobrazena vždy jako celek. Zobrazené články jsou závislé na právě zvolené lokalizaci a pro každý jazyk je nutno vytvořit její články znovu.

V nynější fázi je kladen důraz především na co nejrychlejší představení použitelného portálu, proto jsou články řešeny nejjednodušším možným způsobem – veškerá funkčnost je nyní na straně serveru. Jedinou výjimkou, kde byla použita technologie AJAX (asynchronní javascript), je změna pořadí článků. Administrátor myší chytne článek a posune jej na požadované místo. Server je technologií AJAX informován o novém pořadí článků ve skupině.

Zatím chybí grafický editor článků. Ten bude začleněn v pozdější fázi, kdy bude implementována ucelená knihovna pro formuláře.

Články jsou vytvořeny jako samostatná oblast webu (*web area*[13]) zaregistrovaná pod adresou „/Articles“. Mapování je následující:

- **/Articles/{id}/{language}** zobrazí skupinu článků s identifikátorem `id` (textový řetězec) a jazykem `language` (dvoupísmenná zkratka jazyka, např. „cs“ pro češtinu);
- **/Articles/{id}/{language}/[Delete|Edit]/{articleId}** úprava/ostranění článku s identifikátorem `articleId` ze skupiny;
- **/Articles/{id}/{language}/New** přidání článku do skupiny;
- **/Articles/{id}/{language}/Sort** změna řazení článků pomocí technologie AJAX. Nové pořadí přijde jako pole v `GET`. Akce pro změnu pořadí vrátí zpět bleskovou zprávu ve formátu JSON, ve které bude uživatel informován, zda-li akce proběhla úspěšně či nikoliv a případně proč.

3.1.4 Javascript, jQuery a AJAX

Rozšíření portálu o ucelenou knihovnu pro práci se skripty v jazyku Javascript a pro využití technologie AJAX byla sice plánována až do druhé fáze implementace, avšak řazení článků si vyžádalo tuto implementaci této funkčnosti již nyní. Dle oficiálního textu na webových stránkách knihovny, JQuery je rychlá a konzistentní javascript knihovna, která usnadňuje traverzování po HTML dokumentu, ošetřování událostí, animace a AJAX interakci pro rychlý vývoj webů [6]. Microsoft dodává Visual Studio s již předinstalovanou knihovnou jQuery, která je ihned použitelná ve webových projektech, avšak jedná se o starší verzi. Proto jsem zvolil cestu ruční instalace a začlenění do portálu. To dává jistotu, že knihovna jQuery bude aktuální a budu moci použít oficiální dokumentaci na webu.

V článcích je použita funkce `sortable` [14], která umožní myší přetahávat (*drag and drop*) poduzly daného uzlu HTML DOM stromu [6]. Nové pořadí je pomocí technologie AJAX posláno na server pomocí jQuery funkce `get`. Vzhledem ke komplexnosti práce s Javascript objektem `XmlHttpRequest` [15], který se standardně používá k práci s AJAX, je použití jQuery značným zjednodušením a usnadněním.

13 Novinka představená v druhé verzi aplikačního rámce Microsoft MVC. Umožňuje vytvořit „web ve webu“ – část webové prezentace, která má strukturu stejnou jako nadřazený web. V minulé verzi bylo třeba používat samostatné řešení Visual Studia (*Visual Studio Solutions*), což bylo nepříjemné – pro každou část webu bylo třeba konfigurovat webový server, nebyly integrované a nebylo možné je efektivně vyvíjet zároveň. http://msdn.microsoft.com/en-us/library/ee671793.aspx

14 Viz oficiální dokumentace a příklad na http://jqueryui.com/demos/sortable/

15 Viz oficiální specifikace W3C http://www.w3.org/TR/XMLHttpRequest/

Drobnou nevýhodou je fakt, že pokud má klient vypnutou podporu Javascript ve svém webovém prohlížeči, nebude pro něj tato funkčnost dostupná. Proto bude většina portálu implicitně psána tak, aby běžnému uživateli (ne administrátorovi) fungoval i bez podpory Javascript.

3.1.5 Lokalizace

Zařazení implementace lokalizací na začátek projektu může být sporné rozhodnutí. Na jednu stranu zatím nejsou potřeba, ale je bezpečně známo, že potřebné budou. Vzhledem k tomu, že jejich začlenění nyní bude méně pracné (při dalším psaní zdrojového textu již budou textové řetězce vytvářeny jako překladové konstanty, tedy je nebude třeba v textu ručně vyhledávat[16]).

V bázové třídě pro objekty typu `Controller` (a tedy přístupná ve všech) je funkce `Translate`. Té je možno předat řetězcovou konstantu a ona podle uživatelem právě zvoleného jazyka vrátí překlad. Pokud překlad není nalezen, vrátí se překladová konstanta. Jako překladové konstanty poslouží anglické texty, které budou následně lokalizovány do češtiny (a v pozdějších fázích, kdy projekt bude prakticky využíván firmou REENGINE, pravděpodobně přibude i slovenština).

Lokalizace jsou načítány z databáze portálu. Aby bylo možné vytvořit stránku pro úpravu lokalizací, je vhodné, aby v databázi byly uloženy veškeré lokalizační konstanty, na jejichž překlad bylo dotazováno. Tím pádem bude jednoduché zobrazit „nepřeložené“ konstanty a upozornit administrátora, že je třeba překlad doplnit. V případě uložení konstant do databáze je snadné přidat funkčnost pro import a export, která se bude hodit pro přenos lokalizací mezi více servery.

Nyní je implementována pouze persistence lokalizací v databázi, jejich automatické přidávání a funkce pro vyžádání si překladu. Samotné překládání a export/import lokalizací bude implementován v pozdějších fázích, kde bude více času. Menší prioritu má proto, že tuto funkčnost nyní dostatečně dobře poskytuje Microsoft SQL Server Management Studio, který bývá nainstalován i v produkčním prostředí.

3.1.6 Přihlašování uživatelů, řízení přístupu

Strategický portál může agregovat několik rozdílných služeb, proto je potřeba zajistit adekvátní autentizaci a autorizaci uživatelů. Prostředí Microsoft.Net je schopno zajistit autentizaci na základě účtů Windows nebo záznamů v databázi a autorizaci na základě záznamů v databázi [4]. Vzhledem k tomu, že bude potřeba spolehlivě autentizovat a autorizovat lokální uživatele portálu, např. administrátora, i v době, kdy připojení k databázi nebude funkční případně ještě nebudou tabulky vytvořeny (instalátor bude součástí portálu), je třeba tuto problematiku řešit pro lokální uživatele jinak.

V Microsoft.Net je možno řešit vlastní způsob autentizace uživatelů pomocí objektů implementující rozhraní `MembershipProvider` [4]. Přestože toto rozhraní obsahuje mnoho funkcí, pro dostačující funkčnost v případě lokálních uživatelů je třeba jen `ValidateUser`. Ostatní funkce je možno ponechat ve „výchozím“ stavu, tedy vyhazovat (*throw)* výjimku `NotImplementedException`[17]. Přestože se nejedná o moc šťastné řešení, ušetří nyní mnoho času,

16 Zde na plné čáře vyhrává vývojové prostředí NetBeans pro Javu (pro Eclipse existuje plugin), které má ukázkovou podporu pro překládání projektů. Jakmile se rozhodnete svůj projekt přeložit, umí v textu vyhledat všechny řetězcové konstanty a vytvořit z nich překladové konstanty. Visual Studio tuto funkčnost bohužel ani v roce 2010 nemá.

17 Tuto nečistou metodu používají i tvůrci prostředí Microsoft.Net. Například když si z databáze pomocí LINQ to SQL vytáhnete nějakou kolekci, zjistíte, že přestože implementuje rozhraní IQueryable, mnoho metod vrátí tuto výjimku.

neboť ostatní „předepsaná“ funkčnost nebude pro lokální uživatele potřebná. Jiná situace nastane při ověřování uživatelů proti systému Metastorm BPM. V takovém případě bude plná implementace tohoto rozhraní na místě.

Podobným způsobem je možno řešit autorizaci uživatelů. Zde se jedná o objekt implementující rozhraní `RoleProvider` [4].

Jako úložiště lokálních uživatelů jsem zvolil soubor `web.config`, tedy konfigurační soubor celé webové aplikace. Jedná se o místo, kde každý vývojář znalý prostředí .Net bude očekávat konfiguraci [5]. Aby bylo možno přidat sekci do konfiguračního souboru, je třeba implementovat třídu dědící od objektu `ConfigrationSection` [5]. Jelikož se bude jednat o sérii záznamů, je třeba implementovat třídu pro každý prvek záznamu a ještě pro jejich kolekci. Načítání konfigurace při vytváření objektů ověřujících uživatele a jejich práva jsem implementoval do samostatného objektu obsahující statickou metodu pro toto, neboť se jedná o část společnou oběma těmto objektům.

Konfigurace uživatelů v konfiguračním souboru může být následující:

```
<configSections>
     <section name="LocalUsersAuthenticator"
type="MvcPortal.ReSupport.LocalAuthenticator.LocalAuthenticatorConfiguration" />
</configSections>
<LocalUsersAuthenticator>
  <localUsers>
  <clear />
  <add loginName="user" password="pass" permissions="" />
  <add loginName="localadmin" password="Ô&Dom-5]t_+~6v\rX"
permissions="DatabaseAdmin,LocalizationAdmin" />
  </localUsers>
</LocalUsersAuthenticator>
```

Kód 3.1: Ukázka konfigurace uživatelů v souboru Web.config

V třídách typu `Controller` může být použita následující anotace u metod, které přistupují k chráněným zdrojům:

```
[Authorize(Roles = "LocalizationAdmin")]
public ActionResult Delete(int id)
{...}
```

3.1.7 Navigační menu

Vzhledem k faktu, že Strategický portál může být využíván více nezávislými subjekty a jeho „kopie“ budou mít různý obsah, vniká potřeba mít možnost ovlivnit podobu každé zvlášť. Vzhledem k možnosti, aby si provozovatelé portálů případně mohli podobu navigační menu nastavit (pokud by tuto vlastnost požadovali) je potřeba udržovat informace potřebné k vykreslení menu v samostatném konfiguračním souboru. Nejjednodušší možnost, tedy vytvořit `.ascx` šablonu pro celé menu, není použitelná.

Pro konfigurační soubor menu jsem využil formát `XML` a snažil se co nejvíce dodržet strukturu z již existujících portálů. Hlavní prvek XML stromu tvoří uzel `navigation` , který má 1-n uzlů `node` . Každý tento uzel má atribut, zda-li se jedná o položku menu odkazující na článek či jiný obsah, informace potřebné k vytvoření odkazu na odkazovaný obsah (u základních uzlů jsou to `controller`, `action`, `area` a u uzlů odkazujících na články postačí `articleId`). Každý uzel může mít pod sebou ještě další uzly obou typů (`subnodes`) a také seznam rolí, pro které se má navigační uzel vykreslit (`roles` s potomky `role` mající atribut `name` , tedy název role).

Vzhledem k tomu, že čtení xml souboru může zabrat poměrně velké množství času (u stávajících PHP portálů jsem profilováním zjistil, že v extrémních případech se může jednat i o 700ms), uvažoval jsem o možnosti zpracovávání konfiguračního souboru pomocí `xsl` transformace. Sice by pro každou uživatelskou roli musela být samostatná šablona, ale na druhou stranu by se usnadnilo vytváření cache výsledné nabídky. Tato cesta byla po konzultaci ve firmě REENGINE zamítnuta.

Zpracování konfiguračního souboru tedy probíhá klasicky pomocí knihovny LINQ to XML. Vzhledem k faktu, že požadavky na menu se mohou velmi lišit mezi jednotlivými instalacemi Strategického portálu, je knihovna zpracovávající konfigurační soubor vytvořena jako samostatný projekt kompilovaný do dll knihovny (projekty `MvcPorta.ReSupport.Navigation` a `MvcPorta.ReSupport.NavigationTests`). V této knihovně se nachází jak datové typy pro položky menu tak i parser konfiguračního souboru. Pokud tedy bude potřeba změnit jeho vnitřní strukturu, nebude třeba kompilovat celé jádro portálu, postačí pouze změnit tuto knihovnu. Samotný proces parsování je rozdělen do několika fází a každá fáze má samostatnou metodu bez postranních efektů. Programovací styl se zde tedy blíží spíše funkcionálnímu než imperativnímu stylu, což je možné také díky knihovně LINQ to XML, která umožňuje velmi snadné procházení XML stromu. Díky tomu bylo usnadněno testování. Ke každé metodě, která zpracovává xml vstup, existuje sada unit testů, což usnadnilo nejen začlenění kódu (vím, že funguje), ale přijde vhod v případě, že by bylo potřeba tuto knihovnu měnit. Unit testy jsou opět v samostatném projektu.

Poslední otázkou zůstává vykreslení jednotlivých položek navigačního menu. Aby bylo možno jej ovlivnit opět bez nutnosti při každé změně kompilovat jádro portálu, probíhá v šablonách ovládacích prvků (*control, partial view*). Konkrétně se jedná o 2 šablony – jedna pro celé menu a druhá pro samotné prvky. V rámci těchto dvou šablon se vykreslují i „zpětné odkazy“ (*breadcrumbs*).

Rozhodnutí, které prvky menu jakému uživateli vykreslit a které nikoliv, probíhá již při zpracovávání vstupního souboru. Parseru je předána reference na objekt s identitou uživatele. Tento objekt má metodu `isInRole` , která vrací hodnoty pravda/nepravda pro každou roli. Vzhledem k tomu, že názvy rolí, které mají mít danou položku menu zobrazenou, jsou obsaženy

v konfiguračním souboru, je možno se tímto způsobem rozhodnout, zda-li se bude položka zpracovávat či nikoliv (pomáhají k tomu také i dotazovací schopnost knihovny LINQ to SQL).

Názvy navigačních uzlů jsou při zpracovávání lokalizovány pomocí již hotového modulu pro lokalizace. Tím pádem může být pro všechny jazyky jeden konfigurační soubor.

V rámci vytváření navigačního menu byly přidány akce pro změnu jazyka do třídy `LoginController` , která má na starosti přihlašování uživatelů. Jelikož http protokol je standardně bezstavový, informace o zvoleném jazyku je třeba uchovávat jinde. Po konzultaci bylo jako úložiště zvolena cookie na straně uživatele. Jelikož se pravděpodobně nebude jednat o jediný případ využití cookies, vytvořil jsem i model pro práci s nimi. Tento model je popsán v kapitole 3.1.8. Samotné přepínání jazyků probíhá pomocí vlaječek pod navigačním menu, na opačné straně stránky než jsou zpětné odkazy (*breadcrumbs*).

3.1.8 Práce s cookies

Vzhledem k potřebě ukládat některá data v malých souborech na straně uživatele jsem vytvořil speciální třídu ve jmenném prostoru modelů. (Jelikož se jedná o logiku aplikace, patří do modelů.) Tento objekt zapouzdřuje veškerou práci s cookies, díky čemuž jsou manipulace s nimi na jednom místě, což usnadní pozdější změny. Konkrétně se jedná o soubor `MvcPortal.Models.Shared.CookieSettingsModel` .

Asp.Net rozlišuje cookies, které přijdou s http požadavkem a které se odešlou s odpovědí na něj [5], proto je potřeba toto reflektovat i v portálu. Aby ten, kdo by chtěl přidat další hodnotu do cookie, nemusel stále opakovat stejný stereotyp, jsou v této třídě připraveny privátní metody pro získání a uložení hodnoty do cookie. Veřejně přístupné vlastnosti třídy představují konkrétní hodnoty. V metodě nastavující hodnotu vlastnosti (*setter*) se tedy jen zavolá příslušná privátní metoda třídy s parametry představující jméno, pod kterým má být hodnota uložena v cookie a samotná hodnota. V metodě pro získání hodnoty (*getter*) se pouze zavolá privátní metoda třídy a ověří se, zda-li se vrátila smysluplná hodnota. Pokud ne, je možno tento stav ošetřit již zde. Zjišťování, zda-li opravdu cookie existuje a zda-li existuje hodnota tohoto jména v ní, se děje v privátní metodě a není třeba se tím dále zabývat.

Díky této třídě je práce s hodnotami z cookie v samotné aplikaci snadná. Pokud `cookieSettings` je instancí třídy `CookieSettingsModel` , stačí napsat `cookieSettings.JmenoHodnoty` pro získání hodnoty z cookie a `cookieSettings.JmenoHodnoty = NovaHodnota` pro její nastavení. Cookies sice umožňují ukládat pouze řetězce, ale v případě, že bude potřeba uložit např. datum, je možno převést datum na řetězec v *setter* metodě a zpět v *getter* metodě, což zajistí bezpečné uložení a načtení data.

Podobnou třídu plánuji vytvořit i pro práci se `session`, vznikne-li taková potřeba (pravděpodobně ano).

3.1.9 Komunikace přes webové služby

Pro komunikaci mezi portálem a Metastorm BPM serverem bude využito aplikačního rámce Microsoft WCF (*Windows communication foundation*). Tento rámec umí zapouzdřit různé formy komunikace a poskytuje pro vše jednotné rozhraní [7]. Jak je vidět na obrázku 1.2 na straně 4, Strategický portál by měl volat 2 druhy webových služeb – jedny vygenerované od Metastorm BPM a

další, které přistupují přímo do databáze. Vzhledem k tomu, že jejich rozhraní se liší, je ve stávajících verzích portálu nutno rozlišovat, zda-li služba přistupuje přímo do databáze či k procesům Metastorm BPM [11]. Navíc pro každou modelovou mapu v Metastorm BPM vznikala nová webová služba a ve stávajících verzích portálu bylo tedy nutno tento stav reflektovat [12].

Vzhledem k plánovanému přímému využití knihovny ECL jsem se s konzultantem této bakalářské práce (T. Sotoniakem z firmy REENGINE) dohodl na sjednocení obou druhů služeb. Na serveru tedy poběží pouze jedna webová služba, která bude zapouzdřovat oba druhy komunikace – jak přímo s databází, tak s Metastorm BPM přes knihovnu ECL. Tato webová služba bude využívat Microsoft WCF. Problematika bude podrobněji popsána v kapitole 4.1.

V první fázi vývoje je tedy ještě potřeba vytvořit komunikační protokoly a implementovat jejich model do jádra Strategického portálu. Webová služba bude (alespoň v těchto fázích vývoje portálu) posílat 3 druhy dat. Buď formuláře (prázdné i předvyplněné)[18], data (zobrazí se do tabulky) nebo informace o uživateli (zda-li byl úspěšně přihlášen a jaké má role). Vše bude posíláno jako objekt, případně seznam objektů, neboť se tím usnadní práce a zpřehlední zdrojový kód, navíc Microsoft WCF tuto funkčnost podporuje [7].

3.1.10 Zakončení fáze I

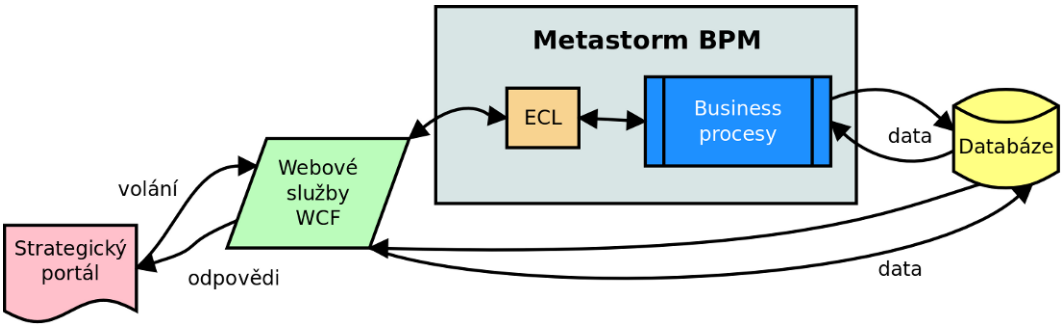
V tuto chvíli je první fáze implementace Strategického portálu u konce. Nyní Strategický portál připomíná spíše systém pro správu obsahu (*Content Management System, CMS*), neboť umí uložit a zobrazit skupiny článků z databáze portálu. Je v něm plně funkční systém pro přihlášení a autorizaci uživatelů, konfigurovatelné navigační menu, systém pro lokalizaci, knihovna pro logování chyb a varování, jsou připravené některé části pro usnadnění dalšího vývoje a je navržena komunikace s Metastorm BPM.

18 Téměř každá Metastorm BPM akce manipulující s procesem může mít (a také dost často mívá) formulář pro danou akci, který je možno vyplnit či upravit.

4 Fáze II

V této části bude popisována implementace komunikace mezi Strategickým portálem a Metastorm BPM. Schema stávajícího způsobu komunikace portálu a Metastorm BPM je na obrázku 1.2 na straně 4. Na konzultaci ve firmě REENGINE byla dohodnuta změna oproti tomuto původnímu schematu především proto, že webové služby, které přistupovaly k procesům Metastorm BPM, měly jiné rozhraní než ty, které přistupovaly přímo k datům z databáze. Oba dva typy webových služeb budou nyní sjednoceny a v portálu již tedy nebude třeba mezi nimi rozlišovat, což významně usnadní práci dalším vývojovým pracovníkům. Aktualizované schema je možno vidět na obrázku 4.1.

Pro zobrazování dat z Metastorm BPM byla vytvořena nová oblast webu a v ní jsou také modelové třídy pro získávání dat z tohoto zdroje.



Obrázek 4.1: Nová koncepce komunikace Strategického portálu a Metastorm BPM

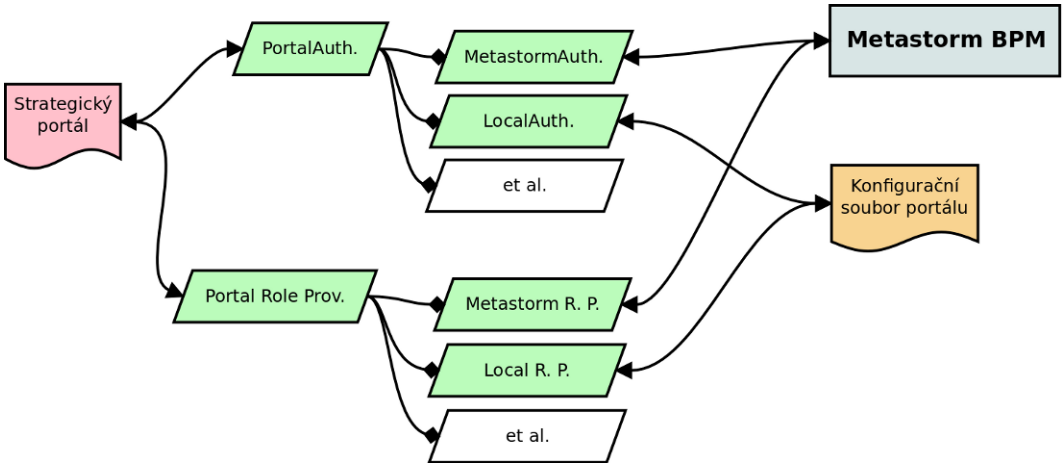
4.1 Možnosti komunikace

Na konzultaci ve firmě jsem se dozvěděl, že s Metastorm BPM je možno komunikovat buď přes knihovnu ECL nebo přes jím vygenerované webové služby. Webové služby vygenerované Metastorm BPM trpí poměrně velkým neduhem, kterým je jejich rychlost – jelikož nedokáží udržet kontext mezi dvěma požadavky, dochází při jejich volání k tomu, že se vždy znovu inicializuje celé jádro, což zabírá poměrně dost času. Proto v případě, že je potřeba pouze získat data a nemanipuluje se s procesy, existují ručně napsané webové služby, které přistupují přímo do databáze. V případě přímého volání knihovny ECL již možnost udržení kontextu existuje. Aby Strategický portál mohl běžet na jiném počítači než Metastorm BPM, volání knihovny ECL bude zapouzdřovat nový firmou REENGINE připravovaný aplikační rámec PX. Díky aplikačnímu rámci PX budou k dispozici WCF webové služby, které budou obsahovat funkčnost jak původně generovaných webových služeb Metastorm BPM pro manipulaci s procesy, tak ručně psaných pro přímý přístup k datům, stejně jako přihlašování uživatelů [11].

Jednotlivé „typy“ komunikace a způsob, jakým s nimi Strategický portál nakládá, rozeberu podrobněji v následujících kapitolách.

4.1.1 Přihlašování uživatelů

Ne všechny informace získané z Metastorm BPM je možno zobrazit všem uživatelům a je tedy nutno i v případě komunikace s Metastorm BPM řešit otázku autentizace a autorizace uživatelů. Vzhledem k tomu, že ASP.Net neumí standardně pracovat s více objekty pro autentizaci a autorizaci uživatelů [5][19], bylo třeba vytvořit vlastní třídy objektů (`PortalAuthenticator`, `PortalRoleProvider`), které zapouzdřují autorizaci a autentizaci do/k více zdrojům. Tyto objekty zatím poskytují možnost přihlášení se a získání seznamu rolí lokálně (popsáno v kapitole 3.1.6) a proti Metastorm BPM (nově vytvořené třídy `MetastormAuthenticator` a `MetastormRoleProvider`). V pozdějších fázích mohou být opět rozšiřovány, především o přihlašování k dalším zdrojům nebo o načítání informací o uživateli[20]. Stručné schema je vidět na obrázku níže.



Obrázek 4.2: Upravený systém autentizace a autorizace uživatelů

Pokud alespoň jeden objekt, který se stará o autentizaci uživatele, uspěje s přihlášením, je uživatel brán jako přihlášený. Vzhledem k tomu, že někdy jsou i nepřihlášeným uživatelům přiřazovány role (např. „Guest“ u Metasotrm BPM), pokusí se pro něj příslušné objekty získat role. `PortalAuthenticator` a `PortalRoleProvider`, stejně jako objekty, které zapouzdřují, jsou kompilovány jako samostatné knihovny. Tím pádem je lze jednoduše „vyměnit“ či upravit bez nutnosti zásahu do samotného jádra Strategického portálu.

Pomocí testovacího klienta WCF služeb[21] je možno snadno zjistit, že samotná webová služba pro přihlášení uživatelů a získání uživatelských rolí má v současné době pouze jednoduché rozhraní s metodami pro ověření správnosti kombinace uživatelského jména a hesla a pro získání seznamu uživatelských rolí[22]. Rozhraní a schema funkce je vidět na obrázku 4.3. Webová služba

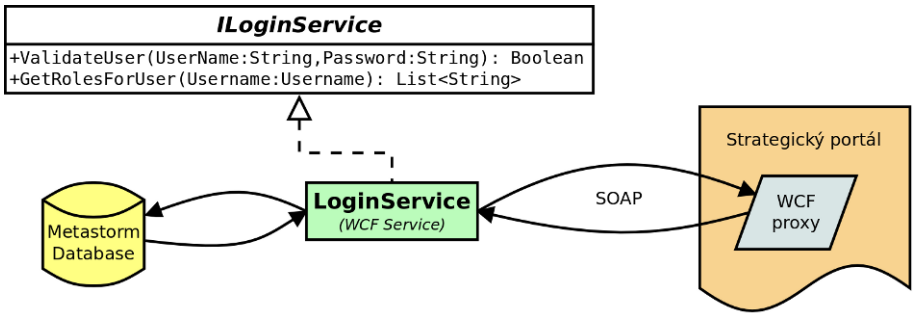
19 Vždy je aktivní ten, který je ve Web.config označen jako výchozí (*default*). Lze mezi nimi programově přepnout. Bohužel jsem ale nepřišel na způsob, jak nějak efektivně mezi nimi přepnout, pokud jeden selže, případně jak zároveň načítat role z více autorizačních objektů.

20 ASP.Net má k tomuto účelu připravené rozhraní jménem „ProfileProvider“. Viz http://msdn.microsoft.com/en-us/library/2y3fs9xs.aspx

21 Jedná se o nástroj dodávaný s MS Visual Studio. Viz http://msdn.microsoft.com/en-us/library/bb552364.aspx

22 Zdje (a i v jiných místech v této práci) jsou používána rozhraní s objektem třídy „System.Collections.Generic.List“ pro kolekce přenášené přes weobvé služby WCF. Ve skutečnosti záleží na nastavení generátoru proxy tříd, zda-li se pro kolekce použije pole, seznam či něco jiného. Pokud se podíváte přímo na SOAP dotaz např. programem Wireshark (program pro sledování provozu na síti) [http://www.wireshark.org/], zjistíte, že se přenáší jako kolekce (pole). Nicméně práce se seznamy je v prostředí .Net příjemnější než s poli a proto nevidím důvod této hezké vlastnosti generátoru proxy

`LoginService` implementuje rozhraní `ILoginService`, které definuje dvě operace – první vrátí logickou hodnotu pravda / nepravda pro kombinaci jména a hesla na základě toho, zda-li je tato kombinace platná či nikoliv (v databázi Metastorm BPM). Druhá vrátí seznam rolí uživatele, které se nachází v téže databázi. Zprávy mezi portálem a webovou službou jsou posílány pomocí SOAP protokolu. V portále je vygenerována proxy třída, která se stará o správné zkonstruování SOAP dotazu a o správnou objektovou reprezentaci odpovědi. S touto třídou pak pracuje modelová část portálu.



Obrázek 4.3: Webová služba pro autentizaci a autorizaci Metastorm BPM uživatelů na portále

4.1.2 Odesílání a přijímání dat a formulářů

Přistupovat k procesům Metastorm BPM a manipulovat s nimi je možné dvěma způsoby. Buď si portál vyžádá data ve formě tabulky nebo si vyžádá formulář pro danou akci procesu. Pro příklad jsem zvolil jednoduchý program na ukládání kontaktů. S procesním řízením sice moc nesouvisí, ale přesto je na něm možno demonstrovat technickou stránku komunikace. Výslednou webovou službu je možno vidět na obrázku 4.4.

Rozhraní `IExampleDataService` reflektuje možnosti manipulace s kontakty.

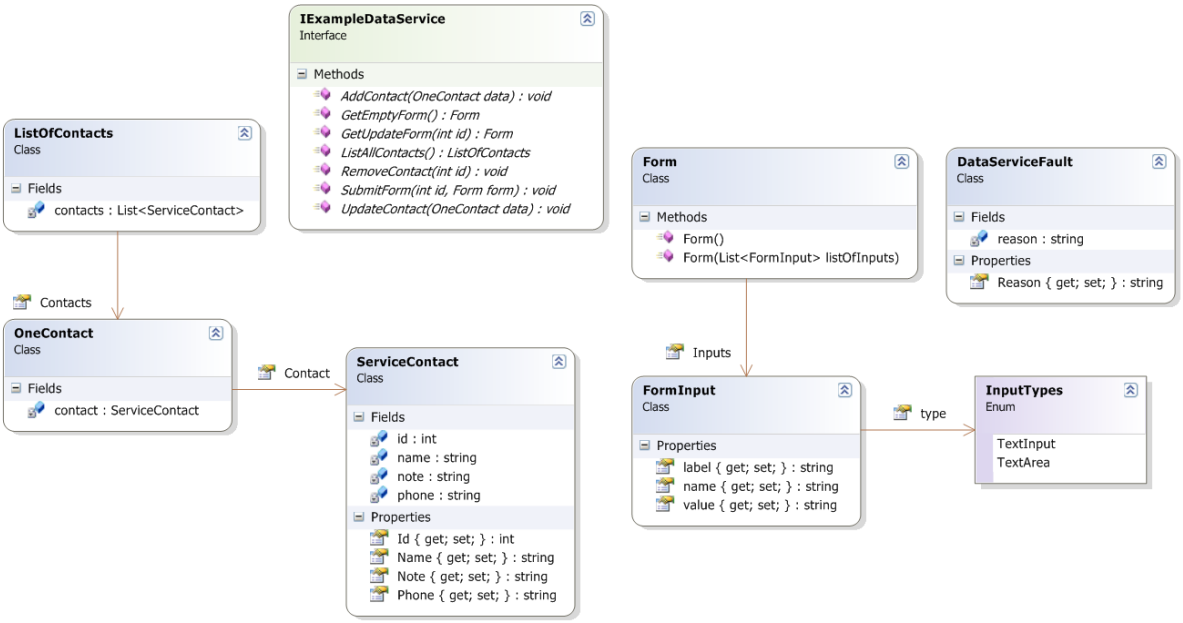
Manipulace s daty:

- *AddContact* (přidání kontaktu);
- *RemoveContact* (smazání kontaktu);
- *UpdateContact* (úprava kontaktu);
- *ListAllContacts* (seznam všech kontaktů).

Formuláře:

- *GetEmptyForm* (prázdný formulář – pro přidání);
- *GetUpdateForm* (předvyplněný formulář – pro úpravu);
- *SubmitForm* (odeslání vyplněného formuláře).

tříd nevyužít.



Obrázek 4.4: Diagram tříd webové služby pro přístup k datům a formulářům

Pro objektovou reprezentaci kontaktu byl vygenerován objekt `ServiceContact`, jehož vlastnosti odpovídají vlastnostem entity ukládané do databáze (Webová služba vnitřně používá Entity framework)[23]. V případě, že metody webové služby pracují pouze s jedním kontaktem, je objekt typu `ServiceContact` zapouzdřen do objektu typu `OneContact` a ten je odesílán či přijímán. Toto zapouzdření je zde z toho důvodu, aby se k samotnému kontaktu v případě potřeby dala přidat další data.

U metody pro získání seznamu všech kontaktů (tedy celé tabulky) je definován typ návratové hodnoty jako seznam objektů typu `ServiceContact`. Pro lepší představu je přiložena ukázka – kód 4.1.

```
<ListAllContactsResponse xmlns="...">
    <ListAllContactsResult xmlns:a="..." xmlns:i="...">
        <a:Contacts>
          <a:ServiceContact>
            <a:Id>5</a:Id>
            <a:Name>Jan Novák</a:Name>
            <a:Note>Poznámka k Janu Novákovi</a:Note>
            <a:Phone>+420 776 998 443</a:Phone>
          </a:ServiceContact>
           <a:ServiceContact>
              ...
           </a:ServiceContact>
        </a:Contacts>
    </ListAllContactsResult>
</ListAllContactsResponse>
```

Kód 4.1: Část odpovědi webové služby zachycená programem Wireshark

23 Viz http://msdn.microsoft.com/en-us/library/aa697427%28v=vs.80%29.aspx

```
<xs:complexType name="ServiceContact">
<xs:sequence>
    <xs:element minOccurs="0" name="Id" type="xs:int"/>
    <xs:element minOccurs="0" name="Name" nillable="true" type="xs:string"/>
    <xs:element minOccurs="0" name="Note" nillable="true" type="xs:string"/>
    <xs:element minOccurs="0" name="Phone" nillable="true" type="xs:string"/>
</xs:sequence>

</xs:complexType>
    <xs:element name="ServiceContact" nillable="true"
type="tns:ServiceContact"/>

<xs:complexType name="ListOfContacts">

<xs:sequence>
    <xs:element minOccurs="0" name="Contacts" nillable="true"
    type="tns:ArrayOfServiceContact" />
</xs:sequence>
</xs:complexType>

<xs:element name="ListOfContacts" nillable="true" type="tns:ListOfContacts"/>

<xs:complexType name="ArrayOfServiceContact">
<xs:sequence>

    <xs:element minOccurs="0" maxOccurs="unbounded" name="ServiceContact"
    nillable="true" type="tns:ServiceContact"/>
</xs:sequence>
</xs:complexType>

<xs:element name="ArrayOfServiceContact" nillable="true"
type="tns:ArrayOfServiceContact"/>
```

Kód 4.2: Ukázka části vygenerovaného WSDL souboru

Jak jsem psal výše a jak je zřejmé z ukázek kódu 4.1 a 4.2, přesné údaje o typu sekvence nejsou pomocí zpráv webových služeb přenášeny. Ukázka části vygenerovaného WSDL (*Web Service Description Language* – jazyk pro popis webových služeb) obsahuje část s definicí typů. Zde je důležitý prvek `ListOfContacts`, který v sobě obsahuje sekvenci prvků typu `ServiceContact` , nazvanou `ArrayOfServiceContact` . Jedná se o prostou sekvenci bez bližší specifikace. Proxy třídy pro komunikaci s webovými službami, které generuje Visual Studio, umí sekvence interpretovat i jako seznamy[24], tedy tak, jak byly nadefinovány v původním rozhraní.

24 Záleží na nastavení generátoru proxy tříd. Více viz http://msdn.microsoft.com/en-us/library/ee354381.aspx

Rozhraní pro získávání a odesílání formulářů je podobné. Každý prvek formuláře je reprezentován objektem typu `FormInput`, celý formulář potom `Form`. Formulář obsahuje kolekci formulářových prvků, z nichž každý má unikátní název, hodnotu, popisek (určen pro zobrazení v HTML značce *label*) a typ, který je jedním z vyjmenovaných ve výčtu `InputType`. Zde nemá smysl vytvářet třídy pro každý myslitelný formulář, neboť se mezi sebou budou lišit pouze svými prvky, které tvoří kolekci. Proto stačí obecná třída, která bude ve všech webových službách stejná. V případě vyžádání si prázdného formuláře pro nový kontakt jsou formulářové prvky prázdné. Pokud je vyžádán formulář pro úpravu, je poslán s předvyplněnými prvky. Oba dva formuláře se pak odesílají pomocí stejné metody webové služby, která již sama rozpozná, zda-li se jedná o zakládání nového či úpravu již existujícího kontaktu.

Poslední třída na diagramu 4.4 je `DataServiceFault` a slouží jako třída pro výjimky, pokud se něco nepovede či pokud něco selže.

4.2 Zakončení fáze II

V této fázi je Strategický portál napojen na Metastorm BPM přes webové služby a může manipulovat jak s procesy, tak s daty přes webové služby. Zatím zde chybí podpora pro stránkování tabulek a řazení či filtrování nad sloupci, avšak tuto funkčnost je nejprve třeba implementovat na straně webových služeb. Dále jsem implementoval ověřování uživatelů a získávání uživatelských rolí proti Metastorm BPM, což si vyžádalo drobnou úpravu systému autorizace a autentizace uživatelů.

5 Možná rozšíření a optimalizace

V této kapitole se pokusím nastínit možnosti dalšího rozšíření a optimalizace portálu. V tomto stádiu vývoje umožňuje Strategický portál získávat data pouze ze 2 zdrojů (nepočítáme-li „natvrdo“ vytvořený obsah, např. titulní stranu v ukázce práce), což v budoucnu nemusí stačit. Dále je možno vytipovat některá místa, ve kterých je běh portálu paralelizovat. Vzhledem k tomu, že v dnešní době je tendence zvyšovat počet procesorů či jader v procesoru spíše než frekvenci procesoru, považuji toto za dobrou cestu optimalizace výkonu.

Zdroje dat

Hlavní myšlenkou Strategického portálu je agregace dat z různých zdrojů a jejich zobrazování stylem, kterým si přeje zákazník. Proto rozšíření, které se přímo nabízí, je možnost získávat data nejen z databáze portálu a Metastorm BPM. V případové studii Strategického portálu [2] se mluví o sociálních sítích Facebook a Twitter a o tabulkách a grafech z Microsoft Excel.

Pro přidání nového zdroje dat je potřeba vytvořit příslušné modelové třídy. Zatím je tato problematika řešena tak, že pro každý zdroj existuje samostatná oblast webu a v její modelové části jsou třídy pro získávání a manipulaci dat z jednoho zdroje. Pokud by bylo potřeba na jedné webové stránce zobrazovat data z více zdrojů, bylo by vhodné zde provést refaktoring.

Paralelizace volání webových služeb

Někdy se stane, že je potřeba volat více webových služeb během jednoho požadavku na portál a volání webových služeb zabere nějaký čas[25]. Pokud není třeba návratovou hodnotu jedné webové služby předávat službě další, bylo by možné je zavolat paralelně.

Díky tomu, že v C# ve verzi 4.0 se objevila novinka jménem PLINQ (*Parallel LINQ*) [5] nebo že jazyk F# velmi snadno dokáže řešit paralelizaci [5] [8], bude technické provedení tohoto rozšíření usnadněné oproti ručnímu hraní si s vlákny.

Ukázka kódu 5.1 názorně ukazuje, jak jednoduše lze v jazyku F# stahovat WWW stránky. Podobně strukturovaný[26] kód se bude dát použít i pro volání webových služeb – výsledný kód by sice nebyl takto krátký a jednoduchý, ale logika by zůstala stejná.

25 Ve firmě REENGINE jsme na jedné z konzultací s vedoucím vývoje, Martinem Tlolkou, změřili, že v produkčním prostředí trvá z PHP portálu volání jedné jednoduché webové služby (vytáhnutí dat z databáze) přibližně 300 ms. Portál vzniklý v rámci této práce jsme měřit nemohli, neboť zatím je v rané fázi vývoje a v produkčním prostředí není nasazen, nicméně se očekává přibližně o třetinu lepší výsledek, především díky rychlejšímu zpracovávání XML.

26 Autor původního článku toto nazývá návrhovým vzorem, viz zdroj [8].

```
let http url = async { let req =  WebRequest.Create(Uri url)
                       use! resp = req.AsyncGetResponse()
                       use stream = resp.GetResponseStream()
                       use reader = new StreamReader(stream)
                       let contents = reader.ReadToEnd()
                       return contents }
      let sites = ["http://www.bing.com";
            "http://www.google.com";
            "http://www.yahoo.com";
            "http://www.search.com"]
      let htmlOfSites =
          Async.Parallel [for site in sites -> http site ]
          |> Async.RunSynchronously
```

Kód 5.1: Ukázka asynchronního stahování obsahu WWW stránek v jazyku F#. Zdroj: [8]

Paralelní modifikátory buněk tabulek

Pokud ze zdroje dat přijde tabulka, ne vždy je možno přímo zobrazit došlé hodnoty. Typickým příkladem tohoto jsou data („datumy“), kdy v databázi může být datum uloženo jinak než v člověkem čitelném formátu[27]. V nynější verzi portálu je téměř pro každou tabulku možno připravit sadu funkcí, které modifikují hodnoty [11]. Tyto funkce se vždy spouští jedna za druhou a v nejhorším případě, tedy kdy se modifikuje každá buňka tabulky, může modifikování hodnot zabrat značné množství času. (Více tabulek na stránce, mnoho záznamů a sloupců v tabulkách.)

Opět podobně jednoduchými způsoby, jak bylo popsáno v části o paralelním volání webových služeb, by bylo možno modifikování hodnot v tabulkách paralelizovat. Měřením na jedné z konzultací ve firmě REENGINE jsme zjistili, že v případě 4 tabulek o 15-20 řádcích a 4-5 sloupcích, kdy se modifikační funkce volá nejen pro jednotlivé buňky, ale i pro celé řádky tabulky, zabere modifikování hodnot přibližně 9200 ms (modifikační funkce, vč. režie volání, zabraly většinou 25-40ms každá). V případě paralelizování těchto modifikací by se mohlo dosáhnout značného zrychlení. Popsaný případ je extrémní a běžně dle vyjádření vedoucího vývoje firmy REENGINE nenastává, ale přes to jsme na konzultaci došli k závěru, že by nemuselo být špatné tento rys implementovat obecně přímo do jádra Strategického portálu.

27 Rád bych sem vložil poznámku, že na fakultě jsem při jednom z týmových projektů potkal člověka, který se podíval na unixový timestamp 514851826 a hned poznamenal „To je někdy kolem Černobylu, ne?“ Nutno dodat, že se spletl jen o 1 jedinou sekundu. Takovíto případové jsou ale velmi vzácní a většina lidí vidí jen „hromadu čísel“.
PS: Pokud byste si chtěli hodnotu timestamp-u ověřit, dodám, že tento je z časového pásma GMT+00:00; Ukrajina je v GMT+03:00

Knihovna pro snadné zobrazování formulářů

V původní verzi Strategického portálu postaveném na aplikačním rámci Nette jsou vcelku hezky řešeny formuláře – kódu programu stačí nadefinovat, co v něm má být a on se sám zobrazí. Asp.Net nic takového v základu nemá, což je škoda. Proto by bylo dobré najít plugin či vytvořit vlastní knihovnu, bude-li čas.

Automatizace procesu sestavení a nasazení

Ve firmě REENGINE v nedávné době proběhla diskuse o sjednocení a automatizaci procesu sestavení produktů a jejich nasazení. Zatím jsou jednotlivé části sestavovány a nasazovány odděleně (proces sestavení a nasazení je jiný pro portály a pro software Metastorm BPM), což v některých případech působí komplikace a hlavně – tento proces není moc pohodlný. Proto je v přípravě projekt, který by celý proces sestavení a nasazení produktů firmy „zautomatizoval nástrojem typu Ant nebo Maven“ [9]. Vzhledem k faktu, že většina produktů firmy REENGINE vyžaduje ke svému běhu Microsoft .Net Framework a nikoliv Java Virtual Machine, je pravděpodobné, že nástrojem pro automatizaci tohoto procesu se stane NAnt[28].

Ze zde uvedeného vyplývá, že dalším dobrým rozšířením je vytvoření NAnt skriptu. Kromě zjednodušení samotného nasazení projektu by přítomnost takového skriptu měla i další významný pozitivní dopad – na produkčních serverech[29] firmy REENGINE není nainstalována profesionální edice Microsoft Visual Studia. Vzhledem k tomu, že Express edice není schopna sestavit celé řešení portálu, znamenala by nutnost drobné změny v kódu otevírání několika projektů v předem určeném pořadí a klikání na tlačítko „build“, což by bylo časově velmi náročné. V případě přítomnosti NAnt skriptu by stačilo pro pohodlné sestavení projektu mít nainstalován pouze kompilátor jazyka C# a knihovny.

28 Nástroj pro automatizaci procesu sestavení a nasazení pro MS .Net Framework, podobný produktu Apache Ant pro Javu. Viz oficiální stránky projektu: http://nant.sourceforge.net/

29 Přestože přítomnost zdrojových kódů není v produkčním prostředí běžná, v určitých případech je možnost sestavit projekt na produkčním serveru velkou výhodou – např. když je potřeba udělat rychlou drobnou změnu a není po ruce počítač s nainstalovaným balíkem Microsoft Visual Studio. V takovém případě stačí stáhnout zdrojové kódy z verzovacího systému na produkční server (např. SVN) a tam provést změnu.

6 Závěr

V rámci této bakalářské práce vznikl webový portál postavený na technologii Microsoft ASP.Net a aplikačním rámci Microsoft MVC Framework (verze 2). Jedná se o znovupoužitelný portál, jehož primárním cílem je agregovat data z různých zdrojů, především pak z BPM softwaru firmy Metastorm.

V úvodní kapitole je podrobněji rozebrán cíl práce a motivace ke vzniku takovéhoto portálu, stejně jako vysvětlení pojmů, které se k této práci váží. Dále byl popsán současný stav portálového řešení firmy REENGINE a na tomto základě navrženo nové jádro portálu. Ve výběru technologie pro implementaci portálu mi byla ponechána určitá volnost, proto v kapitole o návrhu rozebírám i tyto možnosti a na základě vlastního provedeného srovnání a praktických zkoušek zdůvodňuji výsledné rozhodnutí. Vývoj Strategického portálu byl rozvržen na několik fází a implementace prvních dvou proběhla v rámci této práce.

Nakonec rozebírám možnosti dalšího rozšíření a optimalizace, především v oblastech paralelního zpracování dat.

Literatura

[1] KŘENA, B.: *Úvod do softwarového inženýrství – studijní opora*. Brno, Fakulta Informačních technologií VUT v Brně, 2010.

[2] SOTONIAK, T.: *Strategický portál, případová studie*. REENGINE a. s.

[3] PALETA, P.: *Co programátory ve škole neučí: aneb Softwarové inženýrství v reálné praxi*. 1. vyd. Computer Press, 2003. ISBN 80-251-0073-1.

[4] CONERY R., HANSELMAN S., HAACK P. et al.: *Professional ASP.NET MVC 1.0*. Wrox Publishing, Indianapolis, IN, USA. ISBN 978-0-470-38461-9

[5] *Microsoft MSDN library*. [dostupné online: http://msdn.microsoft.com/en-us/library/ms123401.aspx]

[6] *Web knihovny jQuery*. [dostupné online: http://jquery.com/]

[7] *What Is Windows Communication Foundation*. [dostupné online: http://msdn.microsoft.com/en-us/library/ms731082.aspx]; cit. 7.5.2011

[8] *Async and Parallel Design Patterns in F#: Parallelizing CPU and I/O Computations* [dostupné online: http://blogs.msdn.com/b/dsyme/archive/2010/01/09/async-and-parallel-design-patterns-in-f-parallelizing-cpu-and-i-o-computations.aspx]; cit. 13.5.2011

[9] konzultace: Miloš Dědeček, člen představenstva firmy REENGINE CZ a.s.

[10] konzultace: Tomáš Sotoniak, brněnský ředitel firmy REENGINE CZ a.s.

[11] konzultace: Martin Tlolka, vedoucí vývoje firmy REENGINE CZ a.s.

[12] konzultace: Jaroslav Malík, procesní architekt firmy REENGINE CZ a.s.

Seznam příloh

Příloha 1. CD se zdrojovými texty, a souborem README, ve kterém je postup, jak portál vyzkoušet (v době odevzdání práce bude verze popsána v této práci k vyzkoušení na jednom ze serverů firmy REENGINE).