Síťový MVC framework pro vývoj mobilních aplikací

MVC framework for developing mobile applications

Bc. Jiří Baroš

Diplomová práce
2009

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2008/2009

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jiří BAROŠ**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Síťový MVC framework pro vývoj mobilních aplikací**

Zásady pro vypracování:

1. **V teoretické části práce zmapujte a zanalyzuje přístupy stávajících MVC systémů, jejich vlastnosti, a rozeberte vhodnost a přínosy použití obdobného přístupu v oblasti mobilních aplikací.**
2. **Navrhněte a implementujte síťový MVC (model-view-controller) framework pro použití při vývoji aplikací určených pro mobilní zařízení. Framework umožní aplikačnímu vývojáři definovat sadu formulářů uživatelského rozhraní a definici workflow mezi nimi. Univerzální mobilní klient při připojení k serverové části frameworku dokáže tyto definice interpretovat a poskytovat koncovému uživateli ve formě srovnatelné s nativní mobilní aplikací.**
3. **Pro komunikaci mezi serverovou a klientskou částí navrhněte vhodný protokol s ohledem na optimalizaci datového toku.**
4. **Součástí práce bude implementace jednoduché mobilní aplikace, která bude názorně demonstrovat funkce MVC frameworku.**

*

Rozsah práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **SETH, Ladd, et al. Expert Spring MVC and Web Flow. [s.l.] : [s.n.], c2006. 423 s. ISBN 978-1-59059-584-8.**
2. **JONATHAN, Knudsen. Wireless Java Developing with J2ME, Second Edition. [s.l.] : Apress, c2003. 384 s., 384. ISBN 1590590775.**
3. **ROD, Johnson, et al. Professional Java Development with the Spring Framework. [s.l.] : [s.n.], c2005. 672 s. ISBN 0764574833.**
4. **GRAFF, Mark G., VAN WYK, Kenneth R. Secure Coding: Principles & Practices . [s.l.] : O Reilly, c2003. 224 s. ISBN 0-596-00242-4.**
5. **JOHNSON, Rod, et al. The Spring Framework : Reference Documentation [online]. 2007. 2007 [cit. 2009-02-03]. Dostupný z WWW: [http://static.springframework.org/spring/docs/2.5.x/reference/index.html].**
6. **Spring Framework API 2.5 [online]. 2002-2008 [cit. 2009-02-03]. Dostupný z WWW: [http://static.springframework.org/spring/docs/2.5.x/api/index.html].**

Vedoucí diplomové práce: **doc. Ing. Martin Sysel, Ph.D.**
Ústav aplikované informatiky

Datum zadání diplomové práce: **20. února 2009**

Termín odevzdání diplomové práce: **27. května 2009**

Ve Zlíně dne 13. února 2009

L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Práce se zabývá analýzou techniky vývoje aplikací pomocí návrhového vzoru Model-View-Controller, dostupnými a vhodnými technologiemi pro vývoj vícevrstvých aplikací klient-sever. Cílem je navrhnout Model-View-Controller framework, který by byl vhodný pro vývoj tenkých klientů provozovaných na malých mobilních zařízeních typu mobilní telefon nebo PDA. Práce obsahuje výběr vhodných technologií, zpracování architektury systému i návrh vhodného použití výsledného frameworku. Stěžejní součástí praktické části je plně funkční implementace navrženého systému, detailní popis této implementace a funkční vzorový příklad použití.

Klíčová slova:

MVC, Model-View-Controller, J2ME, Java, Midlet, tenký klient, vícevrstvé aplikace, webové aplikace, klient-server aplikace.

ABSTRACT

The topic of this thesis is to analyze techniques for developing applications with a Model-View-Controller design pattern with appropriate technologies. The goal is to propose a Model-View-Controller framework, which would be proper for developing multi-tier client-server applications with use of Model-View-Controller paradigm, where clients are implemented as thin clients for mobile devices like cell phones or PDAs. This thesis contains study on selection of appropriate technologies suitable for this solution, description of proposed system architecture and example of how to use developed framework. The fundamental part of the work is to create a fully functional implementation of the proposed system and a functional example that shows how framework should be used.

Keywords: MVC, Model-View-Controller, J2ME, Java, Midlet, thin client, lean client, multitier application, web application, client-server application.

Poděkování patří především Pavlu Stržínkovi, který byl odborným konzultantem. Stejně tak patří velký dík Radoslavu Slovákovi za motivaci, rady, recenzování a odbornou diskuzi. Také bych rád poděkoval rodině a blízkým za podporu, která se mi dostala nejen po dobu zpracovávání této práce, ale i po dobu celého studia.

Motto: *If an experiment works, something has gone wrong.*

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval.

V případě publikace výsledků budu uveden jako spoluautor.

Ve Zlíně …….

Podpis diplomanta

OBSAH

ÚVOD

Moderní technologie s sebou přinášejí nové, často dříve netušené možnosti. Byly doby, kdy se zdálo neuvěřitelné, že každý člověk bude mít vlastní mobilní telefon o velikosti deset centimetrů, kterým se během vteřin dovolá na druhou stranu zeměkoule. Dnes se s rozvojem internetu a datových služeb otevírají další možnosti využití těchto mobilních technologií. Samozřejmostí jsou dnes online služby na internetu. Na osobním počítači si můžeme například přečíst zprávy, objednat zboží z e-shopu, zaplatit ho atd.

S rozšířením mobilních telefonů vyvstala potřeba tyto služby zpřístupnit i na nich. Potřebujeme tedy technologie, které nám umožní efektivně vyvíjet a provozovat klient-server aplikace ke kterým bude uživatel přistupovat pomocí mobilního telefonu.

K dispozici je technologie WAP (Wireless Application Protocol), která je určena pro zobrazování obsahu na mobilních telefonech. Tato technologie má ale nedostatky které brání plnému využití mobilních klientů. Například svérázný jazyk WML který není příliš podobný HTML, a tím je méně srozumitelný pro webové vývojáře. Velký problém je špatná a neúplná specifikace WAP standardu a tím nestejná implementace protokolu na zařízeních různých výrobců. Zásadní problém bránící serióznímu využití technologie WAP je absence možnosti komunikaci šifrovat či jinak zabezpečit. Od fundamentálních problémů se odvíjejí další, jako například nedostatek kvalitních nástrojů pro tvorbu WML stránek atd.

Technologie WAP tedy není pro snadnou tvorbu univerzálních na zařízení nezávislých klient-server aplikací příliš vhodná.

Technologie WWW, která také bývá dostupná na moderních mobilních telefonech, je dalším možným kandidátem. WWW je však navržena především pro použití na osobních počítačích a rozdíly v interpretaci webových stránek na různých zařízeních bývají překážkou.

Řešením by mohlo být udržovat složitý model generování HTML kódu ve webové aplikaci v závislosti na klientském zařízení.

Nabízí se tedy další možnost, vytvořit vlastního klienta pro klient-server aplikace. Toto řešení by mělo řešit nedostatky zmíněných technologií, a přidávat další výhody, například

vlastní úroveň zabezpečení (vlastní implementace šifrování atd.), lepši optimalizace toku dat atd.

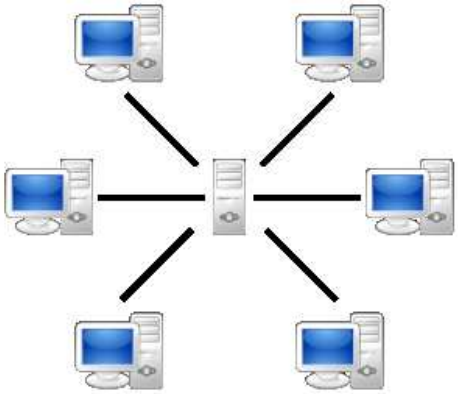
Distribuce, instalace a aktualizace takových aplikací ale nemusí být úplně triviální, nehledě na to, že je potřeba mít vývojáře znalé programování pro danou platformu. Při každé změně v aplikaci je potřeba upravenou aplikaci znovu distribuovat.

Tato práce zkoumá možnost vytvořit aplikační podporu pro vývoj klient-server aplikací. Má umožnit komfortně vyvíjet aplikace, které využívají jako koncové zařízení například mobilní telefon. Serverový vývojář by neměl potřebovat znát techniky programování pro koncové zařízení. Vývojář by měl mít možnost pracovat stejným způsobem jako při vývoji běžných webových aplikací Aplikační podpora má splňovat nároky na moderní přístup k vývoji aplikací, jako jsou vícevrstvá architektura a modulárnost.

I. TEORETICKÁ ČÁST

1 ARCHITEKTURA APLIKACÍ KLIENT-SERVER

Model architektury klient-server rozlišuje serverový systém a klientský systém, které komunikují nejčastěji přes počítačovou síť. Aplikace klient-server je distribuovaný systém, který se skládá jak ze softwaru serverového tak klientského. Nejčastěji iniciuje relaci klient, který se dotazuje serveru, zatímco server čeká na požadavky od klienta.



Obrázek 1 architektura klient-server

Na této architektuře jsou dnes založeny všechny aplikace používající jako médium webové stránky – klientem je zde webový prohlížeč. Služby s touto architekturou jsou také například e-mail, DNS, síťové tiskové služby…

1.1 Tenký versus tlustý klient

Klienty v architektuře klient-server je možné rozdělit na tenké klienty a tlusté klienty.

Tenký klient (nazývaný thin, lean, slim client) obsahuje pouze presentační vrstvu aplikace (bude probráno dále), neobsahuje žádnou logiku či procesní algoritmy, zdroj dat je server, který také obsahuje veškerou logiku aplikace. Výhoda je bezpečnost a menší náchylnost na nekonzistence vlivem komunikace klient-server.

Tlustý klient obsahuje kromě prezentační logiky i logiku funkční a procesní. Jako zdroj dat je použit server. Výhoda může být ve snížení zátěže ne centrálním serveru a výhodnější rozložení prostředků.

Tyto termíny platí pro architekturu klient-server jak na úrovni hardwaru tak softwaru.

1.2 Trendy v komerčním vývoji

Současné požadavky na vývoj software jednoznačně vyžadují přístupy klient-server kvůli centralizaci dat a funkcionality. Dále se s výhodou využívají vícevrstvé architektury kvůli možnosti oddělení velkých bloků a jejich nezávislosti. To umožňuje jednak rozdělení zodpovědností na úrovni vývoje a údržby – každý modul může být vyvíjen a spravován jiným týmem, a také lepší strukturování systému jako takového, například použití různých platforem a technologií, možnost neheterogenního prostředí atd. Tyto přístupy také umožňují mnohem lepší a transparentnější škálování výkonu jednotlivých částí systému.

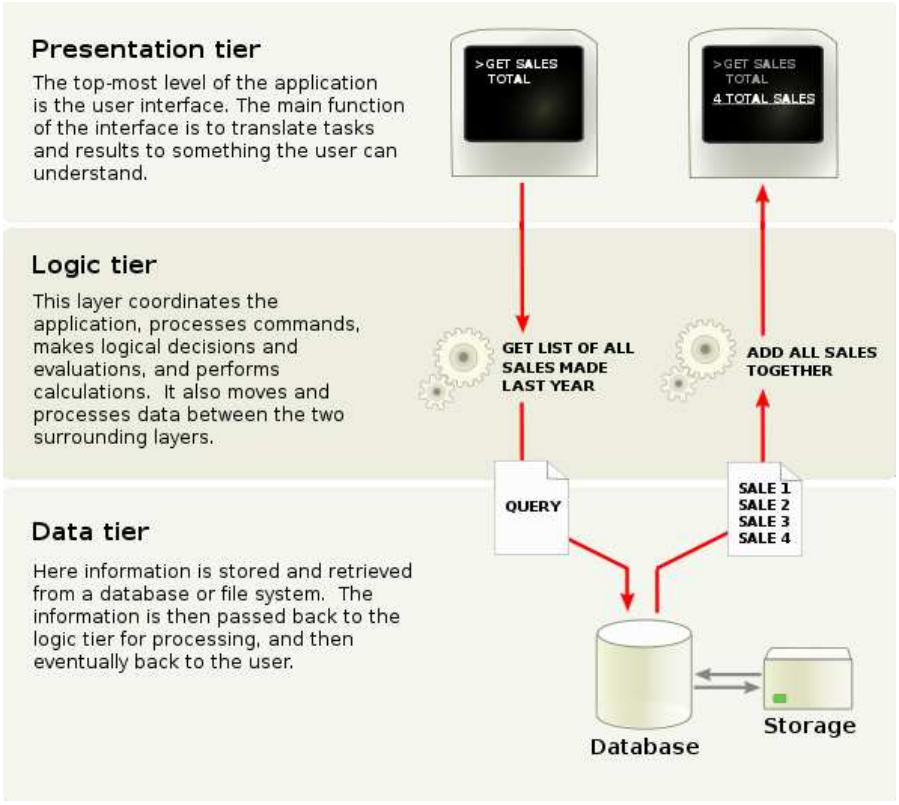
Samozřejmostí je využívání technik a vzorů pro zefektivnění vývoje, například „znovupoužívání“ komponent (D.R.Y. – Don’t Repeat Yourself), zobecnění často řešených problémů knihovními funkcemi a podobně.

1.3 Používané architektury

Z důvodů různé komplexnosti systémů, různých požadavků a náročnosti se systémy navrhují s jinou organizací komponent, jinými procesními toky a podobně. Říkáme, že se používají různé architektury.

1.3.1 Třívrstvá architektura

Třívrstvá architektura je nejpoužívanější u aplikací středního až velkého rozsahu. V této architektuře jsou uživatelské rozhraní, funkční - procesní logika („business logika“) a datové úložiště a operace nad ním, vyvíjeny a udržovány jako nezávislé moduly. Toto umožňuje také například v jednotlivých vrstvách použít různé platformy. Mezi vrstvami-moduly existuje pouze definovaná vazba a moduly neví o vnitřní architektuře jiné vrstvy, díky tomu je možné řešit například škálování, výměnu technologie, nebo kompletní výměnu implementace modulu.



Obrázek 2 třívrstvá architektura[1]

Presentation tier – prezentační vrstva

Nejvyšší vrstva aplikace. Zobrazuje informace relevantní k účelu služeb aplikace jako prohlížení zboží, nakup, zobrazování nákupního košíku. Komunikuje s ostatními vrstvami a zobrazuje výsledky do uživatelského rozhraní nebo ostatních komunikačních rozhraní.

Logic tier – logická/procesní vrstva

Vrstva se stará o funkční algoritmy použité pro zpracování dat z datového zdroje, procesy při zpracování dat, jejich vztahy a přípravu pro prezentační vrstvu.

Data tier – datová vrstva

Vrstva zahrnuje databázové servery, zabezpečuje ukládání a získávání dat. Cílem je mít vyšší vrstvy nezávislé od způsobu ukládání dat, jejich formátu a organizace. Velkou

[1] Obrázek převzat z [9]

výhodou oddělení datové vrstvy je také možnost účinné optimalizace a škálování na úrovni databázového serveru.

Další informace viz [9] – Multitier Architecture.

1.4 Návrhové vzory

Návrhové vzory nejsou součástí programovacího jazyka ani jejich knihoven ačkoli je obvyklé, že pro některé běžně používané vzory bývají dostupné například podpůrné prostředky. Návrhový vzor je obecné řešení pro více problémů podobného typu, jde o specifikaci popisující design řešení konkrétního problému.

Dobrým příkladem mimo softwarové inženýrství může být architektura nebo konstrukce, v obojím se používají vzory, které můžeme ve výsledném produktu vysledovat. Například budova postavené v určitém slohu a podobně.

Návrhový vzor v bodech:

- Standardní řešení obecného programovacího problému
- Technika pro dosažení lepší flexibility kódu dodržením jistých kriterií
- Design nebo implementace struktury, která nám umožní dosáhnout daného cíle
- Vysokoúrovňové (abstraktní) vyjádření programovacího problému
- Zkratka pro vyjádření organizace programu
- Způsob propojení komponent v programu
- Podoba objektového diagramu, modelu.

Návrhové vzory v softwarovém inženýrství dělíme na tyto skupiny:

Creational patterns (vytvářející)

Řeší problémy vytváření objektů, zobecňuje postupy při vytváření objektů daných typů, počtů atd., jde především o problémy které je nutno řešit dynamicky za běhu programu.

Structural patterns (strukturální)

Vzory zaměřující se na uspořádání tříd a komponent systému za účelem zpřehlednění kódu a správné využití strukturalizace kódu pro daný účel.

Behavioral patterns (chování)

Vzory týkající se chování systému případně subsystému či komponent. Většinou na úrovni tříd a objektů s využitím paradigmat objektového programování.

Dále se budu zabývat hlavně strukturálnímu vzory, především vzorem *Model-View-Controller*.

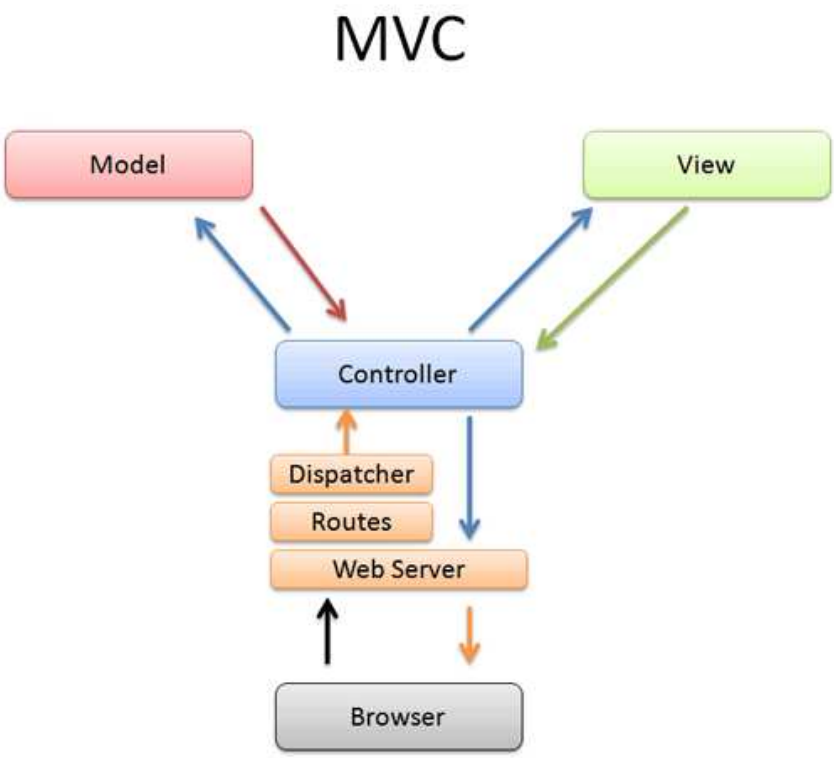
Zdroj: [10] Design patterns.

1.5 Vzor Model-View-Controller

Je strukturální návrhový vzor (někdy se označuje spíše jako architektonický vzor) pro izolaci uživatelského rozhraní od logiky aplikace, cílem je možnost snadných změn do kódu uživatelského rozhraní bez ovlivnění ostatních části aplikace, případně umožnit úplnou výměnu výstupního média – například potřebujeme generovat místo HTML stránek PDF soubory.

Jak napovídá název vzoru, existují tří fundamentální komponenty, které je pro aplikaci vzoru zapotřebí implementovat do řešení:

- **Model** (model), což je doménově specifická reprezentace informací, s nimiž aplikace pracuje. Jde o většinou objektovou reprezentaci dat, které dodává aplikační vrstva prezentační vrstvě.
- **View** (pohled), který převádí data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci uživateli.
- **Controller** (řadič), který reaguje na události (typicky pocházející od uživatele) a zajišťuje změny v modelu nebo v pohledu.



Obrázek 3 návrhový vzor Model-View-Controller

Komponenty řadič a pohled jsou ve standardním rozdělení vrstev na prezentační, doménovou a datovou obvykle zařazovány jako prezentační vrstva. V MVC je tato prezentační vrstva rozdělena mezi komponenty řadič a pohled, nicméně nejdůležitější rozdělení je mezí prezentací a doménovou vrstvou.

Ačkoliv může být koncept MVC realizován různým způsobem, obecně platí tento princip:

1. Uživatel provede nějakou akci v uživatelském rozhraní (např. stiskne tlačítko).
2. Řadič obdrží oznámení o této akci z objektu uživatelského rozhraní.
3. Řadič přistoupí k modelu a v případě potřeby ho zaktualizuje na základě provedené uživatelské akce (např. zaktualizuje nákupní košík uživatele).
4. Model je pouze jiný název pro doménovou vrstvu. Doménová logika zpracuje změněná data (např. přepočítá celkovou cenu, daně a expediční poplatky pro položky v košíku). Některé aplikace užívají mechanizmus pro perzistentní uložení

dat (např. databázi). To je však otázka vztahu mezi doménovou a datovou vrstvou, která není architekturou MVC pokryta.

5. Komponenta pohled použije zaktualizovaný model pro zobrazení zaktualizovaných dat uživateli (např. vypíše obsah košíku). Komponenta pohled získává data přímo z modelu, zatímco model nepotřebuje žádné informace o komponentě View (je na ní nezávislý). Nicméně je možné použít návrhový vzor pozorovatel, umožňující modelu informovat jakoukoliv komponentu o případných změnách dat. V tom případě se komponenta view zaregistruje u modelu jako příjemce těchto informací. Je důležité podotknout, že řadič nepředává doménové objekty (model) komponentě pohledu, nicméně jí může poslat příkaz, aby svůj obsah podle modelu zaktualizovala.
6. Uživatelské rozhraní čeká na další akci uživatele, která celý cyklus zahájí znovu.

Zdroj a více informací: [10] Model-View-Controller, [11].

2 TECHNOLOGIE PRO VÝVOJ KLIENT-SERVER APLIKACÍ

Vývoj aplikací typu klient-server je obecně podpořen ve všech hlavních vývojových prostředcích a nástrojích. Technologie většinou obsahuje sadu podpůrných pro komunikaci a dalších potřebných pro tvorbu distribuovaného systému. Využívají se standardní technologie dostupné pro danou platformu, například protokoly TCP/IP pro komunikaci, standardní prvky uživatelského rozhraní dostupné na platformě klienta a podobně.

2.1 Java

Java je objektově orientovaný programovací jazyk, který vyvinula firma Sun Microsystems a představila 23. května 1995.

Java je jedním z nejpoužívanějších programovacích jazyků na světě. Díky své přenositelnosti je používán pro programy, které mají pracovat na různých systémech počínaje čipovými kartami (platforma JavaCard), přes mobilní telefony a různá zabudovaná zařízení (platforma Java ME), aplikace pro desktopové počítače (platforma Java SE) až po rozsáhlé distribuované systémy pracující na řadě spolupracujících počítačů rozprostřené po celém světě (platforma Java EE). Tyto technologie se jako celek nazývají platforma Java. Dne 8. května 2007 Sun uvolnil zdrojové kódy Javy (cca 2,5 miliónů řádků kódu) a Java bude dále vyvíjena jako open source.

2.1.1 Základní vlastnosti

- jednoduchý – jeho syntaxe je zjednodušenou (a drobně upravenou) verzí syntaxe jazyka C a C++. Odpadla většina konstrukcí, které způsobovaly programátorům problémy, a na druhou stranu přibyla řada užitečných rozšíření.
- objektově orientovaný – s výjimkou osmi primitivních datových typů jsou všechny ostatní datové typy objektové.
- distribuovaný – je navržen pro podporu aplikací v síti (podporuje různé úrovně síťového spojení, práce se vzdálenými soubory, umožňuje vytvářet distribuované klientské aplikace a servery).
- interpretovaný – místo skutečného strojového kódu se vytváří pouze tzv. mezikód (bytekód). Tento formát je nezávislý na architektuře počítače nebo zařízení.

Program pak může pracovat na libovolném počítači nebo zařízení, který má k dispozici interpret Javy, tzv. virtuální stroj Javy - Java Virtual Machine (JVM).

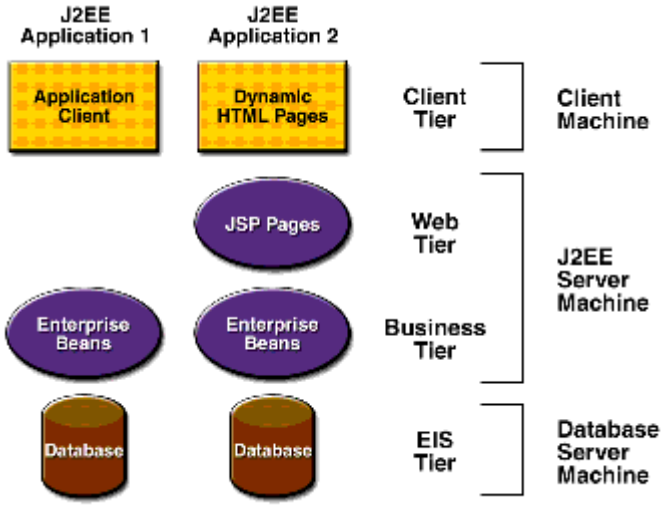
- V pozdějších verzích Javy nebyl mezikód přímo interpretován, ale před prvním svým provedením dynamicky zkompilován do strojového kódu daného počítače (tzv. just in time compilation - JIT). Tato vlastnost zásadním způsobem zrychlila provádění programů v Javě, ale výrazně zpomalila start programů.
- V současnosti se převážně používají technologie zvané HotSpot compiler, které mezikód zpočátku interpretují a na základě statistik získaných z této interpretace později provedou překlad často používaných částí do strojového kódu včetně dalších dynamických optimalizací (jako je např. inlining krátkých metod atp.).
- robustní – je určen pro psaní vysoce spolehlivého softwaru – z tohoto důvodu neumožňuje některé programátorské konstrukce, které bývají častou příčinou chyb (např. správa paměti, příkaz goto, používání ukazatelů). Používá tzv. silnou typovou kontrolu – veškeré používané proměnné musí mít definovaný svůj datový typ.
- Správa paměti je realizována pomocí automatického Garbage collectoru který automaticky vyhledává již nepoužívané části paměti a uvolňuje je pro další použití. To bylo v prvních verzích opět příčinou pomalejšího běhu programů. V posledních verzích běhových prostředí je díky novým algoritmům pro garbage collection a tzv. generační správě paměti (paměť je rozdělena na více částí, v každé se používá jiný algoritmus pro garbage collection a objekty jsou mezi těmito částmi přesunovány podle délky svého života) tento problém ze značné části eliminován.
- bezpečný – má vlastnosti, které chrání počítač v síťovém prostředí, na kterém je program zpracováván, před nebezpečnými operacemi nebo napadením vlastního operačního systému nepřátelským kódem.
- nezávislý na architektuře – vytvořená aplikace běží na libovolném operačním systému nebo libovolné architektuře. Ke spuštění programu je potřeba pouze to, aby byl na dané platformě instalován správný virtuální stroj. Podle konkrétní platformy se může přizpůsobit vzhled a chování aplikace.

- přenositelný – vedle zmíněné nezávislosti na architektuře je jazyk nezávislý, i co se týká vlastností základních datových typů (je například explicitně určena vlastnost a velikost každého z primitivních datových typů). Přenositelností se však myslí pouze přenášení v rámci jedné platformy Javy (např. J2SE). Při přenášení mezi platformami Javy je třeba dát pozor na to, že platforma určená pro jednodušší zařízení nemusí podporovat všechny funkce dostupné na platformě pro složitější zařízení a kromě toho může definovat některé vlastní třídy doplňující nějakou speciální funkčnost nebo nahrazující třídy vyšší platformy, které jsou pro nižší platformu příliš komplikované.
- výkonný – přestože se jedná o jazyk interpretovaný, není ztráta výkonu významná, neboť překladače pracují v režimu „právě včas“ a do strojového kódu se překládá jen ten kód, který je opravdu zapotřebí.
- víceúlohový – podporuje zpracování vícevláknových aplikací
- dynamický – Java byla navržena pro nasazení ve vyvíjejícím se prostředí. Knihovna může být dynamicky za chodu rozšiřována o nové třídy a funkce, a to jak z externích zdrojů, tak vlastním programem.
- elegantní – velice pěkně se v něm pracuje, je snadno čitelný (např. i pro publikaci algoritmů), přímo vyžaduje ošetření výjimek a typovou kontrolu.

Zdroj a více informací: [9] Java.

2.2 J2EE

Java Platform, Enterprise Edition (Java EE, dříve J2EE), je standardizovaná platforma, určená pro vývoj přenositelných, robustních, škálovatelných a bezpečných serverových aplikací v jazyce Java. Rozšiřuje platformu Java SE o podporu pro tvorbu webových aplikací, webových služeb a distribuovaných vícevrstvých aplikací a jejím cílem je poskytnout vývojáři infrastrukturu, která mu usnadní jejich vývoj.



Obrázek 4 vícevrstvá architektura J2EE[2]

Aktuální verzí platformy je Java EE verze 5, která je definována prostřednictvím JCP specifikace JSR 244. Hlavním cílem této verze bylo maximální zjednodušení vývoje pro vývojáře.

Zdroj: [9], další informace: [8].

2.3 Java Beans

Java Bean je znovupoužitelná softwarová komponenta, se kterou je možné manipulovat například ve vizuálním návrháři. Hlavní myšlenka je dát vývojáři k dispozici sadu komponent, které může rychle a jednoduše propojit za účelem vytvoření složitějšího programu bez potřeby ručního psaní kódu.

Softwarové komponenty obecně musí splňovat standardní prostředky pro interakci se zbytkem světa. Například všechny prvky uživatelského rozhraní z knihoven pro tvorbu rozhraní musí dědit určitou třídu a implementovat například její metodu *paint()* aby systém mohl metodu volat pro vykreslení prvku samotného. U Java Beanů není vyžadováno aby

[2] Obrázek převzat z [8]

dědily určitou konkrétní třídu nebo implementovaly specifický interface, ale musí poskytovat podporu pro některé nebo všechny následující klíčové vlastnosti:

- Podpora pro introspekci. Introspekce je proces, při kterém sestavovatel aplikace zjistí všechny vlastnosti, metody a události které daná Java Bean poskytuje a vlastní.
- Podpora pro vlastnosti. Jsou to v podstatě členské proměnné, které ovládají chod a chování Java Beanu.
- Podpora pro přizpůsobení chování Java Beanu.
- Podpora pro události. Mechanismus pomocí, kterého může Java Bean komunikovat s ostatními objekty.
- Podpora pro perzistentní úložiště. Perzistence je schopnost uložit aktuální stav objektu tak aby mohl být později obnoven.

Čerpáno z [8], další informace, tamtéž.

2.4 Servlety a JSP

Technologie Java servetů a JavaServer pages (JSP pages) jsou technologie na straně serveru, které jsou nejrozšířenější ve světě Javových serverových aplikací, staly se standardními prostředky pro vývoj komerčních webových aplikací. Java vývojáři mají tyto technologie rádi z mnoha důvodů, například jednoduchosti, tyto technologie je opravdu jednoduché se naučit a začít používat, výhoda je také že technologie jsou standardizované a pokud je aplikace jednou napsána bude fungovat všude. Ještě důležitější je, že pokud se dodržují osvědčené a doporučené postupy, servlety a JSP stránky pomáhají oddělit prezentaci od obsahu (například pomocí vzoru MVC). Doporučené postupy (best practices) jsou osvědčené přístupy pro vývoj kvalitních, znovupoužitelných a jednoduše udržovatelných aplikací založených na servletech nebo JSP stránkách. Například kousky Java kódu vložené v HTML můžou vyústit ve složitou, špatně čitelnou a efektivně neudržovatelnou aplikaci. „Best practices“ tohle vše můžou změnit.

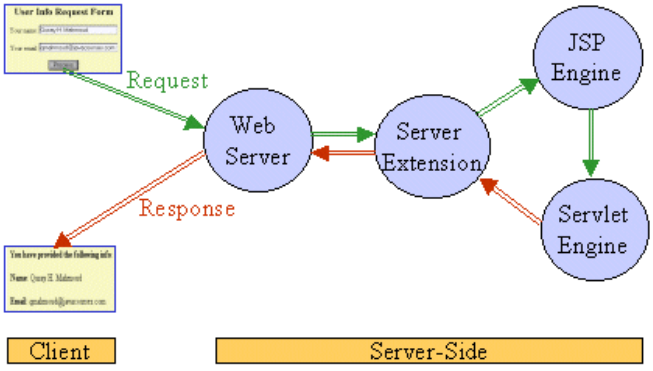
Podobně jako CGI (Common Gateway Interface, viz [9] Common Gateway Interface) skripty, servlety fungují na základě modelu požadavek-odpověď (request-response). Když klient odešle požadavek na server, server přepošle požadavek servletu. Servlet sestaví

odpověď a server ji odešle zpět klientovi. Na rozdíl od CGI, servlety běží v rámci stejného procesu jako HTTP server.

Při zpracovávání požadavku klienta je zavolána metoda služby serveru a jsou předány objekty request a response. Servlet nejdříve rozhodne, jestli je požadavek metodou GET nebo metodou POST, na základě čehož volá danému servletu metodu *doGet* v případě požadavku GET, respektive *doPost* pokud je použita metoda POST. Obě metody přebírají objekty typu *HttpServletRequest* a *HttpServletResponse* reprezentující požadavek a odpověď.

Zjednodušeně, servlety jsou Java třídy, které dynamicky generují obsah (především HTML). Důležité je že servlety běží v kontejneru a API poskytované kontejnerem obsahuje správu sezení (web sessions), správu životního cyklu objektů. Podobně, použitím servletů získáme další výhody platformy Java jako správu bezpečnosti, podporu databází přes rozhraní JDBC, a platformní nezávislost aplikace.

Čerpáno z [8] J2EE 1.4 APIs.



Obrázek 5 tok request/response volání JSP stránky

2.4.1 JSTL

JSTL – JavaServer Pages Standard Tag Library je knihovna zapouzdřující, tagy poskytující základní funkcionality využívanou u většiny JSP aplikací (jedná s např. o využití cyklu a podmínek, nebo podporu práce s XML dokumentem).

Viz [8] JavaServer Pages Standard Tag Library.

2.4.2 Expression language

Expression language je nezbytný technologický prvek, jenž musí být implementován v JSP, tak aby bylo možno efektivně zpracovávat výrazy skriptovacího jazyky a propojit skriptovací prostředí s dalšími součástmi Java platformy (např. s JavaBeans, Servlety či knihovnami značek). Konkrétně se pak jedná především o předávání hodnot prostřednictvím atributů značek (např. právě u některé značky JSTL) a následného zpřístupnění výsledku zpracování (opět např. některé značky JSTL) ve skriptovacím prostředí. V současné verzi JSP však zatím není vnitřně zabudována podpora vybraných expression languages (rozhodnutí jaké EL budou v JSP podporovány a jak mají být vnitře integrovány je předmětem JSR152 expert group), takže jako provizorní řešení byla zvolena externí podpora, kdy je možno použít libovolný EL (výběr jazyku lze učinit jednak pro celou aplikaci prostřednictvím kontextového parametru javax.servlet.jsp.jstl.temp.ExpressionEvaluatorClass, nebo lokálně uvnitř značky <expressionLanguage>), který má implementováno příslušné API (nyní SPEL - Simply Possible Expression Language for JSTL a EcmaScript - standardizovaná verze JavaScriptu dle ECMA-262).

Viz [8] JavaServer Pages Technology/Expression language.

2.4.3 Implementace vlastních JSP značek

Vkládání částí Java kódu do HTML dokumentů nemusí být vhodné, pokud chceme dodržet oddělení logiky aplikace od vzhledu (prezentační vrstvy), také vývojáři tvořící uživatelské rozhraní nemusí být znalí programování. Technologie JavaBeans která se používá k zabalování většiny nutného kódu, toto částečně řeší. Její používání v JSP stránkách však stále vyžaduje určitou znalost programování v Javě.

Technologie JSP umožňuje zavést si své vlastní nové uživatelské značky díky „knihovnám značek“. Jako Java programátor můžu rozšířit JSP stránky o nové moje vlastní značky které můžou být nasazeny a používány se syntaxí podobou jako HTML. Uživatelské značky umožňují lepší strukturalizaci na balíky a vylepšují oddělení aplikační logiky od prezentační vrstvy. Kromě toho poskytují prostředky k úpravě prezentační vrstvy, pokud toho není možné dosáhnout snadno pomocí JSTL.

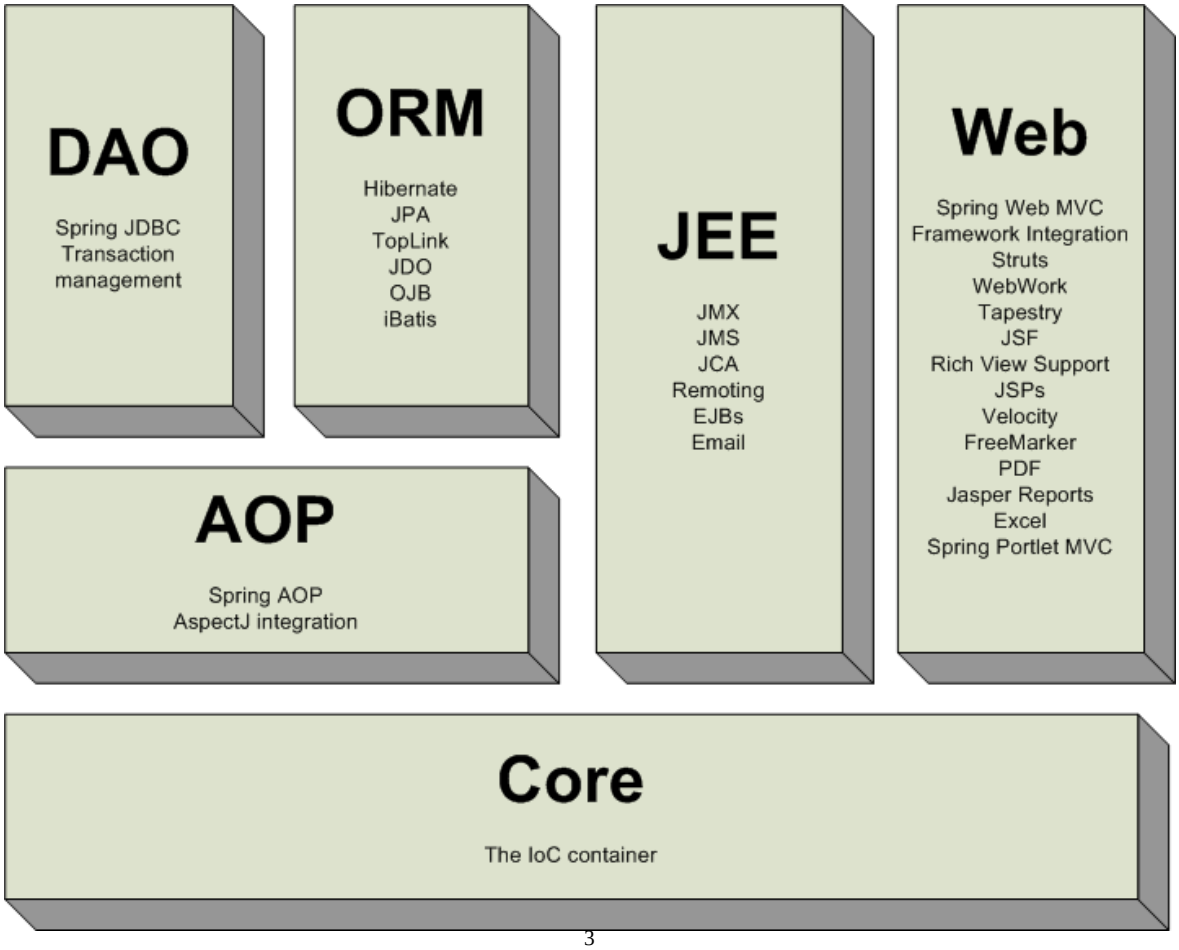
Výhody vlastních JSP značek:

- Eliminují skriptlety (kousky Java kódu) v JSP souborech, všechny potřebné parametry jsou zadány jako XML atributy značky, nebo do těla značky. Žádný další kód není potřeba pro inicializaci nebo nastavování vlastností komponent.
- Mají jednodušší syntaxi, skriptlety jsou psány v jazyce Java zatímco JSP značky se používají jako XML elementy, podobně jako HTML syntaxe.
- Můžou zlepšit produktivitu vývojářů obsahu dodáním nových funkcí kterých není možné dosáhnout jen s pomocí HTML.
- Jsou znovupoužitelné. Šetří tím čas vývoje a testování. Skriptlety nejsou znovupoužitelné, jedině pokud technologii „copy-paste“ nenazveme „znovupoužitím“.

Viz [8] JavaServer Pages Technology/ Custom Tags in JSP Pages.

2.5 Aplikační framework Spring

Spring je open-source J2EE aplikační framework šířený pod licencí Apache 2.0. Mezi open-source softwarem však zaujímá poněkud výstřední postavení – zrodil se totiž z knížky. V roce 2002 vydal Rod Johnson knihu Expert One-on-One J2EE Design and Development, v níž nejen že popsal své zkušenosti s problémy při tradičním J2EE vývoji, ale také uveřejnil 30 tisíc řádek zdrojového textu frameworku Interface21, jehož se na začátku dalšího roku chopila početná komunita vývojářů v čele s Rodem Johnsonem a Juergenem Hoellerem.



[3]

Obrázek 6 Spring framework

Jak snadno nahlédneme z obrázku, Spring je velmi rozsáhlý a pokrývá širokou škálu technologií. Má však dvě základní vlastnosti: je modulární a neinvazivní. Znamená to, že si můžeme ze Springu vybrat to, co se nám právě hodí, bez nutnosti používat zbytek, a náš program (zejména vrstva aplikační logiky) je na frameworku maximálně nezávislý. Spring prosazuje některé aspekty vývoje softwaru, které můžeme bez obav nazvat agilními: jde zejména o koncepty Inversion of Control/Dependency Injection, kterým se budeme podrobněji zabývat níže, a AOP, kterým se zde zabývat nebudeme. Základní stavební jednotkou Springu je rozhraní – programátor by neměl být nucen k dědičnosti, která jeho aplikaci velmi těsně váže k používanému frameworku. Ze stejného důvodu pak bývá

[3] Obrázek převzat z [5]

implementace tvořena znovupoužitelnými POJO třídami. (Obojí platí zejména pro doménové třídy a třídy aplikační logiky. Pomocné třídy naopak od tříd frameworku dědí poměrně často a bezproblémově, jak uvidíme později.) Obecně Spring tíhne ke kvalitní architektuře softwaru, typicky třívrstvé, a tak v textu předpokládáme, že čtenář aspoň tuší, co to jsou např. doménové objekty, persistenční, aplikační a prezentační vrstva a podobně.

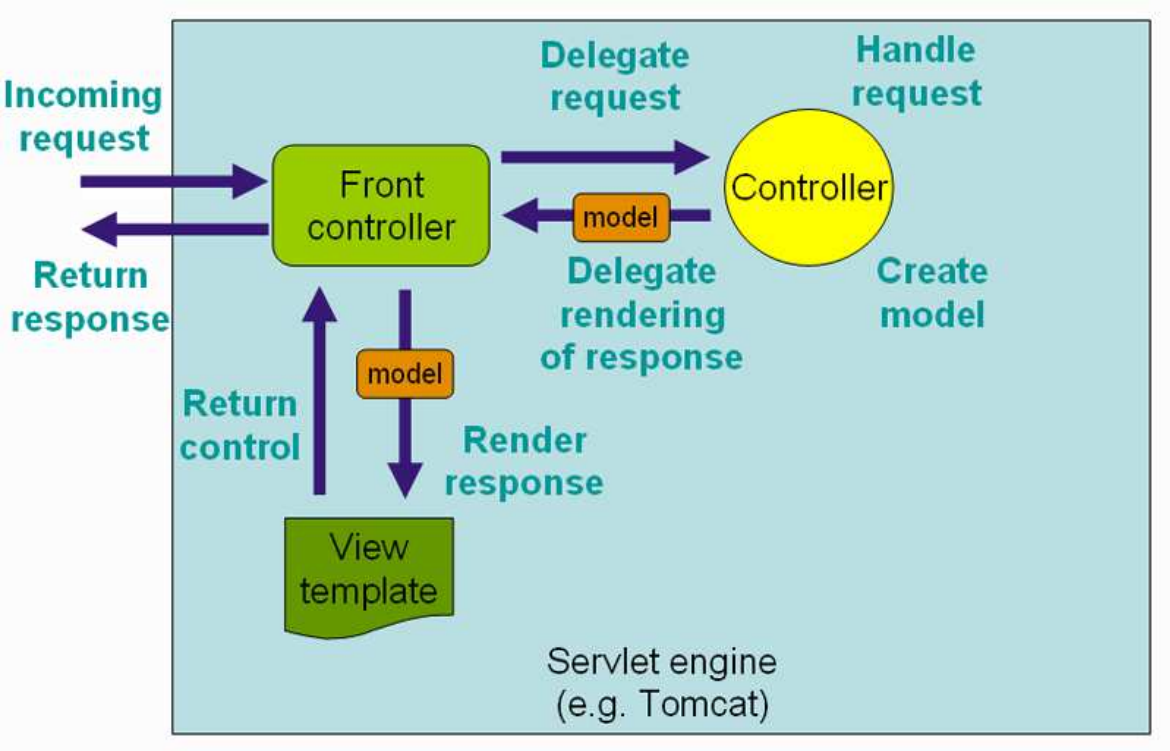
Spring se nesnaží „objevovat kolo“, často poskytuje jednotné rozhraní k několika implementacím nějakého konceptu, a tak je možné Spring provázat s řadou ORM nástrojů, MVC frameworků či EJB. Ve chvíli, kdy je to nezbytné, však neváhá přijít s vlastní implementací, a tak máme k disposici např. deklarativní transakce, remoting či messaging, a to vše pro naše POJO třídy, které vůbec netuší, že nějaký Spring existuje. Jednou z těchto vlastních implementací daného konceptu je i náš cíl, Spring Web MVC.

2.5.1 Dependency injection

Inversion of Control/Dependency Injection (IoC/DI) je technika, se kterou je Spring identifikován nejčastěji. Jde o velmi jednoduchý koncept, který by se nejsnáze dal vyjádřit větou: „Nevolej mě, já zavolám tebe.“ Slouží nám k tomu, abychom se zbavili vzájemných vazeb mezi jednotlivými částmi (vrstvami) aplikace vyjádřených ve zdrojovém textu a také vazeb na aplikační framework.

2.5.2 Spring Web MVC

Spring Framework obsahuje jeho vlastní MVC framework, ten nebyl původně pro projekt plánován. Vývojáři Springu se ale rozhodli napsat svůj vlastní MVC, protože populární webový framework Jakarta Struts se jim zdál špatně navržený, a také kvůli nedostatkům v ostatních frameworcích. V zásadě se jim zdálo, že je nedostatečně oddělena prezentační vrstva a vrstva starající se vyřizování požadavků (řadič), stejně jako model a řadič.



Obrázek 7 struktura Spring Web MVC[4]

Spring MVC je framework založený na požadavcích. Framework definuje strategické rozhraní pro všechny klíčové úkoly, které musí být moderním frameworkem. Cílem je udržet rozhraní jednoduché s jasným účelem, tak aby bylo umožněno vývojářům psát vlastní implementace, pokud chtějí. Všechny rozhraní jsou úzce spjaty se Servlet API, to někteří vnímají jako problém, protože cílem mělo být zavést ještě větší míru abstrakce pro webové aplikace. Nicméně tato spjatost se Servlet API umožňuje používat zároveň jak větší míru abstrakce tak i nízkoúrovňové Servlet API.

Třída ***DispatcherServlet*** je hlavní řadič MVC frameworku a je zodpovědná za delegaci obsluhu požadavků na příslušná rozhraní v průběhu spouštění fází zpracování HTTP požadavku.

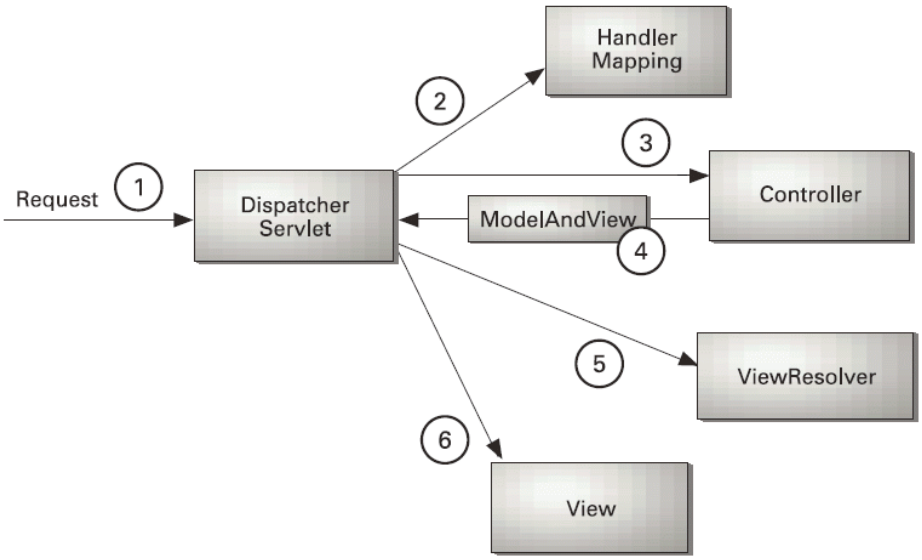
[4] Obrázek převzat z [5]

Nejdůležitější rozhraní definovaná Spring MVC a jejich zodpovědnosti:

- ***HandlerMapping***: vybírá objekty (handlery), které zpracují příchozí požadavky v závislosti na parametru nebo podmínce vnitřní či vnější.
- ***HandlerAdapter***: spuštění objektů které zpracovávají příchozí požadavky.
- ***Controller***: patří mezi Model a Pohled (View) a stará se o příchozí požadavky směruje je na správnou odpověď.
- ***View***: zodpovědná za vrácení odpovědi klientu.
- ***ViewResolver***: vybírá pohled na základě logického názvu pohledu.
- ***HandlerInterceptor***: zachycení a "mezizpracování" příchozího požadavku předtím než je poslán řadiči.
- ***LocaleResolver***: řeší lokalizace jednotlivých uživatelů
- ***MultipartResolver***: usnadňuje uploadování souborů na server

Protože Spring MVC používá kontejner Springu pro konfiguraci a propojení svých vlastních částí, webová aplikace může využít všech výhod přístupu Inversion of control který kontejner nabízí.

Obrázek ukazuje pořadí, v jakém jsou spuštěny jednotlivé fáze zpracování požadavku.



Obrázek 8 pořadí operací při zpracování požadavku ve Spring MVC

3 MOŽNOSTI MOBILNÍCH ZAŘÍZENÍ

Primárním cílem práce je vytvořit aplikační podporu pro vývoj vícevrstvých aplikací typu klient-server, kde klientem bude tenký klient v malém zařízení, typicky mobilní telefon, zařízení PDA a podobně.

Podívejme se tedy na možnosti těchto zařízení a způsoby vývoje aplikací pro tyto zařízení. Důležitým faktorem je taky rozšířenost technologie/platformy, tenký klient by mělo být možné spustit na co největším počtu zařízení a technologie by měla být pokud možno standardní.

3.1 Java Micro Edition

Jde o variantu platformy Java (viz 2.1 Java) pro malé zařízení. Obsahuje většinu standardních prostředků a další zjednodušené pro použití s těchto malých zařízeních. Jde o poměrně standardní technologii a dnes je dostupná na drtivé většině mobilních telefonů a dalších zařízeních.

Java ME se dělí dle API do dvou základních konfigurací:

Connected Limited Device Configuration (CLDC)

Především pro mobilní telefony a PDA s malým množstvím paměti, dnes například dostupná ve většině mobilních telefonů. Obsahuje omezenou sadu vestavěných tříd, čili omezenou množinu základní funkcionality, pro vývoj základních aplikací s ohledem na možnosti zařízení ale dostačující.

Connected Device Configuration (CDC)

Pro zařízení které vyžadují kompletní Java virtual machine, dostupnost API se může lišit dle konkrétního profilu, typicky je pro daný profil dostupná jen množina API v závislosti na účelu profilu.

3.2 .NET Compact Framework

Microsoft .NET Compact Framework (.NET CF) je verze .NET Frameworku navržená speciálně pro běh na zařízeních založených na systému Windows CE, což je systém pro mobilní nebo vestavěné zařízení jako jsou zařízení PDA, mobilní telefony, průmyslové terminály, set-top boxy a podobně. .NET Compact Framework používá některé knihovny společné s „velkým“ .NET Frameworkem a také některé upravené speciálně pro použití na systému Windows CE. Knihovny ale nejsou přesné kopie knihoven z .NET Frameworku ale je u nich optimalizována velikost pro menší prostorou náročnost na malých zařízeních.

Aplikace využívající .NET Compact Framework je možné vyvíjet v prostředí Microsoft Visual Studio .NET 2003 a novějším, nebo bez pokročilých nástrojů Visual Studia jen s překladačem z SDK které je možné získat od společnosti Microsoft.

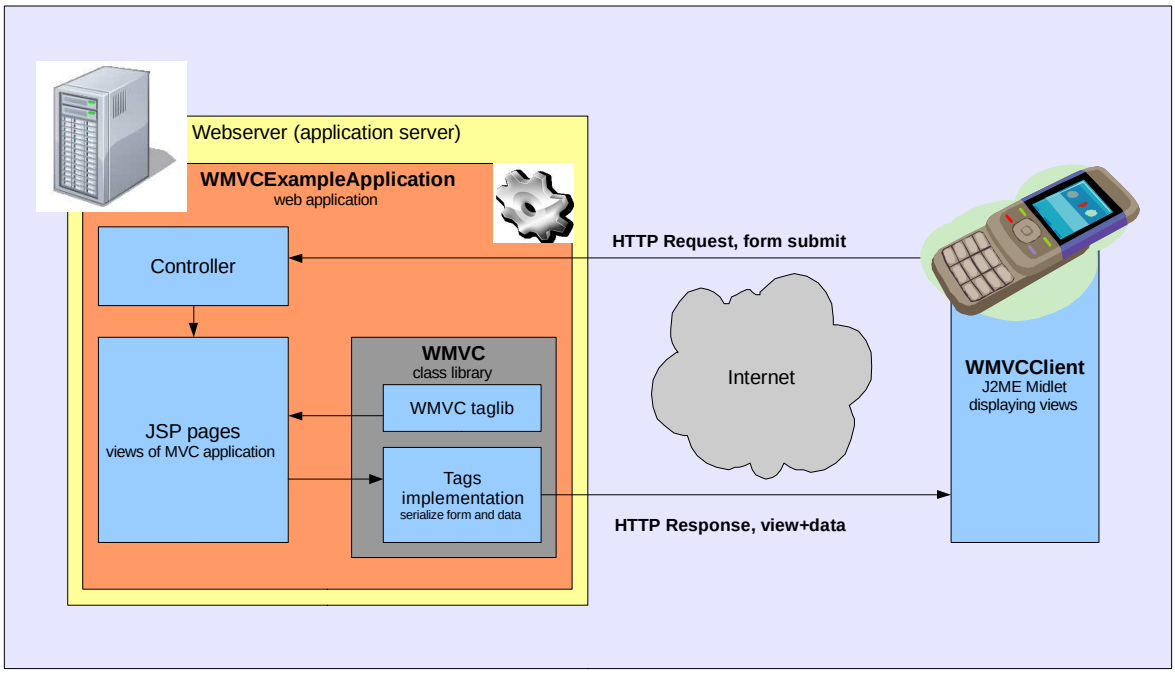
Aby se daly aplikace využívající .NET Compact Framework provozovat musí platforma podporovat Microsoft .NET Compact Framework runtime. Operační systémy které podporují .NET Compact Framework jsou Windows CE 4.1, Microsoft Pocket PC 2002 a novější, Smartphone 2003 a novější, Windows Mobile. Programy pro .NET Compact Framework je možné spustit také na systémech Microsoft Windows s nainstalovaným plným .NET Frameworkem (pro PC) protože .NET Compact Framework obsahuje jen menší množinu knihoven z plného frameworku.

II. PRAKTICKÁ ČÁST

4 ARCHITEKTURA ŘEŠENÍ

Na základě požadavků na řešení, možností byla navržena tato struktura:

- Pro stranu serveru vytvořit knihovnu pro tvorbu pohledů (pro použití MVC vzoru), umožňující integraci do Spring MVC, případně jiného frameworku používajícího pro pohledy technologii JSP
    - Knihovna má obsahovat značky pro základní prvky uživatelského rozhraní.
    - Knihovna se bude starat o přípravu dat pro mobilního (tenkého) klienta
- Tenký klient pro mobilní zařízení bude zobrazovat pohledy definované na serveru.
    - Komunikace standardními kanály (HTTP)



Obrázek 9 architektura systému

4.1 Serverová podpora

Je realizována jako knihovna značek pro použití v JSP souborech. JSP soubory jsou ve Spring MVC používány jako technologie pro pohledy (view). Popis implementace knihovny WMVC v kapitole 5 Implementace serverové části.

4.2 Komunikace klient-server

Jako hlavní komunikační protokol byl zvolen protokol HTTP, kvůli jednoduchosti rozšířenosti, dostupnosti a ověřené implementaci na platformách serveru i klienta. Jelikož řešení je navrženo pro použití modelu požadavek-odpověď, nabízí se HTTP jako zřejmý kandidát. Implementace tímto protokolem navíc umožňuje velmi jednoduše použít jeho šifrovanou variantu HTTPS. Dalším důležitým faktorem je i to, že protokol HTTP je obvykle povolen a nedochází k problémům z důvodu blokování komunikace poskytovatelem a podobně.

4.3 Mobilní klient

Mobilní klient je tenký klient, který se stará o zobrazení pohledu definovaného na serveru. Klient musí obsahovat komunikační část, která přijme pohled i data, na základě toho vytvoří uživatelské prostředí, které zobrazí uživateli. Zabezpečí interakci uživatele s rozhraním a odešle potvrzené akce zpět na server.

5 IMPLEMENTACE SERVEROVÉ ČÁSTI

Implementace je realizována jako knihovna značek pro použití v JSP souborech, které použijeme v samotné aplikaci pro definici Views. Definice značek i jejich implementace je zapouzdřena do jediného souboru JAR. Při používání knihovny ve vlastní aplikaci tedy stačí linkovat jedinou knihovnu. Knihovna je nazvána **WMVC**, celý název je **WirelessMVC**.

5.1 Požadavky na implementaci

Následující prvky uživatelského rozhraní byly implementovány:

- **Checkbox**: pro zadávání hodnoty ano/ne
- **Divider**: oddělovač logických částí formuláře
- **Form**: formulář obsahující samotné prvky
- **InputText**: pro vkládání textu a čísel
- **Link**: přechod na jiný formulář obrazovku bez odeslání formuláře
- **OutputText**: pro zobrazení statického textu
- **Select**: pro výběr z více možností
- **Submit**: pro odeslání formuláře

5.2 Sada značek WMVC

Řešení se skládá ze dvou základních částí:

- TLD (Tag Library Descriptor) XML soubor, osahující definici značek, jejich atributy a specifikaci handleru – plně kvalifikovaný název Java třídy, která obsahuje implementaci značky
- Třídy jazyka Java obsahující implementaci značek

Tabulka 1 značky a implementační třídy

Název elementu	**Implementující třída**	**poznámka**
`form`	`WirelessMvc.tags.FormTag`	Může obsahovat všechny následující
`inputText`	`WirelessMvc.tags.InputTextTag`	
`outputText`	`WirelessMvc.tags.OutputTextTag`	
`checkbox`	`WirelessMvc.tags.CheckboxTag`	
`link`	`WirelessMvc.tags.LinkTag`	
`submit`	`WirelessMvc.tags.SubmitTag`	
`divider`	`WirelessMvc.tags.DividerTag`	
`select`	`WirelessMvc.tags.SelectTag`	Obsahuje objekty `selectItem`
`selectItem`	`WirelessMvc.tags.SelectItemTag`	

Pro kompletní přehled značek a implementací viz tabulku v příloze P III: Seznam značek a jejich implementace.

5.2.1 TLD – definice značek

V TLD souboru jsou definovány všechny elementy ze seznamu Tabulka 1 značky a implementační třídy.

wmvctags.tld:

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.0" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/
                        web-jsptaglibrary_2_0.xsd">

  <tlib-version>1.0</tlib-version>
  <short-name>wmvc</short-name>
  <uri>http://edhouse.cz/j2ee/comps/wmvc/wmvctags</uri>

  <tag>
    <name>form</name>
    <tagclass>WirelessMvc.tags.FormTag</tagclass>
    <info></info>
    <bodycontent>JSP</bodycontent>

    <attribute>
```

```
      <name>name</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>

    <attribute>
      <name>rendered</name>
      <required>false</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>

    <attribute>
      <name>visible</name>
      <required>false</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>

    <attribute>
      <name>title</name>
      <required>false</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>

  …
```

(výpis je zkrácen)

Kořenový element *taglib* obsahuje tyto sub-elementy (pokud není uvedeno jinak, element může být přítomen právě jednou):

- *tlib-version*: verze knihovny značek
- *short-name*: krátký název knihovny
- *uri*: URI (Uniform Resource Identifier) které jednoznačně identifikuje knihovnu značek.
- *tag*: definice značky, viz dále. Těchto elementů může být v TLD souboru více.

Element *tag* obsahuje tyto sub-elementy:

- *name*: unikátní název značky
- *tagclass*: Java třída, která danou značku implementuje, třída musí být rozšířením třídy *TagSupport*.
- *info*: informativní text který se zobrazí například v nápovědě vizuálních editorů JSP souborů

- *bodycontent*: určuje, jak je zpracováno tělo značky.
    - *empty*: element neobsahuje tělo.
    - *JSP*: tělo obsahuje JSP značky a bude zpracováno.
    - *tagdependent*: tělo nebude zpravováno, jen překopírováno jako prostý text.
- *attribute*: definuje atribut značky, má tyto sub-elementy. Těchto elementů může být více:
    - *name*: název atributu
    - *required*: *true* pokud je atribut povinný
    - *rtexprvalue*: *true* pokud má atribut akceptovat expression langure, pro Expression Language viz [8]

5.2.2 Implementace značek

Pro kompletní diagram tříd viz Příloha P I: Class diagram implementace knihovny WirelessMVC. Pro seznam tříd implementujících elementy viz Tabulka 1 značky a implementační třídy.

Popis implementace jednotlivých prvků:

Společný předek *AbstractWMVCTag* obsahuje tyto vlastnosti:

- *name*: název prvku, mapováno jako atribut značky v JSP souboru. Tato hodnota musí být ve formuláři unikátní.
- *value*: hodnota prvku, mapováno jako atribut značky v JSP souboru.
- *additionalProperties*: vektor dalších vlastností, které přidají rozšiřující třídy (neboli konkrétní prvky), tyto vlastnosti budou serializovány a textová data budou escapována (viz 5.3 Serializace dat pro odeslání klient).
- *additionalPropertiesRaw*: vektor dalších vlastností z rozšiřujících tříd, textová data nebudou escapovány.

- *numericItemType*: číselné vyjádření typu prvku, použito při serializaci a deserializaci, každá třída rozšiřující *AbstractWMVCTag* musí vracet unikátní hodnotu.

Všechny elementy jsou potomky třídy *AbstractWMVCTag*, všechny tedy obsahují tyto klíčové vlastnosti.

Třídu *AbstractWMVCTag* dále rozšiřuje třída *AbstractWMVCFormItemTag*, tuto třídu rozšiřují všechny prvky, jejichž vlastnost *value* je upravitelná na klientu a má se odeslat zpět na server. Tyto prvky musí být umístěny uvnitř elementu *form*.

Všechny zdrojové texty jsou uloženy na přiloženém CD.

5.2.2.1 form

Element, obsahující všechny prvky, které obsahují vstup, jejich obsah bude odeslán z klienta na server.

5.2.2.2 inputText

Prvek pro zadávání libovolného textu. Typ dat je *String*. Třída je potomkem *AbstractWMVCFormItemTag*, prvek musí tedy být uvnitř elementu form a jeho vlastnost *value* se odesílá z klienta na server. Prvek obsahuje vlastnost *title*, do které je možno vložit text, ten se zobrazí uživateli jako vysvětlení prvku.

5.2.2.3 outputText

Prvek zobrazující statický text zadaný vlastností *value*. Typ dat je *String*.

5.2.2.4 checkbox

Prvek pro zadávání logické hodnoty ano-ne. Realizován jako zaškrtávací pole. Typ dat je Boolean. V JSP souboru budou v atributu *value* rozpoznány hodnoty *true* pro řetězec *„true“* a *false* pro *„false“*. Prvek obsahuje vlastnost *title*, do které je možno vložit text, ten se zobrazí uživateli jako vysvětlení prvku. Třída je potomkem *AbstractWMVCFormItemTag*.

5.2.2.5 link

Prvek realizující odkaz pro změnu aktuálního formuláře, vlastnost *value* je použita jako adresa v aplikaci (navigace jen v kontextu aplikace, ne mimo ni). Typ dat je *String*. Vlastnost *title* může obsahovat vysvětlení prvku.

5.2.2.6 submit

Prvek implementující akci, akcí se rozumí odeslání aktuální formuláře, včetně dat zadaných uživatelem, na server. Při odeslání bude pro hodnotu prvku *submit* samotného použita hodnota definovaná vlastností *value*, pokud není vlastnost *value* definována bude nastavena hodnota *„true“*. Třída je potomkem *AbstractWMVCFormItemTag*. Vlastností *title* je možno nastavit text se kterým se daná akce objeví v menu aplikace.

5.2.2.7 divider

Prvek sloužící pro oddělení logických celků formuláře. Neobsahuje žádnou funkcionalitu.

5.2.2.8 select

Prvek realizující výběr jedné možnosti z několika. Třída implementující značku obsahuje vlastnost *selectItems* – vektor položek *selectItem*. Element může obsahovat zanořené elementy *selectItem* které přestavují jednotlivé možnosti. Vlastnost value je hodnota vlastnosti value vybraného zanořeného elementu selectItem. Třída je potomkem *AbstractWMVCFormItemTag*. Vlastnost *title* je možno využít pro vysvětlující text.

5.2.2.9 selectItem

Prvek implementuje jednu položku z elementu *select*. Element *selectItem* musí být uvnitř elementu *select*, pokud použit jinak, je vyhlášena chyba. Třída je potomkem *AbstractWMVCFormItemTag*. Hodnota vlastnosti *title* je textový popis který bude zobrazen v menu. Vlastnost *value* je hodnota položky která bude nastavena do elementu *select* pokud uživatel vybere jednu položku ze seznamu. Typ dat vlastnosti value je *Integer*.

5.3 Serializace dat pro odeslání klientu

Serializace v kontextu datových úložišť a přenosu dat, znamená proces konverze objektů na sekvenci bitů tak aby data mohly být uloženy na záznamové médium (soubor, paměťový buffer), nebo přeneseny přes síťové spojení. Opačným procesem, nazývaným **deserializace**, můžeme obnovit původní objekt do stavu, který byl serializován - půjde o identickou kopii.

Serializace dat pro odeslání mobilnímu klientu je implementována ve třídě *AbstractWMVCTag*.

5.3.1 Formát serializovaných dat

Data jsou serializovány do formy holého textu se speciální syntaxí pro popis jednotlivých prvků.

Prvky jsou odděleny značkou konec řádku – ASCII 10 DEC. Popis prvku začíná typem informace na řádku:

- *FORM:BEGIN* začátek formuláře
- *ITEM* položka formuláře
- *FORM:END* ukončení formuláře.

Informace pokračuje výčtem parametrů, každý parametr začíná znakem „#“ (ASCII 35 DEC) a má formát *název=hodnota*.

Jak bylo řečeno, formulář je ohraničen značkami *FORM:BEGIN* a *FORM:END*. Formulář musí obsahovat parametr *name*, všechny prvky formuláře jsou uvnitř dvojice značek *FORM:BEGIN* a *FORM:END*.

Jednotlivé prvky jsou rozlišeny pomocí parametru *type*, tento parametr je klíčový při deserializaci, zaručuje vytvoření správného prvku z těchto dat. Pro přehled typů viz

Tabulka 2 kódy typů použité při serializaci a deserializaci. Další parametry odpovídají parametrům značek popsaným v kapitole 5.2.2 - Implementace značek.

Tabulka 2 kódy typů použité při serializaci a deserializaci

Prvek	**Kód typu prvku**
`inputText`	1
`checkBox`	2
`outputText`	3
`link`	4
`divider`	5
`select`	6
`command`	10

5.3.1.1 Serializace vnořených prvků

Vnořené prvky jsou prvky, které jsou obsaženy v jiném nadřazeném elementu, v aktuální implementaci jde především o položky *selectItem*, které jsou obsaženy v elementu *select* a představují jednotlivé položky seznamu pro výběr. Vnořené prky jsou serializovány následujícím způsobem:

Do parametru *items* je každá položka zapsána dle schématu *hodnota!titulek* („!“ ASCII 33 DEC), jednotlivé vnořené položky jsou odděleny znakem “|” (svislá čára, ASCII 124 DEC).

```
ITEM#type=6#name=box#value=#title=#items=1!OptA|2!OptB|
```

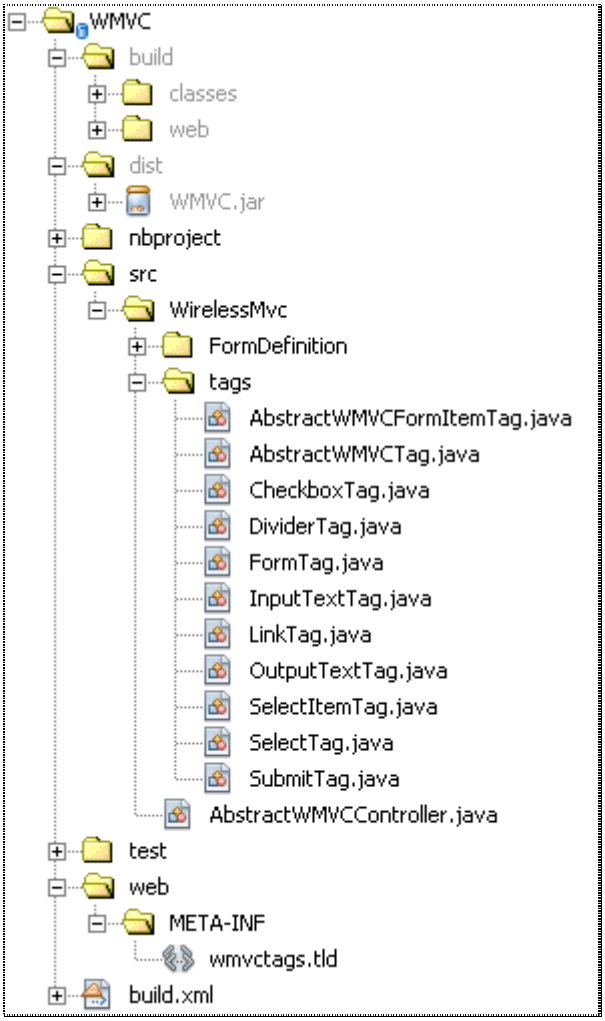
V tomto příkladu máme dvě zanořené položky: *OptA* s hodnotou 1 a *OptB* s hodnotou 2.

5.3.1.2 Příklad serializovaného formuláře

```
FORM:BEGIN#name=form1

ITEM#type=3#name=outText1#value=Hello World

ITEM#type=6#name=selectBox1#value=#title=Choose#items=10!Opti
on A|20!Option B|

ITEM#type=1#name=userName#value=#title=Your name:

ITEM#type=2#name=checkBox1#value=false#title=Yes

ITEM#type=10#name=submit#value=true#title=Save

FORM:END
```

5.4 Struktura výsledné knihovny

Knihovna WMVC je vytvořena jako projekt typu Class library v prostředí NetBeans.



Obrázek 10 Struktura projektu knihovny WMVC

Jednotlivé adresáře popíši v abecedním pořádku, jak jsou na obrázku:

build:

obsahuje výsledek posledního překladu, tedy Java třídy přeložené do bytekódu (soubory .class), a také podadresář web/META-INF do kterého je zkopírován soubor s definicí JSP značek. Viz kapitolu 5.2.1 - TLD – definice značek.

dist:

Obsahuje výsledný soubor JAR, který je vytvořen jako archiv z obsahu adresáře *build*.

nbproject:

projektové soubory prostředí NetBeans, nastavení, projektu a podobně.

src:

Soubory se zdrojovými kódy jsou uloženy v adresáři *src*, podadresáře odpovídají dle zvyklostí Javy „balíkům“ (packages). Například implementace JSP značek jsou uloženy v adresáři *src/WirelessMVC/tags/*, protože implementační třídy spadají do balíku *WirelessMVC.tags*.

web:

v projektu slouží jako umístění pro TLD soubor *wmvctags.tld*, který je při sestavování knihovny automaticky překopírován do adresáře *build*.

build.xml

skript pro sestavení projektu vytvořený prostředím NetBeans, kromě kompilace Java tříd provádí překopírování souboru *wmvctags.tld* do adresáře *build* aby se soubor dostal na správné místo do výsledné knihovny JAR a byl dostupný při používání knihovny v jiných projektech.

Skripty *build.xml* a *nbproject/build-impl.xml* jsou založeny na syntaxi nástroje Apache Ant (viz [7]).

Přidání souboru *wmvctags.tld* do distribuce:

build.xml:

```
…
<target name="-pre-jar">
  <property name="build.meta.inf.dir" value="${build.dir}/web/META-INF"/>
  <mkdir dir="${build.meta.inf.dir}" />
  <copyfile src="web/META-INF/wmvctags.tld"
            dest="${build.meta.inf.dir}/wmvctags.tld" />
</target>
…
```

Úprava v *nbproject/build-impl.xml*:

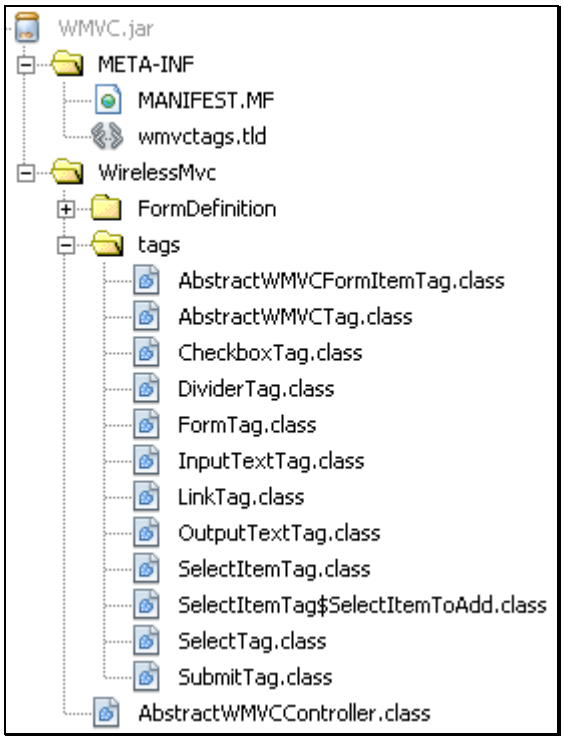
Přidání adresáře web do obsahu JAR souboru.

```
…
<jar compress="${jar.compress}" jarfile="${dist.jar}">
  <j2seproject1:fileset dir="${build.classes.dir}"/>
  <j2seproject1:fileset dir="${build.dir}/web"/>
```

```
</jar>
…
```

Více informací o nástroji Apache Ant pro sestavovací skripty v [7].

Výsledný sestavený JAR soubor má tuto strukturu:

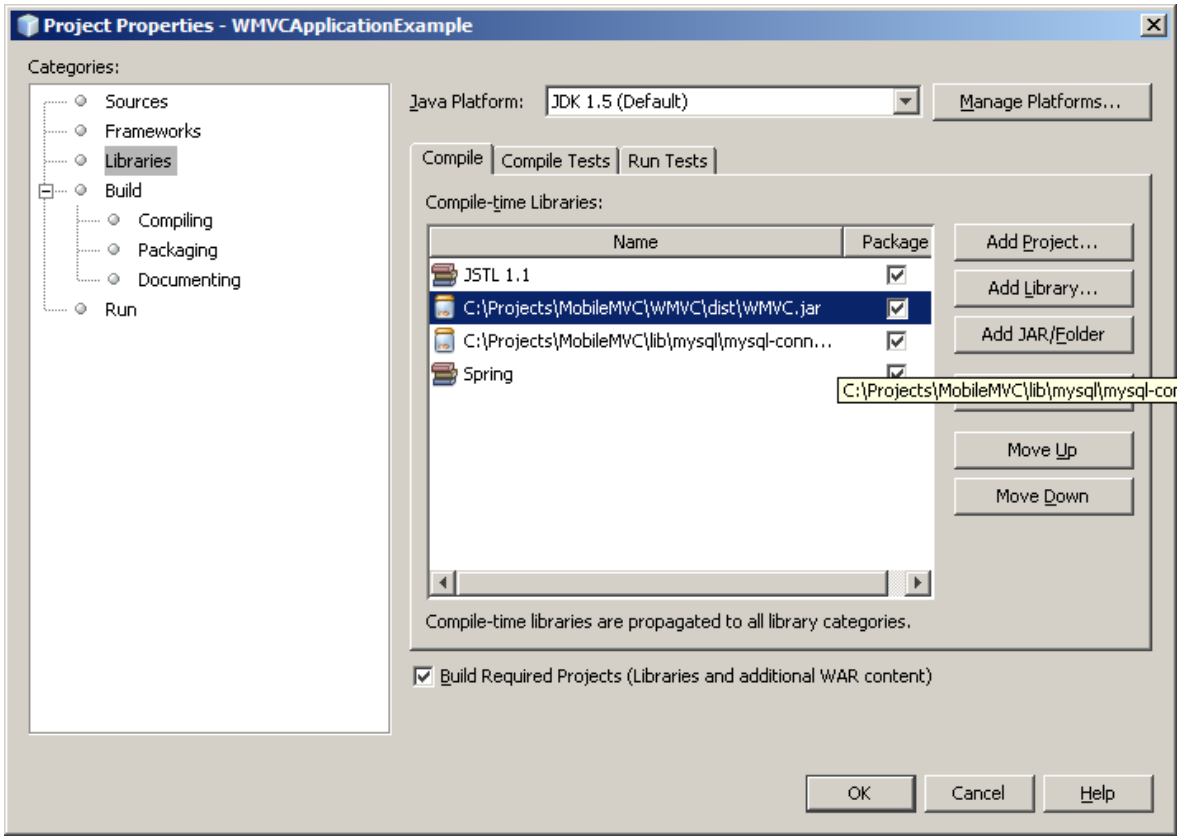


Obrázek 11 struktura sestaveného JAR souboru *wmvc.jar*

5.5 Použití knihovny WMVC

Výslednou knihovnu WMVC která vznikne sestavením projektu WMVC, používáme standardním způsobem jako Java knihovnu. Aby aplikace mohla vlastnosti WMVC využívat musí být třídy z knihovny WMVC dostupné na ***classpath*** webové aplikace nebo webového serveru ve kterém aplikace běží.

Výhodné je použít pokročilých IDE nástrojů pro vývoj, ty umožní pohodlné používání značek v JSP souborech, zobrazí k nim nápovědu, nabídnou dostupné atributy jednotlivých značek a podobně.



Obrázek 12 linkování knihovny WMVC do projektu v prostředí NetBeans

V samotném JSP souboru kde chceme značky WMVC používat je nutné nastavit správný typ obsahu MIME, který bude odeslán tenkému klientu. Klient zpracovává typ obsahu označený jako *application/wmvc-formdefinition*. Při jiném bude vyhlášena chyba a data nebudou zpracována.

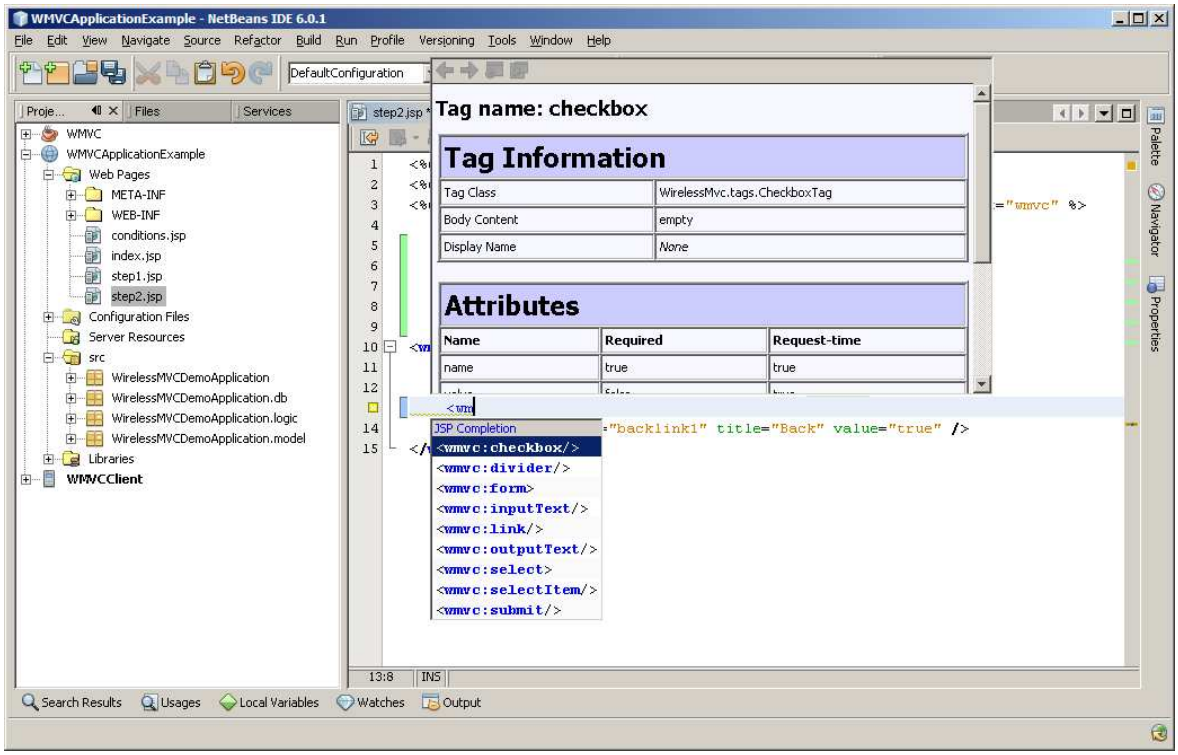
```
<%@page contentType="application/wmvc-formdefinition"%>
```

Dále musíme importovat Tag library které naše značky obsahuje. Pro to se používá identifikátor URI (viz 5.2.1 TLD – definice značek).

```
<%@taglib uri="http://edhouse.cz/j2ee/comps/wmvc/wmvctags"
          prefix="wmvc" %>
```

Poté již můžeme značky WMVC používat:

```
<%@page contentType="application/wmvc-formdefinition"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib uri="http://edhouse.cz/j2ee/comps/wmvc/wmvctags"
          prefix="wmvc" %>


<wmvc:form name="form">

  <wmvc:outputText name="outtext1" value="${message}" />

  <wmvc:submit name="backlink1" title="Back" value="true" />

</wmvc:form>
```



Obrázek 13 používání knihovny WMVC v prostředí NetBeans

6 IMPLEMENTACE TENKÉHO KLIENTA

6.1 Technologie

Jako technologie pro vývoj tenkého klienta byla zvolena Java Micro Edition (3.1 Java Micro Edition), kvůli její rozšířenosti a dostupnosti na prakticky všech mobilních telefonech. Možný kandidát byl i .NET Compact Framework, ten je ale k dispozici jen na omezeném množství modernějších zařízeních typu smartphone nebo PDA.

Komunikační protokol mezi serverem a klientem je ale navržen tak, aby nebyl svázán s platformou a aby byla možnost vytvořit klienty i pro jiné systémy, počítá se tedy, že pro určité aplikace bude vytvořen WirelessMVC klient i pro .NET Compact Framework.

6.2 Architektura mobilního klienta WirelessMVC

Viz UML class diagram: Příloha P II: Class diagram klienta WirelessMVC.

6.3 Implementace

V následujícím popisu budou popsány jednotlivé komponenty systému, pro pojmenování a strukturu viz UML class diagram v předchozí kapitole. Klient je zpracován jako Java projekt v prostředí NetBeans.

6.3.1 Balík WirelessMVCClient.

Obsahuje třídu ***WMVCClientMidlet***, což hlavní třída reprezentující *MIDlet* – samotnou Java aplikaci.

Při spuštění aplikace načte svoji konfiguraci z JAD souboru, součástí konfigurace je adresa na vzdálený interface poskytující pohledy MVC aplikace které máme zobrazit.

Aplikace poté uskuteční počáteční komunikaci se serverem, komunikace je zabalena ve třídě ***HttpConnector***, v balíku *connection*. Komunikace je realizována ve vedlejším vlákně aby nedošlo k blokování hlavního vlákna a tím uživatelského rozhraní aplikace.

Pokud je komunikace úspěšná, aplikace použije třídu ***WMVCFormBuilder*** z balíku *WirelessMVCApplication* pro sestavení objektové reprezentace formuláře. Na základě této struktury je poté v metodě *constructForm()* třídy *WMVCClientMidlet* sestaveno uživatelské

rozhraní aplikace pomocí prvků dostupných na platformě J2ME. Formulář je následně zobrazen uživateli.

Tabulka 3 prvky rozhraní J2ME použité pro implementaci WMVC značek

Prvek WMVC	**Prvek na platformě J2ME**
`inputText`	`javax.microedition.lcdui.TextField`
`checkBox`	`javax.microedition.lcdui.ChoiceGroup`
`outputText`	`javax.microedition.lcdui.StringItem`
`link`	`javax.microedition.lcdui.StringItem`, `javax.microedition.lcdui.Command`
`divider`	`javax.microedition.lcdui.StringItem`
`select`	`javax.microedition.lcdui.StringItem`, po otevření seznamu realizováno v novém formuláři pomocí `javax.microedition.lcdui.List`
`command`	`javax.microedition.lcdui.Command`

6.3.1.1 Implementace prvku select

Výběr z možností je implementován ve dvou úrovních, ve hlavním formuláři je zobrazen jako textová položka a po stisku akce OK je otevřen formulář nový, který zobrazí položky seznamu, po vybrání položky je textová položka v hlavním formuláři aktualizována a řízení je předáno zpět hlavnímu formuláři.

O zpracování akce pro otevření seznamu z hlavního formuláře se stará třída ***FormItemCommandListener***. Instance této třídy je nastavena jako výchozí pro zpracování akcí aktivních položek hlavního formuláře.

O obsluhu samotného formuláře zobrazujícího seznam položek a realizujícího výběr z možností se stará instance třídy ***ListCommandListener***.

6.3.1.2 Implementace prvku link

Za zpracování akce OK nad položkou *link* je zodpovědný ***FormItemCommandListener***, při zpracování akce *link* je iniciována nová komunikace a načten nový formulář definovaný adresou v hodnotě prvku *link*.

6.3.1.3 Implementace prvku submit

Při zpracování prvku *submit* dojde k odeslání aktuálního formuláře na server včetně dat vložených uživatelem. Pro zpracování uživatelem vložených dat a přenesení dat do modelu slouží metoda ***guiToModel()*** ve třídě ***WMVCClientMidlet***. Pro serializaci modelu a odeslání na server je použita metoda ***connectAndRetrieveData()*** třídy ***HttpConnector***.

6.3.2 Balík WirelessMVCApplication

Obsahuje třídy použité pro objektovou reprezentaci formuláře:

- ***WMVCForm***: celý formulář, obsahuje objekty *WMVCFormItem*
- ***WMVCFormItem***: jeden prvek formuláře.
- ***WMVCFormItemInnerItem***: zanořený prvek, například jedna položka v seznamu pro výběr.

Dále obsahuje již zmíněnou třídu *WMVCFormBuilder* sloužící pro deserializaci dat ze serveru a sestavení struktury formuláře. Její metoda *buildForm()* zpracuje serializovaná data a vrátí instanci třídy *WMVCForm*.

6.3.3 Balík WirelessMVCClient.helpers

Obsahuje tyto třídy:

- ***ErrorHelper***: slouží k zobrazování chybových stavů uživateli.
- ***UrlParamEncoder***: pomocník pro serializaci parametrů vkládaných do HTTP požadavků.
- ***StringHelper***: obsahuje pomocné funkce pro práci s řetězci.

6.3.4 Balík WirelessMVCClient.helpers.gui

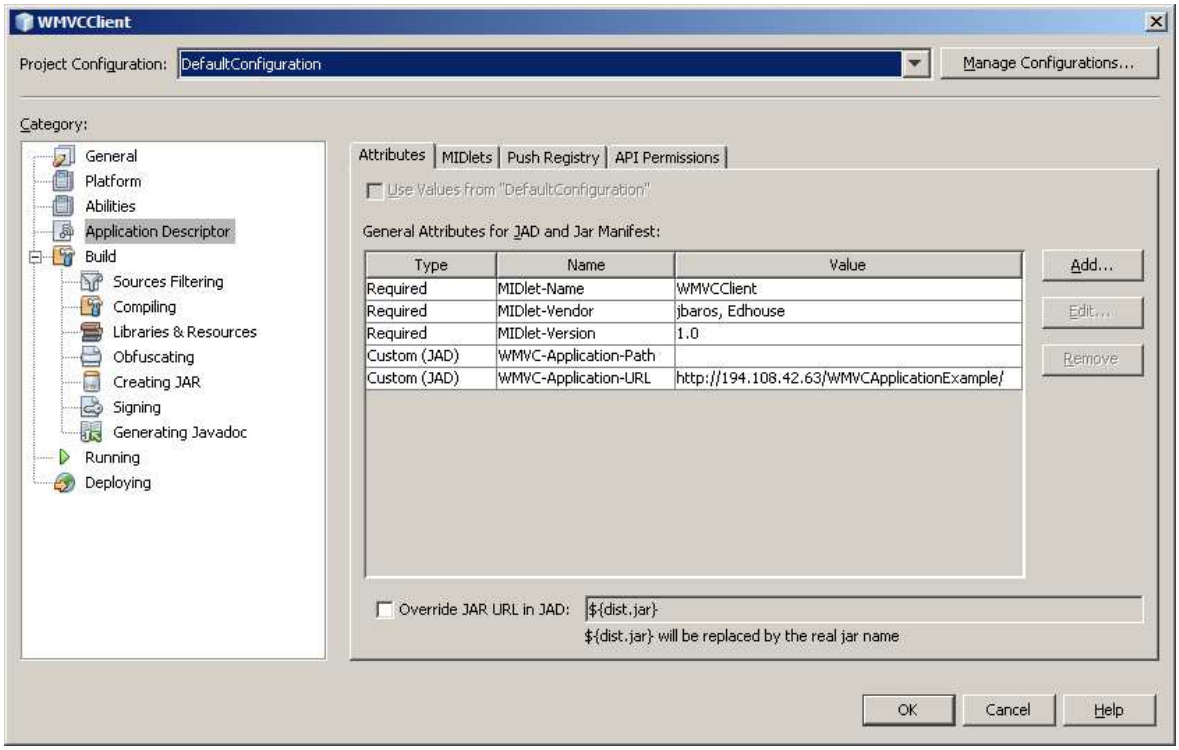
Obsahuje tyto třídy:

- ***OpenListCommand***: pomocná struktura zabalující prvek reprezentující seznam v hlavním formuláři a modelový objekt reprezentující výběr.
- ***LinkCommand***: pomocná struktura zabalující GUI prvek *link* a jeho hodnotu (adresa pro přechod)
- ***SubmitCommand***: pomocná struktura zabalující GUI prvek *submit* a příslušný modelový objekt.
- ***StringProvider***: centrální úložiště řetězců zobrazovaných uživateli, připravená podpora pro lokalizaci a internacionalizaci.

Všechny zdrojové texty jsou uloženy na přiloženém CD.

6.3.5 Konfigurace

Konfigurace klientské aplikaci se provádí v nastavení projektu prostředí NetBeans



Obrázek 14 konfigurace klientské aplikace

Tyto konfigurační parametry se při sestavení aplikace zapíšou do souborů *MANIFEST.MF* a *WMVCClient.jad* které se stanou součástí výsledné distribuce.

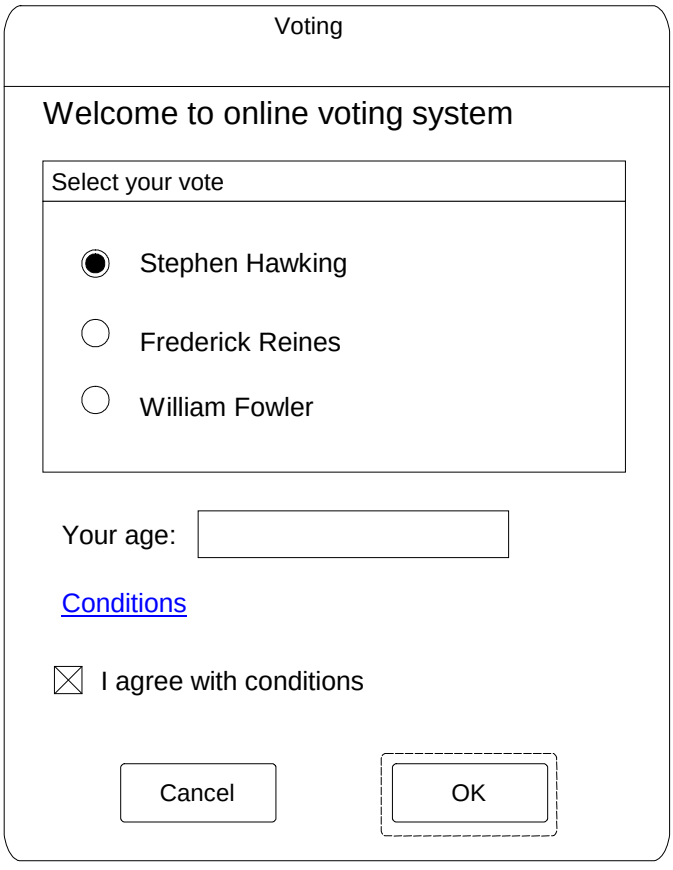
Parametr ***WMVC-Application-URL*** určuje adresu na kterou se má klient připojit pro získávání pohledů z MVC aplikace.

7 UKÁZKOVÁ APLIKACE

Jako ukázková aplikace byl zvolen jednoduchý systém pro online hlasování, aplikace má umožnit vybrání jedné možnosti ze seznamu, vyplnění věku a souhlas s podmínkami provozovatele. Na serveru se hlas uloží do databáze.

7.1 Uživatelské rozhraní aplikace

Na základě požadavků na aplikaci je zpracován následující symbolický návrh uživatelského rozhraní:



Obrázek 15 návrh uživatelského rozhraní ukázkové aplikace

Aplikace pro demonstraci má tedy obsahovat základní prvky potřebné pro sestavení interaktivní aplikace:

- Volba z možností

- Zadání textové hodnoty
- Ano/ne hodnota (checkbox)
- Odkaz na jinou obrazovku
- Příkazy (Ok, Cancel)

7.2 Implementace

Ukázková aplikace je zpracována jako standardní webová aplikace pro platformu Java a webový kontejner, využívající aplikačního frameworku Spring (2.5 Aplikační framework Spring), Spring Web MVC (2.5.2 Spring Web MVC) s rozšířením WirelessMVC.

Pro informace o struktuře Web aplikací pro platformu Java viz [8] Getting Started with Web Applications. Pro aplikace Frameworku Spring viz [5].

Deployment deskriptor (*WEB-INF/web.xml*)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
           http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
  </context-param>

  <servlet>
    <servlet-name>WMVC</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>WMVC</servlet-name>
    <url-pattern>*.w</url-pattern>
  </servlet-mapping>

  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
```

```
  <welcome-file-list>
    <welcome-file>
      index.jsp
    </welcome-file>
  </welcome-file-list>
</web-app>
```

V deployment deskriptoru definuji hlavní *ContextLoaderListener* který je nezbytný pro běh frameworku Spring a cestu ke konfiguraci aplikačního kontextu.

Dále definuji servlet WMVC který je nezbytný pro funkčnost Spring Web MVC včetně rozšíření *WirelessMVC*. Nastavení mapování (servlet-mapping) značí, že všechny HTTP požadavky s URL, které vyhovují masce „*.w“, budou směrovány na Spring MVC dispatcher. Tento dispatcher bude využívat výhradně JSP pohledy se značkami WMVC.

Konfigurace servletu WMVC je ve vyhrazeném XML souboru a má stejný formát jako definice aplikačního kontextu Springu:

WEB-INF/WMVC-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans …">

  <bean id="urlHandlerMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="urlMap">
      <map>
        <entry key="/step1.w" value="Step1Controller" />
        <entry key="/step2.w" value="Step2Controller" />
        <entry key="/conditions.w" value="ConditionsController" />
      </map>
    </property>
  </bean>

  <bean id="viewResolver"
class="org.springframework.web.servlet.view.UrlBasedViewResolver">
    <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView" />
    <property name="prefix" value="/" />
    <property name="suffix" value=".jsp" />
  </bean>

  <bean id="Step1Controller"
class="WirelessMVCDemoApplication.Step1Controller">
    <property name="voteBean" ref="voteBean" />
  </bean>

  <bean id="Step2Controller"
class="WirelessMVCDemoApplication.Step2Controller" />
  <bean id="ConditionsController"
class="WirelessMVCDemoApplication.ConditionsController" />

</beans>
```

(výpis byl zkrácen)

V konfiguraci aplikaci si tedy definuji tyto kontroléry:

- ***Step1Controller***: obsluhuje hlavní formulář aplikace hlasování, jeho pohled *step1.jsp* definuje formulář pro hlasování. Třída kontroléru obsahuje nastavení dat do modelu, validaci odeslaných dat a volání rutiny ukládání hlasu.
- ***Step2Controller***: obsluhuje informační obrazovku, která se zobrazí po odeslání *Step1*, zobrazuje informaci o úspěšném uložení hlasu, nebo chyby validace. Umožňuju návrat na *Step1*. Pohled je definován v *step2.jsp*.
- ***ConditionController***: formulář zobrazující statický text – podmínky pro hlasování. Umožňuje návrat na *Step1*. Pohled je definován v *conditions.jsp*.

index.jsp

```
<%@ taglib="" uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page="" session="false"%>
<c:redirect url="/step1.w"/>
```

Vstupní bod aplikace, obsahuje pouze přesměrování na první krok hlasování.

step1.jsp

```
<%@page contentType="application/wmvc-formdefinition"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://edhouse.cz/j2ee/comps/wmvc/wmvctags"
          prefix="wmvc" %>

<wmvc:form name="form1">

  <wmvc:outputText name="outText1" value="Hello, welcome to voting
                                          system" />
  <wmvc:outputText name="dateOuttext" value="Today is: ${date}" />

  <wmvc:outputText name="generalText" value="Please vote for
                                        best physicist" />

  <wmvc:select name="selectBox1" value="" title="I vote for">
    <c:forEach items="${items}" var="item">
      <wmvc:selectItem name="${item.name}"
           value="${item.value}" title="${item.title}" />
    </c:forEach>
  </wmvc:select>

  <wmvc:inputText name="userAge" value="" title="Your age:" />

  <wmvc:link name="link1" value="conditions.w" title="Conditions:" />
```

```
  <wmvc:checkbox name="checkBox1" value="false"
                 title="Yep, I agree with conditions"/>

  <wmvc:submit name="submit" value="true" title="Vote" />

</wmvc:form>
```

Hlavní formulář aplikace, obsahuje samotné hlasování. V příkladu je možné vidět použití JSTL a Expression Language.

Step1Controller.java

```
public class Step1Controller extends AbstractController {

     // injected by Spring
     private VoteBeanImpl voteBean;


     public Step1Controller() {
     }

     protected ModelAndView
        handleRequestInternal(HttpServletRequest request,
                              HttpServletResponse) throws Exception {

        ModelAndView modelAndView = null;

         // parameters that will come from client
         String userAge = request.getParameter("userAge");
         String physicist = request.getParameter("selectBox1");
         String agree = request.getParameter("checkBox1");
         String submit = request.getParameter("submit");


         // form was submitted
        if(submit != null && submit.equals("true")) {
             if(userAge != null && userAge.length() > 0
                 && physicist != null && physicist.length() > 0
                 && agree != null && agree.length() > 0) {

                 //check voter's age and agreewith condiotions
                 int age = age = new Integer(userAge);

                 boolean agreed = new Boolean(agree);
                 if(age < 10) {
                     modelAndView = new ModelAndView("step2");
                     modelAndView.addObject("message",
                            "Vote was rejected by
                             condition (age must be at least 10 years)");

                     return modelAndView;
                 }
                 else if(!agreed) {
                     modelAndView = new ModelAndView("step2");
                     modelAndView.addObject("message", "Vote was rejected
                                    by condition (you haven't
                                    agreed with conditions)");
                     return modelAndView;
                 }
```

```
                // ok, we've got correct vote, continue

                 // find right physicist
                 for(VoteItem voteItem:getVoteBean().getItems()) {
                     if(voteItem.getValue().equals(physicist)) {
                         physicist = voteItem.getTitle();
                         break;
                     }
                 }

                 // save vote to db
                 getVoteBean().saveVote(age, physicist);

                 // proceed to results and goodbye
                 modelAndView = new ModelAndView("step2");
                 modelAndView.addObject("message", "Thanks for
                                                    your vote");

                 modelAndView.addObject("data", "Your vote:"+physicist);

                 return modelAndView;
             }
             else {
                 modelAndView = new ModelAndView("step2");
                 modelAndView.addObject("message", "You havent filled all
                                                    required information
                                                    (vote, agreement and
                                                    your age)");
                return modelAndView;
             }
         }


         modelAndView = new ModelAndView("step1");
         modelAndView.addObject("items", getVoteBean().getItems());
         modelAndView.addObject("date", new Date().toLocaleString());

         return modelAndView;
     }
}
```

(výpis byl zkrácen)

step2.jsp

```
<%@page contentType="application/wmvc-formdefinition"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib uri="http://edhouse.cz/j2ee/comps/wmvc/wmvctags"
          prefix="wmvc" %>

<wmvc:form name="form2">
  <wmvc:outputText name="outtext1" value="${message}" />
  <wmvc:outputText name="outtext2" value="${data}" />
  <wmvc:submit name="backlink1" title="Back" value="true" />
</wmvc:form>
```

Určen pro zobrazení výsledku po odeslání formuláře *Step1*.

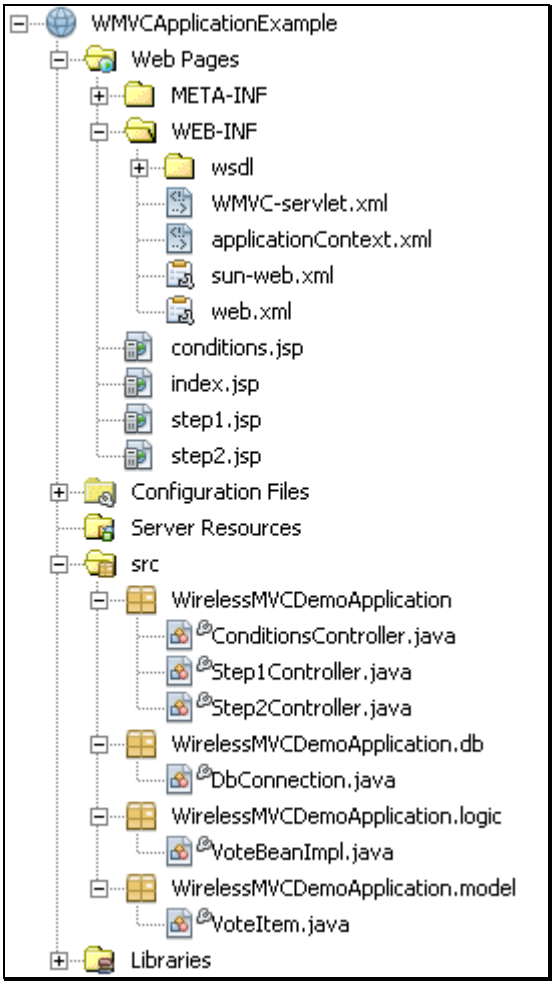
conditions.jsp

```
<%@page contentType="application/wmvc-formdefinition"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib uri="http://edhouse.cz/j2ee/comps/wmvc/wmvctags"
          prefix="wmvc" %>

<wmvc:form name="form1">
  <wmvc:outputText name="conditions"
                   value="if you want to participate..." />
  <wmvc:submit name="backLink" value="true" title="Back" />
</wmvc:form>
```

(výpis byl zkrácen)

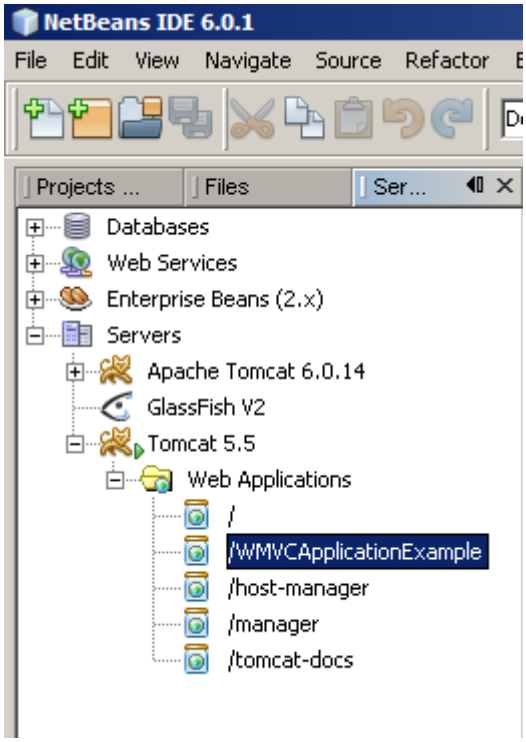
Na tento formulář se přechází pomocí prvku *link* z formuláře *Step1*.

Ukázková aplikace je zpracována v prostředí NetBeans IDE.



Obrázek 16 struktura projektu ukázkové aplikace *WMVCApplicationExample*

Sestavenou aplikaci (soubor WAR) musíme nasadit do webového kontejneru, kde bude spuštěna a dostupná pro klienta. Pro demonstraci můžeme aplikaci nasadit do webového kontejneru Apache Tomcat který je dodáván například s prostředím NetBeans.



Obrázek 17 nasazení ukázkové aplikace do kontejneru Tomcat

Funkčnost nasazené aplikace můžeme ověřit například otestováním adresy *http://localhost:8080/WMVCApplicationExample/step1.w* - měl by se objevit hlavní formulář aplikace v serializované podobě.

Tenkého klienta je nyní potřeba nastavit tak aby se připojoval na adresu naší nasazené aplikace. Adresu je možné upravit přímo v prostředí NetBeans (viz 6.3.5 Konfigurace), potom je nutné aplikaci znovu sestavit. Nebo úpravou souboru *WMVClient.jad* v sestavené distribuci klientské aplikace.

```
MIDlet-1: WMVCClientMidlet,,WirelessMVCClient.WMVCClientMidlet
MIDlet-Jar-Size: 22468
MIDlet-Jar-URL: WMVCClient.jar
MIDlet-Name: WMVCClient
MIDlet-Vendor: jbaros, Edhouse
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
WMVC-Application-Path:
WMVC-Application-URL: http://194.108.42.63/WMVCApplicationExample/
```

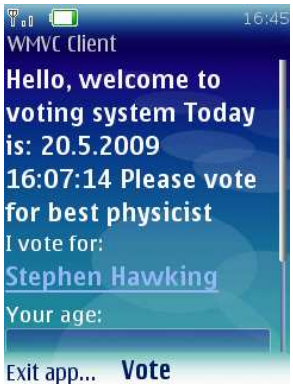
7.3 Instalace klientské aplikace

Klientskou aplikaci je nyní nutné nainstalovat do mobilního telefonu. Typický scénář je že instalační soubory ***WMVCClient.jad*** a ***WMVCClient.jar*** umístíme na adresu dostupnou přes internet a v mobilním telefonu instalujeme aplikaci z této adresy, odkaz je možné rozeslat například pomocí SMS. Na mobilním telefonu musí být pochopitelně funkční datové přenosy a dostupný internet.

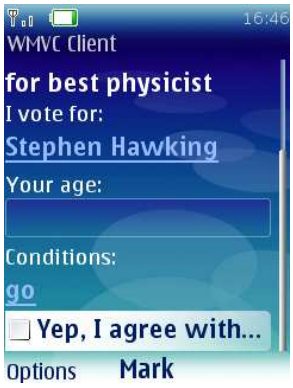
Obrázek 18 instalace klientské aplikace na mobilním telefonu

7.4 Spuštění aplikace v tenkém klientu (mobilním telefonu)

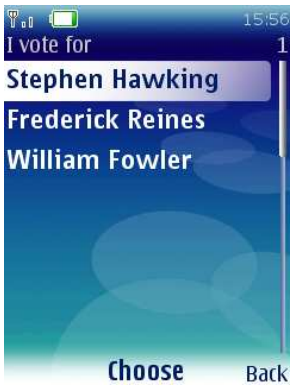
Po instalaci je aplikace dostupná v nabídce nainstalovaných aplikací pod názvem ***WMVCClient***. Při spuštění se aplikace automaticky připojí k serveru a zobrazí úvodní formulář.



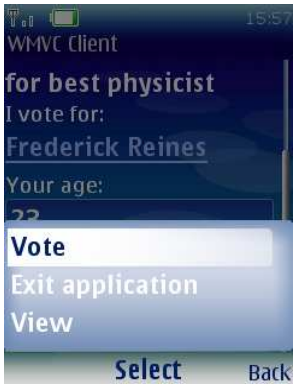
Obrázek 19 ukázková aplikace na mobilním telefonu



Obrázek 20 ukázková aplikace na mobilním telefonu



Obrázek 21 výběr ze seznamu



Obrázek 22 odeslání hlasu

Obrázek 23 uložení hlasu

ZÁVĚR

Analýzou požadavků a dostupných prostředků byly vybrány vhodné přístupy a technologie pro implementaci Model-View-Controller frameworku pro mobilní klient-server aplikace. Podařilo se navrhnout a ověřit architekturu funkčního systému se všemi požadovanými vlastnostmi. Výsledkem je kromě návrhu architektury a technologií i referenční implementace MVC frameworku WirelessMVC, realizovaná jako rozšíření frameworku Spring MVC pro platformu Java. Součástí je implementace univerzální klientské aplikace, která umožní používat aplikace vytvořené pomocí WirelessMVC na drtivé většině dnešních mobilních telefonů a dalších malých zařízeních. Vytvořena je i ukázková aplikace pro online hlasování demonstrující možnosti, funkčnost systému a také doporučený způsob použití vytvořené knihovny.

S dnešním rozvojem datových služeb se otevírají nové možnosti pro systémy tohoto typu, od aplikací na online hlasování, až po například vysoce zabezpečené mikroplatební systémy. Možné využití vidím například i v podnikové sféře, na řízení a sběr dat od pracovníků v terénu, pokud například není možné plošně použít sofistikovanější zařízení typu PDA.

Návrhy na možná vylepšení

Bezpečnost:

- Systém umožňuje snadnou změnu komunikačního protokolu například na šifrovaný HTTPS.

Optimalizace datového toku a výkonu:

- Cacheování dat v klientu, označování formulářů verzemi a stahování jen nových, či aktualizovaných formulářů a dat.

Rozšíření možností uživatelského rozhraní:

- Přidání nových prvků uživatelského rozhraní, například pro zadávání nebo editaci delších textů, zadávání data a času a další.
- Validace dat v klientu, například omezit určité vstupní pole jen na čísla nebo řetězec s určitým formátem.
- Implementace klientů pro další platformy, například .NET Compact Framework.

CONCLUSION

Reasonable technologies and appropriate approaches were chosen for implementation of the Model-View-Controller framework for mobile client-server applications. Architecture of the system was successfully proposed and proven by reference implementation with all required features. Besides the system architecture proposal, this functional reference implementation, called WirelessMVC is a result of this work. Implementation is done as an extension to Spring MVC framework for the Java platform. Implementation of thin client for mobile devices is part of this work, this client allows use of applications developed with WirelessMVC on most of cell phones. Attached is an example application for demonstration of using WirelessMVC library, its features and abilities. With today's growth of mobile-data services new abilities for systems like this are opened, from simple applications like online voting to complex and secured micro-payment systems.

SEZNAM POUŽITÉ LITERATURY

[1] SETH, Ladd, et al. Expert Spring MVC and Web Flow. [s.l.] : [s.n.], c2006. 423 s. ISBN 978-1-59059-584-8.

[2] JONATHAN, Knudsen. Wireless Java Developing with J2ME, Second Edition. [s.l.] : Apress,c2003. 384 s., 384. ISBN 1590590775.

[3] ROD, Johnson, et al. Professional Java Development with the Spring Framework. [s.l.] : [s.n.], c2005. 672 s. ISBN 0764574833.

[4] GRAFF, Mark G., VAN WYK, Kenneth R. Secure Coding: Principles & Practices . [s.l.] : O'Reilly, c2003. 224 s. ISBN 0-596-00242-4.

[5] JOHNSON, Rod, et al. The Spring Framework : Reference Documentation [online]. 2007. 2007 [cit. 2009-02-03]. Dostupný z WWW:[http://static.springframework.org/spring/docs/2.5.x/reference/index.html].

[6] Spring Framework API 2.5 [online]. 2002-2008 [cit. 2009-02-03]. Dostupný z WWW: [http://static.springframework.org/spring/docs/2.5.x/api/index.html].

[7] BAILLIEZ, Stephane, et al. Apache Ant User Manual [online]. Verze 1.7.1. [2005] [cit. 2009-03-02]. Dostupný z WWW: <http://ant.apache.org/manual/index.html>.

[8] ARMSTRONG, Eric, et al. The J2EE 1.4 Tutorial [online]. Update 7. c2005 [cit. 2009-02-03]. Dostupný z WWW: <http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html>.

[9] Wikipedia, the free encyclopedia [online]. 2003-2009 [cit. 2009-03-02]. Dostupný z WWW: <http://en.wikipedia.org/wiki/>.

[10] LISKOV, Barbara. Design Patterns. [s.l.] : [s.n.], 2005. Dostupný z WWW: <http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-170Fall-2005/96A8B7AF-3C20-4D92-8F07-EFA537FD82C2/0/lec18.pdf>. 15 Design Patterns, s. 1-16.

[11] MSDN Library - patterns and practices : Model-View-Controller [online]. Verze 1.0.1. c2009 [cit. 2009-02-03]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms978748.aspx>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

.NET CF	.NET Compact Framework
AOP	Aspect Oriented Programming
API	Application Program Interface
CDC	Connected Device Configuration
CGI	Common Gateway Interface
CLDC	Connected Limited Device Configuration
DAO	Data Access Object
DNS	Domain Name System
EJB	Enterprise Java Beans
GPRS	General packet radio service
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IOC/DI	Inversion Of Control / Dependency Injection
J2EE	Java Enterprise Edition
J2ME	Java Micro Edition
JAD	Java Application Descriptor
JAR	Java Archive
Java EE	Java Enterprise Edition
Java ME	Java Micro Edition
JDBC	Java Database Connectivity
JIT	Just-In-Time
JSP	Java Server Pages

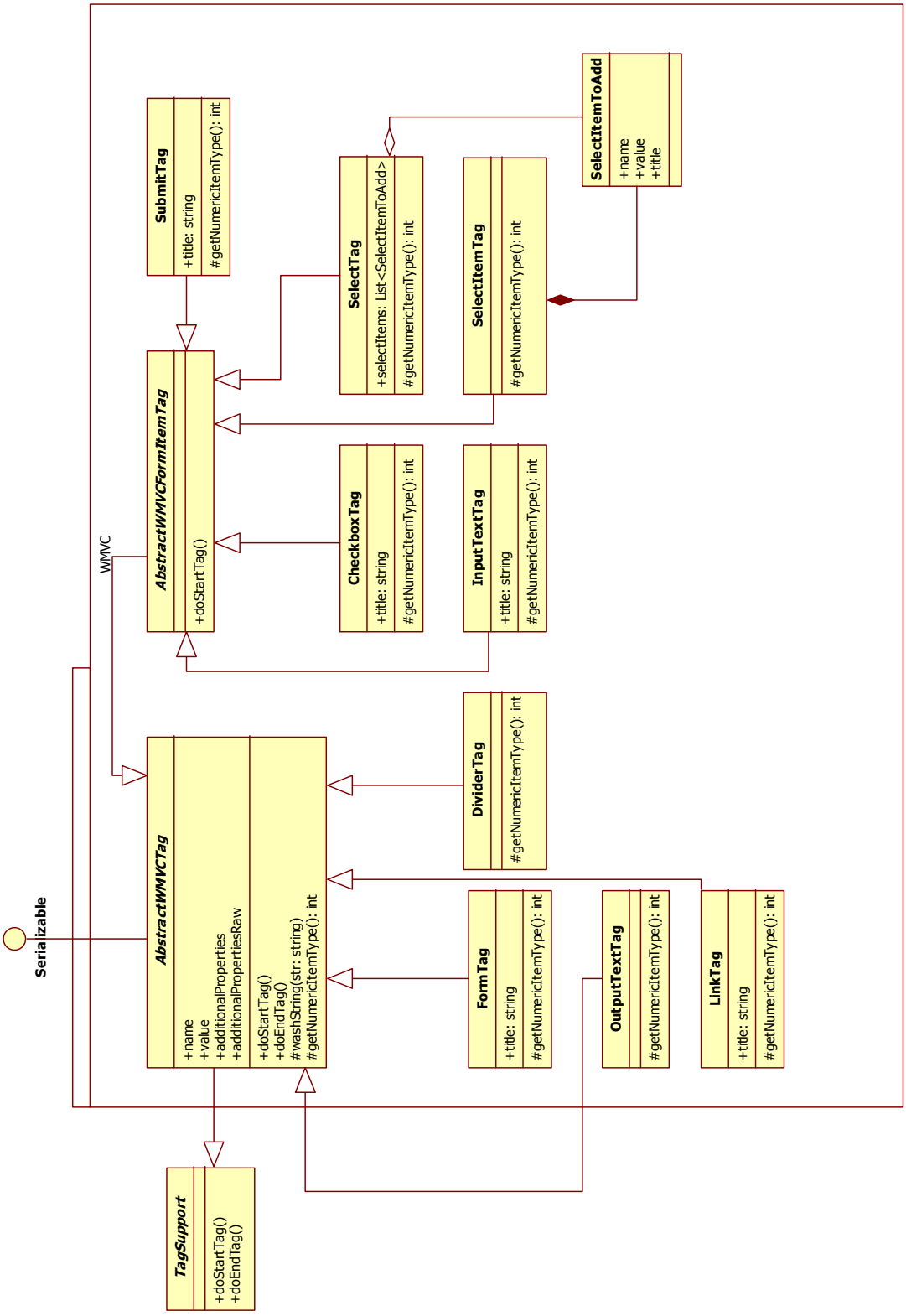
JSR	Java Specification Request
JSTL	Java Standard Tag Library
JVM	Java Virtual Machine
MVC	Model View Controller
ORM	Object Relations Mapping
PC	Personal Computer
PDA	Portable Digital Assistant
POJO	Plain Old Java Object
SMS	Short Message System
TCP/IP	Transmission Control Protocol/Internet Protocol
TLD	Tag Library Descriptor
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAP	Wireless Application Protocol
WAR	Web Application Archive
WML	Wireless Markup Language
WWW	World Wide Web
XML	Extensible Markup Language

SEZNAM OBRÁZKŮ

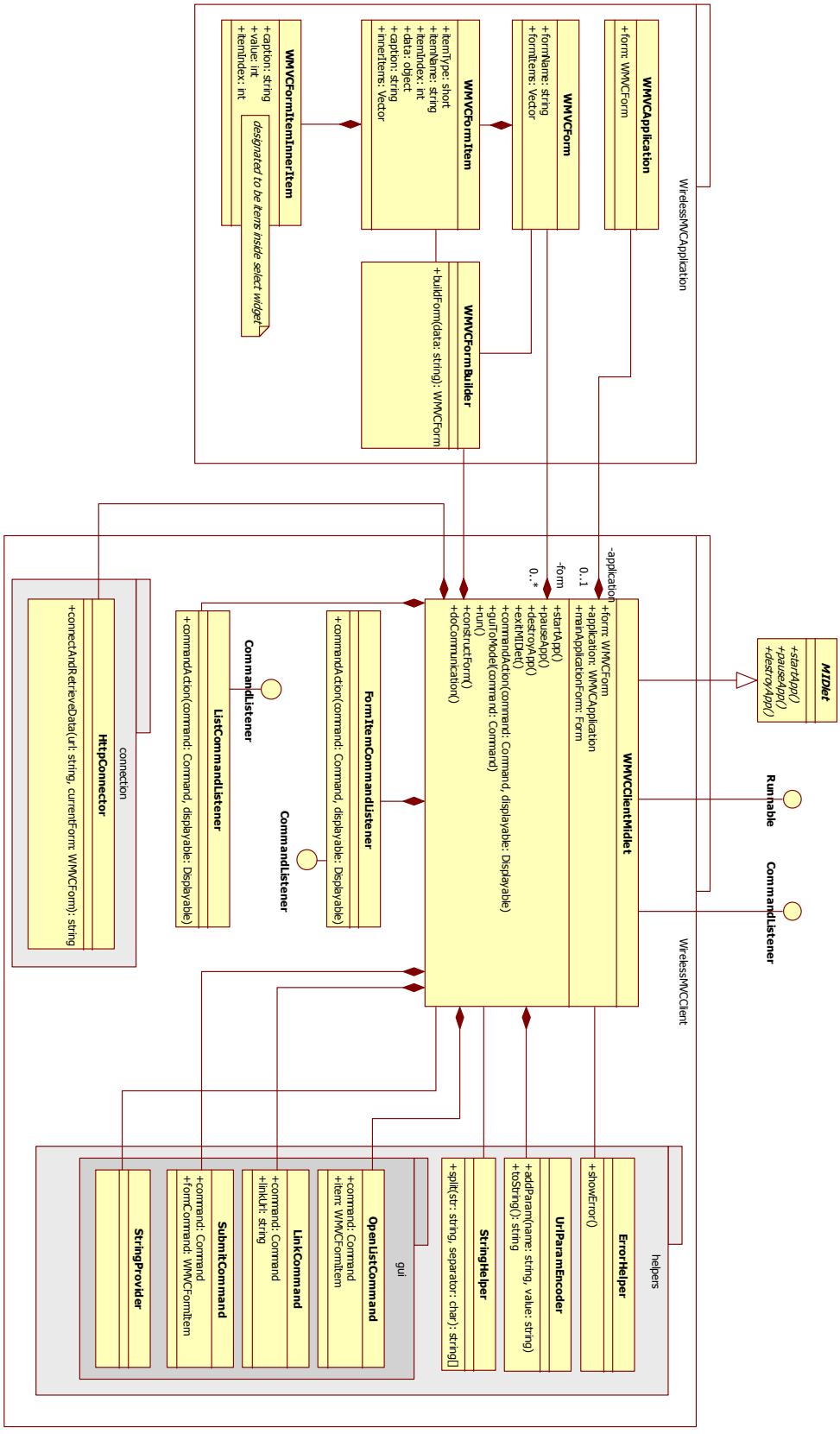
SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CLASS DIAGRAM IMPLEMENTACE KNIHOVNY WIRELESSMVC



PŘÍLOHA P II: CLASS DIAGRAM KLIENTA WIRELESSMVC



PŘÍLOHA P III: SEZNAM ZNAČEK A JEJICH IMPLEMENTACE

Název elementu	Typ dat	Implementující třída	Předek implementační třídy	poznámka
`form`	`-`	`WirelessMvc.tags.FormTag`	`WirelessMvc.tags.AbstractWMVCTag`	Může obsahovat všechny následující
`inputText`	`String`	`WirelessMvc.tags.InputTextTag`	`WirelessMvc.tags.AbstractWMVCFormItemTag`	
`outputText`	`String`	`WirelessMvc.tags.OutputTextTag`	`WirelessMvc.tags.AbstractWMVCFormItemTag`	
`checkbox`	`Boolean`	`WirelessMvc.tags.CheckboxTag`	`WirelessMvc.tags.AbstractWMVCFormItemTag`	
`link`	`String`	`WirelessMvc.tags.LinkTag`	`WirelessMvc.tags.AbstractWMVCTag`	
`submit`	`String`	`WirelessMvc.tags.SubmitTag`	`WirelessMvc.tags.AbstractWMVCFormItemTag`	
`divider`	`-`	`WirelessMvc.tags.DividerTag`	`WirelessMvc.tags.AbstractWMVCTag`	
`select`	`Vector`	`WirelessMvc.tags.SelectTag`	`WirelessMvc.tags.AbstractWMVCFormItemTag`	Obsahuje objekty `selectItem`
`selectItem`	`Integer`	`WirelessMvc.tags.SelectItemTag`	`WirelessMvc.tags.AbstractWMVCFormItemTag`	