TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

Webové služby v mobilních zařízeních: získávání dat a spouštění výpočtů

Web services in mobile devices: collecting data and triggering calculations

Diplomová práce

Autor: Bc. Petr Žáčík

Vedoucí práce: Ing. Roman Špánek

Konzultant: Ing. Roman Špánek

V Liberci dne 10.05.2008

ORIGINÁLNÍ ZADÁNÍ PRÁCE

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že souhlasím s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum:

Podpis:

Poděkování

Chtěl bych tímto poděkovat všem, kteří mi pomohli k vypracování tohoto textu, zejména však vedoucímu práce Ing. Romanu Špánkovi.

Abstrakt

V rámci diplomové práce je řešena problematika webových služeb a jejich využití pro spouštění vzdálených výpočtů a získávání dat ze vzdálených zdrojů. Jako cílová zařízení, realizující klientskou část, byly zvoleny mobilní telefony. Práce obsahuje i pilotní implementaci realizující aplikaci pro získávání dat o úvěrech, která byla oceněna třetím místem v soutěži MPA pořádané předními bankovními a telekomunikačními firmami v České republice.

Abstract

This Master's degree project is focused on the web services while triggering remote calculations and collecting data from the remote data sources. The client side is handled by the mobile device. The final part includes a pilot implementation of the application for collecting data about loans. This application was placed on the third position in MPA competition supported by leading bank and telecommunication companies in Czech republic.

Klíčová slova

programovací jazyk Java, aplikace typu klient-server, webové služby, J2ME, J2EE, půjčka

Keywords

Java programming language, client-server applications, web services, J2ME, J2EE, loan

OBSAH

Úvod

Počítačová síť typu WAN s názvem internet se dnes již stala běžnou součástí každé firmy a byla již díky příznivějším podmínkám zavedena i v domácnostech. Díky internetu vzniklo v tomto virtuálním světě mnoho nových služeb, které zrychlují a zlevňují stávající postupy. Všechny tyto poskytované služby koexistují na společné síti a jsou dostupné svým klientům.

Jednou z neméně důležitých služeb je služba webová. Tato technologie umožňuje komunikaci mezi různými aplikacemi provozovanými na různých platformách díky otevřenému a jednoduchému komunikačnímu protokolu. Webové služby totiž komunikují pomocí protokolu HTTP formou strukturovaných XML dokumentů. Webové služby tedy ve své podstatě dávají možnost realizovat celou logiku aplikace na serverové straně a pomocí jednoduchého klienta pouze volat vzdálené aplikace a přijímat výsledek s minimálními požadavky na hardware a software klienta.

Proč ale přesouvat výpočty na stranu serveru? Například z nutnosti šetřit objemy přenesených dat. Pokud klient potřebuje k vyhodnocení operace sadu dat, která jsou vystavena na serveru, musejí se tato data mezi klientem a serverem přesouvat, a jelikož je XML známé pro svůj nepříliš výhodný poměr dat a velikosti dokumentu, může být tento způsob časově, či finančně nákladný. A nejen v tomto případě je výhodnější provádět operaci přímo na straně serveru a přenášet pouze zadání a výsledek. Výpočty se přesouvají i tam, kde sám klient nemá dostatečný výpočetní výkon nebo prostředky nutné k vyřešení operace.

V dnešní době miniaturizace a integrace lze k mnoha informacím přistupovat i pomocí mobilního telefonu. Sám telefon však vždy nemusí mít dostatečný výkon k jejich zpracování. V těchto případech se uplatňuje webový prohlížeč, který pracuje na podobném principu jako webové služby. Problémem se zde stává míra interakce, kterou prohlížeč zejména u mobilních telefonů umožňuje. Pokud již mobilní zařízení podporuje provádění javascriptů nebo zobrazení flashové animace, je dobré

si uvědomit, kolik přenesených dat tento přístup obnáší nehledě na velikosti displeje zařízení, který nemusí formuláře nebo jejich rozložení zobrazovat tak, jak bylo zamýšleno. V tuto chvíli nastupují webové služby. Uživatel si jednorázově do mobilního zařízení nainstaluje klientský program, který zajistí komunikaci a co nejpříjemnější rozhraní a zobrazení výsledků.

Součástí této práce je návrh a realizace webové služby a klienta určeného pro mobilní zařízení. Realizace je v programovacím jazyce Java s využitím J2EE. Java byla zvolena pro svojí multiplatformnost, která je v souladu s multiplatformností webových služeb nebo síťových protokolů obecně. Tedy kód napsaný pro server i klienta je možné vystavit nezávisle na platformě po naplnění základních požadavků provozu.

Cílem praktické části projektu je nabídnout uživatelům možnost, jak prostřednictvím mobilního telefonu získat nabídku úvěrů současného trhu a také poradit jakou z nabízených možností zvolit. Výhody mobilního zařízení pak uživatelé pocítí v situacích, kdy budou stát před rozhodnutím, zda si výrobek koupit na leasing, nebo zda využít některého z nabízených úvěrů. Rozhodnout se tedy bude moci přímo na místě. Tento návrh se zúčastnil soutěže pořádané Mobile Payment Association pod Research and Development centrem v Praze v kategorii nový produkt na téma „Využití mobilních zařízení ve finančních službách“, kde získal třetí místo. Soutěž byla pořádána za podpory předních mobilních operátorů a nejvýznamnějších bankovních ústavů v České republice.

1 TEORETICKO – METODOLOGICKÁ ČÁST

Tato část se zabývá shrnutím teoretických poznatků v oblasti webových služeb, Javy, půjček a technologií, které pomohou k sestavení metodologie pro tvorbu webové služby a klienta.

1.1 Java

Java je objektově orientovaný programovací jazyk (OOP) vyvinutý firmou Sun Microsystems Inc. Z angličtiny se slovo Java překládá jako káva nebo šálek kávy. Šálek kávy má Java také ve svém logu. Heslo Javy zní: „Write once, run anywhere“ V překladu to znamená: „Napsat jednou, spustit kdekoliv“ [1].

Java je jazyk jednoduchý, objektově orientovaný, síťově důmyslný, robustní, bezpečný, nezávislý na architektuře, přenositelný, interpretovaný, vysoce výkonný, vícevláknový a dynamický [2].

1.2 Diagram tříd

Diagram tříd (class diagram) je nástroj, který umožňuje zobrazení statické struktury systému prostřednictvím tříd a vztahů mezi nimi [3].
Diagram tříd bude využit při návrhu modelu webových služeb v Javě.

1.3 J2EE (Java 2 Enterprise Edition)

Jde o standard, který se snaží zjednodušovat širokou škálu požadavků na soudobou aplikaci. J2EE platforma je založená na vícevrstvém distribuovaném aplikačním modelu, díky kterému lze jednotlivé části aplikace provozovat na různých typech zařízeních. J2EE architektura definuje následující vrstvy [4].

- Vrstva klienta
- Střední vrstva

- Vrstva EIS (Enterprise Information System)

Platforma J2EE má ústřední úlohu při tvorbě projektu, kde bude uplatněna na úrovni střední a EIS vrstvy.

Vrstva klienta

Podporuje vývoj klientských aplikací. Do této vrstvy patří vývoj klientských aplikací pro různá zařízení. Tato vrstva přímo komunikuje se střední vrstvou poskytující klientům služby. Pro tvorbu klienta bude využita technologie J2ME.

Střední vrstva

Podporuje vývoj klientských služeb pomocí Web kontejnerů a business logiku prostřednictvím EJB (Enterprise Java Bean) [5].

Enterprise Java Bean je základní komponenta J2EE na straně serveru, která se podle typů dělí na:

- Session Bean

  Vlastní rozhraní *Local* a *Remote* pomocí kterých se přistupuje k metodám.

  - Stavová

    Používá se tam, kde je třeba si mezi jednotlivými voláními metod pamatovat vnitřní stav Bean. Například v internetovém obchodě to může být nákupní košík, který si pro konkrétní sezení pamatuje svůj obsah.
  - Bezstavová

    Tato Bean si nepamatuje vnitřní stav, chová se tedy pouze jako rozhraní.
- Řízená zprávami (MDB)

  Využívá Java Messaging Service, která nemá rozhraní ke komunikaci. Ovládá se zasláním zpráv. Používá se v situacích, kdy klient nečeká na odpověď.
- Entity Java Bean

  Třída představující databázovou tabulku.

Vrstva EIS

Vrstva EIS – Enterprise Information System zajišťuje persistenci a obsluhu databáze. Je prostředníkem mezi aplikací a databází. Její součástí je JPA a samostatný nástroj Hibernate.

JPA (Java Persistence API) se používá k mapování objektu a relací databáze. To lze provést použitím anotací ve zdrojovém kódu nebo externím mapovacím XML dokumentem. Objekt mapovaný na databázi se nazývá Entity Bean. Zápis anotace v programu začíná znakem „@“. Pokud například chceme říci, že databázová tabulka „bank“ bude mapována na třídu *BankEntity*, zapíše se zdrojový kód následujícím způsobem.

Výpis 1.1 – Použití anotací JPA

```
@Entity
@Table(name = "bank")
public class BankEntity implements Serializable {
  //...
}
```

Podobným způsobem se mapují i jednotlivé atributy objektu, aby odpovídaly atributům tabulky.

Mapování se provádí na Javové objekty tzv. POJO (Plain Old Java Object). Tato třída obsahuje pouze atributy a jejich getter a setter. Každou třídu si lze představit jako tabulku v databázi, takže komunikace mezi databází a programem pak funguje na principu předávání referencí na objekty.

1.3.1 JBoss aplikační server

Jedná se o nástroj, který spouští a poskytuje komponenty založené na J2EE jako jsou servlety, Java Server Pages (JSP) nebo Enterprise Java Beans (EJB). Slouží tedy

obecně jako kontejner pro komponenty střední vrstvy. Webové rozhraní spuštěného a správně nastaveného lokálního serveru je viditelné na adrese *http://127.0.0.1:8080*.

1.3.2 Hibernate

Hibernate je objektově relační mapovací nástroj (ORM) pro Javu, který umožňuje persistenci dat bez nutnosti znát způsob, jakým se s nimi fyzicky manipuluje. Další výhodou je, že se datový zdroj nastavuje pomocí XML souboru, takže je jej možné měnit bez zásahu do kódu. Lze tak jednoduše přecházet mezi databázemi a dokonce i typy datových zdrojů, například mezi MySQL, Oracle, MSSQL, MS Access, XML a dokonce i CSV (Comma-Separated Values), což je soubor s prostým textem, kde jsou jednotlivé položky odděleny čárkou.

HQL (Hibernate Query Language)

Dotazování na data se provádí pomocí HQL, ale lze použít i nativní dotazovací jazyk. V následujícím výpisu je uveden příklad vyhledávacího dotazu jazyka HQL.

Výpis 1.2 – Dotaz HQL

```
"from User u where u.age >= :minAge"
```

Objektový přístup je viditelný. Dotaz vrátí kolekci podle předpisu třídy *User*, kde věk uživatele je menší rovno parametru *minAge*, který je do dotazu automaticky importovaný z lokálních definic programu.

1.4 J2ME (Java 2 Micro Edition)

J2ME [6] nabízí robustní a flexibilní prostředí pro vývoj aplikací pro mobilní zařízení. Pro vývoj a emulaci je nutné stáhnout ze stránek společnosti Sun nástroj Wireless Toolkit. Obsahuje mimo jiné i balíky potřebné pro vývoj klienta v Javě. Pro správu projektů je zde program *ktoolbar* a velmi důležitý nástroj *wscompile*,

který bude použit ke generování stubu klienta mobilního zařízení z existujícího WSDL.

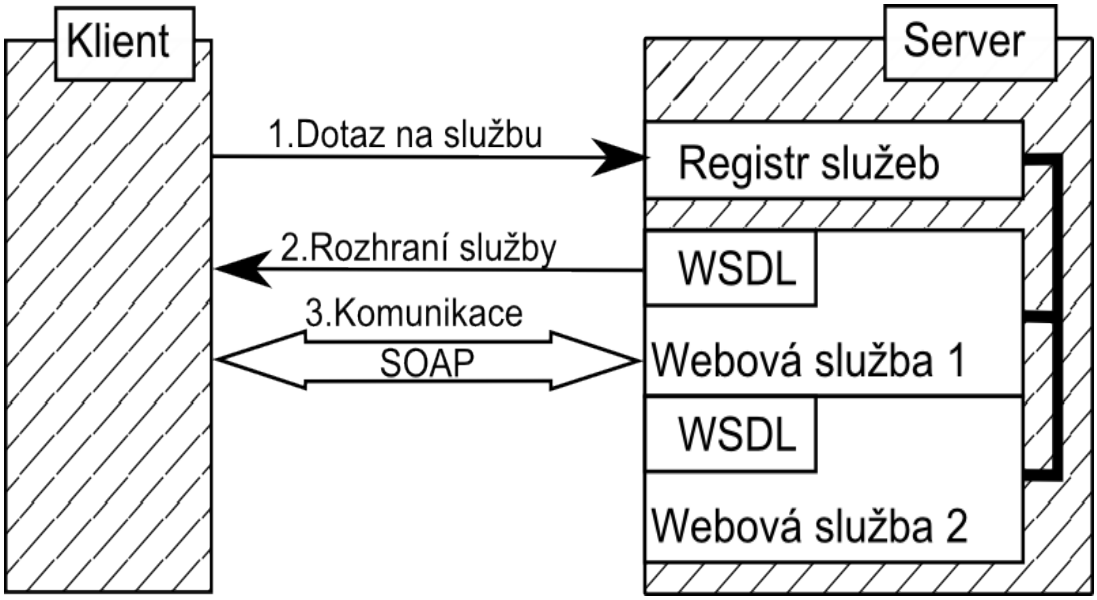
1.5 Webová služba

Webová služba [7] je program na straně serveru s architekturou klient-server, jehož hlavní prioritou je umožnit součinnost aplikací v síti. Jde o technologii pro vzdálené volání metod v distribuovaných systémech jako jsou RPC, CORBA nebo Java RMI. Pomocí této technologie je možné propojit aplikace typu B2B (Business to Business), které jsou rozděleny lokalitami nebo platformami. Ke komunikaci je využito otevřených standardů HTTP, XML a SOAP.

Původní idea vychází ze standardu vzdáleného volání procedur RPC (Remote Procedure Call). Z hlediska vzrůstající složitosti předávaných parametrů způsobených převážně vlivem rozvoje objektově orientovaného přístupu se ztěžoval vývoj agentů pro vysílání a příjímání zpráv, zejména pak na různorodých platformách. To vyvolalo potřebu obecnějšího řešení. Pro zajištění platformní a jazykové nezávislosti se začal jako datový nosič využívat XML dokument – XML-RPC. Ani tento formát však dlouho neobstál. Byl nahrazen protokolem SOAP, který se snaží pokračovat tam, kde XML-RPC skončil implementací uživatelsky definovatelných datových typů. Současná verze SOAP je 1.2.

Komunikace probíhá nejčastěji na síťovém portu 80 (protokolem HTTP) předáváním strukturovaných XML dokumentů. Toto dává protokolu SOAP možnost průchodu skrze firewally, které filtrují komunikaci a ostatní protokoly jako RPC z bezpečnostních důvodů běžně blokují. Možnosti konkrétní služby jsou popsány definičním jazykem webových služeb WSDL (Web Service Definition Language). Tento XML dokument musí být klientům znám před zahájením komunikace. Vystavená služba může být pomocí UDDI (Universal Description, Discovery, and Integration) zařazena do veřejné databáze pro vyhledávání služeb, nebo pomocí

jazyka WSIL (Web Service Inspection Language) zařazena do lokálního seznamu serveru, kde se soubor s dostupnými službami *inspection.wsil* umisťuje do kořenového adresáře serveru. Možný způsob zahájení komunikace je znázorněn na obrázku 1.1.



Obr. 1.1 – Zahájení komunikace s webovou službou

1.6 Problematika výhodnosti úvěru

Ukazatel výhodnosti poskytovaného úvěru se nazývá „roční procentní sazba nákladů“ ve zkratce RPSN. Udává procentní poměr z poskytnuté půjčky, který musí spotřebitel přeplatit za období jednoho roku a to včetně poplatků za zavedení, správu, čerpání atp. Od úrokové míry se liší závislostí na čase. V České republice dle zákona č. 321/2001 Sb. mají poskytovatelé spotřebitelských úvěru povinnost uvádět u svých produktů RPSN. Pro výpočet RPSN se uvádí následující vzorec, který ukazuje ekvivalenci splátek s poplatky na straně jedné s půjčkami na straně druhé.

$$\sum_{K=1}^{K=m} \frac{A_K}{(1+i)^{t_K}} = \sum_{K'=1}^{K'=m'} \frac{A'_{K'}}{(1+i)^{t_{K'}}}$$

Vzorec 1.1 – Výpočet RPSN

K - číslo splátky nebo poplatku

K' – číslo půjčky

A_K - částka konkrétní splátky

$A'_{K'}$ - částka konkrétní půjčky

m - číslo poslední splátky nebo poplatku

m' - číslo poslední půjčky

t_K - interval vyjádřený v rocích a zlomcích roku mezi první až m-tou splátkou

t_K - interval vyjádřený v rocích a zlomcích roku mezi první až m-tou půjčkou nebo poplatkem

i – sazba v procentech

2 ANALYTICKÁ ČÁST

Moderní doba s sebou přináší celou řadu nových možností pro řadové občany, jak si koupit téměř libovolné produkty bez nutnosti mít požadovanou hotovost. Na druhou stranu tento trend s sebou přinesl také řadu problémů. Jedním z nich je vzrůstající zadluženost obyvatelstva, která je způsobena hlavně snadnou dostupností půjček, které mají ovšem mnohdy zavádějící podmínky. Poskytovatelé úvěrů sice RPSN uvádějí, nebývají k němu však mnohdy připočítány poplatky.

Uživatel bude svůj mobilní telefon používat jako rozhraní k webové službě. Službě nahlásí částku, případně podmínky, za kterých bude schopen splácet. Data o dostupných produktech budou v této implementaci zadávána ručně. Je však technicky možné při budoucím rozšiřování funkčnosti aplikace data získávat automaticky pomocí crawlování internetu nebo parsování webových stránek. Následně se vyhodnotí nabídky a provede přepočet na skutečnou cenu půjčky, tedy kolik klient zaplatí celkem za vypůjčenou částku (připočte RPSN a poplatky). Mobilní telefon pak klientovi přehledně zobrazí několik nabídek, včetně kontaktů na bankovní ústavy.

Hlavním přínosem projektu je umožnit klientům nezkreslený pohled na nabídku půjček především bankovních ústavů s tím, že se nebude muset obracet na „samozvané“ bankovní poradce, ale bude moci přímo vybírat z produktů velkých bankovních ústavů. Bankovní ústavy tak budou moci přímo nabízet své produkty daleko širším vrstvám populace.

Orientovat se v nabídkách, které nejsou nikterak centralizované je vždy minimálně časově náročné. Pokud se nabídky změní, je třeba obětovat neméně úsilí k obnovení všech informací. Jaké jsou tedy výhody centralizace? Například internetové vyhledávače. Dnes si již mnoho lidí myslí, že co nenajde ve svém internetovém vyhledávači prostě neexistuje. Podobné a další výhody lze získat i centralizací nabídek půjček. Uživateli se navíc pomocí tohoto projektu otevře i možnost

přistupovat k těmto informacím prostřednictvím svého mobilního telefonu. Informace se mu tedy stanou dostupné ve chvílích, kdy je opravdu potřebuje. Jaké volby bude uživatel mít, zobrazuje následující výčet.

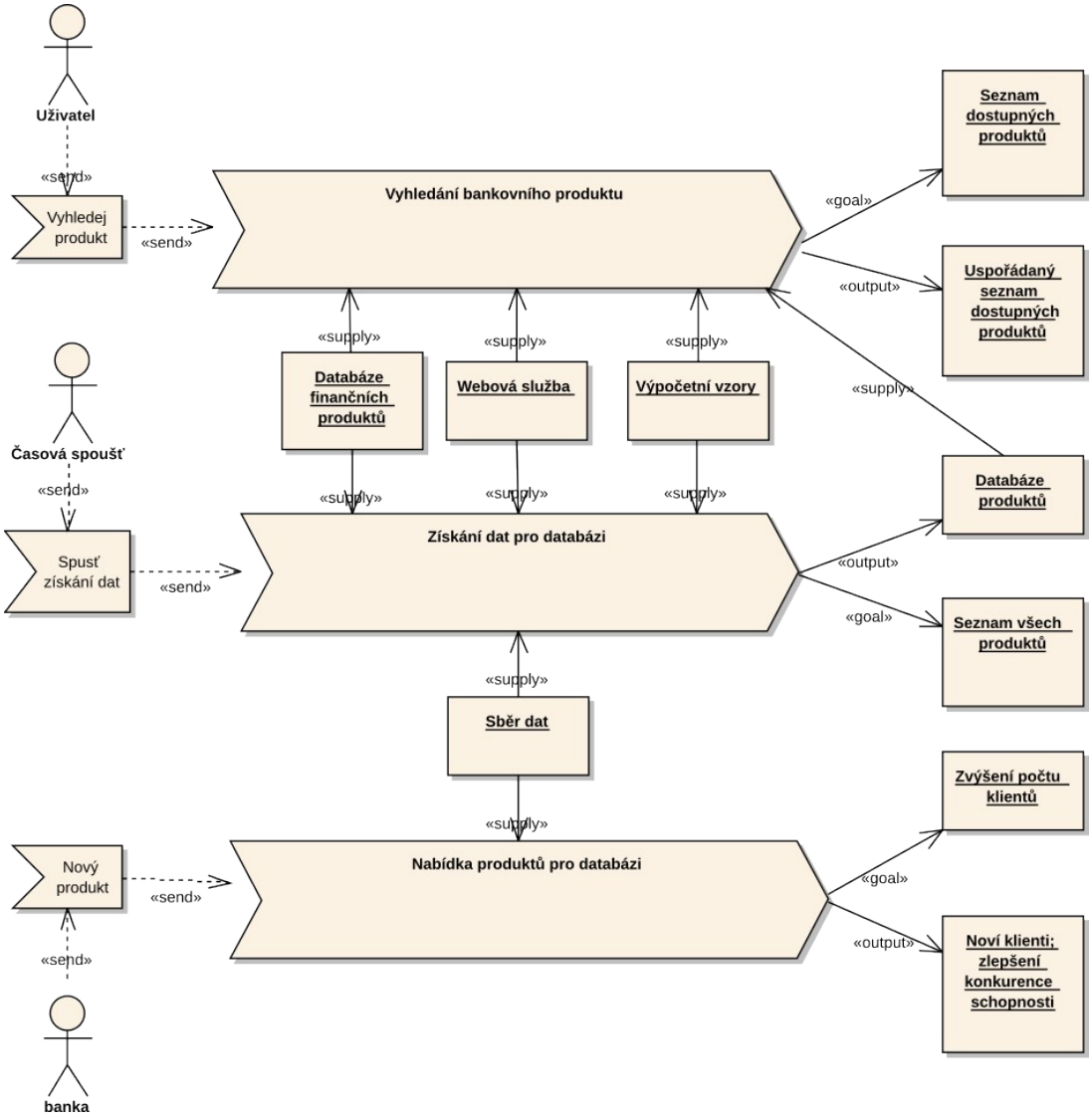
1. Vybere si konkrétní službu
   Každá nabídka úvěru může mít různě nastavené velikosti splátek a poplatků v závislosti na době splatnosti a velikosti půjčky. Tato volba nalezne nejvýhodnější konfigurace pro konkrétní službu.

2. Nevybere si konkrétní službu
   Tato volba obecně nalezne nejzajímavější konfigurace v celé databázi produktů.

3. RPSN kalkulátor
   Uživatel definuje parametry úvěru, pro které je následně spočítáno RPSN.

2.1 Diagram datových toků

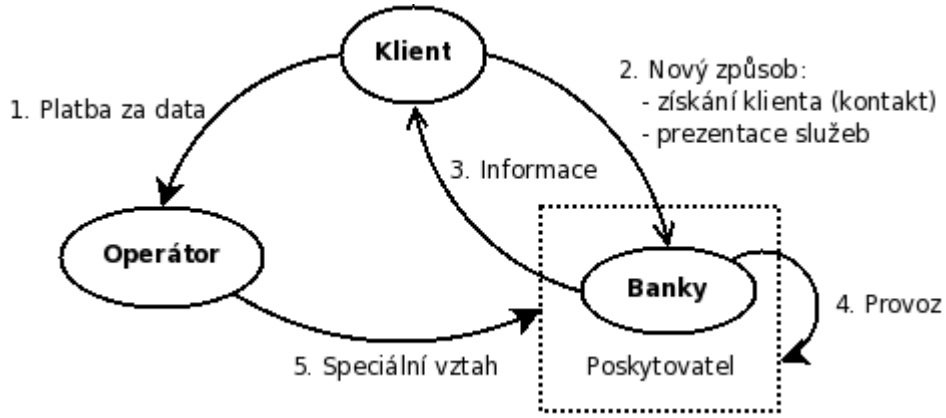
Na diagramu datových toků je vidět řídící toky (send), vzájemnou podporu (supply), výstupy (output) a cíle (goal) jednotlivých bloků. Spouštění a získávání dat nebude součástí této implementace, neboť crawlování a parsování webu s údaji bank obsahově překračuje rámec této práce. Data jsou prozatím do databáze přidávána ručně.



Obr. 2.1 – Diagram datových toků (DFD)

2.2 Obchodní model

Před zavedením služby do provozu je vhodné znát přínosy a investice pro zainteresované subjekty. Analýzu zobrazuje následující diagram.



Obr. 2.2 – Obchodní model

Vysvětlení toků:

1. Platba za data - Klient zaplatí operátorovi za přenos dat.
2. Získání klienta a prezentace produktů - Konkrétní banka získává klienta se zájmem o nabízený produkt.
3. Informace klientovi - Klient obdrží seznam nejzajímavějších nabídek dle jeho potřeb.
4. Provoz - Je třeba udržovat server splňující technické požadavky.
5. Speciální vztah – Banka zvětšuje využívání datových služeb operátora, po společné dohodě může být tento vztah definován a poskytovatel odměněn.

3 PROJEKTOVÁ ČÁST

V rámci projektu musí být ošetřený následující funkce.

- Databáze
  - uložení dat
  - přístup k datům
- Služba
  - distribuce funkcí
  - struktura pro přenos dat
  - aplikační logika
- Mobilní klient
  - GUI klienta
  - přístup ke službě

3.1 Vývojové nástroje

Při volbě vývojových nástrojů je třeba zohlednit programovací jazyk, ve kterém má být aplikace napsána. Čím více podpůrných nástrojů vývojové nástroje mají, tím více bývají provázané s určitým jazykem. Všechny níže popsané vývojové nástroje jsou získány legální cestou s nulovou pořizovací cenou, protože je tento software nabízen k užívání zdarma.

Pro vývoj aplikace v jazyce Java byl vybrán vývojový nástroj Eclipse, který je dostupný pro operační systém Linux i Windows. Jsou s ním výborné zkušenosti i při vývoji aplikací v jiných jazycích. Nemá sice přímou podporu pro vizuální tvorbu grafického uživatelského rozhraní, ale jazyk Java je navržen tak, aby bylo možné tvořit GUI přímo ve zdrojovém kódu. Eclipse má navíc možnost přidávání zásuvných balíčků s různými rozšířeními. Mezi výhody tohoto prostředí patří implementace refaktoringu, podpora pro generování dokumentace, testování, vizuální rozbalení a sbalení kódu uvnitř metod, code-inside a focus tříd a metod.

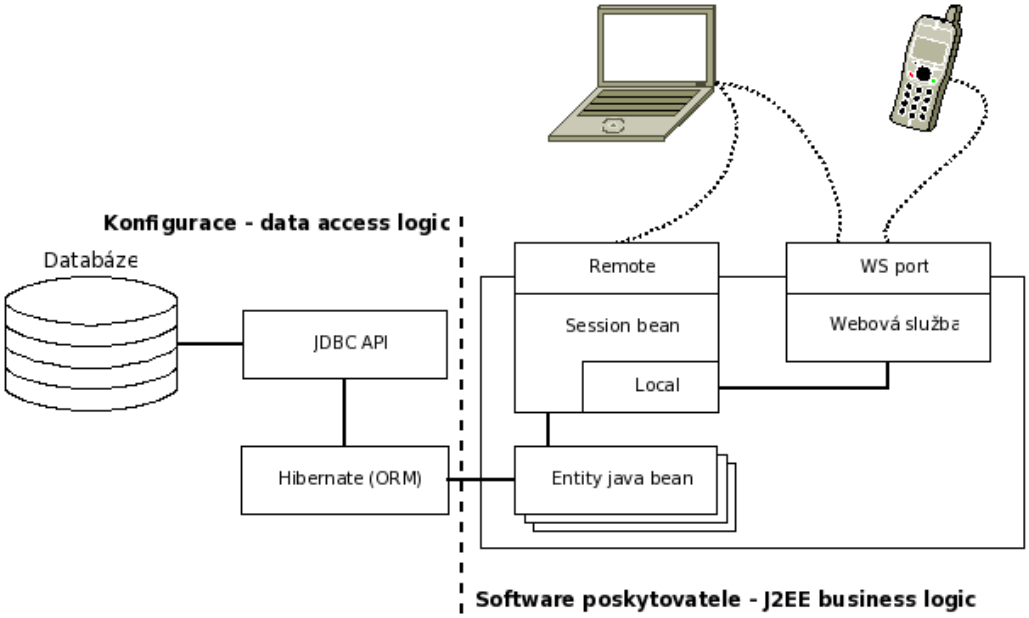
Návrh diagramu tříd byl vytvořen pomocí rozšíření Green UML. Tento nástroj umožňuje provádět návrh diagramu tříd z okamžitým promítnutím do kódu a opačně i každá editace kódu se projeví v diagramu tříd. Nástroj navíc umí i reverse engineering pro tvorbu diagramu ze zdrojových kódů. Model databáze byl navržen pomocí vizuálního nástroje MySQL Workbench, který je zdarma dostupný pro operační systémy Linux i Windows.

Dalším důležitým nástrojem pro psaní komplexního sestavovacího schémata k aplikaci je Apache ANT, který provede kompilaci zdrojového kódu, vyřeší závislosti a výsledný kód implementuje. Nástroj ANT je možné spustit z příkazové řádky nezávisle na operačním systému, jelikož je zapsaný pomocí XML. Příkaz *<copy />* se tedy chová očekávaným způsobem ve všech operačních systémech na rozdíl od unixového *cp*, který v systému Windows není definován. Pomocí ANTu je ošetřen překlad zdrojových kódů a implementace strany serveru.

Pro návrh mobilního klienta bylo vybráno rozšíření s názvem Eclipse ME. Toto rozšíření v sobě integruje jak vývoj, tak i emulaci pomocí WTK, takže po správném nastavení není třeba opouštět prostředí Eclipse.

3.2 J2EE

Při realizaci se postupovalo od vrstvy EIS, která podporuje vrstvu střední a ta nakonec vrstvu klienta. Součástí EIS vrstvy je připojení k databázi a způsob mapování dat. Je využit mapovací nástroj Hibernate. Ten je k relační databázi MySQL připojen rozhraním JDBC. Vyměňování dat probíhá pomocí Entitních Java Bean. Střední vrstva ošetřuje aplikační logiku, která je součástí bezstavové session Beany. Tato má dvě rozhraní: vzdálené a místní. Místní rozhraní je rozšířeno další bezstavovou session Beanou vystavenou jako webová služba. Ke vzdálenému rozhraní je možné přistupovat přes RMI-IIOP (EJB), což je Java Remothe Method Invocation běžící pomocí Internet Inter-Orb Protocol.



Obr. 3.1 – Model využití J2EE

3.2.1 Hibernate

Pro zprovoznění nástroje Hibernate je třeba vytvořit dva XML soubory s nastaveními. První se jmenuje *wirgeek-ds.xml* a týká se nastavení JTA datových zdrojů (viz. Výpis 3.1). Tag *jndi-name* nastavuje název datového zdroje pro vyhledávací službu JNDI. Pomocí tohoto názvu bude s tímto zdrojem dále manipulováno. Následující dva tagy ve výpisu uvozují cestu k databázi a ovladač, který bude připojení prakticky zajišťovat. Tagy *user-name* a *password* slouží k autentizaci programu v databázi. Posledním kritickým tagem je *type-mapping*, který nastavuje typ databáze pro mapování.

Výpis 3.1 – Soubor *wirgeek-ds.xml*

```
<datasources>
 <local-tx-datasource>
  <jndi-name>WirGeekDS</jndi-name>
```

```
  <connection-url>jdbc:mysql://localhost:3306/loan</connection-url>
  <driver-class>com.mysql.jdbc.Driver</driver-class>

  <user-name>loan</user-name>
  <password>2uEE5NtJc7raA4c4</password>

  <min-pool-size>5</min-pool-size>
  <max-pool-size>32</max-pool-size>

<connection-property name="autoReconnect">true</connection-property>

   <metadata>
   <type-mapping>mySQL</type-mapping>
  </metadata>
 </local-tx-datasource>
</datasources>
```

Druhý soubor nastavuje manažera pro práci s JPA (viz. Výpis 3.2). Jeho název je *wirgeek_persistence_manager*, připojuje se k datovému zdroji *WirGeekDS* a je používaný z balíku tohoto projektu *loanService.jar*. Komunikace bude probíhat v dialektu MySQL.

Výpis 3.2 – Soubor *persistence.xml*

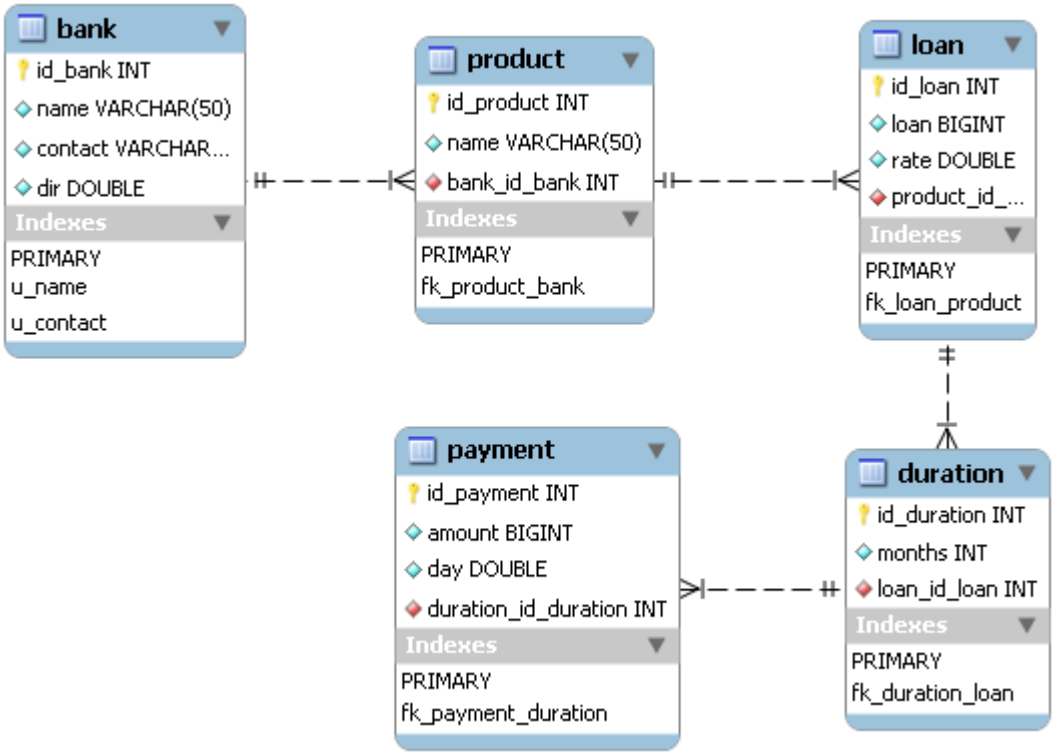
```
<persistence>
   <persistence-unit name="wirgeek_persistence_manager">
       <jta-data-source>java:/WirGeekDS</jta-data-source>
       <jar-file>../loanService.jar</jar-file>

       <properties>
            <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect"/>
       </properties>
   </persistence-unit>
</persistence>
```

Tyto soubory obsahují citlivé údaje, a proto je dobré při implementaci zvážit jejich čitelnost v souborovém systému pro jednotlivé skupiny uživatelů.

3.3 Služba

3.3.1 Uložení dat

Přehled aktuálních služeb, o kterých je klient informován, je uložen na straně serveru pomocí relační báze dat MySQL. Entitně relační diagram je na následujícím obrázku.



Obr. 3.2 – ERD databáze

3.3.2 Mapování dat

Se znalostí uspořádání dat v databázi lze přejít k mapování dat. Nejdříve byla ke každé databázové tabulce vytvořena třída POJO. Každá třída obsahuje soukromé atributy dle přidružené tabulky a veřejné metody pro jejich čtení a ukládání. Pomocí anotací JPA, které jsou v následujícím výpisu zvýrazněny, je pak provedeno vlastní

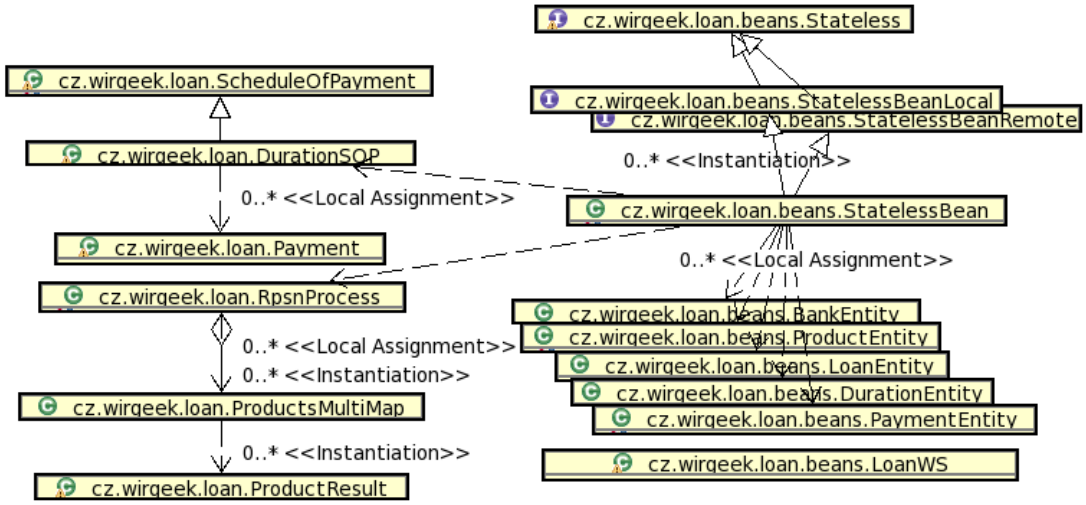
mapování. Anotace je ve většině případů platná pro bezprostředně následující kód. Anotace *@Entity* říká, že třída je Entity Java Bean a *@Table* definuje název propojené tabulky z databáze. Mapování obecného atributu je provedeno anotací *@Basic* a klíčového pomocí *@Id*. Jak již bylo řečeno, ORM umožňuje i relační mapování. Relace 1:N se zapisuje anotací *@OneToMany*. Ta říká, že tento produkt vlastní kolekci objektů *LoanEntity*, které musí samozřejmě být také Entity Java Beany. Relace N:1 jsou zapsány pomocí *@ManyToOne*, kde se pomocí *@JoinColumn* nastavuje název cizího klíče. K čemu se hodí mapování relací? Pokud je z dotazu na databázi obdržena instance na *ProductEntity*, může se programátor volně hýbat ve struktuře, s kterou je tato instance spjata. Například instanci na banku *BankEntity*, která vlastní tento produkt lze získat pomocí metody *productEntity.getBank()*, a to již bez nutnosti volat další výběrový dotaz v databázi.

Výpis 3.3 – Mapování tříd na databázi

```
package cz.wirgeek.loan.beans;

import java.io.Serializable;
import java.util.Collection;

import javax.persistence.*;
/**
 * Entity bean for table "product".
 *
 * @author Petr Žáčík
 */
@Entity
@Table(name = "product")
public class ProductEntity implements Serializable {

      @Id
      @GeneratedValue(strategy = GenerationType.IDENTITY)
      private int id_product;

      @ManyToOne
      @JoinColumn(name = "id_bank")
      private BankEntity bank;

      @OneToMany(mappedBy = "product")
      private Collection<LoanEntity> loanEntities;

      @Basic
      private String name;
// zbytek kódu vynechán. Obsahuje gettery a settery atributů.
```

3.3.3 Diagram tříd

Kód je podle účelu rozdělen do několika balíků. V balíku *cz.wirgeek.loan* jsou umístěny třídy struktur a výpočtů týkající se půjček. Výhodou je jejich znovupoužitelnost. Mohou být podle uvážení použity v servletech, appletech atd. Balík *cz.wirgeek.loan.beans* se vztahuje k přístupu k databázi a zprostředkování dat pomocí technologie J2EE. Je to podbalík balíku *loan*, jelikož bez něho nemá smysl. Část serveru se skládá ze třinácti tříd a třech rozhraní. Je vyobrazena na následujícím obrázku. Pro přehlednost nejsou zobrazeny atributy a metody jednotlivých tříd.



Obr. 3.3 – Diagram tříd poskytovatele

Ústřední třídou je třída *StatelessBean*. Jde o bezstavovou session Beanu, která musí implementovat rozhraní pro místní a vzdálené volání. Jelikož mají obě tato rozhraní stejné metody, rozšiřují pouze rozhraní *Stateless*. Tato třída je jediná, která má styk s daty a to pomocí entitního manažera, jak je to vidět na využití tříd pro mapování tabulek z databáze. Na diagramu jsou to třídy, jejichž název končí slovem „Entity“. Metody *StatelessBean* jsou prodlouženy do webové služby *LoanWS*, která se navíc stará o převedení struktur nevhodných pro přenos pomocí SOAP do vhodnější podoby.

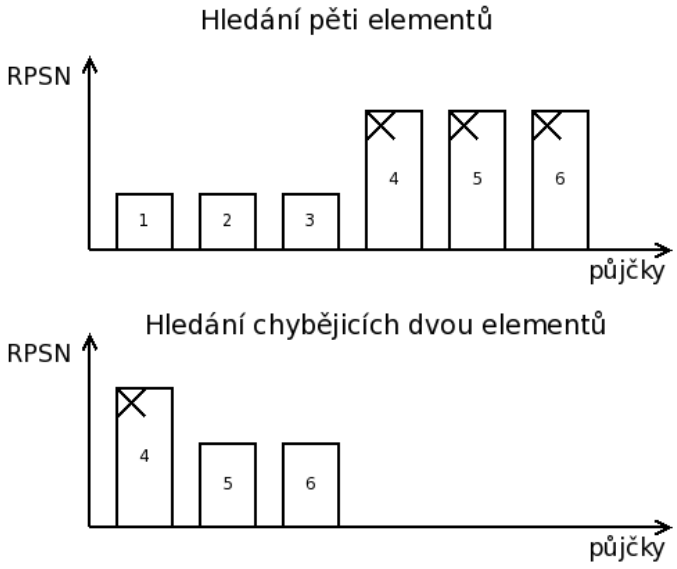
ScheduleOfPayment je splátkový kalendář pro uložení parametrů jednoho produktu včetně poplatků, které jsou agregovány v kolekci instancí třídy *Payment*. Třída obsahuje i metody pro výpočet RPSN. *DurationSOP* je její rozšíření, které zavádí konstruktor pro vytvoření splátkového kalendáře z *DurationEntity*. Toto rozšíření je nutné z důvodu, že třída *ScheduleOfPayment* se používá pro přenos dat mezi webovou službou a klientem a čím méně tříd importuje, tím jednodušší je automaticky generované WSDL dané služby, zvláště když v sobě *DurationEntity* vzhledem k uspořádání dat a vzájemným relacím kaskádovitě drží informace o všech nadřazených entitách podle relace 1:1 a o kolekci podřazených entit podle relace 1:N. To, že by struktura těchto tříd byla popsána definicí webové služby, by nikterak nezlepšilo funkcionalitu.

O řízení výpočtu včetně heuristiky se stará třída *RpsnProcess*. Mapa produktů, které porovnává je uložena ve třídě *ProductsMultiMap*. Výsledný seznam produktů je převeden do struktury podle předpisu *ProductResult*, která je schopna přenosu přes SOAP.

3.3.4 Heuristika

Se zvětšujícím se počtem produktů v databázi se bude zvětšovat i čas potřebný k vyhodnocení. Výpočet hledá specifický počet produktů s nejnižším RPSN. Provádí se půlením intervalu u všech produktů současně, dokud není možná odchylka dostatečně nízká nebo dokud není výběr redukován na hledaný počet produktů. Zde je implementována heuristika tak, že produkty, které se po půlení zařadí do vyššího intervalu jsou odstraněny z množiny hledaných produktů a to pouze v případě, že existuje alespoň jeden produkt zařazený v nižším intervalu. Může nastat případ, že je odstraněno příliš mnoho produktů a hledání tím podteče. Zde se aplikuje rekurze na naposledy odstraněné produkty a hledá se zbývající množství produktů.

Příklad hledání pěti produktů z šesti nastiňuje následující obrázek. Po provedení prvního půlení intervalu jsou vyřazeny produkty čtyři až šest, protože jsou všechny zařazeny ve vyšším intervalu. Jelikož ale hledání podteklo, je na odstraněné produkty aplikována rekurze, ovšem teď jsou hledány pouze dva zbylé produkty. Vynořením z rekurzí je zpětně sestaven kompletní seznam hledaných produktů.



Obr. 3.4 – Využití heuristiky

Setřídění produktů je v této fázi kompletní, nikoliv však výpočet RPSN, který mohl skončit dříve, než byl určen s dostatečnou přesností. O dopočtení se stará getter *getRpsn* konkrétního produktu *ScheduleOfPayment*, který dokončí výpočet z místa, kde skončilo třídění. Žádný výpočet tedy není redundantní.

3.3.5 Apache ANT – sestavovací schéma aplikace

Ant tohoto projektu je uložen v souboru *build.xml*. Nastaveními cest je pak uloženo v souboru *build.properties*. Soubor automatizuje proces překladu a vystavení webové služby na server. Následující výpis obsahuje jeho nedůležitější cíle.

Výpis 3.4 – Nejdůležitější cíle v souboru *build.xml*

```
<target name="install" depends="build">
  <antcall target="deploy-ds" />
  <delete file="${jboss_deploy}/${dist_file_name}" />
  <copy todir="${jboss_deploy}" file="${dir_dist}/${dist_file_name}"
/>
</target>

<target name="build" depends="init">
  <javac srcdir="${dir_src}" destdir="${dir_build}"
classpathref="jboss.as" debug="on" />
  <jar destfile="${dir_dist}/${dist_file_name}" basedir="$
{dir_build}">
    <metainf file="${dir_files}/persistence.xml" />
  </jar>
</target>

<target name="init">
  <antcall target="clean" />
  <mkdir dir="${dir_build}" />
  <mkdir dir="${dir_dist}" />
</target>

<target name="clean">
  <mkdir dir="${dir_build}" />
  <delete>
    <fileset dir="${dir_build}">
      <include name="*" />
    </fileset>
  </delete>
  <delete dir="${dir_dist}" />
</target>

<target name="deploy-ds">
  <delete file="${jboss_deploy}/wirgeek-ds.xml" />
  <copy todir="${jboss_deploy}" file="${dir_files}/wirgeek-
ds.xml" />
</target>
```

Cíl *init* a *clean* připraví adresářovou strukturu, *build* přeloží zdrojový kód do bajtového, zabalí jej do Javového archivu jar a do adresáře *META-INF*, který je součástí struktury archivu, je přidán soubor *persistence.xml*. Cíl *install* vystaví produkovaný archiv na server JBoss a *deploy-ds* do adresáře s vystaveným archivem nakopíruje soubor s nastavením datových zdrojů.

Soubory mají dohromady 90 řádů kódu, ale skutečně ulevují při práci s překladem a implementací, kterou lze nyní spustit jednoduchým příkazem z příkazové řádky v adresáři se souborem *build.*xml a *build.properties* podle zkráceného výpisu 3.5.

Výpis 3.5 – Překlad a instalace pomocí ANTu

```
ant]$ ls
build.properties  build.xml
ant]$ ant
Buildfile: build.xml

init:
clean:
build:
install:
deploy-ds:

BUILD SUCCESSFUL
Total time: 0 seconds
```

3.4 Mobilní klient

Mobilní klient musí podporovat CLDC 1.1, MIDP 2.0 a JSR-172. Tyto požadavky se však stávají pro dnešní mobilní zařízení standardem.

Mobilní klient má za úkol přistupovat k rozhraní webové služby, pomocí něhož odesílá požadavky ke zpracování a přijímá jejich výsledky. Současně je dostatečně přehledný a jednoduchý na ovládání.

3.4.1 Přístup ke službě

Po implementaci serveru může být vygenerován stub, což je objekt pro volání vzdálených procedur, který bude součástí mobilního klienta. Generování se provádí pomocí nástroje WTK s názvem *wscompile* a zajišťuje ho skript *wscompile.*sh dle následujícího výpisu.

Výpis 3.6 – Skript pro generování stubu *wscompile.sh*

```
#! /bin/bash
rm -rf src/cz/wirgeek/loan/client/midlet/stub
wscompile -cldc1.1 -gen:client -d src -classpath src config.xml
```

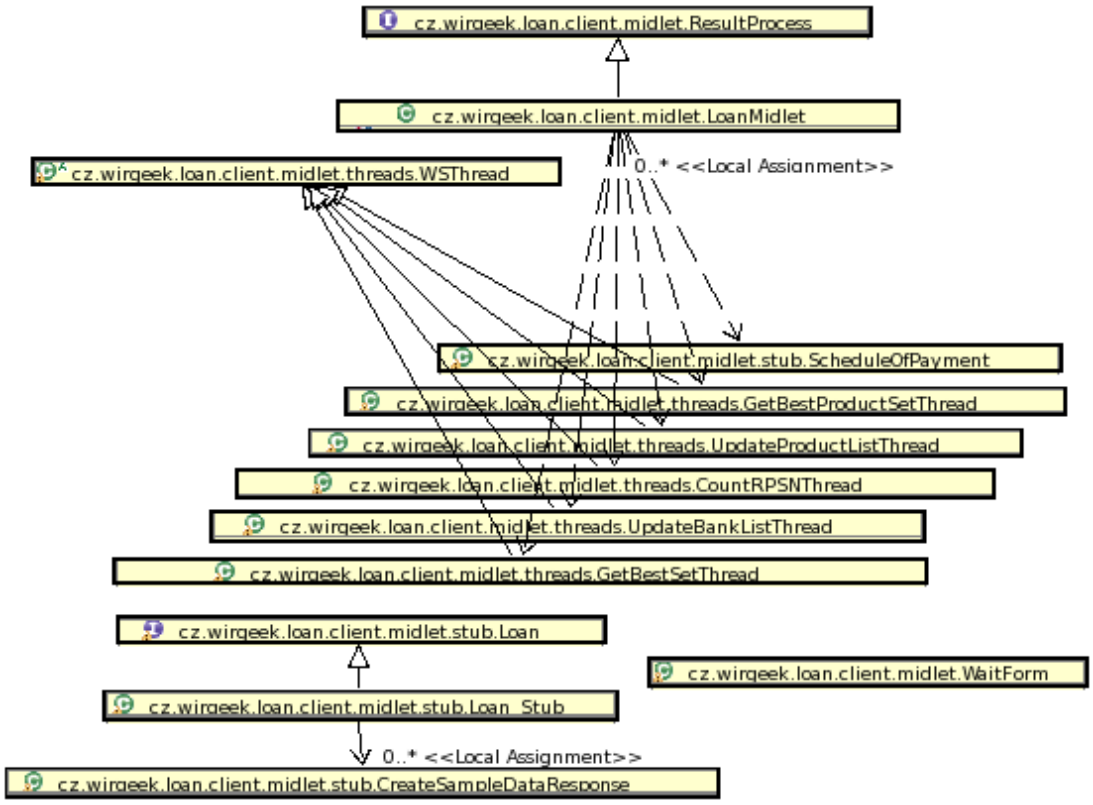
Program *wscompile* se řídí podle konfigurace uložené v XML souboru *config.xml*, kde vyčte informace o umístění webové služby, umístění WSDL a názvu balíčku vygenerovaného stubu.

Výpis 3.7 – Konfigurační soubor *config.xml* programu *wscompile*

```
<?xml version="1.0" encoding="UTF-8"?>
  <configuration
     xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config"
     xmlns:tns="http://127.0.0.1:8080/loanService/LoanWS">
     <wsdl location="http://127.0.0.1:8080/loanService/LoanWS?wsdl"
     packageName="cz.wirgeek.loan.client.midlet.stub"/>
  </configuration>
```

3.4.2 Diagram tříd

Třídy klienta jsou rozděleny do podbalíků *cz.wirgeek.loan.client.* V balíku *midlet* jsou třídy určující chování midletu, balík *stub* obsahuje třídy automaticky generovaného stubu a *thread* uchovává třídy vláken. V následujícím diagramu tříd je zvětšena přehlednost tím, že nejsou zobrazeny některé třídy, které se zabývají komunikací s webovou službou a jsou součástí generovaného stubu.



Obr. 3.5 – Diagram tříd mobilního klienta

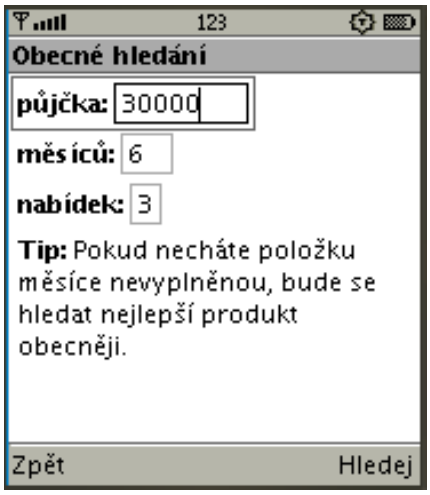
Standardní třída *Midlet* je rozšířena třídou *LoanMidlet*, která ošetřuje základní chování midletu. Jelikož je síťová operace volání metody webové služby proces blokující, jsou jednotlivá volání umístěna ve vláknech. Vlákna spojuje rozšíření ze třídy *WSThread*, které vlastní ukazatel na stub webové služby. Instance vlákna se předává třídě *WaitForm*. Ta spustí vlákno a informuje o dokončení posluchače, který implementuje rozhraní *ResultProcess*, v tomto případě je to *LoanMidlet*.

3.4.3 GUI

Grafické uživatelské rozhraní je navrženo tak, aby uživateli umožnilo co nejrychlejší a nejpřehlednější práci. Výběr se provádí navigačními tlačítky, příkazy jsou volány pomocí pravé a levé funkční klávesy. Data se zadávají numerickými klávesami, přičemž pole pro zadávání čísel automaticky přepnou klávesnici pro vstup čísel.

Po spuštění aplikace a zobrazení uvítací obrazovky se uživatel ocitá v hlavní nabídce (viz Obr. 3.6). První položka uvede uživatele do formuláře pro zadávání dat obecného hledání (viz Obr. 3.7). Pokud zvolí druhou položku, vyhledává konkrétní službu. V tomto případě uživatel projde dva seznamy. Seznam ústavů poskytujících půjčky a seznam jejich produktů. Na obrázku 3.8 se vybírá nejlepší konfigurace produktu s názvem „A1“. Požadavky nalezení služeb mají za výstup seznam podle obrázku 3.9. Poslední volbou v hlavní nabídce je RPSN kalkulátor (viz Obr. 3.10), který spočítá roční procentní sazbu nákladů podle zadaných parametrů.

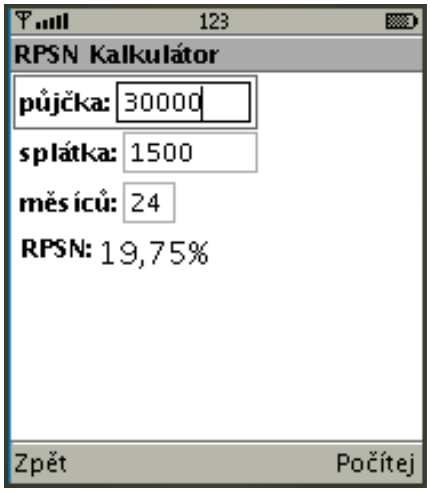
Obr. 3.6 – Hlavní nabídka



Obr. 3.7 – Formulář pro obecné hledání

Obr. 3.8 – Formulář pro hledání konkrétního produktu

Obr. 3.9 – Seznam nalezených služeb



Obr. 3.10 – RPSN kalkulátor

3.5 Bezpečnost

Projekt díky využitým technologiím umožňuje velkou flexibilitu. Díky možnostem nastavování a znovupoužitelnosti ale může vzniknout bezpečností riziko.

Každý program či systém může obsahovat bezpečnostní díru, kterou využívají programy tzv. exploity psané útočníkem. Proti exploitům se lze bránit častější aktualizací software. Síťovým útokům je možné zamezit vhodným nastavením firewallu. Z pohledu subjektů, které do systému zasahují mohou být aktiva ohrožena:

- Klientem

  Klient odesílá dotazy, kterými nikterak neovlivňuje nastavení serveru. Jediným možným útokem je útok typu Denial-of-service, tedy zahlcení serveru dotazy. Tento útok lze odstranit například hlídáním počtu příchozích paketů v čase firewallem.

- Poskytovatelem

  Poskytovatel zajišťuje spojení sítě klientského zařízení se sítí poskytovatele. Zde by mohl vzniknout vlivem technického zázemí poskytovatele výpadek služby. Dnes je tento jev zanedbatelný.

- Bankou

  Z této strany nehrozí jiný typ útoku než na vlastní důvěryhodnost v případě, že by o sobě uvedla nepravdivé informace. Jelikož budou data získávána ze serverů jednotlivých bank, riziko útoku je minimální. Pokud bude server umístěn v neutrálním prostředí, nehrozí ani vzájemné ovlivňování.

- Třetí osobou

  Třetí osoba může útočit na komunikaci mezi klientem a službou. Pokud by byla přenášená data změněna útokem Man-in-the-middle, šlo by o útok na důvěryhodnost. Šifrování lze provést implementací WS-Security. Útoku Denial-of-service lze opět předcházet nastavením firewallu.

3.6 Implementace a testování

Server i klient byl implementován a testován na notebooku s procesorem TL52, operační pamětí 2GB a operačním systémem Linux Fedora 8. Komunikace mezi klientem a serverem probíhala přes rozhraní loopback síťové karty.

3.6.1 Server

Konfigurace serveru proběhla v následujícím pořadí:

- instalace JDK 5
- instalace aplikačního serveru JBoss 4.2.2
- instalace a konfigurace relační databáze MySQL 5.0.45

Podpora Javy – JDK 5

Aby na straně serveru bylo možné spouštět Javové aplikace včetně Jboss aplikačního serveru, je třeba instalovat Java development kit (JDK). Plná kompatibilita je zaručena pro JDK 5. Aktuální verze v době tvorby této práce je 1.5.0 update 15. Instalační soubor *jdk-1_5_0_15-linux-i586-rpm.bin* lze stáhnout ze stránek společnosti Sun. Instalace se provede dle výpisu 3.8.

Výpis 3.8 – Instalace JDK

```
]# chmod +x jdk-1_5_0_15-linux-i586-rpm.bin
]# ./jdk-1_5_0_15-linux-i586-rpm.bin

]# java -version
java version "1.7.0"
IcedTea Runtime Environment (build 1.7.0-b21)
IcedTea Server VM (build 1.7.0-b21, mixed mode)
```

Po úspěšné instalaci ještě soubor */usr/bin/java* a */usr/bin/javac* nemusí ukazovat na požadovanou verzi Javy. Jak ukazuje předešlý výpis, verze se neshodují. Ke změnám těchto linků se ve Fedoře používá nástroj alternatives, do kterého se nainstaluje nová cesta k souboru parametrem install a následně se na tento soubor přepne parametrem config.

Výpis 3.9 – Konfigurace JDK

```
]# alternatives --install /usr/bin/java \
java /usr/java/jdk1.5.0_15/bin/java 2
]# alternatives --install /usr/bin/javac \
java /usr/java/jdk1.5.0_15/bin/javac 2

]# alternatives --config java

4 programů poskytuje 'java'.

  Výběr        Příkaz
-----------------------------------------------
*+ 1           /usr/lib/jvm/jre-1.7.0-icedtea/bin/java
   2           /usr/lib/jvm/jre-1.5.0-gcj/bin/java
   3           /usr/java/latest/bin/java
   4           /usr/java/jdk1.5.0_15/bin/java

Enterem zachováte aktuální výběr [+], nebo napište číslo výběru: 4

]# java -version
java version "1.5.0_15"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_15-
b04)
Java HotSpot(TM) Server VM (build 1.5.0_15-b04, mixed mode)

]# alternatives --config javac

3 programů poskytuje 'javac'.

  Výběr        Příkaz
-----------------------------------------------
*+ 1           /usr/lib/jvm/java-1.7.0-icedtea/bin/javac
```

```
   2           /usr/java/latest/bin/javac
   3           /usr/java/jdk1.5.0_15/bin/javac
Enterem zachováte aktuální výběr [+], nebo napište číslo výběru: 3

]# javac -version
javac 1.5.0_15
```

Posledním krokem je editace souboru /*etc/profiles*, kde je přidáním řádku z výpisu 3.10 nastaven pracovní adresář Javy.

Výpis 3.10 – Nastavení pracovního adresáře Javy v souboru */etc/profiles*

```
# Java paths
export JAVA_HOME="/usr/java/jdk1.5.0_15"
```

Po těchto krocích je Java 1.5.0 update 15 připravena k použití.

AS JBoss 4

I když má Fedora možnost instalovat balíčky přímo z repozitářů, aplikační server JBoss byl stažen ze stránek *http://www.jboss.org/projects/download/* a instalován do adresáře */opt* jak ukazuje výpis 3.11. Ve výpisu 3.12 je spouštěcí skript, kterým se bude aplikační server spouštět.

Výpis 3.11 – Instalace AS JBoss

```
/opt]# unzip jboss-4.2.2.GA.zip
/opt]# ln -s jboss-4.2.2.GA jboss4
```

Výpis 3.12 – Spouštěcí skript AS JBoss

```
]# cd /root/bin
]# cat jboss4
#! /bin/bash
cd /opt/jboss4/bin
./run.sh
cd -
```

Po instalaci lze provést implementaci webové služby, a to nejlépe prostřednictvím ANTu, nebo zkopírováním souborů *loanService.jar* a *wirgeek-ds.xml* do adresáře pro vystavení služeb.

Databáze MySQL

Databáze byla instalována z repozitářů pomocí aktualizačního programu *yum* podle výpisu 3.13. Nastavení se týká zavedení znakové sady UTF-8 standardně pro server i připojované klienty. Tímto je pro většinu případů zajištěna podpora českých znaků. Nastavení se provádí v souboru */etc/my.cnf*.

Výpis 3.13 – Instalace MySQL

```
]# yum install mysql* -y
```

Výpis 3.14 – Soubor */etc/my.cnf*

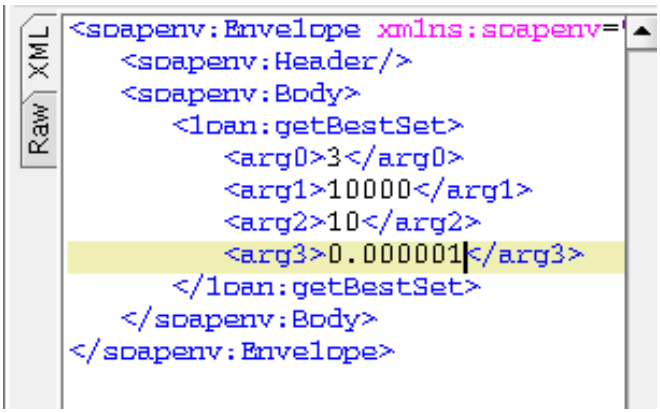
```
[client]
default-character-set=utf8

[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Default to using old password format for compatibility with mysql
3.x
# clients (those using the mysqlclient10 compatibility package).
old_passwords=1
default-character-set=utf8

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

Testování

Objektem testování je síťový provoz a jeho obsah. Jako testovací nástroj byl vybrán program wireshark a soapUI.

SoapUI je jednoduchý Javový program, který dokáže simulovat a testovat webové služby na úrovni SOAP. V této fázi je již webová služba vystavena, takže se nyní ověří její reakce. Po vytvoření nového projektu z WSDL souboru umístěného na aplikačním serveru se zobrazí seznam metod, které jsou z definice extrahovány. Po výběru metody getBestSet, která má vrátit seznam doporučovaných produktů v závislosti na parametrech požadavku se objeví editační okno, kde je možné

parametry ručně doplnit jak je patrné na obrázku 3.11. Význam parametrů je možné vyčíst z rozhraní webové služby. Jsou to vzestupně pro argument 0 až 4: počet hledaných produktů, požadovaná půjčka, minimální období trvání půjčky v měsících a přesnost vypočteného RPSN. Na obrázku 3.12 je odpověď webové služby zachycená taktéž programem soapUI.



Obr. 3.11 – Definování požadavku metody *getBestSet*

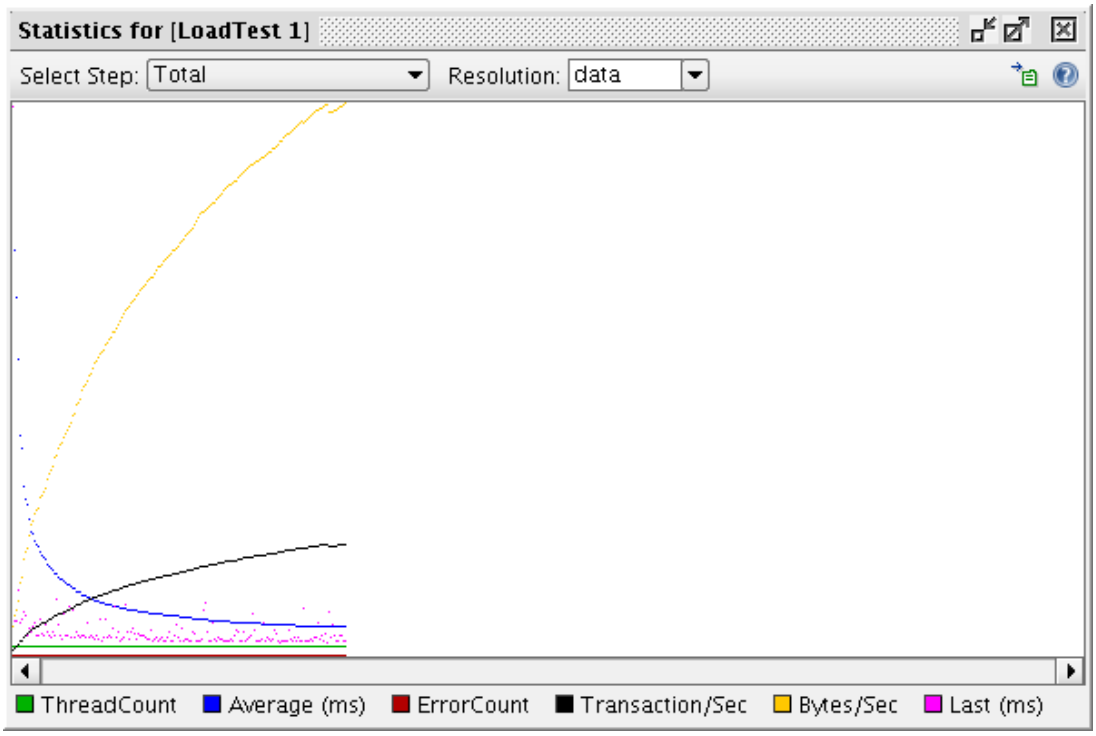
Současně se soapUI byl spuštěn program wireshark, který zachytával síťové pakety na rozhraní loopback (síťová adresa 127.0.0.1). V komunikaci byly zaznamenány protokoly HTTP pro přenos SOAP (viz Obr. 3.13) i protokol MySQL pro hledání informací o produktu v databázi (viz Obr. 3.14). Analýza TCP proudu tedy ukázala podobný obraz jako editor soapUI, tentokrát však na úrovni síťových protokolů. Červeně je označena komunikace klienta, modře pak odpověď serveru.

```
POST /loanService/LoanWS HTTP/1.1
Content-Type: text/xml;charset=UTF-8
SOAPAction: ""
User-Agent: Jakarta Commons-HttpClient/3.0.1
Host: 127.0.0.1:8080
Content-Length: 360

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:loan="http://127.0.0.1:8080/loanws/
LoanWS">
   <soapenv:Header/>
   <soapenv:Body>
      <loan:getBestSet>
         <arg0>3</arg0>
         <arg1>10000</arg1>
         <arg2>10</arg2>
         <arg3>0.000001</arg3>
      </loan:getBestSet>
   </soapenv:Body>
</soapenv:Envelope>

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.4; JBoss-4.2.2.GA (build: SVNTag=JBoss_4_2_2_GA date=200710221139)/Tomcat-5.5
Content-Type: text/xml;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 02 May 2008 15:51:42 GMT

3a9
<env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'><env:Header></
env:Header><env:Body><ns2:getBestSetResponse xmlns:ns2="http://127.0.0.1:8080/loanws/LoanWS"><return><bankContact>e-
mail: alfa@alfa.cz</bankContact><bankName>Alfa</bankName><duration>24</duration><loanCost>12096</
loanCost><productName>P..j..ka A1</productName><rpsn>20.7240441858768 46</rpsn><rpsnString>20,72</rpsnString></
return><return><bankContact>e-mail: alfa@alfa.cz</bankContact><bankName>Alfa</bankName><duration>24</
duration><loanCost>12096</loanCost><productName>P..j..ka A2</productName><rpsn>20.724044185876846</
rpsn><rpsnString>20,72</rpsnString></return><return><bankContact>e-mail: beta@beta.cz</bankContact><bankName>Beta</
bankName><duration>20</duration><loanCost>11740</loanCost><productName>P..j..ka B1</
productName><rpsn>20.682435983419417</rpsn><rpsnString>20,68</rpsnString></return></ns2:getBestSetResponse></
env:Body></env:Envelope>
0
```

Obr. 3.13 – Požadavek a odpověď zachycená programem wireshark

```
.....SELECT * FROM duration WHERE (months, id_loan) IN (SELECT MIN(months), id_loan FROM duration WHERE months >
10 AND id_loan IN (SELECT id_loan FROM loan WHERE (loan, id_product) IN (SELECT MIN(loan), id_product FROM loan
WHERE loan > 10000 GROUP BY id_product)) GROUP BY
id_loan);.....@....def.loan.duration.duration.id_duration.id_duration.?.......B...8....def.loan.duration.duration.id
...!.0....select loanentity0_.id_loan as id1_2_2_, loanentity0_.id_product as id4_2_2_, loanentity0_.loan as
loan2_2_, loanentity0_.rate as rate2_2_, productent1_.id_product as id1_4_0_, productent1_.id_bank as id3_4_0_,
productent1_.name as name4_0_, bankentity2_.id_bank as id1_0_1_, bankentity2_.name as name0_1_,
bankentity2_.contact as contact0_1_, bankentity2_.dir as dir0_1_ from loan loanentity0_ left outer join product
productent1_ on loanentity0_.id_product=productent1_.id_product left outer join bank bankentity2_ on
productent1_.id_bank=bankentity2_.id_bank where
loanentity0_.id_loan=2.....9....def.loan.loanentity0_.loan.id1_2_2_.id_loan.?.......B...<....def.loan.loanentity0_.l
id_product.?.......P...6....def.loan.loanentity0_.loan.loan2_2_.loan.?...........6....def.loan.loanentity0_.loan.rat
id_product.?.......B...<....def.loan.productent1_.product.id3_4_0_.id_bank.?.......P...9....def.loan.productent1_.pr
.def.loan.bankentity2_.bank.name0_1_.name.!.......P...<....def.loan.bankentity2_.bank.contact0_1_.contact.!.......P.
.....?....2.1.49000.18.9.1.1.P..j..ka A1.1.Alfa.e-mail: alfa@alfa.cz.365.........0....select loanentity0_.id_loan
```

Na metodě getBestSet byl proveden zátěžový test. Bylo zjištěno, že webová služba je schopná odpovědět v rozmezí 30 až 1700 ms, průměrně 33 ms. Server byl před testem restartován a podle očekávání právě první odpověď trvala nejdéle. Zde se projevila rychlost zavedení služby do paměti. Rychlost obsluhy prvních 377 požadavků ukazuje následující orientační obrázek. Po opětovném spuštění testu se již největší prodleva pohybovala kolem 200 ms.

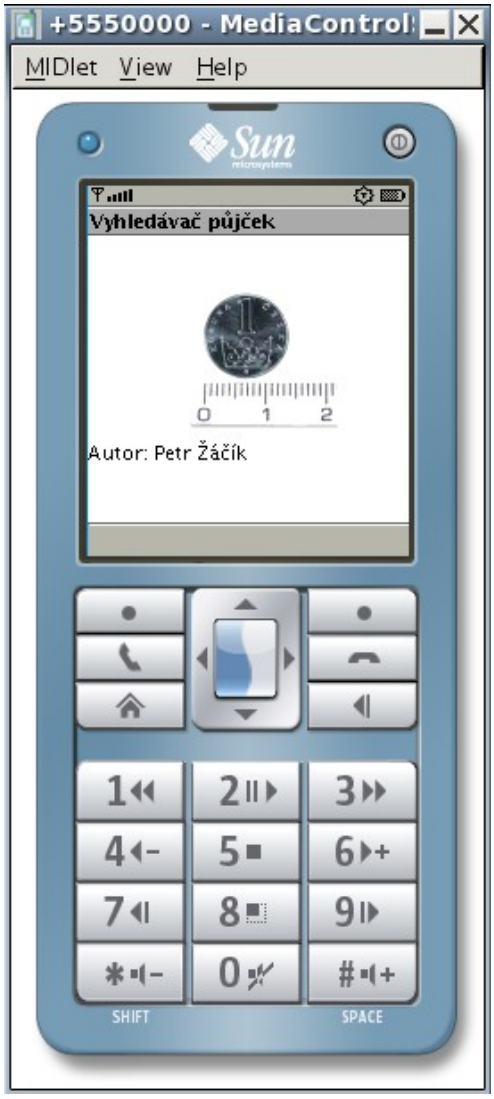


Obr. 3.15 – Rychlost odezvy serveru JBoss po restartu

3.6.2 Klient

Implementace se provede kopírováním souboru *LoanMidlet.jar* do mobilního zařízení. Zařízení balík samo rozpozná a aplikaci nainstaluje do složky v závislosti na typu mobilního zařízení.

Klient je testován pomocí emulačního nástroje balíčku WTK (viz Obr. 3.16) společnosti Sun (aktuálně verze 2.5.2). Po instalaci do libovolného domovského adresáře jsou k dispozici nástroje pro emulaci a balíčky pro vývoj mobilního klienta v Javě. Balíčky musí být importovány do classpath vyvíjeného midletu. Nástroje i balíčky jsou v adresáři *WTK2.5.2/bin*. Funkce midletu byly ověřeny a odpovídají návrhu.



Obr. 3.16 – Emulátor mobilního zařízení

Závěr

Prvním úkolem bylo studium technologií webových služeb a J2EE. Na základě těchto poznatků se pak navrhlo a realizovalo funkční nastavení serveru webových služeb, a to na aplikačním serveru JBoss. Funkční server společně s technologií J2ME pak poskytl dostatečnou základnu pro následný vývoj klienta webových služeb, speciálně v prostředí mobilních zařízení.

Vyvíjená aplikace vychází z reálných požadavků v oblasti půjček, kde může přinést mnoho výhod a nových možností jak pro banky, tak pro samotné uživatele. Uživatel si již nebude muset vyhrazovat volný den pro základní orientaci v problematice a produktech, ale vystačí si s informacemi, které se ho přímo týkají, a to kdekoliv prostřednictvím svého mobilního zařízení. Pro banky je to další možnost jak veřejně prezentovat své produkty a nástroj pro přímé porovnání s konkurencí. Aplikace se umístila na třetím místě v soutěži pořádané Mobile Payment Association pod Research and Development centrem v Praze, v kategorii nový produkt, což prokazuje hlad po podobných službách i krásu a výhodnost použitých technologií. Největším přínosem je pro mě však v tuto chvíli schopnost tyto technologie ovládat.

Výkladový slovník pojmů

API

API (Application Programming Interface) je rozhraní pro programování aplikací. Jde o seznam funkcí knihovny, programu nebo jádra operačního systému.

Autentizace

Autentizace je proces, který jednoznačně identifikuje subjekt, který dále pracuje se systémem. Autentizace probíhá například po zadání jména a hesla do systému.

Autorizace

Autorizace je proces, který ověřuje práva subjektu. Každý subjekt může mít jiná práva podle činnosti, kterou vykonává.

Bajtový kód

Po zkompilování zdrojového kódu v Javě vzniká bajtový kód, který je určený pro JVM a je nezávislý na architektuře počítače nebo zařízení.

CLDC (Connected Limited Device Configuration)

Definuje hardwarové požadavky a základní API pro platformu Javy v mobilních zařízeních.

Crawlování

Proces, při kterém jsou získávána data automatizovaným průchodem webových stránek.

EIS (Enterprise Information System)

Vrstva technologie J2EE zajišťuje obecně připojení a komunikaci s datovými zdroji.

EJB (Enterprise Java Bean)

Jsou to třídy psané v Javě, z kterých se tvoří střední vrstva J2EE architektury.

Firewall

Firewall je síťové zařízení nebo program, který slouží k řízení a zabezpečování síťového provozu mezi sítěmi s různou úrovní důvěryhodnosti a zabezpečení.

GUI

Grafické uživatelské rozhraní (Graphical User Interface) je rozhraní (v dnešní době nejčastěji okna) v grafickém režimu.

HTTP

HTTP (Hyper Text Transfer Protocol) je internetový protokol určený původně pro výměnu hypertextových dokumentů ve formátu HTML.

Implementace

Implementace je praktické použití metody či algoritmů (postupů) za účelem dosažení vytčeného cíle.

Instance

Z hlediska OOP je instance objekt, vytvořený podle konkrétní třídy.

J2ME (Java 2 Micro Edition)

Umožňuje vývoj aplikací pro bezdrátová zařízení. Obsahuje konfigurace (CLDC) a profily (MIDP).

JDBC (The Java Database Connectivity)

Tato technologie umožňuje vytvoření konektivity k relačním databázím nebo obecně k sloupcově orientovaným datům.

JDK (Java SE Development Kit)

Nástroje pro vývoj v Javě. Obsahuje JRE a vývojové nástroje pro příkazový řádek, užitečné pro vývoj appletů a aplikací.

JNDI

The Java Naming and Directory Interface je rozhraní sloužící k vyhledání služeb na základě jejich jmen. Pomocí tohoto rozhraní jsou vyhledávány EJB vystavené na aplikačním serveru Jboss.

JRE (Java Runtime Environment)

Obsahuje JVM, běhové třídy a knihovny. Slouží ke spouštění aplikací v Javě, ale neobsahuje nástroje pro vývoj.

JSR-172 – WSA (Web Services APIs)

Jedná se o balík rozšiřující J2ME, který umožňuje J2ME zařízením stát se klientem webových služeb.

JVM

Virtuální stroj Javy (JVM) se stará o interpretaci bajtového kódu pro prostředí (platforma, architektura), ve kterém je spuštěn.

Klient-server architektura

Jde o síťovou architekturu, kde server poskytuje služby klientům, kteří o ně žádají. Komunikaci začíná vždy klient. Pošle požadavek serveru, který požadavek vyhodnotí a odpověď odešle zpět klientovi.

Kompilace

Je všeobecně stroj, respektive program, provádějící překlad (zobrazení) ze vstupního jazyka do jazyka výstupního.

MIDP (Mobile Information Device Profile)

MIDP operuje nad vrstvou CLDC a zahrnuje v sobě další balíky pro práci s mobilním zařízením.

Midlet

Aplikace pro mobilní zařízení na bázi MIDP.

Open-source

Open source nebo také open-source software (OSS) je počítačový software s otevřeným zdrojovým kódem. Otevřenost zde znamená jak technickou dostupnost kódu, tak legální dostupnost - licenci software, která umožňuje uživatelům zdrojový kód využívat, například prohlížet a upravovat.

ORM (Object-Relational Mapping)

Jde o techniku pro mapování objektů a relací databáze na třídy a reference programovacího jazyka.

POJO (Plain Old Java Object)

Jednoduché Javové objekty, které nerozšiřují, neimplementují a neobsahují anotace.

RPC (Remote Procedure Call)

Vzdálené volání procedur pomocí vlastního protokolu.

Simplexní komunikace

Jako simplexní komunikace (simplex) se označuje jednosměrná komunikace, tj. spojení, při kterém jedna strana vysílá a druhá strana přijímá, přičemž se tyto role nemění.

Skeleton

Je objekt na straně serveru sloužící ke komunikaci, který přijímá volání po síti a předává je skutečnému objektu, který implementuje rozhraní pro jeho zpracování.

SOAP (Simple Object Access Protocol)

Jedná se o protokol sloužící pro přenos objektů, který nahradil XML-RPC.

Strojový kód

Strojový kód je posloupnost instrukcí procesoru vyjádřená čísly.

Stub

Je objekt na straně klienta, který slouží ke komunikaci se skeletonem.

Superuživatel

Superuživatel je uživatel s výhradními právy.

UDDI (Universal Description Discovery and Integration)

Veřejně přístupná databáze pro uložení informací o webových službách.

WAN

Rozlehlé sítě umožňují komunikaci na velké vzdálenosti. Bývají obvykle veřejné, ale existují i soukromé WAN sítě.

WSDL (Web Service Definition Language)

Jazyk pro popis webových služeb ve formátu XML.

WSIL (Web Service Inspection Langueage)

Jazyk pro tvorbu seznamu webových služeb ve formátu XML.

WTK (Java Wireless Toolkit)

Nástroje pro vývoj aplikací pro bezdrátové zařízení.

XML

Rozšířitelný značkovací jazyk, jehož výstupem je textový dokument.

XML-RPC

RPC, jehož volání jsou převedena do XML dokumentu a přenesena pomocí protokolu HTTP.

Literatura

[1] Brůha, L.: *Java Hotová řešení*. 1. vyd. Brno, Computer Press 2003. Počet stran 328. ISBN 80-251-0072-3

[2] Sun Microsystems, Inc.: *The Java Language: An Overview* /online/. Dostupné na http://java.sun.com/docs/overviews/java/java-overview-1.html

[3] Kučerová, H.: *Projektování informačních systémů II 2005/2006* /online/. Dostupné na http://web.sks.cz/users/ku/PRI/tridy.htm

[4] Monica Pawlan: *Java 2 Enterprise Edition Technology Center* /online/. Dostupné na http://java.sun.com/developer/technicalArticles/J2EE/Intro/index.html

[5] Sams Publishing: *Introduction to EJBs* /online/. Dostupné na http://www.developer.com/java/ejb/print.php/1434371

[6] Sun Microsystems, Inc.: *Java ME Technology* /online/. Dostupné na http://java.sun.com/javame/technology/index.jsp

[7] W3C Working Group: *Web Services Architecture* /online/. Dostupné na http://www.w3.org/TR/ws-arch/

Příloha I – Zdrojové kódy

Výpis 1 – Zdrojový kód webové služby

```
package cz.wirgeek.loan.beans;

import java.util.Collection;
import java.util.Vector;

import javax.ejb.EJB;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

import cz.wirgeek.loan.ProductResult;
import cz.wirgeek.loan.ProductsMultiMap;
import cz.wirgeek.loan.ScheduleOfPayment;

/**
 * @author Petr Žáčík
 *
 */
@Stateless
@WebService(name = "Loan", targetNamespace =
"http://127.0.0.1:8080/loanws/LoanWS", serviceName = "LoanWS")
@SOAPBinding(style = SOAPBinding.Style.DOCUMENT, use =
SOAPBinding.Use.LITERAL, parameterStyle =
SOAPBinding.ParameterStyle.WRAPPED)
public class LoanWS {

      @EJB
      private StatelessBeanLocal bean;

      /**
       * Create sample test in database.
       */
      @WebMethod
      public void createSampleData() {
            bean.createSampleData();
      }

      /**
       * Search for best products - simple searching.
       *
       * @param count
       * @param loan
       * @param months
       * @param precision
       * @return Set of best products.
       */
      @WebMethod
      public Vector<ProductResult> getBestSet(int count, long loan,
```

```
int months,
                  double precision) {
            ProductsMultiMap products;
            if (months > 0)
                  products = bean.getBestSet(count, loan, months,
precision);

            else
                  products = bean.getBestSet(count, loan,
precision);

            return products.toProductResult();
      }

      /**
       * @param count
       * @param loan
       * @param months
       * @param id_product
       * @param precision
       * @return Best product options.
       */
      @WebMethod
      public Vector<ProductResult> getBestProductSet(int count, long
loan,
                  int months, int id_product, double precision) {
            ProductsMultiMap products;
            if (months > 0)
                  products = bean.getBestProductSet(count, loan,
months, id_product,
                              precision);
            else
                  products = bean.getBestProductSet(count, loan,
id_product,
                              precision);
            return products.toProductResult();
      }

      /**
       * @param data
       *            Loan data
       * @return rpsn
       */
      @WebMethod
      public String countRPSN(ScheduleOfPayment schedule, double
precision) {
            return bean.getRPSNStr(schedule, precision);
      }

      @WebMethod
      public Vector<String> getBankList() {
            Vector<String> v = new Vector<String>();
            Collection<BankEntity> banks = bean.getBanks();
            for (BankEntity bankEntity : banks) {
```

```
                  v.add(bankEntity.getName());
            }
            return v;
      }

      /**
       * List of products of specific bank.
       *
       * @param bankName
       * @return data serialized in vector, even String is
id_product, odd String
       *         is product name.
       */
      @WebMethod
      public Vector<String> getProductList(String bankName) {
            Collection<ProductEntity> products = bean.getProducts();

            Vector<String> v = new Vector<String>();
            for (ProductEntity productEntity : products) {
                  if
(productEntity.getBank().getName().equals(bankName)) {

      v.add(String.valueOf(productEntity.getId_product()));
                        v.add(productEntity.getName());
                  }
            }
            return v;
      }
}
```

Výpis 2 – Hlavní část midletu

```
package cz.wirgeek.loan.client.midlet;

import java.io.IOException;

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Choice;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.Screen;
import javax.microedition.lcdui.StringItem;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
```

```
import cz.wirgeek.loan.client.midlet.stub.ProductResult;
import cz.wirgeek.loan.client.midlet.stub.ScheduleOfPayment;
import cz.wirgeek.loan.client.midlet.threads.CountRPSNThread;
import
cz.wirgeek.loan.client.midlet.threads.GetBestProductSetThread;
import cz.wirgeek.loan.client.midlet.threads.GetBestSetThread;
import cz.wirgeek.loan.client.midlet.threads.UpdateBankListThread;
import
cz.wirgeek.loan.client.midlet.threads.UpdateProductListThread;

/**
 * @author Petr Žáčík
 *
 * Client midlet for Loan web service.
 *
 */
public class LoanMidlet extends MIDlet implements CommandListener,
            ResultProcess {

      /**
       * Precision of counting RPSN.
       */
      public static final double PRECISION = 0.000001;

      /**
       * Maximum results on screen in case there no limit specified.
       */
      public static final int MAX_RESULTS = 10;

      private Display display;

      private Image logo;

      private String[] screensNames = { "Obecné hledání",
                  "Vybrat Konkretní službu", "RPSN Kalkulátor" };
      private Screen[] screens;

      private List l_mainMenu;
      private List l_bankList;
      private List l_productList;
      private String[] productList;

      private Form f_specific;
      private WaitForm f_waitForm;

      private Alert splash;
      private Alert a_connect;

      private static final Command cmd_exit = new Command("Konec",
Command.EXIT,
                  2);
      private static final Command cmd_back = new Command("Zpět",
Command.BACK, 1);
```

```
	private static final Command cmd_search = new
Command("Hledej",
			Command.SCREEN, 1);
	private static final Command cmd_count = new
Command("Počítej",
			Command.SCREEN, 1);
	private static final Command cmd_select = new Command("Vyber",
			Command.SCREEN, 1);

	// General search
	private TextField tf_count = new TextField("nabídek:", "3", 1,
			TextField.NUMERIC);
	private TextField tf_loan = new TextField("půjčka:", "30000",
6,
			TextField.NUMERIC);
	private TextField tf_months = new TextField("měsíců:", "6", 2,
			TextField.NUMERIC);

	// RPSN calculator
	private TextField tf_loanC = new TextField("půjčka:", "30000",
6,
			TextField.NUMERIC);
	private TextField tf_feeC = new TextField("splátka:", "1500",
6,
			TextField.NUMERIC);
	private TextField tf_monthsC = new TextField("měsíců:", "24",
2,
			TextField.NUMERIC);
	private StringItem si_result = new StringItem("RPSN:", "0%");

	// Specific product
	private StringItem si_productName = new StringItem("null",
null);
	private TextField tf_loanS = new TextField("půjčka:", "30000",
6,
			TextField.NUMERIC);
	private TextField tf_monthsS = new TextField("měsíců:", "12",
2,
			TextField.NUMERIC);

	/**
	 * Default constructor.
	 */
	public LoanMidlet() {

		loadImages();
		createAlert();
		createScreens();
		createMainMenu();
		prepareProductList();
		prepareSpecificForm();
		createSplash();
	}
```

```
	private void loadImages() {
		try {
			logo = Image.createImage("//money.png");
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	private void createSplash() {
		splash = new Alert("Vyhledávač půjček", "Autor: Petr
Žáčík", logo, null);
		splash.setTimeout(4000);
	}

	private void createAlert() {
		a_connect = new Alert("Chyba připojení.", "Služba není
dostupná.",
				null, AlertType.ERROR);
		a_connect.setTimeout(5000);
	}

	private void createScreens() {
		screens = new Screen[screensNames.length];
		screens[0] = createGeneralForm(0);
		screens[1] = prepareBankList();
		screens[2] = createCountForm(2);
	}

	private void createMainMenu() {
		l_mainMenu = new List("Hlavní nabídka",
Choice.IMPLICIT);
		for (int i = 0; i < screensNames.length; i++) {
			l_mainMenu.append(screensNames[i], null);
		}

		l_mainMenu.addCommand(cmd_exit);
		l_mainMenu.addCommand(cmd_select);
		l_mainMenu.setCommandListener(this);
		// menu.setTicker(ticker);
	}

	private List prepareBankList() {
		l_bankList = new List("Vyberte sprostředkovatele",
Choice.IMPLICIT);

		l_bankList.addCommand(cmd_back);
		l_bankList.addCommand(cmd_select);
		l_bankList.setCommandListener(this);
		return l_bankList;
	}

	private Form prepareSpecificForm() {
		f_specific = new Form("Konkrétní produkt");
		f_specific.append(si_productName);
```

```
            f_specific.append(tf_loanS);
            f_specific.append(tf_monthsS);

            f_specific.addCommand(cmd_back);
            f_specific.addCommand(cmd_count);
            f_specific.setCommandListener(this);
            return f_specific;
      }

      private List prepareProductList() {
            l_productList = new List("Vyberte produkt",
Choice.IMPLICIT);
            l_productList.addCommand(cmd_back);
            l_productList.addCommand(cmd_select);
            l_productList.setCommandListener(this);
            return l_productList;
      }

      private Form createCountForm(int nameIndex) {
            Form f = new Form(screensNames[nameIndex]);
            f.append(tf_loanC);
            f.append(tf_feeC);
            f.append(tf_monthsC);
            f.append(si_result);
            Font font = Font.getFont(Font.FACE_SYSTEM,
Font.STYLE_PLAIN,
                        Font.SIZE_LARGE);
            si_result.setFont(font);

            f.addCommand(cmd_back);
            f.addCommand(cmd_count);
            f.setCommandListener(this);
            return f;
      }

      private Form createGeneralForm(int nameIndex) {
            Form f = new Form(screensNames[nameIndex]);
            f.append(tf_loan);
            f.append(tf_months);
            f.append(tf_count);
            f
                        .append(new StringItem(
                                    "Tip:",
                                    "Pokud necháte položku měsíce
nevyplněnou, bude se hledat nejlepší produkt obecněji."));

            f.addCommand(cmd_back);
            f.addCommand(cmd_search);
            f.setCommandListener(this);

            return f;
      }

      protected void destroyApp(boolean arg0) {
```

```
            notifyDestroyed();
      }

      protected void pauseApp() {
      }

      protected void startApp() throws MIDletStateChangeException {
            display = Display.getDisplay(this);
            display.setCurrent(splash, l_mainMenu);
            f_waitForm = new WaitForm(display);

            // If connection refused by user, need to go back.
            f_waitForm.addCommand(cmd_back);
            f_waitForm.setCommandListener(this);
      }

      public void commandAction(Command c, Displayable d) {

            if (c == cmd_exit) {
                  destroyApp(true);

            } else if (c == cmd_back) {
                  if (d == f_waitForm) {
                        display.setCurrent(l_mainMenu);
                  } else if (d ==
screens[l_mainMenu.getSelectedIndex()]) {
                        display.setCurrent(l_mainMenu);
                  } else if (d == f_specific) {
                        display.setCurrent(l_productList);
                  } else

      display.setCurrent(screens[l_mainMenu.getSelectedIndex()]);

            } else if (c == cmd_search) {

                  if (l_mainMenu.getSelectedIndex() == 0) {
                        // getBestSet form
                        int count =
Integer.parseInt(tf_count.getString());
                        long loan =
Long.parseLong(tf_loan.getString());

                        if (tf_months.getString().length() == 0)
                              f_waitForm.runThread(new
GetBestSetThread(this, count,
                                          loan, 0, PRECISION));
                        else {
                              int months =
Integer.parseInt(tf_months.getString());
                              f_waitForm.runThread(new
GetBestSetThread(this, count,
                                          loan, months, PRECISION));
                        }
                  }
```

```
            } else if (c == cmd_count) {
                  if (d == f_specific) {
                        long loan =
Long.parseLong(tf_loanS.getString());

                        int productIndex =
Integer.parseInt(productList[l_productList
                                    .getSelectedIndex() * 2]);

                        if (tf_monthsS.getString().length() == 0)
                              f_waitForm.runThread(new
GetBestProductSetThread(this,
                                          MAX_RESULTS, loan, 0,
productIndex, PRECISION));
                        else {
                              int months =
Integer.parseInt(tf_monthsS.getString());
                              f_waitForm
                                          .runThread(new
GetBestProductSetThread(this,
                                                      MAX_RESULTS,
loan, months, productIndex,
                                                      PRECISION));
                        }

                  } else {
                        // countRPSN form
                        long loan =
Long.parseLong(tf_loanC.getString());
                        long fee =
Long.parseLong(tf_feeC.getString());
                        int months =
Integer.parseInt(tf_monthsC.getString());

                        ScheduleOfPayment schedule = new
ScheduleOfPayment();
                        schedule.setDir(365.25);
                        schedule.setLoan(loan);
                        schedule.setMonths(months);
                        schedule.setFee(fee);

                        f_waitForm.runThread(new
CountRPSNThread(this, schedule,
                                    PRECISION));
                  }
            } else if (c == cmd_select) {
                  if (d == l_productList) {

      si_productName.setLabel(l_productList.getString(l_productList
                                    .getSelectedIndex()));
                        display.setCurrentItem(tf_monthsS);
                        display.setCurrentItem(tf_loanS);
                        display.setCurrent(f_specific);
```

```
                } else if (d == l_bankList) {
                      f_waitForm.runThread(new
UpdateProductListThread(this,

      l_bankList.getString(l_bankList.getSelectedIndex())));
                } else if (l_mainMenu.getSelectedIndex() == 1) {
                      f_waitForm.runThread(new
UpdateBankListThread(this));
                } else {

      display.setCurrent(screens[l_mainMenu.getSelectedIndex()]);
                }

                // } else if (d == l_mainMenu) {
                //
display.setCurrent(screens[l_mainMenu.getSelectedIndex()]);
                // if (l_mainMenu.getSelectedIndex() == 1)
                // f_waitForm.runThread(new
UpdateBankListThread(this));
           }

      }

      /**
       * Result from web service is now completed.
       */
      // TODO @see
      synchronized public void resultCompleted(Object sender, Object
result) {
            if ((sender instanceof GetBestSetThread)
                      || (sender instanceof
GetBestProductSetThread)) {
                Form f = new ProductsResultForm((ProductResult[])
result);
                f.setCommandListener(this);
                f.addCommand(cmd_back);
                display.setCurrent(f);

            } else if (sender instanceof UpdateBankListThread) {
                String[] items = (String[]) result;
                l_bankList.deleteAll();
                for (int i = 0; i < items.length; i++) {
                      l_bankList.append(items[i],
 null);
                }
                display.setCurrent(l_bankList);

            } else if (sender instanceof UpdateProductListThread) {
                productList = (String[]) result;
                l_productList.deleteAll();
                for (int i = 1; i < productList.length; i += 2) {
                      l_productList.append(productList[i], null);
                }
                display.setCurrent(l_productList);
```

```
            } else if (sender instanceof CountRPSNThread) {
                  si_result.setText(result + "%");

      display.setCurrent(screens[l_mainMenu.getSelectedIndex()]);

            } else {
                  display.setCurrent(a_connect, l_mainMenu);
            }
      }
}
```

Příloha II – Obsah CD

Diplomová práce

Diplomová práce je umístěna v adresáři *diplomova_prace* ve formátu pro OpenOfficeWriter (odt) a Portable Document Format (pfd).

Software

- služba je uložena v adresáři *software/LoanService*
- aplikace pro mobil se nalézá v adresáři *software/LoanMidlet*
- balíček *jar* určený pro implementaci na mobilní zařízení je umístěn v adresáři *software/LoanMidlet/deployed*
- balíček *jar* určený pro implementaci webové služby se nachází v adresáři *software/LoanService/server/dist*
- zdrojové kódy jsou v adresářích *src*