MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Systém pro vyhledávání dopravního spojení v rámci IDS JMK

BAKALÁŘSKÁ PRÁCE

Jiří Kremser

Brno, 2008

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Tomáš Pitner, Ph.D.

Poděkování

Chtěl bych poděkovat svému vedoucímu práce RNDr. Tomáši Pitnerovi, Ph.D. za jeho vedení a Vladimíru Dopitovi za poskytnutí dat a informací o nich. Rád bych také poděkoval kamarádům, kteří mi poskytli svůj mobilní telefon na otestování aplikace.

Shrnutí

Potřeba člověka být včas na správném místě je v dnešní uspěchané době zřejmá. S využitím mobilních zařízení si může každý nosit jízdní řády s sebou. Cílem této práce je popsat platformu Java ME a implementovat aplikaci, která pomáhá uživateli zorientovat se v příliš komplexních službách IDS JMK. Aplikace slouží jako vyhledávač spojů v dopravní síti.

Klíčová slova

vyhledávač, IDS JMK, mobilní zařízení, Java ME, MIDP 2.0, CLDC 1.1

Obsah

Kapitola 1

Úvod

V několika posledních desetiletích zažila mobilní zařízení obrovský vývoj. Telefony dnes již neslouží pouze ke komunikaci. Společně se zařízeními PDA otevřely vývojářům nové pole působnosti. S nástupem Javy na tuto platformu tak mohou vznikat užitečné aplikace usnadňující běžný život. Tato bakalářská práce popíše vývoj právě jedné takové mobilní aplikace.

1.1 Vyhledávač spojů

Zadáním je vytvořit aplikaci pro vyhledávání dopravních spojů. Všechna řešení, která jsou v současné době k dispozici pro vyhledávání spojů, buď vyžadují online připojení, nebo jsou určeny jen pro Brno. Moje aplikace si klade za cíl být offline variantou, která bude vyhledávat spoje i pro okolí Brna.

Offline varianta má tu výhodu, že její používání je zcela zdarma. Nevýhodou je horší aktualizovatelnost, protože data nejsou centralizovaná. IDS JMK naštěstí jízdní řády nemění příliš často, navíc aplikace bude obsahovat už nové jízdní řády pro rok 2008.

Pro snadnější aktualizaci dat je společně s aplikací vyhledávače vyvinuta aplikace konvertor, která sestaví aplikaci vyhledávače vždy z aktuálních dat ze systému IDS JMK.

1.2 Struktura této práce

Tato bakalářská práce se nesnaží být kompletní příručkou pro platformu Java ME. Zaměřuje se na témata, která mají co do činění s řešeným problémem. V druhé kapitole jsou stručně popsány architektura mikro edice Javy, distribuce aplikací a nástroje usnadňující vývoj. Třetí kapitola má za cíl probrat možnosti Javy ME pro ukládání většího objemu dat, což je v případě aplikace pro vyhledávání spojů potřeba. Následuje kapitola o vyhledávání v grafech. Pátá kapitola se zabývá analýzou řešeného problému a popisuje, jak je možné využít technologie popsané v předchozích kapitolách. Zbylé kapitoly popisují samotné aplikace.

Kapitola 2

Java ME

2.1 Historie

Podnětem ke vzniku programovacího jazyka, byl vývoj aplikace pro přenosný ovladač domácích spotřebičů. Projekt se jmenoval Stealth Project (později Green Project) a členem jeho týmu byl i zakladatel jazyka Oak James Gosling. Potřeba nového jazyka pramenila ze zkušeností s jazykem C++, který byl velmi náchylný k programátorským chybám. Oak byl navržen tak, aby co nejvíce chyb zachytil už při kompilaci a nedovoloval programátorovi pracovat s ukazateli a správou paměti [1].

Zařízení pro ovládání domácích spotřebičů se pro malou poptávku nezačalo nikdy sériově vyrábět. Společnost Sun Microsystems, jejíž byl Gosling členem, přejmenovala v roce 1995 kvůli kolizi jmen s jiným produktem jazyk Oak na Java. Společnost Netscape Javu integrovala do svých prohlížečů a světlo světa spatřily první aplety.

Java se stávala stále oblíbenější, protože se řídila podle filozofie WORA (Write once, run anywhere). Multiplatformnost byla jednou z hlavních priorit. Platforma Javy se s přidáváním dalších knihoven stále zvětšovala, proto se od verze 2 rozštěpila na platformy J2ME (Java 2 Micro Edition), J2SE (Java 2 Standard Edition) a J2EE (Java 2 Enterprise Edition).

J2ME platforma je velmi rozmanitá, proto obsahuje sadu specifikací, které popisují vždy jen část platformy. Konfigurace tak vymezují minimální množinu vlastností pro danou třídu zařízení a je rozšiřována jedním nebo více profily.

S vydáním Javy verze 5 byly platformy přejmenovány na Java ME, Java SE a Java EE.

Před nedávnem vydala společnost Google své řešení mobilní platformy s názvem Android. Zdrojové kódy pro platformu Android se píší také v jazyce Java. Android však nevyužívá k interpretaci kódu referenční implementaci Virtual Machine od společnosti Sun, nýbrž svou vlastní implementaci Dalvik VM [11].

2.2 Konfigurace

Konfigurace je minimálním společným jmenovatelem pro určitou třídu zařízení. Obsahuje VM (Virtual Machine) a základní Java ME API.

CLDC Connected Limited Device Configuration je nejmenší definovaná konfigurace zaměřená na zařízení s malým množstvím paměti. Minimální hodnoty jsou uvedeny níže.

- 128 kB paměti pro uložení VM a tříd, jež tvoří platformu CLDC.
- 32 kB operační paměti.

Třídy, které jsou součástí konfigurace, se nacházejí v balíčcích [8].

- javax.microedirion.io
- java.io
- java.util
- java.lang

Balíčky nejsou plnohodnotné varianty Java SE, ale značně ochuzené verze.

Specifikace JSR–139 (CLDC 1.1) dává výrobcům velkou volnost v její implementaci. Zařízení nemusí mít displej, dokonce ani klávesnici. Referenční implementace CLDC od společnosti Sun obsahuje Virtual Machine s názvem KVM. Sun také nabízí implementaci s HotSpot překladačem.

CDC Connected Device Configuration je konfigurace pro výkonnější zařízení s většími prostředky. Obvykle pro PDA, chytré telefony a jiné. Vyplňuje tak místo mezi CLDC a Java SE.

Minimální požadavky na zařízení.

- 256 kB paměti ROM pro platformu.
- 512 kB operační paměti.

Referenční implementace VM se nazývá CVM[1]. Od verze 1.1.1 obsahuje HotSpot překladač. Přibyli plnohodnotná podpora pro reflexi, práci se sítí, kolekce a jiné. Součástí konfigurace není například Swing, který však lze získat použitím profilu Personal profile 1.1 a volitelného balíčku AGUI (JSR–209). Současná verze je CDC 1.1.2 odpovídající JSR–218.

2.3 Profily

Rozšiřují vlastnosti konfigurace o nové API.

- **CDC**
  - Foundation Profile

1. Zkratky KVM a CVM nemají žádný význam. Dříve znamenali „Kilobyte“ a „Compact“, ale od jejich používání se upustilo.

  - Personal Basis Profile
  - Personal Profile

- **CLDC**
  - Personal Digital Assistant Profile
  - Information Module Profile
  - Mobile Information Device Profile – **MIDP**

Profil MIDP je určen pro mobilní telefony a obsahuje funkcionalitu pro tvorbu uživatelského rozhraní, her, práci se sítí a funkce trvalého ukládání záznamů. Aplikace jsou nazývány *midlety* podle třídy `MIDlet`. Současná verze specifikace je 2.1, většina stávajících zařízení však podporuje verzi 2.0 [10].

2.4 Volitelné balíčky

Rozšiřují platformu konfigurace s profilem o API pro specifické účely. Nejsou součástí profilu, protože vyžadují přístup ke specifickým funkcím zařízení. Přehled některých volitelných balíčků je na obrázku 2.1, současně je zde znázorněno, které balíčky musí být přidány k dané konfiguraci, aby mohlo být výsledné spojení nazváno platformou MSA (Mobile Service Architecture) resp. JTWI (Java Technology for the Wireless Industry). Výrobci je nejsou povinni implementovat do svých zařízení.

2.5 Midlet

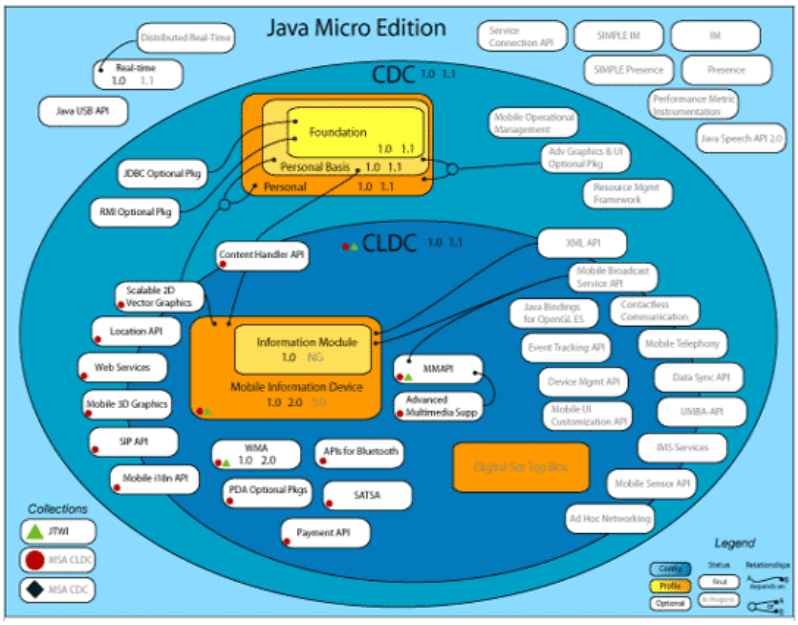
Midlet musí dědit od abstraktní třídy `javax.microedition.midlet.MIDlet` a implementovat její metody pro řízení životního cyklu midletu. Cyklus řídí AMS (application management software). Midlet tak zprostředkovává rozhraní mezi aplikací a runtime prostředím telefonu.

Midlet může být pozastaven například v případě, dojde-li k uskutečnění telefonního hovoru za běhu aplikace, a následně zase obnoven. Volání těchto metod je v kompetenci AMS a je znázorněno na obrázku 2.2.

Midlet musí být umístěn v `jar` archivu aplikace a popsán v `jad` deskriptoru. Obsahuje-li archiv `jar` více midletů, jedná se o tzv. sadu midletů.

2.6 Jar, Jad, distribuce

Všechny midlety, jež mají být součástí sady midletů, musí být společně s dalšími soubory, které aplikace využívá, umístěny v jediném `jar` archivu. Strukturu tohoto archivu popisuje soubor `manifest.mf`, který je zabalený rovněž uvnitř archivu. V manifestu jsou popsány



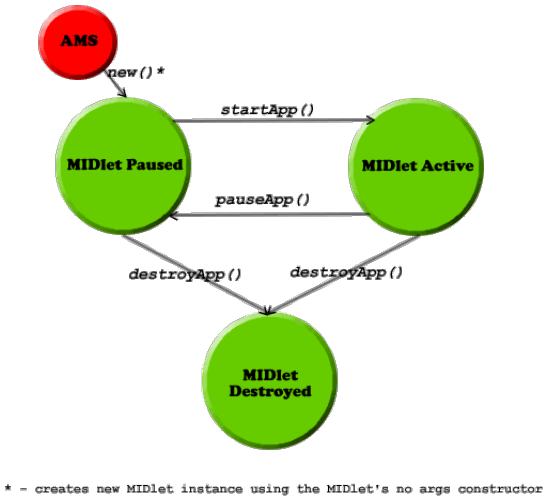
Obrázek 2.1: Schéma Java ME

cesty ke „spustitelným“ třídám. Ve standardní edici jsou popsány cesty k třídám, jež obsahují statickou metodu main. V mikro edici jsou to cesty k jednotlivým midletům. Součástmi manifestu jsou dodatečné meta informace, které popisují aplikaci a mohou být společné i pro `jad` deskriptor. Jedná se o ikony jednotlivých midletů, název sady midletů, verze sady midletů, konfigurace, profil a další.

Soubor `jad` (Java Application Descriptor) je umístěn vně archivu a byl navržen pro síťové prostředí, kde si uživatel nejprve stáhne malé množství informací, které popisuje aplikaci, a teprve potom vlastní aplikaci. Deskriptor tedy musí obsahovat URL `jar` archivu, jeho velikost, verzi a další. Vhodné je také přidat popis, který se uživateli zobrazí ještě před instalací sady midletů. Deskriptor také může obsahovat volitelné atributy `MIDlet-Delete-Notify` a `MIDlet-Install-Notify`, které zajistí poslání HTTP požadavku na příslušný servlet [7].

Distribuce probíhá tak, že oba soubory se umístí na internet a uživatel prostřednictvím protokolu WAP nebo GPRS aplikaci stáhne. Klient může obsahovat také DA (discovery application), která slouží jako portál k jednotlivým midletům. Minimální konfigurace serverové části zahrnuje nastavení MIME typů souborů.

Typ souboru	MIME typ
JAR	`application/java-archive`
JAD	`text/vnd.sun.j2me.app-descriptor`



Obrázek 2.2: Životní cyklus midletu

2.7 Podpora

Podpora pro vývoj mobilních aplikací je poměrně na vysoké úrovni. Vývojová prostředí poskytují jednoduchý způsob, jak aplikaci navrhnout. Například podpora v Netbeans s nainstalovanou podporou pro mobilní aplikace zahrnuje vizuální návrhář pro tok aplikace i samostatné formuláře. Pro Eclipse je vyvíjen zásuvný modul s názvem EclipseME. Obě tyto IDE podporují Sun Java Wireless Toolkit.

Podpora ze strany výrobců mobilních zařízení spočívá ve vytváření nástrojů pro usnadnění vývoje aplikací určených právě pro jejich značku. Tyto sady nástrojů obvykle obsahují emulátor dané značky a lze je integrovat s oběma zmíněnými IDE.

2.7.1 Sun Java Wireless Toolkit

Sada nástrojů pro vývoj mobilních aplikací dodávaná společností Sun se nazývá Sun Java Wireless Toolkit. Součástí sady jsou emulátory, které zapouzdřují referenční implementace obou konfigurací a některých profilů, a software pro správu aplikací (AMS). Obsahuje také jednoduché vývojové prostředí s debuggerem a různé užitečné programy. Například nástroj pro správu certifikátů s veřejnými klíči, profiler, monitor paměti a další [1]. Midlety totiž mohou být podepisovány.

Kapitola 3

Práce s daty v Java ME

Java ME je navržena zejména pro hry a aplikace využívající síť. Tvorbou her se tato práce zabývat nebude. Není-li aplikace připojena k síti, je potřeba dodat data jiným způsobem. Obecně lze uchovávání dat rozdělit, ve smyslu následné měnitelnosti dat, na uchovávání ve „statických“ a „dynamických“ úložištích.

Statické úložiště poskytuje data v průběhu času stále konstantní. Stačí pouze mechanizmus na jejich čtení. Není proto třeba řešit možné inkonzistence způsobené operacemi nad daty. Cenou za tuto vlastnost je ovšem jejich statičnost a většinou i potřeba jejich integrace do systému. Jejich aktualizace není snadná a závisí hodně na implementačních detailech úložiště.

Dynamické úložiště umožňuje všechny CRUD operace. Tento model pokrývá větší problémovou doménu než předchozí, ale nese se sebou také nutnost ošetřit zpracovávání transakcí v multiuživatelském prostředí. Jeho nejčastějšími zástupci jsou různé relační a objektové databáze.

Java ME poskytuje jen několik způsobů, jak je možné data ukládat. Žádá-li si aplikace pracovat pouze se statickým úložištěm, jsou k dispozici resources, ukládání dat do konstant a descriptoru. Zástupci dynamického úložiště jsou RMS nebo JDBC. JDBC API je součástí specifikace JSR–169 (volitelný balíček) a je určeno pro konfiguraci CDC, proto nelze použít. Jednotlivé způsoby budou v kapitole dále rozebrány.

3.1 RMS

Systém pro ukládání dat (Record Management System) je určený pro trvalé uchovávání a manipulaci s daty. Je součástí specifikace MIDP 2.0 a poskytuje programátorovi API pro práci s daty. Přístupovým bodem k datům je „úložiště záznamů“ implementované třídou `RecordStore`.

V MIDP 2.0 je přidána vlastnost sdílení úložišť záznamů mezi různými sadami midletů. Úložiště je totiž jednoznačně identifikováno názvem úložiště (maximálně 32 znaků) a názvem sady midletů. V praxi to znamená, že může jedna aplikace sloužit jako „instalátor“ pro jinou aplikaci. Instalátor ve smyslu aplikace, která naplní úložiště daty. Tento přístup je ale nešťastný hned v několika bodech.

- První aplikace musí data odněkud také čerpat.
- Změna dat může vyžadovat změnu zdrojového kódu „instalační“ aplikace.

- Distribuce je problémová. Buď je „instalační“ midlet součástí stejné sady midletů. Potom bude navždy zobrazován v menu sady midletů. Nebo není součástí stejné sady midletů a uživatel je nucen „zbytečný“ midlet stáhnout, spustit a smazat.

RMS umožňuje programátorovi vytvářet, rušit, otvírat a zavírat úložiště. Úložiště může obsahovat množinu záznamů. Záznamem může být jakýkoli proud bytů. RMS tedy nehlídá typ ukládaných dat. Vzhledem k tomu, že CLDC ani MIDP nepodporují serializaci objektů, je potřeba vlastní objekty serializovat ručně pomocí `DataOutputStream` a `DataInputStream`, které umí serializovat primitivní datové typy a řetězce.

Každému novému záznamu je přidělen identifikátor. Autoinkrementaci identifikátorů RMS zajišťuje, kvůli výkonnosti však nehlídá tvoření „děr“ při odebírání záznamů. Nad záznamy lze iterovat prostřednictvím třídy `RecordEnumeration`, definovat na nich uspořádání díky třídě `RecordComparator` nebo iterovat jen některými (`RecordFilter`) [2].

3.2 Konstanty a deskriptor

Dalším způsobem, jak lze aplikaci naplnit aplikaci daty, je uložit je přímo do souborů tříd jako statické atributy. Modifikátor `final` zajistí jejich neměnitelnost. Tato metoda dovoluje ukládat i binární data, když se jako typ atributu zvolí pole bytů.

Toto řešení zkomplikuje čitelnost kódu a zvětší velikost souboru třídy. Zpomalí také zavádění třídy do paměti, ale protože jsou atributy označené modifikátorem `static`, není toto zpomalení dramatické. Aktualizovat automaticky takováto data je velmi obtížné.

Textová data mohou být ukládána do deskriptoru aplikace. Tato data lze získat metodou `getAppProperty(String key)` třídy `MIDlet`. Žádná metoda `setAppProperty()` není na třídě `MIDlet` definována, proto lze data jen číst. Binární data lze kódovat metodou Base64, ale jelikož deskriptor není součástí archivu, není komprimován a metoda Base64 zbytečně velikost deskriptoru zvětší. Deskriptor může mít na různých zařízeních různou maximální velikost.

3.3 Resources

Jakékoli soubory uvnitř archivu `jar` se nazývají resources. Může se jednat o jednoduché knihovny, multimediální soubory nebo binární data. V aplikaci k jejich načtení slouží metoda `getResourceAsStream(String path)`. Jedná se o součást Java Reflection API, jež sice celé není součástí CLDC 1.1, ale některé metody byly ponechány. Přístupovým bodem k metodě je tedy instance typu `Class`.

Jelikož jsou soubory součástí archivu, není snadná jejich modifikace. Ukládání dat do souborů je z popsaných způsobů nejvhodnější, proto tuto metodu používám ve své práci. Více je popsána možnost aktualizace takovýchto dat v kapitole 6.

Kapitola 4

Vyhledávací algoritmy

Vyhledávání v grafu je podoborem teorie grafů a lze jej aplikovat na mnoho problémů z běžného života, s nimiž zdánlivě nesouvisí. Obecné vyhledávací algoritmy spojené s heuristikou z dané problémové domény ztratí na obecnosti, ale získají na efektivitě (složitost algoritmu klesne). V dnešní informační společnosti, kde správná informace dostupná včas má cenu zlata, je jejich využitelnost obrovská.

- V umělé inteligenci je vyhledávání jedním z klíčových prvků. Budoucnost patří sémantickým sítím a vyhledávacím agentům nad nimi. Datové struktury navrhované v minulých dekádách byly navrhovány pro co největší výkon, proto neobsahovaly žádná metadata a proces vyhledávání nerozlišoval informace podle kontextu výskytu. Dnes se tyto sémantické struktury (formáty) stávají běžnými.

- V managementu velkých firem je nutnost být efektivnější než konkurence. I tady mají vyhledávací a především optimalizační algoritmy své místo. Sestaví-li se co nejvěrnější model problému a určí pákové body, stačí v takovémto modelu naleznout nejlacinější cestu ze stavu A do stavu B. Bohužel sestavit věrný model problému je velice obtížné a v těchto oblastech i subjektivní.

Souvislost hledání spoje v dopravní síti s hledáním nejkratší cesty v grafu je zřejmá, proto budou stručně rozebrány dva nejznámější algoritmy pro nejkratší cestu grafem.

Níže jsou uvedeny algoritmy, které nebudou probrány, ale dokáží nejkratší cestu nalézt.

- *Algoritmus prohledávání grafu do šířky*

  Realizován pomocí fronty. Nezohledňuje ceny hran.

- *Algoritmus prohledávání grafu do hloubky*

  Realizován pomocí zásobníku. Nezohledňuje ceny hran.

- *Bellman–Fordův algoritmus*

  Realizován pomocí prioritní fronty. Dokáže pracovat se záporně ohodnocenými hranami.

4.1 Pojmy

Vysvětlení několika pojmů z teorie grafů. Není třeba zabývat se nesouvislými grafy, protože graf reprezentující dopravní síť IDS JMK je souvislý.

- *Graf*

  Graf je uspořádaná dvojice `(V,E)`, kde `V` je neprázdná množina vrcholů (uzlů grafu) a `E` je množina dvojic prvků z `V`. Jsou-li dvojice uspořádané, je graf orientovaný. Hrana vede mezi vrcholy `x` a `y` z množiny `V`, právě když množina `E` obsahuje takovouto dvojici `(x,y)`. Je-li graf orientovaný, vede šipka z vrcholu `x` do vrcholu, ale nikoliv z vrcholu `x` do vrcholu `y`.

- *Multigraf*

  Mezi dvěma vrcholy může vést libovolný počet hran.

- *Ohodnocení hran*

  Existuje funkce, která každou hranu ohodnotí nějakou informací, nejčastěji číslem. Tuto informaci budu dále v práci označovat střídavě pojmy cena a délka.

- *Cesta*

  Posloupnost, ve které se pravidelně střídají vrcholy a hrany a symbolizuje pomyslný průchod grafem po jednotlivých hranách vedoucích z vrcholu do vrcholu. Cena cesty je potom součet ohodnocení prošlých hran.

4.2 Floyd–Warshallův algoritmus

Floyd–Warshallův (nebo také Roy–Floydův) algoritmus nalezne nejkratší cesty mezi všemi uzly v orientovaném grafu. Využívá matici souslednosti, jejíž hodnoty jsou ceny cest. Matice je reprezentována dvourozměrným polem.

Pseudokód

```
 1  // Předpokládáme funkci cenaHrany(u, v) vracející cenu hrany z u do v
 2  // pokud hrana neexistuje, cenaHrany = nekonečno
 3  // cenaHrany(u, u) = 0
 4
 5  int cesta[][]                    // Dvourozměrné pole.
 6  foreach (u, v) in (V):           // Inicializace
 7     d[u][v] := cenaHrany(u, v)    // Prvky d[u][v] jsou inicializovány
 8                                   // funkcí cenaHrany(u, v)
 9  procedure FloydWarshall(d):
10     for k := 1 to |V|:
11        foreach (u, v) in (V):
12           cesta[u][v] := min(cesta[u][v], cesta[u][k] + cesta[k][v])
```

Asymptotická časová složitost algoritmu je stejná jako u Dijkstrova algoritmu, tzn. $\Theta(n^3)$. Algoritmus podobně jako prohledávání do šířky a do hloubky nezohledňuje hodnocení hran, ale řídí se pouze strukturou grafu. V každé iteraci se pro každý uzel grafu spočítá cena cesty do uzlu, který je o jednu hranu dále.

4.3 Dijkstrův algoritmus

Na rozdíl od Floyd–Warshallova algoritmu, počítá tento algoritmus nejkratší cesty v grafu pouze mezi počátečním uzlem `s` a všemi ostatními. Datové struktury, jež algoritmus využívá jsou pole `d`, kam se ukládají vzdálenosti z `s`, a prioritní fronta[1]. Funkce `extractMin(N)` aplikovaná na dosud nenavštívené vrcholy vyjme z fronty vrchol s nejkratší vzdáleností od `s`. Prioritní fronta proto musí být implementována s uspořádáním $\texttt{u} < \texttt{v} \iff \texttt{d[u]} < \texttt{d[v]}$. Pro každý vrchol `v` sousedící s vrcholem `u` vybraného z fronty se napočítá jeho vzdálenost, pokud je dosavadní vzdálenost z počátku větší než vzdálenost z počátečního uzlu `s` do uzlu `u` (`d[u]`) sečteno s cenou hrany mezi `u` a `v`. Pokud sousedící uzel `v` ještě nebyl navštíven, hodnota $\texttt{d[v]} = \infty$. Toho je zajištěno v inicializaci algoritmu.

Pseudokód

```
 1  procedure Dijkstra(V, E, s):
 2     for each vertex v in E:      // Inicializace
 3        d[v] := infinity          // Zatím neznámá vzdálenost z s do v
 4        d[s] := 0                 // Vzdálenost z s do s
 5        N := V                    // Všechny dosud nenavštívené vrcholy
 6
 7     while N is not empty:        // Samotný algoritmus
 8        u := extractMin(N)        // Vyjme nejlepší vrchol
 9        for each neighbor v of u:
10          alt := d[u] + cenaHrany(u, v)
11          if alt < d[v]
12             d[v] := alt
```

Algoritmus tedy postupně navštěvuje nejméně vzdálené (vzdálenost není počet prošlých hran) uzly z počátečního vrcholu a každá iterace představuje pomyslnou vlnu šířící se z počátečního uzlu.

1. Prioritní fronta může být navržena jako binární halda a výběr minimálního prvku má potom konstantní složitost.

Kapitola 5

Analýza

5.1 Specifikace

Systém bude zohledňovat tarifní zóny a informovat uživatele o dostupných jízdenkách. Aplikace bude implementována na platformě Java ME, bude splňovat specifikace MIDP 2.0 a CLDC 1.1 a bude otestována alespoň na příslušných emulátorech. Uživatel bude mít možnost zobrazit si jízdní řád jednotlivých linek na zastávce pro daný den.

5.2 Způsob ukládání dat

Po dohodě s Vladimírem Dopitou z IDS JMK mi byla poskytnuta data v textové podobě. Současně tak byla zajištěna budoucnost projektu. Data byla vygenerována z již existujícího systému, který slouží řidičům linek. Více o formátu těchto dat bude uvedeno v sekci 6.1. Bylo potřeba zajistit způsob, jak je možno data dostat do aplikace. Po zvážení všech výhod a nevýhod probraných výše, jsem zvolil cestu ukládání dat do souborů (resources).

Toto řešení se sebou nese problém aktualizace dat novější verzí v případě změny v jízdních řádech. Proto je k mobilní aplikaci dodávána i desktop aplikace, která je podrobněji probrána v následující kapitole.

5.3 Vyhledávání spojů

Termínem „spoj“ označuji cestu mezi dvěma a více zastávkami. V teorii grafů má tento termín obraz „cesta grafem“. Budu ještě rozlišovat spoj jedné linky a spoj více linek (možné přestupy), pokud nebude význam z kontextu zřejmý.

Problém hledání spoje ze zastávky A do zastávky B v dopravní síti lze redukovat na problém hledání nejkratší cesty v grafu z uzlu X do uzlu Y. Hrana z uzlu X do uzlu Y vede právě tehdy, když existuje linka, která projíždí zastávkami A a B bezprostředně za sebou. Hrana musí být orientovaná, protože existují linky, jejichž „zpáteční“ trasa není stejná jako ta „prvotní“[1]. Ohodnocení hran je složitější.

1. Například linka, která jezdí v kruhu.

5.3.1 Ohodnocení hran

Nabízí se hodnotit hrany grafu (tj. modelu dopravní sítě) mezi uzly X a Y hodnotou času, který potřebuje linka, aby dojela ze zastávky A do zastávky B, a nad takovýmto grafem vyhledávat. Toto hodnocení hran však nezohledňuje dynamičnost celého modelu. Linka nemusí jet ze zastávky hned. Tuto dobu je nutno připočítat k době přejezdu ze zastávky do zastávky. Rovněž je vhodné hodnotit hranu i číslem linky, aby se zamezilo zbytečným přestupům z jedné linky na druhou, jedou-li různé linky ve stejnou dobu na stejné trase.

5.4 Shrnutí

Z těchto poznatků jsem vyvodil, že použiji silně modifikovanou verzi Dijkstrova algoritmu. Zůstane z něj akorát myšlenka „rozbalování“ nejlacinějších uzlů jako prvních. To zajistí, že v dané fázi výpočtu budou prohledány všechny uzly s menší nebo rovnou cenou než právě aktuální uzel. Algoritmus může hledat efektivněji, spojíme-li ho s vhodnou heuristikou. Nabízí se dvě řešení.

- Využít znalosti geografické pozice zastávek a upřednostňovat uzly, které jsou ve směru cílové zastávky. Tato informace bohužel není v datech dostupná.
- Dálniční hierarchie. Heuristika inspirovaná velkými mapami. Je-li k dispozici informace, ve které části grafu se počáteční a koncový uzel nacházejí, může se vytvořit pomyslný graf na vyšší úrovni a spojit tyto části (nyní už uzly nového grafu) hranami, tzv. dálnicemi. Uvnitř částí pak vyhledávat klasicky. Toto řešení je vhodné spíše pro aplikace obhospodařující více klientů s větším množstvím paměti a výpočetní síly (např. cluster), protože si mohou ukládat spoje mezi jednotlivými částmi do cache paměti.

Z paměťových omezení platformy [7.2] vyplývá, že je nereálné uchovávat celou reprezentaci grafu v paměti. Nicméně, je-li „prohledávaný uzel“ zároveň uzel reprezentující cílovou stanici, může být hledání ukončeno a je zaručen správný výsledek. Pro nejhorší případy (ve smyslu nejvzdálenější cesty) bohužel může dojít k ukončení algoritmu z důvodu omezené velikosti paměti daného mobilního zařízení.

Kapitola 6

Konvertor dat

Protože data s informacemi o linkách dodaná z jiného systému jsou příliš obsáhlá a obsahují spoustu redundantních informací, je s mobilní aplikací dodávána také desktopová aplikace, která zajistí konverzi dat z vstupního formátu na formát, který podporuje mobilní aplikace. Tuto aplikaci budu označovat jako konvertor.

Konvertor je aplikace napsaná v Javě využívající Swing(A.1). Uživatel v ní nastaví cesty k jednotlivým souborům s daty, cestu k `jar` archivu mobilní aplikace, cílovou složku a případně úroveň filtru.

Filtr slouží ke snížení velikosti výsledného `jar` archivu. Cenou za toto snížení velikosti je vynechání některých nepravidelných[1] tras[2] linek. Tato myšlenka vznikla, protože dodaná data obsahují i spoje reprezentující linky jedoucí z, nebo do vozovny. Jedná se o metodu `filter(int treshold)`, kde parametr `treshold` určuje, kolikrát denně nejméně musí linka jet trasu, aby byly informace o trase a spojích linky na této trase zahrnuty do výstupních dat. Metoda se neaplikuje na vlaky a na noční autobusy v Brně. Je-li filtr nastaven na hodnotu 0, filtrace se neprovádí. Do výstupních dat se rovněž nepropustí informace o „neveřejných“ spojích, tj. o spojích označených příznakem `N`. Neveřejné spoje představují linky, které nenabírají pasažéry, ale jsou uvedeny ve vstupních datech.

Aplikace načte všechny soubory (kromě souborů se starými daty) z `jar` archivu mobilní aplikace a vygeneruje z nich nový `jar` archiv s již konvertovanými daty. `Jar` archiv má totiž stejný formát jako soubor `zip` a v Javě nechybí podpora pro práci se soubory `zip`. Tento postup vygeneruje spustitelnou aplikaci, protože do přeložených `class` souborů není třeba zasahovat a soubor `manifest.mf` zůstane stejný.

K `jar` archivu je vygenerován i `jad` deskriptor, do něhož je zapsána nová velikost `jar` archivu a verze midletu je inkrementována.

6.1 Formát vstupních dat

Informace o spojích a zastávkách jsou obsaženy v devíti souborech. Velikost všech těchto souborů je dohromady 20,2 MB.

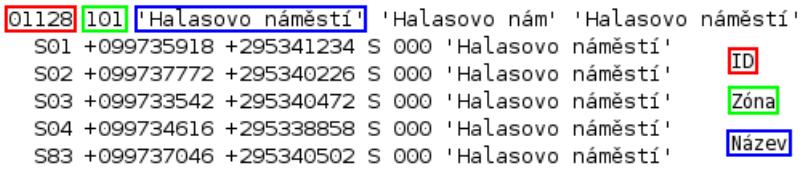
1. Nepravidelných je zde myšleno tak, že spoj linky nejede přes svou obvyklou trasu.
2. Trasou rozumím uspořádanou n-tici zastávek jedné linky.

Soubor zastávky.txt Obsahuje identifikátory zastávek v původním systému, příznaky tarifních zón, názvy, a má-li zastávka více sloupků, informace o nich. Zastávek, kde jezdí linky IDS JMK, je dohromady 2430. Záznam jedné ze zastávek je znázorněn na obrázku 6.1.

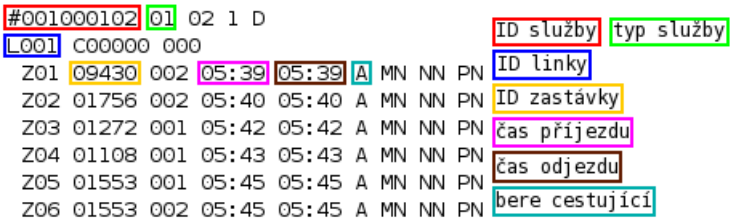
Soubor služby.txt Obsahuje informace o spojích jednotlivých linek. Nejprve je uveden identifikátor dne platnosti a typ služby[3], po něm následuje výpis spojů, které v daný den jezdí. Hlavička výpisu spoje obsahuje identifikátor linky a tělo identifikátory jednotlivých zastávek, kde linka zastavuje, a časů. Časy jsou dva, protože zastávka může mít „pobyt“, kdy čeká na jiný spoj. Záznam jedné z linek je znázorněn na obrázku 6.2.

Soubory pátek.txt - neděle.txt Obsahují identifikátory platnosti služeb, které určují, která služba (množina spojů) v daný den platí. Jízdní řády linek jsou totiž různé nejen přes víkend a v době svátku, ale mohou se lišit i ve všední dny. Toto platí zejména pro vlaky.

Vstupní formát dat je pevně daný, a protože je výstupem jiné aplikace, není třeba nějak důkladněji kontrolovat všemožné nekorektní vstupy. Uživatel by do těchto souborů neměl vůbec zasahovat.



Obrázek 6.1: Záznam se zastávkou Halasovo náměstí



Obrázek 6.2: Záznam se službou 001000102 a spojem linky 1

3. Typ služby je příznak, který určuje, zda-li je linka typu tramvaj, trolejbus, autobus, nebo vlak.

6.2 Formát výstupních dat

Vzhledem k omezením vyplývajícím z platformy Java ME bylo potřeba objem dat co nejvíce redukovat a irelevantní informace odfiltrovat. Protože vstupní data mají nízkou entropii, nebyl problém navrhnout formát lépe stravitelný pro mobilní zařízení.

Specifikace CLDC 1.1 ani MIDP 2.0 nedefinují maximální velikost `jar` archivu aplikace. Toto omezení je způsobeno konkrétními typy mobilních zařízení. Obecně platí, že čím je velikost menší, tím je aplikace z tohoto hlediska kompatibilnější.

Výstupní data jsou generována do balíku `cz.muni.fi.mobileIDS._data`. Data sestávají ze dvou typů souborů.

- **Soubor stations**

  Soubor stations obsahuje identifikátory jednotlivých zastávek, zóny, názvy a linky, které na daných zastávkách zastavují. Pro oddělování jednotlivých atributů jedné zastávky slouží znak `@`. (Nesmí se tedy vyskytovat v názvu zastávky.) Pro oddělování jednotlivých zastávek slouží znak s ASCII kódem 10, tedy `line feed`.

- **Soubor s informacemi o spojích dané linky** - *soubor linky*

  Množství vygenerovaných souborů linek[4] leží v intervalu `<m, n>`, kde `m` značí počet linek vynásobeno sedmi a `n` počet linek vynásobeno čtrnácti. Pro každý den v týdnu se totiž generují jiná data a linka může mít jeden nebo dva soubory v závislosti na tom, zda má dvě různé pravidelné trasy. Jeden soubor s informacemi tedy například mají linky, které jezdí v kruhu, dva různé soubory linky jezdící stejnou trasu z točny na točnu. Tuto vlastnost linky budu označovat jako *směr linky*.

  Směr linky je určen lince tak, že je nejprve zjištěna vzorová trasa s nejvíce jízdami (nejpravidelnější), potom podle porovnání pořadí zastávek ve zjišťované a vzorové trase, se trasa i s časy výjezdu umístí do správného souboru. Jeden soubor linky tak může a zpravidla i obsahuje více tras, protože se trasa může měnit ne tak „dramaticky“, aby byla nová trasa označena jako směr linky. Například pouze dvě zastávky se liší. Trasy jsou potom do souboru zapsány v pořadí od nejfrekventovanější po nejméně frekventovanou.

  Soubory linek jsou vždy pojmenovány identifikátorem linky se sufixem A nebo B, značícím směr linky, a jsou umístěny do balíku `cz.muni.fi.mobileIDS._data.XX`, kde `XX` značí den v týdnu.

  Samotný soubor linky má tento binární formát.

  1. Na prvním bytu je uložen typ služby.
  2. Až do bytu s hexadecimální hodnotou `81` jsou střídavě ukládány identifikátory sousedících zastávek a čas v minutách potřebný k přejezdu mezi dvěma sousedními zastávkami. ID zastávky je ukládáno na dva byty a čas na jeden. Tento řetězec bytů definuje jednu trasu linky.

4. Počet souborů například pro pátek je, použije-li se nulová úroveň filtrace, 401

3. Mezi bytem `81` a `FE` se nachází výčet výjezdů linky z první zastávky na trase. Čas je ukládán jako počet minut od půlnoci a je zakódován na dva byty.
4. Za bytem `FE` následuje další trasa, tedy opět postupně body č. 2, 3, 4, dokud není dosažen konec souboru.

Velikost výsledných dat je 387 kB při stupni filtrace nastaveném na 10, tedy filtraci, kdy nejsou zahrnuty trasy linek, které nejsou typu vlak nebo noční autobusová linka, a které nemají alespoň 10 jízd. V případě bez filtrace je velikost 1,1 MB. Nemá-li linka žádnou trasu, protože jí byli všechny odfiltrovány, není soubor této linky vůbec vytvořen a v souboru `stations` se neobjeví ID linky u zastávek, na kterých by měla linka zastavovat.

Kapitola 7

Vyhledávač

Data, která vyprodukovala aplikace Konvertor, slouží mobilní aplikaci, o níž je tato kapitola.

7.1 Kompatibilita

Specifikace, které jsou součástí Javy ME, jsou velmi benevolentní a je v nich spousta doporučení na implementaci. Implementace různých výrobců mobilních zařízení se tak liší. Aplikace se proto může chovat na různých zařízeních různě.

Aby byla zaručena co největší kompatibilita s co nejvíce typy zařízení, je nutné psát aplikace takové, aby velikost výsledného `jar` archivu, požadavky na operační paměť a velikost displeje zařízení byly co nejmenší.

Na rozdíl od Javy SE a EE, kde hlavními požadavky na aplikace je znovupoužitelnost kódu a snadná údržba aplikací, je to v Javě ME kompatibilita a rychlost. Je užitečné si uvědomit, že řešení psaná pro mobilní zařízení nemohou být nasazena v clusteru, že ve většině případů není klientem firma, která bude vyžadovat dodaný systém udržovat a vylepšovat. Aplikace mívají značně omezené zdroje. Před čtyřmi lety se pohybovala průměrná maximální velikost `jar` archivu podporovaná v zařízeních okolo 64 kB a velikost paměti 210 kB[1] [10], dnes už je situace lepší a zařízení začínají podporovat neomezenou[2] velikost spustitelného souboru, ale ne každý vlastní nejnovější telefon. Má-li aplikace fungovat i na těchto starých zařízeních, je potřeba snížit tyto hodnoty. Některé techniky snižování těchto hodnot nejsou v souladu s principy ve standardní a enterprise edici Javy. Několik příkladů.

- Jakýkoli framework představuje režii navíc.
- MVC není prioritou.
- Držet objekty v paměti jen na dobu potřebně nutnou.
- Zvážit použití neměnitelných tříd [3].
- Udržovat počet tříd na rozumné úrovni.
- Používat profiler a memory monitor, jež jsou součástí Sun Java Wireless Toolkit.

1. Tyto údaje platí konkrétně pro telefon Nokia 3100 a 3120.
2. Je samozřejmě omezena velikostí paměti v telefonu pro ukládání dat.

- Používat obfuskátor.

Obfuskátor zabrání možné dekompilaci tříd z tzv. bytecode podoby do čitelné podoby. Chrání tak know-how, a zároveň sníží velikost výsledného souboru.

Velikost spustitelného `jar` archivu (data s filtrací stupně dva) mobilní aplikace, jež je součástí této práce, je po obfuskaci 730,4 kB, nicméně lze snížit zvýšením filtrace až pod 500 kB. Šířka displeje by měla být pro správné zobrazení spojů alespoň 130 pixelů a rozumná velikost operační paměti je alespoň 1 MB.

Aplikace využívá vysokoúrovňového API pro tvorbu uživatelského rozhraní. To zaručuje, že aplikace bude fungovat na všech přístrojích správně, ale může vypadat jinak. Každý výrobce si totiž implementuje formulářové prvky jinak. Výhodou tohoto řešení jsou i implementované formulářové prvky jako `DateField`, pro výběr času, podpora pro textový vstup od uživatele do prvku `TextField` nebo podpora pro stylus.

Nízkoúrovňové API, sice dovoluje více, protože dovoluje kreslit grafiku na plátno (třída `Canvas`), ale například textový vstup od uživatele by musel být řešen na úrovni událostí, jež klávesy zařízení vyvolávají. Problém by potom byl například se znakem mezery, která je na různých zařízeních na jiné klávese, nebo u zařízení s kompletní klávesnicí.

7.2 Případy užití

Případy užití má aplikace dva. Níže jsou textově popsány a zaneseny do diagramu.

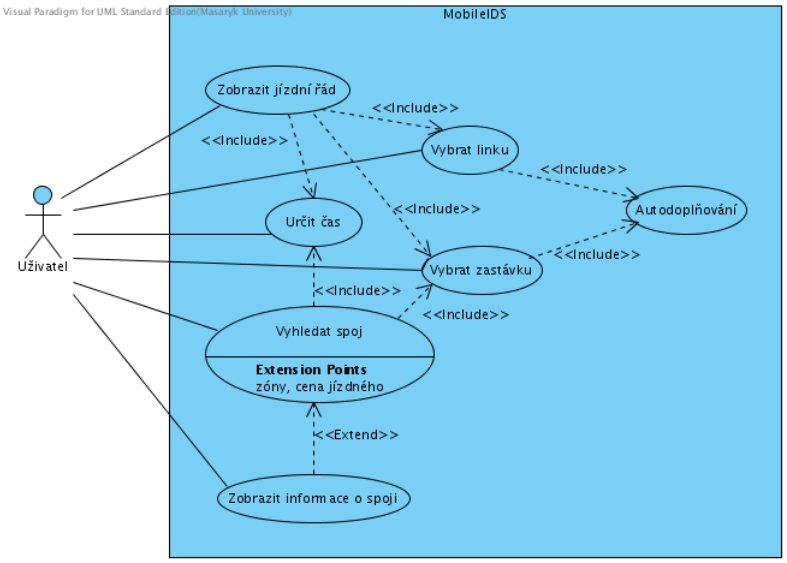
1. **UC Vyhledávání**
   - Uživatel spustí midlet `Vyhledavac`.
   - Název počáteční a cílové zastávky jsou doplněny hodnotami z předchozího spuštění midletu.
   - Nevyhovují-li stávající hodnoty, vyplní nové.
     - Vyplní počáteční zastávku.
       - Aplikace „našeptává“ ty vhodné.
     - Vyplní cílovou zastávku.
       - Aplikace „našeptává“ ty vhodné.
     - Zvolí čas, pokud nevyhovuje aktuální.
   - Zvolí příkaz `Vyhledat`.
     - Pokud na řetězec začíná alespoň jedna skutečná zastávka, je vybrána ta podle abecedy první. To platí pro oba řetězce.
     - Mají-li oba řetězce svůj obraz mezi skutečnými zastávkami, spustí se hledání.
       - Proběhlo-li hledání neúspěšně, vypíše hlášení „Spoj nebyl nalezen.“.
       - Proběhlo-li hledání úspěšně, zobrazí se výpis spoje a přidá se příkaz `Detaily`.

· Uživatel zvolí příkaz `Detaily`, jsou mu zobrazeny podrobnější informace o spoji.

– Pokud na alespoň jeden z řetězců, zadaných jako názvy zastávek, nezačíná žádná skutečná zastávka, je uživatel upozorněn hlášením „XX zastávka neexistuje.“.

2. **UC Zobrazení jízdního řádu**

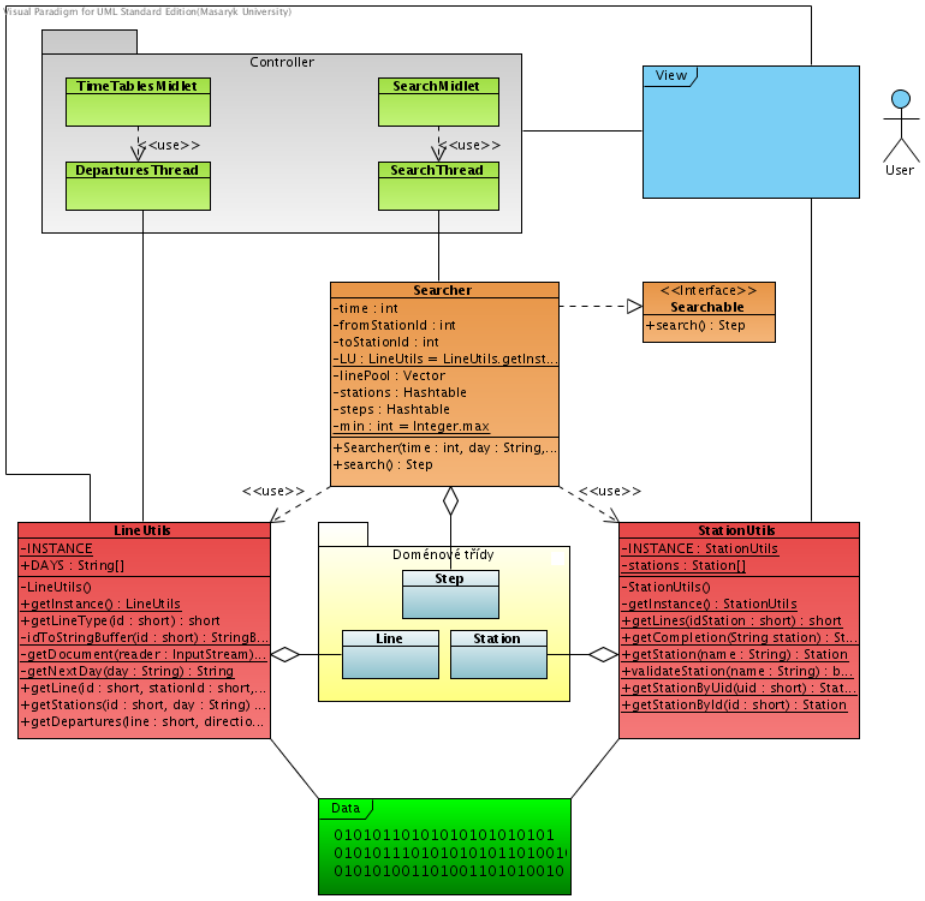
- Uživatel spustí midlet `Jizdni rady`.
- Vyplní hodnoty.
  - Vyplní linku.
    * Existuje-li linka, jsou doplněny zastávky a směr.
  - Jsou-li zastávky a směr vyplněny, uživatel z nich vybere.
- Zvolí příkaz `Zobrazit`.
  - Jsou-li hodnoty vyplněny správně, je mu zobrazen jízdní řád.
  - Nejsou-li hodnoty vyplněny správně, je upozorněn.



Obrázek 7.1: Use case diagram

7.3 Schéma a implementace

Aplikace sice nedodržuje model MVC, protože vrstva view využívá některé metody tříd `LineUtils` a `StationUtils` k překládání identifikátorů na názvy. Nicméně toto porušení není tak dramatické a na schématu je pro lepší pochopení aplikace znázorněna jako MVC. Pro jednoduchost nejsou do diagramu zaneseny také metody netýkající se aplikační logiky aplikace.



Obrázek 7.2: Diagram celé aplikace

7.3.1 Midlety

Sada midletů mobilní aplikace sestává ze dvou midletů. `TimeTableMidlet` a `SearchMidlet`, součástí těchto tříd jsou vnořené třídy dědící od třídy `Thread`.

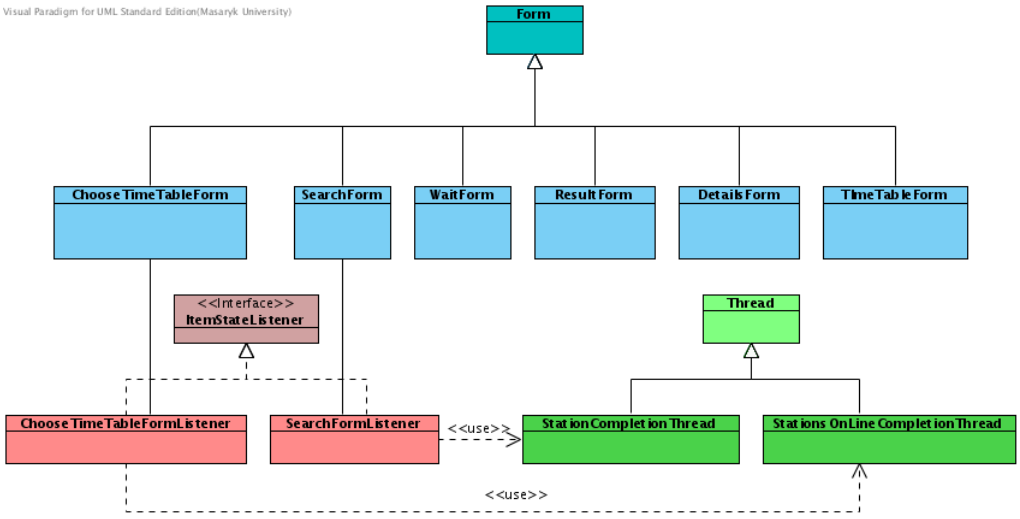
- **SearchThread** Probíhá v něm hledání spojů.

- **TimeTableThread** Probíhá v něm vyhledání jízdního řádu.

Tyto dvě třídy midletů ovládají tok aplikace. Midlet `SearchMidlet` obsahuje navíc vnořené vlákno `LoadSettingsThread`, které je spouštěno při startu midletu a vyplní názvy zastávek názvy z předchozího běhu aplikace. Názvy jsou uloženy ve skladišti záznamů. Zadá-li uživatel některý z příkazů, je událost zpracována právě v těchto midletech. To proto, že midlety implementují rozhraní `CommandListener` a jsou navěšeny na formuláře, jež ovládají.

7.3.2 Formuláře

Základní třídou, od které dědí všechny třídy sloužící pro zobrazování informací, je pro vysokoúrovňové API třída `Form`. Formuláře `ChooseTimeTableForm`, `TimeTableForm` jsou přihlášeny ke třídě `TimeTableMidlet` a formuláře `SearchForm`, `WaitForm`, `ResultForm` a `DetailsForm` jsou přihlášeny ke třídě `SearchMidlet`. Formuláře pro výběr jízdního řádu a pro hledání spoje mají navíc `ItemStateListenery` `ChooseTimeTableFormListener` a `SearchFormListener` kvůli doplňování názvů. Tyto listenery jsou notifikovány, nastane-li změna na prvcích formuláře. Listenery spouští vlákna `StationsOnLineCompletion` a `StationCompletionThread`, díky nimž je obrazovka displeje neustále připravená k interakci.



Obrázek 7.3: Class diagram view vrstvy

7.3.3 Aplikační logika

K datům přistupují třídy `LineUtils` a `StationUtils`, můžeme na ně nahlížet jako na DAO objekty, protože adaptují těžkopádný přístup k datům přes proudy (streamy) na při-

jatelnější rozhraní, vracející už hotové instance doménových tříd. Obě třídy jsou navrženy podle návrhového vzoru Singleton [5].

Doménové třídy představují třídy `Station`, jejíž instance reprezentují zastávky, `Line` pro linky a třída `Step`, jejíž instance slouží pro uchovávání si informací o spoji.

Hledání probíhá ve třídě `Searcher`, přesněji ve vlákně `SearchThread`, které si vytvoří instanci třídy `Searcher`. Algoritmus implementuje myšlenku z Dijkstrova algoritmu. Využívá tříd `LineUtils` a `StationUtils` k přístupu k datům. `Vector linePool` slouží jako cache paměť mezi relativně drahými operacemi na otvírání, čtení a zavírání souborů. `Hashtable stations` je kolekce pro ukládání doposud napočítaných vzdáleností (tedy časů od počáteční zastávky) do navštívených zastávek.

Nebudu-li zbytečně zabíhat do detailů, tak algoritmus v každém kroku vybere zastávku s nejmenší vzdáleností. Pro všechny její linky, které ještě nejsou v kolekci `linePool` (tzn. nová linka a možný přestup), a pro linku, jejíž identifikátor se shoduje s identifikátorem linky zastávky (značí, kterou linkou jsme přijeli na zastávku), ohodnotí a uloží sousední zastávky. Obsahuje i mechanizmus na vypořádávání se se směry linek.

Kapitola 8

Závěr

Platforma Java ME je navržena zejména pro aplikace využívající ke své činnosti síť. Aplikační logika je tak přesunuta na jiné místo a mobilní aplikace jen využívá její služby. Toto řešení si žádá platit poplatky za informační kanál. Tvorba užitečných offline aplikací je kvůli omezeným paměťovým prostředkům ztížená, nicméně možná.

Účelem této práce bylo vytvořit aplikaci, která hledá spoje v dopravní síti. Výsledná aplikace má jednoduché a intuitivní ovládání a je užitečná v případě, že potřebujeme efektivněji cestovat veřejnou dopravou. Aplikace není nijak těsně závislá na obsahu dat, proto lze při dodržení formátu využívat aplikaci i pro jiné města. Aplikace byla otestována na emulátorech Nokia a Sony Ericsson. V současné době vzniká internetová verze v rámci projektu do předmětu PA165. Ta bude poskytovat službu vyhledávání pomocí webových služeb, proto je možné vytvořit jednoduchou mobilní aplikaci, která těchto služeb bude využívat, a zajistit tak centralizovanost a aktuálnost dat za cenu investice ze strany klientů do poplatků za informační kanál.

Literatura

[1] Topley, K.: *J2ME v kostce*, Grada Publishing, 2004, 80-247-0426-9. 2.1, 2.7.1

[2] White, J. a Hemphill, D. a Manning, D.: *Java 2 Micro Edition: Java in small things*, Manning Publications, 2002, 0-7695-1050-7. 3.1

[3] Bloch, J.: *Java efektivně - 57 zásad softwarového experta*, Grada Publishing, 2002, 80-247-0416-1. 7.1

[4] Fowler, M.: *Refaktoring - Zlepšení existujícího kódu*, Grada Publishing, 2003, 80-247-0299-1.

[5] Freeman, E. a Sierra, K. a Bates, B.: *Head First Design Patterns*, O'Reilly, 2004, 0-596-00712-4. 7.3.3

[6] Collins-Sussman, B. a Fitzpatrick, B. a Pilato, M.: *Version Control with Subversion*, Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato, 0-596-00448-6, 2007.

[7] Ortiz, E.: *Introduction to OTA Application Provisioning*, 2002, Dostupný z URL <http://developers.sun.com/mobility/midp/articles/ota/> (prosinec, 2007). 2.6

[8] *MID Profile*, 2006, Dostupný z URL <http://java.sun.com/javame/reference/apis/jsr118/> (prosinec, 2007). 2.2

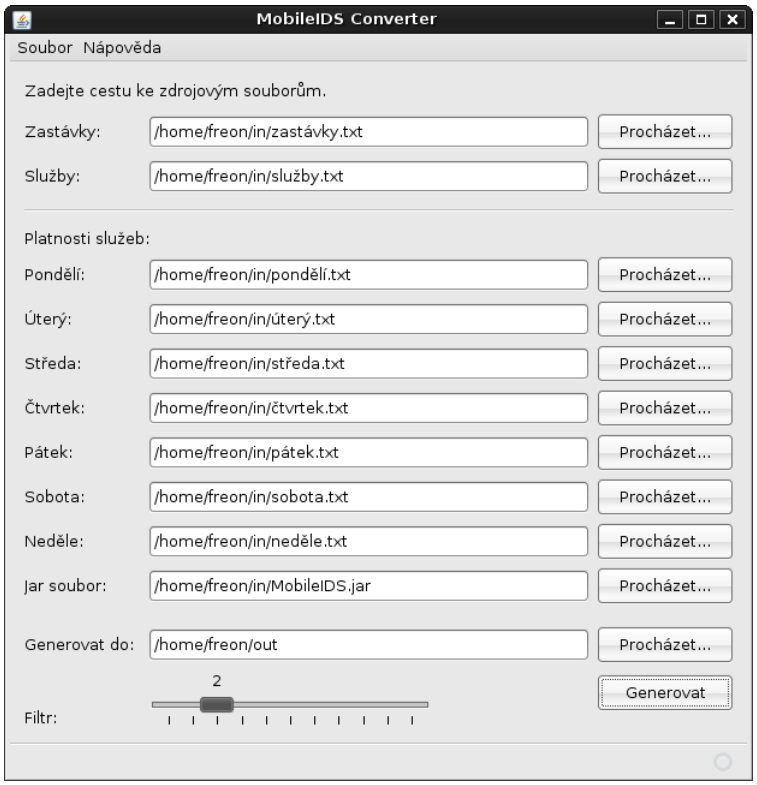
[9] *JSR–000118 Mobile Information Device Profile 2.0*, 2004, Dostupný z URL <http://jcp.org/aboutJava/communityprocess/final/jsr118/> (prosinec, 2007).

[10] *J2ME Polish: MIDP/2.0 Devices*, Dostupný z URL <http://www.j2mepolish.org/devices/platform_MIDP_2.0.html> (prosinec, 2007). 2.3, 7.1

[11] *Android*, Dostupný z URL <http://code.google.com/android/> (prosinec, 2007). 2.1

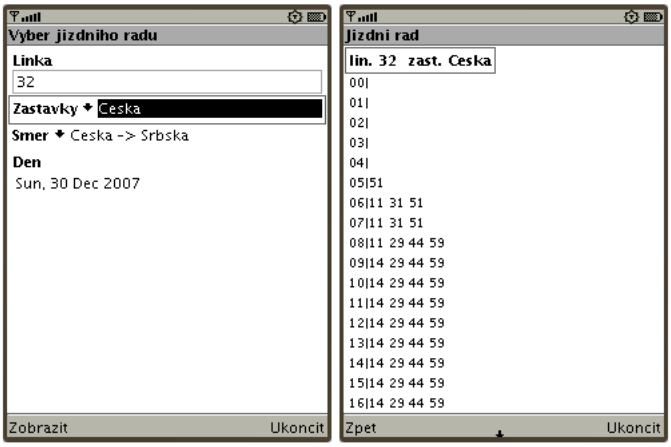
Příloha A

Podoba aplikací



Obrázek A.1: Aplikace Konvertor

Obrázek A.2: Midlet pro vyhledávání



Obrázek A.3: Midlet pro zobrazení jízdních řádů

Příloha B

Obsah CD

Součástí práce je přiložené CD, které obsahuje obě popsané aplikace ve formě projektů pro IDE Netbeans 6.0, data se zastávkami a spoji linek, dokumentaci ve formátu Javadoc pro mobilní aplikaci a tuto bakalářskou práci ve formátech pdf, html, DocBook a LaTeX.