Technická univerzita v Liberci

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B 2612 – Elektrotechnika a informatika

Studijní obor: 2612R011 – Elektrotechnické informační a řídící systémy

Využití Atlas AMapy API ke sledování polohy objektu

Position monitoring by using Atlas AMapy API

Bakalářská práce

Autor: **Martin Obrátil**

Vedoucí bakalářské práce: Ing. Přemysl Svoboda

Konzultant: Ing. Tomáš Martinec

V Liberci dne 15. 5. 2008

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou 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 bakalářské práce a prohlašuji, že **souhlasím** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

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

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

Datum

Podpis

zde patří zadání bakalářské práce

Abstrakt

Účelem této bakalářské práce je vytvoření webové a klientské aplikace pro zaznamenávání polohy objektů systémem GPS a následné zobrazení polohy pomocí webové aplikace založené na JavaScriptové komponentě Atlas AMapy API. Obě aplikace byly vytvořeny v jazyce Java v prostředí J2SE 5.0 a J2EE 5.0 s podporou Spring Frameworku. Je zde popsán způsob komunikace mezi klientskou a webovou aplikací, definovány rozhraní a doporučení pro vytváření nových komponent pro klientská zařízení.

Klíčová slova: AMapy, mapy, sledování objektu, J2SE 5.0, J2EE 5.0, Java, Spring Framework, GPS

Abstract

The purpose of thesis is creation of web and client application for position monitoring by using GPS system and vizualization of routes in web application based on JavaScript component Atlas AMapy API. Both applications were created by Java language in J2SE 5.0 and J2EE 5.0 enviroments with support of Spring Framework. It describes communication methods used between client and web application, interfaces and recommendations for making new componetnts for client devices.

Keywords: AMapy, maps, position monitoring, J2SE 5.0, J2EE 5.0, Java, Spring Framework, GPS

OBSAH

Úvod

Cílem této práce je vytvoření funkčního softwaru pro sledování geografické polohy objektu. První až třetí kapitola se zabývá rešerží technologií, jejichž znalost je nezbytná pro řešení dané problematiky. Čtvrtá kapitola již obsahuje samotné řešení problému sledování objektu. Pátá kapitola řeší další problém, a to způsob zaznamenávání souřadnic a jejich vykreslení v podobě trasy na mapovém podkladu. Poslední kapitola poskytuje jednoduché shrnutí výsledku práce a také vytyčení cílů pro další možné rozšiřování funkcionality softwaru.

S rozvojem internetu se čím dál více rozmáhá trend nahrazovat desktopové aplikace za aplikace webové. Tento trend je dán množstvím výhod, které nám webové aplikace oproti desktopovým přináší. Proto je řešení bakalářské práce postaveno za podpory webových technologií jako je komponenta Atlas AMapy API pro kreslení map a Spring Framework pro snadnou tvorbu robustních webových aplikací v jazyce Java. Mezi hlavní výhody těchto aplikací se řadí především vyšší flexibilita, což znamená dostupnost aplikace odkudkoliv, kde existuje konektivita na celosvětovou síť Internet, absence problémů s *deploymentem*[1] softwaru na pracovní stanice a také možnost vytvářet aplikace skutečně multiplatformně.

[1] *Deployment* je anglické slovo označující činnosti spojené s instalací a údržbou softwaru. [1]

1 Navigační systémy

Tato kapitola se věnuje popisem systémů pro určení geografické polohy objektu.

1.1 Global Positioning System

Global Positioning System [2] (dále jen GPS) je globální navigační satelitní systém vyvinutý a provozovaný Ministerstvem obrany Spojených států amerických. Systém započal vývoj v roce 1974 a v roce 1994 byla vypuštěna poslední družice. Tím se systém stal plně funkční a dostupný po celém světě. Systém původně určený jen pro vojenské účely se stal díky velké poptávce po navigačním systému dostupný také pro civilní účely a nyní je jeho využití možné v nejrůznějších odvětvích.

Systém GPS se dělí na následující podsystémy:

1. Kosmický
2. Kontrolní
3. Uživatelský

1.1.1 Kosmický podsystém

Systém v současnosti tvoří 24 družic, z toho 3 záložní. Obíhají ve výšce 20200 km nad mořem po šesti různých obežných drahách vzájemně posunutých o 60 °. Družice jsou vybaveny přijímačem, vysílačem, velmi přesnými atomovými hodinami a také detektorem kontrolující dodržování zákazu nukleármích zkoušek. Družice dokáže přijímat povely z řídícího centra a na jejich základě koriguje svoji dráhu. Dále obsahuje systém pro diagnostiku, který řídícímu centru posílá informace o stavu družice.

Trilaterace[2]

Je způsob určování relativní polohy objektu pomocí vztahů v trojúhelníku. Poloha se vypočítá s využitím tzv. referenčních bodů, jejichž souřadnice jsou známy. Určujeme-li polohu objektu pouze ve 2D prostoru, potom pro jednoznačné určení polohy jsou zapotřebí 3 referenční body. Nás však zajímá určení polohy objektu ve 3D prostoru a k tomu je nutné znát alespoň 4 referenční body. To dokazují následující odstavce.

Výpočet polohy objektu

GPS přijímač měří rozdíl dvou časů – okamžik odeslání informací z družice a okamžik příjmu signálu. Tento rozdíl zapíšeme takto

$$\Delta t = t_p - t_o, \tag{1}$$

kde t_p reprezentuje čas příjmu signálu a t_o čas odeslání signálu z družice. Dále je nutné definovat opravu hodin na družici

$$\delta_o = t_o - T_o \tag{2}$$

a na přijímači

$$\delta_p = t_p - T_p. \tag{3}$$

T_o a T_p jsou spravné časy na družici a na přijímači. Dosazením vztahů (2) a (3) do vztahu (1) vznikne následující vztah

$$\Delta t = \delta_p + T_p - \delta_o - T_o = \Delta T + \delta_p - \delta_o \tag{4}$$

kde ΔT představuje přesný čas, který signál urazil z družice k přijímači. Nyní z Δt vyjde tzv. pseudovzdálenost družice a přijímače

$$R = c \cdot \Delta t = c \cdot \Delta T + c\delta_p - c\delta_o \tag{5}$$

kde $r = c \cdot \Delta T$ je přesná vzdálenost mezi družicí (v čase T_o) a přijímačem (v čase T_p). Dále je nutné získat tvar rovnice tak, aby v něm vystupovaly souřadnice přijímače a družice

$$R = \sqrt{(x_p - x_o)^2 + (y_p - y_o)^2 + (z_p - z_o)^2} + c\delta_p - c\delta_o \tag{6}$$

Vstupními neznámými potřebné pro výpočet polohy je poloha přijímače (x_p, y_p, z_p) a oprava hodin přijímače δ_p. Proměnné (x_o, y_o, z_o) a oprava hodin družice δ_p jsou obsaženy v odeslaných efemeridách. Celkem jsou čtyři neznámé, tudíž pro určení polohy v 3D prostoru (tj. zeměpisná šířka, délka, a nadmořská výška) je zapotřebí příjem signálu z celkem 4 družic.

1.1.2 Kontrolní podsystém

Úkolem tohoto podsystému je sledování a výpočet trajektorií družic a zároveň jejich systémová údržba. Nezbytností je také zajištění přesného chodu atomových hodin.

Tento podsystém tvoří síť pěti monitorovacích stanic a tří pozemních řídících stanic. Hlavní řídící stanice ma sídlo na letecké základně v Colorado Springs.

1.1.3 Uživatelský podsystém

Uživatelský podsystém tvoří samotný GPS přijímač. Dnes existuje řada různých typů GPS přijímačů určených pro různé účely. Tyto přijímače získávají signál minimálně z tří a maximálně z dvanácti družic. Na základě přijatých dat vypočítají svou polohu s určitou přesností, která závisí na počtu družic od kterých přijímač signál přijal.

1.2 Global Navigation Satellite System

Global Navigation Satellite System [2] (dále jen GLONASS) je ruský navigační systém, jehož vývoj započal již v roce 1976, ale dosud není plně funkční. Plné spuštění GLONASSu se plánuje na rok 2009 a účely pro které bude využíván by měly být hlavně vojenské. Tento systém tvoří, stejně jako u GPS, 24 družic, z toho 3 záložní.

1.3 Galileo

Galileo [2] je projekt Evropské unie, který si klade za cíl vytvořit civilní globální navigační systém a konkurovat tak americkému – armádnímu – navigačnímu systému GPS a ruskému GLONASSu. Systém bude tvořen 27 družicemi obíhajících ve výčce 23 000 km nad zemským povrchem. Na rozdíl od výše uvedených systémů má Galileo poskytovat vyšší přesnost (až pod 1 m) a větší pokrytí signálem.

1.4 Wi-Fi Positioning System

Wi-Fi Positioning System [3] (dále jen WPS) se systém vyvinutý společností Skyhook Wireless, který zjišťuje geografickou polohu pomocí bezdrátové sítě WI-FI. Jedná se čistě o softwarové řešení, kde klient zjistí fyzické adresy (MAC) v okolí dostupných přístupových bodů (AP). Těmto adresám (na základě dotazu na referenční databázi umístěnou na internetu) přiřadí geografickou polohu, ze kterých potom vypočte pravděpodobnou polohu objektu. Tento systém může být za určitých okolností přesnější než satelitní systém GPS, ale jeho využití má smysl hlavně v uzavřených interiérech nebo městských zástavách. Z těchto důvodů je nejlepší využít jej v kombinaci s GPS, protože signál z družic se do zastřešených interiérů nedostane.

2 Problematika komunikace mezi klientem a serverem

V současnosti je dostupno mnoho technologií pro přenos informací z klienta na server. Tyto technologie jsou voleny v závislosti na požadavcích, které jsou na ně kladeny. Mezi možné požadavky patří:

- Mobilita
- Rychlost
- Nízká cena za přenesená data
- Dostupnost

Tento projekt má velké požadavky na mobilitu. Jelikož se klient bude pohybovat, je zapotřebí, aby měl možnost posílat informace pokud možno odkudkoliv. Odesílání souřadnic bude prováděno přibližně jednou za 1 až 10 sekund, takže nároky na přenesená data nebudou nijak veliká. Pro tyto požadavky byly zvažovány následující technologie:

- Short Message Service (SMS)
- Sítě dvou a půlté gerace (2.5G)
- Wi-Fi

2.1 Short Message Service (SMS)

SMS je služba poskytovaná prostřednictvím sítě GSM, která umožňuje posílání krátkých textových zpráv. Jelikož se jedná o mobilní bezdrátovou technologii, splňuje perfektně požadavek na mobilitu. Množství dat které je možné přenést v jedné zprávě je 160 znaků, což postačuje na poslání celkem čtyř souřadnic. Více informací o této problematice je rozebíráno v kapitole 4.6. Problémem tohoto způsobu komunikace je jeho cena za přenos jedné zprávy. V České republice dle ceníků společností provozující mobilní sítě GSM se cena jedné SMS zprávy může pohybovat přibližně od 1 do 2 Kč [4, 5, 6]. Předpokládá-li se, že bude zapotřebí sledovat objekt po dlouhou dobu, mohou náklady na komunikaci být velmi vysoké.

2.2 Internet

Z hlediska ceny se nabízí jako daleko lepší řešení použít pro komunikaci síť internet. Mobilní připojení k internetu je možné nejčastěji prostřednictvím následujících technologií.

2.2.1 Sítě dvou a půlté gerace (2.5G)

Do této množiny technologií spadají technologie využívající infrastrukturu sití GSM a CDMA. Mezi nejznámější stojí za to uvést GPRS a EDGE. Tyto technologie poskytují paketový přenos, který je možné využít mimo jiné pro internetovou komunikaci a jejich dostupnost je velmi vysoká. V České republice na většině území. Datový tok těchto technologií je řádově několik desítek kbps (upload i download), takže je pro účely tohoto projektu dostačující.

2.2.2 Wi-Fi

Zde se jedná o technologie bezdrátových sítí podle standardu IEEE 802.11. Poskytují více než dostačující datový tok, ale mobilita této technologie není pro tento projekt dostatečná.

3 Získávání geografických souřadnic z GPS modulu

Na trhu je dnes k dispozici mnoho různých typů GPS modulů. Pro tento projekt je důležitým parametrem způsob, jakým jsou GPS moduly připojeny k počítači. Způsoby jsou následující: bezdrátově přes technologii Bluetooth, nebo přes konektory USB a RS232. Ve všech případech zajišťuje komunikaci protokol NMEA 0183 [7, 8].

3.1 NMEA 0183

NMEA 0183 je protokol vyvinutý původně pro vojenské účely, aby definoval standard pro komunikaci mezi různými vojenskými zařízeními. GPS moduly tento standard využívají. Je-li GPS modul připojen k počítači, automaticky začne posílat zprávy ve formátu NMEA. Těmto zprávám se též říká věty (sentences) a mají následující tvar

```
$XXYYY,par1,par2,. . .<CR><LF>
```

kde `XX` je identifikátor modulu GPS a `YYY` označuje typ zprávy. Za těmito identifikátory následují parametry oddělené čárkami obsahující samotné informace. `<CR><LF>` (Carriage Return, Line Feed) jsou znaky nového řádku. Data přijatá z GPS modulu mohou vypadat následovně:

```
$GPGSA,A,3,29,26,22,09,07,05,04,,,,,,1.7,1.0,1.4*30

$GPGSV,3,1,11,09,84,297,41,05,48,256,45,07,38,059,41,26,22,178,41*74

$GPGSV,3,2,11,24,13,063,00,14,12,324,00,30,12,251,00,22,12,286,38*78

$GPGSV,3,3,11,29,10,173,35,04,09,105,30,18,06,254,00*46

$GPRMC,170138.615,A,4912.2525,N,01635.0378,E,0.04,16.43,280705,,*32

$GPGGA,170139.615,4912.2526,N,01635.0378,E,1,07,1.0,357.5,M,43.5,M,
0.0,0000*7D
```

Pro potřeby tohoto projektu je nejdůležitější zpráva typu `GGA`, která ve svých parametrech uvádí čas, pozici v trojrozměrném prostoru a další data specifikující kvalitu signálu a kontrolní součet. Dekodování takové věty může vypadat následovně:

`$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47`

Zápis	**Hodnota**	**Popis**
`123519`	12:35:19 UTC	Udává čas ve formátu hhmmss
`4807.038`	48° 7.038’	Zeměpisná šířka ve formátu ddmm.mmmm
`N`	Severní šířka	`N` – severní šířka, `S` – jižní šířka
`01131.000`	11° 31’	Zeměpisná délka ve formátu dddmm.mmmm
`E`	Východní délka	`E` – východní délka, `W` – západní délka
`1`	Pozice byla úspěšně určena	Indikátor kvality určení pozice
`08`	8	Počet dostupných satelitů
`0.9`	0,9	
`545.4`	545,4 metrů	Výška nad mořem
`M`	Výška nad mořem je udávaná v metrech.	Jednotka ve které je udávaná výška nad mořem.
`46.9`	46,9 metrů	Rozdíl hladiny geoidu a elipsoidu WGS84
`M`	Rozdíl hladiny geoidu a elipsoidu je udáván v metrech.	Jednotka ve které je udáván rozdíl hladiny geoidu a elipsoidu.
`*47`	*47	Kontrolní součet

4 Klientská knihovna Observed

Tato kapitola obsahuje návrh a implementaci klientské knihovny *Observed* pro načtení geografických souřadnic a jejich následné odeslání na server.

4.1 Definice požadavků

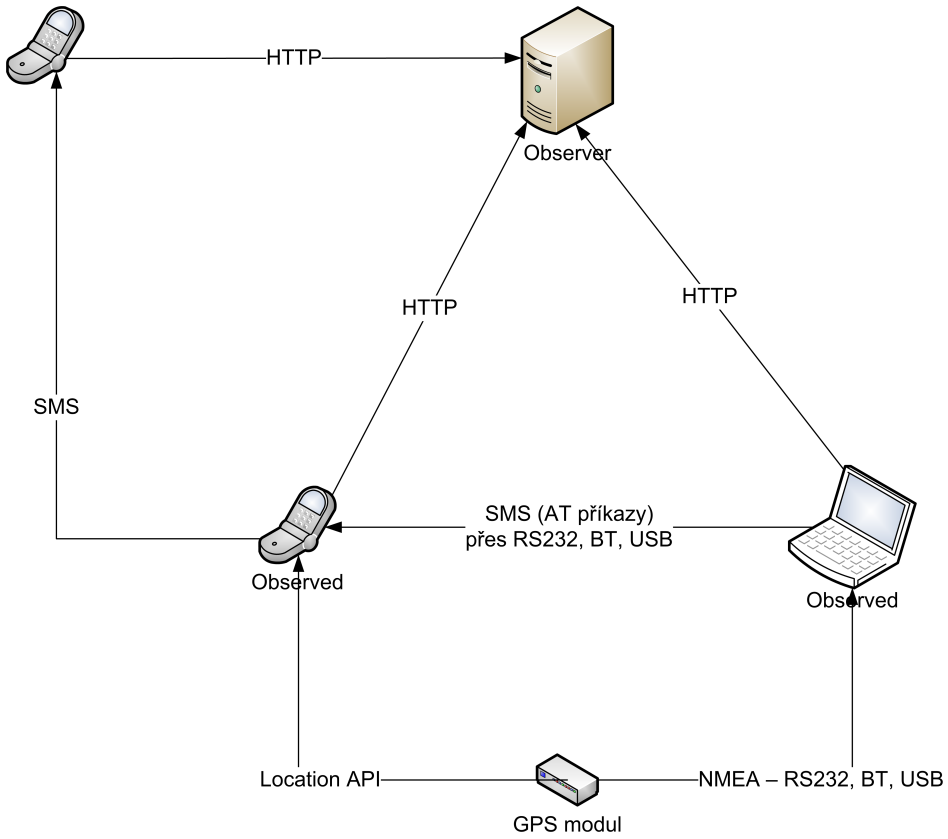
Jedním ze základních požadavků je možnost provozovat software na většině dnes používaných operačních systémech, např. Microsoft Windows, Linux, Mac OS X. Dále je kvůli požadavku na vysokou mobilitu žádoucí, aby aplikaci bylo možné provozovat na mobilních zařízeních, jako jsou mobilní telefony a PDA zařízení. Knihovna musí umět přijímat informace z GPS modulů, které jsou připojeny přes různá komunikační rozhraní (USB, RS232, Bluetooth). Podporu pro další komunikační rozhraní by mělo být snadné dodatečně implementovat. Dále je nutné, aby program uměl přijmout SMS zprávu a z ní vyčíst geografické souřadnice. Odesílání bude realizováno pomocí zprávy SMS (prostřednictvím mobilního telefonu za pomoci AT příkazů), nebo pomocí protokolu HTTP přes internet. Protokolem HTTP bude realizován samotný upload souřadnic na server.

4.2 Programové prostředí

Pro projekt byl zvolen programovací jazyk Java, protože nejlépe pokrývá uvedené požadavky. Běh programu napsaném v tomto prostředí bude zajištěn na většině operačních systému a zároveň je možný běh na velkém množství mobilních telefonů. Zdrojový kód programu napsaný v Javě se nepřekládá do strojového kódu, ale do tzv. bajtkódu, tudíž aplikace bude vyžadovat pro svůj běh *virtuální stroj* (Java Virtual Machine, dále jen JVM). JVM existuje v několika verzích:

- Java Micro Edition
- Java Standard Edition
- Java Enterprise Edition

Velkou výhodou této platformy je skutečnost, že aplikaci stačí napsat jednou a *virtuální stroj* zajistí běh na všech běžně dostupných operačních systémech. Případně se pro různé systémy provedou jen drobné modifikace v kódu.



Obr. 1. Schéma komunikace mezi zařízeními

4.3 Základní programová rozhraní

Základní programová rozhraní se dělí do tří balíčků (packages): `cz.obratil.martin.observed.senders`, `cz.obratil.martin.observed.recievers` a `cz.obratil.martin.observed.drivers`. V knihovně *Observed* jsou jakákoliv odesílaná resp. přijímaná data zapouzdřena do tzv. *zpráv*. *Zprávu* reprezentuje rozhraní `IMessage`, která zapouzdřuje samotná data – v našem případě jsou to geografické souřadnice.

Rozhraní `ISender` v balíku `cz.obratil.martin.observed.senders` definuje formu odesílání *zpráv*:

```
public interface ISender {
    public void sendMessage (IMessage val);
}
```

Metoda `sendMessage(IMessage)` odešle zprávu předanou v argumentu metody. Toto rozhraní implementují všechny třídy implementující posílání zpráv po různých techno-

logiích. Těmto objektům se říka *odesílatelé* (senders).

Rozhraní `Reciever` v balíku `cz.obratil.martin.observed.senders` slouží pro příjem zpráv:

```
public interface IReciever {
    public void setSentenceRecievedListener(
        ISentenceRecievedListener listener);
    public void StartListening();
    public void StopListening();
}
```

Metoda `setSentenceRecievedListener()` nastaví objekt implementující rozhraní `ISentenceRecievedListener`, jehož úkolem je zapouzdřovat kód, obsluhující událost při přijetí zprávy. Tento kód je umístěn v metodě `SentenceRecieved(IMessage)`, jejímž vstupním parametrem je *zpráva* `IMessage`. Tento *listener* je volán pouze v případě, že naslouchání je aktivováno metodou `StartListening()`. Naslouchání se zastavuje metodou `StopListening()`. Toto rozhraní implementují všechny objekty přijímající zprávy a říká se jim *přijímače* (recievers).

Je zde patrné rozdělení úloh obou rozhraní. `ISender` předepisuje metodu pro odesílaní *zpráv* a `IReciever` naopak metody pro příjem *zpráv*. Tyto rozhraní poskytují dostatečné množství metod pro komunikaci s dalšími zařízeními. Pro implementaci těchto rozhraní nad různými technologiemi je však nutné zavést další rozhraní. Téměř každý *přijímač* či *odesílatel* potřebuje pro svojí funkci ovladač zařízení, který mu poskytne metody pro komunikaci s hardwarem.

Rozhraní `IUsingDriver` definuje akcesory pro nasavení ovladače zařízení:

```
public interface IUsingDriver<T extends ICommunicationDriver> {
    public void setDriver (T val);
    public T getDriver ();
}
```

Metoda `setDriver()` nastavuje ovladač, který je použit pro zaslání zprávy. V tomto ovladači se nachází jednoduché API pro přístup k zařízením pro komunikaci (RS232, USB, Bluetooth).

Zde je uvedena ukázka implementace *odesílatele* využívající protokol HTTP.

```
/**
 * Tato třída implementuje rozhraní <code>ISender</code>
 * pro posílání zpráv pomocí bezstavového protokolu HTTP.
 *
 * @author Martin Obrátil
 */
public class HTTPSender
    extends AbstractUsingDriver<HTTPCommunicationDriver>
    implements ISender {

  private HTTPCommunicationDriver driver;
  /**
   * Vytvoří instanci <code>HTTPSender</code>.
   *
   * @param driver ovladač použitý pro komunikaci se zařízeními.
   */
  public HTTPSender(HTTPCommunicationDriver driver) {
    setDriver(driver);
  }
  /**
   * Pošle zprávu na server přes protokol HTTP. Při každém odeslání
   * je vždy navázáno nové spojení a odpově serveru je vypisována
   * na standardní výstup.
   */
  public void sendMessage(IMessage message) {
    try {
      ISentenceFactory sentenceBuilder =
       SentenceFactory.getSentenceBuilder(SentenceType.HTTP_POST);
      // Získání textové podoby HTTP dotazu
      String sentence = sentenceBuilder.getSentence(message);
      // Připojení k˜serveru
      this.getDriver().Connect();
      this.getDriver().getOutputStream()
          .write(sentence.getBytes("ISO-8859-1"));
      InputStream is = this.getDriver().getInputStream();
      int iZnak = -1;
```

```
      char znak;
      // Čtení Odpovědi serveru a vypisování na standartní výstup
      while (true) {
        iZnak = is.read();
        if (iZnak == -1) {
          break;
        }
        znak = (char) iZnak;
        System.out.print(znak);
      }
    } catch (IOException ex) {
      Logger.getLogger(HTTPSender.class.getName())
          .log(Level.SEVERE, null, ex);
    } finally {
      try {
        // Uvolnění prostředků
        this.getDriver().Disconnect();
      } catch (IOException ex) {
        Logger.getLogger(HTTPSender.class.getName())
            .log(Level.SEVERE, null, ex);
      }
    }
  }
}
```

Dále je zde uvedena ukázka implementace přijímače, který přijímá zprávy z GPS modulu.

```
public class GPSModuleReciever<T extends StreamCommunicationDriver>
        extends AbstractUsingDriver<T>
        implements IReciever {

  private ISentenceRecievedListener sentenceRecievedListener;
  private SentenceType sentenceType;
  private boolean listeningIsStopped;

  /**
```

```
 * Vytvoří instanci přijímače <code>GPSModuleReciever</code>.
 * @param sentenceType definuje typ věty, která je parsována a ze
 * které získán objekt LocationMessage.
 * @param driver ovladač použitý pro komunikaci se zařízeními.
 * @param listener listener listener který je spuštěn v˜momentě
 * přijetí zprávy.
 */
public GPSModuleReciever(SentenceType sentenceType, T driver,
    ISentenceRecievedListener listener) {
  this.sentenceType = sentenceType;
  this.setDriver(driver);
  this.sentenceRecievedListener = listener;
  this.listeningIsStopped = true;
}

public void setSentenceRecievedListener(
    ISentenceRecievedListener val) {
  this.sentenceRecievedListener = val;
}

synchronized public void StartListening() {
  this.listeningIsStopped = false;
  final T drv = this.getDriver();
  final ISentenceRecievedListener listener =
      this.sentenceRecievedListener;
  final SentenceType msgType = this.sentenceType;

  try {
    // Připojení k˜zařízení
    this.getDriver().Connect();
    Thread thread = new Thread(new Runnable() {

      public void run() {
        try {
          String sentence = "";
          // Získání parseru pro parsování proudu
```

```
            StreamParser parser =
                StreamParser.getStreamParsing(msgType);
            // Získání továrny na parsování NMEA zpráv
            IMessageFactory messageFactory =
                MessageFactory.getMessageBuilder(msgType);
            IMessage mess;
            // Čtení proudu
            while (!listeningIsStopped) {
              // InputStream je parsován a z~něho je získána NMEA věta
              sentence = parser.getSentence(drv.getInputStream());
              // Parsování NMEA věty a získání objektu typu IMessage
              mess = messageFactory.getMessage(sentence);
              // Zpráva je předána listeneru
              listener.SentenceRecieved(mess);
            }
          } catch (Exception ex) {
            Logger.getLogger(GPSModuleReciever.class.getName())
             .log(Level.SEVERE, null, ex);
          } finally {
            try {
              // Odpojení od zařízení - uvolnění prostředků
              drv.Disconnect();
            } catch (IOException ex) {
              Logger.getLogger(GPSModuleReciever.class.getName())
               .log(Level.SEVERE, null, ex);
            }
          }
        }
      });
      // Start vlákna pro čtení vstupního proudu NMEA zpráv
      thread.start();
    } catch (IOException ex) {
      Logger.getLogger(GPSModuleReciever.class.getName())
          .log(Level.SEVERE, null, ex);
    }
  }
```

```
  public void StopListening() {
    this.listeningIsStopped = true;
  }
}
```

4.4 Řešení přístupu ke komunikačním zařízením

Pro přístup ke komunikačním rozhraním je v aplikaci vyhrazen balík `cz.obratil.martin.observed.drivers`. Obsahuje rozhraní `ICommunicationDriver` definující akcesor pro přidání nastavení pro komunikaci. Jelikož jsou komunikační rozhraní (USB, RS232, Bluetooth) implementována za pomoci vstupních a výstupních proudů, usnadňuje to situaci tím, že komunikace se všemi těmito rozhraními bude dost podobná. Proto byla definována abstraktní třída `StreamCommunicationDriver`, která implementuje pro přístup k proudům, metodu `Disconnect()` pro uvolnění prostředků těchto proudů a metodu `Connect()` pro připojení zařízení. Tato metoda je abstraktní, protože k získání vstupních a výstupních proudů jsou použity různé knihovny a také se provádí různé nastavení.

Na ukázku je zde uveden implementovaný ovladač pro přístup k zařízení RS232.

```
public class RS232CommunicationDriver
    extends StreamCommunicationDriver<RS232Settings> {

  public RS232CommunicationDriver(RS232Settings settings) {
    this.settings = settings;
  }

  @Override
  public void Connect() throws IOException {
    Enumeration portList = CommPortIdentifier.getPortIdentifiers();
    CommPortIdentifier portId;
    boolean portFound = false;
    SerialPort serialPort = null;
    // Cyklus iteruje v˜seznamu dostupný RS232 portů.
    while (portList.hasMoreElements()) {
      portId = (CommPortIdentifier) portList.nextElement();
```

```
    // Hledá se seriový port.
    if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
      // Hledá se port s˜požadovaným identifikátorem
      if (portId.getName().equals(this.settings.getPort())) {
        System.out.println("Port nalezen: " +
            this.settings.getPort());
        portFound = true;
        try {
          // Otevření portu
          serialPort = (SerialPort) portId.open("SeriovyPort", 2000);
        } catch (PortInUseException e) {
          throw new IOException(e.getMessage() + "Port se používá!");
        }
        try {
          // Získání vstupního proudu.
          setInputStream(serialPort.getInputStream());
        } catch (IOException e) {
          throw new IOException(e.getMessage() +
              "Nepodařilo se získat vstupní proud.");
        }
        try {
          // Nastavení parametrů sériového portu
          serialPort.setSerialPortParams(this.settings.getBaudRate(),
                  this.settings.getDataBits(),
                  this.settings.getStopBits(),
                  this.settings.getParity());
        } catch (UnsupportedCommOperationException e) {
          System.out.println(e.getMessage() +
              "Nepodařilo se nastavit parametry sériového portu.");
        }
      }
    }
  }
  // Vyvolání výjimky typu IOException v˜případě,
  // že požadovaný port nebyl nalezen.
  if (!portFound) {
```

```
      throw new IOException("Port "
          + this.settings.getPort() + " nenalezen.");
    }
  }
}
```

Tento ovladač vyžaduje knihovnu pro komunikaci s portem RS232. Java SE ve svých knihovnách podporu pro sériovou linku RS232 neobsahuje, nicméně existuje implementace standardu Java Communications API [9] v podobě volně stažitelné knihovny RXTXcomm.jar [10].

Pro rozhraní Bluetooth ve standardních knihovnách Javy SE rovněž podpora není. Ovšem existuje knihovna implementující část API obsaženou v Java ME, která podporu pro Bluetooth obsahuje. Tato knihovna se jmenuje Bluecove [11] a je také možné ji stáhnout volně z internetu.

4.5 Komunikace prostřednictvím SMS

Komunikaci prostřednictvím zpráv SMS zajišťují třídy `SMSSender` a `SMSReciever`. SMS zprávy jsou přijímány a odesílány prostřednictvím mobilního telefonu připojeného přes různá komunikační zařízení (RS232, Bluetooth, USB). Třídy `SMSSender` i `SMSReciever` vyžadují, aby mobilní telefon podporoval komunikaci prostřednictvím AT příkazů [12] a práci v textovém módu (`AT+CMGF=1`). Konkrétně `SMSSender` vyžaduje podporu pro příkaz `AT+CMGS="telefonní číslo"`, který provede odeslání SMS zprávy na zadané telefonní číslo. `SMSReciever` si žádá podporu pro přkaz `AT+CNMI`, který zpřístupní notifikaci právě přijatých zpráv.[2]

4.6 Serializace souřadnic do SMS zpráv

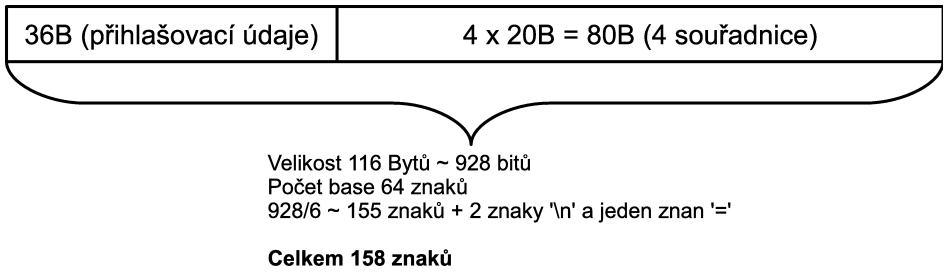
Každá odesílaná souřadnice v SMS zprávě obsahuje následující položky:

- Zeměpisná výška (float – 4B)
- Zeměpisná šířka (float – 4B)
- Výška nad mořem (float – 4B)

[2]Nutno dodat, že po zapnutí naslouchání pro nově příchozí zprávy se žádná přijatá zpráva nedostane do paměti telefonu. Veškerá komunikace je zpracována přes třídu SMSReciever a případné nevalidní zpravy jsou ignorovány a tudíž ztraceny.

- Čas (long – 8B)
- Uživatelské jméno (String – 10B)
- Heslo (String – 10B)
- Jméno zaznamenávané trasy (String – 16B)

Maximální délka SMS zprávy je u 7 bitové znakové sady celkem 160 znaků. Nejjednodušší způsob jakým souřadnice zakódovat, je použití standardu *Base64* [13], který obsahuje pouze *bezpečné znaky*. Strukturu takové SMS zprávy ukazuje obrázek 2.



Obr. 2. Struktura obsahu SMS zprávy

4.7 Použití knihovny Observed

Knihovna *Observed* byla navržena tak, aby její použití bylo co nejsnažší a aby případné vytváření nadstaveb (GUI) nad touto knihovnou bylo co nejjednodušší. Knihovna umožňuje ponechat úlohu složitého zkonstruování objektů přijímače a odesílatele továrním třídám. V knihovně *Observed* se nachází dvě abstraktní tovární třídy[3]: `RecieverFactory` a `SenderFactory`. Jsou obsaženy v balících `cz.obratil.martin.observed.recievers` a `cz.obratil.martin.observed.senders`.

Dále zde bude popsáno získání objektů *přijímače* a *odesílatele* a jejich vzájemné propojení.

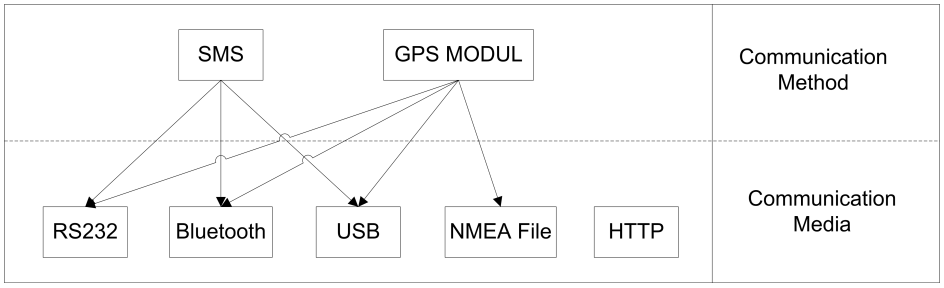
```
CommunicationMedium recCommunicationMedium = CommunicationMedium.RS232;
IRecieverFactory recBuilder =
    RecieverFactory.getRecieverBuilder(recCommunicationMedium);
CommunicationMedium sndCommunicationMedium = CommunicationMedium.HTTP;
ISenderFactory senderBuilder =
```

[3]Tovární třídy obsahují tovární metodu, která je předepsaným návrhovým vzorem. [14]

```
    SenderFactory.getSenderFactory(sndCommunicationMedium);

final ISender sender = senderBuilder.getSender();
final Vector<String> coorListModel = coordinateListModel;

reciever = recBuilder.getReciever(new ISentenceRecievedListener() {
  public void SentenceRecieved(IMessage message) {
    sender.sendMessage(message);
  }
});
reciever.StartListening();
```

Tovární třídy `RecieverFactory` a `SenderFactory` používají ve svých metodách objekty obsahující nastavení pro různé typy komunikací a komunikačních rozhraní. To jakou konkrétní tovární třídu metody `getSenderFactory()` a `getRecieverFactory()` zvolí, určuje parametr `communicationMedium` metod `getSender()` resp. `getReciever()`. Tento parametr je výčet typu `CommunicationMedium` a představuje typ komunikačního rozhraní – RS232, Bluetooth, USB, NMEA File, HTTP. V konkrétních továrních třídách se dále rozhoduje jaká instance *odesílatele* či *přijímače* bude vrácena při volání metody `getSender()` resp. `getReciever()`. To je určeno na základě atributu `communicationMethod`, který je výčet typu `CommunicationMethod` – GPS modul, SMS.

Pro lepší představu je zde uveden obrázek 3, který osvětlí vztahy mezi výčty `CommunicationMethod` a `CommunicationMedium`.



Obr. 3. Schéma závislosti mezi typem komunikace a typem komunikačního rozhraní

4.8 Uživatelská nadstavba Observed GUI nad knihovnou Observed

Pro snadné užívaní knihovny *Observed* byla vytvořena nadstavba, která poskytuje přehledné a intuitivní grafické uživatelské rozhraní (GUI).

Základním ovládacím prvkem aplikace je panel, který umožňuje přepínat mezi nastavením a monitorováním komunikace.

4.8.1 Nastavení

Nastavení se dělí na dvě části. V horní části se nastavuje *přijímač* a v dolní části *odesílatel*.

Přijímač

Možnosti přijímání souřadnic jsou následující:

- GPS modul
- Krátké textové zprávy (SMS)
- Soubor obsahující NMEA zprávy (vhodné pro testování)

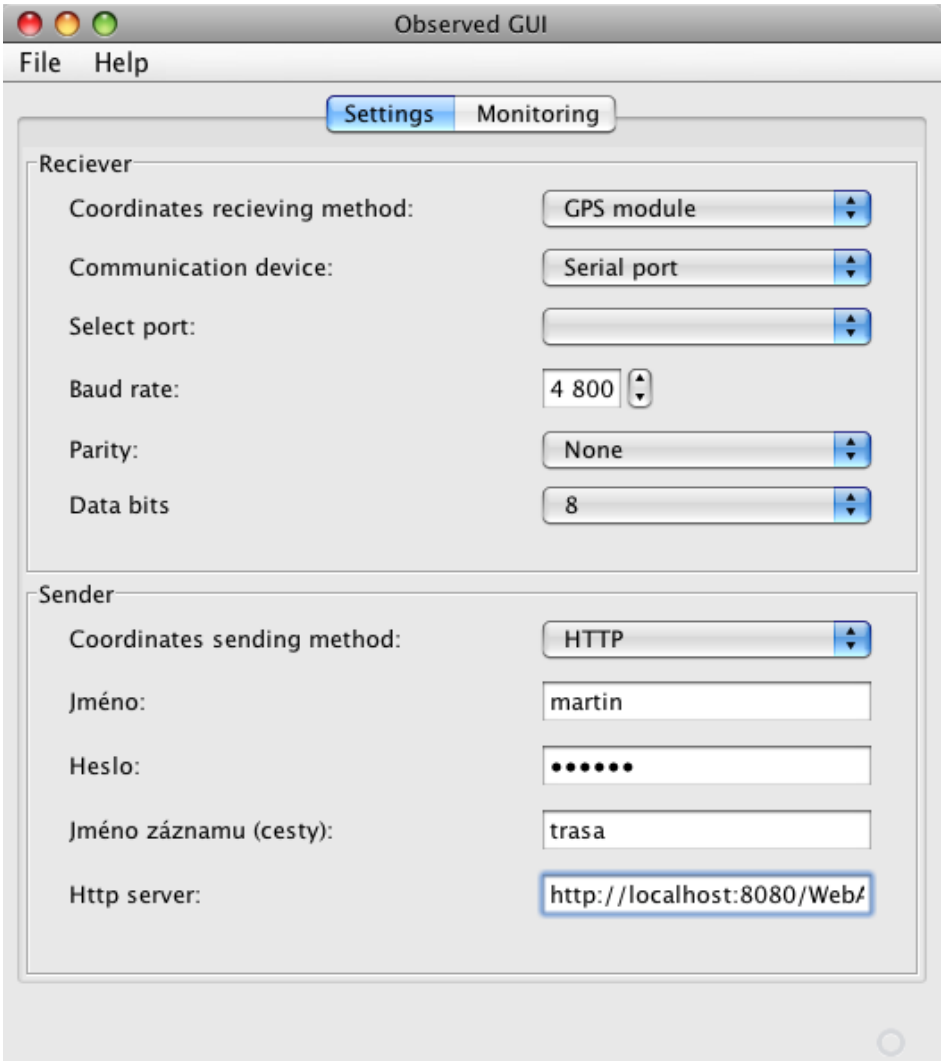
Při zvolení možnosti GPS modul se zpřístupní nabídka komunikačních rozhraní, které je možno využít pro připojení takového zařízení. Společně s volbou tohoto rozhraní je nabízena řada parametrů, které lze nastavovat dle libosti, jak si žádá modul GPS. Příjem zpráv přes SMS zpřístupní stejnou nabídku jako předešlá možnost. Rovněž je tu možnost vybrat na jaké rozhraní bude mobilní telefon připojen a s jakým nastavením. Varianta přijímání souřadnic přes NMEA soubor zpřístupní pouze textové pole, kam se zadává cesta k takovému souboru.

Odesílatel

Možnosti odesílání souřadnic jsou celkem dvě:

- Krátké textové zprávy (SMS)
- Pomocí HTTP přímo na server

Každá varianta má možnost nastavit přihlašovací údaje (uživatelské jméno, heslo a jméno zaznamenávané cesty)[4]. Varianta odesílání souřadnic pomocí SMS zpřístupňuje stejnou nabídku jako v panelu přijímače zpráv pomocí SMS, ale navíc ještě umožňuje nastavit telefonní číslo mobilního telefonu, na který budou zprávy odesílány. Nejdůležitější je pak varianta odesílání souřadnic přes protokol HTTP. To je jediná konečná fáze jak dostat souřadnice na server. Serverová část neobsahuje žádný *přijímač*, ale obsahuje kontroler, který přes metodu POST přijme souřadnice a dle zadaných přihlašovacích údajů souřadnice roztřídí a zapíše do databáze. Při zvolení této varianty už jen stačí zadat URL adresu tohoto kontroleru.

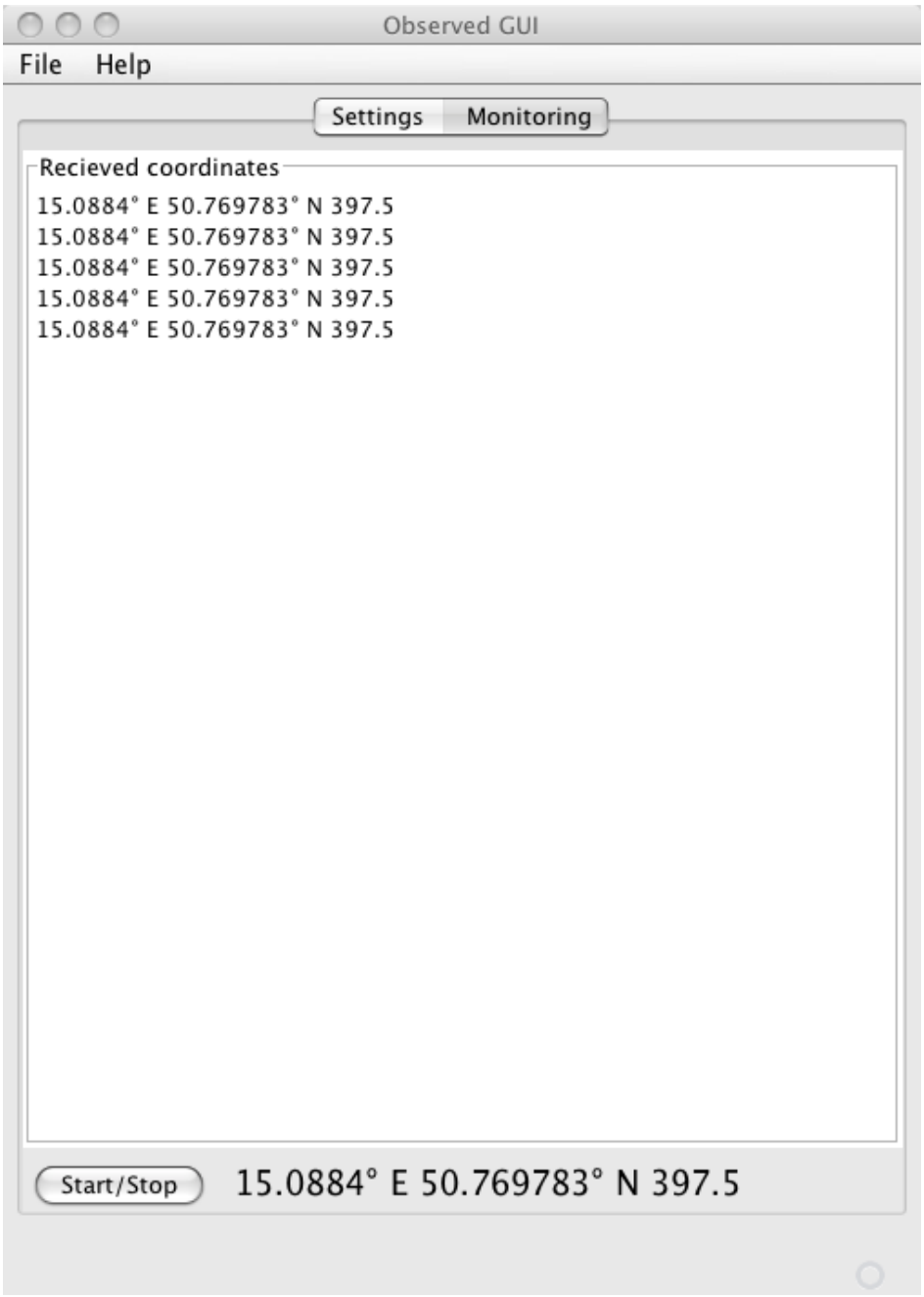


Obr. 4. Panel nastavení aplikace *Observed GUI*

[4]Tyto údaje jsou nezbytné pro správné třídění souřadnic na straně serveru, protože každý uživatel má svůj vlastní profil, kde se zaznamenávají jeho trasy.

4.8.2 Monitor komunikace

Monitor komunikace se v aplikaci vyvolává přepnutím záložky v horní části. Tento monitor obsahuje seznam přijatých souřadnic. Po kliknutí na tlačítko *Start/Stop* se aktivuje přijímání a odesílání souřadnic. Každá přijatá souřadnice je zároveň automaticky odeslána a zobrazena v seznamu.



Obr. 5. Monitor komunikace aplikace *Observed GUI*

5 Serverová aplikace Observer

Tato kapitola obsahuje návrh a implementaci serverové aplikace zvané *Observer*, která obstarává třídění a uchování souřadnic poslaných aplikacemi využívající knihovnu *Observed*. V následujících podkapitolách je kompletně rozebrána funkce této aplikace a způsob jakým je aplikace implementována.

5.1 Definice požadavků

Serverová aplikace *Observer* má dvě úlohy.

- Zaznamenávání a uchování geografických souřadnic.
- Zpřístupňovat webové rozhraní s vizualizací souřadnic resp. trasy na mapovém podkladu prostřednictvím Atlas AMapy API.

Požadavek je stanoven na multiplatformitu pro umožnění co největšího výběru operačních systému, na kterých bude aplikace provozována.

5.2 Návrh aplikace Observer

Pro serverovou aplikaci byl zvolen programovací jazyk Java s běhovým prostředím ve verzi Java Enterprise Edition. Velmi důležitým aspektem návrhu je rovněž volba správného aplikačního *frameworku*[5]. Velmi vhodným řešením pro tento projekt je Spring framework [15, 16].

5.2.1 Spring framework

Spring je open source aplikační *framework* pro platformu Java, který tvoří doplněk k platformě Java Enterprise Edition. Spring se skládá ze sady menších *frameworků* použitelných nezávisle na sobě, ale jejich společné použití zajistí přidanou funkcionalitu. Níže jsou uvedeny základní moduly ze kterých se Spring framework skládá.

Inversion of Control (*IoC*) kontejner [16] – *IoC* je návrhový vzor popisující možnosti inicializace navzájem závislých objektů mimo kód těchto objektů. Tuto práci zajistí Spring z vnějšku za pomoci IoC kontejneru, přičemž injekce těchto závislostí se provádí buď konstruktorem, nebo skrze setter. Tato technika se nazývá

[5]Framework je anglický výraz pro označení sady softwarových komponent, podpůrných programů či návrhových vzorů umožňující snažší vývoj softwaru.

Dependency Injection. Základem tohoto modulu je rozhraní `BeanFactory`, které má na starosti životní cyklus objektů založených na POJO [17] konvenci.

Model-view-controller (MVC) framework [18] – Model-view-controller je aplikační architektura, která odděluje datový model, řídící logiku a vzhled aplikace do tří na sobě nezávislých komponent. Tato aplikační architektura se při vývoji webových aplikací velice často využívá a Spring tuto architekturu implementuje jako jeden ze svých modulů.

Aspect-oriented programming framework [19] – Zpřístupňuje funkcionalitu, která umožňuje uplatnit programovací techniky přesahující rámec objektově orientovaného programovaní.

Data access framework – Modul zahrnující funkcionalitu pro přístup k databázím. Pracuje s databázovými systémy pomocí nastrojů JDBC a ORM (objektově relační mapování).

Transaction management framework – Poskytuje jednotný programovací model pro správu transakcí nad mnoha různorodými transakčními aplikačními rozhraními jako například JDBC, JPA, JTA, JDO, atd.

Remote Access framework – Tento modul poskytuje abstrakci nad mnoha aplikačními rozhraními poskytující vzdálenou komunikace založenou na RPC (Remote Procedure Calling). Do podporovaných protokolů spadá RMI, CORBA a protokoly založené na HTTP včetně webových služeb, které jsou postaveny nad protokolem SOAP.

Testing framework – Zahrnuje třídy pro vytváření jednotkových a integračních testů.

Ke Spring frameworku se váží další přidružené projekty, které zpřístupňují vývojářům další funkcionalitu nad výše uvedenou sadou modulů. Pro účely této práce byl nejdůležitější přídavný modul Spring Web Flow [20].

Spring Web Flow (SWF) – Spring Web Flow umožňuje modelovat komunikaci mezi uživatelem a webovou aplikací do znovupoužitelných modulů zvaných *flows* (toky). Každý *flow* se skládá z množství stavů zahrnující určité chování či rozhodovací logiku. Tyto stavy se odkazují na řídící logiku spouštěnou při přechodu do těchto stavů. *Flow* může obsahovat následující typy stavů:

Start state – Tento stav je obsažen v každém *flow* a odkazuje se na řídící logiku spouštěnou při startu flow.

Action state – Stav definující reakce na uživatelské podněty. Např. odeslání formuláře zpět na server.

View state – Používá se k zobrazení pohledu, tedy k vygenerování a odeslání webové stránky uživateli.

Decision state – Obsahuje podmínku pro přechod o dalšího stavu.

Subflow state – V tomto stavu se spouští jiný *flow*.

End state – Zde je ukončen současný *flow*.

5.2.2 Struktura webové aplikace Observer

Webová aplikace obsahuje mnoho souborů, které dohromady vytvářejí archív s určitou strukturou. Tato struktura obsahuje kód aplikace, konfigurační soubory XML, soubory definující šablonu webových stránek JSP, jejich vzhled CSS souborů a knihovny potřebné pro běh aplikace. Níže je uvedena struktura této webové aplikace.

```
adresář aplikace
    | - - css
    | - - WEB-INF
    |        |
    |        | - - classes
    |        | - - flows
    |        | - - jsp
    |        | - - lib
    |        | - - applicationContext.xml
    |        | - - dispatcher-servlet.xml
    |        | - - web.xml
    |
    | - - META-INF
    |        |
    |        | - - persistence-BCK.xml
    |        | - - MANIFEST.MF
```

web.xml – Obsahuje základní konfiguraci webové apikace. Ta zahrnuje konfiguraci servletů, filtrů, listenerů a vstupních stránek do aplikace tzv. welcome file.

applicationContext.xml – Odkaz na tento soubor je uveden v souboru web.xml:

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

V momentě kdy je aplikace *Observer* nastartována, objekt `ContextLoaderListener` načte soubor `applicationContext.xml` s konfigurací aplikace, který Spring framework použije k nakonfigurování tzv. *bean* objektů. Tento soubor obsahuje konfiguraci *bean* objektů `pathRepo`, `userRepo` a `coordinateRepo` tvořících tzv. DAO vrstvu aplikace. Tato konfigurace vyžaduje konfiguraci dalších *bean* objektů. Jednou z důležitých je *bean* objekt `entityManagerFactory`, který používá DAO vrstva. `entityManagerFactory` je továrnou na objekty typu `EntityManger`. `EntityManager` (neboli manažer entit) spravuje instance entit. Umožňuje instance mazat, vytvářet a provádět nad nimi dotazy. Manažer entit je samozřejmě nutné spravně nakonfigurovat a to rovněž pomocí *bean* objektů. V našem případě to jsou dva *bean* objekty. Jeden s názvem `dataSource`, která je instancí třídy `DriverManagerDataSource`. A druhá která nese název `jpaVendorAdapter` a je instancí třídy `HibernateJpaVendorAdapter`. *Bean* objekt `dataSource` představuje konfiguraci přístupu do databáze. Obsahuje třídu pro ovladač databáze, URL pro její adresu a také přihlašovací údaje.

```
<bean id="dataSource"
  class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <property name="url" value="jdbc:mysql://192.168.79.128/observer" />
  <property name="username" value="observer" />
  <property name="password" value="xxxxxxxxx" />
</bean>
```

Druhý *bean* objekt `jpaVendorAdapter` definuje jaký bude použit *framework* pro perzistenci dat. Jedním z nejověřenějších v této oblasti je open source *framework* Hibernate, který pro potřeby tohoto projektu postačuje. Konfigurace tedy vypadá následovně:

```
<property name="jpaVendorAdapter">
```

```
  <bean
    class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <property name="database" value="MYSQL" />
  </bean>
</property>
```

Aplikační rozhraní Hibernatu však stejně není přímo použito, protože s příchodem J2EE 5.0 je perzistence dat na platformě Java sjednoceno do jednotného rozhraní s názvem Java Persistent API [21].

dispatcher-servlet.xml – Tento soubor má na starosti konfiguraci servletu s názvem *dispatcher*. Úloha tohoto servletu je specifikována ve webovém konfiguračním souboru `web.xml`.

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

Jedná se o tzv. přední kontroler (front controller) což je návrhový vzor platformy J2EE. Má na starosti zpracování veškerých dotazů jdoucích od uživatele na server. V tomto případě je však *dispatcher* omezen pouze na požadavky odpovídající vzoru `"*.htm"`. Soubor `dispatcher-servlet.xml` dále obsahuje informace, na základě kterých DispatcherServlet předává uživatelovi požadavky konkrétním kontrolerům.

Soubory *-flow.xml – V těchto souborech jsou definovány moduly *flow* pro *framework* Spring Web Flow. Jak bylo uvedeno v kapitolách výše, moduly *flow* se skládají z množství stavů a tyto stavy jsou specifikovány právě v těchto souborech. Níže je uvedena část souboru `path-flow.xml` reflektující komunikaci mezi uživatelem a serverem v momentě, kdy je uživatel již přihlášen na serveru.

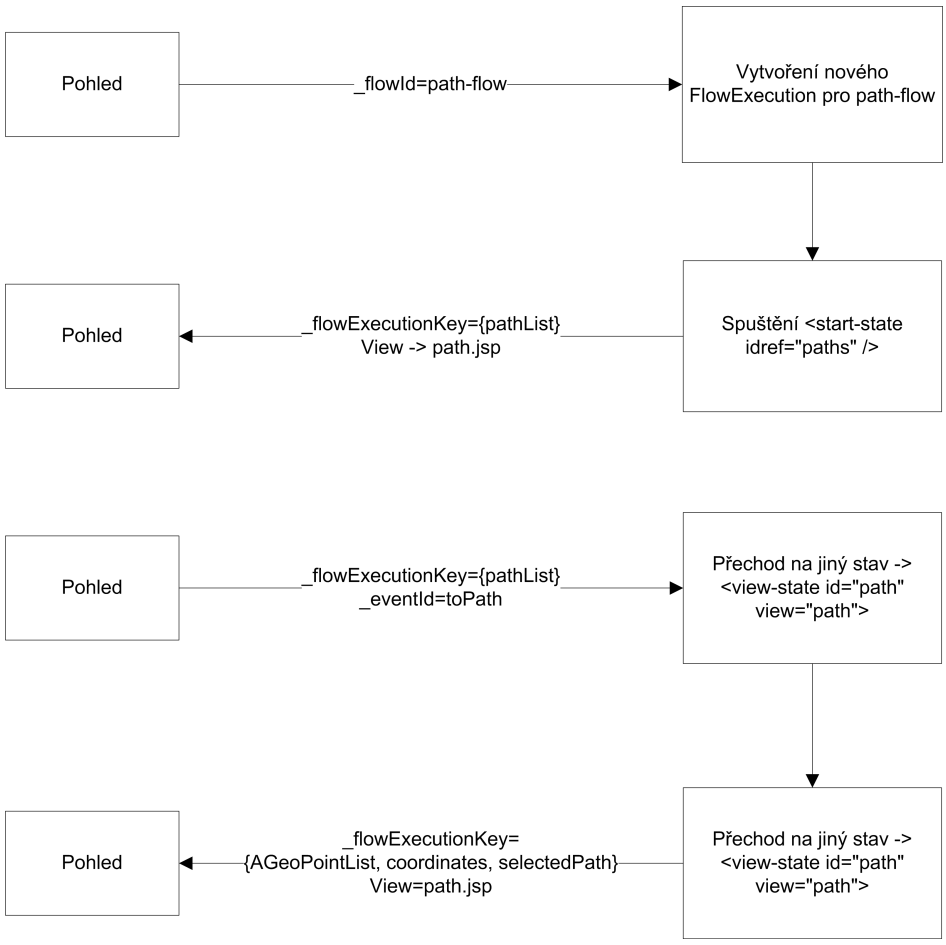
```
<start-state idref="paths" />

<view-state id="paths" view="paths">
  <render-actions>
    <action bean="pathsAction" method="setupForm"/>
  </render-actions>
  <transition on="toPath" to="path">
    <action bean="pathsAction" method="toPathAction" />
  </transition>
</view-state>

<view-state id="path" view="path">
  <render-actions>
    <action bean="pathAction" method="setupForm"/>
  </render-actions>
</view-state>

<import resource="paths-flow-beans.xml" />
```

Fukncionalitu tohoto modulu *flow* popisuje obrázek 6. Aplikace obsahuje celkem tři moduly *flow*, které jsou znázorněny na obrázku 7.

5.2.3 Model, perzistence dat a databáze

Model aplikace se sestává z množství navzájem provázaných datových entit, které jsou implementovány jako tzv. POJO objekty. POJO (Plain Old Java Objects) je zkrácený název pro obyčejné objekty v jazyce Java. Na těchto objektech není nic zvláštního. Jsou to, přesně jak název říká, obyčejné Java objekty, které obsahují metody a vlastnosti (gettery a settery), přičemž každý takový objekt reprezentuje datovou entitu uvnitř aplikace. POJO objekty poskytují programový přístup k těmto entitám, resp. k jejich vzájemným vazbám.

V kódu POJO objektů jsou vazby realizovány pomocí tzv. vnitřních kompoment (component containment) uvnitř objektů. Následující část kódu entity `User` ukazuje vazbu mezi entitou `User` a `Path` – „uživatel může zaznamenat mnoho tras“.

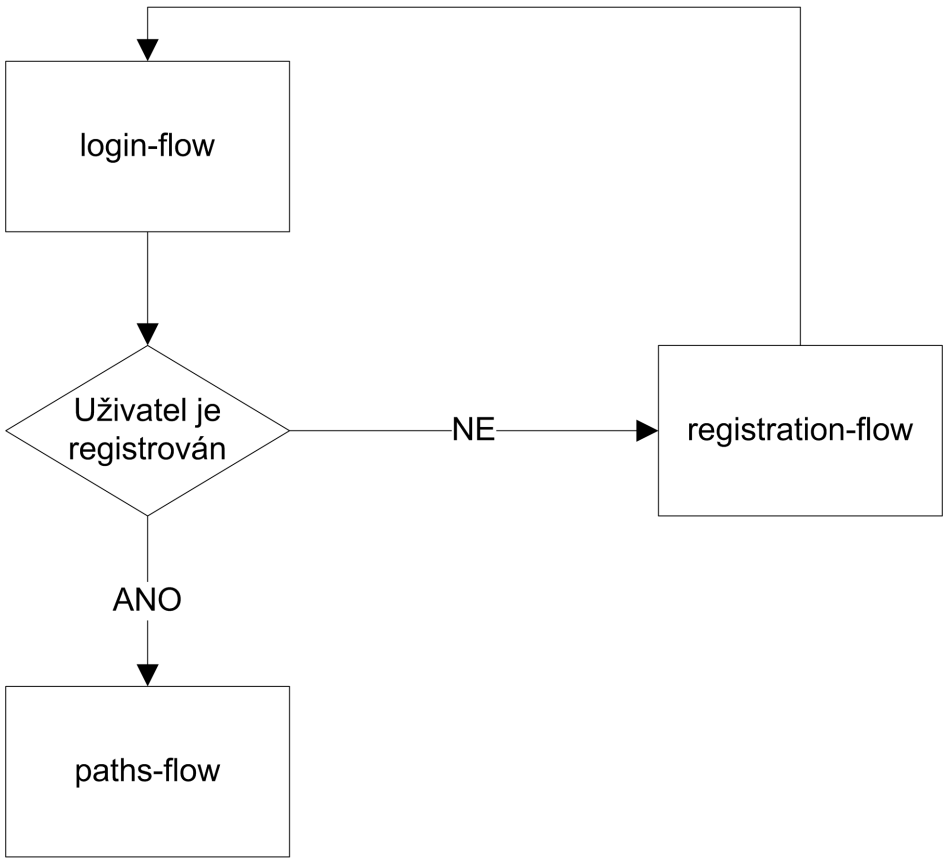


Obr. 6. Schéma popisující komunikaci v paths-flow.

```
@Entity
. . .
public class User implements Serializable {
  . . .
  @OneToMany(fetch=FetchType.LAZY,cascade
      = CascadeType.ALL,mappedBy="user")
  private List<Path> paths = new ArrayList<Path>();
  . . .
```

Naproti tomu entita `Path` má opačnou vazbu – „trasa patří uživatelovi“. Dále je v této entitě vazba na entitu `Coordinate` – „trasa je složena z mnoha souřadnic“.

```
@Entity
. . .
public class Path implements Serializable {
```



Obr. 7. Moduly flow a přechody mezi nimi.

```
. . .
@ManyToOne(cascade = CascadeType.REFRESH, fetch = FetchType.EAGER)
private User user;
@OneToMany(cascade = CascadeType.ALL, fetch =
    FetchType.LAZY, mappedBy = "path")
private List<Coordinate> coordinates = new ArrayList<Coordinate>();
. . .
```

Nakonec entita `Coordinate` vytváří vazbu na entitu `Path` – „souřadnice je součást trasy“.

```
@Entity
. . .
public class Coordinate implements Serializable {
  . . .
  @ManyToOne(cascade = CascadeType.REFRESH, fetch = FetchType.EAGER)
  private Path path;
```

. . .

Nyní jsou vazby mezi entitami zřejmé a už jen zbývá definovat vlastní datové položky entit.

User

- id (int)
- userName (String)
- password (String)

Path

- id (int)
- name (String)

Coordinate

- id (int)
- longitude (float)
- latitude (float)
- altitude (float)
- time (long)

Třídy modelu aplikace se nacházejí v balíku `cz.obratil.martin.bakule.model`. Všechna data aplikace (uživatelé, trasy, souřadnice) jsou uložena v databázi a pro tyto potřeby byl zvolen relační databázový systém (RDBMS) MySQL. Tento databázový systém naprosto vyhovuje potřebám této aplikace, jeho nasazení v oblasti webových aplikací je naprosto běžné. MySQL má na platformě Java podporu v podobě ovladače pro knihovnu JDBC (Java Database Connectivity), která poskytuje metody pro dotazování a upload dat do databáze a unifikuje tak přístup k databázím pomocí procedurálního API.

Pro přístup do databáze v kódu aplikace byly zvažovány následující způsoby. Mezi nejjednodušší patří přímý přístup pomocí knihovny JDBC. Tento způsob však vyžaduje

použití komplikovanějšího procedurální API, který však dostatečně neunifikuje přístup k jakémukoliv datovému úložišti (nejen relačnímu). Jako vhodnější varianta se nabídlo použít jeden ze zavedených objektově relačních mapovacích *frameworků* (ORM).

ORM *frameworků* pro platformu Java existuje celá řada, nicméně volba konkrétního produktu není příliš důležitá. A to proto, že samotná platforma J2EE poskytuje jednotné rozhraní pro přístup k ORM *frameworkům*. Nutno poznamenat, že samotný ORM *framework* v ní obsažen není. Tomuto rozhraní se říka Java Persistence API (JPA) a sestává se ze tří oblastí:

- Balík `javax.persistence` kde se nachází programové rozhraní a knihovny JPA.
- Unifikovaný dotazovací jazyk Java Persistence Query Language (JPQL). Dotazy jsou v aplikaci deklarovány prostřednictvím anotace `@NamedQueries` ve třídách entit. Tyto dotazy jsou následně z JPQL vnitřně překládány do nativního dotazovacího jazyka databáze MySQL.
- Metadata specifikující vazby mezi entitami. Tato metadata jsou v aplikaci vedena prostřednictvím anotací přímo ve třídách entit.

Perzistenci dat v aplikaci zprosředkovávají třídy balíku `cz.obratil.martin.bakule.dao`, které jsou navrženy dle návrhového vzoru DAO (Data Acces Object) a tvoří tzv. DAO vrstvu. DAO je jedním z návrhových vzorů platformy J2EE. Vrstva DAO je abstraktním rozhraním oddělující kód aplikace vyžadující přistup k datům od kódu, který přístup k datům implementuje. Tento vzor zvyšuje flexibilitu aplikace tím, že umožňuje volbu datových zdrojů až ve fázi *deploymentu* aplikace. Navíc datový zdroj není limitován pouze na databázi. Existuje možnost zvolit jiný datový zdroj, třeba XML.

5.2.4 Řadič

Další důležitou částí aplikace je řídící logika. Ta je obsažena v balících `cz.obratil.martin.bakule.action` a `cz.obratil.martin.bakule.controller`. První z výše jmenovaných balíků obsahuje logiku pro ovládání webové aplikace. Tj. přihlášení, registrace, zobrazení seznamu tras a zobrazení trasy. Třídy tohoto balíku jsou potomky třídy FormAction, která je součástí *frameworku* Spring Web Flow. Druhý balík obsahuje pouze jednu třídu `AddingCoordinateController` což je kontroler pro příjem a uložení souřadnic.

5.2.5 Přístup k mapovým podkladům a vykreslení trasy

Internet dnes poskytuje přístup k mnoha službám a jednou z nich je i přístup k mapovým podkladům. Tyto podklady na internetu nabízí mnoho firem. V mezinárodním měřítku mezi nejznámější můžeme zařadit Google Maps, Microsoft Live Maps a Yahoo! Local Maps. Na českém internetu jsou to služby Atlas AMapy a Mapy.cz. Ke všem zmíněným službám je k dispozici API pro programovací jazyk JavaScript, tudíž je nabízena možnost tyto mapy využívat ve vlastních webových aplikacích.

Aplikace *Observer* využívá službu Atlas AMapy a její API. Toto API poskytuje celou řadu předpřipravených tříd a metod pro vývoj mapových aplikací. Následující ukázka osvětlí jednoduchý příklad vykreslení vektorové trasy pomocí AMapy API. Ukazuje spojení vzdušnou čarou mezi městy Liberec a Hradec Králové.

```
var Page = {
  load: function() {
    var mainMap = new AMap("mainmap");
    mainMap.loadMaps();
    mainMap.addMapPart(new AMapControl());
    mainMap.addMapPart(new AMapTypeControl());

    var points = [];
    var point;
    var geoPoint;
    // Geografické souřadnice Hradce Králové
    geoPoint = "50 12’32.24\"N, 15 50’4.57\"E";
    point = new AGeoPoint(geoPoint);
    mainMap.addOverlay(new AMarker(point,{
      icon:new AIcon({imageSrc:"img/point.gif"}),
      draggable:false
    }));
    points.push(point);
    // Geografické souřadnice Liberce
    geoPoint = "50 45’32.21\"N, 15 3’49.59\"E";
    point = new AGeoPoint(geoPoint);
    // Definování vzhledu bodu
    mainMap.addOverlay(new AMarker(point,{
      icon:new AIcon({imageSrc:"img/point.gif"}),
```

```
      draggable:false
    }));
    points.push(point);
    // Výroba trasy z~pole bodů
    var trace = new APolyline(points, {
      color: '#427CC4',
      weight: '7px',
      opacity: 0.55
    });
    // Přidání trasy na mapu
    mainMap.addOverlay(trace);
    // Nastavení zoomu a pohledu na střed trasy
    mainMap.setBestZoomAndCenter(points);
  }
}
window.addEvent('domready', Page.load.bind(Page));
```

5.2.6 Prohlížení zaznamenaných tras (souřadnic)

Funkce prohlížení zaznamenaných tras je možná pomocí běžného webového prohlížeče. Na hlavní stránce webové aplikace *Observer* je k dispozici přihlašovací dialog do aplikace, kde se po zadání přihlašovacích údajů zobrazí seznam zaznamenaných tras. Na položky v tomto seznamu je možno klikat jako na hypertextové odkazy, které vedou na konkrétní zobrazení zaznamenané trasy. Zaznamenaná trasa je vykreslena na mapovém podkladě pomocí Atlas AMapy API. Tento mapový podklad je samozřejmě možno zvětšovat a zmenšovat kolečkem myši a navigovat se v něm pomocí funkce táhni a pusť. Na mapě je samotná trasa zakreslena blědě modrou čarou. Nad mapou je rovněž zobrazen celý seznam zaznamenaných souřadnic.

5.2.7 Upload souřadnic na server

Příjem souřadnic serverem je realizován jednoduchým kontrolerem s názvem AddingCoordinateController. Tento kontroler je přístupný přes URL http://[adresa serveru]/Observer/addingCoordinate.htm. Klientská aplikace Observed tento kontroler využívá tím, že mu přes protokol POST posílá jednotlivé souřadnice. AddingCoordinateController ovšem také umožňuje přijímat souřadnice přes metodu GET a to je možné demonstrovat i v samotném webovém prohlížeči.

Souřadnice měst Prahy a Liberce jsou následující:

Praha 50° 4’ 40,54” S.Š. a 14° 24’ 33,85” V.D.

Liberec 50° 46’ 1,68” S.Š. a 15° 2’ 55,44” V.D.

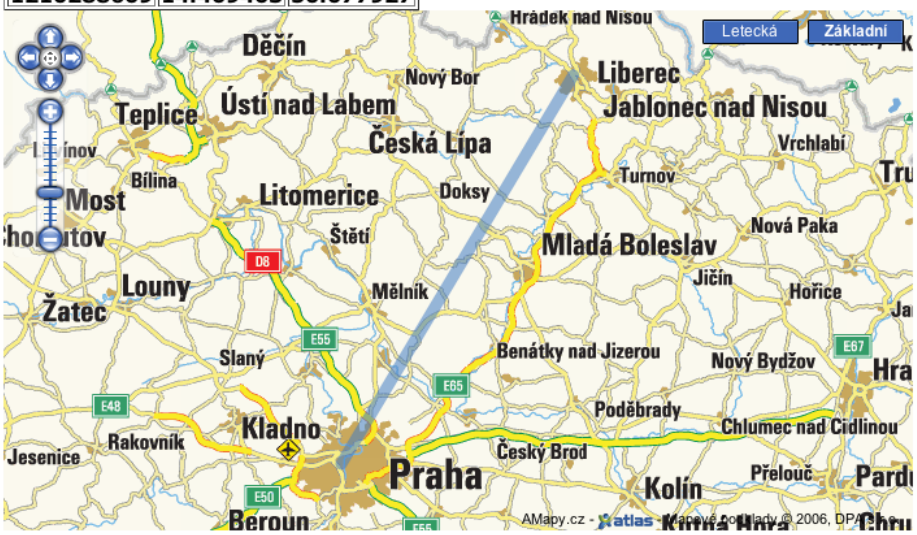
Předpokládejme, že na serveru je vytvořený účet pod uživatelským jménem zkouska a heslem zkouskaHeslo. Jméno trasy nazveme „PrahaLiberec“. Tyto souřadnice je možné odeslat na server zadáním následujících řetězců do adresního řádku ve webovém prohlížeči.

```
http://[adresa serveru]/Observer/addingCoordinate.htm
?latitude=50.077928&longitude=14.409403&altitude=0&time=1210288609
&pathName=PrahaLiberec&userName=zkouska&password=zkouskaHeslo
```

```
http://[adresa serveru]/Observer/addingCoordinate.htm
?latitude=50.767133&longitude=15.048733&altitude=0&time=1210288610
&pathName=PrahaLiberec&userName=zkouska&password=zkouskaHeslo
```

PrahaLiberec

čas	délka	šířka
1210288610	15.048733	50.76713
1210288609	14.409403	50.077927
1210288609	14.409403	50.077927



Obr. 8. Praha – Liberec

Obrázek 9 ukazuje podrobně zaznamenanou trasu mezi kolejemi na Harcově a Technickou univerzitou v Liberci.

Obr. 9. Trasa mezi kolejemi na Harcově a Technickou univerzitou v Liberci

6 Závěr

Výsledkem bakalářské práce jsou dvě aplikace. První aplikaci s názvem *Observed GUI* ponese sledovaný objekt a zprostředkuje odesílání aktuálních geografických souřadnic druhé aplikaci s názvem *Observer*. Ta souřadnice uloží ve své databázi, přičemž její webová část je schopna v případě potřeby tyto souřadnice prostřednictvím webového rozhraní vizualizovat na mapovém podkladě ve formě trasy.

Práce ukázala, že je možné odesílat geografické souřadnice z jakýchkoliv míst, kam dosahuje pokrytí signálem sítě GSM. Tedy sledování polohy objektu může být velice flexibilní. Jako velmi výhodné se ukázalo použití internetové služby Atlas AMapy, díky které bylo možné zdarma použít mapové podklady.

Klientská aplikace *Observed GUI* (pozorovaný) umí odesílat souřadnice buď pomocí zpráv SMS nebo skrze protokol HTTP. Aplikace je kompatibilní s GPS moduly připojenými přes komunikační rozhraní Bluetooth a RS232. Jelikož práce zahrnuje podrobně zdokumentované aplikační rozhraní, je tedy velice snadné podporu pro další rozhraní přidat. Dále aplikace podporuje přijímání a odesílání souřadnic prostřednictvím SMS zpráv, což umožní aplikaci provozovat v mobilním prostředí, kde není dostupná síť Internet.

Obě aplikace se podařilo napsat multiplatformně v technologiích J2SE a J2EE, tudíž jejich provoz je možný na většině dnes aktuálních operačních systémech.

Následující odstavce shrnují možné oblasti, které je zapotřebí v budoucnu vylepšit, aby produkt lépe sloužil danému účelu.

6.1 Redukce nákladů na komunikaci

Aplikace *Observed GUI* zatím nemá žádný algoritmus, který by určoval kdy je skutečně nutné zprávu s geografickou polohou odeslat. V současnosti se odesílání provádí kdykoliv GPS modul poskytne aktuální souřadnici, tedy každou sekundu. Cena jedné SMS zprávy se pohybuje v rozmezí od 1 do 2 Kč, což může být při delším sledování objektu velice nákladné. Algoritmus by mohl zajistit, aby se souřadnice posílaly pouze tehdy, pokud by se objekt pohyboval. I tak tato optimalizace pravděpodobně nestačí a pro větší redukci nákladů bude zapotřebí použít další úrovně optimalizace, např. optimalizace na bázi sledování změny vektoru rychlosti.

6.2 Problém s nedostupnosti signálu GPS

Dalším problémem je skutečnost, že signál z GPS družic se těžko dostává do interiérů a městských zástaveb. Ovšem existuje tu alespoň částečné řešení na bázi kombinace GPS s dalšími metodami určování polohy, např. pomocí akcelerometru.

6.3 Portování aplikace na mobilní telefony

V budoucnu je rovněž možné knihovnu *Observed* modifikovat pro použití na mobilním telefonu. Modifikace nemusí bý nijak rozsáhlé. V aplikačním rozhraní je nutné napsat jiné ovladače komunikačních zařízení a objekty realizující samotné odesílání dat. Na novějších zařízeních je situace ještě jednodušší, protože disponují již standardizovaným rozhraním zvaným Location API, které se samo postará o komunikaci s GPS moduly.

SEZNAM POUŽITÉ LITERATURY

[1] Richard S. Hall, Dennis Heimbigner, and Alexander L. Wolf. *Requirements for Software Deployment Languages and Schema*, 2008. Dostupné z: `http://www.doc.ic.ac.uk/~alw/doc/papers/scm8b.pdf`.

[2] Ing. Martin Šunkevič Doc. Ing. Jan Kolář, CSc. *Globální družicový navigační systém Galileo*. Česká kosmická kancelář, o.p.s., Praha, 2007.

[3] SKYHOOK Wireless. *SKYHOOK Wireless*, 2008. Dostupné z: `http://www.skyhookwireless.com/`.

[4] O2 — Služby - Celý ceník. *Vodafone - Tarify a ceny*, 2008. Dostupné z: `http://www.vodafone.cz/consumer/tariff/index.htm`.

[5] Telefónica O2 Czech Republic, a.s. *Vodafone - Tarify a ceny*, 2008. Dostupné z: `http://www.cz.o2.com/osobni/cz/sluzby/cenik/gsm/cely_cenik/index.html`.

[6] T-Mobile Czech Republic a.s. *Tarify a ceny - T-Mobile*, 2008. Dostupné z: `http://t-mobile.cz/Web/Residential/Tarify-a-ceny/Default.aspx`.

[7] Dale DePriest. *NMEA data*, 2008. Dostupné z: `http://gpsinformation.org/dale/nmea.htm`.

[8] Jan Martínek. *GPS a komunikańćní protokol NMEA*, 2008. Dostupné z: `http://www.abclinuxu.cz/serialy/gps-a-komunikacni-protokol-nmea`.

[9] Sun Microsystems, Inc. *Java Communications API*, 2008. Dostupné z: `http://java.sun.com/products/javacomm/`.

[10] Keane Jarvi. *RXTX: The Prescription for Transmission*, 2008. Dostupné z: `http://users.frii.com/jarvi/rxtx/`.

[11] Google. *Bluecove - Google Code*, 2008. Dostupné z: `http://code.google.com/p/bluecove/`.

[12] *AT Commands - Forum Nokia Wiki*, 2008. Dostupné z: `http://wiki.forum.nokia.com/index.php/AT_Commands`.

[13] S. Josefsson. *RFC 3548 (rfc3548) - The Base16, Base32, and Base64 Data Encodings*, 2008. Dostupné z: `http://www.faqs.org/rfcs/rfc3548.html`.

[14] Rudolf Pecinovský. *Návrhové vzory*. Computer Press, Brno, 2007.

[15] SpringSource. *Spring Framework*, 2008. Dostupné z: `http://www.springframework.org/`.

[16] Thomas Van de Velde, Bruce Snyder, Christian Dupuis, Sing Li, Anne Horton, and Naveen Balani. *Beginning Spring Framework 2*. Wiley Publishing, Inc, Indianapolis, Indiana, 2008.

[17] *Plain Old Java Object - Wikipedia, the free encyclopedia*, 2008. Dostupné z: `http://en.wikipedia.org/wiki/POJO`.

[18] *Model-view-controller - Wikipedia, the free encyclopedia*, 2008. Dostupné z: `http://en.wikipedia.org/wiki/Model-view-controller`.

[19] *Aspect-oriented programming - Wikipedia, the free encyclopedia*, 2008. Dostupné z: `http://en.wikipedia.org/wiki/Aspect-oriented_programming`.

[20] Keith Donald, Erwin Vervaet, and Ross Stoyanchev. *Spring Web Flow - Reference Documentation*, 2008. Dostupné z: `http://static.springframework.org/spring-webflow/docs/current/reference/index.html`.

[21] Sun Microsystems, Inc. *Java Persistence API*, 2008. Dostupné z: `http://java.sun.com/javaee/technologies/persistence.jsp`.

[22] Ivor Horton. *Java 5*. Neocortex, spol. s r. o., Praha, 2005.

[23] Sun Microsystems, Inc. *JavaTM 2 Platform Standard Edition 5.0 API Specification*, 2008. Dostupné z: `http://java.sun.com/j2se/1.5.0/docs/api/`.