Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

TÉRKÉPÉSZETI ADATOK GYŰJTÉSE ÉS FELDOLGOZÁSA

TDK dolgozat

KONZULENS

Imre Gábor

BUDAPEST, 2011

Tartalomjegyzék

Összefoglaló

A dolgozatom egy olyan megoldást szeretne bemutatni, amely segítségével önkéntesek részletes térképészeti adatokat gyűjthetnek és dolgozhatnak fel. A megoldás egy nagyobb rendszerbe illeszkedik, amely támogatja az önkéntesek munkájának koordinálását és eredményeik integrálását, azonban ennek ismertetése a dolgozat témáján kívül esik.

A megoldás célja, hogy a résztvevők a lehető legkisebb erőfeszítéssel tudjanak sávszintű részletességgel felmérni területeket. Ezt egy modellező- és térképszerkesztő alkalmazás teszi lehetővé, aminek segítségével a felhasználó minden tudását még a bejárás előtt rögzítheti, így minimalizálva a terepmunka mennyiségét. Ez a komponens támogatja a rögzített adatok utólagos feldolgozását és különböző jellegű bejárási tervek készítését is. A rendszer saját modellt használ a térképészeti elemek tárolására, amelyet kifejezetten az alkalmazás kedvéért terveztem.

A bejárási tervek készítése többek között Euler- és Hamilton-kör kereső algoritmusok implementálását kívánja meg. Az algoritmusok irodalmi algoritmusok, azonban implementációjuk saját elgondoláson alapul.

A terepmunka közben egy Android platformra írt alkalmazás van az adatrögzítők segítségére, amely a bejárási terv alapján kéri a hiányzó – a modellezés fázisában nem ismert – adatok rögzítést, amiből egy naplóállományt készít. Ezt a naplóállományt a modellező alkalmazás dolgozza fel. Az alkalmazás a pozícióadatokat egy Bluetooth-on keresztül csatlakoztatott GPS vevőből nyeri úgy, hogy a GPS eszközök illesztésére használt NMEA protokoll üzeneteit értelmezi egy saját fejlesztésű könyvtár segítségével.

A fentiek miatt dolgozatban kitértem a GPS helymeghatározó rendszer működésére. Kiemelt hangsúlyt kapott a mérési hibák okainak és lehetséges ellenszereinek tárgyalása, mivel ez a problémakör a leglényegesebb a térképkésztés szempontjából.

A rendszer Java nyelven írt komponensekből épül fel. A téma másodlagos célja a kliens- és szerveroldalon egyaránt széles körben használt Java nyelvvel, a hozzá tartozó könyvtárakkal valamint az egyre növekvő népszerűségnek örvendő Android platformmal kapcsolatos ismeretim bővítése volt.

Abstract

This paper is intended to introduce a system, which allows for its users to collect and process spatial data. The components to be introduced are part of a larger system, which supports teamwork and is responsible for the integration of the collected data.

The main purpose of the developed solution is to help to reduce the amount of work needed at the field. This is supported by a modeling application, which allows the user to draw the graph-based model of the road network and assign all information to the model elements he is able to tell before the field-work. Moreover this application processes the collected data, and is able to create some different types of work plan. A work plan describes to the user an itiner for visiting the crossroads. The model used by the application is an own design, and discussed by this paper in details.

Amongst others, Eulerian- and Hamiltonian cycle finding algorithms were implemented for the plan generation. These algorithms are described by the literature, but their implementation was created by me.

During the field-work an Android application guides the users by the content of the filed-work plan, created by the modeling application. This mobile application asks the user to provide the data needed to complete the map, and produces a log file. This log is processed by the modeling application. The Android client gains the position data from an external device via Bluetooth by parsing the messages of the NMEA protocol. Parsing is done by an own library.

This paper contains a short summary about the GPS navigation system, which provides the position information. But by the effects influencing GPS accuracy, the paper goes more into the details, because this is the most important aspect for the map creation.

The system components are written in Java. The secondary goal of this work was the developing of my knowledge related to the widely used Java programming language and associated libraries as well as to the growing popularity Android platform.

1 A feladat célja és a rendszer vázlata

Az alábbi fejezetben szeretném ismertetni a célokat, amelyek jelen dolgozat elkészítését motiválták, és amelyek a bemutatásra kerülő rendszer terveit nagymértékben befolyásolták. A célok ismeretében kitérek a rendszer vázlatos felépítésére, és ismertetem a legmagasabb szintű komponenseinek szerepét.

1.1 Célok

Munkám célja, hogy egy olyan rendszert hozzak létre, amely támogatja navigációra alkalmas térképek közösségi erővel való elkészítését. A közösség erőfeszítésének eredményeként előálló térképnek sávszintű navigációra kell alkalmasnak lennie, ami túlmutat a legtöbb hasonló projekt képességein.

A navigációs képesség nem csak a személyautókra kell, hogy kiterjedjen. A modellezés során figyelmet fordítottam arra, hogy a kerékpárosok és gyalogosok számára fontos adatok tárolására is legyen lehetőség.

Szintén fontosnak tartottam a fennálló forgalmi korlátozások tárolását a térképben, hiszen a választott járműtípustól függően esetleg más-más utakat kell választani ezek miatt. Ezen kívül a sebességre vonatkozó korlátozások fontos információt jelentenek a tervezett útvonal időigények megbecsüléséhez.

Természetesen egy ilyen részletesebb adatgyűjtő munka nagyobb erőfeszítést követel meg a résztvevőktől, mint a hagyományos rendszerek, ezért a rendszer és komponenseinek tervezése során szem előtt kellett tartanom az önkéntesek munkájának teljes körű támogatását. A vezető szempont itt a terepen történő adatrögzítés munkaigényének minimalizálása volt.

A tervezés kezdetén szembesültem azzal, hogy az útvonalhálózat milyen sokféle elemből áll össze, és azzal, hogy ezeknek az elemeknek számos olyan attribútuma van, ami a navigáció szempontjából releváns lehet. Számba véve az ismert megoldásokat (pl.: Open StreetMap), úgy döntöttem, hogy saját modellt tervezek (3.1), amivel le tudom írni az igényimnek megfelelően az utak hálózatát, a meglévő modellek korlátozásai nélkül. Saját modellem tervezésekor a lehető legnagyobb rugalmasságra és bővíthetőségre törekedtem. Ezek a szempontok még a tárolási hatékonyság elveit is felülírták, amit a szelvény-koncepcióval (1.2.1) hidaltam át.

1.2 A rendszer felépítése

A rendszer egy elosztott rendszer, amely három fő komponensre tagolható. Az egyik ilyen komponens a térképkészítő alkalmazás, amely egy Java nyelven írt modellező rendszer. Ezzel lehetséges felrajzolni a feltérképezni kívánt terület útjainak topológiáját, mint egy gráfot. Az ilyen területeket szelvényeknek nevezem, amelyeket részletesen a szelvény-koncepcióról szóló fejezetben ismertetek (1.2.1). A gráf pontjai lehetnek kereszteződések és töréspontok, míg az élei az utak.

A modellező alkalmazás támogatja méretarányos modellek elkészítését, ha rendelkezésre áll egy alátét réteg (pl.: légi felvétel), amelyen az utcák elhelyezkedése kivehető. Ha ez nem lehetséges, akkor csak topológiai modell készül.

A felhasználónak lehetősége van minden tudását a szelvényhez tartozó területről a modellbe betölteni (utcanevek, házszámok, stb…). Ez azért különösen fontos, hogy a céljaimmal (1.1) összhangban, minimalizálni lehessen a terepi munka mennyiségét, amit ráadásul egy korlátozott beviteli képességű okostelefonon kell elvégezni.

Az előzetes modellezés, amennyiben méretarányosan történik, lehetőséget teremt arra, hogy egy optimális bejárási tervet készítsünk (4.3.1), a bejárandó távolság szempontjából. Nem méretarányos terv esetén olyan bejárási terv készül, ami a gráf minden élének bejárást garantálja (**Hiba! A hivatkozási forrás nem található.**).

Miután a szelvény modellezését befejeztük, és a bejárási tervet elkészítettük, a program exportálja a bejáráshoz szükséges adatokat, amelyeket aztán feltölthetünk egy Android alapú okostelefonra. A feltöltött modell tartalmaz mindent, amit a felhasználó tud az adott területről, így a kliens feladata a hiányzó részek kitöltetése a felhasználóval.

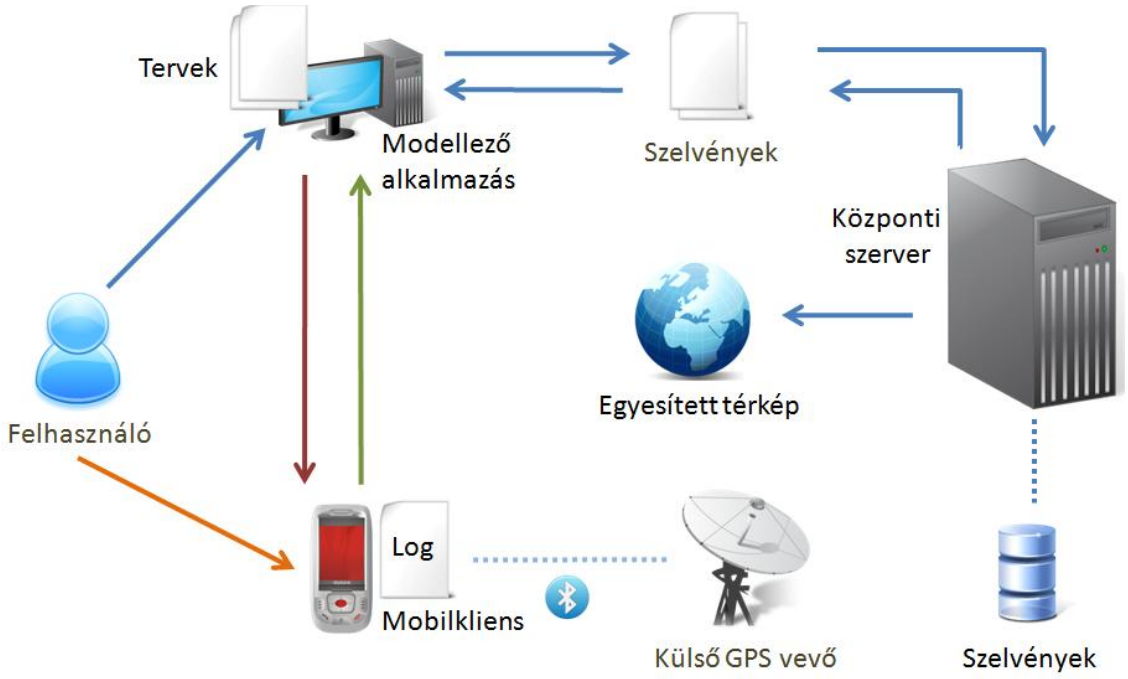
A következő fő rendszerkomponens tehát a mobilkliens (5), amely végigvezeti a felhasználót az előzetesen megtervezett bejárási útvonalon, és a megfelelő helyeken kéri a szükséges adatok megadását, illetve bizonyos helyek pozíciójának megjelölését (pl.: kereszteződés). A bejárás során az alkalmazás folyamatosan rögzíti a földrajzi pozíciónkat, amelyek forrása egy külső GPS vevő.

A bejárás végeztével tehát rendelkezésre áll a modell, és a hiányzó adatok. Ez utóbbiakat a felhasználónak importálnia kell a modellező alkalmazás segítségével. Ezután a felhasználó még dolgozhat a térképen, utólagosan szerkesztheti, pontosíthatja. A távlati terveim között szerepel, hogy a szerkesztés végeztével a szelvényt fel lehessen tölteni egy központi szerverre, ami a harmadik fő komponense lesz a rendszernek.

A központi szerver hangolná össze az önkéntesek működését. A szerveren az adminisztrátorok megadhatnák, hogy milyen szelvények felmérést kérik a felhasználóktól.

Az önkéntesek böngészhetnék a szelvények listáját és abból egy-egy szelvényt a saját részükre lefoglalhatnának. Ekkor a szelvény zárolódna, és ahhoz más, a zár fennállása alatt, nem férhetne hozzá. A központi szervert fel kell majd készíteni arra, hogy valaki elkezdi egy szelvény feltérképezését, de azt más fejezi be. Ebben az esetben a korábban feltöltött szelvényfájlt kellene visszaadni az újonnan belépő önkéntesnek, tehát ezek megőrzéséről a szervernek gondoskodnia kell.

A szerver feladata nemcsak a felhasználók által készített szelvények összegyűjtése lesz, hanem a feltöltött szelvények alapján egy egységes térkép összeállítása is. Ez utóbbi a tárolt szelvényekben található adatok kinyerésével kell majd végbemenjen, és szükségszerűen az adatok tömörítésével, illetve a tárolási struktúrák megváltoztatásával fog járni, hiszen a szelvények tárolási módja a könnyű módosíthatóságra fókuszál.



1-1. ábra A rendszer fő komponensei [rajzelemek: http://www.iconspedia.com]

1.2.1 A szelvények koncepciója és szerepük

A szelvény, ennek a dolgozatnak a kontextusában, egy kis földrajzi területet jelent, amely feltérképezését egy időben legfeljebb egy önkéntes végzi. A szelvények diszjunktak, átfedés közöttük nem megengedett.

A szelvények szerepe a rendszer szempontjából kettős. Egyrészt egyszerű eszközt biztosítanak a csoportmunka támogatására, másrészt lehetővé teszik azáltal, hogy kis területet fednek le, hogy olyan adattárolási megoldást alkalmazzunk az adatgyűjtés stádiumában, amely támogatja a módosítások hatékony elvégzését.

Természetesen ennek a megoldásnak a használatával elkerülhetetlenné válik a szelvényadatok konvertálása, amikor az egységes térképet készítjük. Azonban ezt az árat véleményem szerint megéri, hogy normalizált relációs sémát alkalmazhattam a modellezőprogram tárolási megoldásaként, és nem kellett programozottan biztosítanom az anomáliák kivédését. További előny, hogy az ilyen sémák objektum-relációs leképzése egyszerűbben megvalósítható, így sok nehézkes programozási feladatot spórolhattam meg.

Az egységes térkép előállításakor vagy frissítésekor tudni kell, hogy a szelvények határai hol húzódnak. Ennek támogatására a modellben a gráf csúcspontjaihoz létezik egy attribútum, amivel a kapupontok megjelölhetők. A kapupontok a szelvény határán helyezkednek el, és léteznek olyan élek, amelyek a szelvényen kívülről csatlakoznak beléjük. Látható, hogy kapupont legalább még egy másik szelvényben is kapupont, ezért ez a megoldás elégséges a szelvények illeszkedésének biztosítására.

2 A GPS működése [1]

Ebben a fejezetben szeretném vázlatosan bemutatni a GPS rendszer működési elvét, és ennek az alkalmazásom működésére vonatkozó relevanciáit. Ehelyütt nincs mód a rendszer működését teljes részletességgel ismertetni, így gyakran csak hivatkozást adok majd a dolgozat keretein túlmutató részletek irodalmára.

2.1 A műholdas navigáció rövid története

A műholdas navigáció igénye lényegében egyidős a műholdtechnikával. Manapság szinte mindenki az amerikai GPS rendszert használja, de meg kell említeni, hogy más hatalmak is létrehozták a saját rendszereiket (Compass [2], GLONASS [3], Galileo [4]), illetve dolgoznak ezen. Ezeket összefoglaló névvel **GNSS – Global Navigation Satelite Systems**, vagyis globális műholdas navigációs rendszereknek nevezik.

A ma ismert GPS rendszer, amellett, hogy az eredeti változatától számos modernizációs lépés választja el [5], nem volt előzmény nélküli. Ezek az előzmények mind hozzájárultak a közkedvelt GPS rendszer sikeréhez, amit a legjobban az bizonyít, hogy számos, a GPS által is használt technológia először az előd rendszerekben bizonyított.

Ez első működőképes műholdas navigációs rendszer a Transit volt, ami Doppler-effektust használta. Ezt a Timation követette, amiben már használtak űrbe telepített atomórákat. Az amerikai légierő Project 621B kódnevű megoldásában pedig megjelent a kódosztásos többszörös hozzáférés.

A **Navstar-GPS – NAVigation Satellite Timing And Ranging - Global Positioning System** projekt azért jött létre, hogy összefogja a különféle ilyen irányú kutatásokat, és azok tapasztalatait hasznosítva egy közös rendszert hozzon létre. A program 1972-ben indult, és 1978-ban már felbocsátották az első Block I sorozatú műholdat [6]. Ezek azóta már elöregedtek, jelenleg a Block II fősorozatbeli műholdak különféle változatai népesítik be a pályákat [7].

A teljes működőképességét 1995-ben érte el a rendszer, majd a rákövetkező évben civil felhasználásra is elérhetővé vált. Ekkor a civil használatra szánt vivőket (a civil és katonai felhasználásra szánt adások egymástól frekvenciában is elkülönülnek) az

amerikai fegyveres erők zavarták (**SA – Selective Availability**). A zavarást 2000. május 2-án kapcsolták ki, mivel a differenciális megoldások amúgy is lehetővé tették a zavarójel szűrését [8], és a civil szféra igénye a pontos helymeghatározásra is egyre jobban erősödött.

2.2 A GPS rendszer működése

Ahogy arra már az előző fejezet is utalt, a műholdas navigáció igénye a 60-as években merült fel. Számos kísérleti projekt indult, míg végül az amerikai kormányszervek kitűzték egy egységes navigációs rendszer – a későbbi NAVSTAR GPS – legfontosabb paramétereit: globális lefedettség, időjárástól független rendelkezésre állás, gyorsan mozgó vevők támogatása és nagy pontosság.

A GPS rendszer képes eleget tenni ezeknek a legkevésbé sem triviális követelményeknek [9].

2.2.1 A GPS rendszer felépítése

A GPS rendszer három fő szegmensből áll [10]. Az űrszegmens tartalmazza a műholdakat, amelyek 20183 km magasan keringenek. Az eredeti tervek szerint 6 pályasíkon 4-4 műhold foglal helyet, ami 6 műhold egyidejű megfigyelését teszi lehetővé a Föld majdnem minden pontjáról. Jelenleg azonban a működőképes műholdak száma valamivel 30 db felett ingadozik, így 9-10 műhold is látható lehet a Föld egy tetszőleges pontjáról. A műholdpályák 55°-os inklinációjúak, és egymással 60°-os szöget zárnak be. A műholdak keringési magassága úgy van meghatározva, hogy periódusidejük egy sziderikus nap fele (kb.: 11óra 58perc) legyen [11]. A sziderikus nap hossza, szemben a szoláris napéval, egész évben állandó, mivel a Föld körülfordulását egy távoli csillaghoz képest értelmezi.

A második szegmens a földi szegmens, amely egy fő irányító központból (Colorado Spings) és két tartalék központból, illetve több, a Föld körül elhelyezkedő megfigyelőpontból áll (Kwajalein, Hawaii, Ascension Island, Diego Garcia) [11]. A rendszert az amerikai légierő tartja karban [12]. Feladatuk, hogy folyamatosan figyeljék a műholdak pályáját, és felügyeljék a rajtuk található atomórák együttfutását. A pályahibákat és órahibákat 2 óránként frissítik, és töltik fel a műholdakra földi antennák segítségével. A műholdak ezeket az adatokat a navigációs üzeneteikben visszasugározzák a vevőknek.

A harmadik szegmens a felhasználói eszközök szegmense. A GPS rendszer képes korlátlan számú vevőt kiszolgálni, mivel a vevők nem kommunikálnak a műholdakkal, csupán figyelik azok adását.

2.2.2 A pozíció meghatározásának elve

A GPS esetén a pozíció számítása azon alapul, hogy ismerjük néhány ismert pozíciójú műholdtól vett távolságunkat.

Az egyes műholdaktól vett távolság meghatározásához a jelterjedési időt használják a készülékek. Azonban a rádiójelek közel fénysebességgel terjednek, így minden nanoszekundum órahibáért súlyos métereket fizetünk a pontosságban. A problémát tovább bonyolítja, hogy a felhasználói vevőkben olcsó kvarckristály órák vannak, amelyek pontossága ennél nagyságrendekkel rosszabb.

Alapvetően három műholdtól vett távolság ismerete elegendő lehetne a vevő számára, hogy meghatározza a pozícióját (az eredményül kapott 2 pozíció közül az egyik kizárható), de ehhez nagyon pontos órára lenne szükség. Feltételeznünk kell tehát, hogy a vevőnk órája a mérés pillanatában t_u idővel el van csúszva a GPS rendszeridőhöz képest a jelterjedési idő alapján. A vevő keresett pozícióját jelölje a következő vektor (x_u, y_u, z_u). A vevő meghatározza a távolságát 4 műholdhoz képest. Mivel a távolság meghatározásakor az óraeltolódást nem küszöbölte ki, ezért ezt a távolságot pszeudotávolságnak szokás nevezni. Az $i.$ műholdtól vett pszeudotávolságot jelölje d_i, míg az $i.$ műhold pozícióját a (x_i, y_i, z_i). Ekkor a pszeudotávolság ismeretében az $i.$ műholdra az alábbi egyenlet adódik:

$$d_i = \sqrt{(x_i - x_u)^2 + (y_i - y_u)^2 + (z_i - z_u)^2} + ct_u$$

Négy műholdra tehát az alábbi nemlineráris egyenletrendszert kapjuk:

$$d_1 = \sqrt{(x_1 - x_u)^2 + (y_1 - y_u)^2 + (z_1 - z_u)^2} + ct_u$$

$$d_2 = \sqrt{(x_2 - x_u)^2 + (y_2 - y_u)^2 + (z_2 - z_u)^2} + ct_u$$

$$d_3 = \sqrt{(x_3 - x_u)^2 + (y_3 - y_u)^2 + (z_3 - z_u)^2} + ct_u$$

$$d_4 = \sqrt{(x_4 - x_u)^2 + (y_4 - y_u)^2 + (z_4 - z_u)^2} + ct_u$$

Az egyenletrendszert a vevő megoldja, és így birtokába kerül az saját órahibájának, és a pozíciójának is.

Ezek szerint tehát szükséges, hogy a műholdak pontos helyét ismerjük. A hely megadásához pedig koordinátarendszerre van szükség. A műholdak helyzetét az ún. **ECI - Earth-Centered Inertial** koordinátarendszerben tudjuk megadni [13]. Ennek egy

inerciarendszernek kell lennie, így a műholdak mozgására érvényesek maradnak Newton mozgástörvényei. Az ECI koordinátarendszer *x* tengelye általában az éggömb egy rögzített pontja irányába mutat. Az *xy*-sík egybeesik a Föld egyenlítőjének síkjával, a *z* tengely a földrajzi északi sarkon megy keresztül, és merőleges a *xy*-síkra. Az *y* tengely úgy kerül meghatározásra, hogy egy jobbsodrású, derékszögű koordinátarendszert alkosson az *x* és *z* tengelyekkel. A koordinátarendszer origója a Föld tömegközéppontja.

Azonban a Föld, és így annak tömegközéppontja, és vele a koordinátarendszer mozgása nem egyenletes az éggömbhöz képes (főleg a Napnak és Holdnak a szabálytalan alakú Földre gyakorolt gravitációs vonzása miatt), így nem beszélhetünk inerciarendszerről. A megoldás az, hogy a koordinátarendszer paramétereit egy adott időpillanathoz rögzítik. A GPS esetében ez 2000. január 1. 1200UTC (ún. *epoch*). Az *x* tengely iránya a tavaszpont felé mutat. Ezt szokás J2000 viszonyítási rendszernek is nevezni.

Mindemellett egy földi felhasználó számára érdekesebb egy Földhöz rögzített koordinátarendszerbeli pozíciója. Az ilyen rendszereket **ECEF - Earth-Centered Earth-Fixed** koordinátarendszernek nevezik. Az *x* tengely ebben az esetben a 0°-os meridián irányába, a z tengely pedig a földrajzi északi sark irányába mutat. Az *xy* tengelyek síkja egybeesik az egyenlítő síkjával, míg az *y* tengely úgy kerül meghatározásra, hogy a többi tengellyel együtt egy jobbsodrású derékszögű koordinátarendszert alkosson.

Ezek a koordináták azonban még mindig nem a megszokott polárkoordináták, amelyeket a GPS vevők szolgáltatnak. Ahhoz, hogy a vevőkészülékek meghatározhassák a szélesség, hosszúság és magasság értékeket, szükség van egy dátumra, amihez képest a méréseket végezzük. A GPS a WGS-84-es dátumot használja erre a célra [13]. Érdekesség, hogy a WGS-84 az IERS által definiált referencia meridiánt **(IRM – International Reference Meridian)** használja 0° meridiánként, ami a greenwich-i obszervatóriumban kijelölt híres 0 meridiántól 5.31 szögmásodperccel keletre húzódik.

Megérdemel néhány szót a GPS által szolgáltatott magasságértékek taglalása is. Szinte minden ország saját vertikális dátumot használ a tengerszint feletti magasság megadására. Jelenleg Magyarországon a kronstadt-i (más néven balti) alapszinthez képest határozzuk meg a magasságokat [14]. Azonban ezek a tengerszintek nem esnek egybe. A GPS által szolgáltatott magasság csupán egy nyers érték, amely a WGS-84

által definiált referencia ellipszoidhoz képest adja meg a vevő magasságát. Ez az ellipszoid egyes helyeken jelentősen eltérhet a geoid alakjától.

2.2.3 GPS rádiós interfész [15]

A GPS műholdak rádióüzeneteket sugároznak, amelyek olyan információkat hordoznak a vevők számára, amelyek fontosak a helyzetük meghatározásához. Ebben a fejezetben a rádiós interfész működését és az üzenetek fontosabb részeit tekintem át.

Eredetileg minden GPS műhold két vivőfrekvencián ad, amelyeket L1 és L2 vivőnek hívunk. Valamennyi műhold ezt a két azonos frekvenciát használja, az egyes műholdak adása sem időben sem frekvenciában nem különül el egymástól. Ehelyett az egyes műholdak az adásukat más-más szórókóddal (**PRN – Pseudo-Random Noise**) modulálják, így kódosztásos többszörös hozzáférésről (CDMA) beszélhetünk.

Az L1 vivő $154 \times 10{,}23MHz = 1575{,}42MHz$ míg az L2 vivő $120 \times 10{,}23MHz = 1227{,}60MHz$ frekvencián ad. A navigációs üzenetet szóró és opcionálisan titkosító kódokkal többször modulálják, majd BPSK modulációval ültetik a vivőkre.

A civil eszközöknek szánt navigációs üzeneteket **(SPS – Standard Positioning Service**) egy ún. **C/A - Coarse Aquisition** álvéletlen kóddal modulálják, jelölje C(t). A C-kód egy álvéletlen kód, amelynek minden kódszava 1023 bites (ezeket a biteket nevezik *chip*-nek). A chipsebessége $1{,}023\ Mchip/s$, tehát egy kódszó nagyjából 1 ms hosszú, ami alatt a rádiójel nagyjából 300 métert tesz meg. Ez egy fontos korlátja az SPS által elérhető pontosságnak, mert a vevők elektronikája csak 1% körüli pontossággal tudja megállapítani a bitidő kezdetét (ld. még: 2.2.4), ami így 3 méter hibát okoz a mérésben.

A jelenleg kizárólag katonai célokra (**PPS – Precise Positioning Service**) használt P-kód ennél lényegesen hosszabb, periódusideje 1 hét és chipsebessége $10{,}23\ Mchip/s$, vagyis az előző gondolatmenetet követve itt a mérési hiba csak 30 cm. Jelenleg a P-kódot kriptográfiai védelemmel látják el, ami kizárja az illetéktelen (és a civil) használatot.

Eredetileg az L1 vivőn kisugárzott jel tartalmazott csak civil célokra is igénybe vehető jelet a katonai célú mellett, míg az L2 vivő csak a katonai kód számára volt fenntartva. Összefoglalásképpen megadom a jelek matematikai leírását, amelyekben $P(t)$ a P-kódot, $W(t)$ az ezt titkosító kódot, $C(t)$ pedig a C/A-kódot jelöli. $D(t)$-vel a

navigációs üzenetet jelöltem. Jól látható, hogy a civil adás intenzitása a titkosított adáshoz képest kétszeres.

$$L1(t) = a_1 W(t) P(t) D(t) \cos(\omega_1 t) + \sqrt{2} a_1 C(t) D(t) \sin(\omega_1 t)$$
$$L2(t) = a_2 W(t) P(t) D(t) \cos(\omega_2 t)$$

2.2.4 Pozíció mérése navigációs üzenetekkel – kódmérés [15]

A GPS vevő feladata tehát figyelni a jeleket és meghatározni azt az időt, ami alatt a jel a műholdtól az antennájáig elért, így már a vevő és műhold távolsága kiszámítható. A GPS vevő ismeri a műholdak PRN-kódját, így ezt felhasználva generál egy-egy referenciajelet, amit az ismert PRN kóddal modulál. Ezt a generált jelet ezután időben eltolva igyekszik megtalálni a maximális korrelációt a vett jellel. A maximális korreláció azt jelzi, hogy az időeltolás hatására a műholdból jövő kód, és a generált kód szinkronba került. (Az álvéletlen kódok keresztkorrelációja minimális, míg autokorrelációja maximális.) Tegyük fel, hogy a maximális korreláció eléréséhez t időeltolásra volt szükség. Ekkor a vevő már tudja, hogy közte és a műhold között a távolság $(t + np)c$, ahol $n \in \boldsymbol{N}$ és p egy kódszó leadásának ideje. Tehát ha 1023 bites kódszót veszünk alapul, aminek chipsebessége 1,023Mchip/s, akkor látható, hogy a vevő számára már kb. 300 km felbontással egyértelmű a távolság értéke. Már csak azt kell megtudni, hogy hány egész, 300 km hosszú kódszó volt a mérés pillanatában úton a műhold felől.

Ebben segítenek a műholdak által kibocsátott navigációs üzenetek. Egy navigáció üzenet egy keretből (frame) áll, ami öt alkeretre (subframe) tagoldóik. Minden alkeret 10 gépi szó terjedelmű, egy gépi szó 30 bitet tartalmaz. Tehát egy üzenet összesen 1500 bit, amelyet a műholdak 50bps sebességgel sugároznak, vagyis egy keret elküldése 30 másodpercet igényel.

Valamennyi alkeret egy TLM (telemetry) szóval indul, ami a vevővel való szinkronizációt szolgálja. A 2. szó minden alkeretben az aktuális GPS-hét kezdete óta eltelt idő. A maradék 8 szó alkeretenként eltérő.

Az első alkeretben az időadatok, időkorrekciós adatok és a műhold egészségjelzése kapott helyet. A 2. és 3. alkeretben közli a műhold a saját pályaadatait (*ephemeris*), amelyeket a földi szegmens a megfigyelések alapján feltöltött rá. A 4. és 5. alkeret lapokat tartalmaz (25-25 db), így ezek a részek az egymást követő üzenetekben változnak. A műholdak – egyebek mellett – itt közölnek hozzávetőleges pályaadatokat a társaikról (*almanach*).

Erre azért van szükség, mert a műholdakkal való szinkronizáció nagyon komplex feladat egy vevő számára. A legrosszabb esetben (pl.: első beüzemelés) a vevőnek elképzelése sincs, hogy hol van. Tehát nem tudja azt sem meghatározni, hogy milyen frekvencián keressen műholdak után, hiszen azok relatív sebességének ismerete nélkül nem tudja az adást torzító Doppler-effektust sem korrigálni. Emellett a vevőnek még a korábban ismertetett időeltolódás meghatározásával is foglalkoznia kell. A probléma tehát lényegében egy kétdimenziós keresési feladat. Ha a vevő egyszer sikeresen rátalál egy műholdra, akkor az *almanach* adatok segítségével már egyszerűen képes lehet újabb műholdakat találni.

Ennek a problémának az áthidalására bizonyos vevők támogatják, hogy *almanach* adatokat töltsünk le rájuk. Az A-GPS képes vevők pedig képesek fogadni ilyen adatokat például a mobilhálózattól, vagy az interneten keresztül.

2.2.5 Fázismérés [16]

A teljesség kedvéért meg kell említeni, hogy az imént ismertetett kódmérés mellett létezik egy teljesen másik megközelítés is a műhold-vevő távolság és így a pozíciónk meghatározására.

A fázismérés esetén a vevő először végez egy kódmérést a fentiek szerint, hogy pozícióját hozzávetőlegesen meghatározza. Ezt követően megkísérli kiszámolni, hogy hány egész vivőjel peridus van a vevő és műhold között, illetve figyeli, hogy a vivőjel milyen fázisban éri el az antennát. Ilyenkor a navigációs üzenetekkel már nem törődik, csak a helyreállított vivőjelet figyeli. Az egész periódusok számának meghatározása a ciklus-többértelműségi probléma megoldását igényli.

Az ilyen módon működő készülékek rendkívül drágák, és ráadásul a hibák kiszűrése miatt kell egy második, ismert pozícióra telepített készülék is. Pontosságuk azonban akár az építőipar, vagy a geodézia számára is kielégítő, hiszen képesek 1-2 cm (hosszabb mérés esetén akár néhány milliméter) pontossággal a pozíció meghatározására [17].

2.3 A méréseket terhelő hibák és javításuk

Ebben a fejezetben ismeretem azokat az okokat, amelyek a méréseink eredményét eltorzítják, és így hibákat okoznak. Az okok mellett kitérek azokra az eszközökre, amelyekkel a hibák hatását minimalizálni lehet.

A pozíció meghatározásakor fellépő hiba két tényező függvénye: A hibák egy része a vevő-műhold távolság meghatározásakor lép fel. Ezen hibák eredőjét nevezik **URE**-nek **(User Range Error).** A pozíció meghatározásakor több műhold távolságának meghatározására van szükség. Azonban a műholdak elhelyezkedése hatással van arra, hogy az URE mennyire terheli a mérésünk eredményét. Ezt a faktort nevezik pontossághígulásnak **(DOP - Dilution of Precision)**. A pozícióhiba az URE és a DOP szorzata.

2.3.1 Hibaforrások [9]

Ebben a szakaszban a műhold-vevő távolság meghatározása közben fellépő hibákat, vagyis az URE komponenseit ismertetem.

Ha az űr felől végigkövetjük a jel útját a vevőig, akkor az első hibaforrás nem más, mint a műholdak helyzetének és órájának pontatlansága. Ezeket a hibákat azonban a földi szegmens méri, és a korrekciós adatokat visszasugározza a műholdra, amely az *ephemeris* adatok között visszaküldi a vevőnek, így ez a hibaforrás nem okoz komoly problémát.

A következő probléma az ionoszféra, amely a Napból érkező elektromágneses sugárzás hatására számtalan töltött részecskét tartalmaz. Ezek aztán módosítják a rádiójelek terjedési sebességét, így okozva mérési hibát. Az ionoszféra adatait is monitorozza a földi üzemeltetés, de a dolgukat nehezíti, hogy a töltött részecskék száma a térben és az időben is dinamikusan változik számtalan tényező hatására. Az egyik lehetőség az, hogy a megfigyelt értékek alapján paraméterezett hibamodellel igyekeznek a vevők a hibahatás csökkentésére. Az ionoszféra hatása nem egyforma az egyes frekvenciákon, így ha módunk van mindkét vivő (L1 és L2) figyelésére, akkor a hiba hatása kiküszöbölhető. Ez a P-kód figyelése esetén lehetséges, és szerencsére a Block IIR-M és újabb műholdak által az L2 vivőre tett új civil kóddal ez a lehetőség már a civil szféra számára is nyitott.

A légkörben tovább haladva a troposzféra következik, amelyben a vízgőz jelenléte okoz nem frekvenciafüggő hibát. Szerencsére ezek a hibák jól modellezhetők és így hatásuk is túlnyomórészt csökkenthető.

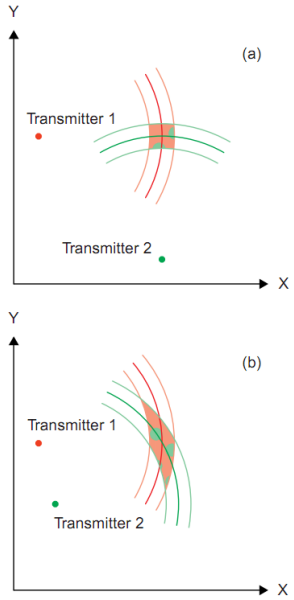
A troposzféra és az ionoszféra hatása annál nagyobb, minél többet utazik egy jel rajtuk keresztül, vagyis minél alacsonyabb pályán keringő műholdról érkezik. A legtöbb vevő támogatja a túl alacsony pályán keringő műholdak jeleinek szűrését.

A mérési eredményt tovább rontják a jelek többutas terjedése, az antenna hibái és a relativisztikus hatások. Ezek hatása többnyire kicsi a dolgozatban részletezett alkalmazás szempontjából, vagy jól kompenzált (relativisztikus hatások).

2.3.2 Pontossághígulás [18]

A pontossághígulás a pozíció meghatározásához használt GPS műholdak geometriai helyzetének a mérés pontosságára gyakorolt hatását hivatott kifejezni. Minél közelebb esnek a műholdak egymáshoz az égen, annál kedvezőtlenebb pontossággal határozható meg a pozíció, tehát a DOP érték nagy. Ez gyakorta előfordul, ha magas házak között csak egy keskeny szeletét látjuk az égboltnak (városi kanyoneffektus).

A pontossághígulás értékeket a GPS vevők képesek kiszámítani és az erre vonatkozó adatokat szolgáltatni (5.1.1). A következő táblázatban a mérés minőségét foglaltam össze a DOP értékek függvényében:



2-1 Pontossághígulás [18]

DOP érték	Minőség	Leírás
1	Ideális	A legnagyobb elérhető pontosság
1-2	Kiváló	Megfelelő szinte az összes alkalmazás számára
2-5	Jó	Ez a szint ahol még elfogadható navigációs döntések hozhatók a mérés alapján
5-10	Közepes	Még felhasználható értékek, de nem ajánlott
10-20	Elégséges	Csak nagyon durva becsléshez
>20	Elégtelen	Az adatokat el kell dobni, használhatatlanok

2-1. táblázat: DOP értékek jelentése

3 A modell bemutatása

A megvalósítandó alkalmazás legfontosabb képességei közé tartozik az összegyűjtött térképészeti adatok perzisztens tárolása, olyan formában, hogy azok utólagos szerkesztése és bővítése is könnyen megvalósítható legyen. Ezeknek a kívánalmaknak megfelel egy relációs adatbázisban történő tárolás.

A relációs adatbázis választása előtt mérlegelni kellett annak előnyeit és hátrányait. Egyik legfőbb előnye az, hogy széles körben támogatott, és jól használható API-k állnak rendelkezésre az adatok saját programból történő kezelésére, amelyek megkönnyítik az alkalmazás fejlesztését. Sok triviális kódrészlet megírásától szabadított meg, hogy objektum-relációs leképzést használtam a fejlesztés során. További előnye a relációs adatbázisoknak, hogy segítségükkel a módosítások hatékonyan kivitelezhetőek a tárolt adatokon. Ez fontos, hiszen az alkalmazás korábban ismertetett koncepciója nagyban épít az összegyűjtött adatok utólagos szerkesztésére.

Hátrányai is látszanak azonban a relációs adatbázisnak, amelyek főleg a várható hatalmas rekordszámból fakadnak. A bejárás során a földrajzi pontok legalább másodperces gyakorisággal kerülnek rögzítése, ami miatt az indexstruktúrák karbantartása – a méretüknél fogva – sok erőforrást igényelhet. Egyebek mellett erre a problémára kíván megoldást nyújtani a térképadatok területalapú partícionálása, a szelvények (1.2.1) segítségével.

Az adatbázisnak tehát alkalmasnak kell lennie arra, hogy egy navigációra alkalmas térkép illetve térképrészlet adatait tárolja. A következő szakaszban a modell részletes bemutatására kerül sor.

3.1 A térinformatikai modell áttekintése

A térképészeti adatok tárolási sémája hierarchikus kompozíciót követ. A legkisebb egység a földrajzi koordináta (továbbiakban *pont*), ami bármilyen objektum helyét jelölheti. Jelenleg a tárolás csak az utak vonalvezetésére és egyéb tulajdonságaira terjed ki, ezért a *pontokat* is csak erre a célra használom, de a modell további bővítése (például szeretnénk a buszmegállók helyét tárolni) könnyen megoldható.

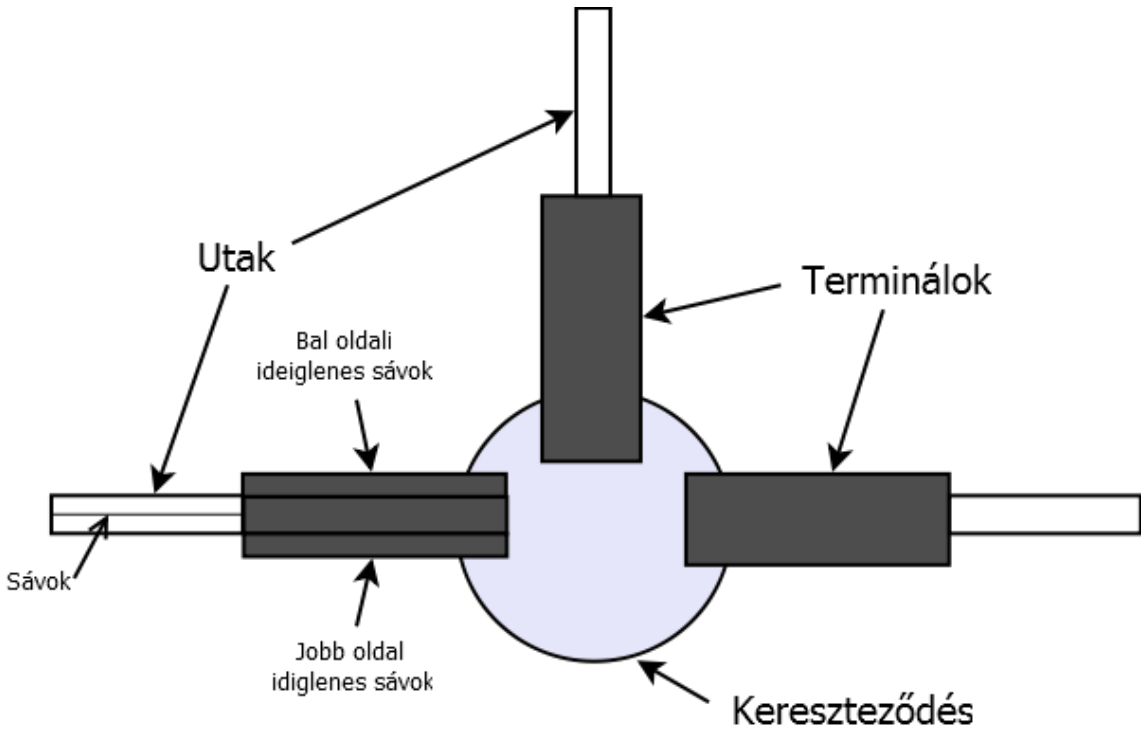
Az *utak* legkisebb egysége a *szegmens,* amely egy két *pontot* összekötő egyenes. A *szegmensek* segítségével az *utakat* tetszőleges pontossággal felbonthatjuk kis egyenesek sorozatára. Ilyen módon a kanyarulatok reprezentációja is megoldható,

ugyanakkor egyenes *út* esetén akár egyetlen *szegmens* is elegendő lehet, ami magában hordozza a bejárás optimalizálásának igényét.

Az *utak* a modell következő szintjén helyezkednek el. *Út*nak illetve *útszakasz*nak nevezem azokat az objektumokat, amelyek két szomszédos *kereszteződés*t kötnek össze. Az utak – ahogy korábban ismertettem - *szegmens*ekből állnak. Egy *út*nak számos attribútuma van, mint például a házszámozási adatok, vagy az, hogy mely *közterület* része. Lényeges tulajdonsága még az *útszakasz*nak a típusa is, ami a közúthálózati besorolását adja meg. Az *út* számos további attribútummal is bír, amit később részletesen ismertetek.

A *közterületek* lehetnek utcák, terek, dűlők, fordulók, tehát bármilyen névvel rendelkező, útként használható entitások. Fontos paraméterük az irányítószám, illetve az, hogy mely *településhez* tartoznak.

A navigációs feladatok egy lehetséges interpretációja, hogy navigációs döntések egy optimális sorozatát kell meghoznunk. Azokat a pontokat, ahol ilyen döntések meghozatala szükségessé válhat, *kereszteződésnek* nevezem. Ez a *kereszteződés* tehát kicsit általánosabb jelentéssel bír, mint a köznyelvben megszokhattuk. A *kereszteződéseknek* – egyéb tulajdonságaik mellett – van egy listája a hozzá kapcsolódó *terminálokról.*



3-1 A metamodell áttekintő ábrája

A *terminálok* az utak csatlakozópontjai a *kereszteződéshez*. Egy *terminálnak* lehet térbeli kiterjedése. A *terminálok* legfőbb szerepe a forgalmasabb kereszteződéseknél gyakran megjelenő kanyarodást és besorolást segítő sávok nyilvántartása. Ezeket a sávokat ideiglenes sávoknak nevezem. Egy *terminál* az első ideiglenes *sáv* megnyílásával kezdődik, és a *kereszteződésbe* való belépés pontjáig tart.

A navigációs döntések *sáv*szintű támogatása indokolt, hiszen hasznos, ha már jó előre tudjuk, hogy mely *sáv*on kell folytatnunk az utunkat. S*ávok* tartozhatnak az *útszakaszok*hoz, vagy *terminál*okhoz. Előbbiek az *út* teljes hosszában jelen vannak, míg utóbbiak csak a terminálokon belül léteznek.

A navigációs döntéseket jelentősen befolyásolhatják a járművekre vonatkozó *korlátozás*ok. A modell támogatja a *korlátozás*ok sávszintű megadását. Egy korlátozás bármilyen megszorítás lehet, ami a KRESZ megenged (pl.: sebességkorlátozás, behajtási tilalom, súlykorlátozás, stb...).

A *kereszteződés*ek – különösen az összetettebbek – gyakran nem teszik lehetővé, hogy tetszőleges *sáv*ba kanyarodjuk, így a *sáv*ok rendelkeznek a lehetséges *sávkapcsolatok* listájával is. Ezek a *kapcsolatok* a modellben önálló entitásként jelennek meg, így lehetőség van *korlátozás*okat rendelni hozzájuk.

3.2 A térinformatikai modell részletes tervei

Az alkalmazás implementációja során objektum-relációs leképzést alkalmaztam. Az alábbi fejezetben sorra veszem a modell elemtípusait, és mindegyikhez megadom annak a táblának a nevét, amelyben az adatbázis tárolja az egyes típusok példányait.

3.2.1 Földrajzi pontok – GEOPOINTS tábla

A térképészeti adatok legkisebb egysége a földrajzi *pont*, amely bármilyen másik objektum helyzetét jelölheti. A pontokat a GEOPOINTS nevű táblában tárolja a rendszer, amelyet a modellező alkalmazás a **`GeoPoint`** entitásosztály példányaira képez le. A *pont* három koordinátával adott: szélesség, hosszúság, és magasság. Ezen kívül tárolni kell a rögzítés időpontját és a mérés pontosságát megadó pontossághígulás értékeket (2.3.2), amik hasznosak lehetnek, ha a régi, vagy rossz minőségű adatok megújításával kapcsolatos igény lépne fel. Emellett a *pont*ban tárolva van, hogy hány műhold figyelembevételével történt a meghatározása.

A szélességi és a hosszúsági koordináták esetén az előjeleket úgy definiáltam, hogy a Föld északi féltekéjén lévő szélességek pozitív, míg a déliek negatív előjellel szerepelnek. Hasonlóan, a hosszúságok közül a keleti hosszúságok kapták a pozitív, míg a nyugatiak a negatív előjelet.

A magasságértékek meghatározása a WGS-84 geoidhoz képest értendő (2.2.2).

3.2.2 Szegmensek – SEGMENTS tábla

Egy *szegmens* két szomszédos *pont*ot köt össze, tehát mindig egy mértani szakaszt jelöl. A *szegmens*ek főleg a térkép rajzolását könnyítik meg, hiszen bármilyen görbe útszakaszt közelíthetünk szakaszokkal, és a rajzoláshoz szükséges számítások ezekkel egyszerűen megvalósíthatók. Az utak (3.2.3) *szegmens*ekből épülnek föl, és egy *szegmens* pontosan egy úthoz tartozik.

A *szegmens*ek, a **`Segment`** entitás osztály példányai, amely pontosan két *pont*ra tartalmaz referenciát.

3.2.3 Utak – ROADS tábla

A modellben *út*nak vagy *útszakasznak* nevezek minden olyan entitást, amely szomszédos *kereszteződés*eket köt össze. Az *ut*ak *szegmens*ek láncolataként épülnek fel, és minden *út*hoz egyértelműen tartozik egy közterület (út, utca, tér, stb…), aminek az adott *útszakasz* a részét képezi.

Az *ut*ak tartalmazzák a házszámozási adatokat is: Minden *útszakasz* esetében megadhatjuk, hogy milyen számozási séma (**`NumerationShema`**) jellemző rájuk. A számozási séma adja meg, hogy a házszámok kiosztása milyen rendszert követ. Például a terek esetében gyakran csak az út egyik oldalán van házszámozás, máshol a hagyományos páros-páratlan oldal van érvényben. Valamennyi *út* képes rögzíteni a kezdő- és végpontján a páros illetve páratlan oldali házszámokat. Illetve, ha ezt a számozási séma nem teszi lehetővé (pl.: terek esetén), akkor a kezdő- és végszámot. Az alábbi táblázatban foglaltam össze a lehetséges számozási sémákat:

Id	Séma neve	Leírás
0	UNKNOWN	Ismeretlen a házszámozás, vagy nincs
1	NONE	Nincs számozás
2	NORMAL	Hagyományos páros-páratlan oldalas felosztás
3	SINGLE-SIDED	Az egyik oldalon folyamatos számozás
4	ODD-ONLY	Csak páratlan számok a páratlan oldalon
5	EVEN-ONLY	Csak páros számok a páros oldalon

3-1. táblázat Számozási sémák összefoglalása

Az *ut*akra jellemző a típusuk is, ami a megadja, hogy milyen osztályú utat reprezentál az adott modellelem. Az utak osztályozását, és az egyes útosztályokba való tartozás kritériumait jelenleg a 19/1994. KVHM [19] rendelet rögzíti, amely a modellem alapjául szolgált. Azonban a modell a rendeletben foglaltaknál szélesebb skáláját nyújtja az útosztályoknak, így biztosítva bizonyos speciális igények (gyalogos- és kerékpáros navigáció) támogatását. Az alábbi táblázat foglalja össze a modell által ismert útosztályokat:

Típus neve	Sávrendszer	Leírás
PEDESTRIAN	SIMPLE	Gyalogút, nincs járműforgalom
STAIRWAY	SIMPLE	Lépcső
STAIRWAY_WITH_LIFT	SIMPLE	Lépcső mozgássérült felvonóval
RAMP	SIMPLE	Rámpa
BICYCLE	SIMPLE	Kerékpárút
BICYCLE_AND_PEDESTRIAN	SIMPLE	Kerékpár- és gyalogút
PATH	SIMPLE	Ösvény
TOURIST_WAY	SIMPLE	Turistaút
ZERO-WAY	SIMPLE	sikátor, köz, ahol autó nem fér el
ONE-WAY	ONE_LANE	keskeny út egy forgalmi sáv szélességben
TWO-WAY	ONE_LANE	szélesebb út, minimum két gépkocsi széles
BIDIRECTIONAL	BIDIRECTIONAL	Jelentősebb út, útburkolati jelekkel jelölt
UNIDIRECTIONAL	UNIDIRECTIONAL	Osztott pályás út egy útteste
ROAD	BOTH	Bekötő és összekötő utak
MAIN_ROAD	BOTH	Országos főút
HIGHWAY	UNIDIRECTIONAL	Autóút egy útteste
MOTORWAY	UNIDIRECTIONAL	Autópálya egy útteste

3-2. táblázat Útosztályok összefoglalása

A fenti táblázatban minden útosztályhoz fel van tüntetve, hogy milyen sávrendszere lehet. A sávrendszerek lényegében megszorítások arra nézve, hogy milyen *sáv*okat és mennyit vehetünk fel az egyes *ut*akhoz. Ezekről a kényszerekről az alábbi táblázat nyújt összefoglalást.

Sávrendszer neve	Leírás
SIMPLE	Egy darab határozatlan irányú sáv
ONE_WAY	Egy darab határozott irányú sáv
ONE_LANE	Egy darab sáv határozott vagy határozatlan irányítással
UNIDIRECTIONAL	Tetszőleges számú sáv, de azonos határozott irányítással
BIDIRECTIONAL	Tetszőleges számú sáv, legalább egy, a többivel ellentétes irányítással (csak határozott irányítás)
BOTH	Tetszőleges számú sáv, tetszőleges határozott irányítással

3-3. táblázat Sávrendszerek ismertetése

Az utak további jellemzője a kiépítés módját adja meg. Az utak lehetséges kiépítési módjait az alábbi táblázat foglalja össze. Ezek a kiépítési módok főleg a térképrajzolás vezérlése során kapnak szerepet.

Sávrendszer neve	Leírás
NONE	Normális felszíni út
BRIDGE	Híd
TUNNEL	Alagút
OVERPASS	Felüljáró
UNDERPASS	Aluljáró

3-4. táblázat Utak lehetséges kiépítése

Egy érdekes kérdés lehet a gyalogos alul- illetve felüljárók leképezése a térképre. Ezen építmények modellezése úgy történik, hogy felvesszük a lejáratokat (STAIRWAY, vagy RAMP típusok) és az őket összekötő *ut*akat OVERPASS vagy UNDERPASS kiépítéssel.

Az *ut*ak a két végükön *terminál*okhoz kapcsolódnak, amelyek pedig a *kereszteződés*ek részei. Minden *út* alapértelmezés szerint irányított a *kezdőterminál*jától a *végterminál*jáig. Ez az irányítás csak referenciaként szolgál, nem jelenti feltétlenül azt, hogy az út valójában is egyirányú forgalmat bonyolít. A *sáv*ok irányítását például ehhez a referenciairányhoz képest határozhatjuk meg (3.2.4).

Az *ut*akat a modellező alkalmazás számára a **`Road`** entitásosztály testesíti meg. Tárolhatjuk egy útról még azt is, hogy van-e szilárd burkolata, közútként használható-e, illetve azt is, hogy része-e lakó-pihenő övezetnek.

3.2.4 Sávok – LANES tábla

A LANES tábla a *sáv*ok tárolására szolgál, amelyeket a **`Lane`** entitásosztály testesít meg az alkalmazás számára. *Sáv*okat rendelhetünk az *ut*akhoz és a *terminál*okhoz is. Előbbiek az *út* teljes hosszán léteznek, míg utóbbiak – az *ideiglenes sávok* – csak *terminál* területén.

A *sáv*ok irányítását kétféleképpen közelíthetjük meg, attól függően, hogy mihez viszonyítjuk. A modellben valamennyi *út*nak van irányítása, amelyet referenciaként használunk. A *sáv*ok irányítása tehát az őket tartalmazó *út* irányításához képes lehet megegyező (FORWARD) vagy ellentétes (BACKWARD). Ezeket *határozott irányítású* sávoknak nevezem. Léteznek még *határozatlan irányítású* (UNSPECIFIED) *sáv*ok is, amelyek célja a nagyszámú lakóút modellezésének egyszerűbbé tétele. A modellezés során ezt a fajtájú irányítást adjuk meg.

A *kereszteződés*ekből szemlélve a *sáv*ok irányítása lehet befelé jövő (EFFERENT) vagy kifelé menő (AFFERENT). Ez a fajtájú irányítási rendszer azért jelentős, mert a *kereszteződés*ek összeköttetési szabályaiban mindig EFFERENT *sáv*okat kapcsolhatunk össze AFFERENT irányítású *sáv*ot tartalmazó *terminál*lal. A határozatlan irányítású *sáv*ok ebből a szempontból egyszerre AFFERENT és EFFERENT irányításúként kezelendők.

Abban az esetben, ha *határozottan irányított sáv*okat használunk, akkor a FORWARD és BACKWARD *sáv*okat két külön halmaznak kell tekintenünk, éppen úgy, mint a különböző elhelyezkedésű *ideiglenes sáv*okat. Valamennyi halmaz elmei rendelkeznek számozással, ami a *sáv*ok sorrendjét hivatott jelölni. A *sáv*ok a belső *sáv*októl a külsők felé haladva 0-tól kezdődő egész sorszámokat kapnak. A többértelműséget elkerülendő, ha egy *út*on elhelyeztünk egy *határozatlan irányítású sáv*ot, akkor oda már több *sáv*ot nem vehetünk fel. Emellett az úttípusok korlátozhatják a kétirányú *sáv*ok használatát, illetve a *sáv*ok számát és irányítását (3-3. táblázat).

A *sáv*okat megkülönböztetjük még a típusuk (**`LaneType`**) szerint is. Az alábbi táblázat ismerteti a lehetséges *sávtípus*okat:

Típus neve	Leírás
GENERAL	Általános célú sáv
BUS	Buszsáv
BICYCLE	Kerékpársáv
PEDESTRIAN	Gyalogosforgalom számára fenntartott sáv
STOP	Leállósáv

3-5. táblázat Sávok típusai

Valamennyi *út* rendelkezik *sáv*okkal, de bizonyos típusú *ut*ak esetén az alapértelmezett sávkiosztás megváltoztatása csak korlátozottan, vagy egyáltalán nem lehetséges, amit az *út* típusához rendelt *sávrendszer* határoz meg (ld.: 3-2. táblázat és 3-3. táblázat).

3.2.5 Korlátozások – CONSTRAINTS tábla

A megalapozott navigációs döntésének egy további előfeltétele az, hogy a térkép leképezze az egyes utakon megadott *korlátozás*okat. A *korlátozás*okat a programban *sáv* szinten adhatjuk meg. A *korlátozás*ok halmaza a KRESZ [20] szabályaival összhangban került meghatározásra. A korlátozásokat a **`Constraint`** típus testesít meg.

Minden *korlátozás*nak van típusa, és rendelkezik egy számértékkel, aminek jelentése a típustól függ. Például a sebességhatár esetén ez a szám a megengedett legnagyobb sebesség értéke. A *korlátozás* érvényessége korlátozott lehet járműtípus

vagy időszak szerint. A *korlátozás* hatálya alól bizonyos járművek mentességet élvezhetnek.

A KRESZ rendelkezéseivel összhangban az alábbi járműtípusok lehetnek korlátozások alanyai.

Id	Járműtípus	Leírás
0	ANY	Minden alább felsorolt típus, kivéve gyalogos
1	MOTOR_VEHICLE	Gépjármű
2	PEDESTRIAN	Gyalogos (nem jármű, de logikailag ide tartozik)
3	CART	Kézikocsi
4	BICYCLE	Kerékpár
5	MOPED	Segédmotoros kerékpár
6	MOTORCYCLE	Motorkerékpár
7	TRUCK	Tehergépkocsi
8	SEMITRAILER	Nyerges vontató
9	BUS	Busz
10	TRACTOR	Mezőgazdasági vontató
11	ANIMAL	Állati erővel vont jármű
12	HAZARDOUS	Veszélyes anyagot szállító jármű
13	LONG-VEHICLE	Járműszerelvény

3-6. táblázat Járműtípusok

A modell az alább felsorolt típusú korlátozások sávokhoz rendelését támogatja.

Id	Korlátozástípus	Leírás	Számérték mértékegysége
0	SPEED_LIMIT	Maximális sebesség	km/h
1	MINIMUM_SPEED	Kötelező legkisebb sebesség	km/h
2	WIDTH_LIMIT	Szélességkorlátozás	m
3	HEIGHT_LIMIT	Magasságkorlátozás	m
4	LENGTH_LIMIT	Hosszkorlátozás	m
5	WEIGHT_LIMIT	Megengedett max. össztömeg	tonna
6	AXLE_LIMIT	Megengedett max. tg.terhelés	tonna
7	ENTRY_LIMIT	Behajtási tilalom	[nincs értelmezve]

3-7. táblázat Korlátozástípusok és jellemzőik

3.2.6 Kereszteződések – CROSSROADS tábla

Jelen modellben a *kereszteződés*ek – amelyeket a **`Crossroad`** entitásosztály testesít meg – olyan helyeket jelölnek, ahol navigációs döntések meghozatala szükséges lehet. Például ha egy helyen megszűnik egy *sáv*, akkor ezen a helyen egy *kereszteződés*t kell felvenni, és meg kell adni, hogy melyik *sáv*ból melyik kivezető úton folytathatjuk az utunkat. Természetesen ez a szituáció automatizálható az üzleti logika szintjén, de az adatbázis szintjén a fenti logika érvényesül.

Az utak, amelyek a *kereszteződés*ben találkoznak, nem közvetlenül csatlakoznak a *kereszteződés*hez, hanem *terminál*okon keresztül, így a *kereszteződés* lényegében egy lista a *terminál*okról.

A *kereszteződés*eknek van típusa, amely a grafikus megjelenítésére és az adatgyűjtés módjára van hatással. A lehetséges kereszteződés-típusokat a következő táblázat ismerteti:

Id	Név	Leírás
0	SIMPLE	Egyszerű kereszteződés, a termináloknak nincs kiterjedésük
1	CROSSROAD	Kereszteződés
2	BINARY	Két terminállal rendelkező kereszteződés
3	NUMERATION	Házszámozás megadása egyenes úton
4	ROUNDABOUT	Körforgalom
5	EXIT	Autópálya/autóút lehajtó
6	ENTRANCE	Autópálya/autóút felhajtó

3-8. táblázat Támogatott kereszteződés-típusok

A SIMPLE típus bevezetését az a felismerés indokolja, hogy a legnagyobb számban az olyan *kereszteződés*ek fordulnak elő, ahol csak kis utcák metszik egymást. Ezeken a helyeken nincs értelme a modell által kínált valamennyi lehetőség kitöltését megkövetelni (pl.: ilyen helyeken nincsenek kanyarodósávok). A SIMPLE típusú *kereszteződés*ek esetében minden *terminál* kezdő és végpontja egybe esik, a *kereszteződés* körülbelüli középpontjába. Így a bejárás során mindössze egy pontot kell rögzíteni.

A SIMPLE típus alkalmazása csak bizonyos körülmények esetén engedélyezett. Szükséges, hogy a kereszteződés valamennyi útja vagy legfeljebb irányonként egy forgalmi sávval rendelkezzék, vagy egy kétirányú sávot tartalmazzon. Ezen kívül fontos, hogy ne legyen kanyarodósáv egyik úton sem.

A BINARY típus használata akkor indokolt, ha nincs szó tényeges útkereszteződésről, nem változik meg az utca neve sem, csupán a *sáv*ok tulajdonságaiban áll be változás. A NUMERATION típus esetén nincs semmilyen változás. Használata akkor indokolt, ha egy hosszú *út*on szeretnénk megadni egy pontban a házszámozást. Ezáltal az inverz geokódolás esetén az interpoláció pontosabb lehet.

Lehetőség van a *kereszteződés*eknek nevet is adni, ami hasznos lehet például autópálya csomópontok vagy más nevezetes csomópontok (pl.: BAH-csomópont) esetén.

3.2.7 Terminálok tárolása - TERMINALS tábla

A *terminál*ok (**`Terminal`**) jelölik az *ut*ak végeit, és egyben összekötő szerepet töltenek be az *ut*ak és a *kereszteződés*ek között. A *terminál*ok egy csoportja egy *kereszteződés*t alkot, és minden *terminál* pontosan egy *kereszteződés*hez tartozhat. Minden *terminál* rendelkezik kezdő- és végponttal, amelyek akár egybe is eshetnek. A *terminál* kezdőpontja ott található, ahol az első *ideiglenes sáv*ja megnyílik. *Ideiglenes sáv* minden olyan *sáv*, amely a *kereszteződés* előtt röviddel jön létre az irányváltoztatás megkönnyítésére.

A *terminál* végpontja általában a *kereszteződés* belsejébe való belépési pontnál található, gyakorlatilag ott, ahol az út szegélye véget ér. Ez alól egy kivétel van, az úgynevezett egyszerű (SIMPLE) *kereszteződés*, amelynél a *kereszteződés* minden *terminál*jának kezdő- és végpontja egy pontba, a *kereszteződés* körülbelüli középpontjába esik.

A *terminál* több listát is tárol az *ideiglenes sávok*ból. Egy-egy listája van a *kereszteződés*be befelé tartó balra és jobbra kanyarodó *sáv*oknak, éppen úgy, mint a *kereszteződés*ből kifelé tartó balról illetve jobbról történő besorolást segítő *sáv*oknak. A *sáv*ok sorrendjét a sorszámozásuk (3.2.4) adja meg, amely mind a négy listára függetlenül kerül kiosztásra.

Minden *terminál*nak van egy `ordinal` paramétere is, amely a *kereszteződés*beli helyzetét jelöli. Az `ordinal` meghatározása úgy történik, hogy a 0-t a *kereszteződés* azon *terminál*ja kapja, amely iránya a legkisebb szöget zárja be az északi iránnyal kelet felől. A további számozás az óramutató járásával megegyező körüljárási irányban történik, úgy, hogy a szomszédos *terminál*ok szomszédos sorszámokat kapjanak. A terminálról tároljuk azt is, hogy a hozzá kapcsolódó útnak a kezdő- vagy a végterminálja.

A *terminál*okról ezen kívül még azt is tároljuk, hogy lehetséges-e gyalogosan átkelni a *terminál*ba csatlakozó *út*on a *terminál* területén belül.

3.3 A tervezés során használt modell

A modellezés kezdetén lényegében egy gráf rajzolása történik meg. A gráf csúcsokból és élekből áll. Kezdetben, a bejárás előtt a kereszteződések helyzete és az utak vonalvezetése nem ismert, így róluk csak a tervezőrendszerbeli koordinátákat tudjuk rögzíteni. Szerencsétlen dolog lenne ezeket a koordinátákat az egyes

modellelemekhez tárolni, hiszen a térinformatikai modellhez szemantikailag a földrajzi koordináták tartoznak.

Az élek megadása során, ha egy alátétréteg segítségével rajzoljuk a modellünket, törekedni kell arra, hogy az utak vonalát a lehető legjobban kövessük. Ennek módja az, hogy a gráf éleit nem egyenesekkel, hanem törtvonalakkal határozzuk meg. Ezen vonalak töréspontjai nyilván nem számítanak kereszteződésnek, hiszen ezeken a pontokon nem kell navigációs döntést hozni.

Az itt felsoroltak miatt indokoltnak tartottam a térinformatikai modellről leválasztani a tervezési modellt. A tervezési modell tehát lényegében egy gráf, így a bejárás tervezéséhez az irodalom által ismertetett gráf algoritmusok használhatóak (4.3.1 és **Hiba! A hivatkozási forrás nem található.**).

A tervezési modell – ami egy gráf – tehát valójában egy réteg a térinformatikai modell felett. A gráf pontjait két osztályba lehet sorolni a térinformatika modellel való kapcsolatuk szempontjából: *töréspont*ok, amelyek csak az utak alakjának pontosabb modellezését szolgálják; és *valódi pont*ok, amelyek tényleges kereszteződések helyét jelölik.

A gráf élei között a modell nem tesz különbséget, azonban látható, hogy több *él* is tartozhat egy-egy *út*hoz. Például ha két *valódi pont* egy vagy több *töréspont*on keresztül kapcsolódik egymáshoz, akkor az a gráf szintjén *él*ek sorozatát jelenti, holott a térinformatika modellben csak egy *út* van a valódi pontokhoz tartozó kereszteződések között. Az ilyen utakhoz tehát az élek egy listája tartozik.

3.3.1 A töréspontok problémája

Egy kis kitérőt érdemes tenni a *töréspont*ok felé. Azért van szükség a modell gráfszerű megadására, mert ennek segítségével tudunk rövid bejárást tervezni az adott területhez . A *töréspont*ok lényegében 2-es fokszámú csúcsok a gráfban. Elvileg ezeket eliminálhatnánk, és az így kapott gráfon kereshetnénk Hamilton-kört a bejáráshoz. Emellett szól az is, hogy így a gráf csúcsainak száma redukálható, és a kör keresése ez által egyszerűbb lesz.

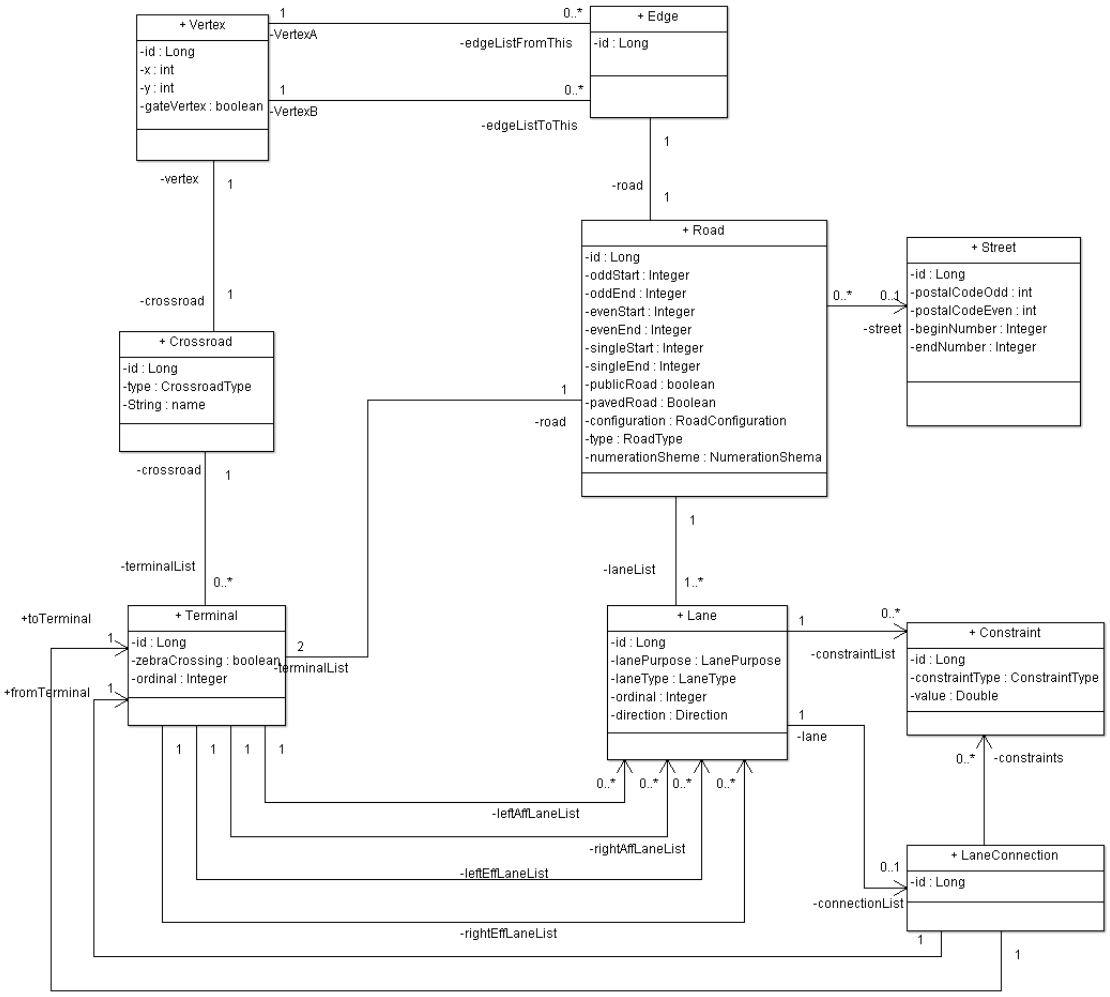
Ez az okfejtés meg is állná a helyét, ha a naiv algoritmusokkal szeretnénk a legkisebb súlyú Hamilton kört megtalálni, hiszen ezek futásideje legjobb esetben is exponenciálisan növekszik a csúcsok számával. Azonban a program megelégszik a közelítő algoritmusok használatával a gyorsaságért cserébe.

Az implementált közelítő algoritmusok viszont erősen építenek a feladat Euklideszi mivoltára (4.3.1), vagyis arra, hogy az élek hosszára a háromszög egyenlőtlenség fennáll. Ha az élek hosszának a törtvonal hosszát vennénk, akkor ennek a feltételnek az igazsága a továbbiakban nem állna fönt.

Ezért a 2-es fokszámú pontokat nem vonja össze az algoritmus. Szerencsére a tapasztalataim alapján így is kielégítő sebességgel fut. Azonban így előállhat olyan helyzet, hogy a tervezett bejárás egy *töréspontból* visszafordít minket, és így nem jutunk el a következő *kereszteződés*hez. A *töréspont* helye, amit fel kellene venni, sajnos kevésbé egzakt, mint egy *kereszteződés*é.

Ezt a problémát úgy oldja meg a rendszer, hogy a mérést végző önkéntest a mobilkliens úgy utasítja, hogy addig menjen el, amíg a szomszédos *töréspont* vagy *kereszteződés* egyenes vonalban láthatóvá nem válik. Ez a probléma a töréspontok nélküli tervezés során nem jelentkezik, amely funkció a jelenlegi verzióban még nem támogatott.

3.3.2 Egyesített modelldiagram



3-2. ábra Egyesített modelldiagram

4 A modellező alkalmazás

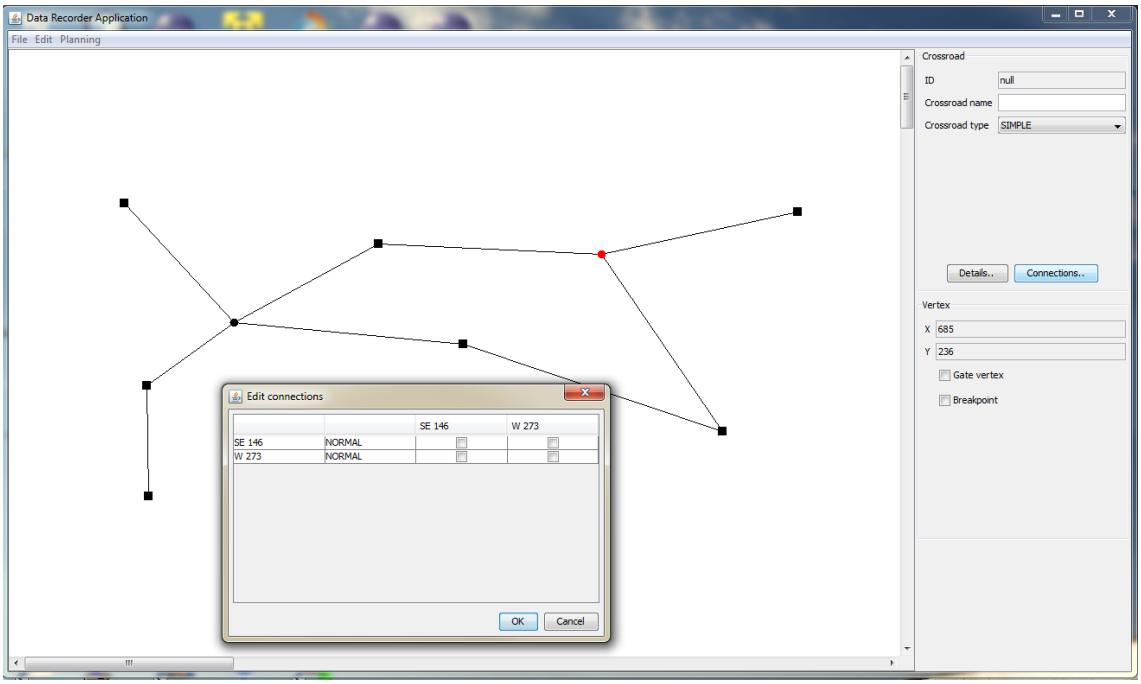
A modellező alkalmazás a rendszer fő komponenseinek egyike (1.2). Az alkalmazás feladata, hogy egy kényelmes grafikus környezetet nyújtson a felhasználók számára, amely segítségével rögzíthetik az ismereteiket az adott területről. Ezután az alkalmazás segítségével ezeket a még hiányos információkat exportálhatják a mobilkliens számára. Ezt követően a mobilkliens által összegyűjtött kiegészítő információkat (pl.: földrajzi koordináták) visszaimportálhatják a modellbe az alkalmazás segítségével, ami gondoskodik az adatoknak a modellbe történő megfelelő befésüléséről. Végül a program képes egy vázlatos térképet rajzolni az egyesített adatok alapján, amely a későbbiekben az adatok utólagos szerkesztését is támogatni fogja.

4.1 Az alkalmazás működésének ismertetése

Az alkalmazás segítségével a felhasználó egy *szelvény*t szerkeszthet egy időben, amit projektnek neveztem el. A projekteket az alkalmazás képes megnyitni és bezárni, illetve lehetőség van a módosítások mentésére valamint új projekt létrehozására is.

A program felhasználói felülete három nagy egységre tagolódik: felül egy menüsáv húzódik az alkalmazás funkcióinak elérésére, míg az ablak legnagyobb részét a szerkesztőfelület uralja. Itt van lehetősége a felhasználónak az egér segítségével az úthálózat vázlatos rajzát betáplálni. A szerkesztőfelülettől jobbra található sáv tartalma midig a szerkesztőpanelen aktuálisan kijelölt elem típusától függ. Ezen a jobb oldali panelen van lehetőség a kijelölt elemek tulajdonságainak beállítására.

Ahogy azt korábban bemutattam a szerkesztőpanel segítségével lényegében egy gráfot rajzolhatunk meg. Ebben a gráfban vannak *töréspont*ok és *valódi pont*ok, illetve élek. A *töréspontok* és a *valódi pont*ok rajzjeleiket tekintve is elkülönülnek. A *valódi pontok* – amelyek egy *kereszteződés*t jelölnek – körrel, míg a *töréspontok* négyzetekkel jelöltek. Minden *kapupont* kettős kör, míg a bejárás kezdőpontja háromszög jelölést kapott.



4-1 Az alkalmazás főképernyője

A szerkesztőfelület elkészítésekor fontos szempont volt, hogy a modell rajzolása gyorsan történjen. Az alábbi táblázat azt foglalja össze, hogy milyen billentyű és egérkombinációk érhetők el a rajzolás során.

Kombináció	Leírás
BAL EGÉR	Elem kijelölése
JOBB EGÉR	Új pont hozzáadása. Ha van kijelölt pont, az új pontot összeköti vele.
CTRL + JOBB EGÉR	Új pont hozzáadása, nincs kötés más ponthoz.
ALT + JOBB EGÉR	Kijelölt él kettéválasztása egy ponttal.
DELETE	Kijelölt elem törlése

4-1 táblázat Billentyű és egérkombinációk a szerkesztőfelületen

Ha kezdetben elhelyezünk egy pontot, akkor ez az új pont kapja meg a kijelölést. Ha van kijelölve pont, akkor egy új pont létrehozásakor rögtön létrejön egy él a korábban kijelölt pont és az új pont között, kivéve, ha nyomva tartjuk a CTRL billentyűt.

További kényelmi funkció, hogy ha egy él végpontját egy már meglévő ponton szeretnénk kijelölni, akkor az él automatikusan hozzákapcsolódik ehhez a ponthoz. Ez az automatizmus a modell mélyebb rétegeire is igaz. Tehát ha két *valódi pont*ot kötünk össze, akkor létrejön az élhez tartozó *út*, és a megfelelő *terminál*okon keresztül hozzákapcsolódik a *pont*okhoz tartozó *kereszteződés*hez is.

Ez akkor is igaz, ha egy éllel úgy kötünk össze két *töréspont*ot, hogy ez által két, távolabbi *valódi pont* között létrejön egy *töréspont*okon keresztül vezető út a gráfban.

Ekkor is létrejön az *út* (térinformatikai modellelem) a *valódi pont*ok *kereszteződés*ei között, és az *út*hoz hozzárendelődik a *töréspont*okra illeszkedő élekből álló lista. Ha egy pont törlése miatt egy ilyen gráfbeli út megszűnik két *valódi pont* között, akkor a hozzá tartozó élet is eltávolítja a program a modellből.

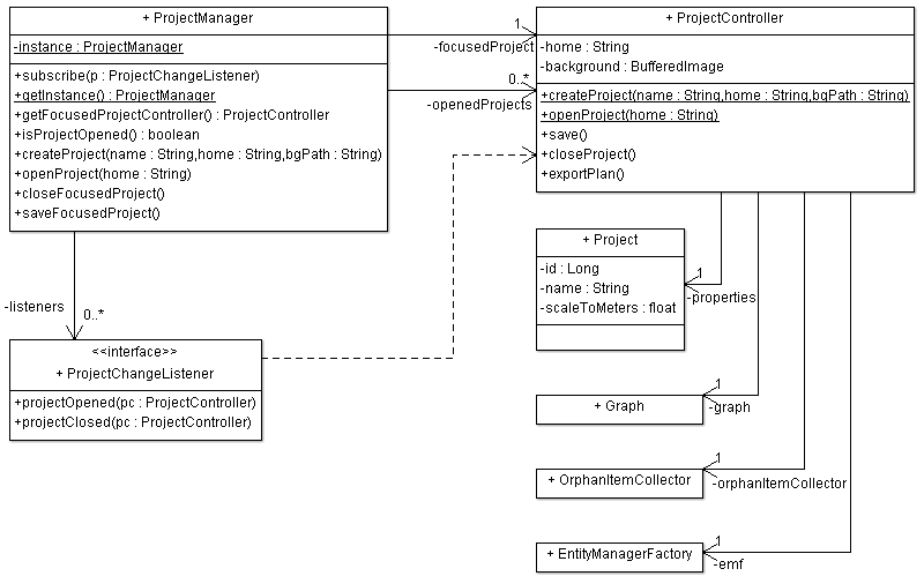
A modellező rendszer úgy működik, hogy egy pont mindig *töréspont*ként kezdi az életét. Azonban ha új élek hozzáadásával a fokszáma 3-ra nő, akkor a program előlépteti *valódi pont*tá, elkészíti a hozzá tartozó *kereszteződés*t, és az esetleg szükséges útkapcsolatokat is. Emellett lehetőség van a pontot kézzel valódi ponttá előléptetni, ami az előbbihez hasonló következményekkel jár.

4.2 Implementációs kérdések

4.2.1 Projektek kezelése

Ahogy az előző fejezetben említettem az alkalmazás segítségével projekteket kezelhetünk. A program megírása közben úgy jártam el, hogy megmaradjon a lehetősége annak, hogy az alkalmazás a későbbiekben képes legyen több projektet egyszerre kezelni.

A projektek kezeléséért a **`ProjectManager`** nevű singleton osztály a felelős. Amikor a felületen egy projekre vonatkozó akciót hajtunk végre, akkor ezek a kérések ehhez az osztályhoz futnak be. Ha szükséges a **`ProjectManager`** létrehoz egy **`ProjectController`** példányt, amelyen keresztül érhetjük el a projekt specifikus műveleteket. A **`ProjectController`** tárolja a projektszintű tulajdonságokat (**`Project`**), a modell gyökerét jelentő **`Graph`** osztályt, a tervezési nézet hátterét **`BufferedImage`**-ként, **`EntityManagerFactory`**-t, ami az adatbázis elérésének alapja, és számos más, később részletezett segédosztályt.



4-2 A projektek kezelését végző osztályok (részlet)

A grafikus felület számos elemének szüksége van arra az információra, hogy éppen van-e megnyitva projekt. Ezt a publisher-subscriber minta segítségével oldottam meg. A **`ProjectManager`** képes értesíteni minden regisztrált objektumot arról, hogy egy projekt megnyílt, vagy bezáródott. Ha egy osztály szeretne ezekről az eseményekről értesülni, meg kell valósítania a **`ProjectChangeListener`** interfészt, ami egy-egy metódust ír elő a megnyitás illetve bezárás eseményekre.

4.2.2 Perzisztencia

Az alkalmazás tervezése során nagyon fontos szempont volt az, hogy a fejlesztési idő meghatározó része ne az adatok mentését és visszatöltését végző kódrészletek elkészítésével teljen. Ezért esett a választásom a HSQL-DB nevű, java nyelven íródott adatbázis-kezelő rendszerre, amelyet a programom in-process módon futtat.

A HSQL-DB adatbázis-kezelőt szabványos SQL interfészen keresztül érhetjük el, és lehetővé teszi azt is, hogy JPA-t használjunk vele, ami az objektum-relációs leképzés nehézségeit is leveszi a vállamról. A JPA implementációk közül a Hibernate 3.6.1-es verzióját használom.

A HSQL-DB az adatbázist fájlrendszerben tárolja. Minden projekt könyvtárstruktúráján belül van egy `data` könyvtár, amelyben az adatokat tartalmazó fájlok találhatók. Egy projekt betöltésekor a **`GraphLoader`** nevű osztály gondoskodik

arról, hogy a teljes modell betöltődjön a memóriába. Ezt úgy éri el, hogy a gráf pontjaitól és éleitől kezdve rekurzívan bejárja az egész modellt.

A mentés során más okból bonyolult a helyzetünk. Ekkor a **`GraphSaver`** osztályt használja a program az adatok mentésére. Ez az osztály menti a gráf éleit és pontjait az **`EntityManager`** használatával. Az entitások közötti kapcsolatokon beállított kaszkádosítás miatt ez elegendő ahhoz, hogy a kapcsolódó modellelemek változásai is bekerüljenek az adatbázisba. Azonban az időközben törölt modellelemek eltávolításáról is gondoskodni kell. Ezek a modellelemek a törlés során minden referenciájukat elveszítik a modell más elmeivel, így a kaszkádosítás hatására a törlésük nem történne meg az adatbázisból. Az ilyen árva elemeket ezért mindig hozzá kell adni egy **`OrphanItemCollector`** típusú objektumhoz, ami minden **`ProjectController`**-hez létrejön. A mentés során aztán az **`OrphanItemCollector`** tartalmát kitörli a rendszer az adatbázisból, majd kiüríti a gyűjtőosztályt.

Megvalósításukat tekintve a **`GraphSaver`** és **`GraphLoader`** osztályok a **`SwingWorker`** osztály leszármazottjai, így a mentés és betöltés úgy futtatható, hogy a felhasználó felület a futás ideje alatt is fogadja a felhasználói eseményeket.

4.2.3 Modellelemek kezelése

Az eddigi leírásából jól kivehető, hogy egy összetett, számos kapcsolatot és még annál is több kényszert kezelő modellre épül az alkalmazás. Annak érdekében, hogy ennek a komplexitásnak a kezelése az implementációban ne vezessen káoszhoz, kidolgoztam egy minden modellelemre egységes programozási konvenciót.

A konvenció lényege, hogy minden modellelemet két osztály definiál, amelyek 1-1 kapcsolatban állnak egymással. Az egyik osztályt entitásosztálynak, míg a másikat konfigurátor osztálynak nevezem. Az entitásosztály tartalmazza az entitás definícióját a mezőkkel, a hozzájuk tartozó getter és setter metódusokkal, valamint az objektum-relációs leképezést vezérlő annotációkat. Ezek az osztályok semmi olyan logikát nem tartalmazhatnak, ami bonyolultabb egy egyszerű mező beállításánál vagy lekérdezésénél. Ha bármely mező megváltoztatása vagy lekérdezése valamilyen összetettebb logikát igényel – például azért, hogy a modell konzisztenciáját fenntartsuk –, akkor az ilyen műveletek elérhetőségét a protected kulcsszóval korlátozni kell.

Valamennyi entitásobjektum a létrejöttétől fogva rendelkezik referenciával a konfigurátorára, amelynek szintén van referenciája az entitásra. A konfigurátor osztály az entitással egy csomagban található meg. A konfigurátor feladata, hogy minden olyan műveletet elvégezzen az entitás helyett, ami veszélyt jelent a modell konzisztenciájára nézve. Emiatt akár az entitás megfelelő példányosítása is a hatáskörébe utalható a factory method tervezési minta segítségével.

A konzisztencia szempontjából veszélyes műveletre jó példa egy modellelem eltávolítása a modellből. Ez számos kapcsolat elvágását teheti szükségessé a modellben, így az ilyen feladatokat a konfigurátorra kell bízni. Másként fogalmazva egy elem törlésének a modell egészére kiterjedő hatással lehet, és az ilyen műveleteket a konfigurátor osztályok tartalmazzák.

Az imént ismertetett rendszer előnye, hogy az entitás definíciója és a rajta végezhető műveletek külön fájlba kerülnek, ezáltal a kód átláthatóbbá válik. A kódrészek ilyen módon történő szeparálása egyértelműen rákényszeríti a fejlesztőt, hogy minden hívásnál végiggondolja, hogy az adott metódus meghívása milyen következményekkel járhat a modell egészére nézve.

4.3 Optimális bejárást kereső motor

Az optimális bejárást kereső motor feladata, hogy olyan bejárási utat állapítson meg az önkéntesek számára, amely lehetővé teszi valamennyi rögzítendő pozíciójú hely felkeresését minimális bejárandó úthossz mellett.

Ha sikerül közel méretarányos modellt készítenünk a *szelvény* útjairól, és fel tudjuk venni a nem egyenes utak töréspontjait a modellben, akkor reményünk van arra, hogy egészen kedvező hosszúságú utat találjunk a bejáráshoz. Ebben az esetben elég csak a kereszteződéseket és a töréspontokat (továbbiakban: pontok) felkeresni a mérés során, és eközben csak a feltétlen szükséges utakat érinteni, hiszen a pontok közti szakaszokat egyenesnek tekintjük.

Ha nem tudunk méretarányosan modellezni, de azért a töréspontokat ismerjük, akkor is használhatjuk az előbbi esetre érvényes módszert, bár ekkor a bejárás optimumot közelítő hossza nem biztosított.

Ha a modellünk nem tartalmazza a töréspontokat, akkor sajnos csak olyan terv készülhet, ami minden úton végigvezet minket, hiszen az élek alakjával kapcsolatban semmilyen előfeltevéssel sem élhetünk.

Összefoglalva tehát a keresőmotor külön-külön módszert kell, hogy biztosítson az ismert és az ismeretlen töréspontos esetre. A megfelelő módszert a felhasználó választhatja majd ki, de egyelőre csak a méretarányos tervezés támogatott.

4.3.1 Megoldás ismert töréspontos modell esetén: Hamilton-kör

Ahogy fejezet bevezetőjében jeleztem, az ismert töréspontok esetén alkalmazott módszer azt feltételezi, hogy a pontok között az utak egyenesek. Ebből következik, hogy ha minden pont helyét meghatározzuk a mérés során, akkor a bejárással nem érintett útszakaszok – lévén egyenes vonalúak – a végpontjaikkal meghatározásra kerülnek.

Az algoritmus feladata tehát az, hogy a pontok bejárására egy olyan sorrendet adjon, amellyel a bejárás hossza minimalizálható. Azonban ez a feladat lényegében egy utazóügynök (TSP) probléma [21], amely célja egy optimális, minden csomópontot érintő kör találása. Ennek a problémának a megoldása azonban nem triviális, hiszen a megoldást a gráf minimális súlyú Hamilton-körének megtalálása jelentené.

Ezzel az a probléma, hogy már annak az eldöntése is NP-teljes, hogy a gráfban van-e Hamilton-kör, így Karp-redukcióval belátható, hogy a minimális kör keresése is az. Továbbá abban sem lehetünk biztosak, hogy az úthálózat gráfja egyáltalán tartalmaz Hamilton-kört [22].

További rossz hír, hogy nem létezik olyan algoritmus sem, amely képes lenne az optimumot tetszőleges pontossággal megközelíteni, legalábbis ha elfogadjuk, hogy $P \neq NP$. Szerencsére azonban a probléma – mivel valóságos fizikai távolságokkal dolgozunk – az TSP euklideszi esete, amire léteznek közelítő algoritmusok [21].

Az általam alkalmazott megoldás kétfokozatú optimalizációt használ, mielőtt azonban ebbe belekezdene, biztosítja, hogy legyen Hamilton-kör a gráfban. Az megoldás keresésének első lépése – egy 2-approximációs algoritmus a metrikus TSP-re – szolgáltat számunkra egy Hamilton-kört, amely legfeljebb 2-szer hosszabb az optimálisnál. A második fokozat a Lin-Keringhan heurisztika [23] segítségével próbál javítani ezen a körön. Ez a heurisztika a gyakorlati tapasztalatok szerint egészen jól bevált [23], igaz nincs garancia arra, hogy az optimumhoz közeli értéked adjon. Az első fokozat miatt azonban a végeredmény nem lehet rosszabb az optimum kétszeresénél. Mindazonáltal ez a második fokozat a gyakorlatban jól vizsgázott, az általam kipróbált esetekben 20-30%-kal redukálta az első fokozat által talált kör hosszát.

4.3.1.1 2-optimális kör keresése [21]

Mielőtt az optimalizációhoz hozzákezdenénk, a gráfot teljes gráffá kell alakítani, hogy garantálva legyen a Hamilton-kör létezése [22]. A teljes gráf úgy áll elő, hogy az eredeti gráf által nem tartalmazott éleket is felvesszük. Egy ilyen hozzáadott i→j él hossza az (i,j) pontpár között futó legrövidebb út hossza. A gráf ilyen átalakítására kiválóan alkalmas a Floyd-algoritmus. Ennek melléktermékeként előáll mátrix, amely (i,j) cellájában annak a pontnak az azonosítója szerepel, amelyen keresztül vezet a legrövidebb i→j út. Ennek a mátrixnak később még jelentősége lesz, így nevezzük F-nek.

Ezután az így kapott gráfban keres a program egy minimális súlyú feszítőfát, amelyhez a mohó jellegű Prim-algoritmus [24] könnyen implementálható megoldást szolgáltat.

Ezután megduplázzuk a feszítőfa éleit, ami által minden pont páros fokszámú lesz, ami miatt létezik a duplázott élű gráfban Euler-kör, amelyet Fleury-algoritmusával [25] meg is kereshetünk. Ezután az így nyert Euler-kört speciális módon járjuk be. Ha olyan pontba mennénk, amelyben már jártunk, akkor megnézzük a következő indexű pontot, addig, amíg olyan pontot nem találunk, amelyet még nem látogattunk meg, vagy el nem fogynak a pontok. Ez utóbbi esetben az algoritmus véget ért. Ha találunk még meg nem látogatott pontot, akkor közvetlen oda megyünk (megtehetjük, hiszen teljes gráfon dolgozunk), és folytatjuk a futást.

Az így nyert bejárás egy Hamilton-kör (a teljessé tett gráfon), amely súlya legfeljebb az optimális kétszerese [21].

4.3.1.2 Heurisztikus javítás és utómunkák

Az előbb ismertetett módon megtalált kört tovább javíthatjuk a Lin-Keringhan heurisztika segítségével, amely minden lehetséges módon kiválaszt két nem szomszédos élet a körből, és törli őket, ami által a kör szétesik. Azután a kört úgy köti össze, hogy az első él kezdőpontját a másik él kezdőpontjával köti össze, és hasonlóan jár el a végpontokkal is.

Ha az így nyert kör kedvezőbb költségű az eredetinél, akkor megtartja, és a továbbiakban ezzel folytatja az optimalizációt.

A két fokozat lefutása után tehát van egy várhatóan elég kedvező hosszúságú Hamilton-kör a kezünkben, azonban ez a kör a teljes gráfon van értelmezve, így ahhoz, hogy a bejárásról szolgáltasson információt, vissza kell alakítani. Ez úgy történik, hogy

azokat az éleket, amelyek az eredeti gráfban nem voltak jelen, a végpontjaik között futó legrövidebb úttal kell helyettesíteni.

Szerencsére ez az út visszaállítható a korábban említett F mátrix felhasználásával, amely a Floyd-algoritmus melléktermékeként jött létre. Az i→j utólag beszúrt él átmegy az F mátrix (i,j) eleme által jelzett ponton, amit jelöljön k. Ez nyilván igaz i→k és k→j utakra is, így ezeket az F mátrix (i,k) illetve (j,k) eleme segítségével rekurzívan tovább bontjuk, amíg az út két végpontja nem egy szomszédos pontpár az eredeti gráfban. A rekurzív eljárás eredményeként előáll a bejárási terv.

4.4 Adatok átadása a mobilkliens számára

A tervezőalkalmazás segítségével megadhatjuk az úthálózat vázlatát, és minden fontos információt megadhatunk a bejárandó területről. Miután végeztünk az adatrögzítéssel, és megterveztük a bejárást, a bejárási tervet, és minden szükséges adatot a mobilkliens rendelkezésére kell bocsátani.

Az adatformátum meghatározásánál tekintettel kellett lenni arra, hogy a mobilkliens, amely feldolgozza ezeket az adatokat, korlátozott számítási kapacitással bír. Jelenleg még nincs rá lehetőség, de a későbbiekben szeretném, ha a tervek interneten keresztül is letölthetőek lennének a telefonra, ezért a fájlok méretét is célszerű korlátozni. A harmadik szempont pedig a fájl tartalmának egyszerű feldolgozhatósága volt.

A legfontosabb információ a bejárás során, hogy milyen sorrendben szeretnénk bejárni a területen elhelyezkedő kereszteződéseket, hogy a bejárást végző személyt a mobilalkalmazás egyértelmű utasításokkal tudja ellátni. Ez csak látszólag könnyű, hiszen minden olyan modellezési adat megadása, amelyek alapján könnyen meg lehetne ezt tenni (pl.: utcanevek), opcionális. Az egyetlen információ, ami biztos rendelkezésre áll, és amire támaszkodni lehet az a kereszteződésekbe befutó utak topológiája. Ez az oka, amiért a kereszteződés termináljai sorszámozottak. Ezek a sorszámok fejezik ki, hogy melyik él melyikkel szomszédos. A rajzolt topológia alapján – amelyről feltétezhetjük, hogy nagyjából megfelel a valóságnak – még annyi segítséget is lehet adni a felhasználónak, hogy égtáj szerint melyik irányba folytassa az útját.

A bejárási terv lépések sorozatából áll. A lépések a **`PlanItem`** osztályból származnak. Jelenleg két lépéstípust különbözetek meg. Az egyik típusa, a **`CrossroadPlanItem`**, amely a felhasználót egy kereszteződés meglátogatására utasítja. Ez tartalmazza, hogy melyik kereszteződést melyik terminálján keresztül

közelítse meg, és melyik terminálján keresztül folytassa az útját. Ezen kívül még tárolja, hogy a kereszteződést milyen hosszú úton tudja az önkéntes elérni az azt megelőzőből.

A másik bejárási lépés típus a **`BreakpointPlanItem`**, amely bevezetését a korábban részletezett töréspontok jelenléte tette szükségessé. A tervezési algoritmus által szolgáltatott eredményben előfordulhat, hogy egy 2-fokszámú ponton megfordul a bejárás, és visszalép az előző pontra, azonban ha ez éppen egy töréspontra esik, akkor ahhoz nem lehetséges kereszteződést rendelni. Az ilyen esetek jelzésére szolgál a **`BreakpointPlanItem`**. Az ilyen pontokat úgy kell felvenni a terepen, hogy a legutóbb meglátogatott kereszteződésből még éppen látható pont elérésekor kell megfordulni.

A bejárási tervnek az itt felsorolt elemei hivatkoznak kereszteződésekre és terminálokra is. Továbbá azt is szeretnénk, hogy a felhasználók az utak adatait is láthassák vagy rögzíthessék, vagyis az utak adatait is továbbítani kell a mobilkliens számára. Emiatt tehát szükség van az utak és a kereszteződések, illetve az alárendelt elemeik sorosítására is.

A könnyebb feldolgozhatóság, és a referenciák feloldhatósága miatt a tervet, és a területről rendelkezésre álló információkat leíró fájlok a projekt könyvtárstruktúrájában a `plan` mappába kerülnek. Minden terv négy fájlt tartalmaz. Az `address.dat` a területen megtalálható utcaneveket tárolja, a `plan.dat` a bejárást leíró **`PlanItem`** szekvencia sorosításával áll elő, a `crossroads.dat` a *kereszteződés*ek, míg a `roads.dat` az *ut*ak adatait tartalmazza. A fájlok EBNF szintaxisa megtalálható a függelékek (6.1) között.

Az adatok exportálása úgy van implementálva, hogy minden modellelem konfigurátor osztálya, amelyek az **`AbstractSpatialConfigurator<T>`** osztály példányai, megvalósítják az **`Exportable<T>`** interfész által előírt két metódust. A `void exportToPlan(int level, PrintWriter printWriter)` segítségével lehet a modellelemet és az alárendeltjeit exportálni, míg a `T importFromPlan(List<String> lines)` metódussal lehet a modellelemet a szöveges leírásából visszanyerni.

A modellelemek az export és az import során egyaránt sorosítják, illetve visszaállítják az alárendelt modellelemeiket (pl.: egy kereszteződés a termináljait). Ehhez az alárendelt elemek konfigurátorának is **`Exportable<T>`**-nek kell lennie.

A projekt teljes exportálását a korábban ismertetett **ProjectController** osztály `void exportProject()` metódusa segítségével végezhetjük el. Ez létrehoz egy **ProjectExportController**-t, amely az export teljes folyamatának levezénylését végzi. Először menti a projektet, hogy minden modellelem egyedi azonosítót kapjon, és az exportált terv az adatbázis állapotát tükrözze. Ezután elvégzi a bejárás tervezését, aminek eredménye a bejárási terv, ami szintén mentésre kerül. Ez az osztály készíti el a többi fájlt is. Minden fájl írását egy-egy **FileGenerator** végzi, fájlonként a megfelelő leszármazott típus egy-egy példánya.

Az adatok importálása a **FileParser** osztályok megfelelő leszármazottjainak segítségével történik. Ezek az osztályok beolvassák a sorokat, és gondoskodnak róla, hogy a modellelemet és az alárendeltjeit leíró sorok együtt legyenek feldolgozva. Ez úgy történik, hogy a konfigurátort példányosítjuk meg először, amely később feltölti az általa kezelt modellelem-példányt tartalommal a megfelelő sorok alapján.

Az adatok importálásakor nagyon fontos, hogy a modellelemek közti referenciákat helyesen oldjuk fel. Ahogy az a fájlok definíciójából (6.1) is látszik, a hivatkozás mindig az egyedi azonosító (ID) alapján történik. A rendszernek tehát biztosítani egy komponenst, ami képes a típus és az ID ismeretében a megfelelő példányt szolgáltatni. Azonban ennek a komponensnek az implementációja platformonként változhat az adott platformon alkalmazott tárolási megoldás függvényében. Az előbb vázolt komponenst az **EntityResolver** interfész írja le, amelyet az **EntityResolverFactory**-tól kérhetünk el. Az **EntityResolverFactory** osztály a betöltése során egy vele azonos csomagban lévő konfigurációs fájlból olvassa fel annak az **EntityResoverLocator**-nak a nevét, amelyet példányosít, hogy ennek segítségével **EntityResolver** példányokat szolgáltasson a `getEntityResolver()` nevű statikus metódusát hívó klienseinek. A tervezőalkalmazásban az **EntityResolverLocator** mindig az aktuálisan előtérben lévő projekt **OrphanItemCollector** osztályát adja vissza, amely implementálja az **EntityResolver** interfészt. Ez az osztály DAO osztályok felhasználásával kapcsolódik az adatbázishoz, ahonnan az adatokat nyeri.

4.5 A rögzített adatok importálása

Az alkalmazás másik fő feladata a modellezés mellett az, hogy képes legyen a *kereszteződés*ek helyét és az *ut*ak alakját rögzíteni. Az *ut*ak kis egyenes szakaszokból,

az úgynevezett *szegmen*sekből állnak. Ha az *út* kanyargós, akkor több, ha viszonylag egyenes, akkor kevesebb is elég lehet ezekből az elemekből. Nyilván nem járható az a megközelítés, hogy naplófájlban rögzített minden szomszédos pontpárhoz létrehozzon az alkalmazás egy-egy *szegmen*st. A legalább másodperces gyakorisággal rögzített pontok miatt ez rövidesen kezelhetetlen mennyiségű adathoz vezetne. Emiatt szükség van arra, hogy tömörítsük az adatokat úgy, hogy a létrejövő *szegmen*sek még elfogadható hibával közelítsék az *út* alakját, de a számuk is kellően alacsony legyen.

A pontok értelmezésekor sajnos annak is tudatában kell lennünk, hogy a mérésünk hibákkal terhelt, tehát a feladat az, hogy következtessünk az út valódi alakjára a felvett pontok helye alapján. A mérés során fellépő hibát tekinthetjük normális eloszlásúnak, és feltehetjük, hogy az egyes pontok mérési hibái függetlenek, legalábbis az itt bemutatott konkrét alkalmazás céljai szempontjából [24]. Az importálás során célszerű, ha ki tudjuk szűrni azokat a pontokat, amelyek pontatlansága már elfogadhatatlan. Ilyenek lehetnek az egy bizonyos DOP értéknél magasabb értékkel bíró pontok, vagy azok, amelyeknél a műholdak száma elégtelen. Emellett az import még vizsgálja azt is, hogy az utolsó mért pozíció és sebesség alapján (egy meghatározott tűréssel) a vevő eljuthatott-e a következő mért pozícióba. Azokat a pontokat, amelyek fennakadnak a szűrőn, ki kell hagyni a további számításokból. Azonban ha egy *út*hoz tartozó mért pontok közül sokat el kell dobni, akkor előfordulhat, hogy nem is lehetséges az *út* alakját megfelelően közelíteni a megmaradó pontok felhasználásával. Lehetőség van tehát annak a megadására is, hogy az egyes utakhoz tartozó pontok hány százalékának kell érvényesnek lennie, hogy az *út*ra adott közelítést elfogadjuk.

Összefoglalva a követelményeket, tehát szükség van egy megoldásra, hogy interpolálhassuk az *út* alakját a mérések alapján egy törtvonallal, ami elég kis hibával közelíti a valóságot, ugyanakkor képes jelentősen csökkenteni az interpolált alakzat tárolásához szükséges *szegmen*sek számát.

4.5.1 Röviden a vetületekről

A következő szakaszokban ismertetett algoritmusok számtalan geometriai számítást alkalmaznak egyebek mellett a távolságok meghatározására. Ezek a számítások meglehetősen körülményesen lennének elvégezhetőek a szélesség és hosszúság koordinátákkal, amelyeket ráadásul nem gömbi, hanem egy ellipszoid koordinátarendszerben kell értelmezni. Sokkal kényelmesebben lehet műveletek végezni egy hagyományos derékszögű koordinátarendszerben.

Ahhoz, hogy a pontok koordinátái ilyen formában is a rendelkezésünkre álljanak, vetületet kell képezni. Az alkalmazás a vetületek közül a traverzális Mercator (TM) vetületet használja. Ez egy hengervetület, amely esetében a henger tengelye az egyenlítő síkjára illeszkedik, és a hengerpalást egy meridiánjánál érinti a Földet. A vetületet széles körben használják a térképészetben (UTM) is. A vetület megalkotásánál kulcsfontosságú az érintési meridián meghatározása, hiszen ettől távolodva a vetület torzítása növekszik. Ennek a vetületi családnak a tagjait, amennyiben ellipszoidra, mint a geoid közelítésére alkalmazzuk, szokás még Gauss-Krüger vetületnek is nevezni.

Az alkalmazásban minden ponthalmaznak a középső meridiánját használom középső meridiánként. A koordinátákat, a NATO UTM-hez hasonlóan méterben szolgáltatja a transzformáció, ami a WGS-84 által megadott geoid paramétereit veszi alapul. Az easting a középmeridiántól keleti irányba vett távolság (nyugati irányban negatív), míg a northing az egyenlítőtől északi irányban vett előjeles távolság. A számítások elvégzése után a TM vetületen megadott koordinátákat visszatranszformálom szélesség és hosszúság értékekre, és így tárolom az adatbázisban.

A koordináták közötti transzformációt a **`CoordinateFactory`** osztály metódusai végzik. A TM vetületi koordinátákat a **`CustomTM`** osztály reprezentálja.

4.5.2 Az út interpolálása

Az utak interpolálására szánt algoritmus arra a heurisztikára épít, hogy a valódi útvonalunkat jobbról és balról „szegélyezik” a mért pontok, tehát az út valahol „közöttük” húzódik. Felfoghatjuk ezt úgy is, hogy minden mért pont hibáját felbontjuk egy a méréskori mozgásunk irányvektorára merőleges és egy vele párhuzamos komponensre. A merőleges komponens eltávolít minket az úttól, míg a párhuzamos a pontok sűrűsödését illetve ritkulását idézi elő az út mentén. Ez utóbbi azonban veszélyes, mert a mért pontok átsorrendeződését okozhatja, ami az utakon nem kívánt hurkokat eredményezhet. Ennek kiküszöbölését a szegmensek számát csökkentő algoritmusra bíztam.

Az út alakját egy harmadfokú NURBS görbével közelíti a program. A NURBS görbe vezérlőpontjai a mért pontok, amelyek súlya a DOP értékükkel fordítottan arányos, így biztosítva, hogy a pontok közül a pontosabbak nagyobb hangsúllyal legyenek figyelembe véve. A görbe csomóvektorát úgy határoztam meg, hogy a görbe átmenjen a végpontjain. Ezáltal az egy kereszteződéshez csatlakozó utak egy pontban

fognak találkozni. Az utat tehát egy NURBS görbe írja le, amelynek valamennyi pontja a vezérlőpontok konvex burkán belül található.

4.5.3 Szegmensszám csökkentése

A szegmensek számát úgy kívánja minimalizálni az algoritmus, hogy a NURBS görbét egy törtvonallal helyettesíti, amely kellően sok töréspontot tartalmaz. Ezután az algoritmus minden futamban megpróbálja összevonni a szomszédos szakaszokat, amit akkor tehet meg, ha a szakaszok közös pontja nincs egy előre meghatározott távolságnál messzebb a nem közös pontokra illesztett szakasztól, ami az összevonás eredménye is egyben, ha az megtehető. Az algoritmus addig ismétli ezeket a futamokat, amíg azok csökkentik a szakaszok számát.

Az eredményben megmaradt szakaszok mindegyikéhez létrejön egy-egy *szegmens*, amelyeket az importálást végző programrészlet a megfelelő *út*hoz rendel. A tapasztalataim alapján ez az algoritmus eredményesen csökkenti a tárolandó szakaszok számát még viszonylag alacsony megadott összevonhatósági távolság esetén is.

5 A mobilkliens

A térképépítő rendszer következő komponense egy mobilkliens, amely a modellező alkalmazástól kapott bejárási terv alapján végigvezeti a felhasználót a bejárandó területen. Eközben kéri a felhasználót, hogy a hiányzó adatokat a megfelelő helyen adja meg (pl.: tömbsarki házszámok), illetve jelezze, ha a soron következő objektumot elérte (kereszteződés, terminálok kezdőpontja, stb…).

A mobilklienst Android platformon készítettem el, ami mellett a Java nyelvű fejlesztés és a platform növekvő népszerűsége szólt.

A kliens három adatforrást kell, hogy kezeljen úgy, hogy az ezekből származó adatokat egy log állományba írja. Az első adatforrás egy külső GPS vevő, amelyhez Bluetooth-on keresztül kapcsolódik a készülék. A másik adatforrás a bejárási terv, amely meghatározza a bejárás menetét, és tartalmazza azokat az információkat az objektumokról, amelyeket a modellezőalkalmazással vettünk föl. A harmadik adatforrás pedig a felhasználó, aki az alkalmazás felületén keresztül viszi be az információkat.

Az alábbiakban a kliens működésének részleteit bottom-up megközelítésben ismertetem: Az alsó szintű komponensek ismertetését követően térek ki ezek összehangolásának megvalósítására.

5.1 Kapcsolat a külső GPS vevővel

A külső GPS vevő használatát a telefonok beépített képességeivel szemben azért részesítem előnyben, mert így lényegesen több információt kaphatok a mérés körülményeiről. Erre azért van szükség, mert így azonosítható a mérési pontatlanságok oka, és ez által lehetőség nyílik azok javítására.

5.1.1 NMEA szabvány és protokoll [26]

A GPS vevők és más hajókon található műszerek számára létezik egy szabványos interfész és kommunikációs protokoll, amelyet az **NMEA (National Marine Electronics Association)** nevű szervezet gondoz, és az aktuális verziója a 0183-as számot viseli.

Sajnos az említett szabvány nem nyilvános, térítés ellenében érhető el az NMEA honlapján keresztül. A protokoll által szolgáltatott adatok feldolgozását Dale DePriest

útmutatója alapján készítettem el, akinek sikerült a szabvány jelentős részét visszafejteni. Mivel az információk nem hivatalos forrásból származnak, ezért a fejlesztés során külön figyelmet fordítottam a leírásban található információk ellenőrzésére.

A szabvány elektronikai része az EIA-422 kompatibilitást írja elő. A szabvány főbb előírásai a következők: adatátviteli sebesség 4800b/s, 8 adatbit, nincs paritásbit, 1 stopbit. Az NMEA 0183 egy ASCII karakteres protokollt definiál az adatok átvitelére. A protokoll mondatokra tagolódik, minden mondat egy $ jellel kezdődik és egy *-ot követő, 2 hexadecimális számjegyből álló, ellenőrzőösszeggel végződik. Minden mondat több adatelemet tartalmaz, amelyeket vessző választ el. Az USB illetve Bluetooth felett kommunikáló eszközök is ezt a protokollt emulálják.

A mondatok kezdetén mindig egy szó áll, ami meghatározza a kibocsátó készülék típusát és a mondat fajtáját. A GPS vevőkhöz tartozó mondatok kezdőszava GP-vel kezdődik Az egyes mondatfajták különböző információkat hordoznak. A fontosabb mondattípusok jellemzőit a következő táblázat foglalja össze. A dőlttel szedett mondattípusok jelenleg még nem támogatottak.

Azonosító	Név	Tartalma
GPGGA	Fix adatok	pozíció, mérés ideje, műholdak száma, HDOP, magasság adatok, mérés minősége (2D, 3D, stb...)
GPGLL	Pozíció	pozíció, mérés ideje
GPGSA	Műhold státusz	felhasznált műholdak listája, HDOP, VDOP, PDOP, 3D fix
GPGSV	Részletes műhold státusz	műholdak száma, műhold azonosítója, eleváció és azimuth (4 műholdra)
GPRMC	Ajánlott mondat	pozíció, mérés ideje (dátum is), sebesség, irányszög, mágneses elhajlás
VTG	Sebesség	sebesség, irányszög

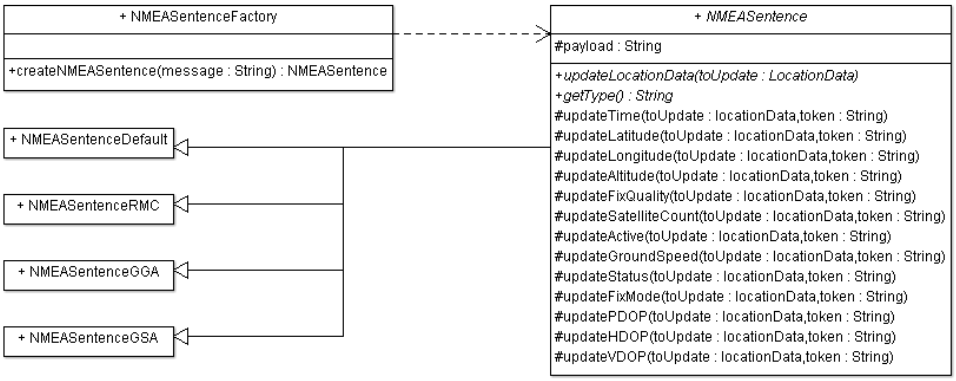
5-1. táblázat Fontosabb GPS specifikus NMEA mondatok

5.1.2 NMEA adatfolyam feldolgozása

A protokollon keresztül érkező adatok feldolgozására egy saját osztálykönyvtárat terveztem és implementáltam, amely részét képezi az alkalmazásomnak. Ezt a csomagot úgy terveztem meg, hogy nem Android alapú Java alkalmazásokban is használható legyen.

A feldolgozó központi komponense az **`NMEAReaderThread`**. Ez szálként van implementálva és egy **`InputStream`**-hez csatlakozik, amin keresztül a protokoll szerinti adatfolyam érkezik. A szál olvassa az adatfolyamot, és feldarabolja mondatokra. A mondatok karakterláncait, a kezdő $ jeltől megfosztva az

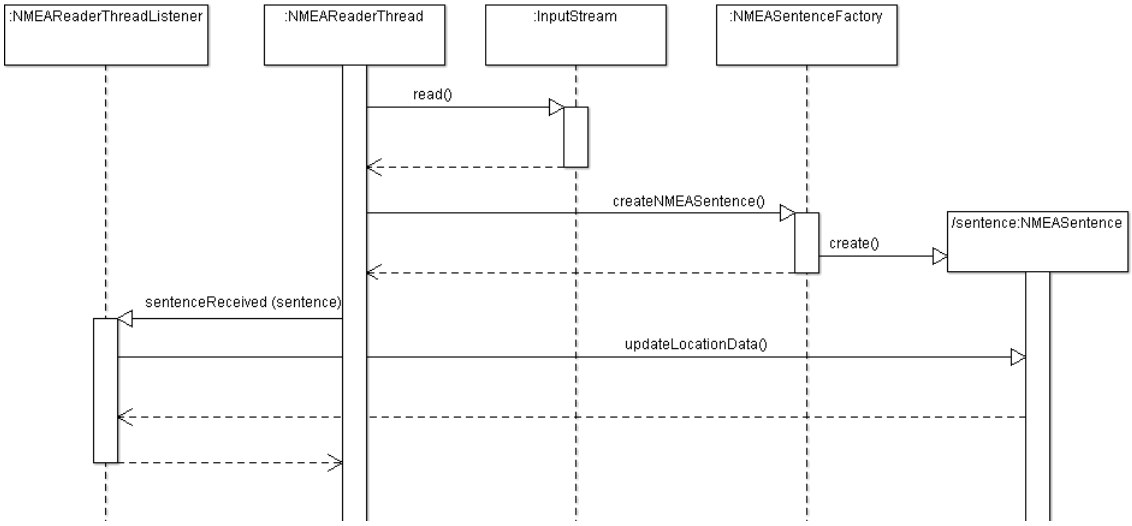
NMEASentenceFactory `createNMEASentence(String sentence)` metódusának adja át. Ez a metódus a szöveg alapján elkészíti a megfelelő mondatfajtát reprezentáló **NMEASentence** példányt. Az osztálydiagramon látható, hogy valamennyi fajtájú NMEA mondathoz, amit a csomag kezel, saját **NMEASentence**-ből származó típus van definiálva. A factory számára ismeretlen mondatok **DefaultNMEASentence** típusra képződnek le.



5-1NMEA mondtattípusok

Az **NMEASentence** egy absztrakt osztály, amely legfontosabb absztrakt metódusa a `updateLocationData(LocationData toUpdate)`. Ennek a metódusnak a feladata, hogy a paraméterként kapott **LocationData** objektum állapotát frissítse. Ezt a metódust – lévén absztrakt – minden mondattípus egyedileg implementálja, így minden mondatfajta csak azt a részét frissíti a **LocationData** állapotának, amelyről információval rendelkezik. Az **NMEASentence** osztály támogatást nyújt erre a frissítésre. A **LocationData** valamennyi állapot-attribútumára van egy-egy metódusa (update kezdetűek). Ezek egy-egy **String**-et várnak paraméterként, elvégzik a szükséges típus- és egyéb konverziókat, majd frissítik a megfelelő attribútumot. Így tehát az **NMEASentence** leszármazottjainak csak annyi kell tudniuk, hogy az egyes adatelemeknek mi a pozíciója az általuk reprezentált mondaton belül a protokollban.

A **LocationData** frissítését azonban már nem az értelmező csomag végzi, hanem rábízza az őt használó kódra, hogy mit kezd a kiolvasott mondatokkal. Ezt úgy értem el, hogy minden alkalommal, amikor egy-egy mondat összeállítása befejeződik, vagy a feldolgozó szál állapota megváltozik, az **NMEAReaderThread** egy értesítést küld egy **NMEAReaderThreadListener**-nek.



5-2. ábra Az NMEA értelmező csomag működése

5.2 Naplófájl készítése

A bejárás során a felhasználó által bevitt adatok egy naplófájlba kerülnek. A naplófájl formátumának (5.2.2) könnyen értelmezhetőnek kell lennie, hiszen a benne tárolt adatokat be kell tölteni a modellező alkalmazás segítségével a *szelvény*be.

A naplózás eseményalapon történik. Ha felhasználó megad egy adatot (pl.: tömbsarki házszámozás), vagy megjelöli egy objektum helyét, amire az alkalmazás kéri, akkor ebből egy esemény generálódik, ami bekerül a naplóba a megadott adatokkal együtt.

5.2.1 Események naplózása

Az események naplózásáért egy **`LogWriterThread`** nevű osztály felel, amely szálként van implementálva. Ennek a szálnak van egy postaládája, egy **`Queue`**, ahová az **`AbstractLogItem`** típusú naplóelemeket várja. Kezdetben a szál alszik. Amikor felébred, ellenőrzi a bementi sort, és ha az nem üres, akkor a benne található elemeket egy **`BufferedWriter`** segítségével egy meghatározott helyen lévő fájlba kiírja. A sor kiürítése után a **`BufferedWriter`** `flush()` hívásával kényszeríti minden esetleg pufferelt adat fájlba írását. Ezután a szál elalszik. Amikor felébred, ismét ellenőrzi a bejövő sort, és a kör kezdődik elölről.

Látható, hogy a sorhoz való hozzáférés több szálról történhet, ezért az implementáció során a kölcsönös kizárást biztosítani kellett erre az objektumra.

Az **AbstractLogItem** osztály szolgál valamennyi típusú naplóelemet reprezentáló osztály őséül. A naplóelemek saját maguk képesek a bennük tárolt adatokat sorosítani a `serialize()` metódus segítségével.

5.2.2 Az eseménytípusok és a naplófájl formátuma

A jelenlegi implementációban mobilkliens két eseménytípust ismer. A **PositionLogItem** akkor jön létre, ha a kliens rögzíti egy pont adatait. A **PlanLogItem** pedig akkor keletkezik, ha a felhasználó egy töréspont vagy egy kereszteződés rögzítését jelzi. A **PlanLogItem** a **PositionLogItem** leszármazottja, és egyetlen új mezője van, amelyben annak a **PlanItem** id-jét tárolja, amely utasítására létrejött.

A fájl formátumát a függelékben adtam meg (ld.: 6.1.6).

5.3 A komponensek együttműködése

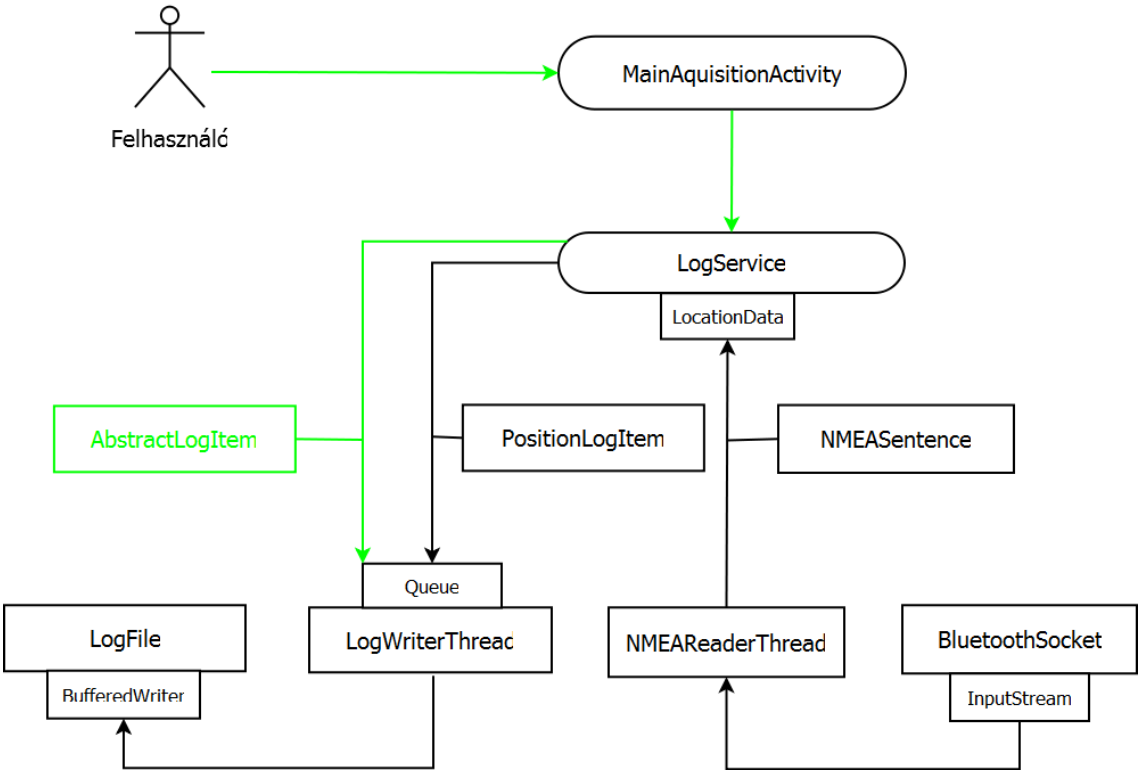
Az alkalmazás alapkomponenseinek ismertetése után szeretném felvázolni, hogy miként zajlik ezeknek az egységeknek az együttműködése, különös tekintettel arra, hogy az adatok hogyan áramlanak a szoftver egyeses komponensei között.

Szerettem volna elérni, hogy az alkalmazásból való kilépés, vagy az alkalmazás háttérbe kerülése (pl.: bejövő hívás) miatt a naplózás ne álljon le. Ezért a naplózásért egy **LogService** nevű komponens a felelős. Ez egy Android szolgáltatás, amelyet az alkalmazás indít. Ez a komponens mindaddig fut, amíg kifejezetten nem utasítjuk leállásra, vagy amíg a készülék Bluetooth adóvevője bekapcsolt állapotban van. A Bluetooth állapotváltozásáról az Android broadcast üzenetben értesíti a szolgáltatást.

Az alkalmazás képernyőit – az Android alkalmazásoknál szokásos módon – **Activity**-k valósítják meg. Az 5-3. ábra foglalja össze a kliens működésének részleteit. A fekete nyilak a GPS vevőből, a zöldek pedig a felhasználói felületről érkező adatok útját jelölik. Az indítóképernyő egyben a fő adatrögzítés-vezérlő képernyő is, amit **MainAquisitionActivity**-nek hívnak. Ez indítja a **LogService**-t, ha még nem futna, és miután elindította, hozzákapcsolódik (bind), majd a továbbiakban ezen a kapcsolaton keresztül vezérli annak működését.

Ilyen vezérlési esemény a külső GPS vevő kiválasztása, amelynek címét a `setBtAddress(String address)` metóduson keresztül közöljük a

LogService-zel. Ennek hatására a naplózó szolgáltatás csatlakozik a készülékhez, majd elindítja az NMEA adatfolyamot olvasó (5.1.2) és naplót író szálat (5.2.1).



5-3. ábra Adatáramlás a mobilkliensben

A **LogService** implementálja az **NMEAReaderThreadListener** interfészt, és tárol egy **LocationData** példányt, amelyet a beérkező mondatok felhasználásával frissít. Az aktuális pozíció természetesen a szolgáltatás valamennyi kliense, így a főképernyő számára is elérhető, akik bizonyos időnként lekérdezhetik. A pozícióról bizonyos időközönként egy pillanatképet készít a **LogService**, amelyet egy **PositionLogItem**-be csomagol, és behelyezi a **LogWriterThread** bemeneti sorába.

A **LogService** biztosít metódusokat a legfontosabb események jelzésére is. Egy ilyen metódus meghívása után a szolgáltatás előállít egy, az eseménynek megfelelő **AbstractLogItem** példányt, és ezt küldi el a naplóíró szálnak.

5.4 Tervadatok importálása

Az Android alkalmazás képes fogadni a tervezőalkalmazás által exportált adatokat, hogy azok alapján utasítsa a felhasználót a megfelelő helyek felkeresésére és az utak rögzítésére. A modell és a hozzá tartozó logika legnagyobb részére ezen a

platformon is szükség van, ezért a megfelelő osztályokat egy osztálykönyvtárba csomagoltam, és így osztottam meg a mobilkészüléken futó alkalmazás és a tervezőprogram között. Az importálás tehát szinte teljesen megegyezik a korábban leírtakkal (4.4), a különbség az entitások azonosítóját feloldó **`EntityResolver`** példányosításában keresendő.

Az Android platformon egy terv betöltése során az alkalmazás által kezelt memóriaterületre töltődnek be az adatok, egy **`SpatialDataStore`** típusú objektumba. Az **`EntityResolverLocator`** platform specifikus implementációja ezt használja **`EntityResolver`**-ként, így minden kérés ehhez az objektumhoz fut be.

6 Függelék

6.1 Adatfájlok szintaxisa

6.1.1 Közös definíciók

```
nzdigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
int = (nzdigit, {0 | nzdigit}) | "0"; (* egész szám típus (int/long/short) *)
nullable_int = "" | int
uchar = ? any unicode character ?
nl = ? line break ?
boolean = "true" | "false"
float = int, {"0" | nzdigit}
```

6.1.2 address.dat

```
street_id        = int (* egyedi azonosító *)
postal_code      = nullable_int
postal_code_odd  = postal_code (* páratlan oldali irányítószám *)
postal_code_even = postal_code (* páros oldali irányítószám *)
begin_number     = nullable_int      (* utca legkisebb házszáma *)
end_number       = nullable_int      (* utca legnagyobb házszáma *)

street = "STREET(", street_id , ",", street_name, ",", postal_code_odd, ",",
postal_code_even, ",", begin_number, ",", end_number, ")", lb
address_file = {street}
```

6.1.3 plan.dat

```
crossroad_id       = int (* felkeresendő kereszteződés ID-je *)
arrive_terminal_id = nullable_int (* kereszteződésbe bevezető terminál ID-je
*)
leave_terminal_id  = nullable_int (* kereszteződésbõl kivezető terminál ID-je
*)
road_length        = nullable_int (* az út hossza a kereszteződésig *)

crossroad_item  =  "CROSS(",  crossroad_id,  ",",  arrive_terminal_id,  ",",
leave_terminal_id, ",", road_length, ")"
breakpoint_item = "TURN"
```

```
item = (crossroad_item | breakpoint_item), lb
plan_file = {item}
```

6.1.4 crossroads.dat

```
crossroad_id   = int (* felkeresendõ keresztezõdés ID-je *)
crossroad_type = ? CrossroadType enum literals ?
crossroad_name = {uchar}

lane_id      = int (* sáv ID-je *)
lane_pos     = "AFF_LEFT" | "AFF_RIGHT" | "EFF_LEFT" | "EFF_RIGHT" | "NORMAL"
lane_ordinal = int (* sáv pozíciója *)
lane_type    = ? LaneType enum literals ?
lane_dir     = ? Direction enum literals ?
lane_purpose = ? LanePurpose enum literals ?

lane = "LANE(", lane_id, ",", lane_pos, ",", lane_ordinal, ",", lane_type,
",", lane_dir, ",", lane_purpose, ")", lb;

compass = "N" |     "NNE" | "NE" | "ENE" | "E" | "ESE" | "SE" | "SSE" | "S" |
"SSW" | "SW" | "WSW" | "W" | "WNW" | "NW" | "NNW"
degree  = int

terminal_id      = int (* terminál ID-je *)
terminal_ordinal = int (* terminál pozíciója *)
road_id          = int (* terminálba futó út ID-je *)
road_dir         = compass, degree (* út égtáj *)
terminal = "TERM(", , ")", lb, {lane}

crossroad = "CROSS(", crossroad_id, ",", crossroad_type, ",", crossroad_name,
")", lb, {terminal}
crossroad_file = {crossroad}
```

6.1.5 road.dat

```
road_id    = int (* az út ID-je *)
street_id (* ld. address.dat *)
road_type  = ? RoadType enum literals ?
road_cfg   = ? RoadConfiguration enum literals ?
road_paved = boolean (* szilárd burkolatú-e *)
```

```
road_res   = boolean (* lakó-pihenõ övezet-e *)

(* számozási paraméterek *)
odd_start    = nullable_int
odd_end      = nullable_int
even_start   = nullable_int
even_end     = nullable_int
single_start = nullable_int
single_end   = nullable_int

numeration = "NUM(", odd_start, ";", odd_end, ";", even_start, ";", even_end,
";", single_start, ";", single_end, ")";

lane (* ld. crossroads.dat *)

road = "ROAD(", road_id, ",", street_id, ",", road_type, ",", road_cfg, ",",
numeration, ",", road_paved, ",", road_res ")", {lane}, lb
road_file = {road}
```

6.1.6 Log file

```
time = int (* rögzítés ideje *)
lat  = float (* szélesség *)
lat_comp = "N" | "S"
lon  = float (* hosszúság *)
lon_comp = "E" | "W"
alt  = float (* magasság *)
ground_speed = float (* sebesség *)
hdop = float
vdop = float

loc = "LOC(" + time, ";", lat_comp, ";", lat, ";", lon_comp, ";", lon, ";",
alt, ";", groud_speed, ";", hdop, ";", vdop, ")";
crossroad_log = "CROSS(", plan_item_id", ",", loc)";

item = (loc | crossroad_log), lb
log_file = {item}
```

Ábrajegyzék

Irodalomjegyzék

[1] Bányai L., Borza T., Busics Gy., Kenyeres A., Krauter A., Takács B.. Ádám J., Műholdas helymeghatározás. Budapest, Magyarország: Műegyetemi Kiadó, 2004.

[2] Sinodefence. (2010, June) Sinodefence - CNSS. [Online]. http://www.sinodefence.com/space/spacecraft/beidou2.asp

[3] Andrews Space and Technology. (2001) GLONASS Summary. [Online]. http://www.spaceandtech.com/spacedata/constellations/glonass_consum.shtml

[4] European Space Agency. (2010, May) ESA Galileo. [Online]. http://www.esa.int/esaNA/GGGMX650NDC_galileo_0.html

[5] United States Department of Commerce. (2011) Space Commerce - GPS modernization. [Online]. http://www.space.commerce.gov/gps/modernization.shtml

[6] United States Naval Observatory. (1996, Apr.) Block I satellite information. [Online]. ftp://tycho.usno.navy.mil/pub/gps/gpsb1.txt

[7] United States Naval Observatory. (2010, Aug.) Block II satellite information. [Online]. "ftp://tycho.usno.navy.mil/pub/gps/gpsb2.txt" ftp://tycho.usno.navy.mil/pub/gps/gpsb2.txt

[8] Space-based navigation and timing - National executive committee. (2007) Information about selective availability. [Online]. "http://www.pnt.gov/public/sa/" http://www.pnt.gov/public/sa/

[9] Elliot D. Kaplan and Christopher J. Hegarty, Eds., Understanding GPS: principles and applications. Norwood, MA, USA: Artech House, Inc., 2006, ch. 8.2.

[10] Ahmed El-Rabbany, "Introduction to GPS: The global positioning system," in Introduction to GPS: The global positioning system. Norwood, MA, USA: Artech House, Inc., 2002, ch. 1., pp. 1-6.

[11] Ahmed El-Rabbany, "Introduction to GPS: The global positioning system," in Introduction to GPS: The global positioning system. Norwood, MA, USA: Artech House, Inc., 2002, ch. 1., pp. 1-6.

[12] United States Air Force. (2011, Oct.) 50th Space Wing factsheet. [Online]. http://www.schriever.af.mil/library/factsheets/factsheet.asp?id=3909

[13] Elliot D. Kaplan and Christopher J. Hegarty, Eds., Understanding GPS: principles and applications. Norwood, MA, USA: Artech House, Inc., 2006, ch. 2.

[14] Hodobay-Böröcz András, "A magyar felsőrendű hálózat helyzete és jövője," Geodézia és

kartográfia, Sep. 2001.

[15] Navstar GPS Joint Program Office. (2006, Mar.) Interface Specification IS-GPS-200, Revision D: Navstar GPS Space Segment/Navigation User Interfaces. [Online]. http://www.losangeles.af.mil/shared/media/document/AFD-070803-059.pdf

[16] Benjamin W. Remondi. (1985, Oct.) NOAA technical memorandums. [Online]. http://docs.lib.noaa.gov/rescue/cgs_tech_memorandum/NOAA%20%20Technical%20 Memorandum%20NOS%20NGS-43.pdf

[17] Gerhard Wübbena, Andreas Bagge, Gerald Boettcher, Martin Schmitz, and Peter Andree. (2001, Sep.) Geo++ GmbH. [Online]. http://www.geopp.de/download/ion2001-gnpom_p.pdf

[18] Richard B. Langley, "Dilution of precision," GPS World, pp. 52-59, May 1999.

[19] KVHM. (1994, May) 19/1994. (V. 31.) KHVM rendelet a közutak igazgatásáról. [Online]. http://www.complex.hu/jr/gen/hjegy_doc.cgi?docid=99400019.KHV

[20] KPM-BM. (1975, Feb.) 1/1975. (II. 5.) KPM-BM együttes rendelet. [Online]. http://net.jogtar.hu/jr/gen/hjegy_doc.cgi?docid=97500001.KPM&kif=k%F6zleked%E9s#xcel

[21] Tibor Jordán, András Recski, and Dávid Szeszlér, "Az utazó ügynök probléma," in Rendszeroptimalizálás. Budapest, Magyarország: Typotex, 2004, ch. 3.2.3, pp. 100-102.

[22] Y. Gyula Katona, András Recski, and Csaba Szabó, "NP-beli problémák," in A számítástudomány alapjai. Budapest, Magyarország: Typotex, 2006, ch. 3.5, pp. 96-103.

[23] Shen Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," Operations Research, pp. 498-516, Oct. 1973.

[24] Lajos Rónyai, Gábor Ivanyos, and Réka Szabó, "Prim módszere," in Algoritmusok. Budapest, Magyarország: Typotex, 2005, ch. 6.6.1, pp. 156-160.

[25] Joan M. Aldous and Robin J. Wilson, "Fleury's algorithm," in Graphs and applications: an introductory approach. UK: Springer-Verlag, 2004, ch. 9.1, pp. 202-204.

[26] Dale DePriest. (2011, Oct.) NMEA data. [Online]. http://www.gpsinformation.org/dale/nmea.htm

[27] Christian Tiberius, Niels Jonkman, and Frank Kenselaar, "The Stochastics of GPS Observables," GPS World, pp. 49-54, Feb. 1999.