Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Softvér na organizáciu pretekov v orientačnom behu

(bakalárska práca)

Lukáš Poláček

Odbor: informatika
Vedúci: Mgr. Michal Forišek

Bratislava, 2007

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Poďakovanie

Chcel by som sa poďakovať vedúcemu mojej bakalárskej práce Mgr. Michalovi Foriškovi za cenné rady, ktoré mi pomohli pri tvorbe práce.

Ďalej by som sa rád poďakoval celej svojej rodine za všetko, čo pre mňa urobili. A ďakujem všetkým kamarátom za rozptýlenie v čase najväčšieho stresu a nielen počas neho.

Abstrakt

Orientačný beh je šport náročný na organizáciu, preto vyžaduje pomoc počítača. Existujúce aplikácie nespĺňajú náročné kritéria kvality a sú zastaralé. V práci analyzujeme, aké funkcie od aplikácie požadujeme. Navrhujeme architektúru aplikácie a technológie, ktoré použijeme. Na priloženom CD sa nachádza samotná implementácia aplikácie, ktorá postačuje na organizáciu menších pretekov od tvorby tratí až po spracovanie výsledkov.

Kľúčové slová: informatika, orientačný beh.

Obsah

Kapitola 1

Úvod

Orientačný beh je šport, v ktorom sa pretekár snaží prejsť v čo najkratšom čase označenými miestami v teréne – kontrolnými stanovišťami. Pohybuje sa len pomocou mapy a buzoly, pričom v mape má zaznačené kontrolné stanovištia ([POB]). Je to šport náročný na organizáciu. Organizátor musí rozmiestniť kontrolné stanovištia v lese, pripraviť mapu a nakoniec usporiadať samotné preteky. V tejto práci sa zameriame na fázu príprav tratí pre tlač a organizáciu pretekov.

V druhej kapitole si vysvetlíme, v čom spočívajú nevýhody existujúcich aplikácií. V tretej kapitole rozoberieme vlastnosti aplikácie, ktorú budeme implementovať. Podrobnejšie sa pozrieme na všetky vlastnosti, ktoré by mala aplikácia mať. Vo štvrtej kapitole sa budeme venovať samotnej implementácií. Popíšeme rozdelenie aplikácie do modulov a ako medzi sebou spolupracujú. Popíšeme si použité technológie a súborový formát pre našu aplikáciu.

Kapitola 2

Cieľ práce

Pre orientačný beh bolo vytvorených mnoho softvérových produktov. Najznámejší program OCAD, určený na tvorbu máp, sa presadil aj mimo orientačných športov a používa sa aj na iné účely. Ďalej existujú softvérové produkty, ktoré sú určené na organizáciu samotných pretekov. Tieto aplikácie väčšinou vedia len vyhodnotiť výsledky a nedajú sa v nich stavať trate. Ďalšie negatívne vlastnosti aplikácií sú:

1. Aplikácie používajú zastaralé technológie a dátové formáty.
2. Vykresľovanie grafiky nepoužíva priesvitnosť a vyhladzovanie.
3. Aplikácie sú nestabilné.
4. Aplikácie sú napísané len pre operačný systém Windows alebo DOS.
5. Aplikácie používajú jednobytové kódovanie znakov, čo spôsobuje problémy pri medzinárodných účastníkoch.

V tejto práci sa zameriame na funkcionalitu pre stavanie tratí a vyhodnotenie výsledkov.

Kapitola 3

Prehľad problematiky

V tejto kapitole popíšeme celý proces prípravy pretekov v orientačnom behu. Rozoberieme, aké vlastnosti a funkcie očakávame od aplikácie.

Organizácia pretekov je náročný proces, hlavne ak sa jedná o celoslovenské alebo medzinárodné preteky. Najprv organizátor určí miesto a termín pretekov. Potom sa začnú kartografické práce v teréne (tvorba mapy). Tie trvajú niekoľko mesiacov, pokiaľ ide o nový a neznámy terén, a niekoľko dní, ak je k dispozícii aktuálna mapa a stačí dokresliť pár zmien – väčšinou sú to zmeny, ktoré v lese vykonali lesníci alebo to môžu byť nové cesty a budovy. Práca kartografa spočíva v zakresľovaní situácie v teréne a neskôr ju prekresľuje do počítača. Mapa sa skladá z vektorových útvarov. Ukážka mapy Červeného Kameňa je na obrázku 3.1.

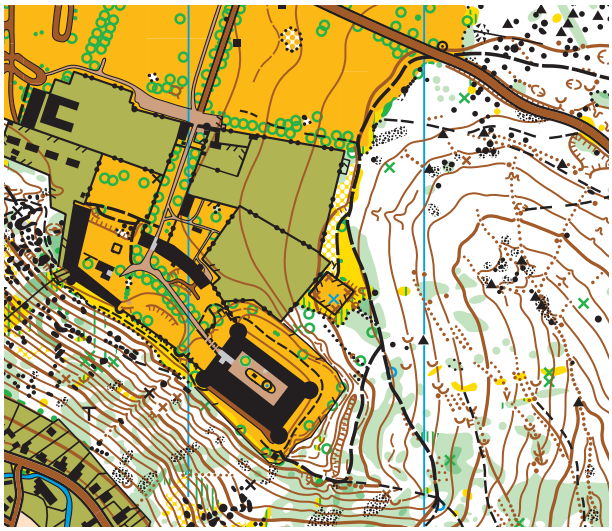
3.1 Príprava tratí

V druhej fáze staviteľ tratí pomocou mapy v lese nájde vhodné miesta pre kontrolné stanovištia a vytýči trate. Trať sa začína štartom, nasledujú kontrolné stanovištia a končí sa cieľom. V niektorých prípadoch môže byť trať prerušená kvôli výmene mapy, ide však len o technický detail. Ukážku trate vidíme na obrázku 3.2.

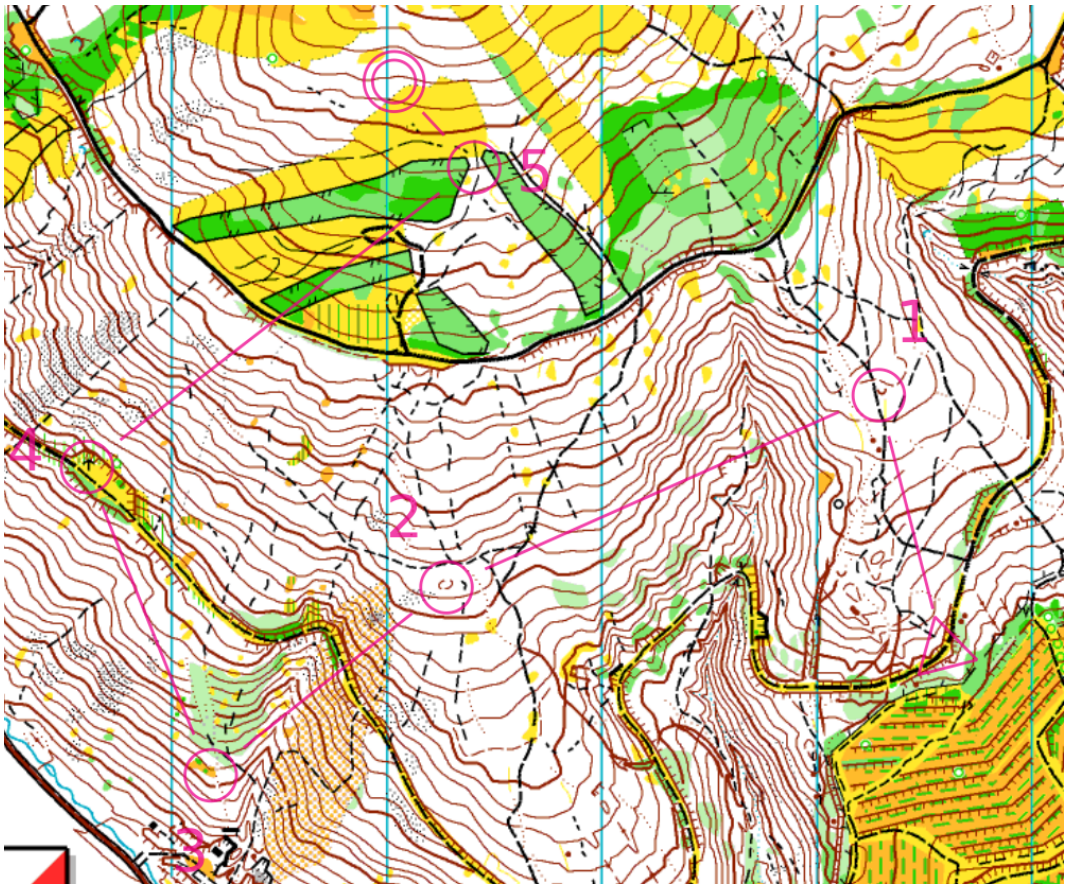
Štart sa značí trojuholníkom, kontrolné stanovištia krúžkom a cieľ dvomi sústrednými krúžkami. Pretekár musí prejsť kontrolné stanovištia v danom poradí a dobehnúť do cieľa. Kresba máp a tratí sa musí riadiť normou medzinárodnej federácie [ISOM]. Sú v nej definované presné rozmery objektov a hrúbky čiar.

Stavba tratí je náročná, pretože staviteľ musí odhadnúť správnu dĺžku pre rôzne vekové kategórie a navyše trate musia byť zaujímavé. V tejto fáze je veľmi užitočný práve počítač, lebo v ňom si môže staviteľ rýchlo kontrolovať svoje nápady – či nie je trať príliš dlhá alebo či nebude naraz veľa pretekárov na jednom mieste. Všetko si môže skontrolovať rýchlo a nemusí si varianty kresliť na mapu. Navyše pri ručnom kreslení je vysoká pravdepodobnosť chyby.

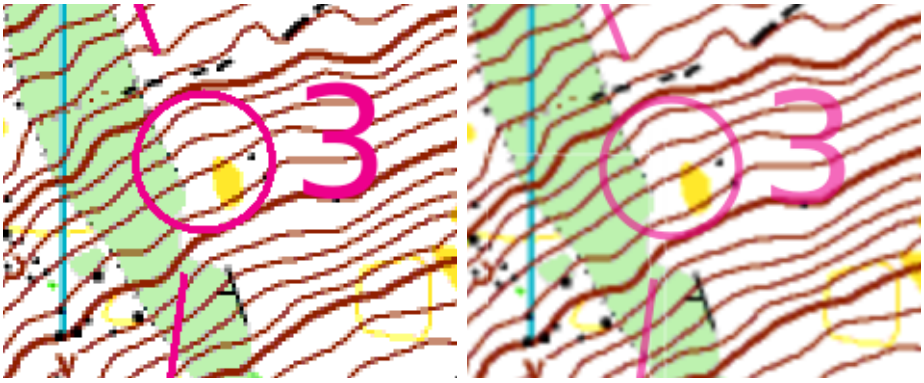
Keď staviteľ navrhne trate, sú dve možnosti. Buď trate nakreslia ľudia ručne alebo sa trate vytlačia. Prvá možnosť má zmysel pri malom počte pretekárov – do 50 až 100



Obr. 3.1: Mapa Červeného Kameňa



Obr. 3.2: Ukážka trate



Obr. 3.3: Porovnanie nevyhladzovaného a vyhladzovaného vykresľovania

ľudí. Kreslenie tratí je časovo náročné a náchylné na chybu. Pri tlači máp sú znova dve možnosti. Buď sa každá trať vyexportuje spolu s mapou a môže sa potom vytlačiť ako obrázok alebo sa vyexportujú špeciálne iba trate a tie sa dotláčajú na hotové mapy. Druhá možnosť sa používa častejšie, pretože výsledok býva kvalitnejší. Mapy pre orientačný beh sa nezvyknú tlačiť technológiou CMYK, ale každá z použitých farieb sa tlačí zvlášť. Trať má svoju vlastnú farbu (fialová) a tlačí sa nakoniec. Preto je veľmi dôležitý export tratí do rôznych vektorových a bitmapových formátov. Aplikácia umožňuje export do bitmapového formátu PNG. Situácia s vektorovými formátmi sa trochu skomplikovala, lebo knižnica Cairo 1.2, ktorú sme chceli použiť na export do vektorových formátov, nemá ešte prepojenie s projektom Mono v niektorých funkciách (nižšie verzie Caira prepojenie majú). Preto vývojári projektu Mono toto prepojenie celé vypli a znova bude zapnuté až v neskorších verziách. Po ručnom odomknutí podpory v zdrojových kódoch export do formátov PDF a SVG fungoval dobre. Export do formátu PS však trval okolo minúty. Kód na export je teda hotový, treba však počkať, kým bude knižnica Cairo prepojená aj v oficiálnej verzii Mona.

Pri exporte do bitmapových formátov v existujúcich aplikáciách sa stával nepríjemný jav, ak sa nepoužili priesvitné farby na dokresľovanie tratí do mapy. Trať prekryla niektoré objekty na mape blízko kontrolného stanovišťa. Všimnime si čiernu bodku na okraji krúžku na obrázku 3.3.

Preto moja aplikácia používa priesvitné farby. Používateľ si môže nastaviť mieru priesvitnosti farby. Ako tiež vidno v ľavej časti obrázku 3.3, staršie aplikácie používajú nevyhladzované vykresľovanie a trať spolu s mapou vyzerajú škaredšie.

3.2 Prihlasovanie pretekárov

Fáza vytvárania tratí končí asi dva týždne pred pretekmi odovzdaním tratí do tlače. Asi mesiac pred pretekmi býva termín prihlášok na preteky a vtedy sa začína napĺňanie databázy pretekárov. Pretekári sa môžu prihlásiť aj po termíne prihlášok, ale poplatok za

štart býva vyšší. Pretekári sú organizovaní do klubov a každý registrovaný pretekár ma pridelené registračné číslo. Prihláška jedného pretekára obsahuje meno, kategóriu, klub a registračné číslo. V súčasnosti sa na vyhodnocovanie pretekov používajú elektronické zariadenia a každý pretekár má svoj čip, ktorý má jednoznačné číslo. Všetky tieto údaje musia byť uložené v aplikácií.

Tesne pred pretekmi sa vylosuje z databázy štartová listina. Orientačný beh je individuálny šport, kde sa každý orientuje samostatne. Preto pretekári štartujú samostatne – každý má pridelený štartový čas. Existujú však aj výnimky, keď sa štartuje hromadne – napríklad počas štafetových pretekov. Na vylosovanie sa kladie niekoľko podmienok. Napríklad nemôžu štartovať dvaja pretekári z rovnakého klubu tesne po sebe v kategórii, aby sa zamedzilo spolupráci počas pretekov. Na majstrovstvách štartujú pretekári s vyššou licenciou na konci štartového poľa. Navyše počas jednej minúty nemôžu vybiehať viacerí pretekári na rovnaké kontrolné stanovište.

3.3 Vyhodnocovanie výsledkov

Vyhodnocovanie výsledkov sa robí počas samotných pretekov. V súčasnosti je uľahčené vďaka používaniu elektronického raziacemu systému. Každý pretekár má v čipe zaznamenaný prechod kontrolnými stanovišťami a cieľom. V cieli sa z čipu načítajú dáta a program zistí, či pretekár korektne prešiel celú trať. V minulosti to boli papierové preukazy a každé kontrolné stanovište malo svoje železné kliešte s dierovacím vzorom. To, či pretekár prešiel stanovišťom sa posudzuje podľa toho, či dierovací vzor sedí. Túto činnosť vykonávajú ľudia. Neskôr do počítača ručne zadávame pre každého človeka či prešiel korektne trať a ak áno, zadáme aj jeho čas. Ak je použité elektronické razenie, počítač toto všetko vykoná sám. V súčasnosti sa používajú dva systémy: SportIdent a EMIT. EMIT je rozšírený len v Škandinávií. Pri tvorbe bakalárskej práce sme mali k dispozícii hardvér SportIdent a aj popis komunikácie. Úspešne sme získali dáta z čipu, ale firma SportIdent chce uchovať svoj protokol v tajnosti, preto sme nemohli zahrnúť kód do tejto práce.

Podľa pravidiel pretekár prešiel korektne trať, ak je trať vybranou podpostupnosťou jeho orazených kontrolných stanovíšť. V tabuľke vidieť korektné prejdenie trate:

Trať:	31		32	33		34	35
Čip pretekára:	**31**	47	**32**	**33**	31	**34**	**35**

Nevadí, ak pretekár navštívi stanovište, ktoré nie je na jeho trati (47) a tiež nevadí, ak navštívi druhýkrát to isté stanovište (31). Ak pretekár úspešne prešiel trať, zaradí sa do poriadia podľa času, inak je diskvalifikovaný.

Softvér vyhodnotí časy pretekárov a zotriedi ich podľa nich. Diskvalifikovaní pretekári sú na konci. Výsledky môžeme exportovať do formátu HTML. Takisto sa dajú zobraziť výsledky s medzičasmi – ako dlho trvali každému pretekárovi jednotlivé úseky trate medzi kontrolnými stanovišťami.

Po skončení pretekov sa výsledky ďalej spracúvajú, aby sa mohli vyhodnocovať dlhodobé súťaže – napríklad Slovenský rebríček. Program tiež umožňuje export do formátu pre Slovenský rebríček.

Kapitola 4

Technické riešenie

V tejto časti podrobnejšie popíšeme samotnú implementáciu aplikácie a použité technológie.

4.1 Moduly

Aplikácia je pomerne veľká a obsahuje časti s odlišnou funkcionalitou. Zvolili sme rozdelenie do viacerých modulov. Moduly majú anglické názvy Data, Competition, GUI. Tieto moduly majú za úlohu rozdeliť funkcionalitu do vrstiev, aby bola implementácia prehľadnejšia. Najnižšia vrstva je Data, nad ňou je Competition a najvyššie je GUI. Ak napríklad vrstva GUI chce vykresliť objekt na obrazovku, spýta sa vrstvy Competition, aké má objekt súradnice. Táto vrstva sa spýta vsrtvy Data, ktorá sa spojí s databázou.

Komunikácia dvoch vrstiev vyzerá tak, že v obidvoch vrstvách existujú objekty s rovnakým menom a objekt vyššej vrstvy je zaobalením objektu nižšej vrstvy. Pod zaobalením máme na mysli to, že objekt vyššej vrstvy si pamätá referenciu na objekt nižšej vrstvy a pridáva k nemu ďalšiu funkcionalitu. Okrem referencie na objekt z nižšej vrstvy, preto obsahuje len metódy rozširujúce funkcionalitu alebo zaobaľujúce funkcionalitu nižšej vstvy.

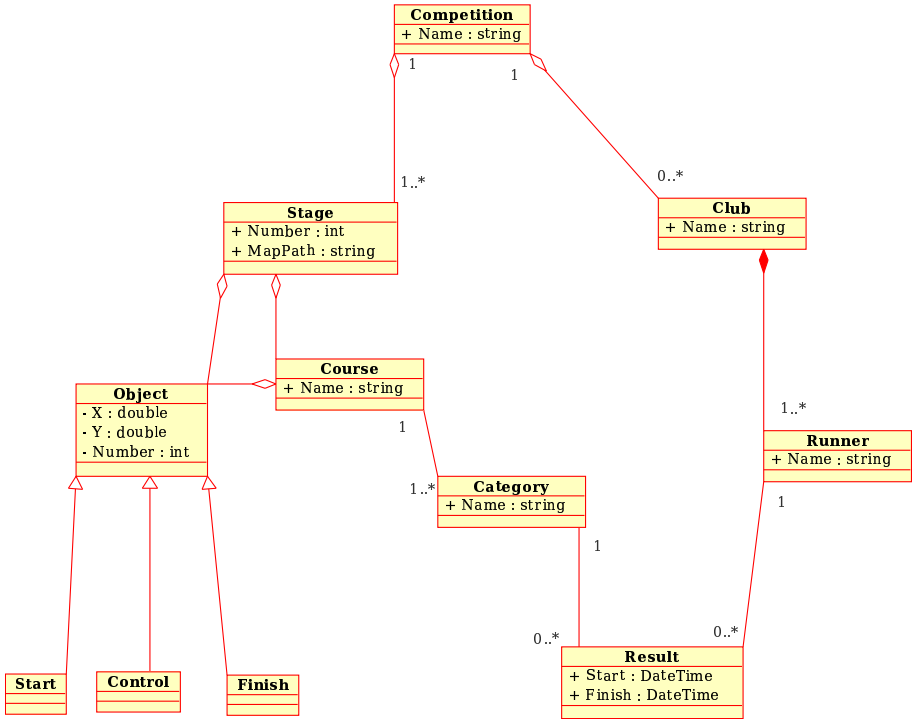
Každý modul tvorí svoj vlastný menný priestor – Orienteering.Data, Orienteering.Competition a Orienteering.GUI.

4.1.1 Modul Data

Jedinou úlohou tohto modulu je komunikovať s databázou. Modul Competition preto ani nevie, že dáta sa v skutočnosti vyberajú z databázy.

4.1.2 Modul Competition – Preteky

V module Competition (po slovensky preteky) sa nachádza veľká časť funkcionality. Keď sa preteky ukladajú do súboru, v činnosti je práve tento modul. Alebo keď sa vyhodnocujú výsledky, losuje štartová listina a exportujú výsledky.



Obr. 4.1: UML diagram pre triedy modulu Data (neobsahuje metódy)

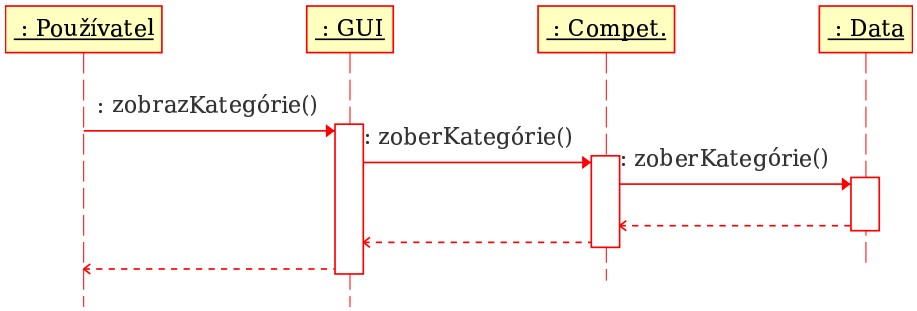
4.1.3 Modul GUI – Grafické rozhranie

Grafické rozhranie nerobí žiadne operácie s dátami, iba komunikuje s používateľom a požiadavky posiela nižšej vrstve (Competition). Takisto sa stará o vykresľovanie tratí na obrazovku alebo o export do grafických formátov.

4.2 Použité knižnice

Pri výbere programovacieho jazyka malo najväčšiu váhu to, aby bola aplikácia multiplatformná. Vybrali sme si jazyk C#, pretože popri originálnej implementácii od spoločnosti Microsoft existuje aj implementácia Mono od firmy Novell, ktorá beží aj na Unixe. Navyše je automaticky k dispozícií .NET Class Library, ktorá obsahuje veľa užitočných funkcií – napríklad kompresný algoritmus Gzip alebo knižnicu na prácu s XML. Grafickú knižnicu sme si vybrali GTK#. Druhá možnosť bola použiť Windows.Forms, ale táto implementácia používa umiestňovanie grafických prvkov elementov podľa absolútnych súradníc v pixeloch, čo je v súčasnosti zastaralý prístup.

Druhou veľmi dôležitou knižnicou je knižnica Db4o. To je objektová databáza, ktorá si pamätá priamo objekty z jazyka C#. Databázový dopyt v nej sa robí tak, že ako požiadavku odovzdávame funkcii napísanú v jazyku C#, ktorá vracia boolovskú hodnotu. Táto funkcia je v skutočnosti predikát. Databáza vráti všetky objekty, ktoré daný predikát spĺňajú.



Obr. 4.2: Sekvenčný UML diagram spolupráce modulov pri editácii kategórií

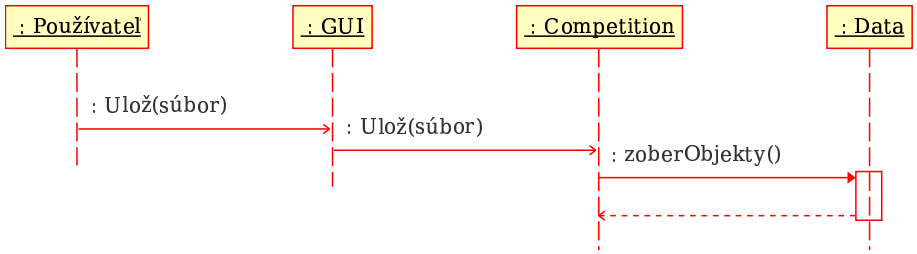
V nasledovnej ukážke je dopyt, ktorý vráti kontrolné stanovište s daným číslom:

```
public class Stage {
    ...
    public Control Control(int num, DataBase dataBase) {
        IList<Control> res = dataBase.db.Query<Control>(
                delegate(Control control) {
                    return control.Number == num && control.Stage == this;
                }
            );
        if (res.Count == 0)
            throw new DoesntExist();
        else return res[0];
    }
    ...
}
```

Parameter `delegate(Control control)` je predikát, ktorý odovzdávame databáze ako parameter. Ako vidíme, predikát spĺňajú iba kontrolné stanovištia s daným číslom a v danej etape (stage).

4.3 Spolupráca modulov

V diagramoch 4.2 a 4.3 vidíme, čo sa medzi modulmi udeje, keď si používateľ chce zobraziť všetky kategórie danej etapy a keď chce uložiť súbor.



Obr. 4.3: Sekvenčný UML diagram spolupráce modulov pri ukladaní súboru

4.4 Súborový formát

Vytvoriť vhodný súborový formát pre aplikáciu je neľahká úloha. V našej aplikácii sme sa rozhodli použiť prístup známy napríklad z raných verzií produktu OpenOffice. Ako základný formát ukladania dát použijeme formát XML. O jeho výhodách sa zmienime neskôr. Keďže tento formát dokáže objem dát zväčšiť, ešte pred uložením na disk sa súbor skomprimuje kompresným algoritmom Gzip. Počas tvorby práce bol kompresný pomer 1 : 5 a lepší.

Formát XML sa vďaka stromovej štruktúre hodí pre našu aplikáciu. Vzťahy objektov sú väčšinou hierarchické. Preteky sa skladajú z niekoľkých etáp. Každá etapa obsahuje niekoľko tratí, každá trať môže prislúchať viacerým kategóriám atď. Existujú však aj objekty, ktoré vytvárajú v stromovej štruktúre cykly. Tie sme riešili referenciami na iné objekty pomocou identifikátorov. Keďže objekty ako štart, cieľ a kontrolné stanovište svoje unikátne číslo majú, netreba pre ne vyrábať nový identifikátor. Rovnako každá kategória má jednoznačný názov a ten poslúži ako identifikátor.

Nasleduje ukážka jednoduchého XML súboru (pozrieť súbor sa dá v Unixe jednoducho napríklad pomocu príkazu `zcat GPS.oxz`, kde `GPS.oxz` je konkrétny súbor s pretekmi):

```
<?xml version="1.0" encoding="utf-8"?>
<competition name="Grand Prix Slovakia">
  <stage number="1" organizer="KOB Sokol Pezinok"
  mapPath="/home/lukas/svn/bakalar/bin/ompital.png">
    <objects>
      <start number="1" X="100" Y="600" />
      <control number="31" X="100" Y="250" />
      <control number="32" X="600" Y="450" />
      <control number="33" X="400" Y="350" />
      <control number="34" X="800" Y="500" />
      <finish number="1" X="800" Y="250" />
    </objects>
    <courses>
      <course name="T1">
        <start number="1" />
```

```
        <control number="31" />
        <control number="32" />
        <control number="33" />
        <control number="34" />
        <finish number="1" />
        <category name="M -16" firstTime="1" interval="2" />
        <category name="W -18" firstTime="2" interval="2" />
      </course>
    </courses>
  </stage>
  <club name="KOB Sokol Pezinok" abbreviation="SPE" country="SVK">
    <runner name="Lukáš Poláček" regNumber="8601"/>
  </club>
</competition>
```

Koreňový element dokumentu je `competition`, teda preteky. Každý súbor obsahuje práve jedny preteky. Každý XML element predstavuje jeden skutočný objekt v aplikácii a vlastnosti elementu sú zároveň vlastnosťami objektu. Ako vidíme v príklade, element pre etapu (`stage`) má napríklad vlastnosť číslo (`number`), čo určuje poradie tejto etapy v pretekoch. Ak má element synov, znamená to, že synovia si uchovávajú referenciu na svojho predka. Napríklad v aplikácii si objekt pre pretekára (`runner`) pamätá referenciu na klub (`club`).

Kapitola 5

Záver

Táto práca mala za cieľ navrhnúť softvér pre orientačný beh tak, aby spĺňal náročné kritériá kvality. Popísali sme funkcie, aké by mal softvér obsahovať. Problematiku sme analyzovali a navrhli štruktúru modulov, z ktorých sa skladá aplikácia. Implementovali sme základnú funkcionalitu, ktorá umožní organizáciu menších pretekov.

Nepodarilo sa nám implementovať všetky funkcie, ktoré by mala správna aplikácia obsahovať. Tento vývoj je časovo náročný, keďže požadovaných funkcií je veľmi veľa. Väčšinu týchto funkcií treba implementovať v module GUI, ktorý komunikuje s používateľom. Dalo by sa povedať, že zvyšné dva moduly sú z väčšej časti hotové.

Vo vývoji softvéru budem pokračovať vo voľnom čase a neskôr, keď už bude obsahovať všetky potrebné funkcie, uvoľním ho aj so zdrojovými kódmi.

Dodatok A

Ukážky kódu

V tomto dodatku sa nachádzajú ukážky kódu jednotlivých modulov. Celá aplikácia má viac ako 3500 riadkov kódu, preto uvedieme len niektoré časti.

A.1 Kód modulu Data

Ukážeme si úryvok kódu z modulu Data. Konkrétne je to trieda Stage (po slovensky etapa). Ako vidíme, táto trieda má niekoľko vlastností – číslo, ukazovateľ na preteky, začiatok pretekov. Ďalej má napríklad metódy, ktoré vrátia z databázy všetky trate, ktoré sú v tejto etape.

```
using System;
using System.Collections.Generic;
using Orienteering.Standard;

namespace Orienteering.Data {
    public class Stage {
        private int number;
        private DateTime startTime;
        private string organizer, mapPath;
        private Orienteering.Data.Competition competition;

        public Stage() {
        }

        public Stage(int _number, string _organizer, Competition _comp) {
            number = _number;
            organizer = _organizer;
            competition = _comp;
        }
```

```
public DateTime StartTime {
    get { return startTime; }
    set { startTime = value; }
}

public string MapPath {
    get { return mapPath; }
    set { mapPath = value; }
}

public string Organizer {
    get { return organizer; }
    set { organizer = value; }
}

public int Number {
    get { return number; }
    set { number = value; }
}

public Competition Competition {
    get { return competition; }
    set { competition = value; }
}

public void AddCourse(Course course, DataBase dataBase) {
    if (dataBase.db.Query<Course>(delegate(Course cour) {
            return cour.Stage == this && course.Name == cour.Name;
        }).Count == 0) {
        dataBase.db.Set(course);
    }
    else throw new AlreadyExists();
}

public void AddObject(Object obj, DataBase dataBase) {
    if (dataBase.db.Query<Object>(delegate(Object o) {
        return o.Stage == this && obj.Number == o.Number &&
            o.GetType() == obj.GetType();
        }).Count == 0) {
        dataBase.db.Set(obj);
    }
    else {
        throw new AlreadyExists();
```

```
    }
}

public List<Object> Objects(DataBase dataBase) {
    return new List<Object>(
        dataBase.db.Query<Object>(
            delegate(Object obj) {
                return obj.Stage == this;
            }
        )
    );
}

public List<Course> Courses(DataBase dataBase) {
    return new List<Course>(
        dataBase.db.Query<Course>(
            delegate(Course course) {
                return course.Stage == this;
            }
        )
    );
}

public List<Category> Categories(DataBase dataBase) {
    return new List<Category>(
        dataBase.db.Query<Category>(
            delegate(Category category) {
                return category.Course.Stage == this;
            }
        )
    );
}

public Course Course(string name, DataBase dataBase) {
    IList<Course> res = dataBase.db.Query<Course>(
            delegate(Course course) {
                return course.Name == name && course.Stage == this;
            }
        );
    if (res.Count == 0)
        throw new DoesntExist();
    else return res[0];
}
```

```
public Start Start(int num, DataBase dataBase) {
    IList<Start> res = dataBase.db.Query<Start>(
            delegate(Start start) {
                return start.Number == num && start.Stage == this;
            }
        );
    if (res.Count == 0)
        throw new DoesntExist();
    else return res[0];
}

public Control Control(int num, DataBase dataBase) {
    IList<Control> res = dataBase.db.Query<Control>(
            delegate(Control control) {
                return control.Number == num && control.Stage == this;
            }
        );
    if (res.Count == 0)
        throw new DoesntExist();
    else return res[0];
}

public Finish Finish(int num, DataBase dataBase) {
    IList<Finish> res = dataBase.db.Query<Finish>(
            delegate(Finish finish) {
                return finish.Number == num && finish.Stage == this;
            }
        );
    if (res.Count == 0)
        throw new DoesntExist();
    else return res[0];
}

public List<Start> Starts(DataBase dataBase) {
    return new List<Start>(
        dataBase.db.Query<Start>(
            delegate(Start start) {
                return start.Stage == this;
            }
        )
    );
}
```

```
        public List<Finish> Finishes(DataBase dataBase) {
            return new List<Finish>(
                dataBase.db.Query<Finish>(
                    delegate(Finish finish) {
                        return finish.Stage == this;
                    }
                )
            );
        }

        public List<Control> Controls(DataBase dataBase) {
            return new List<Control>(
                dataBase.db.Query<Control>(
                    delegate(Control control) {
                        return control.Stage == this;
                    }
                )
            );
        }

        public void Save(DataBase dataBase) {
            dataBase.db.Set(this);
        }
    }
}
```

A.2 Kód modulu Competition

V tomto module sa nachádza pokročilejšia funkcionalita. Znova si ukážeme triedu Stage. Oproti modulu Data je tu naviac implementované napríklad ukladanie do XML súboru a načítavanie z XML súboru. Všimnime si, že trieda obsahuje iba jednu referenciu na objekt z modulu Data a ďalej má už len metódy a vlastnosti, ktoré buď pridávajú novú funkcionalitu alebo zaobaľujú funkcionalitu modulu Data.

```
using System;
using System.Xml;
using System.Collections.Generic;
using Orienteering.Standard;

namespace Orienteering.Competition {
```

```
public class Stage {
    internal Orienteering.Data.Stage data;

    private void SaveNew(DataBase dataBase) {
        try {
            Competition.Stage(Number, dataBase);
            throw new AlreadyExists();
        }
        catch (DoesntExist) {
            this.Save(dataBase);
        }
    }

    public Stage(int num, string organiz, Competition comp,
            DataBase dataBase) {
        data = new Orienteering.Data.Stage(num, organiz, comp.data);
        SaveNew(dataBase);
    }

    public Stage(XmlNode xml, Competition comp, DataBase dataBase) {
        data = new Orienteering.Data.Stage();
        Competition = comp;

        foreach (XmlAttribute xmlattr in xml.Attributes) {
            if (xmlattr.Name == „number“)
                Number = Convert.ToInt32(xmlattr.Value);
            if (xmlattr.Name == „organizer“)
                Organizer = xmlattr.Value;
            if (xmlattr.Name == „mapPath“)
                MapPath = xmlattr.Value;
            if (xmlattr.Name == „startTime“)
                StartTime = Convert.ToDateTime(xmlattr.Value);
        }
        SaveNew(dataBase);

        foreach (XmlNode node in xml.ChildNodes)
            if (node.Name == „objects“)
                foreach (XmlNode node1 in node.ChildNodes) {
                    if (node1.Name == „start“)
                        AddObject(new Start(node1, this), dataBase);
                    if (node1.Name == „control“)
                        AddObject(new Control(node1, this), dataBase);
                    if (node1.Name == „finish“)
```

```
                    AddObject(new Finish(node1, this), dataBase);
                }

            foreach (XmlNode node in xml.ChildNodes)
                if (node.Name == „courses“)
                    foreach (XmlNode node1 in node.ChildNodes)
                        if (node1.Name == „course“)
                            new Course(node1, this, dataBase);

        }

        public DateTime StartTime {
            get { return data.StartTime; }
            set { data.StartTime = value; }
        }

        public string MapPath {
            get { return data.MapPath; }
            set { data.MapPath = value; }
        }

        public int Number {
            get { return data.Number; }
            set { data.Number = value; }
        }

        public string Organizer {
            get { return data.Organizer; }
            set { data.Organizer = value; }
        }

        public Competition Competition {
            get { return new Competition(data.Competition); }
            set { data.Competition = value.data; }
        }

        public Stage(Orienteering.Data.Stage _data) {
            data = _data;
        }

        public void Save(DataBase dataBase) {
            data.Save(dataBase.data);
        }
```

```
public void AddCourse(Course course, DataBase dataBase) {
    data.AddCourse(course.data, dataBase.data);
}

public void AddObject(Object obj, DataBase dataBase) {
    data.AddObject(obj.data, dataBase.data);
}

public List<Object> Objects(DataBase dataBase) {
    List<Object> res = new List<Object>();
    foreach (Orienteering.Data.Object obj in data.Objects(dataBase.data)) {
        if (obj is Orienteering.Data.Start)
            res.Add(new Start(obj as Orienteering.Data.Start));
        if (obj is Orienteering.Data.Control)
            res.Add(new Control(obj as Orienteering.Data.Control));
        if (obj is Orienteering.Data.Finish)
            res.Add(new Finish(obj as Orienteering.Data.Finish));
    }
    return res;
}

public List<Course> Courses(DataBase dataBase) {
    List<Course> res = new List<Course>();
    foreach (Orienteering.Data.Course course in
            data.Courses(dataBase.data))
        res.Add(new Course(course));
    return res;
}

public List<Start> Starts(DataBase dataBase) {
    List<Start> res = new List<Start>();
    foreach (Orienteering.Data.Start start in data.Starts(dataBase.data))
        res.Add(new Start(start));
    return res;
}

public List<Finish> Finishes(DataBase dataBase) {
    List<Finish> res = new List<Finish>();
    foreach (Orienteering.Data.Finish finish in
            data.Finishes(dataBase.data))
        res.Add(new Finish(finish));
    return res;
```

```
}

public List<Control> Controls(DataBase dataBase) {
    List<Control> res = new List<Control>();
    foreach (Orienteering.Data.Control control in
            data.Controls(dataBase.data))
        res.Add(new Control(control));
    return res;
}

public List<Category> Categories(DataBase dataBase) {
    List<Category> res = new List<Category>();
    foreach (Orienteering.Data.Category category in
            data.Categories(dataBase.data))
        res.Add(new Category(category));
    return res;
}

public Course Course(string name, DataBase dataBase) {
    return new Course(data.Course(name, dataBase.data));
}

public Start Start(int num, DataBase dataBase) {
    return new Start(data.Start(num, dataBase.data));
}

public Control Control(int num, DataBase dataBase) {
    return new Control(data.Control(num, dataBase.data));
}

public Finish Finish(int num, DataBase dataBase) {
    return new Finish(data.Finish(num, dataBase.data));
}

public XmlNode Xml(XmlDocument xml, DataBase dataBase) {
    XmlNode res = xml.CreateNode(„element“, „stage“, „“);
    XmlAttribute numberAttr =
        (XmlAttribute)xml.CreateNode(„attribute“, „number“, „“);
    numberAttr.Value = Number.ToString();
    res.Attributes.Append(numberAttr);

    XmlAttribute organizerAttr =
        (XmlAttribute)xml.CreateNode(„attribute“, „organizer“, „“);
```

```
            organizerAttr.Value = Organizer;
            res.Attributes.Append(organizerAttr);

            XmlAttribute mapPathAttr =
                (XmlAttribute)xml.CreateNode("attribute", "mapPath", "");
            mapPathAttr.Value = MapPath;
            res.Attributes.Append(mapPathAttr);

            XmlAttribute startTimeAttr =
                (XmlAttribute)xml.CreateNode("attribute", "startTime", "");
            startTimeAttr.Value = Convert.ToString(StartTime);
            res.Attributes.Append(startTimeAttr);

            XmlNode obj = xml.CreateNode("element", "objects", "");
            foreach (Object k in Objects(dataBase))
                obj.AppendChild(k.Xml(xml));
            res.AppendChild(obj);

            XmlNode cour = xml.CreateNode("element", "courses", "");
            foreach (Course course in Courses(dataBase))
                cour.AppendChild(course.Xml(xml, dataBase));
            res.AppendChild(cour);

            return res;
        }
    }
}
```

A.3 Kód modulu GUI

Tento modul obsahuje ovládanie grafických prvkov a reakcie na akcie používateľa. Požiadavky posiela nižšej vrstve. Všimnime si, že okrem referencie na objekt nižšej vrstvy obsahuje táto trieda aj dva ďalšie objekty – meno vybranej trate, ktorá sa vykresľuje na obrazovku a obrázok mapy.

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Drawing2D;
using OrComp = Orienteering.Competition;
using Gtk;
```

```
namespace Orienteering.GUI {
    public class Stage {
        internal Orienteering.Competition.Stage data;
        private string drawCourse;
        internal System.Drawing.Image map;
        public Stage(Orienteering.Competition.Stage _data) {
            data = _data;
            drawCourse = null;
            map = System.Drawing.Image.FromFile(data.MapPath);
        }

        public void AddObject(Object obj, DataBase dataBase) {
            data.AddObject(obj.data, dataBase.data);
        }

        public List<Object> Objects(DataBase dataBase) {
            List<Object> res = new List<Object>();
            foreach (OrComp.Object obj in data.Objects(dataBase.data)) {
                if (obj is OrComp.Start)
                    res.Add(new Start(obj as OrComp.Start));
                if (obj is OrComp.Control)
                    res.Add(new Control(obj as OrComp.Control));
                if (obj is OrComp.Finish)
                    res.Add(new Finish(obj as OrComp.Finish));
            }
            return res;
        }

        public void Save(DataBase dataBase) {
            data.Save(dataBase.data);
        }

        public void Draw(System.Drawing.Graphics gr, DataBase dataBase) {
            if (Config.DrawMap)
                gr.DrawImage(map, 0, 0);
            else
                gr.Clear(System.Drawing.Color.White);

            Dictionary<int, bool> set = new Dictionary<int, bool>();
            if (drawCourse != null) {
                Config.Style = Config.PenStyle.Dark;
                Config.FStyle = Config.FontStyle.Big;
```

```
            Course course = this.Course(drawCourse, dataBase);
            course.Draw(gr);
            foreach (Object obj in course.Objects) {
                int tmp = 3*obj.Number;
                if (obj is Control) tmp++;
                if (obj is Finish) tmp += 2;
                set[tmp] = true;
            }
        }

        Config.Style = Config.PenStyle.Light;
        Config.FStyle = Config.FontStyle.Small;
        foreach (Object obj in Objects(dataBase)) {
            int q = obj.Number*3;
            if (obj is Control) q++;
            if (obj is Finish) q += 2;

            if (!set.ContainsKey(q)) {
                obj.Draw(gr, Math.PI);
                obj.DrawNumber(gr, obj.Number, Math.PI/4);
            }
        }

    }

    public ComboBox Starts(DataBase dataBase) {
        ComboBox combo = ComboBox.NewText();
        List<OrComp.Start> starts = data.Starts(dataBase.data);
        foreach (OrComp.Start start in starts)
            combo.AppendText(start.Number.ToString());
        return combo;
    }

    public ComboBox Finishes(DataBase dataBase) {
        ComboBox combo = ComboBox.NewText();
        List<OrComp.Start> starts = data.Starts(dataBase.data);
        foreach (OrComp.Start start in starts)
            combo.AppendText(start.Number.ToString());
        return combo;
    }

    public Course Course(string name, DataBase dataBase) {
        return new Course(data.Course(name, dataBase.data));
```

```
}

private ListStore CourseListStore;
private TreeView tree;
private List<Orienteering.Competition.Course> courses;
private DataBase db;
public VBox Courses(DataBase dataBase) {
    db = dataBase;
    courses = data.Courses(dataBase.data);

    CourseListStore =
        new ListStore(typeof(Orienteering.Competition.Course));
    foreach (OrComp.Course course in courses)
        CourseListStore.AppendValues(course);

    tree = new TreeView(CourseListStore);
    tree.EnableGridLines = TreeViewGridLines.Both;
    tree.AppendColumn( Standard.treeColText("Meno",
                new TreeCellDataFunc(RenderName), EditName) );
    tree.AppendColumn( Standard.treeColText("Dĺžka (km)",
                new TreeCellDataFunc(RenderLength), null) );
    tree.AppendColumn( Standard.treeColTogg("Vykreslená trať",
                new TreeCellDataFunc(RenderDrawn), EditDrawn) );

    tree.RulesHint = true;
    Viewport viePo = new Viewport();
    viePo.Add(tree);
    ScrolledWindow viewer = new ScrolledWindow();
    viewer.Add(viePo);

    VBox vbox1 = new VBox();
    vbox1.PackStart(viewer, true, true, 2);

    return vbox1;
}

private void RenderLength(TreeViewColumn column, CellRenderer cell,
        TreeModel model, TreeIter iter) {
    OrComp.Course course = (OrComp.Course)model.GetValue(iter, 0);
    (cell as CellRendererText).Text =
        (course.Length/1500*2.54).ToString();
}
```

```
private void RenderName(TreeViewColumn column, CellRenderer cell,
        TreeModel model, TreeIter iter) {
    OrComp.Course course = (OrComp.Course)model.GetValue(iter, 0);
    (cell as CellRendererText).Text = course.Name;
}
private void EditName(object o, EditedArgs args) {
    TreeIter iter;
    CourseListStore.GetIter (out iter, new TreePath (args.Path));
    OrComp.Course course =
        (OrComp.Course)CourseListStore.GetValue(iter, 0);
    course.Name = args.NewText;
    course.Save(db.data);
}

private void RenderDrawn(TreeViewColumn column, CellRenderer cell,
        TreeModel model, TreeIter iter) {
    OrComp.Course course = (OrComp.Course)model.GetValue(iter, 0);
    (cell as CellRendererToggle).Active =
        (drawCourse != null && drawCourse == course.Name);
}
private void EditDrawn(object o, ToggledArgs args) {
    TreeIter iter;
    CourseListStore.GetIter (out iter, new TreePath (args.Path));
    OrComp.Course course =
        (OrComp.Course)CourseListStore.GetValue(iter, 0);
    drawCourse = course.Name;
    data.Save(db.data);
}



private ListStore CategoryListStore;
private List<OrComp.Category> categories;
public VBox Categories(DataBase dataBase)  {
    db = dataBase;
    courses = data.Courses(dataBase.data);
    CourseListStore = new ListStore(typeof(string));
    foreach (OrComp.Course course in courses)
        CourseListStore.AppendValues(course.Name);

    categories = data.Categories(dataBase.data);
    CategoryListStore = new ListStore(typeof(OrComp.Category));
    foreach (OrComp.Category category in categories)
```

```
            CategoryListStore.AppendValues(category);

        tree = new TreeView(CategoryListStore);
        tree.EnableGridLines = TreeViewGridLines.Both;
        tree.AppendColumn( Standard.treeColText(„Meno“,
                    new TreeCellDataFunc(RenderCatName), EditCatName) );
        tree.AppendColumn( Standard.treeColComb(„Trať“, CourseListStore,
                    new TreeCellDataFunc(RenderCourse), EditCourse) );
        tree.AppendColumn( Standard.treeColText(„Prvý štart“,
                    new TreeCellDataFunc(RenderFirstTime), EditFirstTime) );
        tree.AppendColumn( Standard.treeColText(„Interval“,
                    new TreeCellDataFunc(RenderInterval), EditInterval) );

        tree.RulesHint = true;
        Viewport viePo = new Viewport();
        viePo.Add(tree);
        ScrolledWindow viewer = new ScrolledWindow();
        viewer.Add(viePo);

        HBox hbox = new HBox();
        Button button1 = new Button(„Pridať kategóriu“),
               button2 = new Button(„Zmazať kategóriu“);
        button1.Clicked += AddCategory;
        hbox.Add(button1);
        hbox.Add(button2);

        VBox vbox1 = new VBox();
        vbox1.PackStart(hbox, false, false, 2);
        vbox1.PackStart(viewer, true, true, 2);

        return vbox1;
    }

    private void AddCategory(object o, EventArgs args) {
        CategoryListStore.
            AppendValues(new OrComp.Category(„“, courses[0], db.data));
    }

    private void RenderCatName(TreeViewColumn column, CellRenderer cell,
            TreeModel model, TreeIter iter) {
        OrComp.Category category =
            (OrComp.Category)model.GetValue(iter, 0);
        (cell as CellRendererText).Text = category.Name;
```

```
}
private void EditCatName(object o, EditedArgs args) {
    TreeIter iter;
    CategoryListStore.GetIter (out iter, new TreePath (args.Path));
    OrComp.Category category =
        (OrComp.Category) CategoryListStore.GetValue(iter, 0);
    category.Name = args.NewText;
    category.Save(db.data);
}

private void RenderFirstTime(TreeViewColumn column, CellRenderer cell,
        TreeModel model, TreeIter iter) {
    OrComp.Category category =
        (OrComp.Category) model.GetValue(iter, 0);
    (cell as CellRendererText).Text = category.FirstTime.ToString();
}
private void EditFirstTime(object o, EditedArgs args) {
    TreeIter iter;
    CategoryListStore.GetIter (out iter, new TreePath (args.Path));
    OrComp.Category category =
        (OrComp.Category) CategoryListStore.GetValue(iter, 0);
    try {
        category.FirstTime = Standard.String2Int32(args.NewText);
        category.Save(db.data);
    }
    catch {
    }
}

private void RenderInterval(TreeViewColumn column, CellRenderer cell,
        TreeModel model, TreeIter iter) {
    OrComp.Category category =
        (OrComp.Category) model.GetValue(iter, 0);
    (cell as CellRendererText).Text = category.Interval.ToString();
}
private void EditInterval(object o, EditedArgs args) {
    TreeIter iter;
    CategoryListStore.GetIter (out iter, new TreePath (args.Path));
    OrComp.Category category =
        (OrComp.Category) CategoryListStore.GetValue(iter, 0);
    try {
        category.Interval = Standard.String2Int32(args.NewText);
        category.Save(db.data);
```

```
            }
            catch {
            }
        }

        private void RenderCourse(TreeViewColumn column, CellRenderer cell,
                TreeModel model, TreeIter iter) {
            OrComp.Category category =
                (OrComp.Category)model.GetValue(iter, 0);
            (cell as CellRendererCombo).Text = category.Course.Name;
        }
        private void EditCourse(object o, EditedArgs args) {
            TreeIter iter;
            CategoryListStore.GetIter (out iter, new TreePath (args.Path));
            OrComp.Category category =
                (OrComp.Category)CategoryListStore.GetValue (iter, 0);
            category.Course = data.Course(args.NewText, db.data);
            category.Save(db.data);
        }

    }
}
```

Dodatok B

Obsah priloženého CD

CD obsahuje adresár `LukasPolacek`. V tomto adresári sú zdrojové kódy k aplikácii. Pre spustenie v Linuxe treba mať nainštalované Mono, Gtk#, Glade#, Cairo a prípadne NAnt. Adresár treba skopírovať a spustiť v adresári `bin` aplikáciu príkazom `mono Main.exe`. V adresári `bin` sa nachádza hotový súbor GPS.oxz. Po spustení aplikácie klikneme na otvoriť.

Ak sa nepodarí aplikáciu spustiť, pravdepodobne to bude kvôli starším verziám GTK#. Preto treba aplikáciu skompilovať – v príkazovom riadku napísať `nant clean` a potom `nant`.

Literatúra

[.NET] .NET Framework Class Library. http://msdn2.microsoft.com/en-us/library/d11h6832(VS.71).aspx. © 2007 Microsoft Corporation

[Mono] Mono Documentation http://www.go-mono.com/docs.

[Mapy] Všetky mapy použité v tejto bakalárskej práci sú majetkom KOB Sokol Pezinok

[POB] Pravidlá orientačného behu. http://www.orienteering.sk/doc/prav_ob.pdf Slovenský zväz orientačných športov, 2004

[ISOM] International Specification for Orienteering Maps 2000 http://www.orienteering.org/i3/index.php?/iof2006/content/download/866/4026/file/International%20Specification%20for%20Orienteering%20Maps%202000.pdf

[ISOSM] International Specification for Sprint Orienteering Maps 2007 http://www.orienteering.org/i3/index.php?/iof2006/content/download/1170/5529/file/International%20Specification%20for%20Sprint%20Orienteering%20Maps%202007.pdf

[UML] Jim Arlow, Ila Neustadt UML a unifikovaný proces vývoje aplikací Computer Press, 2003