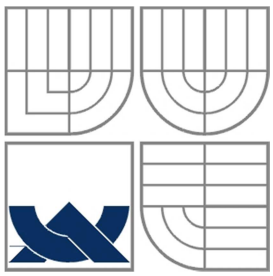
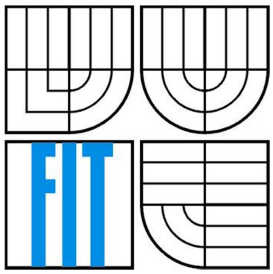


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

APLIKACE PRO PODPORU GEOCACHINGU

INFORMATION SYSTEM FOR GEOCACHING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. MICHAL KUCHTA

VEDOUCÍ PRÁCE
SUPERVISOR

ING. TOMÁŠ KAŠPÁREK

BRNO 2012

Abstrakt

Práce se zabývá představením hry Geocaching, zejména z pohledu aplikace pro správu dat. Věnuje se studiu existujících aplikací, vyjmenovává jejich výhody a nevýhody. V druhé části práce jsou z tohoto studia formulovány požadavky na aplikaci, která řeší nedostatky studovaných existujících produktů. Následně jsou požadavky podrobněji rozepsány a vyplývá z nich návrh aplikace. Dále je popsána samotná implementace aplikace, zejména je poukázáno na různé problémy, které bylo nutno během implementace řešit. V závěru je obsaženo shrnutí a nastíněn budoucí vývoj aplikace.

Abstract

The work contains presentation of the Geocaching game, mainly from the view of an information system. It contains study of existing applications with their advantages and disadvantages. There is a specification of new application, that should solve major disadvantages of existing products in second part of the work. Design of such application, that meets the requirements, follows. The application is implemented, which is described in following chapters, with the aim to target specific problems that were solved during the implementation. There is summary and future plans cotained in the closing chapter.

Klíčová slova

Geocaching, informační systém, keš, cache, geocache, waypoint, GPS, navigace, Qt, C++, multiplatformní

Keywords

Geocaching, information system, cache, geocache, waypoint, GPS, navigation, Qt, C++, multiplatform

Citace

Kuchta Michal: Aplikace pro podporu geocachingu, diplomová práce, Brno, FIT VUT v Brně, 2012

Aplikace pro podporu geocachingu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Tomáše Kašpárka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Michal Kuchta
23.5.2012

Poděkování

Rád bych poděkoval Ing. Tomáši Kašpárkovi za odborné vedení při tvorbě práce, osloveným testerům za věcné připomínky k aplikaci a dále komunitě okolo hry geocaching, kteří se zasloužili o to, že tato hra vůbec existuje.

Obsah

1 Úvod

Tato práce se zabývá návrhem a implementací aplikace pro podporu geocachingu na počítačích primárně se systémem MS Windows a Linux.

Úvodní kapitoly obsahují stručný popis hry Geocaching v rozsahu potřebném pro určení požadavků na aplikaci. Dále jsou rozebrány existující programy, jsou vyjmenovány jejích výhody a nevýhody.

Ve třetí kapitole jsou specifikovány požadavky na aplikaci, které vyplývají z úvodních kapitol a z identifikovaných výhod a nevýhod existujících produktů.

Další kapitoly se věnují samotnému návrhu aplikace pro podporu Geocachingu vyplývající ze specifikované funkcionality.

Následuje popis implementace aplikace, zejména je kladen důraz na popis problémů, kterým bylo nutno během implementace čelit.

Závěrečná kapitola shrnuje přínos aplikace a nastiňuje její další vývoj v budoucnu.

1.1 Poznámka o terminologii

V této práci je použito množství pojmů, které nejsou přeloženy do češtiny, nebo se jedná o anglické výrazy použité s českým skloňováním. Jedná se o pojmy, které se v široké komunitě okolo Geocachingu vžily v té formě, v jaké jsou v práci použity. Z důvodu zachování shody s dalšími zdroji a vyjadřováním hráčů jsem se rozhodl použít osvědčenou a vžitou terminologii, aby čtenář práce neměl problém orientovat se v jiných zdrojích, běžně takovou terminologii používajících.

Tyto pojmy jsou v textu práce zvýrazněny kurzívou.

Dále používám některé anglické pojmy z oblasti programování, které jsem se rozhodl také nuceně nepočešťovat, protože se běžně používají jejich anglické originály. Tyto pojmy jsou také zvýrazněny kurzívou.

2 Stručně o Geocachingu

Geocachingu se věnuje již má bakalářská práce [1], zabývající se návrhem a vývojem aplikace pro zařízení Windows Mobile. V jejích úvodních kapitolách je Geocaching podrobně rozebrán, v této práci se tedy již nebudu opakovat a zmíním pouze stručně základní informace a dále se již budu věnovat podrobně pouze konkrétním částem které jsou podstatné pro potřeby aplikace.

Geocaching je celosvětová navigační hra využíjající technologie GPS k hledání ukrytých schránek (*keší*, anglicky *cache*) v přírodě. Přesný počet lidí, věnujících se Geocachingu není znám, avšak předpokládá se, že se jedná o nejméně několik desítek tisíc lidí. Na začátku roku 2012 bylo po

celé Zemi ukryto více než jeden a půl milionu schránek. Geocachingu se může věnovat každý, bez ohledu na společenské postavení nebo schopnosti. Každý si v něm najde to, co mu vyhovuje.

Geocaching začíná u počítače. Hráč (*geocacher*, česky se též vžilo označení *kačer*, *geokačer*) se musí zaregistrovat na oficiálních stránkách www.geocaching.com, kde pak nalezne veškeré informace o existujících *keších*. Nejdůležitější informací, kterou každá *keš* musí mít, je právě údaj o poloze – její zeměpisné souřadnice. Ty si hráč zadá do svého nagivačního přístroje, kterým může být třeba chytrý mobilní telefon s navigační aplikací nebo specializované GPS zařízení.

Jakmile má hráč ve své GPS nahrané souřadnice *keší*, vyráží do terénu hledat ukrytou schránku. Po jejím nalezení se zapíše do papírového sešitu (tzv. *logbooku*), který je v keši obsažen a kdy pouze zápis v něm opravňuje hráče k zapsání nálezu na oficiálních internetových stránkách. Takovému zápisu se říká *log*. *Keš* pak znovu ukryje a zamaskuje přesně stejně jak byla, když ji nalezl. Může také vyměnit něco z „pokladu“, který je v keši umístěn – různé drobné předměty, hračky, knihy, ... Musí ale zachovat princip takzvaného *fair-trade*, tedy spravedlivé výměny. Předmět, který hráč z *keše* vezme, musí nahradit předmětem stejné nebo vyšší hodnoty, aby se tak zabránilo postupné degradaci obsahu.

Samotné provedení schránky, ukryté někde v terénu, se vždy liší. *Keš* lze udělat z čehokoliv, kde je možné umístit nějaký prostor pro zapsání hráčů. Nejzajímavější je například magnetická guma umístěna jako nějaký nápis, kde z druhé strany je umístěn logbook. *Keše* mohou být i velmi technicky propracované, mohou například obsahovat nějaké elektronické zařízení nebo být jen mistrně řemeslně zpracované, například v podobě ptačího krmítka, v jehož střeše je umístěna samotná schránka chráněna kódovým zámkem před neoprávněným zásahem lidí, kteří geocaching nehrají.

Po příchodu zpět domů se hráč opět přihlásí na stránky www.geocaching.com, kde zaznamená svůj pokus o nález, ať již úspěšný či nikoliv.

Tímto způsobem, tedy ručním opisováním souřadnic do navigace a vybíráním konkrétních *keší*, které chce hráč navštívit, je hru samozřejmě možné provozovat a určité skupině lidí (převážně začátečníkům) to takto vyhovuje a stačí. Pokud ale chce hráč najít za den třeba i několik desítek *keší*, nebo neví, které přesně chce nalézt, je výhodné mít nějaký způsob, jak do navigace nahrát větší množství *keší* bez nutnosti přepisovat ručně všechny souřadnice. Přímo oficiální internetové stránky www.geocaching.com poskytují funkce, kterými je možné stáhnout údaje o větším počtu *keší* ve standardizovaném formátu GPX [2], se kterými pak umí některé navigační aplikace a přístroje pracovat. Tato funkce se nazývá *Pocket Query*, zkráceně *PQ*. Problém nastává, pokud uživatel nemá navigaci, která s formátem GPX umí pracovat, nebo pokud si chce u *keše* poznačit nějaké další údaje, jako je třeba doporučená přístupová trasa nebo nějaká důležitá poznámka (kód ke kódovému zámku bránícímu přístup cizích lidí, otevírací hodiny, ...).

Dalším problémem je uložení finálních souřadnic u *mystery* a *multi* keší (viz kapitola 2.2), uložení způsobu řešení, data a času nálezu a dalších údajů, které jsou pro hráče zajímavé a zaslouží si uschovat. O tom ale více v kapitole 2.4.

2.1 Historie

Geocaching vznikl dne 3.5.2000 v USA, přesně den poté, co tehdejší americký prezident Bill Clinton nařídil vypnout zavádění umělé chyby do systému GPS, kdy se jeho přesnost zlepšila z desítek či stovek metrů na jednotky metrů. Onoho osudného dne jistý Dave Ulmer z Oregonu schoval první schránku poblíž svého domu (tehdy se jí ještě neříkalo *cache*, tento výraz se vyvinul až později). Hned druhý den byla tato schránka objevena dalším budoucím geocacherem a během prvního měsíce bylo na území USA už několik desítek schránek. Na původním místě je dnes samozřejmě také ukryta *keš*, byť to již není ta originální. Od té doby se začal rozšiřovat jak počet geocacherů tak počet uložených schránek [3]. V České Republice byla první *keš* založena 1.6.2001 a dodnes funguje [4].

2.2 Typy keší

V Geocachingu se postupně vyvinulo několik typů *keší*, které se liší jak provedením v terénu, tak obsahem webové stránky každé keše, tzv. *listingu*. V následujících podkapitolách popíši jednotlivé typy, protože tato problematika je pro aplikaci zcela zásadní. Každý typ *keše* má také na oficiálních stránkách svoji specifickou ikonku pro snadné rozlišení.

2.2.1 Tradiční keš (Traditional Cache)

Tyto *keše* jsou tím nejzákladnějším, co v geocachingu existuje. Hráč se rovnou dozví souřadnice umístění schránky a tu jde hledat. V listingu se pak hráč může dočíst nějaké další informace o místě, kde se schránka nachází, například historii památky, v jejíž blízkosti je *keš* ukryta. Tato forma *keší* je nejrozšířenější a mezi podstatnou částí hráčů také nejoblíbenější, protože jediným jejím cílem je dostat hráče ven ať už do přírody či do města – a to je pravděpodobně jediná věc, kterou mají úplně všichni hráči společnou – rádi se pohybují venku.

2.2.2 Multi keš (Multi-cache)

Tento typ *keše* vychází z *tradiční keše*, ale pro objevení schránky s *logbookem* je nejprve potřeba navštívit několik míst, které postupně obsahují informace (nejčastěji opět zeměpisné souřadnice) následujícího stanoviště. Často je také potřeba na daném místě něco zjistit – například spočítat okna na stěně budovy – a z tohoto údaje následně souřadnice další zastávky vypočítat.

Tento typ *keší* je velmi rozšířen zejména v Rakousku a Německu, v České Republice není tolik oblíben.

2.2.3 Mystery keše (Unknown cache)

Mystery keše jsou nejčastěji schránky, kde ke zjištění souřadnic je nutné vyřešit nějakou šifru nebo podobný úkol. Vyřešení šifry, uvedené v *listingu keše*, však ještě nemusí vést k nálezu – opět to mohou být pouze souřadnice nebo indicie k nalezení první ze zastávek, kde opět může být další šifra nebo jiný úkol, tedy princip podobný jako u *multi keší*. Důležitým rozdílem mezi *multi kešemi* a *mystery kešemi* je však v tom, že mystery mohou obsahovat šifry, tedy že k jejich nálezu nestačí umět jen základní matematiku a zadávat souřadnice.

Do mystery keší se také počítají všechny další způsoby provedení, které nespadají do žádné z dalších kategorií.

Mystery keše jsou velice oblíbené mezi hráči, které baví keše zakládat – mohou zde totiž téměř bez omezení uplatnit svého tvůrčího ducha. Mnoho hráčů pak takové šifry baví luštit, protože se třeba také účastní různých šifrovacích her, takže mají spoustu zkušeností a je to pro ně výzva. Stejně tak ale existuje i značná skupina hráčů, které luštění šifer nebaví. Pokud vzniká velké množství *mystery keší* na úkor těch tradičních, hráče přestane bavit neustále luštit další a další šifry, takže se pak často přiklánějí k různému více či méně tajnému sdělování finálních souřadnic, aby nemuseli vše luštit a jen mohli jít pro krabičku v přírodě.

2.2.4 Wherigo cache

Speciálním typem *mystery keší* jsou takzvané *Wherigo cache*. Vznikly přidáním interaktivního prvku do hledání. K odlovu *Wherigo keše* potřebuje hráč speciální vybavení – tzv. přehrávač, který nainstaluje do svého mobilního telefonu, nebo který je přímo obsažen v jeho turistické navigaci.

Do tohoto přehrávače nahraje tzv. *cartridge*, která je ke stažení na specializovaném webu www.wherigo.com a následně postupuje podle instrukcí, které *cartridge* obsahuje. Je tak možné vytvořit lokačně závislý příběh, který donutí hráče ho prožít a zažít tak dobrodružství přesně tak, jak to autor zamýšlel. Opět je možné příběh kombinovat s úkoly nebo šiframi.

Wherigo cache nejsou pro každého – jak již bylo zmíněno, je nutné vlastnit kompatibilní mobilní telefon, nebo specializovanou turistickou navigaci, která přehrávač obsahuje. Problematické je také zakládání nových *Wherigo cache*. Projekt *Wherigo* byl spuštěn v létě roku 2008 a od té doby se mu nedostalo žádné oficiální podpory (byť oficiální stránky www.wherigo.com jsou udržovány ve funkčním stavu). Oficiální nástroj pro sestavování *cartridge* v podstatě nefunguje a je značně uživatelsky nepřívětivý. Oficiální přehrávač existuje pouze pro dnes již mrtvou platformu Windows Mobile a nejsou žádné zprávy o tom, že by se to mělo v budoucnu změnit.

Naštěstí komunita vzala věc do vlastních rukou, takže dnes již existuje kvalitní software pro tvorbu *cartridge* a také existují neoficiální přehrávače pro mobilní telefony vybavené Javou, stejně tak jako aplikace pro operační systém Android nebo iPhone a iPad. Stojí za zmínku, že všechny tyto nástroje vznikly v České Republice jako díla českých autorů.

I přes nedostatek oficiální podpory je v ČR *Wherigo* velmi oblíbené a jsme jedni z nejaktivnějších tvůrců *wherigo keší* na světě.

Stejně jako všechny doposud uvedené typy *keší*, i *Wherigo keš* je zakončena nálezem schránky obsahující minimálně *logbook*.

2.2.5 Letterbox Hybrid keše

Letterboxing [5] je hra podobná Geocachingu. Hráči postupně dostávají slovní instrukce, kde hledat další stanoviště. Jedná se tedy v podstatě o *multi keš*, avšak bez přesně určených souřadnic stanoviště. Další odlišností Letterbox keší je právě jejich spojení s Letterboxingem – v krabičce je kromě *logbooku* i razítko, které si hráči letterboxingu otisknou do svého deníčku. Toto razítko, na rozdíl od ostatních předmětů, neslouží k výměně.

2.2.6 Earthcache

Existují i *keše* bez fyzické schránky. Tedy takové, kde právo k zalogování keše na internetu nevzniká zapsáním se do fyzického logbooku, ale splněním nějaké předem dané podmínky autorem takové keše.

Earthcache jsou jedním z případů takovýchto *keší*. Jsou umístěny na nějakém geologicky zajímavém místě a úkolem pro uznání *logu* obvykle bývá zodpovědět nějaké geologicky zaměřené otázky týkající se dané lokality. Odpovědi na tyto otázky by měly být zjistitelné na místě například s pomocí informační tabule, avšak ne vždy je to pravda a někdy je třeba informace dohledávat později na internetu.

2.2.7 Virtual Cache

Dále existují *keše*, které nejsou spojeny s nějakou geologickou zajímavostí, ale jsou umístěny na místě, kde není možné umístit fyzickou schránku nebo se o ní autor nemůže starat. Podmínkou pro uznání *logu* je opět splnění nějaké podmínky – například pořízení fotografie nebo zodpovězení otázky.

Virtuální keše už však od konce roku 2005 není možné zakládat a tak pouze dožívají ty, které byly založeny před tímto datem. Z *virtuálních* a *webcam* (viz následující kapitola) *keší* se později vyvinula speciální hra Waymarking.

2.2.8 Webcam cache

Na světě existuje množství kamer, jejichž snímky jsou veřejně dostupné na internetu. Některé z nich jsou zároveň *keše*. Úkolem hráče je pořídit svoji fotku vyfocenou danou webkamerou. Tuto fotku poté přiloží k elektronickému *logu* na internetu a tím je mu tento *log* uznán.

Stejně jako *virtuální keše*, *webcam cache* již není možné od konce roku 2005 zakládat, pouze dožívají ty stávající. Webové kamery byly opět přesunuty pod Waymarking.

2.2.9 Event cache

Všude, kde existuje nějaká komunita, existuje touha po vzájemném setkávání. Za tímto účelem existují tzv. *eventy*, neboli oficiálně *Event cache*. Jedná se o setkání *geocacherů* s nějakým předem daným programem. Bývá zvykem mít na *eventu* k dispozici *logbook* jako v případě klasické *keše*, kam se účastníci setkání zapíší a poté *zalogují* na internetu stejně jako v případě *tradičních keší*. Přítomnost logbooku na eventu však není podmínkou, účast na eventu lze zalogovat i pouze v případě, že se hráč krátce dostaví na místo konání. Takovéto chování je ale považováno za bezohledné a hráč takovýmto způsobem navštěvující *eventy* není komunitou přijímán v dobrém.

Náplň takového setkání může být prakticky libovolná. Může jít pouze o společné posezení v nějakém zařízení restauračního typu, nebo například o adrenalinový závod, kde celé týmy hráčů prochází vytyčenou trasu a plní různé úkoly.

Event má jasně dané datum konání a po proběhnutí je *archivován*, tj. zrušen, čímž se přestane zobrazovat ve výsledcích vyhledávání na oficiálních stránkách. Doba, kdy je možné *zalogovat* účast je tedy předem omezena. Setkání může trvat od pár minut (takovému *eventu* se pak říká tzv. *flash-mob event*) až po několik dní, kdy je pro účastníky zajištěno stravování a nocleh.

Existují některé zakázané aktivity, jimž se event věnovat nemůže, avšak to pro potřeby této práce není důležité, zájemci si mohou přečíst oficiální pravidla [6].

2.2.10 Cache In, Trash Out Event (CITO)

Jedná se o speciální druh *eventu*, kdy hráči společně provedou úklid nějakého prostoru od nepořádku. Účastí na *CITO eventu* demonstrují, že mají úzký vztah s přírodou a není jim její osud lhostejný. *CITO eventy* mají často podporu od oficiálních institucí, kdy taková instituce například zařídí přistavení kontejneru pro odpad, dodání pracovních nástrojů, rukavic nebo igelitových pytlů.

2.2.11 Mega event

Jak již název napovídá, jedná se o setkání geocacherů ve velkém. Tím velkém je konkrétně myšleno účast alespoň 500 různých uživatelských účtů zaregistrovaných na www.geocaching.com.

Jakmile je dosažena hranice 500 účtů, mohou organizátoři *eventu* požádat o jeho přepnutí na *mega event*. *Mega eventy* jsou poměrně vzácné, protože sehnat dohromady 500 různých hráčů není lehký úkol. Ročně je tak pořádáno pouze několik takových akcí na celém světě, zatímco běžných *eventů* jsou tisíce.

Na rozdíl od *eventu*, *mega event* musí mít *logbook*, aby bylo možno prokázat přítomnost 500 různých hráčů. Vzniká tím docela zajímavý paradox, kdy pořadatel *eventu* vůbec nemusí tušit, že se

mu nakonec na akci sejde více než 500 lidí, nepřipraví *logbook* a přijde pak o možnost požádat o dodatečnou změnu na *mega event*. Proto je, alespoň v ČR, zvykem mít na *eventu logbook* vždy. Navíc pro organizátora takové akce je *logbook* příjemnou vzpomínkou na proběhlé setkání.

2.2.12 Další typy keší

Existují ještě některé další vzácné typy *keší*, avšak ty již jsou buďto všechny *archivovány*, nebo se vyskytují pouze v jednotkových exemplářích na území Spojených států Amerických, takže není třeba se jim podrobněji věnovat. Zájemci si mohou o těchto typech přečíst na oficiálních stránkách [7].

2.3 Hustota keší

Důležitou částí pravidel je pravidlo o hustotě *keší* [8]. Toto pravidlo se začíná uplatňovat v místech, kde je nahuštěných hodně *keší* a říká, že žádné dvě fyzické schránky dvou různých *keší* by neměly být blíže než 0,1 míle (tedy 161m). Blíže než 161m mohou být dvě fyzická stanoviště jedné a té samé *keše*, nebo fyzické stanoviště jedné *keše* a nefyzické stanoviště jiné *keše*. Například místo, kde se mají počítat okna na domě může být umístěno 50m od fyzické *tradiční keše*, avšak na tomto místě už nesmí být umístěna další schránka se souřadnicemi, ta může být až ve vzdálenosti větší než 161m od *tradiční keše*.

Stejně tak, pokud existuje *multi keš*, kde jednotlivé zastávky s ukrytými souřadnicemi jsou od sebe vzdáleny pouze 100m, není to problémem. V této vzdálenosti už nesmí být žádná fyzická schránka z jakékoliv jiné *keše*, pro takovou opět platí pravidlo o 161m.

Toto pravidlo má ten smysl, aby hráči náhodou nenašli stanoviště nebo krabičku z jiné keše, než kterou právě hledají – zabraňuje se tak zmatkům s tím, co hráč vlastně našel. Pokud nějaká nově zakládaná keš toto pravidlo poruší, nedojde k jejímu zveřejnění reviewerem [9] a říká se, že taková keš je *v kolizi* s jinou. Autor takové keše musí problém s *kolizí* vyřešit posunem své schránky, aby mohla být keš zveřejněna. Ve výjimečných případech může být z tohoto pravidla udělena výjimka, kdy vždy dochází k individuálnímu posuzování *reviewerem* a nemusí být tomuto požadavku vyhověno.

Vhodné je, pokud se hráč snaží vzniku *kolize* zabránit už během prvotního výběru místa tak, aby bylo dostatečně daleko od všech dalších schránek. Proto se doporučuje, aby hráč pokud možno před založením vlastní *keše* našel všechny *keše* v oblasti, kde chce svoji *keš* umístit. Problémem je, že informace o *kolizích* není nikde na oficiálních stránkách k dispozici, tuto informaci mají pouze *revieweři*. Pro hráče je tedy obtížné udržet přehled o *kolizních* vzdálenostech, pokud si pečlivě neuchovává databázi informací o *keších* právě v nějaké specializované aplikaci.

2.4 Geocaching z pohledu aplikace

V této kapitole se věnuji Geocachingu z pohledu aplikace pro správu *keší*. Zmíním, jaké jsou vlastnosti *keší*, se kterými musí taková aplikace pracovat, a co vše je o keších nutno uchovávat.

2.4.1 Vlastnosti keší

Každá *keš*, libovolného typu, má několik základních údajů. Podle těchto údajů je možné *keše* vyhledávat, třídit nebo řadit.

2.4.1.1 GC kód

Každá *keš* má svůj jedinečný identifikační kód, který začíná písmeny GC a následuje kombinace číslic a písmen anglické abecedy. Pomocí tohoto kódu je možné na *keš* snadno odkazovat. Kód je tvořen postupně posloupností s vynecháním podobných znaků (v posloupnosti je pouze číslo 0, nikoliv písmeno O a podobně), takže později založená *keš* má vyšší kód než ta dříve založená.

Pro *waypointy* (viz kapitola 2.4.3) se místo prefixu GC použije prefix specifikovaný u *waypointu* a zbytek kódu zůstává stejný jako u *keše*, ke které *waypoint* patří. Pokud tedy máme *keš* s kódem GC3018T, pak její parkovací waypoint s prefixem PA bude mít kód PA3018T. Tak je možno snadno spárovat *keš* s jejími *waypointy*.

GC kód má přímou vazbu na identifikátor keše – je možné ho převést na unikátní číslo odpovídající ID keše v oficiální databázi stránek geocaching.com a zpět. Konkrétní převod se provádí tak, že pro kódy do hodnoty GCFFFF je použito převedení z hexadecimální soustavy do dekadické (a naopak v případě opačného převodu), pro hodnoty začínající GC10000 a vyšší je použito 31 znaků anglické abecedy a číslic.

2.4.1.2 Souřadnice

Již zmíněnou základní vlastností každé *keše* jsou její zeměpisné souřadnice. Jedná se o zeměpisnou šířku a délku, kdy v geocachingu se obvykle používá formát N/S DD° MM.MMM‘ W/E DD° MM.MMM‘ – tedy nejprve určení polokoule (severní, jižní, západní, východní) a poté celá část stupňů následována minutami ve formě desetinného čísla s přesností na tři desetinná místa s doplněním nul, pokud jsou souřadnice tvořeny méně než třemi desetinnými místy.

V aplikaci je však možno ukládat souřadnice v počítačově lépe zpracovávatelném formátu dvou desetinných čísel se znaménkem (pro severní a východní polokouli kladné, jinak záporné) a konverzi na toto zobrazení provést až v případě zobrazení souřadnic uživateli.

Spolu se souřadnicemi je u každé *keše* veden údaj o její státní příslušnosti a v případě, že pro daný stát existuje rozdělení do krajů nebo obdobných územních celků, je k dispozici i tento údaj.

2.4.1.3 Stav keše

Každá *keš* může být buďto k dispozici – standardní stav, kdy je keš připravena, je hledatelná a je s ní vše v pořádku. Dále může být *disabled* (nedostupná) – v případě, že je s *keší* nějaký dočasný problém – například se ztratila a je nutno na místo umístit novou schránku. Takovouto *keš* obvykle není možné nalézt a hráči musí počkat, až autor provede údržbu a dá vše do pořádku.

Poslední možností je *archivovaná keš* – tato *keš* byla fyzicky zrušena a nelze ji tedy hledat. *Archivované keše* se běžně nezobrazují ve výsledcích vyhledávání, avšak pokud se nějak hráč dostane k *listingu* (například přes seznam nalezených nebo založených *keší* jiného hráče), je tento zobrazen pro možnost prohlížení.

2.4.1.4 Velikost, obtížnost, terén

Další vlastností každé *keše* je její velikost – jedná se o přibližné rozdělení keší do 6 velikostních kategorií – *micro, small, regular, large, unknown* a *not chosen*.

Mikro schránka obsahuje pouze *logbook* případně tužku a nevejdou se do ní žádné předměty na výměnu. Typicky se může jednat o krabičku od 35mm filmu, nebo o PET prefabrikát. Mikro schránky jsou oblíbeným úkrytem v osídlených oblastech, kdy schovat větší schránku může být nemožné.

Velikosti *small* a *regular* jsou větší krabičky, kam se vejde větší notýsek jako *logbook*, tužka a drobné předměty na výměnu. Liší se kapacitou schránky – za schránku velikosti *regular* je považována krabička o objemu alespoň 1l.

Large keše jsou *keše* s objemem alespoň 20l. Jsou to největší možné *keše*, kam se vejdou třeba celé knihy nebo velké hračky. Schovat *large keš* je největší problém, protože existuje pravidlo zakazující *keše* zakopávat, takže najít vhodné místo pro uložení je značně problematické.

Velikost *unknown* se použije v případě, že velikost samotné *keše* není možné určit – například pokud je *keš* celá budova muzea betlémů (taková skutečně v Jižních Čechách existuje), jedná se o velikost keše *unknown*, protože do ní nelze umístit předměty na výměnu, přesto se svou velikostí nejvíce podobá keši velikosti *large*.

V případě, že velikost keše nespadá do žádné z uvedených kategorií, použije se velikost *not chosen*. Jedná se například o schránku ještě menší než *mikro*, kdy *logbookem* je pouze tenký proužek papíru. Může se jednat o *keš*, kdy nemá žádnou schránku a pouze je někde schován *logbook* (již zmíněný případ magnetické gumy) a další podobné případy.

Existuje ještě další zvláštní velikost *keší* označena jako *virtual*, která je použita pro všechny *keše*, které nemají žádné fyzické stanoviště – tedy pro keše typu *virtual*, *earthcache* a *webcam*.

Každá *keš* má také informaci o její terénní přístupnosti a náročnosti nalezení – označené jako obtížnost (*difficulty*) a terén (*terrain*). Toto hodnocení je označeno hvězdičkami v počtu od 1 do 5 s půlhvězdičkovými kroky.

Terén 1 značí *keš* přístupnou pro vozíčkáře, zatímco *keš* s terénním hodnocením 5 je přístupná pouze se speciálním (například horolezeckým nebo potápěčským) vybavením.

Obtížnost 1 znamená, že *keš* je snadno nalezitelná a obvykle ji hráč objeví na první pohled, zatímco obtížnost 5 znamená, že se hráč musí vrátit na místo třeba i několikrát, protože je *keš* schovaná velice náročně. U *mystery keší* má obtížnost obvykle význam hodnocení složitosti rozluštění souřadnic umístění schránky, kdy obtížnost 5 vyluští jen malé procento hráčů, zatímco obtížnost 1 by měl zvládnout každý.

2.4.1.5 Text listingu

Součástí každé keše je text *listingu*. Jedná se o dvě políčka – tzv. *short description* a *long description*. *Short description* je omezeno 500 znaky a obsahuje stručné informace o *keši*. *Long description* nemá známé délkové omezení a může obsahovat další informace, například zadání šifry v případě *mystery keší* nebo informace o místě, kde je keš uložena. Obě tyto políčka mohou obsahovat volitelně HTML kód.

2.4.1.6 Hint

Důležitou součástí listingu každé keše je *hint* – tedy nápověda. Naprostá většina *keší* je *hintem* vybavena. *Hint* použijí hráči v případě, že nemohou keš najít nebo přijít na řešení šifry. Měl by jim napovědět, jak dále postupovat. Jedná se o textové políčko, které je v listingu zašifrováno posuvem abecedy – tzv. ROT13 [10].

2.4.1.7 Zakladatel

Každá *keš* má jednoznačně určený uživatelský účet, který za ní má zodpovědnost. Je to právě ten účet, který založil *listing* keše, případně ten, kdo takový listing později *adoptoval* (tedy převzal od předchozího majitele). Na tvorbě *keše* se ale mohou podílet i další lidé, a proto je možnost určit manuálně, kdo všechno *keš* založil. Slouží k tomu speciální textové políčko, které majitel *listingu* může vyplnit libovolným textem, obsahující seznam osob nebo název týmu, který *keš* založil. Toto políčko má pouze informativní charakter a osoby v něm uvedené nemají žádné další pravomoci s *listingem* manipulovat. Toto právo připadá pouze vlastníkovi *listingu*.

2.4.1.8 Další vlastnosti

Další vlastností *keše* je její datum umístění. Toto datum vyplňuje vlastník a mělo by zachycovat datum uložení schránky v terénu, případně datum zveřejnění *listingu*. V případě *eventů* se jedná o datum konání, resp. první den konání v případě vícedenních akcí.

Ke každé *keši* je také možno připojit odkaz na referenční webovou stránku. Tento odkaz může hráčům poskytnout další informace o lokalitě, může vést například na oficiální stránky zámku, kterému se *keš* věnuje. U *mystery keší* může tento odkaz vést na nutnou součást řešení a u *wherigo keší* obvykle vede na stránku www.wherigo.com, kde je ke stažení *cartridge* pro splnění keše.

Uživatelé mohou *keš* ohodnotit udělením bodu oblíbenosti. Podle počtu obdržených bodů oblíbenosti je pak možné vypočítat procentní hodnocení *keše* podle toho, kolik z úspěšných nálezců *keš* ohodnotilo bodem.

Dále je ke každé *keši* možno umístit několik obrázků. Tyto obrázky jsou poté v *listingu* uvedeny v jednoduchém seznamu. Obrázek může například obsahovat fotku zachycující místo uložení, což může hráčům pomoci při hledání skrýše v terénu v místech, kde je špatný GPS signál, takže určení pozice není přesné.

2.4.2 Atributy

Každá z *keší* může mít nějaké *atributy*. Jedná se o ikonky symbolizující omezení nebo doporučení pro odlov *keše*. Může obsahovat například informaci o tom, že *keš* je nutno lovit v noci, nebo naopak to není v noci doporučeno. Může obsahovat informaci o tom, že keš není lovitelná v případě sněhové pokrývky, nebo že k jejímu nalezení je potřeba speciální nářadí – například šroubovák. Dále může obsahovat například informaci o tom, že keš je vhodná pro děti a nebo je naopak umístěna na nebezpečném místě, kde si musí dávat pozor i dospělí, aby se jim něco nestalo. *Atributy* mohou také obsahovat různé zákazy – například že keš není dostupná autem, tedy je nutné pro ni jít nějaký kus pěšky.

Atributů je mnoho a postupně podle potřeby vznikají další, takže uvádět jejich kompletní seznam nemá smysl, aplikace musí počítat s možností specifikovat libovolný atribut. U každého *atributu* je třeba uchovávat informaci o tom, jestli je dané keši přiřazen a jestli je přiřazení pozitivní nebo negativní – tedy „povolení“ nebo „zakázání“. Zakázaný *atribut* se na oficiálních stránkách projevuje červeným přeškrtnutím zatímco povolený atribut se objevuje bez přeškrtnutí. Nepřiřazené *atributy* se nezobrazují.

2.4.3 Additional waypoints (další body zájmu)

Ke každé keši je možné přiřadit další referenční body. Mohou být následujících typů: Finální umístění (Final location), Parkoviště (Parking), Otázka k zodpovězení (Question to answer), Referenční bod (Reference point), Stanoviště *multi keše* (Stage of multicache), Cesta (Trailhead). Každý bod má svůj název, dvoupísmenný kód, šestimístný *lookup* kód, popisek a samozřejmě souřadnice.

Body typu *Final location* a *Stage of multicache* musí být vyplněny, pokud je keš obsahuje – tj. všechny *mystery*, *multi* a *wherigo keše* musí mít alespoň *final waypoint*, a pokud postupová stanoviště obsahují fyzické schránky, musí být tato místa uvedena jako *waypointy* typu *stage of multicache*. Vyplněním těchto bodů je poté možné efektivně zabránit kolizím v době, kdy je keš kontrolována *reviewerem*.

Hráč nemá možnost si ke *keši* přiřadit vlastní bod, například vyluštěné finální souřadnice. Existuje poměrně čerstvá podpora pro změnu úvodních souřadnic *multi* a *mystery keší*, kam si hráč

může uložit vyluštěné souřadnice, avšak tato podpora není úplná – jde tak uložit jen jeden bod, originální poloha je tím ztracena a není možno přidat žádné další doplňující informace. Indikace, zdali se jedná o upravené souřadnice nebo o originální není uvedena v GPX souborech. Není tedy vhodné tuto funkci na oficiálních internetových stránkách používat, pokud hráč má v úmyslu zároveň používat některou z aplikací pro podporu Geocachingu, protože tato aplikace nemá možnost jak se o upravených souřadnicích dozvědět.

2.4.4 Uživatelská poznámka

Uživatel, který si zaplatil *Premium Membership* (viz kapitola 2.4.6), může ke každé keši přidat vlastní poznámku. Tato poznámka je soukromá a zobrazuje se pouze jemu samotnému. Může si tak uchovat například část řešení nebo poznámku důležitou pro nález *keše* v terénu. Opět bohužel, stejně jako u korigovaných souřadnic (viz kapitola 2.4.3), že tato informace není k dispozici v GPX souborech, uživatelská poznámka je tedy k dispozici pouze na webu geocaching.com.

2.4.5 Logy

U každé keše je také seznam existujících záznamů hráčů – nejčastěji záznamů o nalezení keše (*Found it*, *Attended* pro *eventy* a *Webcam photo taken* pro *webcam keše*), ale *logy* mohou mít i další typy. Nenalezeno (*Didn't find it*) - zaloguje hráč v případě, že keš nenalezl a domnívá se, že na místě není – značí to, že by se autor keše měl jít na místo podívat a zkontrolovat, zdali je vše v pořádku. *Write note* – značí pouze poznámku, kterou chtěl hráč ke keši napsat, aniž by ji hledal a našel případně nenašel. *Needs maintenance* – hráč našel keš poničenou a ta potřebuje údržbu. Posláním tohoto *logu* se automaticky aktivuje *atribut Needs maintenance* a je tak zvýrazněno, že s keší není něco v pořádku. *Log needs maintenance* neznamená automaticky *log* o nalezení schránky, takže v případě, že hráč schránku nalezl a chce poslat tento typ *logu*, musí poslat dva *logy* po sobě – jeden s nálezem a jeden s popisem problému. *Needs archived* – keš je nutno okamžitě zrušit, obvykle z důvodu závažného porušení pravidel. Logem typu *Will attend* dává hráč najevo svůj záměr zúčastnit se *eventu*. Dává tak organizátorovi takového *eventu* najevo, s kolika lidmi má počítat.

Jako majitel keše má pak hráč ještě k dispozici několik dalších logů – *disable listing* (dočasné zneaktivnění listingu keše), *enable listing* (znovuspuštění listingu keše po dočasném disable), *archive* (zrušení – archivace – keše), *owner mainteance* (provedení údržby keše), *update coordinates* (posun souřadnic keše) a u *eventů* speciální log *announcement*, který obešle všechny hráče, kteří zalogovali *will attend* emailem. Tento typ logu se používá, pokud je hráčům třeba sdělit nějakou důležitou informaci.

Každý *log* má svůj typ (viz předchozí odstavce), datum zápisu (bez času), uživatele, který *log* poslal a samozřejmě samotný text *logu*.

2.4.6 Získávání dat

K získání dat ze serveru geocaching.com pro potřeby dalšího využití například v nějaké aplikaci je možné použít dva různé způsoby.

Prvním z nich je využít API (aplikační rozhraní), kterým je server geocaching.com vybaven [11]. Tato funkce je dostupná všem uživatelům, avšak pro různé uživatele jsou aplikovány různé limity, kolik toho smí uživatel stáhnout.

Nevýhodou API je, že je v době psaní této práce stále vedeno jako beta verze a přístup k němu z pohledu vývojáře je omezen pouze na vybrané jedince, kteří si dokáží jeho použití obhájit u firmy Groundspeak. Úspěšným vývojářům je poté přidělen speciální klíč, kterým se k API autentizují. Primárně z licenčních důvodů (takovýto klíč se nesmí dostat do nepovolaných rukou, smí být využit pouze autorem dané aplikace jen pro schválené operace, kdy schvalování provádí firma Groundspeak) není možné prozatím v aplikaci přístup k API použít z důvodu nutnosti zveřejnit kompletní zdrojový kód.

Druhou možností přístupu ke kompletním datům je využití *PocketQuery*. Tato funkce je dostupná pouze uživatelům, kteří si zaplatí rozšířené členství (Premium Membership). Mohou si pak nadefinovat seznamy *keší* odpovídající různým vyhledávacím kritériím a tyto seznamy si poté nechat poslat jako GPX soubor s limitem max. 1000 keší na jedno *PocketQuery*. Vyhledávací možnosti jsou omezené a není možné *keše* vyhledávat podle všech možných kritérií. Pro specifičtější výběr je tedy výhodné stáhnout větší množství keší a vyfiltrovat si jej pomocí některé ze specializovaných aplikací.

3 Existující aplikace

V této kapitole popisuji dvě nejrozšířenější existující aplikace pro Geocaching – GeoGet a GSAK. Oběma se věnuji podrobně, popisuji jejich pozitivní a negativní vlastnosti, ze kterých poté vyplynou požadavky na vlastní aplikaci.

V závěru kapitoly jsou zmíněny některé další existující aplikace, které nejsou tak rozšířené, ale obsahují nějakou zajímavou vlastnost nebo funkcionalitu.

3.1 GSAK

GSAK (neboli Geocaching Swiss Army Knife) je pravděpodobně první aplikace určená pro správu *keší*, která kdy existovala. Je velmi rozšířená ve světě a dostává se jí oficiální podpory ze strany Groundspeaku, provozovatele oficiálních stránek Geocaching.

GSAK je k dispozici pouze pro operační systém MS Windows a jeho největší nevýhodou je, že se jedná o komerční aplikaci, za jejíž používání je třeba zaplatit. Po vypršení zkušební 21 denní lhůty

je uživatel neustále vyrušován vyskakujícími obrazovkami upozorňujícími na nutnost aplikaci zakoupit, kdy tuto obrazovku není možné po určitý, postupně se zvyšující časový interval, zavřít. Cena aplikace je 30$ a v této ceně jsou zahrnuty i veškeré budoucí aktualizace.

K dispozici je rozsáhlá nápověda a diskuzní fórum v angličtině.

GSAK je k dispozici ke stažení na jeho oficiálních internetových stránkách www.gsak.net.

3.1.1 Základní práce s daty

Základním pohledem na data je seznam *keší* zobrazených ve formě tabulky v hlavním okně aplikace.

Seznam *keší* neobsahuje zobrazení *additional waypoints* a k této informaci je možné se pouze složitě proklikat přes úpravu vlastností keše. Ke *keši* je možno uložit tzv. *corrected coordinates*, které poté upraví výchozí souřadnice dané *keše*, podobně, jako to v současnosti umožňují oficiální internetové stránky geocaching.com. Tato úprava je vhodná zejména pro majitele navigačních přístrojů, které nemají konkrétní podporu pro Geocaching. V takovém případě je zbytečné mít v navigaci dva různé body – úvodní souřadnice a vyluštěné finální souřadnice, stačí pouze finální souřadnice. Opět ale, stejně jako v případě oficiálních stránek, lze tímto způsobem uložit pouze jedny souřadnice a na další je třeba použít *additional waypoints*.

3.1.2 Vyhledávání a filtrování

GSAK podporuje více různých databází. Vždy je však možné pracovat pouze s jednou aktuálně otevřenou databází.

Na základní rozdělení *keší* je tedy možné použít různé oddělené databáze. Dále je možné *keše* filtrovat podle všech možných kritérií s poměrně širokými možnostmi nastavení. Bohužel dialogové okno pro filtrování je poměrně nepřehledné, takže je složité se v něm vyznat. Je to ale zajisté otázka zvyku, nicméně začátečníky to může odradit.

Filtry je možné ukládat a později se k nim vracet. Není však možné žádné zřetězení filtrů.

Standardně se do okna aplikace načítají všechny *keše* uložené v databázi, což může u velkého množství dat trvat delší dobu.

3.1.3 Podpora pro ukládání vlastních bodů

Ke každé *keši* je možno uložit libovolný počet dalších bodů, které jsou pak exportovány spolu s *keší*, pokud to daný exportní formát podporuje. Kromě toho je možné u každé keše uložit *corrected coordinates*, což změní výchozí souřadnice keše při exportu.

3.1.4 Mapy

GSAK standardně obsahuje integrované google mapy pro snadný náhled jedné konkrétní vybrané *keše*. Není však možnost zobrazit na mapě více *keší* pro snadný náhled na určitou oblast.

3.1.5 Možnosti rozšíření

GSAK podporuje poměrně rozsáhlý systém maker využívajících vlastního skriptovacího jazyka. Pomocí maker je možno pracovat s *kešemi*, upravovat jejich vlastnosti a *atributy*, vytvářet a upravovat seznamy *keší*, je možno přistupovat na webové stránky, nebo jen provádět nějaké výpočty nebo řešit šifry. Makra mohou obsahovat vlastní formuláře pro přívětivou interakci s uživatelem.

Pomocí maker je možné provádět dodatečné filtrování seznamu *keší* nebo exportovat data v jiném než standardně podporovaném formátu.

Na oficiálních stránkách aplikace existuje široká databáze existujících maker, která je možno do GSAKu přímo nainstalovat.

3.1.6 Naplnění databáze – import dat

Data je možno importovat ze souborů GPX, které je možné stáhnout na oficiálních internetových stránkích www.geocaching.com pro uživatele, kteří mají zaplaceno *premium membership* [11]. Dále v nejnovější verzi GSAK poskytuje podporu pro přístup přes *GC.Live API* [12], kdy je možno stáhnout data i pro uživatele bez *premium membership*. Tím však možnost importu dat do aplikace končí.

3.1.7 Export dat

Data je možno přímo nahrávat do kompatibilního GPS zařízení (podporovány jsou navigace Garmin), nebo použít některý z široké nabídky exportních formátů. K dispozici jsou formáty pro všemožné automobilové navigace, formáty pro nahrání offline HTML seznamů do PDA a další specializované formáty.

Nevýhodou je, že formát výstupních dat nelze upravovat, jediným řešením je napsat si vlastní makro, které bude řešit kompletní export.

3.1.8 Implementace a uložení dat

GSAK od verze 7 (aktuální – květen 2012 – je verze 8) používá SQLite databázi. V této databázi jsou uloženy veškeré údaje o keších, jejich bodech a uživatelských datech. GSAK také podporuje práci s více různými databázemi, avšak lze pracovat vždy pouze s jednou databází současně.

Program je implementován v prostředí Borland Delphi, z čehož plynou některá omezení, zejména podpora dalších platforem nebo archaicky vypadající uživatelské rozhraní.

3.2 GeoGet

GeoGet je mladší než GSAK, jedná se o český výtvor a je nejvíce rozšířen právě v České Republice. Existuje pro něj české diskuzní fórum [13], kde je uživatelům poskytována podpora a právě česká komunita okolo GeoGetu je velice početná.

Aplikace je dostupná pouze pro operační systém MS Windows a s určitými omezeními je možné ji provozovat i v emulovaném prostředí Wine v Linuxu. Pod Wine aplikace ale není podporována a toto použití je určeno jen pro zdatné uživatele, kteří se nebojí experimentovat.

GeoGet je k dispozici zdarma, autora je možno odměnit dobrovolným příspěvkem skrz PayPal.

GeoGet je k dispozici ke stažení na jeho oficiálních internetových stránkách geoget.ararat.cz.

3.2.1 Základní práce s daty

Základem je, stejně jako u GSAKu, okno se seznamem právě načtených *keší*. V případě GeoGetu se jedná o seznam *keší*, které odpovídají právě aktivnímu filtru. U každé *keše* jsou zároveň viditelné i všechny její *additional waypoints*, avšak je možné je hromadně skrýt a u některé z *keší* poté jedním kliknutím zobrazit. Seznam *keší* je různě barevně podbarven podle stavu *keše*. Běžné *keše* jsou podbarveny bíle, nalezené zeleně. *Keše*, které obsahují nějaké *additional waypoints* jsou podbarveny žlutě a *keše*, které obsahují uživatelem vložený *waypoint* jsou podbarveny tmavší žlutou barvou. *Archivované keše* jsou poté podbarveny červeně. Toto barevné rozlišení značně pomáhá v rychlé orientaci v seznamu.

Je možné uložit standardní podobu filtru, která se použije při prvotním načtení *keší* po spuštění aplikace, takže se nikdy nemusí načítat obsah celé databáze.

Ke každé *keši* je možno přiřadit libovolné množství tzv. *tagů* – což je kombinace název/hodnota. Pomocí těchto *tagů* je pak možné keše filtrovat nebo vyhledávat, případně skripty je mohou používat k ukládání nějakých svých informací.

Dále keš může obsahovat přílohy – libovolné soubory, které jsou svázány s *keší* a vždy je možno je snadno zobrazit.

GeoGet nemá podporu pro *corrected coordinates*, ale tento nedostatek je řešen v exportních skriptech, které pro majitele navigací vyexportují soubor stejně, jako by šlo o *corrected coordinates*.

3.2.2 Vyhledávání a filtrování

GeoGet poskytuje široké možnosti filtrování. Filtrovat je možno podle všech myslitelných kritérií. Kromě základních možností vyplývajících z vlastností *keše* je to zejména vyhledávání okolo dané trasy nebo uvnitř určitého polygonu.

Nastavení filtrování je možno ukládat a následně tyto filtry opětovně načítat. Filtry je možné kombinovat pomocí operací sčítání nebo odečítání výsledků filtru od aktuálně nahraného seznamu

keší. Zřetězení filtrů ale nelze ukládat, takže posloupnost filtrovacích kroků je vždy nutno provést znova.

Pro nejzákladnější vyhledávání existují i speciální možnosti rychlého hledání – hledat jde podle GC kódu, autora keše a názvu keše.

GeoGet také monitoruje systémovou schránku (clipboard) a jakmile se v ní objeví odkaz na *keš* na oficiálních internetových stránkách, automaticky ji zobrazí v okně se seznamem *keší*, pokud tato *keš* existuje v databázi.

3.2.3 Podpora pro ukládání vlastních waypointů

Jak již bylo naznačeno, ke každé *keši* si uživatel může uložit vlastní *waypointy*. Může si tak uchovávat databázi finálních souřadnic a případně i všech postupových zastávek. Při exportování dat jsou pak tyto body přidány stejně jako autorem *keše* zadané *additional waypoints*.

Ke každému bodu je možno uložit veškeré informace, které je možné připojit k bodu u vlastní *keše* na oficiálních stránkách (tedy název, popisek, souřadnice, prefix a lookup kód) a také některé další – zejména vlastní uživatelskou poznámku.

GeoGet také podporuje výpočet souřadnic bodu pomocí projektování – tedy určení vzdálenosti a směru.

Okno s úpravou/přidáním nového bodu si pamatuje uložené hodnoty, takže je snadno možné přiřadit podobný bod více keším po sobě.

Nevýhodou je, že po vyplnění políčka se souřadnicemi je nutné tyto souřadnice ručně uložit, zatímco ostatní údaje se ukládají automaticky při zavření tohoto okna.

3.2.4 Mapy

GeoGet sám o sobě nepodporuje zobrazení žádné mapy přímo v aplikaci. Pokud si uživatel zobrazí listing *keše* v aplikaci, obsahuje statické obrázky mapy umístění podle souřadnic samotné *keše* a všech *waypointů*.

GeoGet také obsahuje HTML implementaci mapy, kterou je možno přímo z prostředí GeoGetu otevřít ve výchozím internetovém prohlížeči. Jedná se však o mapu se staticky generovaným seznamem bodů (jak *keší* tak *waypointů*) a není možná žádná interakce s prostředím GeoGetu. Mapa navíc není napsána kvalitně, takže jediným podporovaným prohlížečem je Mozilla Firefox.

Podpora map je velkou slabinou GeoGetu vzhledem k tomu, že pohled do mapy je jeden z nejpřehlednějších způsobů vyhledávání *keší*.

3.2.5 Možnosti rozšíření

GeoGet podporuje psaní rozšíření formou uživatelských skriptů [14]. Tyto skripty jsou psány ve speciálním dialektu programovacího jazyka Pascal a jejich možnosti jsou značně omezené. Prakticky

není možné nijak interagovat s prostředím GeoGetu, kromě manipulace s aktuálně načteným seznamem keší. Složitější skripty tedy fungují tak, že jsou vlastně pouze spouštěčem nějaké další aplikace, která provede požadovanou složitější funkčnost a vrátí výsledek například v podobě vygenerovaného souboru.

Skripty v GeoGetu se dělí do několika kategorií. Jsou to skripty umožňující obecně libovolnou činnost, dále to jsou exportní skripty, které generují určitý výstupní formát dat podle aktuálně vyfiltrovaných bodů v hlavním okně, vizualizační skripty, které umožňují do seznamu keší vykreslovat grafická data (například atributy, hvězdičky hodnocení terénu a obtížnosti a další) a nebo knihovny, které poskytují další funkce ostatním skriptům.

Důležité jsou zejména zmíněné exportní skripty – pomocí nich je možné definovat širokou škálu formátů, ve kterých je možné data z databáze získat pro použití v celé řadě navigačních aplikací a speciálních navigací. Existují exportní skripty, které exportují data ve stejném formátu jako se dají stáhnou z webu geocaching.com (Pocket Query GPX), existují exportní skripty pro autonavigace TomTom a iGo, speciální exportní skripty generující GPX soubory speciálně určené pro navigace Garmin, a celá řada dalších [15].

Existuje poměrně široká řada všemožných rozšíření, která je dána širokou uživatelskou základnou okolo této aplikace.

Skripty mohou obsahovat formuláře pro snadnou interakci s uživatelem.

3.2.6 Naplnění databáze – import dat

Databázi je možno naplnit pomocí GPX souborů, které jsou stažitelné z oficiálních stránek geocaching.com. Dále existuje podpora pro přístup k oficiálním stránkám pomocí nově zveřejněného GC.Live API, kdy je tato možnost dostupná i pro majitele, kteří nemají zaplacený *premium membership*. Existuje pak také neoficiální aplikace GeoJarry, kterou vyvíjí neznámý autor a která umožňuje data získávat s pomocí parsování HTML stránek geocaching.com. Toto použití ale porušuje podmínky použití webu geocaching.com, se kterými každý uživatel souhlasí při registraci, takže pokud se někdo uchýlí k použití této aplikace, musí počítat s tím, že je tu možnost, že z toho budou vyvozeny určité následky.

3.2.7 Export dat

Data je možno exportovat do libovolných formátů. Všechny exporty jsou řízeny skripty, takže je možné každý z formátů upravit v případě, že uživateli nevyhovuje. Není třeba kvůli tomu psát celý skript od začátku.

3.2.8 Implementace a uložení dat

GeoGet používá k ukládání dat databázi SQLite. Práce s ní je v některých případech značně pomalá. Nejnáročnější je práce s tagy, kde je zbytečně využito číselníků a referencí do těchto číselníků, a to jak pro názvy tagů, tak pro jejich hodnoty. Každé zobrazení tagů tedy znamená dotaz do tří databázových tabulek, zatímco by stačil dotaz pouze do jedné jediné. Zápis tagů pak trvá také dlouho, protože je nutné vyřešit unikátnost záznamů a správně nastavit kombinaci název/hodnota pro konkrétní upravovanou *keš*.

Delší textové údaje (short description, long description a hint) jsou v databázi uloženy ve speciální tabulce mimo základní informace o keších, a jsou komprimovány pomocí gzip komprese.

GeoGet podporuje práci s více databázemi, avšak lze pracovat vždy jen s jednou databází současně. Mezi databázemi není možné body přesouvat nebo kopírovat, je nutné tento nedostatek obejít s použitím exportu a následně importu do druhé databáze.

3.3 Další nástroje

Existují i některé další pokusy o nástroje, které však nejsou zdaleka tak rozšířené jako GeoGet a GSAK a jejich vývoj není nikterak intenzivní. Vybírám několik aplikací, kdy se zaměřuji na platformy Windows a Linux (případně web jako univerzální prostředek na obou platformách), protože to jsou cílové platformy kam směřuje moje aplikace. Neřeším tedy například aplikace pro mobilní přístroje, protože to je zcela jiný segment, pro který je třeba navrhnout aplikaci se zcela jinými požadavky než jsou kladeny na aplikaci pro stolní počítač.

3.3.1 Open Cache Manager

Open Cache Manager je jediná existující aplikace, která je k dispozici i pro Linux. Podporuje pouze základní funkcionalitu. Má velice uživatelsky přívětivé uživatelské rozhraní, které lze použít jako inspiraci.

3.3.2 „Klikátko“

Jedná se o online aplikaci dostupnou pouze pro registrované uživatele na adrese http://gc.zlej.net/. Podporuje základní práci s kešemi a pouze základní možnosti filtrování. Zásadní nevýhodou této aplikace je, že je k dispozici pouze online. Není s ní tedy možno pracovat, pokud není k dispozici připojení k internetu.

3.3.3 MoZiGo

Tento program by měl umožňovat základní správu keší, bohužel jeho uživatelské rozhraní je natolik nepřehledné, zmatené a složité, že nebyl podroben hloubějšímu zkoumání. Jeho jediná známá nevýhoda je tedy v uživatelském prostředí.

4 Požadavky na aplikaci

4.1 Motivace

V předchozích kapitolách jsem se snažil nastínit možnosti správy většího počtu *keší* jednak s pomocí standardních nástrojů na oficiálních stránkách www.geocaching.com tak pomocí specializovaných programů na to určených. Snažil jsem se, aby vyplynulo, že pokud uživatel chce mít věci pod kontrolou, nějaký z takových programů potřebuje, aby se v *keších* vyznal. Obecně uživatelé, kteří potřebují nějak pracovat s offline verzemi *keší* provádějí alespoň jednu, ale často všechny, následující činnosti:

- Správa seznamů *keší* – jejich procházení, plánování, zobrazení na mapě a následné nahrání do GPS.
- Uchovávání finálních souřadnic *keší* nebo postupových souřadnic pro pozdější využití
- Příprava dat pro offline použití v telefonu/PDA/čtečce elektronických knih
- Uchovávání postupů řešení
- Generování statistik nalezených a/nebo založených *keší*.

Z tohoto seznamu vyplývají základní činnosti, které by taková aplikace měla umět. V této kapitole se pokusím sepsat seznam požadavků na takovou aplikaci a vybrat ty, které by aplikace měla do základu obsahovat a ty, které jsou určeny pro pozdější rozšíření aplikace.

4.2 Základní požadavky

- Multiplatformnost – podpora alespoň systému MS Windows a Linux.
- Zobrazení seznamu keší odpovídajících zvoleným filtrovacím kritériím ve formě tabulky.
- Možnost filtrovat seznam keší podle všech možných kritéritií.
- Možnost zřetězit filtry matematickými operacemi (alespoň sjednocení, průnik, rozdíl) a toto zřetězení uložit.
- Do budoucna zobrazení *keší* na mapě – všech *keší* z databáze, *keší* odpovídajících právě aktivnímu filtru, jedné konkrétní *keše* – spolu s (volitelně) všemi nebo jen některými *waypointy*.

- Podpora pro ukládání vlastních *waypointů* a *corrected coordinates*.
- Podpora pro ukládání vlastních poznámek ke *keši* a dalších souborů s řešením.
- Vstup dat z formátu PocketQuery GPX
- Do budoucna spolupráce s GC.Live API (problematická komunikace s firmou Groundspeak, proto s podporou API není počítáno jako s primárním zdrojem dat, pouze jako možnost o tuto funkcionalitu aplikaci později rozšířit, až se s firmou Groundspeak povede vyjednat přístup).
- Export dat v nejpoužívanějších formátech (primárně PocketQuery GPX)
- Samozřejmostí je příjemné uživatelské rozhraní, které neodradí ani začátečníky, ale pokročilí uživatelé snadno naleznou všechny pokročilé funkce.

4.3 Možnosti rozšíření

Aplikace by měla podporovat rozšíření funkcionality pomocí nějakého systému pluginů. Bude navržen systém umožňující naprostou flexibilitu a možnost i tento systém do budoucna rozšířit o podporu skriptovacích jazyků.

4.4 Spolupráce s dalšími nástroji

Do budoucna by měla být zajištěna spolupráce s dalšími nástroji – zejména s GeoGetem a GSAKem alespoň ve formě obousměrné synchronizace dat pomocí některého z podporovaných formátů. Aplikace by mohla umět pracovat přímo s databázemi GeoGetu a GSAKu, aby uživatelé mohli pokud možno používat oba dva nástroje, než se rozhodnou buďto že GeoGet / GSAK opustí nebo že u něj naopak zůstanou.

5 Návrh aplikace

V této kapitole se budu věnovat samotnému návrhu aplikace. Úkolem je navrhnout aplikaci, která bude splňovat požadované vlastnosti z kapitoly 4 a zároveň bude možno ji do budoucna snadno rozšiřovat o další funkcionalitu.

5.1 Výběr prostředí

Z důvodu požadavku na přenositelnost minimálně mezi platformami MS Windows a Linux je nutné sáhnout po nějakém prostředí, které podporuje alespoň tyto dvě platformy. Nabízelo by se využít prostředí Java, ale protože Javu neumím a nechci se jí učit, nebude použita.

Sáhnu tedy raději k osvědčenému prostředí C++, kde bude se zajištěním přenositelnosti aplikace trochu víc práce, ale alespoň nebude problém s výkonem a paměťovou náročností jako by

byl v případě Javy. Co se grafického uživatelského rozhraní týče, rozhodl jsem se použít knihovnu Qt, se kterou mám pozitivní zkušenosti co se týče přenositelnosti a čistoty jejího návrhu. Qt je navrženo kvalitně objektově a dobře se v něm vyvíjí aplikace. Qt ale bude použito jen pro grafické uživatelské rozhraní, ostatní části systému na něm musí být nezávislé, aby bylo možno případně do budoucna použít i prostředí napsané pro jiný toolkit, například GTK a aplikace tak nenesla hluboké závislosti na Qt.

Myslím také zároveň na serverové využití aplikace, kdy bude k dispozici pouze rozhraní příkazové řádky, kdy je závislost na Qt také na škodu, protože není důvod na serveru mít Qt vůbec nainstalované.

Do budoucna je možná také přenositelnost na mobilní platformy, kde bude také potřeba použít speciální uživatelské rozhraní a aplikace nemůže nést hromadu závislostí na knihovny, které pro danou platformu třeba vůbec neexistují.

5.2 Základní koncept rozšiřitelnosti

Pod pojmem Základní koncept rozšiřitelnosti myslím samotné základní jádro aplikace, které se bude starat jen o spolupráci mezi pluginy a nebude samo obsahovat žádnou pokročilou funkcionalitu. Odsunutím veškeré funkcionality do pluginů je zajištěna snadná vyměnitelnost a nahraditelnost jednotlivých komponent a velká rozšiřitelnost celého prostředí. V této podkapitole popíši, jak bude celý systém fungovat.

5.2.1 Základní koncept jádra

Jádro samotné se bude starat o načtení pluginů do paměti. Každý plugin po své inicializaci zaregistruje, jakou funkcionalitu poskytuje spolu s ukazatelem na funkci, která má být k obsluze této funkcionality použita. Tím je zajištěna naprostá flexibilita toho, co pluginy mohou umožňovat, protože společné komunikační rozhraní bude dáno pouze názvem funkce a ukazatelem na ni. Všechny ostatní potřebné informace, jako například počet a typ parametrů už budou specifické pro konkrétní funkci a jádro aplikace je vůbec nebude řešit.

Jádro vůbec nebude tušit, které funkce jsou pomocí pluginů implementovány. Plugin tedy může vyžadovat implementaci funkce, která v době psaní jádra vůbec nebyla známa. Tuto funkci pak může poskytovat libovolné množství pluginů. K výběru konkrétní funkce, která se pro jeden konkrétní případ použije, slouží tzv. Selector.

Pojem funkce je v tomto případě poměrně vzdálen běžnému označení funkce z programovacích jazyků. Jedná se tedy spíše o „funkcionalitu“, například „uživatelské rozhraní“ nebo „konfigurace“ nebo „databáze“ jsou typickými představiteli takových funkcí, které plugin může zajišťovat.

5.2.2 Selector

Jak již bylo zmíněno v předchozí podkapitole, pokud je k dispozici více pluginů zajišťujících konkrétní funkcionalitu, bude uživateli zobrazen selector, kde může zvolit, který z pluginů chce použít pro konkrétní volání určité funkce. Selector bude reprezentován abstraktním rozhraním, jehož konkrétní implementaci dodá každé uživatelské rozhraní. Grafické uživatelské rozhraní dodá vhodnou implementaci grafického Selectoru, zatímco textové rozhraní dodá jeho textovou implementaci. I tuto část jádra tedy bude možné v budoucnu rozšířit nebo vyměnit, aniž by se muselo zasahovat do základní aplikace.

5.3 Uživatelské rozhraní

Z požadavků na aplikaci vyplývá, že by měla být dostupná pro široké množství uživatelů. To implikuje vybavení aplikace grafickým uživatelským rozhraním. Z důvodu snadného použití pro automatizaci činností však aplikace bude obsahovat i textové rozhraní ovládané z příkazové řárky.

Uživatelské rozhraní bude pro větší univerzálnost provedeno v anglickém jazyce s možností lokalizace do dalších jazyků v budoucnosti.

5.3.1 GUI

Grafické uživatelské rozhraní bude využívat knihovny Qt a jejích ovládacích prvků. Jedná se o základní uživatelské rozhraní, které bude uživatel primárně používat. Základním prvkem na obrazovce bude okno se seznamem keší, které bude obklopeno nástrojovými lištami obsahujícími přístup k dalším funkcím. V uživatelském rozhraní musí být možnost provést všechny akce, které jsou v aplikaci požadovány. Seznam *keší* musí fungovat automaticky, aby ihned po spuštění programu obsahoval data podle aktuálně zvoleného filtru. Další tlačítko typu „načíst seznam podle filtru“ je zbytečné a jen znepřehledňuje používání programu. Stejně tak při jakékoliv změně (například filtru) musí dojít k automatické aktualizaci seznamu aby vždy obsahoval aktuální data.

5.3.1.1 Práce se seznamem keší

V načteném seznamu keší bude možnost provádět alespoň následující operace:

- Prohlížet detaily keše
- Úprava keše
- Přidání waypointu
- Úprava existujícího waypointu
- Smazání existujícího waypointu
- Odstranění keše z výběru
- Otevřít listing keše na webu gc.com (ve výchozím internetovém prohlížeči)

5.3.2 Příkazový řádek

Textové rozhraní bude rozhraní poskytující práci převážně využitelnou ve skriptech – tedy automatizace běžných činností jako je aktualizace databáze nebo export dat. Bude ovládáno parametry příkazového řádku a v základu musí umět alespoň import dat z podporovaných formátů a jejich export včetně specifikace filtru, který se má použit pro vytřídění seznamu bodů, které mají být exportovány.

5.4 Uložení dat

Z hlediska snadné přenostitelnosti a možné rozšiřitelnosti bude výhodné sáhnout k osvědčené SQLite databázi, kterou využívají i konkurenční produkty. Nabízelo by se využít schéma databáze kompatibilní s některou ze zmiňovaných aplikací, ale to by vedlo k nutnosti nacházet kompromisy pokud by databáze některou z požadovaných vlastností neměla kam uložit nebo se její schéma ukázalo jako nevhodné. Proto bude lepší navrhnout schéma databáze vlastní s případnou možností pracovat s databázemi ostatních aplikací jako volitelný doplněk.

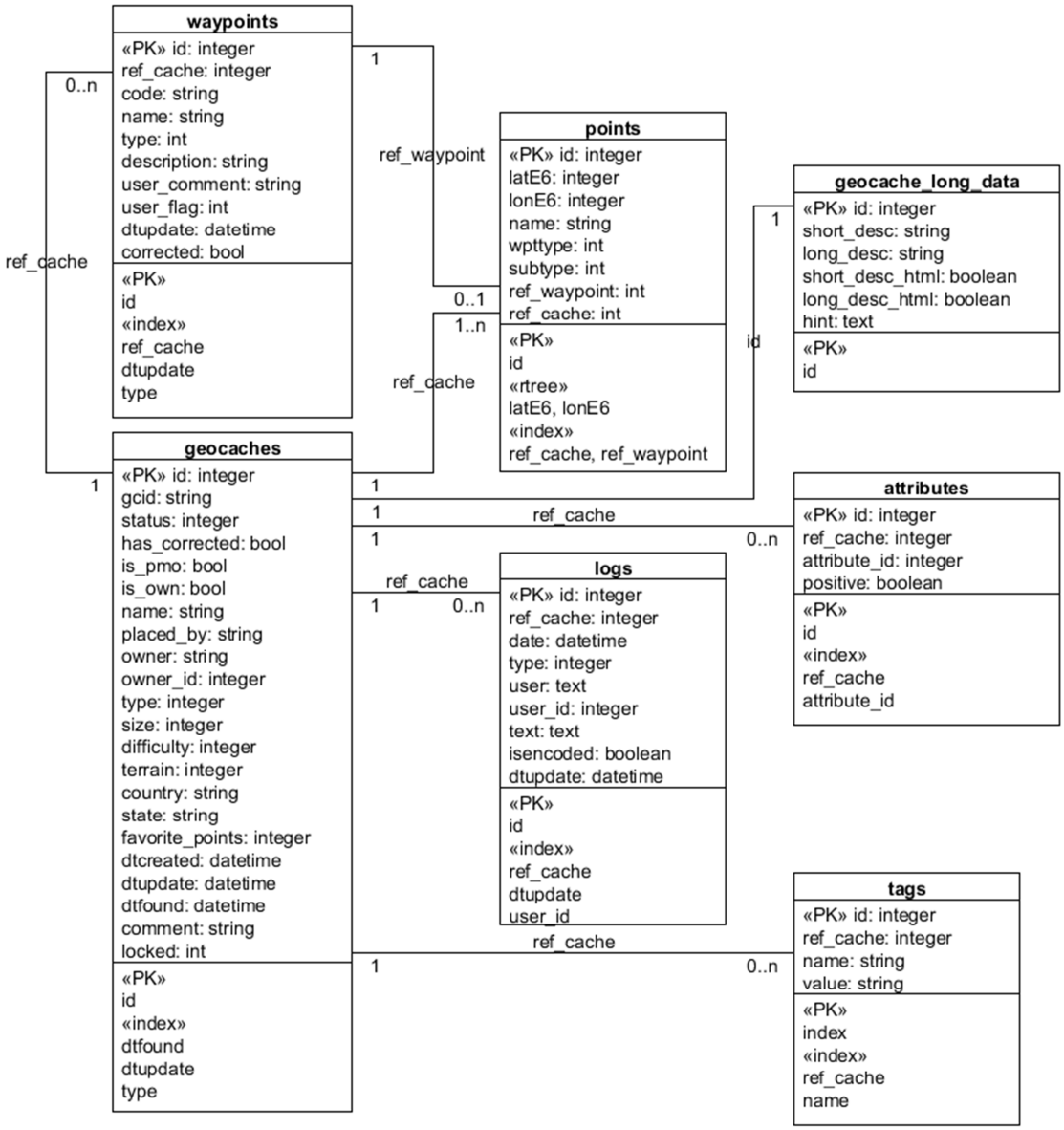
Uložení základních dat je zřejmé – tabulka pro *keše*, *waypointy*, *logy*. Dále bude nutné zavést podporu *tagů* podobně, jako to má GeoGet. Pro tagy však bude stačit pouze jedna tabulka – jednoduchá dvojice klíč-hodnota s referencí na keš, které se daný *tag* týká. Každý *tag* bude moci jako svoji hodnotu obsahovat libovolná data. *Waypointy* budou mít příznak, zdali jsou uživatelsky zadané, nebo je jejich zdrojem import dat. Uživatelsky zadané *waypointy* nebudou přepsány *waypointem* se stejným prefixem při importu. Nastavení uživatelského příznaku by mělo jít změnit v uživatelském rozhraní při editaci *waypointu*.

U každé *keše* bude informace o tom, zdali jsou její data uzamčena. Uzamknutí bude bitové pole, kdy jednotlivé nastavené bity budou znamenat uzamčení dílčích informací o *keši*. Uzamykat půjde buďto celá keš nebo jen některé skupiny údajů – název + autor, velikost + obtížnost + terén, souřadnice, text *listingu* (short description, long description, hint), *waypointy*.

U každé položky bude také datum její poslední aktualizace, které bude nastaveno na aktuální hodnotu vždy v momentě, kdy dojde ke změně. Bude tak do budoucna možné připravit podporu pro synchronizaci databáze mezi více místy, případně zavést podporu pro distribuovanou databázi uloženou na serveru.

5.4.1 Schéma databáze

Na obrázku Obrázek 1 - Schéma databáze je zobrazeno schéma databáze pro počáteční verzi programu.



Obrázek 1 - Schéma databáze

5.4.1.1 Poznámky ke schématu databáze

Databáze je navržena pro co nejrychlejší přístup k datům. Zejména pro budoucí implementaci mapových podkladů je pamatováno na snadný přístup k základním údajům využitelným v mapě. Z toho vyplývá zavedení tabulky *points*, která sdružuje veškeré zeměpisné údaje o každém z bodů spolu se základními metadaty. Jedním dotazem na databázi je tak možno získat vše potřebné, co by mělo být zobrazeno na mapě včetně základního filtrování a není tak potřeba složitě spojovat výsledky z několika dotazů do jednoho.

Dále je oddělen text *listingu* od *keší* do zvláštní tabulky, zejména z důvodu rychlého přístupu k základním datům s myšlenkou, že text *listingu* není třeba vždy, ale až na vyžádání uživatele (například při zobrazení detailů). To umožňuje rychlé hledání v tabulce, kdy při získávání dat není potřeba přeskakovat velké bloky textu a tyto jsou uloženy na jiném místě.

Tabulka s *kešemi* obsahuje některé redundantní informace (například příznak *is_own*, pokud se jedná o vlastní *keš* - není tak třeba pokaždé porovnávat majitele *keše* s všemi možnými definovanými jmény autorů, pod kterými uživatel *keš* zakládá), což je opět z důvodu zajištění rychlejšího vyhledávání pomocí jednoduchého dotazu na hodnotu sloupce typu integer.

Samotná tabulka *points* je poměrně komplikovaná z hlediska správné aktualizace dat. Souřadnice jsou uloženy v podobě celého čísla typu integer, což zajišťuje při udržení dostatečné přesnosti mnohem rychlejší operace než při práci s čísly v plovoucí desetinné čárce. Souřadnice jsou v této tabulce uloženy tak, že jejich desetinná hodnota je vynásobena 10^6, což zajistí uložení šesti desetinných míst souřadnic, což je hodnota dostatečná pro přesnost, kterou GPS nabízí. K tomuto formátu mi bylo inspirací API Google Maps z operačního systému Android [16], kde se souřadnice v tomto formátu používají napříč veškerými funkcemi.

Kromě zeměpisných souřadnic jsou v této tabulce uloženy i další údaje – zejména se jedná o název bodu a jeho typ a dále odkaz na konkrétní *keš* a v případě souřadnic *waypointu* i *waypoint*, ke kterému souřadnice patří.

Typ bodu je rozdělen na dvě položky – jednou je základní typ – sloupeček *wpttype*, který obsahuje informaci, zdali se jedná o souřadnice *keše*, korigované souřadnice *keše* nebo souřadnice *waypointu*. Konkrétně v případě keše může hodnota *wpttype* nabývat tří různých hodnot – WPT_GEOCACHE, WPT_GEOCACHE_INITIAL, WPT_GEOCACHE_CORRECTED.

První z hodnot se použije v případě, že ke keši jsou známy jen jedny souřadnice. Druhé dvě hodnoty se používají v momentě, kdy ke *keši* existují korigované souřadnice – *initial* jsou pak označeny původní souřadnice, které zadal autor a *corrected* - korigované zadané uživatelem. Výhoda oddělení „jen původních“ a „původních + korigovaných“ souřadnic pomocí rozlišení typu je z důvodu snazšího zobrazování keší na mapě. Mohou pak být snadno zobrazeny všechny keše, ke kterým korigované souřadnice nejsou zadány spolu s kešemi, ke kterým zadány jsou, aniž by se na mapě zároveň zobrazovaly i jejich původní souřadnice.

Hodnotou ve sloupečku *subtype* je konkrétní typ daného bodu – tedy typ *keše* nebo typ *waypointu*. Do budoucna dojde pravděpodobně ještě k rozšíření tohoto typu o bitovou masku obsahující příznak zdali se jedná o vlastní nebo nalezenou keš.

5.5 Import dat

Import dat bude probíhat ze souborů PocketQuery GPX. Je nutno pamatovat na to, že v současnosti existují tři možné formáty PocketQuery GPX – verze 1.0, 1.0.1 a 1.0.2. První verze je původní a obsahuje všechny důležité informace. Ve verzi 1.0.1 jsou navíc přidány atributy keší, avšak je zachována zpětná kompatibilita.

Verze 1.0.2 prozatím nebyla oficiálně zveřejněna a existuje pouze její návrh, není však prozatím (v době psaní této práce) možné keše v tomto formátu ze serveru geocaching.com získat.

Tato verze není zpětně kompatibilní, zavádí některé změny, na které je třeba v kódu pamatovat. Jedná se zejména o záměnu veškerých číselných identifikátorů za jejich GCID varianty (viz kapitola 2.4.1.1). Dále je zavedena podpora pro korigované souřadnice, kdy však v případě výskytu korigovaných souřadnic už nejsou k dispozici ty originální. Kromě těchto zásadních změn jsou doplněny i informace o počtu *favorite* bodů a je přidán seznam fotek, které jsou součástí *listingu*.

Volitelně do budoucna musí být počítáno i s dalšími možnostmi importu dat, zejména importu ZIP archivů a vstupu dat z GC.Live API, pokud se povede získat k němu přístup.

5.6 Filtrování

Filtrování dat bude možné podle všech kritérií, která jsou uložena v databázi. Jmenovitě tedy jde o filtrování podle názvu, typu, data založení, autora, dostupnosti, velikosti, terénu, obtížnosti, státu, kraje, data změny, data založení, data nálezu, počtu bodů oblíbenosti (avšak poslední jmenované až v době, kdy bude tuto informaci možno získat).

Dále podle přítomnosti final waypointu případně jakéhokoliv uživatelského waypointu.

Filtrování podle polohy bude v základu uvažováno jako „okruh o daném poloměru od zadaného bodu“ a do budoucna dojde k rozšíření o filtrování v rámci konkrétního polygonu nebo podél specifikované trasy.

Filtry bude možno řadit do složených filtrů, kde složený filtr bude posloupnost základních filtrů a matematická operace, která se má s výsledkem filtrování provést. Vše bude možné uložit jako definici filtru v XML souboru a následně opět načíst.

5.7 Export dat

Standardně bude podporován export dat ve formátu PocketQuery GPX tak, aby byla zajištěna kompatibilita s ostatními nástroji. Volitelně pomocí dalších *pluginů* je možné přidat libovolný další exportní formát.

5.8 Mapa

Mapa bude v budoucnu implementována jako vlastní *widget*, starající se o kreslení mapových podkladů, jejichž zdrojem může být offline databáze nebo i online webová služba. Každý mapový zdroj bude opět poskytován pomocí *pluginu*. Na mapě budou zobrazeny keše odpovídající aktuálně načtenému filtru s přihlédnutím na zobrazený region mapy. Bude možné zobrazit kolizní okruhy o poloměru 161m kolem všech relevantních bodů (fyzických *keší*, finálních *waypointů*, *waypointů* typu *stage of a multicache*). Dále bude možno na mapě plánovat trasu jejím přímým kreslením, což bude zároveň použitelné i pro měření vzdálenosti.

Načítání *keší* do mapy bude fungovat automaticky, aby uživatel nemusel klikat na tlačítko „načti keše“. Stejně tak při jakékoliv změně (například filtru) se změna automaticky promítne i do mapy.

6 Implementace

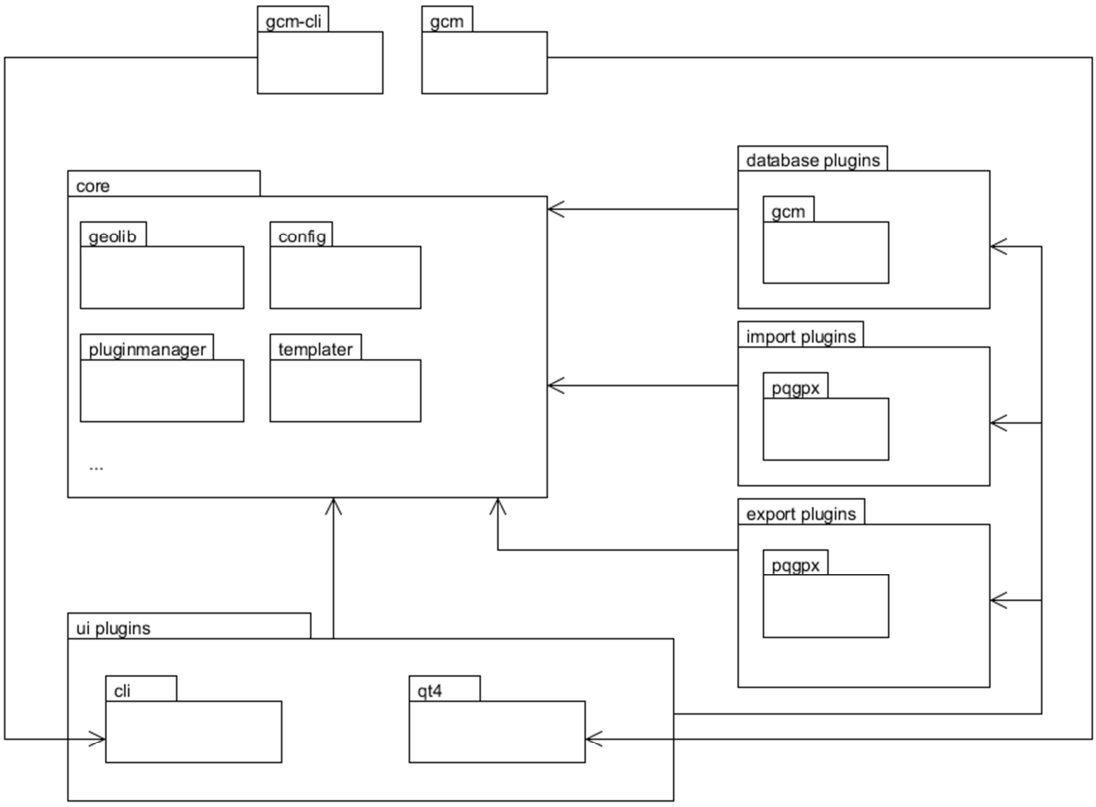
V této kapitole popisuji implementaci samotné aplikace GCManager (zkráceně GCM) a problémy, které bylo nutno během implementace řešit.

Celý zdrojový kód se dá rozdělit na dvě části – jádro a jednotlivé pluginy. Jádro je pak pro přehlednost rozděleno ještě do jmenných prostorů, které odpovídají adresářové struktuře dělení zdrojových textů. Celé jádro je součástí velkého jmenného prostoru *GCM* a dílčí části jsou pak součástí dalších zanořených prostorů. Nevýhodou tohoto přístupu je občas složité pojmenování třídy, když není možno použít direktivu *using namespace* – například *GCM::GC<GCM::geolib::Geocache>* není zrovna nejkratší název třídy, ale alespoň je jasné, kam co patří.

V dalších podkapitolách se budu věnovat již konkrétním částem implementace.

6.1 Architektura systému

Architektura systému, tak, jak je v současné době implementována, vychází z návrhu. Existuje jedno jádro, tvořené knihovnou libgcm.(so|dll) a *pluginů*, které zajišťují doposud implementovanou funkcionalitu odpovídající požadavkům kladeným na aplikaci v návrhu. Nejlépe je architektura vidět na přiložené ilustraci na obrázku Obrázek 2.



Obrázek 2 - Architektura systému

Obrázek zachycuje všechny aktuálně implementované pluginy a znázorňuje závislosti jednotlivých celků na sobě – ze závislostí je vidět, že například plugin uživatelského rozhraní nezávisí na jednom konkrétním exportním pluginu, ale závisí pouze na tom, že existují pluginy schopné poskytnout export dat. Stejně tak s importem i databází.

Jediné přímé závislosti jsou co se týče binárních souborů, které vyžadují jeden přítomnost CLI pluginu a druhý přítomnost Qt4 pluginu, protože přímo určují, které uživatelské rozhraní se použije.

6.2 Garbage collector

Už v počátečních fázích implementace se ukázalo, že v některých případech by bylo výhodné, aby se programátor sám nemusel starat o uvolňování paměti. Zejména proto, že záměrem bylo vyhnout se pokud možno kopírování objektů z důvodu co nejnižší paměťové náročnosti aplikace při práci s velkými databázemi.

Ideální stav tedy je, aby každá instance objektu byla v paměti pouze jednou a předával se na ni pouze ukazatel – například každá *keš* jen jednou, protože její kopírování v paměti pro použití v jiných místech aplikace by bylo příliš paměťově i výkonově náročné. Takový objekt může být zároveň používán na několika místech – může být zobrazen v seznamu, může být právě editován, může být zobrazen *waypoint*, který na *keš* odkazuje, může být zobrazen na mapě. Je tedy nevýhodné pracovat

se samostatnými ukazateli, protože by nebylo jasné, kdy konkrétně se objekt má uvolňovat a zavádět kontrolu použití objektu na každém místě by znamenalo velké riziko zanesení chyby vzhledem k nutnosti na každém potenciálním místě tuto kontrolu provádět.

Přistoupil jsem tedy k implementaci jednoduchého *garbage collectoru*.

Garbage collector funguje na principu počítadla referencí a funguje zcela transparentně. Existuje generická třída *GC*, která jako parametr třídy očekává název třídy, kterou má spravovat. Taková třída musí dědit od globální třídy *Object*, kde je uloženo počítadlo referencí.

Objektu, obaleném *garbage collectorem* reprezentovaným zmíněnou generickou třídou *GC* říkejme *reference*.

Počítadlo referencí je alokováno jako instance třídy *RefCount* a v objektu je uložen vždy jen ukazatel na tuto instanci. Do každé *reference* reprezentované jednou instancí třídy *GC* je pak tento ukazatel zkopírován a je inkrementováno počítadlo referencí v tomto objektu. V destruktoru *reference* dojde k dekrementaci čítače referencí a pokud počet referencí dosáhne počtu 0, objekt je z paměti uvolněn.

Z toho vyplývá, že jakmile je objekt jednou spravován *garbage collectorem*, nesmí existovat nikdy mimo prostředí GC, protože jinak by byl uvolněn a takový ukazatel by se stal neplatným.

6.2.1 Cyklické reference

Obecným problémem *garbage collectoru* jsou cyklické reference. Lze si to představit jako situaci, kdy máme objekt (A), který obsahuje referenci na jiný objekt (B), který ovšem obsahuje referenci zpět na původní objekt (A). Počítadlo referencí nikdy nedosáhne hodnoty 0, protože aby se uvolnil objekt B, musí se nejprve uvolnit objekt A, který ale je stále z B instancován, tj. používán.

Tento problém jsem také musel řešit a řešení tohoto problému se jmenuje slabé reference, v mém případě reprezentované generickou třídou *WeakGC*. Tato reference neinkrementuje čítač referencí. Používá se právě pro zpětné reference na objekty, v našem případě tedy pro referenci na objekt A v objektu B. V momentě, kdy dojde k zániku reference na třídu A je tato správně uvolněna, v závislosti na tom dojde zároveň i k uvolnění reference na B uvnitř třídy A, které následně uvolní i slabé reference zpět na A. Při tomto případu je nutno řešit problém, pokud se reference na třídu B dostane ven z třídy A. V takovém případě, v momentě, kdy dojde k uvolnění třídy A už na ni neexistuje žádná silná reference, pouze slabá v třídě B. V tomto případě je nutno na to pamatovat a ošetřit stav, kdy silná reference (a tedy i odkazovaný objekt) již neexistuje. Řešením je například objekt znovu vytvořit v momentě, kdy na něj z třídy B vznikne požadavek.

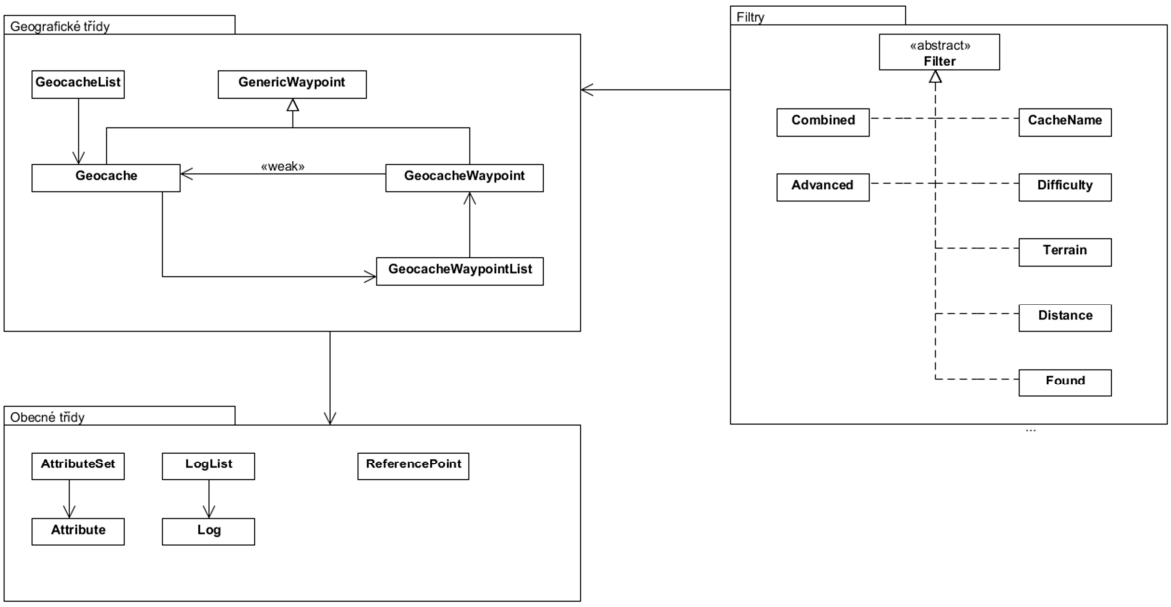
Z důvodu počítání slabých referencí obsahuje třída *RefCount* ještě jedno počítadlo, které se inkrementuje jak v případě silné tak v případě slabé reference. Slouží ke správné správě paměti právě objektu třídy *RefCount*, kdy ten je uvolněn v momentě, kdy počítadlo slabých referencí spadne na nulu. Jeho dekrementace probíhá opět automaticky v destruktoru tříd *GC* i *WeakGC*. V tomto případě

není problém s cyklickými referencemi, protože objekt třídy *RefCount* nikdy nebude obsahovat referenci na žádný ze spravovaných objektů.

6.3 Geolib

Pod pojmem *geolib* je chápána kolekce tříd zapouzdřující veškerou práci s geografickými a geocachingovými daty. Jako taková je také implementována jako celek v kódu jádra aplikace.

Celá knihovna je rozdělena na tři základní části – práce s geografickými body (tj. *keše* a *waypointy*), filtry a podpůrné třídy. Celé schéma geolib je znázorněno na obrázku Obrázek 3.



Obrázek 3 - Schéma geolib

6.3.1 Práce s geografickými body

Jedná se konkrétně o třídy *GenericWaypoint*, *Geocache*, *GeocacheWaypoint* a seznamy *GeocacheList* a *GeocacheWaypointList*. Třída *GenericWaypoint* reprezentuje obecně jakýkoliv bod zobrazitelný na mapě – tedy primárně bod vybavený zeměpisnými souřadnicemi. Z této třídy poté dědí třídy *Geocache* a *GeocacheWaypoint* reprezentující jednu konkrétní *keš* nebo jeden konkrétní *waypoint* určité *keše*. Práce se souřadnicemi je tedy zapouzdřena ve třídě *GenericWaypoint* a potomci této třídy pouze přidávají další vlastnosti ke konkrétnímu bodu.

6.3.2 Filtry

Celý systém filtrů tvoří jeden z pilířů celé práce. Požadavky na filtrování jsou vysoké z hlediska jejich primární důležitosti při práci s programem.

Každé kritérium, podle kterého je možné filtrovat, je reprezentováno jedním z filtrů (jednou třídou) – například pro jméno keše slouží filtr *CacheName*. To ale nestačí, protože obvykle vzniká

požadavek na filtrování podle více kritérií naráz. Existuje tedy třída *Combined*, která obsahuje seznam filtrů, kterými musí daný bod projít, aby byl zařazen do výsledku. S pomocí třídy *Combined* a dílčích filtrů je možné poskládat téměř libovolný jednoduchý filtr.

Může ale vzniknout požadavek na složitější filtrování, například „Chci všechny *keše* s terénem 5 a zároveň chci *keše*, které existují už alespoň 10 let.“ K tomu základní filtrování nestačí, je potřeba mít možnost filtry skládat. K tomu slouží třída *Advanced*, která zapouzdřuje seznam (obvykle *Combined*) filtrů a množinové operace, které se mezi nimi mají provést. Tyto operace jsou: „počáteční načtení“ pro první položku ve filtru, která nemá žádnou předchozí množinu, se kterou by mohla provést nějakou operaci, a dále sloučení, rozdíl a průnik.

S možností takto kombinovat filtry je možné postavit libovolně složité vyhledávací kritérium.

Filtry je možné ukládat do XML formátu a opět je z něj načítat, takže je možná jejich přenositelnost a hlavně perzistence mezi jednotlivými spuštěními aplikace.

Díky použití návrhového vzoru *Composite* je práce s jednotlivými filtry pro zbytek aplikace zcela transparentní a jediná oblast, kde je nutno s konkrétními třídami filtrů pracovat je v době sestavování filtru ať už v GUI nebo během načítání z XML.

6.4 Plugin manager

Plugin manager je z větší části popsán už při návrhu aplikace v kapitole 5.2. Jeho implementace přesně z tohoto návrhu vychází. V této kapitole se chci věnovat problémům přenositelnosti této části kódu mezi platformami a pak některým konkrétním funkcím poskytovaných pluginy.

6.4.1 Obecně o plugin manageru

Plugin manager je navenek reprezentován jedinou třídou *PluginManager*, která je navíc typu *Singleton*, tedy existuje od ní vždy pouze jedna instance. *PluginManager* zapouzdřuje veškerou práci s *pluginy*. Stará se o jejich počáteční načtení, zjištění jejich dispozic a poté slouží zbytku systému k vyhledání konkrétního *pluginu* pro požadovanou funkcionalitu.

Oproti návrhu v současném stádiu implementace není používán *selector*, protože se ukázalo, že zatím není žádný dostatečně obecný případ použití, kdy by se nehodilo mít specifičtější dialog pro výběr. Nicméně funkcionalita *selectoru* je v *PluginManageru* implementována a tak je možné ji využít později, pokud se vyskytne důvod.

Každý plugin je poté reprezentován jednou instancí abstraktní třídy *Plugin*. Proč abstraktní je více rozebráno v kapitole 6.4.2. Tato třída slouží k načtení konkrétního pluginu, zjištění jeho poskytovaných funkcí a jeho korektní uzavření v době ukončování aplikace.

6.4.2 Multiplatformnost

Z důvodu odlišnosti platforem obecně založených na standardech POSIX a platformy MS Windows bylo potřeba řešit zcela odlišný způsob načítání dynamických knihoven do paměti při startu programu. Z tohoto důvodu jsem přistoupil k dvojí implementaci některých metod sloužících pro práci s pluginy. Konkrétně se jedná o metody sloužící k načtení pluginu z disku do paměti, jejich správné inicializaci a ukončení v době ukončování programu.

Existuje tedy specifická implementace těchto metod pro POSIX a pro MS Windows a v době sestavování výsledného programu se použije podle cílové platformy jedna nebo druhá.

Z tohoto důvodu bylo nutné třídu *Plugin* implementovat jako abstraktní s tím, že konkrétní implementace se poté liší právě podle použité platformy, protože každá z platforem používá jiný přístup, jak již bylo zmíněno. Existuje tedy třída *PosixPlugin* a *Win32Plugin*, avšak z hlediska zbytku systému se tyto dvě třídy chovají zcela transparentně a všude je používána pouze instance základní třídy *Plugin*. Tak je do budoucna možné, v případě potřeby, přidat podporu pro další zcela odlišný systém, aniž by bylo nutno upravovat jakoukoliv další část systému.

6.4.3 Co pluginy poskytují

Ze začátku jsem předpokládal, že *pluginy* budou poskytovat konkrétní funkce, tedy například „import“, „database“ apod. Později se ale ukázalo, že hlavně z hlediska vazby na GUI a uživatelskou přívětivost, je vhodné předávat s každou funkcionalitou i další metainformace – například lidsky čitelný název databázového *pluginu* nebo popis exportního formátu.

Přistoupil jsem tedy k přepracování myšlenky pluginů a místo konkrétních funkcí se vždy vrací seznam *Info tříd, které samy obsahují metody zajišťující konkrétní funkcionalitu.

Dobře je to vidět například právě na exportu. Každý exportní plugin zaregistruje funkci „export“, kterou zavolá uživatelské rozhraní v momentě, kdy chce uživatel exportovat. Tato funkce vrací seznam instancí třídy *ExportInfo*, která obsahuje jednak název exportního formátu (například „PocketQuery GPX“), dále výchozí příponu souboru pokud uživatel příponu nespecifikuje sám (například „xml“) a samotnou funkci, která export provede.

Výhodou tohoto řešení je jednak uživatelská přívětivost a díky vracení seznamu namísto jedné konkrétní instance se otevírá možnost pro plugin poskytovat více funkcí stejného typu, což bude v budoucnu využito minimálně v době implementace některého ze skriptovacích jazyků pro snazší rozšiřitelnost.

6.5 Databáze

Celá práce s databází, stejně jako ostatní součásti aplikace, je navržena jako plugin. Bylo tak nutné specifikovat rozhraní, které každý databázový plugin musí implementovat. To mimo jiné znamená, že

pro každou funkci, kterou je možno v aplikaci udělat, a nějak pracuje s daty v databázi, je nutno mít v rozhraní databáze patřičnou funkci. Není možné nikde jinde mimo databázový plugin pracovat přímo se SQL příkazy, protože z podstaty aplikace nelze zaručit, že databáze v pozadí je vůbec založena na některém z SQL jazyků a už vůbec není možné znát schéma dané databáze. Rozhraní databáze je představováno abstraktní třídou *Database* ve jmenném prostoru *GCM::database*.

Toto řešení má nevýhodu v tom, že některé optimalizace nelze provádět snadno a bude je v budoucnu třeba řešit přidáním dalších funkcí do API, které budou specifické pro konkrétní činnost, kterou je třeba optimalizovat.

V budoucnu bude například nutné provést optimalizaci importu dat, což bude znamenat načíst některé údaje o aktuálně existujících keších v databázi do nějaké interní struktury, kterou bude sama databáze poté používat při ukládání keší. Pro ukládání keší během importu je však v současnosti používána stejná funkce, která se používá při úpravě detailů jedné jediné keše v uživatelském rozhraní, takže databáze nemůže tušit, jestli má provést náročnější přípravu na import většího množství dat, nebo má rychle uložit jen jednu jedinou, pro kterou se příprava nevyplatí. Bude tedy nutno zavést nějakou funkci, která databázi řekne, že je pravděpodobné, že teď bude chodit velký objem dat a má přípravu provést.

To je jen jeden z případů, který bude nutno v budoucnu řešit pro zajištění větší rychlosti aplikace na úkor zesložitění API.

6.5.1 SQLite C++ API

Pro databázi programu využívám knihovny SQLite. Tato knihovna však v základu obsahuje pouze C API, které je sice možné využít i v prostředí C++, avšak není to natolik pohodlné z pohledu programátora. Existuje několik zapouzdření tohoto API pro C++, ale vzhledem k tomu, že krátce před zahájením vývoje této aplikace jsem měl možnost setkat se s vývojem pro Android a s jednou knihovnou pro SQLite v Javě, rozhodl jsem se jít cestou objevování kola a jednoduchý wrapper SQLite funkcí si napsat sám se zachováním pokud možno kompatibility názvosloví a způsobu práce právě s onou Javovskou knihovnou, abych se nemusel učit další rozhraní ale mohl použít již známé praktiky.

Celé API je tedy zapouzdřeno do dvou tříd, které pro moje potřeby postačují, další funkcionalita zatím nebyla třeba, avšak pravděpodobně jak vyvstane potřeba, dojde k rozšíření celé této knihovny o další funkce. Obě dvě třídy zároveň automaticky řeší správu chybových hlášení a automaticky v případě neúspěchu operace informují o chybě, což zjednodušuje ladění aplikace v momentě, kdy programátor zapomene otestovat výsledek a případně si chybu zobrazit.

Jedná se o třídy *Database* a *Stmt* ve jmenném prostoru *GCM::sqlite*. Rozhodl jsem se umístit toto API přímo do jádra systému, protože všechny v budoucnu uvažované databázové pluginy (tj. podpora pro GeoGet a GSAK) právě databázi SQLite používají, tak proč si neulehčit v budoucnu

práci a nemít API vždy snadno dostupné. Umístěním do jádra také mohou všechny *pluginy* toto API využívat aniž by bylo nutné nést s *pluginem* závislost na nějaké další knihovně.

6.6 Templater

Pro zobrazení detailů *keše* a později třeba pro webové rozhraní celého systému, bylo nutno vytvořit jednoduchý šablonovací jazyk, který by dovolil snazší úpravy vzhledu aniž by bylo nutno zasahovat do kódu.

Byl tedy vyvinut Templater. Ten je založen na principu jednoduchého interpretu bezkontextového jazyka. Obsahuje klasické součásti – lexikální analyzátor, syntaktický analyzátor, který zároveň tvoří bytecode, který je poté interpretován v samotném interpretu. Celý preklad je řízen syntaktickým analyzátorem, který si od lexikálního bere postupně tokeny a ty v jednoduché rekurzivní syntaktické analýze postupně zpracovává a při tom rovnou tvoří instrukce pro interpret.

Interpret pak lineárně prochází polem instrukcí a jednu za druhou vykonává. V momentě, kdy dojde k instrukci, která vyžaduje skok (podmínka, cyklus), může se změnit čítač instrukcí i jinak než na další instrukci, podle vyplněné adresy u aktuální instrukce.

Templater nebyl testován ani optimalizován na rychlost zpracování, prozatím je podstatné, že funguje a je schopen správně zpracovat šablonu pro zobrazení detailů keše, což je v současnosti jeho jediné využití. Jeho rychlost je tedy čistě subjektivní záležitost, na testovaných strojích nebyly zaznamenány citelné prodlevy.

Pro potřeby budoucího webového rozhraní bude pravděpodobně nutné nějaké optimalizace provést a rozšířit jeho funkcionalitu.

Nebudu zde podrobněji rozebírat návrh *templateru*, protože to není předmětem této práce.

6.7 Konfigurace

Konfigurace je poměrně návrhově složitá součást aplikace. Správným řešením by bylo, aby celá konfigurace byla poskytována některým z *pluginů*. Jenže pro načtení konfigurace je nutné vědět, který *plugin* k její obsluze použít, k čemuž je potřeba konfigurace.

Prozatím jsem tedy přistoupil k zahrnutí základních konfiguračních možností přímo do jádra aplikace, kdy se ukázalo, že momentálně bude tento přístup dostatečný a až v budoucnu dle potřeby přibude případná možnost odsunutí některých konfiguračních možností do *pluginu*.

Konfigurační třída pracuje s konfigurací v podobě jednoduchých dvojic klíč-hodnota, kdy klíč je vždy textový identifikátor a hodnota může být text, celé číslo nebo desetinné číslo. Tyto tři základní datové typy stačí pro všechny doposud prováděné operace.

Jediný problém je v uložení složitější struktury, konkrétně momentálně šířky jednotlivých sloupců v seznamu keší v GUI. Zde by se hodila implementace pole, která by ale porušovala

jednoduchou strukturu konfiguračního souboru. Proto je tento problém řešen až na úrovni GUI tím, že za identifikátor konfigurační volby je postupně přidáváno inkrementující se číslo podle počtu sloupců.

Konfigurace je standardně ukládána při ukončení aplikace do datového adresáře aplikace. O datovém adresáři více v kapitole 6.8.

6.8 Proces spuštění

Spouštění aplikace *gcm* není tak přímočaré, jak by se mohlo zdát. Je potřeba vše ve správném pořadí inicializovat a myslet na uživatelskou přívětivost.

Celý systém je rozdělen do třech základních bloků, které se postupně musí inicializovat a které se sekvenčně spouští. Prvním z nich je samotný binární soubor – *gcm* nebo *gcm-cli*. O jeho inicializaci se postará operační systém. Druhým blokem je *libgcm* – knihovna obsahující veškerou funkcionalitu. Ta je načtena operačním systémem během načítání binárního souboru a je spuštěna právě z onoho binárního souboru po nastavení preferovaného pluginu pro uživatelské rozhraní (qt4 nebo cli podle zvoleného binárního souboru). Binární soubor tedy obsahuje pouze jednoduché volání *gcmcore_run()* a o zbytek se postará knihovna *libgcm*. Zkompilovaný kód se tak zbytečně neduplikuje do obou binárních verzí a vše společné je obsaženo až ve sdílené knihovně.

Samotná knihovna už poté obsahuje již zmíněnou funkci *gcmcore_run()*, což je ekvivalent funkce *main()*, která se postará o inicializaci všeho ostatního.

Nejprve provede inicializaci *PluginManageru*, což zajistí načtení všech pluginů. To je věc, kterou ještě nejde dělat s uživatelským rozhraním, protože UI je také plugin. Okamžitě po načtení pluginů však už startuje zvolené uživatelské rozhraní, aby o ostatních inicializačních činnostech mohl být uživatel informován způsobem, který je vlastní zvolenému uživatelskému rozhraní.

V případě GUI se zobrazí „splash screen“ (tedy úvodní obrazovka), který zobrazuje informace o právě prováděné akci. Na pozadí pak běží vlastní inicializační vlákno, které nejprve musí načíst konfiguraci, která řídí zbytek systému. Hledání správného konfiguračního souboru sestává z několika kroků, které se sekvenčně zkouší dokud alespoň jeden z nich neuspěje.

První pokus se provede v aktuálním pracovním adresáři, ve kterém byla aplikace spuštěna. Pokud se v něm konfigurační soubor nenachází, systém se pokouší hledat v domácím adresáři uživatele. To je v případě POSIX platformy ~/.gcm/gcm.cfg a v případě Windows se hledá v adresáři AppData/Roaming/GCM/gcm.cfg nebo obdobném ekvivalentu závislém na konkrétní verzi operačního systému.

Pokud ani tam systém žádnou konfiguraci nenalezne, poslední krok se liší opět v závislosti na operačním systému. V případě POSIX se hledá soubor /etc/gcm.cfg a v případě Windows se hledá soubor gcm.cfg v adresáři, kde se nachází spustitelný soubor – tedy v distribučním adresáři aplikace.

Pokud ani jeden z uvedených konfiguračních souborů není nalezen, systém se nakonfiguruje tak, aby pracoval s uživatelovým domovským adresářem, tedy obdobně jako by existoval soubor ~/.gcm/gcm.cfg resp. ekvivalent na Windows.

Po úspěšném načtení konfigurace je určena cesta k datovému adresáři. Pokud je uvedena konfigurační volba *datadir*, použije se hodnota uvedená v této volbě. Pokud uvedena není, opět nastává proces automatického určení. V tomto případě se jedná o proces velice jednoduchý, protože se prostě použije uživatelův domovský adresář doplněný o konkrétní cestu, tedy ~/.gcm/ resp. AppData/Roaming/GCM/ v případě Windows.

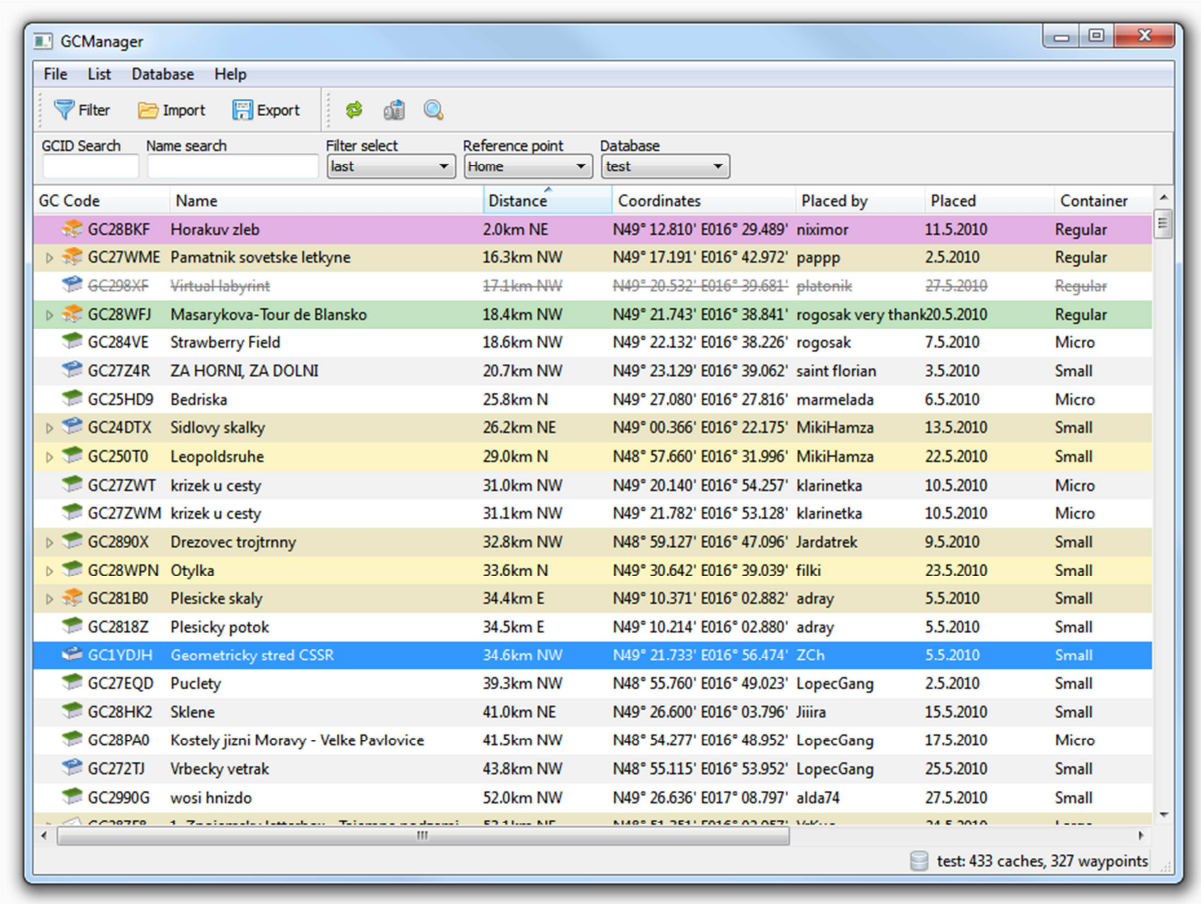
Nyní, když je znám datový adresář, může se systém plně spustit. Nejprve se pracovní adresář změní právě na datový a poté je načtena poslední otevřená pracovní databáze. Dále se načítá seznam referenčních bodů a v případě GUI také seznam uložených filtrů a seznam všech dostupných databází pro umožnění rychlého přepínání. Tím celá inicializace končí a uživatelské rozhraní je připraveno plnit příkazy, které mu uživatel dá.

V této části aplikace bylo nutno vyřešit jeden poměrně zásadní problém. Z hlediska aplikace je výhodné, pokud je aktuální pracovní adresář (working directory) nastaven na datový adresář, čímž se zajistí jistá konzistence uložení dat na správném místě aniž by se o to musela každá část systému starat zvlášť. To přináší ale problém při používání relativních cest zadaných uživatelem, primárně v momentě použití příkazové řádky, kdy se předpokládá, že pracovní adresář je ten, ve kterém uživatel aplikaci spustil. Řeším to tak, že než provedu změnu pracovního adresáře do datového, uložím si cestu k aktuálnímu pracovnímu adresáři do proměnné a poté jakmile některá z funkcí potřebuje pracovat se vstupem od uživatele, si může tuto proměnnou přečíst a vhodně tak upravit cestu zadnou uživatelem, aby získala absolutní cestu k požadovanému souboru. Je tak zajištěno, že se aplikace navenek chová přesně tak, jak od ní uživatel očekává a zároveň je možno pracovat s datovým adresářem.

6.9 Qt GUI

Celé grafické uživatelské rozhraní je natolik zásadní, že jsem se mu, na rozdíl od příkazového řádku, rozhodl věnovat vlastní podkapitolu.

Tvorba tohoto rozhraní pro mě byla prvním setkáním s Qt toolkitem z hlediska programátora, nikdy dřív jsem neměl tu čest s Qt pracovat. Musel jsem si tak projít náročným procesem poznávání a objevování technik, které se v Qt předpokládají. V této práci však nebudu psát žádnou teorii na téma práce s Qt, protože o tomto tématu pojednává množství jiných prací a navíc na internetu je k dispozici obsáhlá nápověda [17], množství příkladů [18] a tutoriálů [19], takže každý má dostatek informačních zdrojů ke studiu.



Obrázek 4 - Hlavní okno aplikace

Na obrázku Obrázek 4 je zobrazeno hlavní okno aplikace, které odpovídá návrhu uvedenému v kapitole 5.3.1. Oproti návrhu je zde navíc panel s možností rychlého vyhledávání a přepínání filtrů, referenčního bodu a databáze.

6.9.1 Seznam keší

Seznam keší je implementován za pomocí virtuálního stromu realizovaného třídou *QTreeView*, která je standardní součásti Qt frameworku. Výhoda tohoto řešení je, že není třeba data pro strom uchovávat v paměti odděleně od skutečných dat. Namísto toho se použije na míru naprogramovaný tzv. model, který propojuje skutečná data v jejich nativním úložišti (v tomto případě seznam keší vrácených z databáze v instanci třídy *GeocacheList*), s rozhraním stromu. Pomocí několika metod (počet potomků rodiče, počet sloupců a samozřejmě hlavně zobrazovací data jednotlivých polí) je tak možno zajistit libovolná data, která strom může zobrazovat. Model se také stará o řazení seznamu, takže veškeré řadící operace musí být také implementovány uvnitř modelu.

Tak, jako model se stará o zpřístupnění dat, je možno podobným způsobem změnit i některé další chování stromu, primárně jeho grafické ztvárnění. O to se nestará *model*, ale tzv. *delegate*, kterému strom deleguje různé operace, v tomto konkrétním případě hlavně ono vykreslování polí.

Díky implementaci vlastního *delegate* bylo možné implementovat barevné pozadí jednotlivých řádků v závislosti na stavu keše, zobrazovat ikonky typu keší a v budoucnosti umožní i plně vykreslovaný grafický sloupeček.

Při implementaci delegate jsem narazil na problém, že strom standardně vykresluje v prvním sloupečku ikonky sloužící k rozbalení a skrytí podstromu, které ovšem nejsou součástí vykreslované položky, takže není možné změnit jejich barvu pozadí. Pomohl jsem si tu tedy drobným trikem, že v případě, že se jedná o vykreslování prvního sloupečku, ručně upravím předané rozměry sloupečku tak, aby levý okraj začínal na 0px. Vykreslím pozadí a poté ručně vykreslím rozbalovací ikonky. Funkčnost zůstane zachována a grafické ztvárnění je přesně takové, jaké by mělo být.

6.9.2 Filtrování

Celý systém konfigurace aktivního filtru je dostupný přes tlačítko „Filter“ v hlavní nástrojové liště. Otevře se dialog, kde je možnost konfigurace všech existujících možností filtrování. Prozatím není možno tento dialog rozšířit pomocí pluginů, avšak v budoucnu počítám s tím, že se filtrování rozroste i o možnost specifikovat další filtry v pluginech. Přináší to však ještě několik problémů, které bude do té doby nutno vyřešit, což zatím nebylo nutné (zejména jde o nějakou standardizaci uživatelského rozhraní, které budou moct používat i pluginy).

Ve funkcích pro obsluhu dialogu filtrování jsou zajímavé dvě funkce – *loadFilter()* a *buildFilter()*. Ta první, jak lze odtušit již z názvu, slouží k nastavení *widgetů* v dialogu podle existujícího filtru a ta druhá funguje přesně opačně – tedy z *widgetů* v dialogu sestaví filtr.

Funkce *loadFilter()* musí umět pracovat i s *Advanced* filtry, tedy složenými filtry pomocí matematických operací, jak je popsáno v kapitole 6.3.2. Tato funkce pracuje rekurzivně – jako parametr očekává jeden z filtrů, podle jeho typu buď nastaví ovládací prvky, nebo v případě *Combined* filtru zavolá sebe samu pro všechny filtry v *Combined* obsažené. V případě *Advanced* filtru vytvoří jednotlivé položky v seznamu filtrů a poté první z těchto filtrů načte do *widgetů* v dialogu, opět rekurzivním voláním sebe sama.

Zároveň si tato funkce musí umět poradit i se speciálními chybnými případy, kdy například v *Advanced* filtru není ani jeden další filtr. V takovém případě je vytvořen jeden výchozí filtr s výchozími hodnotami.

Funkce *buildFilter()* pracuje, jak již bylo zmíněno, obráceně. Vezme hodnoty z *widgetů* v dialogu a sestaví z nich jeden *Combined* filtr. Funkce musí znát výchozí hodnoty každého z filtrů, aby mohla vytvořit jen ty filtry, které skutečně něco dělají. Díky této funkcionalitě neprochází při načítání keší každý objekt zbytečně filtry, které nemají žádný efekt, čímž je dosaženo maximální možné optimalizace v této části procesu.

Celý dialog má ještě jednu zajímavou vlastnost – pamatuje si, zdali došlo k nějaké uživatelem vyvolané změně v nastavení filtru, což je využito hlavně u rychlého výběru filtrů v hlavním okně –

pokud nedošlo k žádné změně, je vybrán správný filtr s názvem souboru. Pokud ale došlo ke změně a uživatel změnu neuložil (k ukládání do souboru nedochází automaticky, aby uživatel drobnou změnou vyvolanou aktuálními požadavky nepřepsal filtr, který chce i nadále využívat v nezměněné podobě), změna se aplikuje do filtru „last“, který nepřepisuje uživatelem zvolený filtr.

6.10 Sestavovací systém

Mám ve zvyku si ke svým projektům psát *Makefile* sám. Je to primárně z důvodu toho, že jsem si nikdy nenašel dostatek času proniknout do tajů programů *autoconf* a *automake* a prozatím jsem se svým přístupem nenarazil na výraznější problémy.

Nejinak tomu tedy bylo i v případě této práce. Celá kompilace projektu je řízena jedním jediným Makefile, který obsahuje hlavně seznam souborů, které má zakompilovat do *libgcm*, seznam *pluginů*, jejichž kompilaci má spustit a pak pravidla pro sestavení obou spustitelných souborů *gcm* a *gcm-cli*.

Jednotlivé *pluginy* poté mají každý svůj *Makefile*, protože jejich kompilace se může lišit, což zejména v případě Qt4 GUI platí více než dostatečně. Pro každý plugin tedy hlavní kompilační postup přejde do jeho zdrojového adresáře a tam spustí *make*. Pluginový *Makefile* obsahuje samotné pravidlo pro sestavení pluginu a pravidlo *clean* sloužící pro vyčištění binárních souborů.

Konkrétně v případě pluginu Qt4 GUI takový Makefile spustí celý automatizovaný proces překladu Qt aplikace stejně, jako to dělá vývojové prostředí. Je tedy zajištěna kompatibilita i s budoucími změnami v sestavovacím procesu Qt aplikací v případě aktualizace rozhraní na novější verzi.

Protože chci mít možnost kompilovat jedním *Makefile* jak na Windows tak na Linuxu, bylo nutné zavést některé platformě závislé specifikace. Oddělil jsem proto definici některých proměnných do zvláštních souborů, které se podle konkrétní platformy použijí. Jedná se konkrétně o soubory *Makefile.w32* a *Makefile.posix*. Zde se nalézá platformě závislé nastavení kompilace, zejména nastavení správných přípon souborů (*.exe* a *.dll* na Windows a *.so* na Linuxu), připojení správné implementace *PluginManageru* pro konkrétní platformu (viz kapitola 6.4.2) a názvy knihoven třetích stran, které jsou připojeny k výsledným binárním souborům.

Později jsem ještě oddělil samotnou definici společných kompilačních proměnných do souboru Makefile.vars, který je společný pro obě podporované platformy. Soubor Makefile tedy obsahuje ve výsledku jen definici objektů pro zkompilování a základní pravidla pro sestavení pluginů, spustitelných souborů i *libgcm*.

Jedním *Makefile* je možná tedy kompilace jak na Windows tak v Linuxu. V Linuxu se použije standardní kompilátor gcc a podpůrné utility ze základního systému. Na Windows se předpokládá přítomnost balíku *mingw* a *msys* včetně základních utilit běžně používaných v Linuxu (*rm*, *mkdir*, ...)

Z důvodu lepší podpory ladění kódu jsem ale později přistoupil k úpravě kódu tak, aby bylo možno přeložit celý projekt i v Microsoft Visual Studiu, jehož ladící nástroje jsou pro systém Windows více pokročilé než ty, které jsou součástí *mingw*. O tom více v následující kapitole.

6.11 Portace na windows

Celý projekt byl od začátku vyvíjen s důrazem na přenositelnost mezi platformami Microsoft Windows a Linux. Z hlediska kódu to znamenalo v některých místech (zejména při práci se soubory nebo při práci s pluginy) počítat s oběma systémy a kód pomocí přepínačů rozdělit na dvě odlišné větve pro každý z obou systémů. Naprostá většina kódu je ale společná pro obě platformy, zejména veškerá funkcionalita je zcela totožná.

Později, při řešení pádů aplikace na Windows se ukázalo, že ladící nástroj *gdb* si na této platformě s projektem moc neporadí. Bylo tedy třeba najít jiný nástroj, který bude schopen říct, kde došlo v aplikaci k chybě.

Po několika pokusech jsem se rozhodl, že bude nutné projekt upravit tak, aby byl přeložitelný i pomocí nástroje Microsoft Visual Studio. V projektu tedy existuje *solution* pro MS Visual Studio 2008, pomocí které je možné projekt v tomto prostředí zkompilovat.

Z toho důvodu, že Windows nejsou POSIX kompatibilním systémem a tomu, že na rozdíl od *mingw*, které se snaží standardní rozhraní z větší části zpřístupnit i pro Windows, ve Visual Studiu žádné takové snahy neexistují, bylo nutno doplnit implementaci několika standardních funkcí, jejichž ekvivalent sice na Windows obvykle existuje, avšak liší se rozhraním. Bylo tedy výhodnější dopsat adaptér Windows rozhraní na standardní POSIXové, než zavádět na množství míst v kódu rozlišování platforem a používání konkrétní implementace vlastní cílové platformě.

Dále bylo třeba vyřešit konvenci zápisu funkcí, které mají být dostupné z knihovny *libgcm* pro použití mimo tuto knihovnu například v *pluginech*. V Linuxu není potřeba pro tento případ zapisovat žádnou zvláštní konvenci, pro MS Windows je však třeba zapsat funkci pomocí tohoto vzoru:

<návratový typ funkce> ***__declspec(dllexport)*** *[*|&] <název funkce>([<parametry>...])*

a pro následné použití tohoto rozhraní externě je třeba místo *dllexport* použít *dllimport*. Vyřešil jsem to definicí makra *GCM_API*, které v době kompilace obsahuje správnou konvenci (*dllexport* verzi při kompilaci *libgcm* a *dllimport* verzi ve všech ostatních případech) a pro Linux je prázdné.

Projekt je tedy nyní kompilován pro prostředí MS Windows ve vývojovém nástroji Visual Studio a pro Linux pomocí standardních nástrojů. Kompilace pomocí *mingw* není nadále udržována, avšak do budoucna, pokud vyvstane potřeba kompilátor opět změnit, by neměl být zásadní problém znovu pomocí *mingw* projekt zkompilovat.

7 Závěr

V této práci jsem nejprve popsal Geocaching ve stručné formě tak, aby z něj vyplynuly požadavky na aplikaci pro správu geocachingových dat. Z potřeb vyplynulo, že je třeba aplikace, která bude umět uchovávat seznamy *keší*, dalších *waypointů* k nim a s těmito daty vhodným a uživatelsky přívětivým způsobem pracovat. Základem takové aplikace by mělo být okno, ve kterém bude zobrazen seznam keší se kterými je možné aktuálně pracovat. Dále by aplikace měla umožňovat druhý pohled na data prostřednictvím mapy, kdy tato mapa bude provázána s daty a bude umožňovat stejné operace jako textový seznam. Protože většina faktů se v průběhu času mění, je výhodné, aby aplikace byla co nejvíce přizpůsobitelná a do budoucna změnitelná.

Dále jsem se v práci věnoval analýze existujících programů GeoGet a GSAK. Ze zkoumání vyplynulo především to, že oba programy mají pevně daná některá omezení, která není možno ani s pomocí jejich možností rozšíření upravit. Oba programy mají i své výhody – zejména v možnosti filtrování dat jsou oba programy na vysoké úrovni. I v této oblasti je ale co zlepšovat.

Každý z programů má odlišnou filozofii práce – GSAK sází primárně na používání *corrected coordinates* a cílí na uživatele navigací Garmin, zatímco GeoGet se snaží být až příliš univerzální a umožňuje tak pracovat pouze s *additional waypoints*, což je naopak pro uživatele Garmin navigací nevýhodné.

Z analýzy existujících produktů vyplynuly jisté požadavky na aplikaci, která by měla řešit zejména špatnou flexibilitu rozšíření obou programů a jejich vzájemnou nekompatibilitu.

Byla navržena aplikace, která by neměla mít do budoucna problém s rozšiřitelností nebo výměnou jednotlivých jejích komponent. Aplikace by měla zajišťovat všechny požadavky na ní kladené s důrazem na uživatelskou přívětivost a snadnost ovládání.

Aplikace byla implementována a problémy řešené při implementaci jsou popsány v kapitole 6.

V této kapitole ještě nastíním budoucnost projektu a zhodnotím celkový přínos aplikace.

7.1 Plány do budoucna

Celá aplikace je v současnosit ve své první verzi, která umožňuje pouze základní práci. Do budoucna je v plánu dle specifikace požadavků implementovat řadu dalších funkcí – zejména bych rád implementoval pořádnou podporu mapy, která chybí v obou hlavních konkurenčních produktech, dále zavedl možnost psaní rozšíření pro aplikaci s pomocí některého z běžných interpretovaných jazyků (pravděpodobně Python) a samozřejmě dále rozšiřoval funkčnost zejména podle zpětné vazby uživatelů.

V nejbližší době zatím neplánuji projektu větší propagaci, protože v současnosti ještě nemůže být tou pravou konkurencí pro existující nástroje. Je třeba ještě projekt podrobit zejména

uživatelskému testování, které bude probíhat v následujících měsících a doplnit možnost spolupráce s konkurenčními programy, aby se uživatelé mohli snadno a bezbolestně rozhodnout, který z programů budou v budoucnu používat a po přechodné období mohli používat oba dva.

Také plánuji rozšíření počtu vývojářů, aby vývoj mohl probíhat nezávisle na několika frontách, čímž se jednak dosáhne větší kontroly nad koncepcí, kdy „bláznivé“ nápady budou ostatními vývojáři okamžitě odmítnuty a hlavně se zrychlí vývoj aplikace a zlepší reakce na připomínky uživatelů.

7.2 Přínos aplikace

Aplikace byla představena prozatím pouze omezené skupině důvěryhodných uživatelů. Jejich primární dojmy byly veskrze pozitivní. Vzešlo několik připomínek jak k funkčnosti tak k vzhledu aplikace. Některé byly již implementovány ve finální verzi práce, některé budou teprve implementovány v budoucnu, některé nebudou implementovány vůbec, protože koncepce programu to neumožňuje.

Všichni oslovení uživatelé si chválili rychlost práce s programem, což byl jeden z hlavních cílů implementace. Prozatím žádný z uživatelů nezkoumal možnosti rozšíření a spolupráce s aplikací, takže z tohoto pohledu zatím nemohu učinit závěr. Sám jsem se ale snažil při návrhu dbát na maximální rozšiřitelnost, což se projevuje zejména v možnosti libovolně implementovat další databázové *pluginy*, importní i exportní *pluginy* a také libovolné další uživatelské rozhraní (například webové).

Projekt hodnotím jako úspěšný start a sám se těším, co se z něj v budoucnosti podaří udělat.

8 Literatura

[1] M. Kuchta, „Podpora Geocachingu pro kapesní počítače s GPS,“ FIT VUT Brno, Brno, 2009.

[2] Groundspeak, Inc., „Geocaching Software,“ [Online]. Available: http://www.geocaching.com/software/. [Přístup získán 9 Ledna 2012].

[3] Groundspeak, Inc., „The History of Geocaching,“ [Online]. Available: http://www.geocaching.com/about/history.aspx. [Přístup získán 29 Prosinec 2011].

[4] M. a. E. Kenneth, „Tex-Czech,“ 1 Června 2001. [Online]. Available: http://www.geocaching.com/seek/cache_details.aspx?wp=GCE50. [Přístup získán 14 Května 2012].

[5] LbNA, „Letterboxing,“ [Online]. Available: http://www.letterboxing.org/. [Přístup získán 9 Leden 2012].

[6] Groundspeak, Inc., „Geocache Listing Requirements / Guidelines,“ 11 Únor 2011. [Online]. Available: http://www.geocaching.com/about/guidelines.aspx. [Přístup získán 29 Prosinec 2011].

[7] Groundspeak, Inc., „Geocache types,“ [Online]. Available: http://www.geocaching.com/about/cache_types.aspx. [Přístup získán 29 Prosinec 2011].

[8] Groundspeak, Inc., „Fundamental Placement Guidelines - Cache Saturation,“ [Online]. Available: http://support.groundspeak.com/index.php?pg=kb.page&id=304#saturation. [Přístup získán 29 Prosinec 2011].

[9] GeoWiki, „Reviewer,“ 28 Prosinec 2011. [Online]. Available: http://wiki.geocaching.cz/wiki/Reviewer. [Přístup získán 9 Leden 2012].

[10] rot13.com, „ROT13,“ [Online]. Available: http://www.rot13.com/info.php. [Přístup získán 29 Prosinec 2011].

[11] Groundspeak, Inc., „Geocaching Live API,“ [Online]. Available: http://www.geocaching.com/live/default.aspx. [Přístup získán 9 Leden 2012].

[12] Groundspeak, Inc., „Groundspeak Memberships,“ [Online]. Available: http://www.geocaching.com/membership/comparison.aspx. [Přístup získán 9 Leden 2012].

[13] Geocaching.cz, „Fórum GeoGet,“ [Online]. Available: http://www.geocaching.cz/forum/viewforum.php?forum_id=20. [Přístup získán 9 Leden 2012].

[14] GeoGet, „Uživatelské skripty,“ 9 Říjen 2011. [Online]. Available: http://geoget.ararat.cz/doku.php/user:skript. [Přístup získán 29 Prosinec 2011].

[15] GeoGet, „Exportní skripty,“ [Online]. Available: http://geoget.ararat.cz/doku.php/user:skript#exportni_skripty. [Přístup získán 9 Leden 2012].

[16] Google, „Google APIs Add-On,“ [Online]. Available: https://developers.google.com/maps/documentation/android/reference/. [Přístup získán 2012].

[17] Nokia Corporation, „Online Reference Documentation,“ 2011. [Online]. Available: http://doc.qt.nokia.com/. [Přístup získán 17 května 2012].

[18] Nokia Corporation, „Qt Examples,“ 2011. [Online]. Available: http://doc.qt.nokia.com/4.7/all-examples.html. [Přístup získán 17 Května 2012].

[19] Nokia Corporation, „Qt Tutorials,“ 2011. [Online]. Available: http://doc.qt.nokia.com/4.7/tutorials.html. [Přístup získán 17 Května 2012].

Přílohy

Příloha č. 1: CD se zdrojovými texty a originálem práce ve formátu PDF a DOC