VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

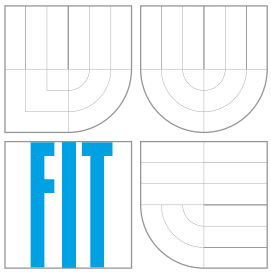
HAM STANIČNÍ DENÍK PRO LINUX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAN KALUŽA
AUTHOR

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HAM STANIČNÍ DENÍK PRO LINUX

HAM LOGBOOK FOR LINUX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN KALUŽA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÁCLAV ŠIMEK

BRNO 2012

Abstrakt

Cílem této bakalářské práce je návrh a implementace software realizujícího staniční deník amatérské radiokomunikační služby. Po stručném úvodu do problematiky následuje popis existujících softwarových řešení staničních deníků včetně analýzy jejich výhod a nevýhod. Na základě této analýzy je dále proveden návrh nového modulárního a otevřeného řešení, které bylo implementováno jako softwarová aplikace pro operační systém GNU/Linux. Zdrojové kódy byly uvolněny pod svobodnou licencí GNU GPL. Na závěr je provedeno zhodnocení implementace a jsou diskutovány možnosti dalšího rozšíření.

Abstract

The goal of this bachelor's thesis is to design and implement an amateur radio logbook software for a GNU/Linux operating system. After the short introduction there are described existent logbook applications together with analysis of their advantages and disadvantages. Based on this analysis the logbook application is designed and implemented. Finally, there is a summary and discussion of possibilities for further improvements.

Klíčová slova

HAM, amatérské radio, deník, klient, server, databáze, C++, Qt, Boost

Keywords

HAM, amateur radio, logbook, client, server, database, C++, Qt, Boost

Citace

Jan Kaluža: HAM staniční deník pro Linux, bakalářská práce, Brno, FIT VUT v Brně, 2012

HAM staniční deník pro Linux

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Václava Šimka.

.....
Jan Kaluža
14. května 2012

Obsah

Kapitola 1

Úvod

Cílem této práce je seznámit s oblastí radioamatérkého vysílání a se způsobem vedení staničního deníku, zhodnotit dosavadní aplikace používané pro vedení staničního deníku a na základě tohoto zhodnocení vytvořit aplikaci pro vedení staničního deníku s moderní modulární architekturou, která umožní jednoduché přidávání nových funkci.

Staniční deník je důležitou pomůckou operátora radioamatérské komunikační služby. Dle platných právních předpisů (vyhláška 156/2005 Sb. o technických a provozních podmínkách amatérské radiokomunikační služby [1]) musí operátoři klubových stanic provozující amatérské vysílání vést staniční deník obsahující seznam provedených spojení. Současná vyhláška již sice nezavádí povinnost vést staniční deník i pro operátory jednotlivce, tak jako předchozí vyhláška, nicméně zavedenou praxí je, že staniční deník vede i většina operátorů jednotlivců a to zejména kvůli statistikám, diplomům a QSL (viz dále). Díky rozvoji výpočetní techniky je dnes prakticky standardem elektronické vedení staničního deníku. To přináší oproti klasickému papírovému deníku mnoho výhod, jež operátorům usnadňují práci a šetří čas. Lze tak např. sledovat aktuální stav a rozložení stanic na mapě světa, automaticky přeladit radiostanici na kmitočet protistanice, získat detailní informace o operátorovi protistanice prostřednictvím Internetu, rychle vyhledávat a vytvářet statistiky podle nejrůznějších kritérií a mnohé další.

1.1 Členení práce

Tato bakalářská práce je rozdělena do šesti kapitol.

První kapitola obsahuje úvod. Dále následuje kapitola popisující stručně historii amatérské radiokomunikační služby, v jejíž podkapitolách jsou rovněž vysvětleny důležité pojmy a popsány služby používané radioamatéry. Třetí kapitola popisuje současný stav, jsou krátce představeny existující aplikace pro vedení staničních deníků včetně jejich výhod a nevýhod. V následující čtvrté kapitole je proveden analýza a jsou navrženy jednotlivé části staničního deníku. Pátá kapitola pojednává a implementaci. V šesté kapitole je zmíněn jednoduchý příklad použití výsledné aplikace a v poslední šesté kapitole je práce zhodnocena a jsou zmíněny možnosti jejího dalšího rozšíření.

Kapitola 2

Amatérská radiokomunikační služba

V této kapitole je definována amatérská radiokomunikační služba, dále je stručně popsána historie amatérského vysílání a jsou zde vysvětleny důležité pojmy používané dále v práci. Informace byly převzaty zejména z [13].

2.1 Definice

Radiokomunikační řád definuje amatérskou službu jako radiokomunikační službu, která slouží k vlastnímu vzdělávání, vzájemné komunikaci a technickému zkoumání uskutečňovanému amatéry, to je plně autorizovanými osobami, které se zajímají o radiotechniku jedině z osobního snažení a bez peněžních zájmů. Obdobně je definována i amatérská družicová služba, která k radiové komunikaci používá vesmírné stanice na družicích.

Radiokomunikační řád je jedním ze základních dokumentů Mezinárodní telekomunikační unie (anglicky International telecommunication union, dále jen ITU). ITU byla založena již 17.května 1865 dvaceti evropskými státy, tehdy pod názvem Mezinárodní telegrafní unie. Naše země stála u jejího zrodu ještě jako součást rakouského mocnářství. ITU je dnes stálou mezinárodní organizací při OSN pro dálkové telekomunikace a správu kmitočtového spektra. Opatření ITU jsou v ČR právně závazná z titulu členství ČR v ITU a v mezích základních dokumentů ITU. O národní koordinaci se v ČR stará Český telekomunikační úřad (ČTÚ).

2.2 Historie

Prvotní motivací pro vznik amatérského vysílání byla myšlenka našich předků radio nejen poslouchat, ale také nezávisle vysílat. To dalo vzniknout novému druhu zábavy - amatérskému vysílání (HAM Radio). Lidem, kteří vysílali se pak začalo říkat radioamatéři.

Amatérské vysílání se začalo rychle šířit po první světové válce hlavně díky vlivu rozhlasu. V těchto dobách radioamatéři značně přispěli k získání znalostí o využití celého spektra radiových vln. Vysílali na dlouhých a středních vlnách, které byly do té doby považovány za bezcenné, a také na vlnách krátkých, o jejichž existenci se nevědělo vůbec.

Na dlouhých vlnách potřeboval například Marconi mnoho kilowattů výkonu a kilometrové antény pro první spojení mezi Evropou a Amerikou. Oproti tomu stačila radioamatérům v roce 1923 díky využití krátkých vln pro stejné spojení jen energie jedné žárovky

a anténa o velikosti pár desítek metrů. To odstartovalo další komerční využívání těchto pásem.

První pokusy o transatlantické spojení začaly lákat také Pravoslava Motyčku - prvního známého radioamatéra u nás. Koncem roku 1924 navázal Motyčka první spojení v Československu. Díky Motyčkově inspiraci bylo u nás kolem 30. let 20. století až několik set radioamatérů.

Dalším významným rokem byl rok 1928, kdy byl ruský radioamatér prvním, kdo zachytil volání vzducholodi ITALIA, která ztroskotala na cestě od severního pólu. To změnilo pohled na amatérské rádio, do té doby vnímané spíše jako zábava. Při mnoha pohromách na celém světě pak pomáhala připravenost radioamatérů poskytnout spojení do jiných míst.

2.3 Amatérské vysílání dnes

Radioamatéři jsou aktivní prakticky na všech pásmech, pracují s analogovými i digitálními druhy provozů, využívají moderní technologii jako např. spojení přes převaděče na nízkoorbitálních satelitech, vesmírné stanicii ISS, atp.

Za základní pilíř radioamatérského snažení lze však stále považovat vysílání na krátkých vlnách, jež umožňuje spojení do celého světa včetně exotických zemí. K tomu se nejčastěji využívá šíření vln odrazem od různých vrstev ionosféry.

Řada radioamatérů se snaží o co nejdelší spojení s co nejvíce exotickými zeměmi. Tato aktivita se obecně nazývá DX provoz neboli DXing. Zkratka DX je původně souhrné pojmenování pro dálkový příjem/vysílání. Pochází z anglického „distance x“, neboli vzdálenost neznámá. V současné době radioamatéři pod pojmem DX nejčastěji rozumí mezikontinentální spojení. Po úspěšně navázaném spojení si radioamatéři vyměnují QSL lístky (písemné potvrzení o navázaném spojení ve formátu pohlednice). Nepsaným pravidlem je výměna QSL lístků při prvním spojení s novou stanicí. Jedním z cílů tohoto snažení je získání QSL lístků z co možná nejvíce zemí světa (ideálně všech).

U profesionálního komerčního vysílání (například FM rozhlas nebo televize) je třeba pro dosažení kvalitního příjmu být od vysílače vzdálen maximálně v desítkách kilometrů. Tyto služby jsou navrženy pro stabilní pokrytí určité oblasti v definované kvalitě. Radioamatérům však nejde o profesionální kvalitu. Důležitější je úspěšné navázání spojení. Radioamatáři se neustále snaží o překonávání limitů. Pokud se jim například podaří uskutečnit spojení na jednom pásmu, snaží se o totéž i na dalších pásmech, jež však nemusí být za daných podmínek pro spojení nejvhodnější a například komerční služby by o takovém použití vůbec neuvažovaly.

K navázání spojení radioamatéři využívají kromě kvalitní techniky také dobrých znalostí faktorů ovlivňujících šíření krátkých vln. Těmi jsou např. sluneční aktivita, stav ionosféry, denní a roční doba a taktéž aktuální meteorologické podmínky jako například výskyt rozhraní teplých a studených mas vzduchu. Využívají se i znalosti o výskytu meteoritických rojů nebo polárních září. Profesionálové se nemohou na tyto přechodné faktory spolehnout, ale radioamatéři jsou díky nim schopni navázat spojení prakticky po celém světě. Radioamatéři pracující na krátkých vlnách jsou nezávislí na externí infrastruktuře. O jejich úspěšnosti či neúspěšnosti rozhoduje kvalita použité techniky, jejich znalosti a schopnosti, kvalita použitých anténních systémů a příroda. Nezávislost na externí infrastruktuře je velkou výhodou např. v případě živelních pohrom. V těchto krizových situacích, kdy jsou často veřejné komunikační sítě vyřazeny z provozu, jsou radioamáteři schopni poskytovat spojení s okolním světem a předávat informace nezbytné pro řízení záchranných operací.

V tomto směru se radioamatéři v poslední době pozitivně vyznamenali např. při nedávných neštěstích v Japonsku a v Thajsku.

Další radioamatérskou aktivitou je práce na VKV/UKV (neboli velmi krátkých/ultra krátkých vlnách). Tato spojení se vyznačují mnohem vyšší kvalitou srovnatelnou s profesionálními službami, nicméně pro jejich navázání je třeba přímá radiová viditelnost mezi jednotlivými stanicemi (v praxi je přímá radiová viditelnost většinou srovnatelná nebo díky odrazům a lomům trochu vyšší než přímá optická viditelnost). Z toho důvodu se na řadu vysokých budov a kopců umisťují tzv. pozemní rádiové převaděče. Převaděč přijímá signál na jedné frekvenci, zesiluje ho a přeposílá dále na frekvenci jiné. Díky tomu lze spolehlivě pokrýt oblasti jako města či kraje. Dalším častým vylepšením je propojování těchto převaděčů do sítí. Díky tomu mohou radioamatéři navazovat spojení i se stanicemi nacházejícími se v dosahu propojených převaděčů. Velice oblíbený v tomto směru je např. systém Echolink [19], který umožňuje celosvětově vytvářet tzv. červí díry (wormholes) mezi libovolnými převaděči v této síti.

Příklad použití: stanice v dosahu brněnského převaděče zadá kód 366111, což zajistí propojení s převaděčem v Honolulu (propojení je nejčastěji realizováno pomocí sítě Internet). Radioamatéři se tak mohou pomocí převaděče dovolat na velké vzdálenosti i s použitím vysílačů o malém výkonu (často jde o kapesní radiostanice o velikosti srovnatelné s mobilním telefonem). Jistou daní je však závislost na externí infrastruktuře, což komplikuje využití této technologie v případě krizových situací.

V počátcích 60. let byl také vypuštěn první radioamatérský satelit. Dnes jich existuje několik desítek. Tato činnost je koordinována neziskovou organizací AMSAT [5]. Narozdíl od geostacionárních družic, které šíří např. televizní signál a jsou tzv. zaparkovány na vysokých oběžných drahách, však amatérské družice kolem Země obíhají na nízkých oběžných drahách (tzv. LEO - Low Earth orbit). Díky tomu jsou schopny pokrýt postupně signálem všechny obydlené oblasti.

Radioamatéři rovněž pořádají závody. Cílem je například v určitém čase navázat spojení s co nejvíce jinými amatérskými stanicemi v co nejvíce zemích. Jinou variantou je například součet vzdáleností všech spojení vykonaných za určitý čas. Jednotlivé závody mají svoje definovaná pravidla a jsou vypisovány pro různé kategorie.

K radioamatérům neodmyslitelně patří Morseova abeceda. Morseova abeceda byla prvním dorozumívacím prostředkem a je radioamatéry používána dodnes. Výhodou Morseovy abecedy je její velká odolnost proti rušení, diký níž lze překonat velké vzdálenosti i s malým výkonem. Taktéž pro její příjem a vysílání stačí principiálně jednoduchá zařízení. Další výhodou je to, že odbourává jazykové bariéry, protože se v ní komunikuje převážně pomocí krátkých mezinárodních kódů (tzv. Q kódy). Při hlasové komunikaci je mezinárodním jazykem angličtina (pokud však protistanice rozumí, lze pro spojení použít i libovolný jiný jazyk) a využívá se nejčastěji amplitudové modulace s potlačeným postraním pásmem SSB nebo FM modulace. Nejnověji se experimentuje i s různými digitálními režimy přenosu hlasu, např. D-STAR, P25 C4FM, atp.

Historicky se však digitálně přenášely jen data o malém objemu a původně jen nízkou přenosovou rychlostí. S postupným vývojem však dochází k nárůstu přenosové rychlosti. Tyto systémy přenosu dat se nazývají Packet radio. V podstatě se jedná o radioamatérský Internet, kde se informace přenáší pomocí modifikace protokolu X.25 (tzv. AX.25) [7]. Jednotlivé uzly sítě jsou vzájemně propojeny (většinou radiovou cestou, nověji i přes Internet) a pakety putují přes jednotlivé uzly. Síť packet radia pokrývá celý vyspělý svět, její součástí jsou i BBS (Bulletin board system - databanky pro ukládání a výměnu informací) a elektronická pošta. Poslat e-mail kamarádovi do Austrálie tedy nemusíme jen po Internetu,

ale také amatérským packet radiem, případně lze trávit hodiny zkoumáním nových zpráv i zajímavých programů v desítkách BBS.

Lze vysílat i obrazové signály. Dlouhou tradici má tzv. pomalá televize (Slow scan TV - SSTV), jaká byla použita i u přenosu obrazu z prvních přistání amerických astronautů na Měsíci, dnes se experimentuje s různými digitálními přenosy a na UHF kmitočtech je možné použít i klasickou televizní normu.

Z ryze sportovních aktivit je dobře známý rádiový orientační běh (ROB). Cílem je v nejkratším čase najít pomocí přijímače v terénu ukryté zamaskované vysílače. Tyto sporty jsou atraktivní a pěstují je i lidé, kteří se jinak o radioamatérství příliš nezajímají.

Z dalších aktivit radioamatérů jistě zaslouží zmínku EME (Earth Moon Earth), kdy se navazují spojení odrazem od měsíce. Tato aktivita se v poslední době těší čím dál tím vyšší oblibě. Vzhledem k vysokým ztrátám přenosové cesty komunikace většinou probíhá hluboko pod úrovní šumu pomocí digitálního režimu WSJT.

Zajímavou oblastí jsou i experimenty v oblasti mikrovln, kde jsou radioamatéři držiteli řady světových rekordů.

Radioamatérem se dnes může stát každý, kdo složí státní zkoušku z radiokomunikačních předpisů, radiokomunikačního provozu, elektrotechniky a radiotechniky, díky čemuž získá průkaz odborné způsobilosti a následně pak i koncesi a vlastní značku. Více informací lze nalézt např. na [14].

Tato podkapitola byla pokusem o stručný popis nejrozšířenějších činnosti v současnosti provozovaných radioamatéry. Rozhodně zde nebyly popsány všechny činnosti, protože to by si vyžádalo několikanásobně více prostoru.

2.4 Staniční deník

V této podkapitole je stručně popsán staniční deník a také údaje do něj zapisované. Při popisu jednotlivých údajů v této kapitole se vychází z [11].

Dle §5 Vyhlášky o technických a provozních podmínkách amatérské radiokomunikační služby [1] mají klubové stanice povinnost vést staniční deník a ukládat do něj zejména tyto údaje: datum, čas a dobu trvání uskutečněného rádiového spojení, použité kmitočtové pásmo, druh provozu, volací značku stanice a stanoviště klubové stanice, se kterou bylo uskutečněno rádiové spojení.

Ačkoliv je tato povinnost nyní stanovena pouze pro klubové stanice (předchozí vyhláška ji stanovovala pro všechny stanice), většina radioamatérů si vede elektronický staniční deník kvůli přehledu, statistikám a vyřizování QSL agendy.

Staniční deník si lze představit jako databázi spojení provedených operátorem dané stanice. Kromě základních údajů definovaných výše zmíněnou vyhláškou se do staničních deníků ukládají i další údaje nad její rámec. V následující části jsou tyto jednotlivé údaje popsány a vysvětleny.

Obrázky použité dále v této podkapitole byly získány z [8].

2.4.1 Volací značka

Každá stanice má přidělenu jedinečnou volací značku, kterou operátor stanice uvádí při každém spojení. První dva znaky (někdy méně) značky jsou většinou určeny podle přehledu mezinárodních volacích znaků přidělených dané zemi. Dalším znakem je číslice 0-9. Kombinaci těchto znaků se říká prefix (většinou 1 až 3 znaky). Zbytek značky je tvořen jedním až čtyřmi znaky.

Na základě prefixu tak lze určit zemi, ze které daný operátor pochází a tak i jeho přibližnou polohu. K tomu slouží seznam DXCC (DX Century Club) [3] obsahující všechny aktuálně používané prefixy a informace k nim.

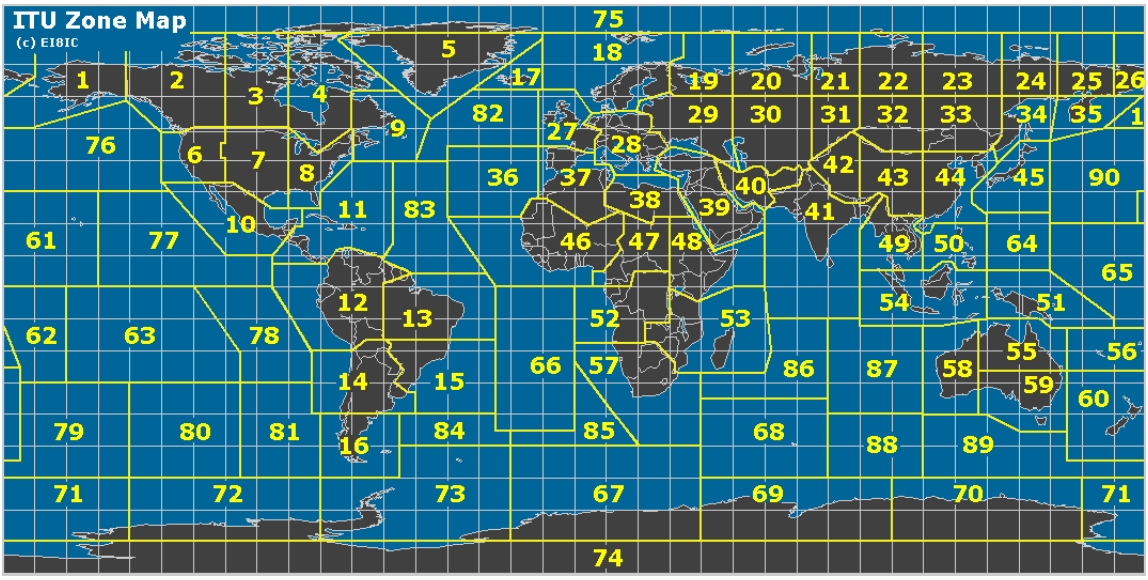
2.4.2 Druh provozu

Na jednom kmitočtu může být spojení uskutečněno různými druhy provozu. Pro úspěšné navázání spojení je tedy nutné, aby obě stanice použily stejný kmitočet a stejný druh provozu. Analogové druhy provozu jsou například: telegrafie (CW), frekvenční modulace (FM), amplitudová modulace (AM), amplitudová modulace s jedním potlačeným postraním pásmem (SSB). Digitální druhy provozu jsou například: RTTY, AMTOR, PACTOR, PSK, WSJT, D-Star, P-25, atd.

Aby nedocházalo k vzájemnému rušení je pásmo rozděleno na jednotlivé úseky s definovaným využitím (např. kmitočty pro telegrafii, kmitočty přednostně určené pro mezinárodní spojení, atp.). Toto rozdělení definuje kmitočtový plán (pro KV k dispozici např. na [10], pro VKV k dispozici např. na [10]).

2.4.3 ITU zóny

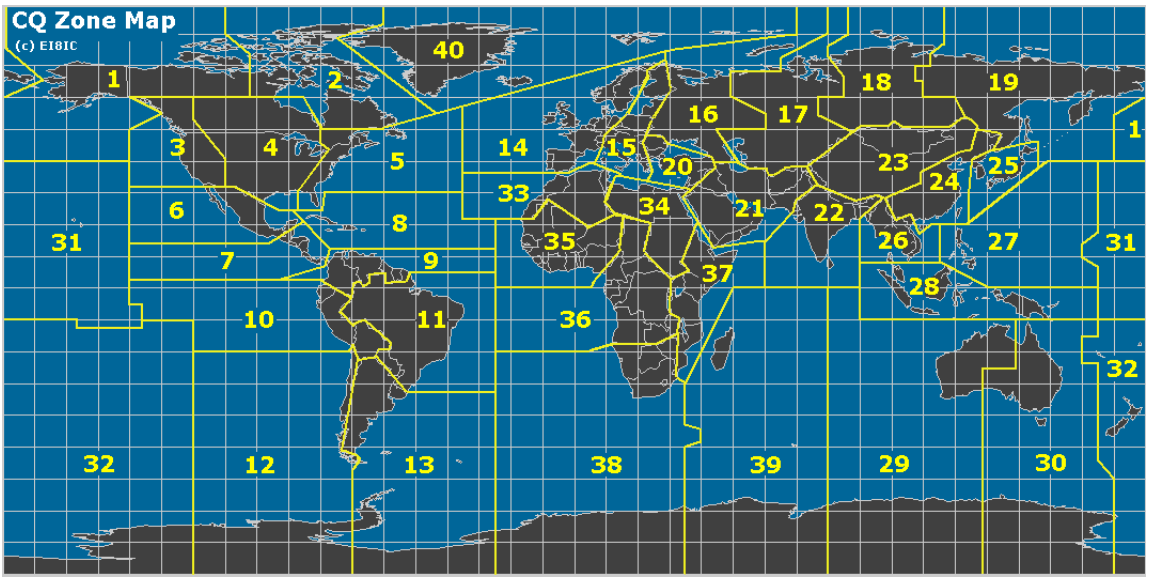
Lokalizace protistanice je velmi důležitá, protože nám umožňuje zjistit, jak daleko jsme se vlastně dovolali. V dnešní době tak rozšířené GPS, na toto není úplně nejvhodnější, protože telegrafický přenos velkého množství číslic by nebyl praktický a mohlo by docházet ke zbytečným chybám. Z tohoto důvodu se používají různé jiné systémy rozdělení světa. Jedním z nich je rozdělení do takzvaných ITU zón. Toto rozdělení lze vidět na obrázku 2.1. Systém je pojmenován podle organizace ITU (viz úvodní kapitola). Svět je rozdělen do celkem 89 zón ve třech regionech.



Obrázek 2.1: Rozdělení ITU zón.

2.4.4 CQ zóny

Dalším z možných rozdělení světa používaných radioamatéry je dělení na CQ zóny, pojmenované podle časopisu CQ. jak je zřejmé z obrázku 2.2, jde o pouze 40 zón.



Obrázek 2.2: Rozdělení CQ zón.

2.4.5 QTH lokátor

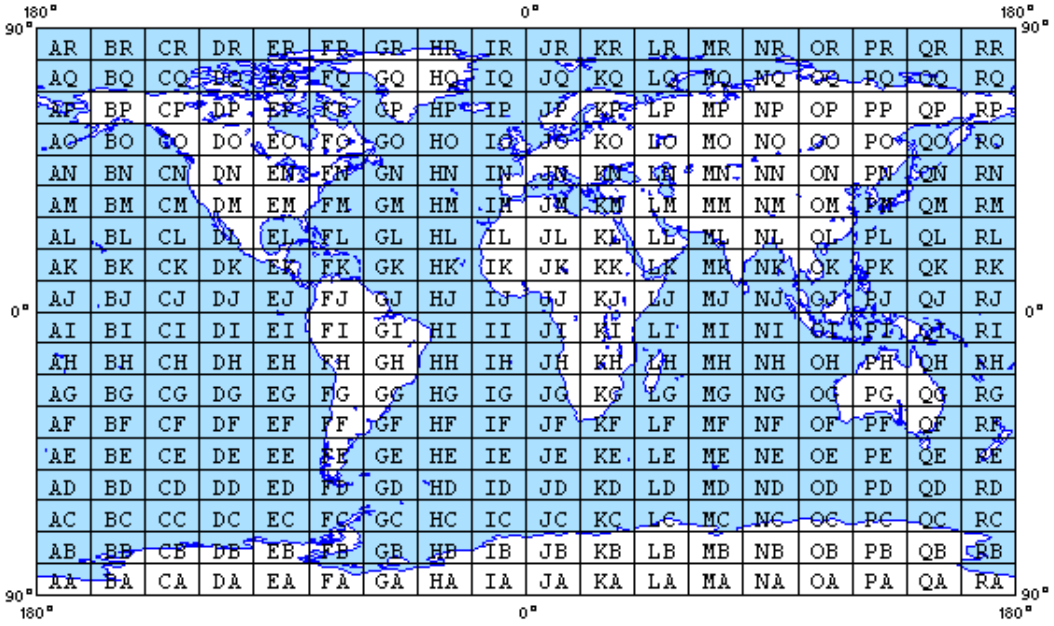
Poslední z používaných systémů pro rozdělení světa je QTH lokátor. Jde o rozdělení na základě zeměpisné šířky a délky. Země je rozdělena do obdélníků o velikosti 1°x 2°(tzv. velký čtverec). Ty lze hierarchicky dělit na menší obdélníky. Vzniká tak hierarchický souřadnicový systém tvořený 2-8 znaky/číslicemi, jehož výhodou oproti GPS souřadnicím je, že stačí zachytit např. jen první dva znaky a známe přibližnou polohu protistanice. Další znaky pak slouží ke zpřesnění polohy. Nejčastěji se mezi radioamatéry používá 4 (na KV) nebo 6 (na VKV) znaků, např. JN89GE označuje západní Brno. Toto rozdělení je zobrazeno na obrázku 2.3

2.4.6 Systém RST

Důležitým údajem, který si stanice navzájem sdělují, je kvalita přijatého signálu. V praxi se pro toto používá systém RST. Výměna RST reportu je nezbytným předpokladem platnosti radioamatérského spojení. Název systému RST je tvořen iniciálami veličin, které obsahuje: Readability (čitelnost), Strenght (síla), Tone (tón). RT se určuje pouze sluchem (vyžaduje určitý cvik), S se odečítá z měřiče síly signálu (tzv. S-meter). Při telegrafii se používá systému RST, při analogovém přenosu řeči se používá pouze systému RS, při digitálním přenosu se používá systému RSQ (Readability, Strenght, Quality).

Každá veličina se udává pomocí číslice z intervalu 1 - 9 (čitelnost z intervalu 1 - 5), kde nejnižšímu číslu odpovídá nejnižší kvalita. Maximálně nejlepší report v systému RST je vyjádřen údajem 599, v systému RS údajem 59.

Slovní popis jednotlivých hodnot veličin RST pro analogové a digitální módy zde z kapacitních důvodů není zmíněn, lze jej však nalézt na stránkách Českého Radioklubu [12].



Obrázek 2.3: Mapa rozdělení pomocí QTH lokátoru.

2.4.7 Informace o výměně QSL lístků

Jak již bylo zmíněno v kapitole 2.3, po úspěšně navázaném spojení si radioamatéři vyměnují QSL lístky. Jedná se o kartičky podobné pohlednicím. QSL lístek obsahuje především nejzákladnější informace o operátorovi. Jde hlavně o jeho značku, zemi, umístění stanice včetně zařazení do zón a lokátoru. Musí také obsahovat značku protistanice, datum a čas navázání spojení, kmitočet, druh modulace a report v systému RST.

Ve staničním deníku je pak vhodné u konkrétního spojení uchovávat informaci o tom, jestli operátor již poslal druhé straně QSL lístek a jestli již od druhé strany QSL lístek obdržel.

2.5 Služby používané radioamatéry

Díky rozšiřování Internetu se začaly rozvíjet i služby provozované radioamatéry pro další radioamatéry. V této podkapitole jsou stručně popsány online služby, které jsou v navrhovaném staničním deníku využity.

2.5.1 DXCluster

DXCluster je komunitní službou umožňující radioamatérovi informovat ostatní radioamatéry o tom, kdo, kde a odkud aktuálně vysílá. Díky této službě tak lze jednodušeji navazovat spojení. Protokol služby DXCluster je v Internetu postaven na službě telnet a není nikterak složitý. Lze jej jednoduše číst a zpracovávat. Tato služba byla historicky šířena přes Packet radio, ale v současné době se k ní lze pohodlně připojit i přes Internet.

2.5.2 QRZ.com

QRZ.com je v současnosti asi největší online databází radioamatérů z celého světa. Na základě volací značky lze s její pomocí zjistit detailní informace (například jméno nebo přesnou polohu) o jejím majiteli. Vyhledávat lze i pomocí speciálního webového rozhraní postaveného na technologii XML. To je velmi výhodné pro tvorbu aplikací využívajících QRZ.com pro zjištění informací o operátorech.

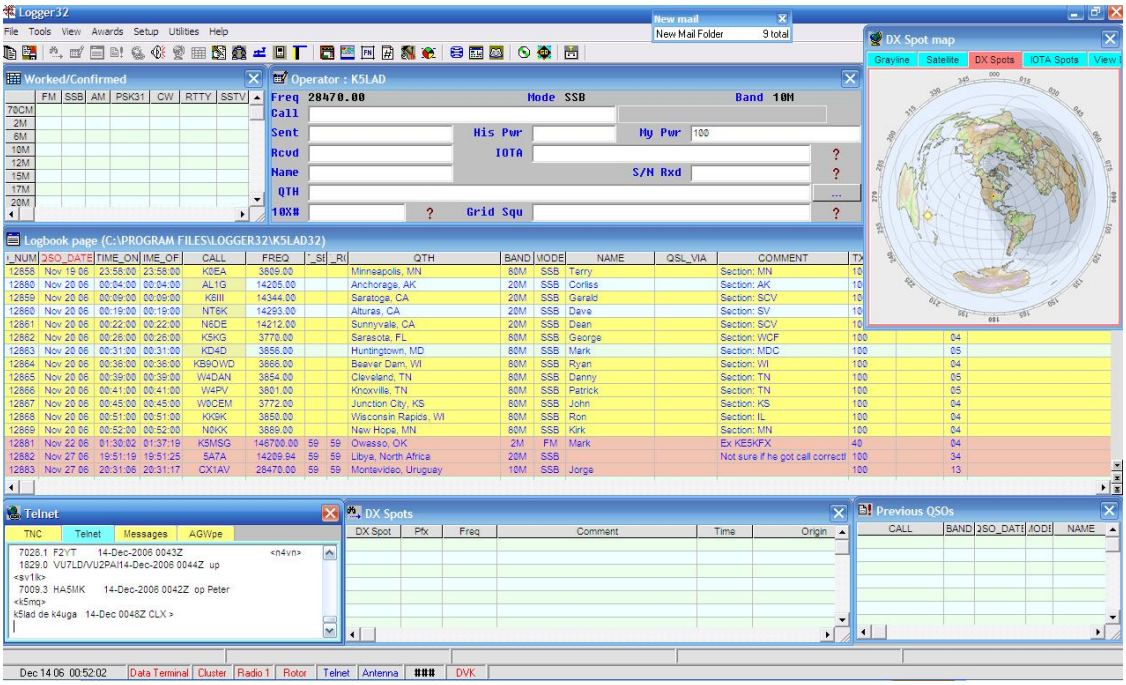
Kapitola 3

Současný stav

V této kapitole jsou popsány existující softwarové staniční deníky. Každá aplikace je stručně popsána a jsou zmíněny i její klady a zápory. Na základě této kapitoly bude dále vytvořen vlastní návrh staničního deníku.

3.1 Logger32

Logger32 je jedním z nejznámějších staničních deníků pro operační systém Windows. Jeho hlavní výhodou je přehledné a funkční uživatelské rozhraní (viz obrázek 3.1), které poskytuje pokročilé možnosti přizpůsobení a podpora přídavných pluginů.



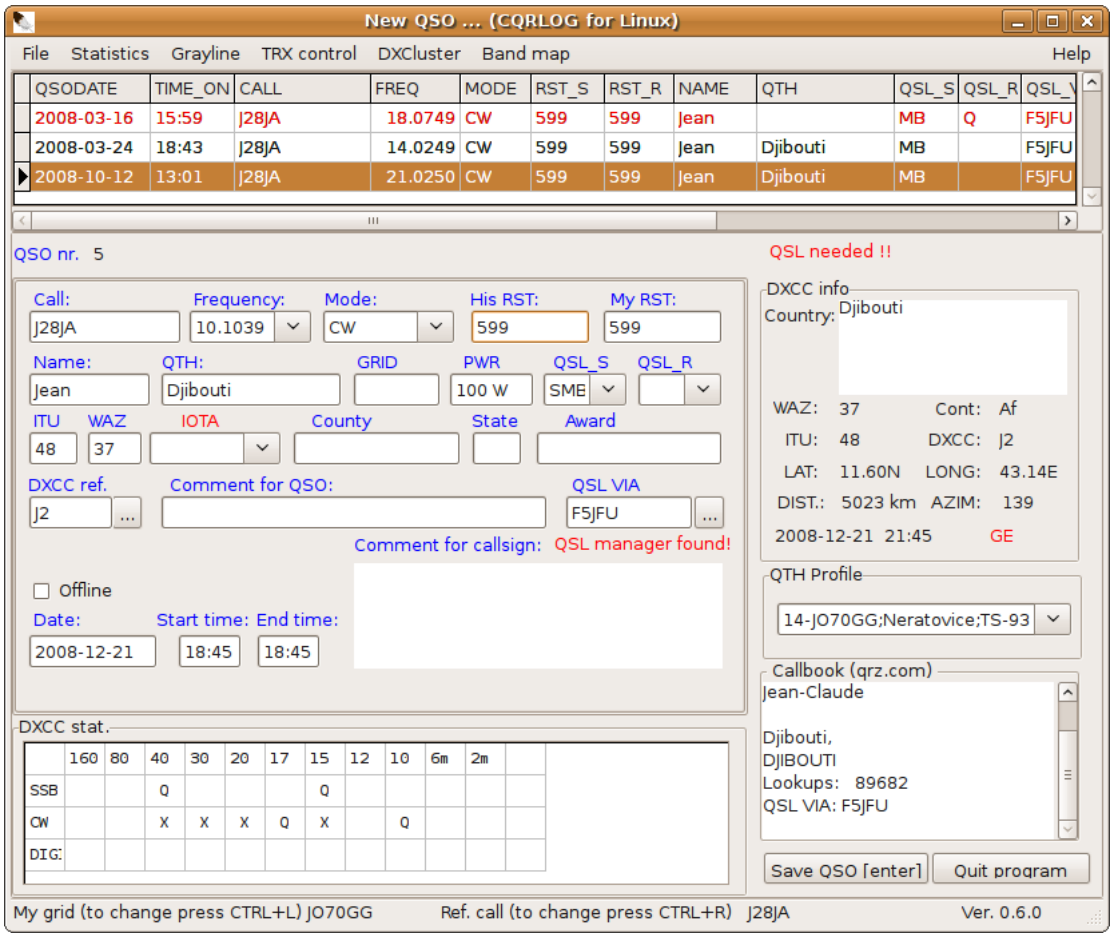
Obrázek 3.1: Uživatelské rozhraní Logger32.

Lze vytvářet pluginy pro vyhledávání detailních informací na základě volací značky a také pluginy vytvářející externí okno s rozšiřujícími informacemi. Logger32 také umožňuje dálkové ovládání radiostanice a příslušenství. Jeho velkou výhodou je univerzálnost.

Hlavní nevýhodou aplikace Logger32 je její závislost na systému Windows. Další nevýhodou je absence zdrojových kódů, takže je rozšiřitelnost aplikace limitována rozhraním modulů definovaným tvůrcem aplikace. Nevýhodou je též implementace v jazyce Visual Basic (výkonostní problémy). Taktéž pro ukládání dat nelze použít externí databázi jako například SQLite3 nebo MySQL. Problémem může být rovněž nemožnost přistupovat do databáze jiným způsobem, např. není možné řešit situaci, kdy by uživatel potřeboval z terénu zapsat nové spojení z mobilního telefonu nebo jiného zařízení s nestandardním operačním systémem.

3.2 CQRLOG

CQRLOG je nejpoužívanějším staničním deníkem z řad multiplatformních aplikací. Je možné jej používat jak v systému Windows, tak v Linuxových distribucích.



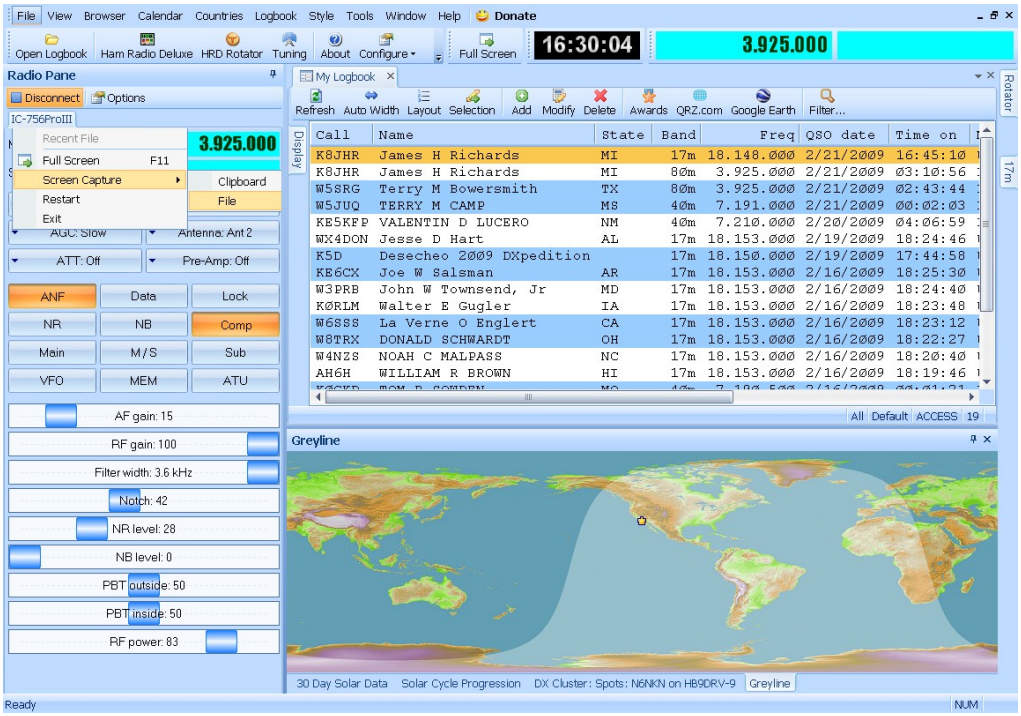
Obrázek 3.2: Uživatelské rozhraní CQRLOG.

Jak lze vidět na obrázku 3.2, je jeho uživatelské rozhraní jednodušší než rozhraní programu Logger32 a je také více přehledné. Na druhou stranu však chybí podpora rozšíření formou pluginů. K dispozici jsou zdrojové kódy v jazyce Delphi. Ke kompilaci je potřeba vývojového prostředí Lazarus. Volba tohoto programovacího jazyka a závislosti na prostředí Lazarus značně ztěžuje (až znemožňuje) distribuční integraci, t.j. tvorbu oficiálních instalačních balíčků pro linuxové distribuce.

CQRLOG umožňuje použití MySQL databáze, ale návrh aplikace znemožňuje jednoduchou změnu databázové aplikace. Také volba MySQL databáze pro použití v desktopové aplikaci může být vnímána jako negativum. Návrh CQRLOGu rovněž nedefinuje rozhraní pro jednoduché přidávání nebo úpravu záznamů pomocí aplikací třetích stran a není tak jednoduché postavit další aplikaci nad CQRLOGem.

3.3 HAM Radio Deluxe

HAM Radio Delux, jehož uživatelské rozhraní lze vidět na obrázku 3.3, je dalším ze zástupců staničních deníků pro operační systém Windows. Uživatelské rozhraní vypadá na první pohled přívětivě a odladěně. Problémem však může být množství voleb, ve kterých se uživatel lehce ztratí.



Obrázek 3.3: Uživatelské rozhraní HAM Radio Deluxe.

Jedná se o uzavřenou aplikaci, což brání jejímu dalšímu rozšiřovaní zájemci z řad široké veřejnosti. Oproti aplikaci Logger32 neposkytuje ani API pro tvorbu pluginů. Na druhou stranu obsahuje HAM Radio Deluxe všechny potřebné nástroje pro provoz staničního deníku. Umožňuje také použití MySQL nebo MSSQL databáze. Nevýhodou tak zůstává jeho uzavřenost.

Kapitola 4

Návrh

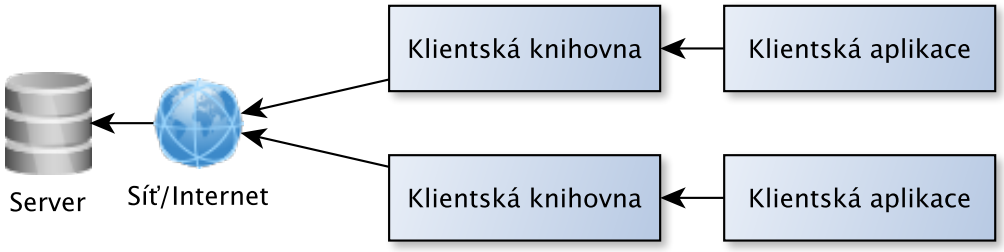
Cílem této kapitoly je návrh aplikace a použitého komunikačního protokolu.

Na základě analýzy z předchozí kapitoly vyplynuly základní požadavky na výslednou aplikaci.

- **Multiplatformnost** - Nutným požadavkem je možnost spuštění aplikace (nebo přeložení zdrojového kódu) na různých operačních systémech / platformách.
- **Architektura klient-server** - Dalším požadavkem je použití klient-server architektury. Tato architektura je výhodná, protože umožňuje zapisovat do deníku nové záznamy z různých zařízení (klientů) a přitom udržovat deník na centrálním místě (serveru). Server by měl umožnit vedení deníků více uživatelů zároveň a poskytnout rozhraní pro přídavné moduly. To umožní v budoucnu vést jednotný staniční deník s centrálním uložištěm i pro radioamatérské kluby s více radiostanicemi. Klientská aplikace se pak pouze připojí k serveru a prostředníctvím komunikačního protokolu s ním komunikuje. Díky tomuto rozdělení je možné přidávat záznamy do centrální databáze z více zařízení a pracovat tak všude se stejnými daty. Využití klient-server architektury je výhodné také pro soutěže, kdy může vyhodnocení výsledků všech účastníků proběhnout současně na centrálním místě.
- **Modulárnost** - Třetím požadavkem vyplývajícím z předchozí kapitoly je nutnost modulárního návrhu. Díky použití modulů je zajištěna jednoduchá rozšiřitelnost aplikace o nové funkce bez nutnosti změny jejího jádra. Serverová aplikace bude na modulech založena. Moduly budou umět zpracovávat klientské požadavky a rovněž jim bude umožněn přístup k deníkům jednotlivých uživatelů.
- **Uložení dat** - Jedním z dalších požadavků je uložení dat jednotlivých uživatelů. V rámci této bakalářské práce budou záznamy jednotlivých uživatelů uloženy v databázi SQLite3. Tato databáze je zvolena zejména pro svou jednoduchost a možnost používat ji i bez jakéhokoliv databázového serveru. Návrh aplikace však počítá s možností použití různých formátů pro uložení uživatelských dat.
- **Oddělení grafického rozhraní od komunikace se serverem** - Klientská aplikace využívající grafického rozhraní bude oddělena od nízkoúrovňové komunikace se serverem díky klientské knihovně. Klientská knihovna bude obsahovat základní funkce pro komunikaci se serverem a umožní použití libovolného grafického rozhraní bez zbytečné duplikace kódu.

Aplikace bude tedy složena z modulárního serveru, klientské knihovny a grafického uživatelského rozhraní. Toto rozdělení lze vidět i na obrázku 4.1. Serverová část bude fungovat nezávisle na klientské části. Obě části však budou moci bez výrazného dopadu na výkonnost běžet o lokálně na jednom počítači a výsledná aplikace tak bude použitelná i bez jakéhokoliv speciálního hardwaru.

V následujících podkapitolách jsou jednotlivé části aplikace stručně popsány.



Obrázek 4.1: Základní architektura aplikace

4.1 Návrh komunikačního protokolu

Komunikační protokol je důležitou součástí síťových aplikací. Určuje, jakým způsobem spolu komunikuje klientská a serverová část. Použití příliš jednoduchého a úzce zaměřeného protokolu by mohlo znemožnit budoucí rozšiřitelnost. Naopak použití zbytečně komplikovaného protokolu by znamenalo složitější řízení komunikace a náročnější implementaci.

Pro účel staničního deníku jsem se rozhodl použít vlastní protokol inspirovaný protokolem HTTP [9], převážně kvůli jeho jednoduchosti a účelnosti z hlediska modularity.

Jednotlivé moduly serveru budou mít přiřazeno své vlastní URI a klientské požadavky budou směrovány podle URI na konkrétní modul. Ten pak vygeneruje odpověď pro klienta. Komunikovat se tedy bude v režimu požadavek-odpověď.

Důležitá je i zpětně kompatibilita s protokolem HTTP. Díky této kompatibilitě bude možné serverovou aplikaci jednoduše napojit na ověřené technologie postavené na bázi HTTP jako je například AJAX [21]. Nezanedbatelnou výhodou takového protokolu je i dobrá prostupnost přes existující HTTP proxy a minimální omezení existujícími firewally.

Pro jednoduchost se pro přenos užitečných dat využije formátu CSV [18]. První řádek bude reprezentovat hlavičku dat. Názvy jednotlivých sloupců byly zvoleny následovně:

- **id** - ID záznamu v logu. Koresponduje s ID v databázi a slouží jako klíč pro vybrání konkrétního řádku.
- **user_id** - ID uživatele, jemuž záznam v logu patří.
- **qsodate** - Datum záznamu ve formátu unixového timestampu.
- **callsign** - Volací značka (například OK2JRQ).
- **mode** - Druh provozu.
- **qth** - QTH lokátor.

- **name** - Jméno operátora.
- **latitude** - Zeměpisná šířka.
- **longitude** - Zeměpisná délka.
- **county** - Okres.
- **continent** - Zkratka kontinentu.
- **cq** - CQ zóna.
- **itu** - ITU zóna.
- **rst_tx** - Odeslaný report v systému RST.
- **rst_rx** - Přijatý report v systému RST.
- **qsl** - Příznak žádosti o QSL lístek. Pokud je tento příznak nastaven, ví operátor, že si chce s protistranou vyměnit QSL lístky. To umožňuje případné automatické generování QSL lístků na základě informací uložených v databázi.
- **qsl_sent** - Příznak poslání QSL lístku. Pokud byly QSL lístky vygenerovány a odeslány, je tento příznak nastaven.
- **qsl_received** - Příznak přijetí QSL lístku. Tento příznak je nastaven v případě, kdy operátor obdržel QSL lístek od radioamatéra, se kterým bylo spojení navázáno.
- **comments** - Komentáře.

Jednotlivé moduly serveru však budou moci využít i své vlastní názvy sloupců pro položky, které nejsou v tomto seznamu.

4.1.1 Případy použití

V této podkapitole jsou uvedeny případy použití (use cases), které vyplynuly z analýzy funkce existujících deníků. Tyto případy musí navrhovaný protokol obsloužit. Pro uvedené případy použití budou implementovány samostatné moduly (viz podkapitola 4.2.1). Jedná se o následující případy použití:

- Registrace uživatele.
- Přihlásení uživatele - Přihlášení metodou Digest Access Authentication definovanou v RFC 2617 [20]. Díky tomu se nebude muset heslo posílat při přihlášení po sítí a na straně serveru bude možné jej ukládat pouze jeho otisk vytvořený funkcí MD5 [17]. To znemožní zjištění hesla odposlechem nebo jeho odcizení ze serveru.
- Vyžádání kompletního deníku nebo jeho částí.
- Vložení nového záznamu, jeho následná editace a případné odstranění.
- Zjištění detailních informací o majiteli volacího znaku.
- Ovládání radiostanice - změna a zjištění frekvence.
- Získání dat ze služby DXCluster.

4.2 Návrh serveru

Server je aplikace bez uživatelského rozhraní zpracovávající klientské požadavky. Administrátor musí být schopen konfigurovat základní nastavení serveru pomocí konfiguračního souboru. Jako formát tohoto konfiguračního souboru byl zvolen formát INI. Jedná se o jednoduchý formát umožňující rozdělit konfiguraci do kategorií a v každé definovat dvojice klíč=hodnota sloužící ke konfiguraci. Tento formát více než dostačuje navrhovanému použití, umožňuje konfiguraci serveru bez speciálních nástrojů a je běžně používaný ve většině operačních systémů. Příklad použití INI formátu lze vidět na následující ukázce konfigurace serveru:

```
[server]
hostname = 127.0.0.1
port = 8080

[database]
type = sqlite3
database = test.sql
```

Jak již bylo zmíněno, server bude schopen obsluhovat více uživatelů současně. Uživatelé se k serveru přihlásí pomocí jména a hesla. Noví uživatelé se budou muset nejprve registrovat.

Návrh serveru počítá s použitím libovolné databáze pro uchování perzistentních dat avšak v rámci této bakalářské práce je používa databáze SQLite3.

Server bude modulární a veškeré služby, které uživateli poskytne, budou součástí modulů. Server samotný bude pouze spravovat připojené uživatele a delegovat požadavky na jednotlivé moduly.

4.2.1 Moduly

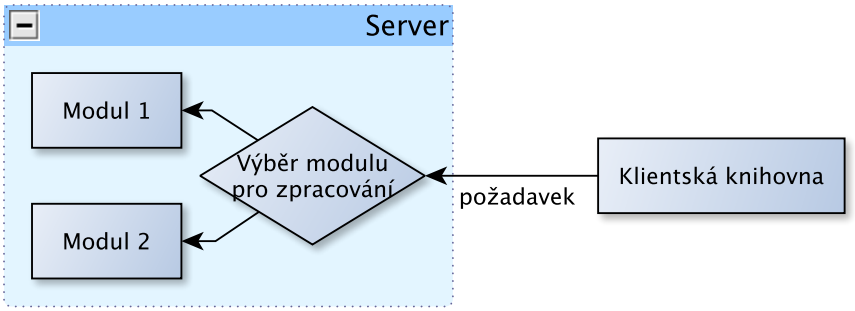
Moduly umožňují dynamicky rozšiřovat funkčnost serveru. Každý nový požadavek, který server od klienta přijme bude předán příslušnému modulu na základě URI (viz obrázek 4.2). Modul jej zpracuje a odešle klientovi zpět odpověď. Klient si může od serveru vyžádat seznam všech modulů pomocí požadavku na speciální URI „/modules“.

Modulům bude také umožněno přistupovat k databázi záznamů všech uživatelů a provádět nad ní dotazy. Tato problematika je více rozebrána v následující podkapitole 4.2.2.

Jednotlivé moduly serveru budou realizovány jako dynamické knihovny. Při spuštění serveru budou nahrány všechny moduly z adresáře nastavitelného pomocí konfiguračního souboru.

Každý modul bude obsahovat následující informace:

- **URI** - URI modulu, na kterém daný modul pracuje. Každý modul serveru musí mít své jedinečné URI.
- **Typ** - Typ modulu. Všechny moduly daného typu poskytují stejné komunikační rozhraní. To umožní klientské knihovně komunikovat i s pro ni neznámými moduly pouze na základě znalosti jejich typu.
- **Popis** - Krátký text popisující funkci modulu. V grafickém rozhraní bude tento text zobrazen v seznamu modulů.



Obrázek 4.2: Návrh využití modulů.

V rámci této bakalářské práce budou uvažovány moduly následujících typů (lze však snadno rozšířit):

- **CALLINFO** - Moduly tohoto typu budou poskytovat rozličné informace (například polohu nebo jméno operátora) na základě volací značky. Data lze získat např. z veřejně dostupných služeb, nebo z databáze uložené na serveru.
- **DXCLUSTER** - Moduly typu DXCLUSTER budou realizovat službu typu DXCluster, tedy budou poskytovat informace (například frekvenci a polohu) o aktuálně vysílajících stanicích.
- **UNKNOWN** - Tento typ bude použit pro moduly, které budou základní součástí serveru a nepředpokládá se, že by bylo potřeba mít na serveru více modulů se stejnou základní funkcí. Půjde například o moduly poskytující registraci a přihlášení uživatele, nebo například modul pro základní editaci deníku.

Rozhraní modulů typu CALLINFO

Pro moduly typu CALLINFO byl zvolen následující způsob komunikace:

- **Požadavek typu POST na URI „/modul“** - Jako data se v požadavku zašle volací značka operátora. V odpovědi modulu se pak vrátí veškeré zjistitelné informace o volací značce ve formátu CSV definovaném v podkapitole 4.1.
- **Požadavek typu GET na URI „/modul/username“** - Pokud bude modul vyžadovat registraci uživatelského jména a hesla pro přístup k databázi volacích značek (například v případě, že bude modul přistupovat ke službě, která je jen pro registrované uživatele), musí modul implementovat zpracování tohoto URI a vrátit aktuálně registrované uživatelské jméno nebo prázdný řetězec. Klient pak může tohoto chování využít pro zjištení, jestli je potřeba být pro použití modulu registrován.
- **Požadavek na URI „/modul/register“** - Využije se k registraci uživatelského jména a hesla pro použití modulu. Jméno a heslo se předá v hlavičce paketu pod klíči „username“ a „password“.

Rozhraní modulů typu DXCLUSTER

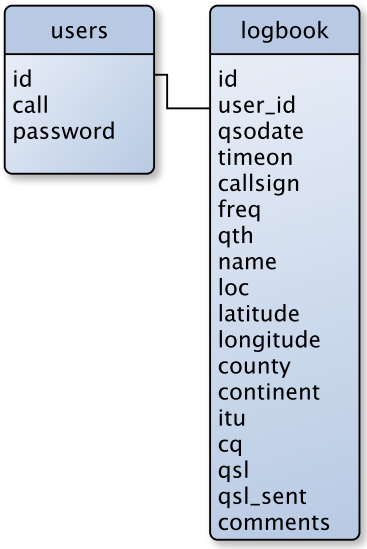
Pro moduly typu DXCLUSTER byl zvolen následující způsob komunikace:

- **Požadavek typu GET na URI „/modul“** - Modul vrátí seznam nově vysílajících stanic od posledního dotazu.
- **Požadavek typu GET na URI „/modul/username“** - Obdnobný význam jako u modulu typu CALLINFO (zjištění nutnosti registrace).
- **Požadavek na URI „/modul/register“** - Obdobný význam jako u modulu typu CALLINFO (registrace).

4.2.2 Databáze

Součástí serveru bude rozhraní pro přístup k databázi. Veškerá data všech uživatelů budou uložena v této databázi. Návrh serveru počítá s využitím databáze jakéhokoliv typu (SQLite3, MySQL, PostgreSQL, ...). Jednotlivé moduly musí být schopny s databází pracovat prostřednictvím databázového rozhraní a spouštět nad ní dotazy.

Na obrázku 4.3 je zobrazeno obecné schéma databáze jež by mělo být dostatečné pro uložení všech potřebných informací. Tabulky z tohoto obrázku jsou pak dále stručně popsány.



Obrázek 4.3: Návrh databáze.

Tabulka users

Tabulka users bude sloužit pro uložení informací potřebných pro přihlášení uživatele (deník může používat více uživatelů). Každý uživatel zde bude mít uloženo ID, volací značku a heslo.

Tabulka logbook

V tabulce logbook budou uloženy deníky všech uživatelů. Pro uložení dat byly zvoleny následující sloupce:

- **id** - ID záznamu.
- **user_id** - ID uživatele, jemuž záznam patří.
- **qsodate** - Datum záznamu ve formátu Unix timestamp.
- **callsign** - Volací značka (například OK2JRQ).
- **mode** - Druh provozu.
- **qth** - QTH lokátor (Maidenhead Locator).
- **name** - Jméno operátora.
- **latitude** - Zeměpisná šířka.
- **longitude** - Zeměpisná délka.
- **county** - Okres.
- **continent** - Zkratka kontinentu.
- **cq** - CQ zóna.
- **itu** - ITU zóna.
- **rst_tx** - Odeslaný report v systému RST.
- **rst_rx** - Přijatý report v systému RST.
- **qsl** - Příznak žádosti o QSL lístek. Pokud bude tento příznak nastaven, bude to značit, že si chce operátor s protistranou vyměnit QSL lístky. To umožní případný automatický tisk QSL lístků na základě informací uložených v databázi.
- **qsl_sent** - Příznak poslání QSL lístku. Pokud byly QSL lístky vygenerovány a odeslány, bude tento příznak nastaven.
- **qsl_received** - Příznak přijetí QSL lístku. Tento příznak bude nastaven v případě, kdy operátor obdržel QSL lístek od protistanice, se kterou navázal spojení.
- **comments** - Komentáře.

4.3 Klientská knihovna

Klientská knihovna poskytne grafickému rozhraní jednotné API pro přístup k serveru. To zamezí duplikaci kódu při případné implementaci více různých grafických rozhraní.

Požadavkem je, aby klientská knihovna měla minimální závislosti a byla multiplatformní. Knihovna bude pomocí komunikačního protokolu komunikovat se serverem, zpracovávat příchozí data a předávat je klientovi.

Knihovna je navržena tak, aby mohla spolupracovat s jakýmkoliv grafickým rozhraním (například Qt, GTK, wxWidgets, ncurses, ...).

Základní funkce klientské knihovny byly navrženy následovně:

- Připojení k serveru, registrace a přihlášení uživatele.
- Generování požadavků pro server a jejich předání serveru.
- Zpracování odpovědí serveru a předání zpracovaných dat klientské knihovně.

4.4 Klient

Klient bude uživateli umožňovat připojení k serveru, prezentaci aktuálních dat a jejich změny. Pro komunikaci se serverem klient využije klientskou knihovnu. Pro komunikaci s uživatelem pak klient využije grafického rozhraní. Po startu klienta bude uživatel vyzván k přihlášení se k serveru. Uživateli bude rovněž nabídnuta možnost registrace nového účtu.

Po přihlášení zobrazí klientská aplikace veškeré záznamy v deníku a umožní jejich editaci. Klientská aplikace také zobrazí aktuální vysílání získané ze služby DXCluster. Stanice, které aktuálně vysílají budou zobrazeny graficky na modelu zeměkoule.

Po vybrání stanice se zobrazí dialog pro přidání nového záznamu s předvolenými hodnotami získanými ze služby DXCluster. Dalším z požadavků je možnost automatického naladění radistanice na danou frekvenci.

V okně pro nový záznam se uživateli zobrazí historie všech spojení s daným operátorem. Po potvrzení dialogu se nový záznam odešle na server a uloží do databáze.

Klient také umožní zobrazit všechny moduly dostupné na serveru včetně jejich textového popisu.

Kapitola 5

Implementace

V této kapitole je popsána implementace serverové aplikace, klientské aplikace a klientské knihovny.

5.1 Serverová aplikace

Server byl implementován v jazyce C++ umožňujícím lepší dekompozici aplikace a použití objektově orientovaného přístupu. Byla rovněž použita knihovna Boost, která poskytuje základní metody pro asynchronní síťovou komunikaci a nabízí programátorské prostředky nad rámec standardní STL knihovny.

Pro implementaci modulu QRZ bylo potřeba použít knihovnu pro zpracování XML. K tomuto účelu byla použita knihovna TinyXML.

Dále budou stručně popsány nejdůležitejší třídy serveru.

Třída Server

Třída Server je základní třídou serveru. Vytváří soket, na kterém server přijímá připojení z klientské knihovny. Jakmile je akceptováno nové připojení, je vytvořena instance třídy Session, která dále zpracovává všechny požadavky klienta.

Třída Session

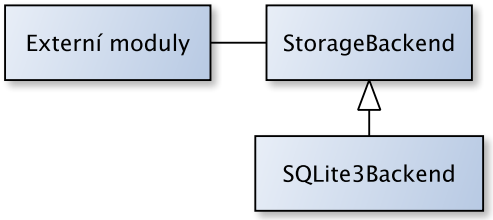
Tato třída reprezentuje sezení jednoho uživatele. Při obdržení nových dat od klienta jsou tato předána instanci třídy RequestParser, která slouží k jejich rozparsování. Pokud byla obdržena kompletní zpráva, je předána instanci třídy ModuleManager metodou handleRequest, která pak řídí její další zpracování. Výsledná odpověď je pak instancí třídy Session poslána zpět klientovi.

Třída RequestParser

Třída RequestParser reprezentuje konečný automat pro parsování zpráv podle specifikace komunikačního protokolu. Metoda parse zpracovává přijatá data, parsuje je, a výsledek uchovává v instanci třídy Request. Pokud dojde během parsování k chybě, vrací funkce parse hodnotu false a uživatel, jehož klient poslal neplatná data, je odpojen.

5.1.1 Databázové rozhraní

Návrh a implementace serveru umožňuje použití libovolného databázového rozhraní. V rámci bakalářské práce však byla implementována pouze podpora pro databázi SQLite3 (lze však snadno rozšířit). Třída implementující konkrétní databázové rozhraní musí dědit třídu StorageBackend a implementovat její čistě virtuální metody [15]. Vztah jednotlivých tříd implementujících a využívajících databázové rozhraní lze vidět na obrázku 5.1.



Obrázek 5.1: Diagram databázového rozhraní.

Třída StorageBackend

Třída StorageBackend je základní abstraktní třídou pro implementaci jakéhokoliv databázového rozhraní. Je u ní použit návrhový vzor Singleton. Jedná se o způsob návrhu třídy, který počítá s jedinou instancí dané třídy v rámci celé aplikace. Tato instance je pak přístupná ze všech částí aplikace. Typicky má třída používající návrhový vzor Singleton privátní konstruktor a statickou přístupovou metodu vracející instanci této třídy.

Smyslem třídy StorageBackend je poskytnout rozhraní pro získávání dat z databázového rozhraní bez znalosti typu databáze. Názvy metod a podtříd jsou inspirovány terminologií známou z oblasti SQL databází, ale prakticky lze pomocí třídy StorageBackend implementovat rozhraní pro přístup k jakémukoliv typu databáze.

Třída StorageBackend obsahuje základní podtřídy pro definici dotazů typu SELECT (používaný pro získání dat z databáze), INSERT (používaný pro vložení nových dat do databáza), UPDATE (sloužící k aktualizaci již uložených dat v databázi) a CREATE (umožňující vytvoření nové tabulky) známých z jazyka SQL:

- StorageBackend::Column - Slouží pro definici sloupce při vytváření nové tabulky metodou StorageBackend::createTable(). Obsahuje veškeré informace o sloupci tabulky:
  - Jméno - Jednoznačně pojmenovává sloupec. Jméno sloupce je pak použito pro výběr konkrétních sloupců v dalších typech dotazů.
  - Typ - Určuje typ sloupce. V současné implementaci existují tyto 4 typy: celé číslo, desetinné číslo, řetězec, datum a čas.
  - Velikost - Používá se pouze pro sloupce typu řetězec. Udává maximální velikost řetezce uloženého v tomto sloupci.
  - Příznak NOT NULL - Pokud je nastaven, musí být při přidání nebo editaci řádku v tabulce definována hodnota tohoto sloupce.
  - Příznak UNIQUE - Pokud je nastaven, musí být při přidání nebo editaci řádku tabulky hodnota tohoto sloupce jedinečná v rámci celého sloupce dané tabulky.

  - Příznak PRIMARY KEY - Pokud je nastaven, je tento sloupec primárním klíčem tabulky.

- StorageBackend::Select - Zapouzdřuje data potřebná pro provedení výběru dat z databáze nebo odstranění dat z databáze. Obsahuje jméno tabulky, ze které se výběr provádí, a ukazatel na dvourozměrné pole, do kterého se uloží případné výsledky. Dále třída StorageBackend::Select umožňuje definovat omezení výběru (v SQL jazyce klíčové slovo WHERE) a umožňuje výběr konkrétních sloupců, které vrátí ve výsledku. Instance této třídy je předána metodě StorageBackend::select() nebo StorageBackend::remove().

- StorageBackend::Insert - Obsahuje data pro vložení (v SQL jazyce klíčové slovo INSERT) nebo aktualizaci (v SQL jazyce klíčové slovo UPDATE) záznamu v databázi. Obsahuje název tabulky, ve které se budou data měnit, a samotná data ve formě název sloupce - hodnota. Umožňuje definovat omezení (v SQL jazyce klíčové slovo WHERE) aplikovaná při aktualizaci dat. Instance této třídy je předána metodě StorageBackend::insert() nebo StorageBackend::update().

V závislosti na implementaci metod třídy StorageBackend je pak v některé z jejich dceřiných tříd provedna konkrétní změna v databázi. Třída StorageBackend dále umožňuje získání identifikačního čísla naposledy vloženého záznamu metodou lastInsertedID().

Třída SQLite3Backend

Tato třída dědí třídu StorageBackend a implementuje její metody pro použití databázového systému SQLite3. V metodách update(), insert(), select(), createTable() a remove() se na základě předaných dat vygeneruje dotaz v SQL jazyce, ten se spustí a je vrácen výsledek.

5.1.2 Moduly

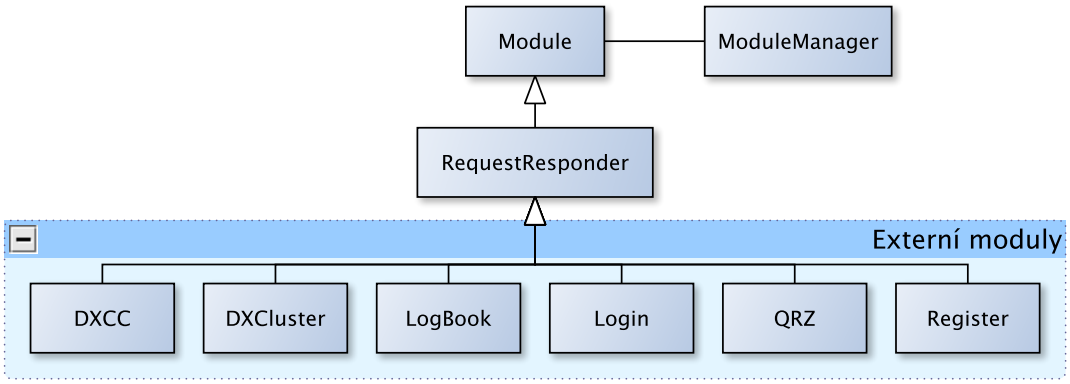
Moduly jsou implementovány jako dynamické knihovny. Každý implementovaný modul dědí třídu Modul (zprostředkovaně například přes třídu RequestResponder) a implementuje její čistě virtuální (pure virtual) metody ([15]). Toto demonstruje obrázek 5.2. Veškeré klientské požadavky jsou pak směrovány na konkrétní modul podle URI instancí třídy ModuleManager.

Třída Module

Třída Module poskytuje základní třídu, kterou musí implementovat každý externí modul. Obsahuje základní informace o modulu (jeho jméno, typ a popis).

Třída RequestResponder

Tato třída dědí třídu Module a rozšiřuje ji o data a metody specifické pro modul odpovídající na klientské požadavky. Přiřazuje modulu jeho URI a informaci o tom, jestli musí být uživatel pro jeho použití přihlášen. Obsahuje také deklaraci metody handleRequest(), která je volána instancí třídy ModuleManager pro každý příchozí požadavek směřující na modul.



Obrázek 5.2: Diagram tříd modulů.

Třída ModuleManager

U třídy ModuleManager je použit návrhový vzor Singleton. Tato třída zabezpečuje veškerou práci serveru s externími moduly. Pomocí metody loadModules lze načíst všechny moduly z adresáře zvoleného v konfiguračním souboru. Veškeré požadavky od klientů jsou předány instanci této třídy metodou handleRequest, která je pak dále směruje podle URI na konkrétní modul. Třída také umožňuje poslat seznam všech modulů klientské aplikaci.

5.1.3 Implementované moduly

V této podkapitole jsou popsány jednotlivé implementované moduly.

Modul Register

Modul Register (URI „/register“) slouží k registraci nových uživatelů. Při svém spuštění vytvoří pomocí instance třídy StorageBackend tabulku „users“. V metodě handleRequest pak přijímá případné požadavky na registraci uživatele a přidá nového uživatele do databáze. Pokud je již uživatel zaregistrován, vrací klientské aplikaci chybový kód.

Modul Login

Modul Login má URI „/login“. Jeho cílem je umožnit uživatelům přihlášení k systému. V metodě handleRequest je implementován princip přihlášení WWW-Authenticate definovaném v RFC 2617 [20].

Ukázka přihlášení uživatele

Tento příklad ukazuje použití komunikačního protokolu pro přihlášení uživatele.

Dotaz na modul poskytující URI „/login“:

```
GET /login HTTP/1.1
```

Odpověď serveru vybízí uživatele k přihlášení podle RFC 2617 [20]:

```
HTTP/1.0 401 Unauthorized
Content-Type: text/html
Content-Length: 14
WWW-Authenticate: Digest realm=,,realm@hamlog‘‘,qop=,,auth,auth-int‘‘,nonce=,,dcd98b710

Authentication
```

Klient se přihlásí s použitím správného jména a hesla, které je však přenášena zahashované:

```
GET /login HTTP/1.1
Authorization: Digest username=,,ok2jrq‘‘,realm=,,realm@hamlog‘‘,nonce=,,dcd98b7102dd2f
```

Server informuje klienta o úspěšném přihlášení:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 10

Authorized
```

Modul LogBook

Tento modul je základem celého projektu, protože umožňuje uživateli ukládat nové záznamy na server. Při svém načtení vytvoří tabulku „logbook“. V metodě handleRequest na základě URI provádí následující akce:

- URI „/logbook“ - Jako odpověd na dotaz pošle celý deník v CSV formátu získaný z databáze pomocí instance třídy StorageBackend.
- URI „/logbook/add“ - Přidá do tabulky „logbook“ nový záznam podle CSV dat přijatých v dotazu.
- URI „/logbook/remove“ - Odstraní z tabulky „logbook“ záznam definovaný pomocí ID přijatého v dotazu.
- URI „/logbook/call“ - Jako odpověď na dotaz pošle pouze záznamy o spojeních s konkrétním operátorem určeným jeho značkou v těle dotazu.

Ukázka vyžádání logu

Tento příklad ukazuje použití protokolu pro získání všech záznamů z logu.

Dotaz na modul poskytující URI „/logbook“:

```
GET /logbook HTTP/1.1
```

Odpověď serveru:

```
HTTP/1.1 200 OK
Content-Type: text/hamlog
Content-Length: 74

id;user_id;callsign;date;qth;loc
1;1;TEST;2011;qt;
2;1;LKS;2011;;location
```

Modul DXCC

Modul DXCC (URI „/dxcc“) umožňuje získat z volací značky lokalizační informace. Modul je typu CALLINFO. Data o jednotlivých prefixech jsou po startu modulu načtena ze souboru „cty.csv“ v CSV formátu. Tento soubor byl získán z [16]. V metodě handleRequest modul získá z požadavku prefix volací značky, vyhledá jej v datech načtených při startu a jako odpověď odešle informace o lokaci stanice. Pokud prefix není nalezen, vrací chybový kód.

Modul DXCluster

Modul DXCluster (URI „/dxcluster“) slouží k připojení k DXClusteru, implicitně se používá adresa dxspots.com. Modul je typu DXCLUSTER. Při prvním požadavku od klienta dojde k připojení na DXCluster. Veškerá data přijatá z DXClusteru jsou rozparsována a uložena v CSV formátu. Na každý další klientský požadavek odpoví modul daty získanými z DXClusteru. Jde tedy o jistou formu pollingu, kdy si klient opakovaně žádá o nová data.

Modul QRZ

Modul QRZ (URI „/qrz“) je typu CALLINFO. Umožňuje tedy získávat uživateli další informace o ostatních uživatelích na základě jejich volací značky. K tomuto využívá službu qrz.com. V metodě handleRequest se na základě URI provádí následující akce:

- URI „/qrz“ - Pošle QRZ serveru požadavek pro získání informací o uživateli na základě jeho volací značky. Ke komunikaci s QRZ serverem je využíváno XML API, které tento server poskytuje. Odpověď na požadavek je rozparsována pomocí knihovny TinyXML a odeslána klientské aplikaci ve formátu CSV.
- URI „/qrz/register“ - Umožňuje uživateli zvolení nebo změnu hesla použitého pro přihlášení k QRZ serveru.

Komunikace se serverem QRZ je rovněž asynchronní a odpověď na dotaz na QRZ modul není odeslána ihned. Je tak třeba řešit problém, kdy klientská aplikace pošle dotaz na QRZ modul následovaný dotazem na modul jiný. V tomto případě by mohl být druhý dotaz zpracován před prvním a došlo by tak k narušení způsobu komunikace dotaz-odpověď.

Řešením tohoto problému je možnost modulu zastavit dočasně zpracovávání dalších požadavků od konkrétního klienta, dokud nebude vyřízen požadavek aktuální. To lze provést zavoláním metody Reply::setAsync(). Jakmile je odpověď připravena k odeslání, lze ji odeslat metodou Session::sendAsyncReply(). Po zavolání této metody je pak opět povoleno zpracovávání dalších požadavků od klienta.

Modul Hamlib

Modul Hamlib (URI „/hamlib“) umožňuje ovládat radiostanici připojenou k počítači, na kterém běží serverová aplikace. Pro změnu a získání frekvence využívá knihovnu Hamlib (Ham Radio Control Libraries) [4]. Tato knihovna poskytuje jednotné rozhraní pro ovládání radiostanic od různých výrobců a typů. Implementuje nízkoúrovňovou komunikaci s radiostanicí a dovoluje programátorovi ovládat radiostanici jednoduchými příkazy. Modul byl prakticky otestován s radiostanicí ICOM IC-706MKIIG.

V metodě handleRequest se na základě URI provádí tyto akce:

- dotaz typu POST na URI „/hamlib“ - Přeladí frekvenci radiostanice na frekvenci získanou z dotazu. Frekvence je v dotazu formátována jako desetinné číslo (v kHz).
- dotaz typu POST na URI „/hamlib“ - Odešle zpět aktuální frekvenci jako desetinné číslo (v kHz).

5.1.4 Logování

Logování je implementováno s použitím knihovny Log4cxx vyvíjené Apache Software Foundation a licencované pod licencí Apache License [2]. Pokud však není při kompilaci knihovna Log4cxx nalezena, je pro logování použit standardní výstup. Výhodou použití Log4cxx je možnost široké konfigurace logování pomoci konfiguračního souboru, možnost přesměrovat logování do souboru a tento pak automaticky rotovat na základě jeho velikosti nebo času.

Každá třída serveru má vlastní statickou instanci třídy log4cxx::LoggerPtr, kterou využívá k logování. U každého zápisu do logu se loguje datum a čas, závažnost záznamu (Informace, varování, chyba), název modulu a samotný logovaný záznam. Standardní výstup pak vypadá například následovně:

```
2012-03-28 19:12:44,423 INFO  Server: Starting the server
```

5.2 Klientská knihovny

Klientská knihovna spojuje serverovou aplikaci se samotným klientským rozhraním. Klientská knihovna je navržena a implementována tak, aby ji bylo možno použít s jakýmkoliv grafickým (případně i konzolovým) rozhraním. Kvůli přenositelnosti a širší využitelnosti je napsána v jazyce C s důrazem na co nejméně závislostí na jiných knihovnách.

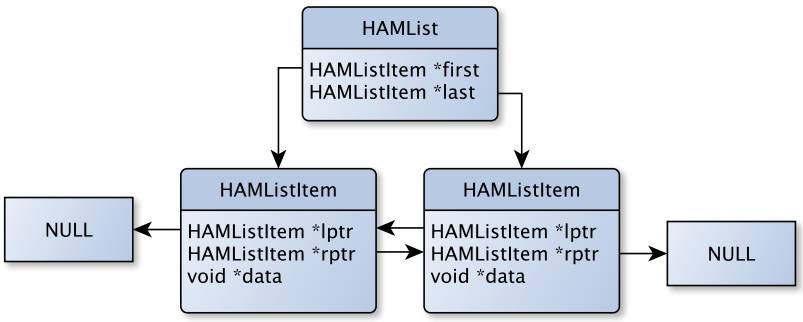
Klientská knihovna je rozdělena do menších bloků, které budou v této podkapitole postupně popsány.

5.2.1 Abstraktní datové typy

V této podkapitole je popsána implementace abstraktních datových typů požitých v klientské knihovně. S pomocí abstraktních datových typů lze zapouzdřit libovolná data a provádět nad nimi jednotné operace. To vede ke zjednodušení tvorby aplikace, protože je využíván již jednou naprogramovaný kód. Implementaci abstraktního datového typu lze také v budoucnosti změnit nezávisle na částech aplikace, které jej používají.

HAMList - Seznam

Prvním abstraktním datovým typem používaným v klientské knihovně je dousměrný seznam (HAMList). Slouží hlavně k uchování informací o odeslaných požadavcích na server za účelem správného zpracování odpovědí. Dále je tento datový typ použit například pro uložení rozparsovaných CSV dat obdržených v odpovědi.



Obrázek 5.3: Diagram dvousměrného seznamu.

HAMList je implementací dvousměrného seznamu. Základní datové struktury použité pro definici seznamu jsou HAMList a HAMListItem (viz obrázek 5.3):

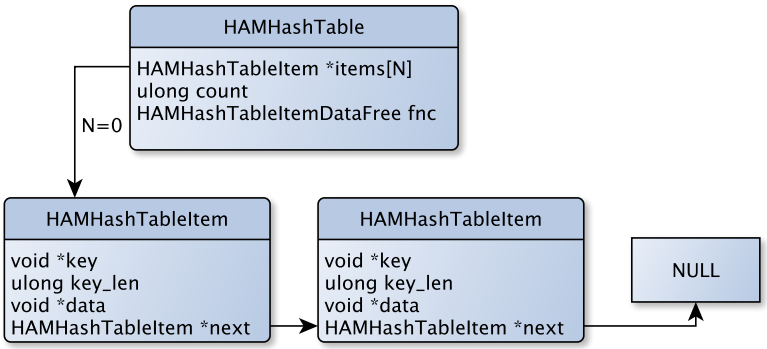
```
typedef struct _HAMListItem {
void *data;
struct _HAMListItem *lptr;
struct _HAMListItem *rptr;
} HAMListItem;

typedef struct _HAMList {
HAMListItem *first;
HAMListItem *last;
HAMListItemDataFree free_func;
} HAMList;
```

Každá položka HAMListItem obsahuje odkaz na svého předchůdce (lptr) a následníka (rptr) a samotná data spjatá s položkou (data). Struktura HAMList obsahuje odkaz na první a poslední položku a ukazatel na funkci free_func, která je použita pro uvolnění uživatelských dat z paměti. Pokud není tato funkce definována, nejsou uživatelská data při uvolňování seznamu z paměti uvolněna.

HAMHashTable - Hashovací tabulka

Druhým abstraktním datovým typem používaným v klientské knihovně je hashovací tabulky (HAMHashTable). Hlavním využitím hashovací tabulky je implementace signálů, kde slouží pro rychlé vyhledání správného signálu podle jeho jména. Je také použita pro uchování seznamu modulů dostupných na serveru.



Obrázek 5.4: Diagram hash tabulky seznamu.

HAMHashTable je implementací hash tabulky. Na obrázku 5.4 lze vidět základní datové struktury použité při implementaci hash tabulky - HAMHashTableItem a HAMHashTable:

```
typedef struct _HAMHashTableItem {
const void *key;
void *data;
unsigned long key_len;
struct _HAMHashTableItem *next;
} HAMHashTableItem;

typedef struct _HAMHashTable {
HAMHashTableItem *items[HAM_HASH_LEN];
unsigned long count;
HAMHashTableItemDataFree free_func;
} HAMHashTable;
```

Každá položka uložená v hash tabulce obsahuje svůj klíč (key), jeho délku (key_len), data svázaná s položkou a ukazatel na další položku. Při vložení nové položky do tabulky je vypočten hash jejího kliče pomocí SDBM hashovacího algoritmu [22]. Na základě hodnoty hashe je ukazatel na položku uložen do pole položek items.

5.2.2 Komunikace s klientskou aplikací

Pro komunikaci s klientskou aplikací je podstatné napojení na její smyčku událostí a možnost předávat asynchronně výsledky požadavků odeslaných serveru. V této podkapitole jsou popsány řešení obou těchto problémů

EventLoop

Eventloop (neboli smyčka událostí) sdružuje metody sloužící k napojení na hlavní smyčku klientské aplikace. Pro správnou funkci klientské knihovny musí klientská aplikace implementovat všechny funkce definované ve struktuře HAMEventLoopUICallbacks a předat je klientské knihovně prostřednictvím metody ham_eventloop_set_ui_callbacks().

Funkce definované ve struktuře HAMEventLoopUICallbacks jsou pak používány dalšími částmi klientské knihovny na následující činnosti:

- timeout_add - Přidá do hlavní smyčky aplikace běžící v klientské aplikaci nový časovač. Klientská aplikace musí vrátit ukazatel na strukturu jednoznačně identifikující časovač a volat opakovaně ve zvoleném intervalu funkci předanou jako ukazatel.
- timeout_remove - Odebere z hlavní smyčky časovač na základě ukazatele na strukturu, která jej identifikuje.
- input_add - Přidá do hlavní smyčky klientské aplikace ukazatel na funkci, která je volána když jsou k dispozici nová data na definovaném soketu. Klientská aplikace musí vrátit ukazatel na strukturu jednoznačně identifikující tuto událost.
- input_remove - Odebere z hlavní smyčky ukazatel na funkci vstupu na základě ukazatele na strukturu, která jej identifikuje.

Díky této abstrakci je tak možno napojit klientskou knihovnu na jakoukoliv smyčku událostí.

Signály

Jednotlivé části klientské knihovny umožňují definovat signály, na které se pak může klientská aplikace napojit. Seznam signálů je uložen v hash tabulce, kde klíčem je název signálu a daty seznam funkcí, které jsou zavolány pokud je signál emitován. K registraci nových signálů slouží funkce ham_signals_register_signal().

Klientské aplikaci je umožněno funkcí ham_signals_register_handler() zaregistrovat funkci, která je zavolána při emitování signálu. Funkce musí být ve formátu HAMFetchHandler. Při registraci signálu lze rovněž definovat ukazatel na data, která jsou při emitování signálu zpracovávající funkci předána. Toho lze využít pro udržování kontextu při zpracovávání signálu.

5.2.3 Komunikace se serverem

Tato podkapitola popisuje implementaci komunikace se serverem v klientské knihovně. Je zde popsáno rozhraní pro připojení k serveru, parser komunikačního protokolu a pomocné struktury HAMRequest a HAMReply pro reprezetanci odchozích a příchozích paketů.

Připojení k serveru

Pro připojení k serveru je nutné vytvořit novou instanci struktury Connection funkcí ham_connection_new(). Touto funkcí se definuje adresa a port serveru, uživatelské jméno a heslo. Samotné připojení proběhne až po zavolání funkce ham_connection_connect(). Tato funkce vytvoří nový soket pro připojení k serveru a pomocí funkce ham_eventloop_input_add() přidá do hlavní smyčky aplikace ukazatel na funkci pro parsování přijatých dat.

Odeslání požadavku

Požadavek je reprezentován strukturou HAMRequest. Její instanci lze vytvořit metodou ham_request_new(). Lze určit URI, na které bude požadavek zaslán, typ požadavku (GET nebo POST) a zasílaná data. Samotný požadavek pak lze odeslat metodou ham_connection_send() (případně ham_connection_send_destroy(), která jej i uvolní z paměti).

Pro použití standardních modulů jsou však v klientské knihovně implementovány jednoduché metody, které vytváření a odesílání požadavků provádějí automaticky. Rovněž jsou definovány signály pro zpracování odpovědí na tyto standardní dotazy. V praxi tak stačí aplikaci využívající klientskou knihovnu zavolat například metodu ham_dxcc_fetch(), která odešle potřebný požadavek serveru a při přijetí odpovědi automaticky emituje signál, na který se může klienstká aplikace napojit.

Parsování odpovědi

K parsování dat v klientské knihovně slouží HAMParser. Ten funguje na stejném principu jako RequestParser na straně serveru. Parsovaná data jsou ukládána do struktury HAMReply, která pak předána prostřednictvím signálu nebo zpětného volání (callbacku) funkci, která požadavek na server iniciovala.

5.3 Klientská aplikace

Referenční klientská aplikace byla naprogramována v jazyce C++ s využitím grafického frameworku Qt [6]. V této kapitole jsou stručně popsány jednotlivé části klientské aplikace, třídy, které klientskou aplikaci implementují a jejich napojení na klientskou knihovnu.

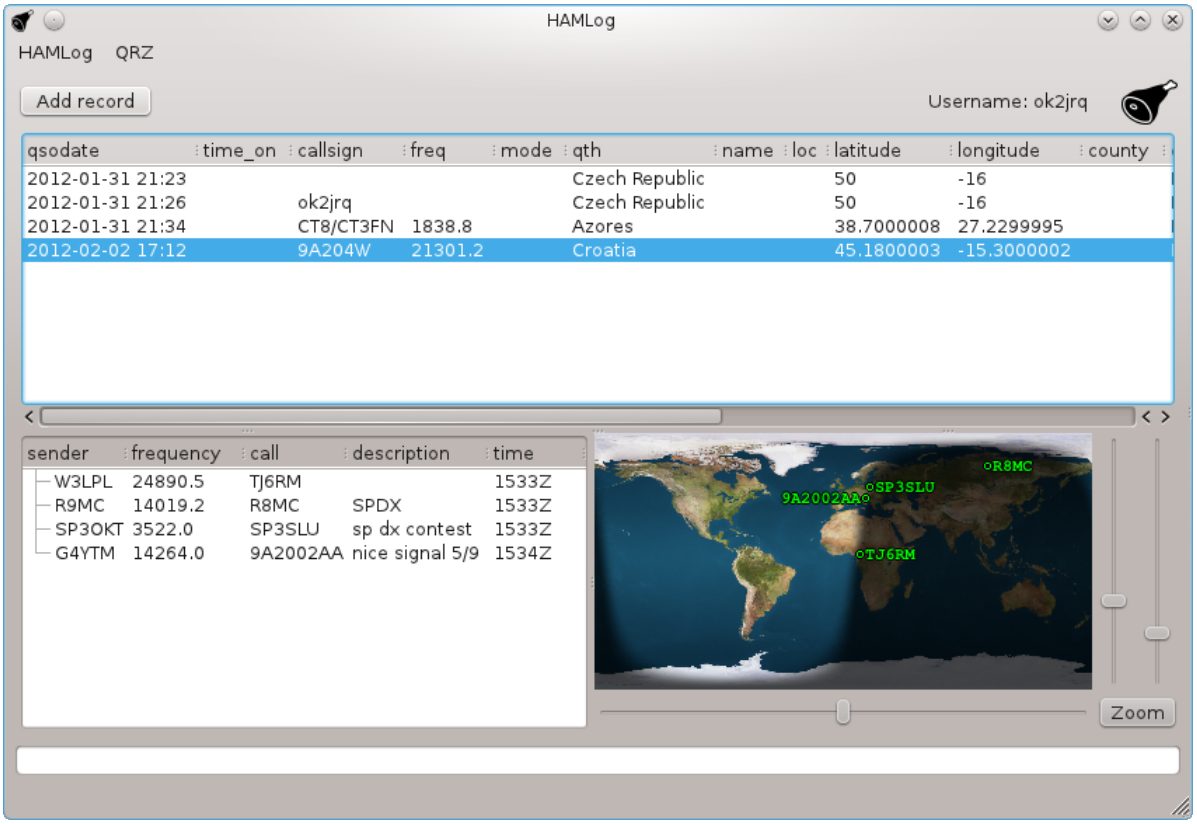
5.3.1 Hlavní okno aplikace

V této podkapitole je popsána implementace hlavního okno aplikace (znázorněno na obrázku 5.5) a všech prvků, které jej tvoří.

Třída MainWindow

Třída MainWindow reprezentující hlavní okno je také hlavní třídou klientské aplikace. Po svém vytvoření zobrazí dialog pro připojení uživatele k serveru. Pomocí metody connectServer() se pak přihlásí k serveru. Nový uživatelský účet se registruje metodou registerAccount.

Třída MainWindow se o úspěšném, případně neúspěšném přihlášení k serveru dozví díky napojení na patřičné signály klientské knihovny. V této třídě je také implementována komunikace mezi jednotlivými prvky tvořícími hlavní okno prostřednictvím Qt signálů.



Obrázek 5.5: Hlavní okno aplikace.

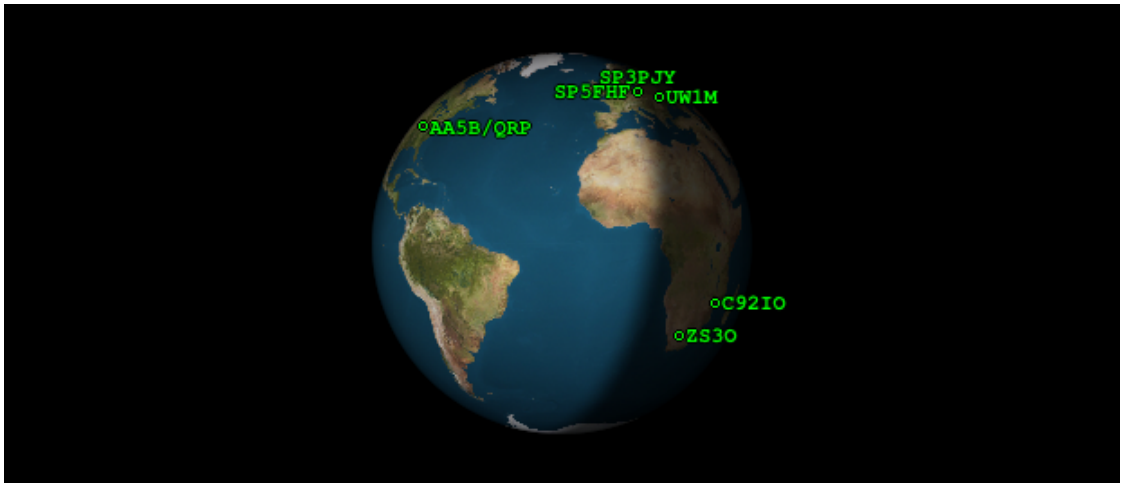
Třída LogbookTreeWidget

Třídou LogbookTreeWidget je uživateli umožněno zobrazení a editace logu prostřednictvím rozhraní podobného tabulce. Je implementována nad Qt třídou QTreeWidget a komunikuje se serverovým modulem Logbook prostřednictvím klientské knihovny. Jednotlivé položky jsou editovatelné a změny se přenášejí na server. Pokud uživatel poklepe na některou z položek, je zpracován příslušný Qt signál a otevřen dialog NewRecordDialog, ve kterém může uživatel položku také editovat.

Třída EarthWidget

Třída EarthWidget je technologicky nejzajímavější třídou klientské aplikace. Umožňuje zobrazit zeměkouli promítnutou do roviny i formou klasického globusu (obrázek 5.6). Na zeměkouli lze pak zobrazovat značky s popisky. Toho je využito pro zobrazení polohy právě vysílajících stanic.

Pro zobrazení zeměkoule je využita aplikace Xplanet. Při požadavku na překreslení okna je spuštěn nový proces Xplanet s požadovanými parametry a je mu předáno ID widgetu, do kterého má vykreslovat. Značky s popisky jsou uloženy do dočasného souboru a tento soubor je pak načten aplikací Xplanet a použit k vykreslení značek. EarthWidget umožňuje také přiblížení na předdefinované lokace (například přiblížení Evropy) pomocí kontextového menu.



Obrázek 5.6: Ukázka možnosti zobrazení zeměkoule.

Třída DXClusterWidget

Pomocí třídy DXClusterWidget je zobrazen seznam aktuálně vysílajících stanic. Třída se prostřednictvím klientské knihovny v pravidelných intervalech dotazuje serveru na seznam aktuálně vysílajících stanic a tento pak zobrazuje. Pokud uživatel na některou ze stanic poklepe, otevře se dialog pro přidání nového záznamu. Seznam stanic je také předáván za pomoci Qt signálu třídě EarthWidget, která pak stanice vykresluje na mapě.

5.3.2 Přidání nového záznamu a editace

Nový záznam lze přidat dvěma způsoby. Buď tak lze učinit pomocí tlačítka „Add Record“ v hlavním okně aplikace, nebo poklepáním na stanici získanou z DXClusteru. Obě možnosti pak vedou k vyvolání dialogu pro přidání nového záznamu (zobrazen na obrázku 5.7). Stejný dialog je použit také pro editaci již existujícího záznamu. Editaci lze vyvolat poklepáním na existující záznam v hlavním okně aplikace.

Třída NewRecordDialog

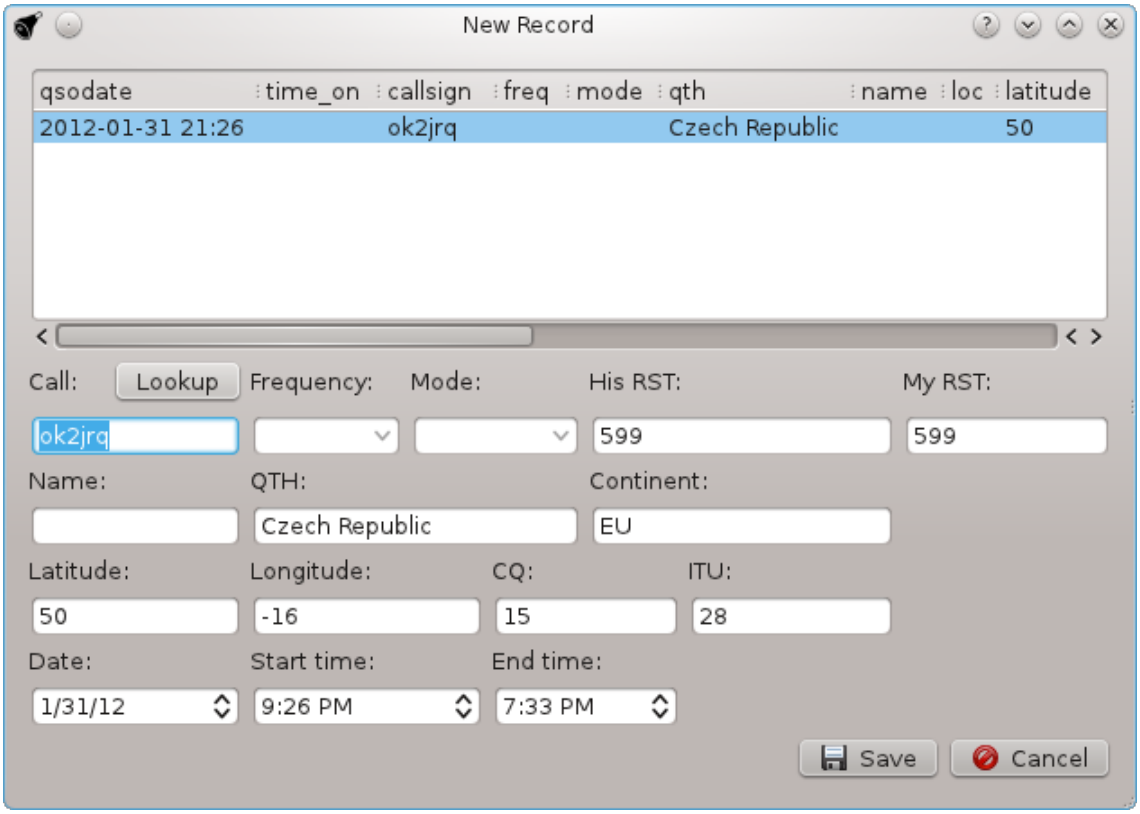
Dialog pro přidání nového záznamu a jeho editaci je implementován v třídě NewRecordDialog. Dialog se skládá ze dvou částí:

- Tabulka zobrazující předchozí spojení s přidávanou stanicí.
- Formulář s jednotlivými poli o nově přidávaném záznamu.

Pro implementaci tabulky s předchozími spojeními je použita třída LogbookTreeWidget. Je tak znovu použit již jednou vytvořený kód.

V případě změny pole pro zadání volací značky (Call sign), je aktualizována tabulka s historií, a uživatel tak vidí předešlá spojení s danou stanicí. Také se pomocí klientské knihovny zjistí podrobné informace o zadané stanici a předvyplní se patřičná pole. To zrychluje zadávání nových záznamu. V případě změny hodnoty v poli frekvence je na server odeslán požadavek o přeladění radiostanice na zadanou frekvenci.

Po potvrzení dialogu jsou hodnoty všech polí prostřednictvím klientské knihovny přeneseny na server, kde jsou uchovány.



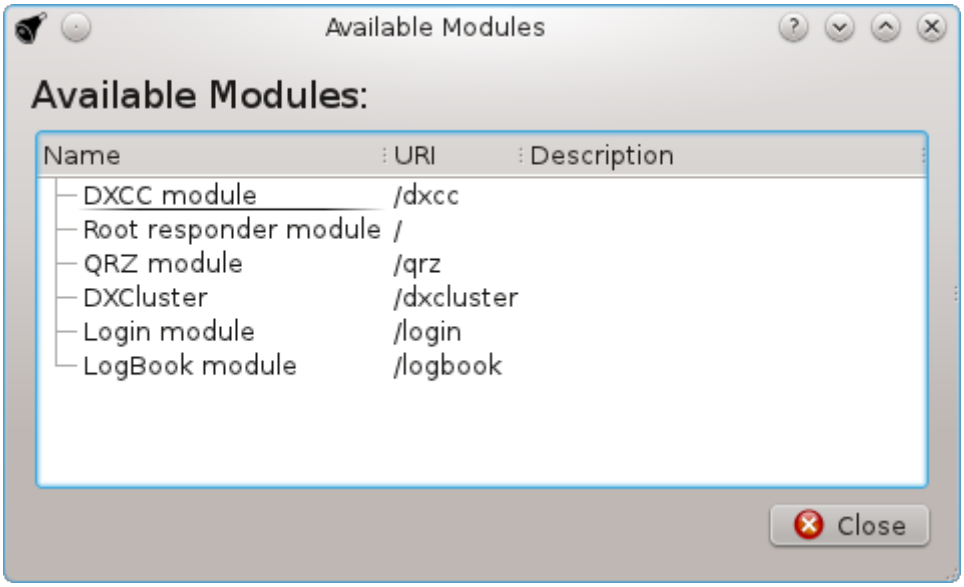
Obrázek 5.7: Dialog pro přidání a editaci záznamu.

5.3.3 Zobrazení dostupných modulů

Dialog zobrazující moduly dostupné na serveru (viz obrázek 5.8) lze spustit prostřednictvím hlavního menu aplikace.

Třída NewRecordDialog

Dialog zobrazující dostupné moduly je implementován třídou NewRecordDialog. Jedná se o jednoduchou třídu, která pomocí klientské knihovny získá ze serveru seznam modulů a zobrazí je formou tabulky vytvořené díky Qt widget QTreeWidget.



Obrázek 5.8: Dialog zobrazující dostupné moduly.

Kapitola 6

Ukázka použití aplikace

V této kapitole je stručně popsáno typické použití aplikace na jednoduchém příkladu.

Po přihlášení k serveru vidí operátor radiostanice hlavní okno aplikace (obrázek 5.5). V tomto hlavním okně se po chvíli začnou v jeho dolní části zobrazovat ostatní radioamatéři, kteří právě vysílají (DX Cluster). Operátor má možnost vidět na mapě světa umístění těchto stanic. Pro získání většího detailu lze mapu světa přiblížit na požadovanou oblast. Může také změnit zobrazení a zobrazit klasický globus (obrázek 5.6).

Pokud se operátor rozhodne začít komunikaci s některým s dalších vysílajících radioamatérů, stačí poklepat na jeho volací značku v seznamu a dojde k otevření dialogového okna pro navázaní nového spojení (obrázek 5.7). Otevřením dialogového okna se pomocí serveru QRZ.COM a DXCC databáze získají další detailní informace o radioamatérovi, se kterým se operátor snaží navázat spojení. V tento moment se také automaticky přeladí radiostanice připojená k jeho počítači a on může začít zkoušet navázat spojení.

V tomto okně lze také vidět předešlá spojení s tímto radioamatérem a operátor se tak může rozhodnout, zda-li chce navázat spojení s někým, s kým už se v minulosti spojil.

Během komunikace vyplní operátor další požadované údaje jako například přijatý a odeslaný RST report. Jakmile je komunikace u konce, uzavře operátor dialogové okno pro přidání nového záznamu a spojení se zapíše do deníku.

Pokud se operátor v budoucnu rozhodne záznam o spojení upravit (například za účelem změny příznaku o přijatém QSL lístku), lze jednoduše poklepat na spojení a editovat ho ve stejném dialogu jako při jeho přidání.

Kapitola 7

Závěr

Cílem této bakalářské práce bylo navhrnout a implementovat softwarovou aplikaci pro vedení staničního deníku určeného pro operátory amatérské radiokomunikační služby. Důraz byl kladen zejména na modularitu a otevřenost návrhu. Pro úspěšné dokončení práce bylo nutné se seznámit se základními principy, aplikacemi a terminologií používanou radiamatéry. Bylo nutné se zorientovat v radioamatérské technologii a pochopit fungování celého radioamatérského společenství. Výsledný návrh se snaží odstranit problémy jimiž trpí v současné době dostupné aplikace a zaplnit volné místo mezi aplikacemi umožňujícími vedení staničního deníku.

Výstupem implementace jsou tři oddělené projekty (serverová aplikace, klientská knihovna a grafické uživatelské rozhraní), které spolu spolupracují a vzájemně se doplňují.

Serverová aplikace je plně modulární a umožňuje jednoduché přidávání nových funkcí pomocí modulů. Splňuje veškeré požadavky na základní vedení staničního deníku (přidávání/editace záznamů, vyhledávání informací o ostatních radioamatérech nebo například ovládání vysílačky připojené k počítači).

Pokud serverová aplikace poběží na skutečném serveru, je díky systému modulů možné např. jednoduše vytvořit modul pro generování deníku na webové stránky. Případně, díky využití protokolu kompatibilního s protokolem HTTP, použít rovnou serverovou aplikaci jako jednoduchý webový server.

Klientská knihovna je důležitým prvkem implementace, protože implementuje nízkoúrovňovou komunikaci s serverovou aplikací a umožňuje tak grafickému uživatelskému rozhraní zaměřit se pouze na komunikaci s uživatelem. Klientskou knihovnu je možné použít pro jednodušší tvorbu klientských aplikací a v budoucnu se předpokládá vznik dalších specializovaných klientských aplikací například pro zapisování spojení z terénu prostřednicvím mobilního telefonu.

Grafické uživatelské není přiliš rozsáhlé, ale umožňuje vést staniční deník pohodlně a poskytuje všechny základní funkce potřebné pro vedení staničního deníku. Je graficky přehledné a pro stávající radioamatéry intuitivní.

Celkově je výsledná aplikace dobrým základem pro implementaci dalších nadstandardních funkcí. Byla uvolněna pod licencí GPL a je k dispozici na webu Githubu. Během vývoje byla aplikace konzultována se skutečnými radioamatéry, kteří rovněž přislíbili účast na dalším vývoji. V budoucnu je pak v plánu integrace aplikace do nejpoužívanějších linuxových distribucí (balíčky RPM, DEB, ebuild, ...).

7.1 Možnosti budoucího rozšíření

Během implementace a testování aplikace vyplynuly některé další možnosti, jakým směrem aplikaci v budoucnu rozšiřovat.

7.1.1 Podpora MySQL

Pro standardní použití jedním uživatelem je databázový systéme SQLite3 zcela jistě dostačující. Pro větší nasazení s více uživateli využívajícími funkce serveru současně by však bylo vhodné implementovat podporu pro databázový systém MySQL. Přidat podporu pro jinou databázi je díky současnému návrhnu jednoduché.

7.1.2 Moduly pro tvorbu statistik

Dalším možným rozšířením je implementace modulů umožňujících generování statistik a přehledů o jednotlivých záznamech v deníku. Uživateli tyto moduly umožní lepší orientaci v jeho deníku a na straně serveru je pak možné generovat žebříček uživatelů na základě různých klíčů, snadno zobrazovat bodu získáné v soutěžích v reálném čase nebo například generovat diplomy (DXCC, IOTA, SOTA).

Díky zvolenému návrhu a implementaci databázového rozhraní lze konstruovat kompletní dotazy pro databázi a lze tedy získat data pro nepřeberné množství statistik. Navíc je díky implementaci modulů serveru jednoduché tyto nové funkce přidávat, protože nikdy nejsou součástí jádra aplikace, ale pouze toto jádro rozšiřují.

Literatura

[1] VYHLÁŠKA č. 156/2005 Sb. ze dne 19. dubna 2005, o technických a provozních podmínkách amatérské radiokomunikační služby. [Online], Poslední modifikace: 19. 4. 2005, [cit. 2012-04-24].
URL http://aplikace.mvcr.cz/archiv2008/micr/files/205/provadeci_p.pdf

[2] Apache Software Foundation: Short introduction to Apache log4cxx. [Online], Poslední modifikace: 2008, [cit. 2012-04-29].
URL http://logging.apache.org/log4cxx/

[3] ARRL: DXCC Challenge. [Online], Poslední modifikace: 20. 4. 2012, [cit. 2012-04-25].
URL http://www.arrl.org/dxcc-challenge

[4] Bargmann, N.: Ham Radio Control Libraries. [Online], Poslední modifikace: 9. 10. 2011, [cit. 2012-04-21].
URL http://www.hamlib.org/

[5] Biddle, A.: The Radio Amateur Satellite Corporation. [Online], Poslední modifikace: 1. 4. 2012, [cit. 2012-04-24].
URL http://www.amsat.org/amsat-new/index.php

[6] Blanchette, J.: *C++ GUI Programming with Qt 4*. Upper Saddle River: Prentice-Hall, druhé vydání, 2008, ISBN 0-13-187249-4, 718 s.

[7] Buthod, B.: AX.25 Amateur Packet-Radio Link-Layer Protocol. [Online], Poslední modifikace: 27. 11. 1997, [cit. 2012-04-24].
URL http://www.tapr.org/pub_ax25.html

[8] EI8IC: Ham Radio Maps. [Online], Poslední modifikace: 18. 4. 2012, [cit. 2012-04-18].
URL http://www.mapability.com/ei8ic/maps/maps.php

[9] Fielding, R.: RFC 2616: Hypertext Transfer Protocol – HTTP/1.1. 1997, [Online], Poslední modifikace: 11. 6. 1999, [cit. 2012-04-15].
URL http://www.ietf.org/rfc/rfc2616.txt

[10] IARU: Kmitočtový plán KV pásem IARU REGION 1. [Online], Poslední modifikace: 2001, [cit. 2012-04-25].
URL http://www.crk.cz/CZ/BPLANVKVC

[11] KADLČÁK, J.; PROSTECKÝ, M.: *Požadavky ke zkouškám operátorů amatérských rádiových stanic*. Vsetín: Český Radioklub, Čtvrté vydání, 1996, ISBN 80-902161-0-2, 170–225 s.

[12] OK1XU: Systém RST. [Online], Poslední modifikace: 18. 2. 2009, [cit. 2012-04-16].
URL http://www.crk.cz/CZ/RSTC

[13] OK1XU: Co je HAM Radio? 1997, [Online], Poslední modifikace: 18. 2. 2009, [cit. 2012-04-16].
URL http://www.crk.cz/CZ/CJHRC

[14] OK1XU: Jak žádat o koncesi. 1998, [Online], Poslední modifikace: 18. 2. 2009, [cit. 2012-04-16].
URL http://www.crk.cz/CZ/KONCEC

[15] POKORNÝ, P.: *Objektově orientované programování v C++*. Zlín: Univerzita Tomáše Bati ve Zlíně, třetí vydání, 2010, ISBN 80-7318-913-6, 92 s.

[16] Reisert, J.: Contest Country Files. [Online], Poslední modifikace: 23. 3. 2012, [cit. 2012-04-25].
URL http://www.country-files.com/cty/

[17] Rivest, R.: The MD5 Message-Digest Algorithm. [Online], Poslední modifikace: 1992, [cit. 2012-04-29].
URL http://www.ietf.org/rfc/rfc1321.txt

[18] Shafranovich, Y.: RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files. 2005, [Online], Poslední modifikace: 1. 8. 2005, [cit. 2012-04-15].
URL http://www.ietf.org/rfc/rfc4180.txt

[19] Synergenics: Introducing EchoLink. [Online], Poslední modifikace: 20. 4. 2012, [cit. 2012-04-24].
URL http://echolink.org/

[20] W3C/MIT: RFC 2617: HTTP Authentication: Basic and Digest Access Authentication. Červen 1999, [Online], Poslední modifikace: 10. 6. 1999, [cit. 2012-04-16].
URL http://www.ietf.org/rfc/rfc2617.txt

[21] Wikipedia: Ajax (programming) — Wikipedia, The Free Encyclopedia. 2012, [Online], Poslední modifikace: 20. 4. 2012, [cit. 2012-04-29].
URL http://en.wikipedia.org/w/index.php?title=Ajax_(programming)&oldid=489545757

[22] Yigit, O.: Hash Functions. [Online], Poslední modifikace: 22. 3. 2003, [cit. 2012-04-29].
URL http://www.cse.yorku.ca/~oz/hash.html

Dodatek A

Obsah CD

- **./doc/** – Programová dokumentace ke klientské knihovně (Doxygen).
- **./latex/** – Zdrovojé kódy technické zprávy.
- **./src/server/** – Zdrovojé kódy serverové aplikace.
- **./src/library/** – Zdrovojé kódy klientské knihovny.
- **./src/ui/qt/** – Zdrovojé kódy klientské aplikace.
- **./zprava.pdf** – Tato technická zpráva.
- **./readme.txt** – Návod na zprovoznění a otestování aplikace.