Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Simulátor distribuované anténní řady

Jan Kozák

Vedoucí práce: Ing. Viktor Černý

12. května 2015

Poděkování

Chtěl bych poděkovat Ing. Viktoru Černému za trpělivost a ochotu při vedení této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 12. května 2015 .....

Odkaz na tuto práci

Kozák, Jan. *Simulátor distribuované anténní řady.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Cílem této práce je návrh a implementace simulátoru distribuované anténní řady.

Klíčová slova distribuovaná anténní řada, simulátor, vyzařovací charakteristika, C++, Qt framework

Abstract

The aim of this thesis is to design and implement a distributed antenna array simulator.

Keywords distributed phase-shift beamforming, simulator, radiation pattern, C++, Qt framework

Obsah

Seznam obrázků

Úvod

Se vzrůstajícím počtem zařízení bezdrátově připojených k Internetu vzniká stále větší potřeba optimalizace provozu bezdrátových sítí. Je třeba nejen pokrýt co největší oblast, vznikají také požadavky na co nejnižší úroveň rušení, které negativně působí na propustnost bezdrátové sítě, a na co nejnižší energetickou náročnost. Distribuovaná anténní řada je jedním z konceptů, který může tyto požadavky splnit. Důležitou vlastností distribuované anténní řady je její vyzařovací charakteristika. Právě z ní je možné vyčíst, jak se signály jednotlivých transceiverů navzájem ovlivňují, kde vznikají hluchá místa a která místa jsou naopak pokryta silnějším signálem. Lze tak jednoduše monitorovat efektivitu algoritmů pracujících nad distribuovanou anténní řadou. K vykreslovaní vyzařovací charakteristiky se běžně používá matematický software jako Wolfram Mathematica či MATLAB, proces vizualizace tedy zatím není automatizován.

Mým úkolem je tedy vytvořit aplikaci s grafickým uživatelským rozhraním, která automatizuje proces vykreslovaní vyzařovací charakteristiky. Vstupem pro aplikaci budou parametry jednotlivých transceiverů, jejím výstupem bude vyzařovací charakteristika výsledné distribuované anténní řady.

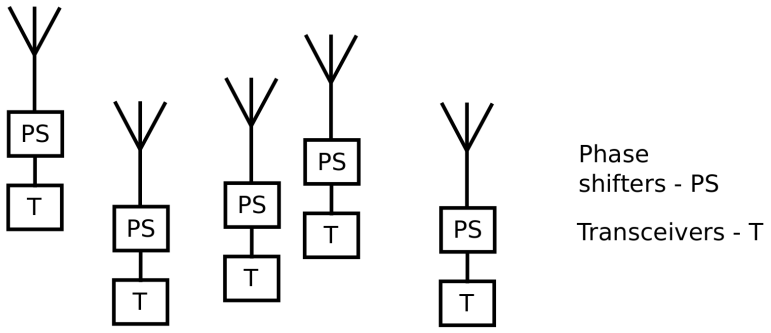
V první kapitole přibližuji koncept distribuovaných anténních řad a uvádím vzorce, pomocí kterých je možné zobrazit vyzařovací charakteristiku. V druhé kapitole se zabývám analýzou požadavků na aplikaci, vybírám programovací jazyk, framework a architekturu aplikace. Ve třetí kapitole pak popisuji implementaci, tato kapitola zároveň slouží jako dokumentace aplikace. Ve čtvrté kapitole pak aplikaci testuji.

Kapitola 1

Distribuovaná anténní řada

Jedním z největších problémů současných bezdrátových sítí je elektromagnetická interference. Interference primárně ovlivňuje spotřebu a propustnost sítě, sekundárně pak QoS v dané síti [1]. Vzniká proto řada technik, které se snaží nežádoucí účinky interference odstranit.

Metoda, na které je postavena myšlenka distribuované anténní řady, se nazývá *distributed phase shift beamforming*. Na rozdíl od jiných metod se nesnaží pouze snížit míru interference v celé síti – dokáže s ní naopak pracovat tak, aby pokryla pouze oblast, ve které se nacházejí přijímače, s použitím minimálního množství energie. Hlavní myšlenka této metody tkví v použití dvou a více antén připojených k jednomu transceiveru, které přenášejí stejný signál. Signály jednotlivých antén jsou navzájem fázově posunuty tak, aby v místech, která mají být pokryta, docházelo ke konstruktivní interferenci, případně aby docházelo k destruktivní interferenci tam, kde je vysílání nežádoucí.



Obrázek 1.1: Schéma distribuované anténní řady

Jak je vidět z obrázku 1.1, distribuovaná anténní řada je tvořena nezávislými moduly, které se skládají z jednoho transceiveru a jedné antény. Na rozdíl od předchozího případu už není poloha jednotlivých antén známá. Před

začátkem vysílání je tedy nutno najít takovou kombinaci fázových posunů, aby byly pokryty všechny přijímače. Tímto tématem se dále zabývá [2].

1.1 Vyzařovací charakteristika

Vyzařovací charakteristika je definována jako matematická funkce či grafická reprezentace vyzařovacích vlastností antény jako funkce prostorových souřadnic [3]. V našem případě pracujeme s vyzařovací charakteristikou, která zobrazuje zisk distribuované anténní řady v jednotlivých směrech. Zobrazení vyzařovací charakteristiky je vhodné, je z ní totiž možné vyčíst, kde se signály jednotlivých antén skládají a kde naopak dochází k rušení.

$P_T \ldots$ vysílací výkon transceiveru $[dBm]$
$P_R \ldots$ výkon přijatý přijímačem $[dBm]$
$\lambda \ldots$ vlnová délka $[m]$
$d \ldots$ vzdálenost mezi vysílačem a přijímačem $[m]$
$n \ldots$ počet transceiverů

$$P_R = P_T + 20 \log_{10} \left(\frac{\lambda}{4\pi d} \right) \tag{1.1}$$

$$\hat{P}_R = P_R \left(\cos \varphi + \sin \varphi \right) \tag{1.2}$$

$$\varphi = \varphi_{init} + \frac{360 \left(d \% \lambda \right)}{\lambda} \tag{1.3}$$

Celkový výkon všech vysílačů přijatý přijímačem lze vyjádřit jako:

$$P_{R-total} = abs \left(\sum_{i=1}^{n} P_{R_i} \right) \tag{1.4}$$

Výkon přijatý ideálním přijímačem:

$$P_{R-ideal} = P_{T-ideal} + 20 \log_{10} \left(\frac{\lambda}{4\pi d} \right) \tag{1.5}$$

$$P_{T-ideal} = \sum_{i=1}^{n} P_{T_i} \tag{1.6}$$

Pokud známe $P_{R-total}$ a $P_{R-ideal}$, můžeme zisk přijímače vyjádřit takto:

$$G = P_{R-total} - P_{R-ideal} \tag{1.7}$$

Výpočet vyzařovací charakteristiky v simulátoru bude realizován umístěním přijímačů na kružnici se středem v prostředním bodě všech transceiverů a výpočtem zisku každého takového přijímače.

1.2 Pokrytí přijímače

Další funkcí simulátoru, pro kterou je potřeba připravit teoretický základ, je zobrazení pokrytí přijímačů. Pro každý transceiver bude pomocí rovnic 1.1 a 1.2 vypočítáno komplexní číslo, které vyjadřuje signál na straně přijímače. Pokud je absolutní hodnota sumy všech signálů na straně přijímače (1.4) rovna 0, přijímač není pokryt.

Kapitola 2

Analýza a návrh

2.1 Změny v požadavcích na simulátor

Po úvodních konzultacích s vedoucím práce byl požadavek na vykreslování animací vyzařovací charakteristiky podle vstupních parametrů měnících se v čase vypuštěn a nahrazen požadavkem na možnost přidávání přijímačů a vykreslování jejich pokrytí. Hlavním požadavkem ale stále zůstává vykreslování vyzařovací charakteristiky transceiverů tvořících distribuovanou anténní řadu a její export do vektorového grafického formátu.

2.2 Modelování business procesů

Během analýzy bylo identifikováno 7 procesů, které bude aplikace běžně vykonávat.

2.2.1 Přidání entity

Přidání entity je základním procesem, který umožňuje uživateli přidat na scénu jednu ze dvou entit – transceiver nebo přijímač. Přidání bude možné provést 3 způsoby, specifikací parametrů v kontextovém menu, kliknutím na požadované místo na scéně a vyplněním zbylých parametrů v kontextovém menu, nebo načtením dat ze souboru.

V případě přidávání transceiveru bude nutné určit jeho souřadnice, vysílací výkon, fázový posun a volitelně název. Při přidávání přijímače jsou povinným parametrem pouze jeho souřadnice, název je volitelný.

2.2.2 Odebrání entity

Pokud budou na scéně umístěny nějaké entity, bude možné je stisknutím tlačítka odebrat. Pro odebírání transceiverů i přijímačů bude sloužit jediné tlačítko. Pokud bude vybráno více entit najednou, budou smazány všechny.

2.2.3 Změna parametrů entity

Parametry entit umístěných na scéně bude možné měnit. Všechny parametry budou měnitelné pomocí kontextového menu, pozice entit pak bude měnitelná pomocí kliknutí a tažení myši.

2.2.4 Vykreslení vyzařovací charakteristiky

Po umístění alespoň 2 transceiverů na scénu bude možné stisknutím tlačítka zobrazit vyzařovací charakteristiku. Předpokládá se, že všechny přidané transceivery patří do jedné distribuované anténní řady, jejíž vyzařovací charakteristika bude zobrazena.

2.2.5 Vykreslení pokrytí přijímačů

Tlačítkem bude možné zapnout / vypnout zobrazení pokrytí přijímačů na scéně signálem z přidaných transceiverů. S ohledem na výkon se nebude pokrytí aktualizovat během přesouvání přijímače myší, zobrazí se až po jeho upuštění.

2.2.6 Uložení aktuálního stavu do souboru

Parametry zadaných entit bude možné uložit do souboru, který bude možné do aplikace znovu načíst.

2.2.7 Načtení stavu ze souboru

Ze zvoleného souboru bude možné načíst data dříve uložená programem nebo specifikovaná uživatelem.

Soubor bude mít textovou podobu a bude snadno upravitelný, aby jej bylo možné upravovat i bez použití aplikace.

2.3 Doménový model

Během analýzy byly identifikovány 3 modelové třídy, se kterými bude program pracovat.

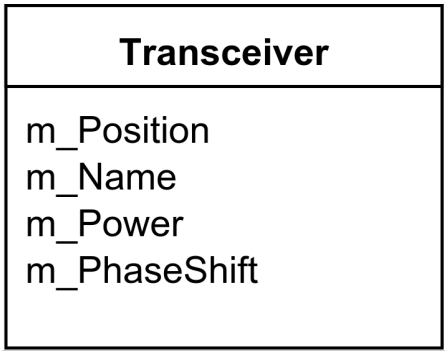
2.3.1 Transceiver

První modelová třída představuje transceiver.

m_Position Pozice transceiveru na scéně

m_Name Zobrazované jméno transceiveru

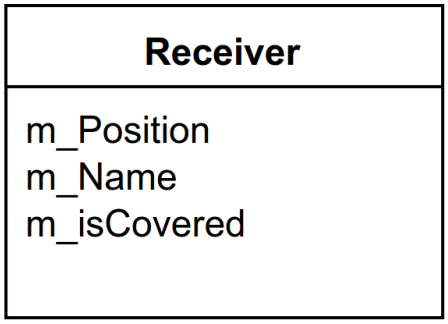
m_Power Vysílací výkon transceiveru



Obrázek 2.1: Doménový model transceiveru

m_PhaseShift Fázový posun vysílání transceiveru od referenční hodnoty

2.3.2 Receiver (přijímač)



Obrázek 2.2: Doménový model přijímače

Druhá modelová třída představuje přijímač.

m_Position Pozice přijímače na scéně

m_Name Zobrazované jméno přijímače

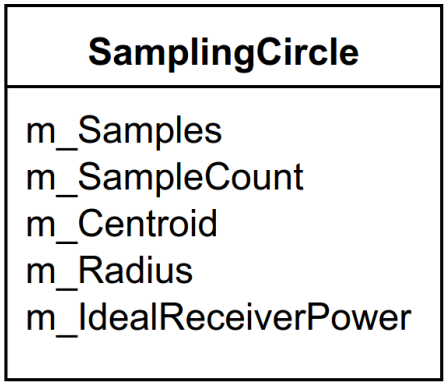
m_isCovered Určuje, jestli je přijímač pokrytý signálem

2.3.3 Sampling circle

Třetí modelová třída uchovává údaje potřebné k vykreslení vyzařovací charakteristiky.

m_Samples Kolekce naměřených hodnot zisku

m_SampleCount Počet naměřených hodnot



Obrázek 2.3: Doménový model vzorkovací kružnice vyzařovací charakteristiky

m_Centroid Střed vzorkovací kružnice

m_Radius Poloměr vzorkovací kružnice

m_IdealReceiverPower Výkon přijatý ideálním přijímačem

2.4 Model požadavků

2.4.1 Funkční požadavky

Obrázek 2.4: Funkční požadavky

2.4.1.1 F1 – Správa modelových entit

Aplikace bude umožňovat přidávat, upravovat a odebírat modelové entity. Všechny přidané entity budou zobrazeny na scéně, transceivery a přijímače pak ještě v seznamu transceiverů a přijímačů. Entity bude možné zadávat buď ručně, nebo načtením ze souboru.

2.4.1.2 F2 – Zobrazení vyzařovací charakteristiky

Aplikace bude schopna po přidání 2 a více transceiverů zobrazit jejich vyzařovací charakteristiku. Ta bude zobrazena po kliknutí na tlačítko, nebude pohyblivá a na scéně bude vždy nejvýše 1.

2.4.1.3 F3 – Zobrazování pokrytí přijímačů

Aplikace bude umožňovat zobrazování pokrytí přijímačů. Zobrazování pokrytí bude možné zapnout / vypnout, zobrazí se tak pokrytí buď všech, nebo žádného přijímače. Při pohybu s přijímačem bude pokrytí zobrazeno až po dokončení pohybu.

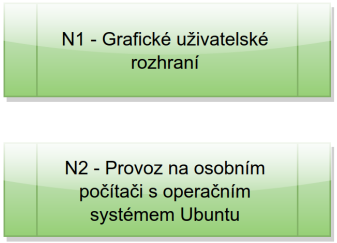
2.4.1.4 F4 – Ukládání a načítání stavu scény

Transceivery a přijímače přidané na scénu bude možné uložit do souboru o specifikovaném názvu a později je načíst v takovém stavu, v jakém byly uloženy. Vyzařovací charakteristika a pokrytí přijímačů nebude ukládáno, po načtení transceiverů a přijímačů je totiž lze jednoduše zobrazit znovu.

2.4.1.5 F5 – Export scény do vektorového grafického formátu

Celý stav scény bude možné exportovat do vektorového grafického formátu `svg`. Soubor bude uložen pod zadaným názvem.

2.4.2 Nefunkční požadavky



Obrázek 2.5: Nefunkční požadavky

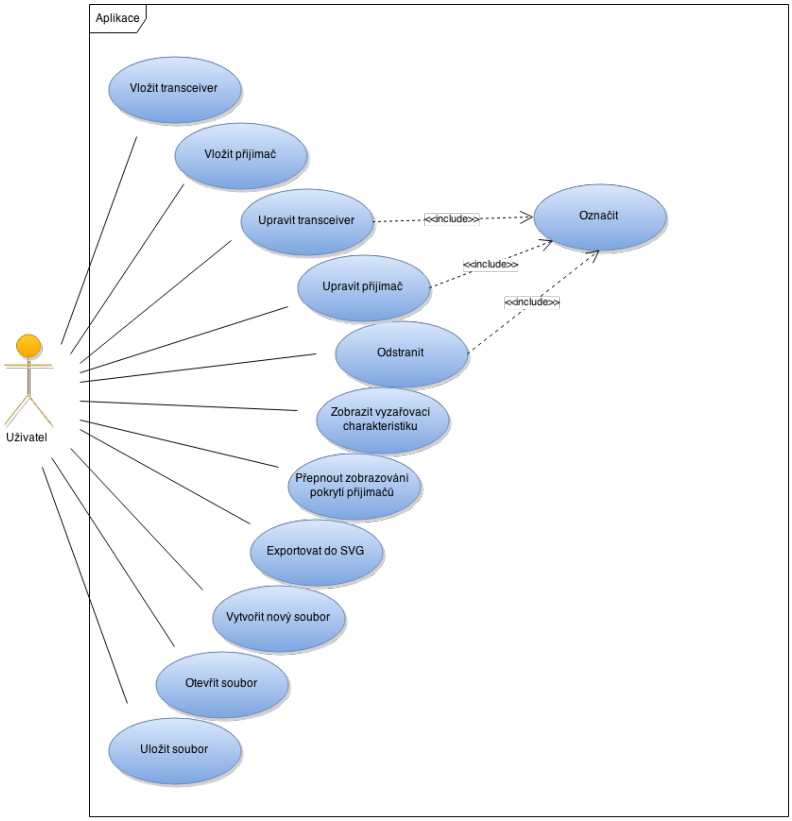
2.4.2.1 N1 – Grafické uživatelské rozhraní

Aplikace bude obsahovat grafické uživatelské prostředí, přes které bude přístupná veškerá funkcionalita aplikace.

2.4.2.2 N2 – Provoz na osobním počítači s OS Ubuntu

Aplikaci bude možné spustit na osobním počítači s operačním systémem Linux, konkrétně s jeho distribucí Ubuntu.

2.5 Model případů užití



Obrázek 2.6: Případy použití

2.6 Cílová platforma

Cílovou platformou je prostředí, ve kterém si zadavatel přeje spouštět program. V mém případě tedy jde o stolní počítač s operačním systémem Linux, konkrétně s jeho distribucí Ubuntu.

Multiplatformnost není uvedena v požadavcích na aplikaci, byla by však výhodou.

2.7 Volba programovacího jazyka

Po domluvě s vedoucím práce byl jako programovací jazyk práce vybrán jazyk C++. Vedoucí práce tento jazyk používá, bude tedy případně schopen si aplikaci doupravit. Pokud by navíc bylo nutné aplikaci v rámci další bakalářské práce přepsat, bude díky povinné výuce C++ na FIT ČVUT k dispozici dostatek studentů schopných práci upravit.

Nevýhodou jazyka C++ je fakt, že není multiplatformní a že nenabízí jednoduché nástroje pro tvorbu GUI. Obě tyto nevýhody však lze odstranit správnou volbou frameworku.

2.8 Volba frameworku

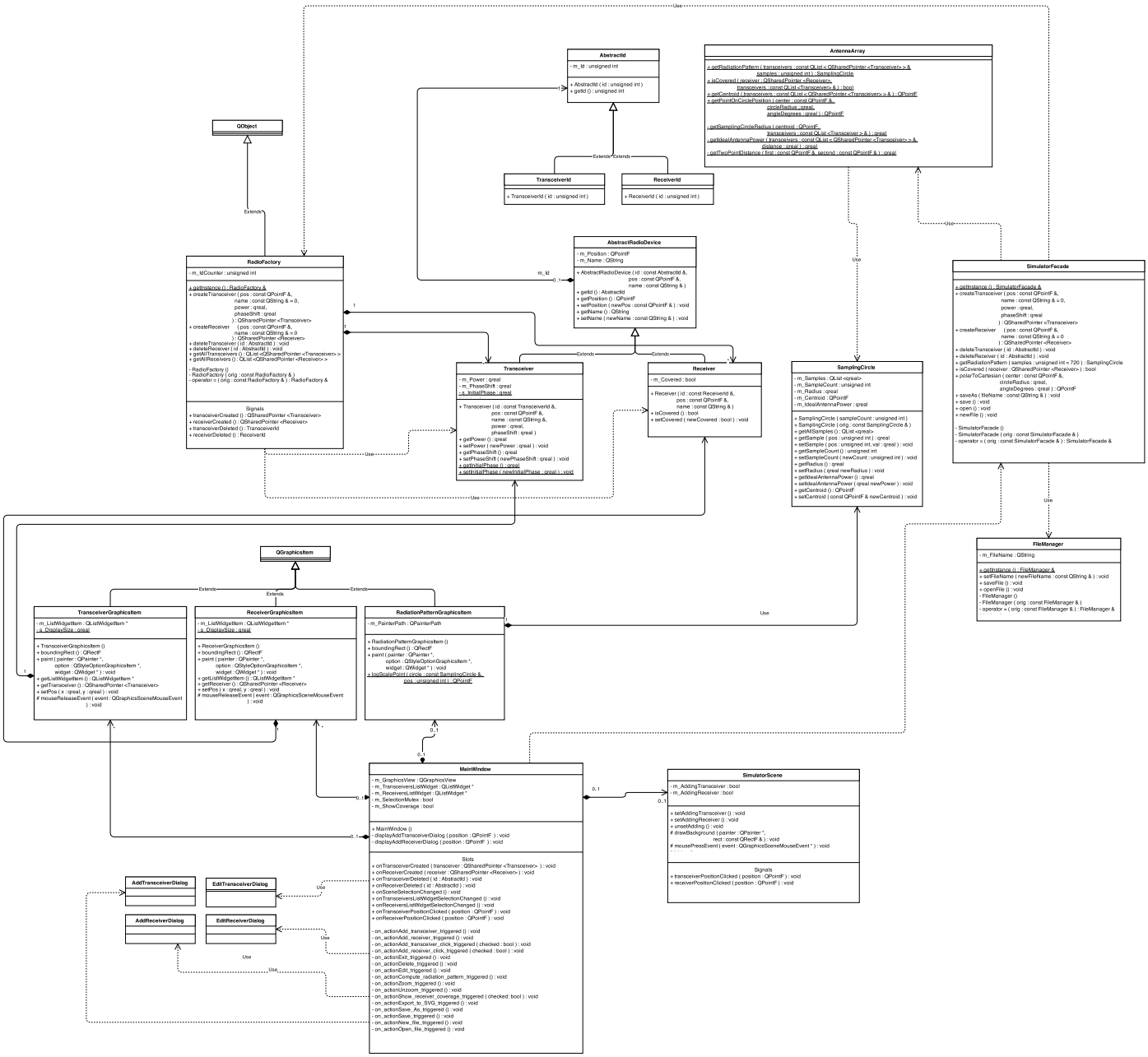
Framework, který hledám, musí fungovat pod systémem Linux, musí umožňovat vytvořit GUI, jeho použití pro účely této aplikace nesmí být zpoplatněno a musí umožňovat export do formátu `svg`. Výhodou je pak multiplatformnost.

Těmto požadavkům nejlépe odpovídají frameworky Qt a GTK+. Ovšem zatímco framework Qt je kompletním, konzistentním nástrojem, GTK je spíše nástrojem na tvorbu GUI, který dokáže spolupracovat s několika knihovnami [4]. Framework Qt také poskytuje obsáhlou dokumentaci a práce s grafickým formátem `svg` je v něm jednodušší. Dá se tedy předpokládat, že použitím frameworku Qt vznikne jednodušší a srozumitelnější kód. Jako framework pro tvorbu aplikace jsem tedy zvolil Qt.

2.9 Architektura aplikace

Vzhledem k tomu, že moje aplikace se neskládá z více částí nebo modulů, nezávisí na vzdálených prostředcích, ani neposkytuje funkcionalitu jiným aplikacím, je hlavní náplní architektury oddělit data od zobrazování. K tomu se hodí architektura model-view-controller, která má ve frameworku Qt podobu Model/View [5], přičemž funkčnost controlleru se rozdělí mezi model a view.

2.10 UML diagram tříd


Use
AbstractId
- m_Id : unsigned int
+ AbstractId (id : unsigned int)
+ getId () : unsigned int
AntennaArray
+ getRadiationPattern (transceivers : const QList < QSharedPointer <Transceiver> > &
samples : unsigned int) : SamplingCircle
+ isCovered (receiver : QSharedPointer <Receiver>,
transceivers : const QList < QSharedPointer <Transceiver> > &) : bool
+ getCentroid (transceivers : const QList < QSharedPointer <Transceiver> > &) : QPointF
+ getPointOnCirclePosition (center : const QPointF &,
circleRadius : qreal,
angleDegrees : qreal) : QPointF
- getSamplingCircleRadius (centroid : QPointF,
transceivers : const QList <Transceiver > &) : qreal
- getIdealAntennaPower (transceivers : const QList < QSharedPointer <Transceiver> > &,
distance : qreal) : qreal
- getTwoPointDistance (first : const QPointF &, second : const QPointF &) : qreal
QObject
Extends
TransceiverId
+ TransceiverId (id : unsigned int)
Extends
ReceiverId
+ ReceiverId (id : unsigned int)
Extends
RadioFactory
- m_IdCounter : unsigned int
+ getInstance () : RadioFactory &
+ createTransceiver (pos : const QPointF &,
name : const QString & = 0,
power : qreal,
phaseShift : qreal
) : QSharedPointer <Transceiver>
+ createReceiver (pos : const QPointF &,
name : const QString & = 0
) : QSharedPointer <Receiver>
+ deleteTransceiver (id : AbstractId) : void
+ deleteReceiver (id : AbstractId) : void
+ getAllTransceivers () : QList <QSharedPointer <Transceiver> >
+ getAllReceivers () : QList <QSharedPointer <Receiver> >
- RadioFactory ()
- RadioFactory (orig : const RadioFactory &)
- operator = (orig : const RadioFactory &) : RadioFactory &
Signals
+ transceiverCreated () : QSharedPointer <Transceiver>
+ receiverCreated () : QSharedPointer <Receiver>
+ transceiverDeleted () : TransceiverId
+ receiverDeleted () : ReceiverId
AbstractRadioDevice
- m_Position : QPointF
- m_Name : QString
+ AbstractRadioDevice (id : const AbstractId &,
pos : const QPointF &,
name : const QString &)
+ getId () : AbstractId
+ getPosition () : QPointF
+ setPosition (newPos : const QPointF &) : void
+ getName () : QString
+ setName (newName : const QString &) : void
m_Id
0..1
Transceiver
- m_Power : qreal
- m_PhaseShift : qreal
- s_InitialPhase : qreal
+ Transceiver (id : const TransceiverId &,
pos : const QPointF &,
name : const QString &,
power : qreal,
phaseShift : qreal)
+ getPower () : qreal
+ setPower (newPower : qreal) : void
+ getPhaseShift () : qreal
+ setPhaseShift (newPhaseShift : qreal) : void
+ getInitialPhase () : qreal
+ setInitialPhase (newInitialPhase : qreal) : void
Receiver
- m_Covered : bool
+ Receiver (id : const ReceiverId &,
pos : const QPointF &,
name : const QString &)
+ isCovered () : bool
+ setCovered (newCovered : bool) : void
SamplingCircle
- m_Samples : QList <qreal>
- m_SampleCount : unsigned int
- m_Radius : qreal
- m_Centroid : QPointF
- m_IdealAntennaPower : qreal
+ SamplingCircle (sampleCount : unsigned int)
+ SamplingCircle (orig : const SamplingCircle &)
+ getAllSamples () : QList <qreal>
+ getSample (pos : unsigned int) : qreal
+ setSample (pos : unsigned int, val : qreal) : void
+ getSampleCount () : unsigned int
+ setSampleCount (newCount : unsigned int) : void
+ getRadius () : qreal
+ setRadius (qreal newRadius) : void
+ getIdealAntennaPower () : qreal
+ setIdealAntennaPower (qreal newPower) : void
+ getCentroid () : QPointF
+ setCentroid (const QPointF & newCentroid) : void
SimulatorFacade
+ getInstance () : SimulatorFacade &
+ createTransceiver (pos : const QPointF &,
name : const QString & = 0,
power : qreal,
phaseShift : qreal
) : QSharedPointer <Transceiver>
+ createReceiver (pos : const QPointF &,
name : const QString & = 0
) : QSharedPointer <Receiver>
+ deleteTransceiver (id : AbstractId) : void
+ deleteReceiver (id : AbstractId) : void
+ getRadiationPattern (samples : unsigned int = 720) : SamplingCircle
+ isCovered (receiver : QSharedPointer <Receiver>) : bool
+ polarToCartesian (center : const QPointF &,
circleRadius : qreal,
angleDegrees : qreal) : QPointF
+ saveAs (fileName : const QString &) : void
+ save () : void
+ open () : void
+ newFile () : void
- SimulatorFacade ()
- SimulatorFacade (orig : const SimulatorFacade &)
- operator = (orig : const SimulatorFacade &) : SimulatorFacade &
Use
FileManager
- m_FileName : QString
+ getInstance () : FileManager &
+ setFileName (newFileName : const QString &) : void
+ saveFile () : void
+ openFile () : void
- FileManager ()
- FileManager (orig : const FileManager &)
- operator = (orig : const FileManager &) : FileManager &
QGraphicsItem
Extends
Extends
Extends
TransceiverGraphicsItem
- m_ListWidgetItem : QListWidgetItem *
+ s_DisplaySize : qreal
+ TransceiverGraphicsItem ()
+ boundingRect () : QRectF
+ paint (painter : QPainter *,
option : QStyleOptionGraphicsItem *,
widget : QWidget *) : void
+ getListWidgetItem () : QListWidgetItem *
+ getTransceiver () : QSharedPointer <Transceiver>
+ setPos (x : qreal, y : qreal) : void
# mouseReleaseEvent (event : QGraphicsSceneMouseEvent
) : void
1
1
ReceiverGraphicsItem
- m_ListWidgetItem : QListWidgetItem *
+ s_DisplaySize : qreal
+ ReceiverGraphicsItem ()
+ boundingRect () : QRectF
+ paint (painter : QPainter *,
option : QStyleOptionGraphicsItem *,
widget : QWidget *) : void
+ getListWidgetItem () : QListWidgetItem *
+ getReceiver () : QSharedPointer <Receiver>
+ setPos (x : qreal, y : qreal) : void
# mouseReleaseEvent (event : QGraphicsSceneMouseEvent
) : void
*
*
RadiationPatternGraphicsItem
- m_PainterPath : QPainterPath
+ RadiationPatternGraphicsItem ()
+ boundingRect () : QRectF
+ paint (painter : QPainter *,
option : QStyleOptionGraphicsItem *,
widget : QWidget *) : void
+ lodScalePoint (circle : const SamplingCircle &,
pos : unsigned int) : QPointF
1
0..1
0..1
Use
MainWindow
- m_GraphicsView : QGraphicsView
- m_TransceiversListWidget : QListWidget *
- m_ReceiversListWidget : QListWidget *
- m_SelectionMutex : bool
- m_ShowCoverage : bool
+ MainWindow ()
- displayAddTransceiverDialog (position : QPointF) : void
- displayAddReceiverDialog (position : QPointF) : void
Slots
+ onTransceiverCreated (transceiver : QSharedPointer <Transceiver>) : void
+ onReceiverCreated (receiver : QSharedPointer <Receiver>) : void
+ onTransceiverDeleted (id : AbstractId) : void
+ onReceiverDeleted (id : AbstractId) : void
+ onSceneSelectionChanged () : void
+ onTransceiversListWidgetSelectionChanged () : void
+ onReceiversListWidgetSelectionChanged () : void
+ onTransceiverPositionClicked (position : QPointF) : void
+ onReceiverPositionClicked (position : QPointF) : void
- on_actionAdd_transceiver_triggered () : void
- on_actionAdd_receiver_triggered () : void
- on_actionAdd_transceiver_click_triggered (checked : bool) : void
- on_actionAdd_receiver_click_triggered (checked : bool) : void
- on_actionExit_triggered () : void
- on_actionDelete_triggered () : void
- on_actionEdit_triggered () : void
- on_actionCompute_radiation_pattern_triggered () : void
- on_actionZoom_triggered () : void
- on_actionUnzoom_triggered () : void
- on_actionShow_receiver_coverage_triggered (checked: bool) : void
- on_actionExport_to_SVG_triggered () : void
- on_actionSave_As_triggered () : void
- on_actionSave_triggered () : void
- on_actionNew_file_triggered () : void
- on_actionOpen_file_triggered () : void
0..1
0..1
SimulatorScene
- m_AddingTransceiver : bool
- m_AddingReceiver : bool
+ setAddingTransceiver () : void
+ setAddingReceiver () : void
+ unsetAdding () : void
# drawBackground (painter : QPainter *,
rect : const QRectF &) : void
mousePressEvent (event : QGraphicsSceneMouseEvent *) : void
Signals
+ transceiverPositionClicked (position : QPointF) : void
+ receiverPositionClicked (position : QPointF) : void
AddTransceiverDialog
EditTransceiverDialog
AddReceiverDialog
EditReceiverDialog
Use
Use
Use
Use

Kapitola 3

Implementace

3.1 Model

3.1.1 Identifikační třídy

Tyto třídy představují ID jednotlivých modelových objektů, slouží k jejich identifikaci při mazání a jako index do HashMapy. Pro třídu AbstractId je definována metoda `uint qHash ( const AbstractId & )` , která jako hash vrací m_Id předaného objektu.

3.1.2 Entitní třídy

3.1.2.1 AbstractRadioDevice

Tato třída je společným předkem tříd Transceiver a Receiver, obsahuje jejich ID, pozici a název.

3.1.2.2 Transceiver

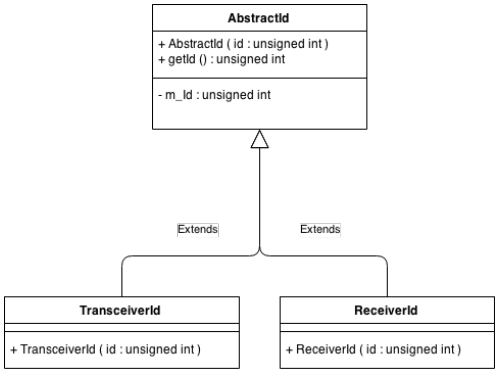
Třída Transceiver dědí od AbstractRadioDevice ID, pozici a jméno, navíc přidává údaj o vysílacím výkonu transceiveru a o jeho fázovém posunu. Statická proměnná s_InitialPhase představuje referenční fázový posun.

3.1.2.3 Receiver

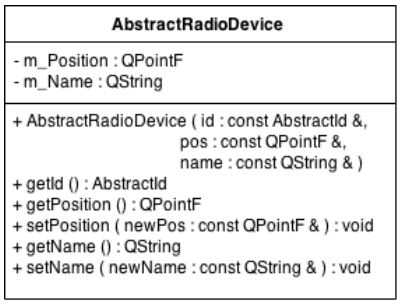
Třída Receiver k vlastnostem zděděným od AbstractRadioDevice přidává pouze proměnnou m_Covered, která udává, jestli je přijímač pokryt signálem.

3.1.2.4 SamplingCircle

SamplingCircle reprezentuje vzorkovací kružnici, na jejíž obvod jsou umísťovány přijímače, pomocí kterých je vypočítáván zisk distribuované anténní řady



Obrázek 3.1: Identifikační třídy

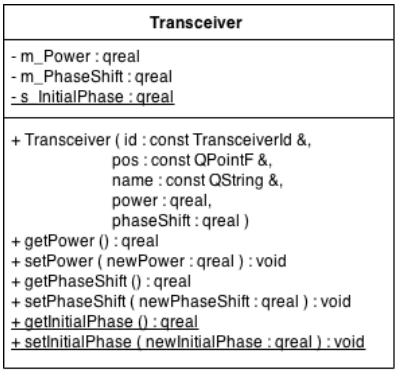


Obrázek 3.2: Třída AbstractRadioDevice

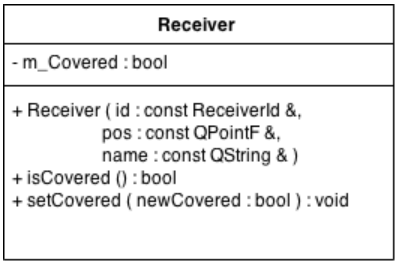
v daném bodě. Proměnná m_Samples obsahuje seznam vypočtených zisků antény, prvek na pozici i představuje hodnotu zisku v úhlu $\frac{i}{\frac{m\_SampleCount}{360}}$. Proměnné m_Centroid a m_Radius představují pozici středu a poloměr vzorkovací kružnice. Proměnná m_IdealAntennaPower pak obsahuje výkon přijatý ideálním přijímačem.

3.1.3 Konkrétní továrna

Instance entitních tříd Transceiver a Receiver jsou vytvářeny třídou RadioFactory, což je singleton, který implementuje návrhový vzor konkrétní továrna. Nepoužívám abstraktní továrnu, protože rozšíření aplikace o modifikované třídy Transceiver a Receiver se neočekává. Třída RadioFactory obsahuje proměnnou m_IdCounter, která zajišťuje, že nevzniknou 2 objekty o stejném ID. Třída si udržuje HashMapu vytvářených objektů, aby s nimi mohla dále pracovat. Po přidání nebo odebrání instance entitní třídy je vyslán příslušný signál, aby se o přidání / smazání objektu dozvěděli příjemci signálu.



Obrázek 3.3: Třída Transceiver



Obrázek 3.4: Třída Receiver

3.1.4 Fasáda simulátoru

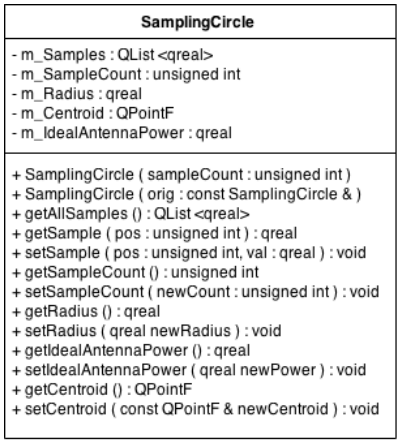
Třída SimulatorFacade je opět singleton, tentokrát implementující návrhový vzor fasáda. Samotná fasáda nevykonává žádnou činnost, pouze zjednodušuje přístup z view k metodám modelu.

3.1.5 File Manager

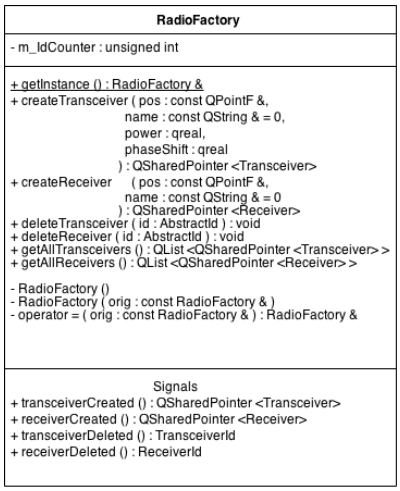
FileManager je singleton, který se stará o práci se soubory. Proměnná m_FileName obsahuje název souboru, ze kterého se čte nebo do kterého se zapisuje. Pokud je m_FileName prázdná, nebo pokud dojde k chybě při práci se soubory, je vyhozena výjimka. Metoda *voidopen*() parsuje zadaný soubor na základě regulárních výrazů. K vytváření instancí se používá třída RadioFactory, která po přidání vyšle signál, tím je zaručeno přidání instance i do view.

3.1.5.1 Formát ukládaného souboru

Ukládaný soubor obsahuje 2 druhy záznamů. Jeden představuje Transceiver, je ve formátu "T<mezera><Souřadnice X><mezera><Souřadnice Y> <mezera><Výkon><mezera><Fázový posun>". Druhý záznam představuje Receiver, má formát "R<mezera><Souřadnice X><mezera><Souřadnice Y>".



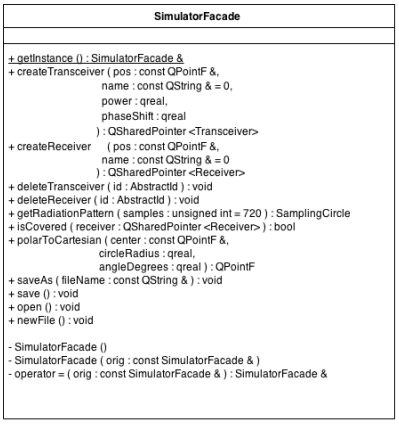
Obrázek 3.5: Třída SamplingCircle



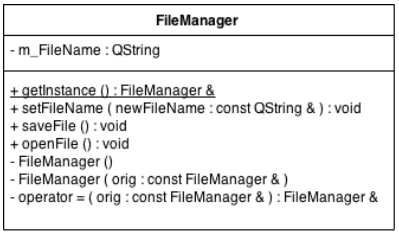
Obrázek 3.6: Třída RadioFactory

3.1.6 AntennaArray

Třída AntennaArray obstarává všechny výpočty, které souvisí s vykreslováním vyzařovací charakteristiky a výpočtem pokrytí přijímačů. Všechny metody třídy jsou statické, od třídy se tedy nevytváří instance. Metoda *getRadiationPattern*() vypočte pomocí výpočtu zisku popsaného v sekci 1.1 vyzařovací charakteristiku zadaných transceiverů. Metoda *isCovered*() zase s použitím vzorců ze sekce 1.2 vypočte, zda je zadaný přijímač pokryt. Za zmínku stojí ještě metoda *getPointOnCirclePosition*(), která je v třídě RadiationPatternGraphicsItem používána k přepočtu bodů v polárních souřadnicích na body v kartézských souřadnicích.



Obrázek 3.7: Třída SimulatorFacade



Obrázek 3.8: Třída FileManager

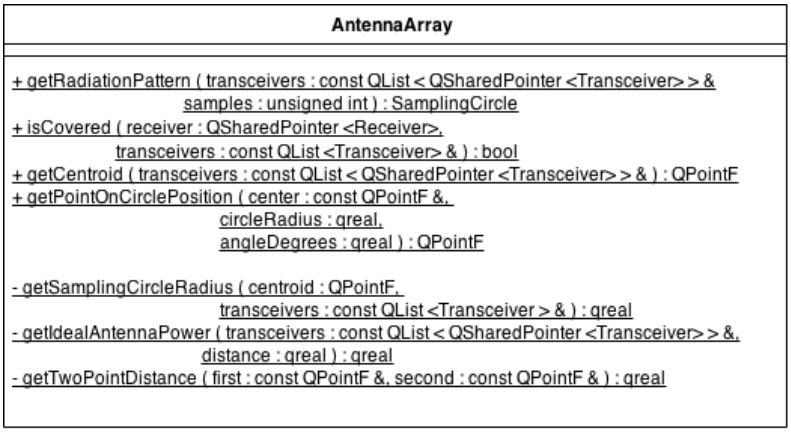
3.2 View

3.2.1 TransceiverGraphicsItem

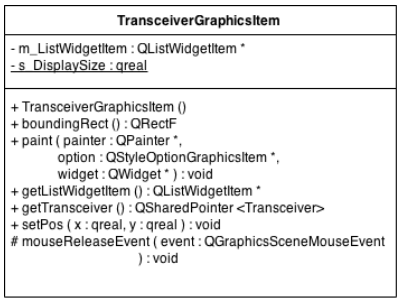
TransceiverGraphicsItem dědí od abstraktní třídy QGraphicsItem, definováním metod *boundingRect*() a *paint*() určuje způsob, jakým bude zobrazena. Třída představuje grafickou reprezentaci modelové třídy Transceiver. TransceiverGraphicsItem obsahuje pointer na instanci Transceiveru, té se při přesunu TransceiverGraphicsItem mění pozice. Dále obsahuje pointer na QListWidgetItem, aby bylo možné při označení TransceiverGraphicsItem označit i příslušnou QListWidgetItem. V statické proměnné s_DisplaySize je uložena velikost TransceiverGraphicsItem. Třída má překrytou metodu *setPos*(), aby byla při vytvoření správně umístěna.

3.2.2 ReceiverGraphicsItem

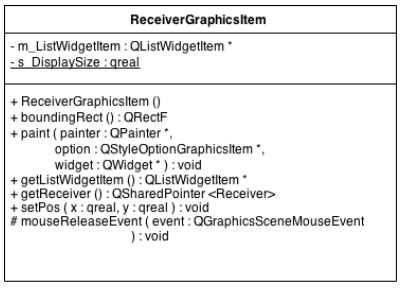
ReceiverGraphicsItem je vpodstatě to samé, co TransceiverGraphicsItem, pouze reprezentuje třídu Receiver. Metody, které definuje, mají podobnou funkčnost jako ty z TransceiverGraphicsItem.



Obrázek 3.9: Třída AntennaArray



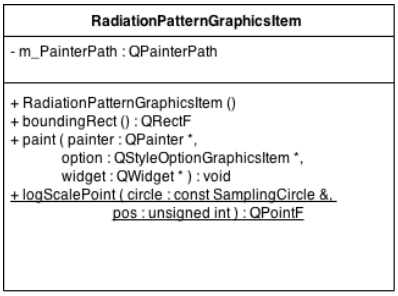
Obrázek 3.10: Třída TransceiverGraphicsItem



Obrázek 3.11: Třída ReceiverGraphicsItem

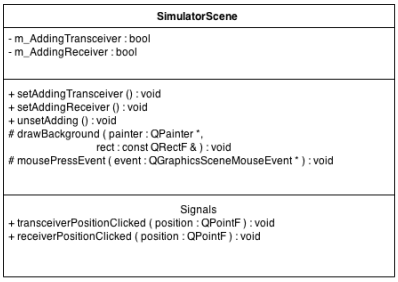
3.2.3 RadiationPatternGraphicsItem

RadiatonPatternGraphicsItem představuje vyzařovací charakteristiku, na scéně je vždy nejvýše jedna. V konstruktoru dojde k přepočtu zisku na body na scéně, mezi těmi je nakreslena cesta a zobrazena.



Obrázek 3.12: Třída RadiationPatternGraphicsItem

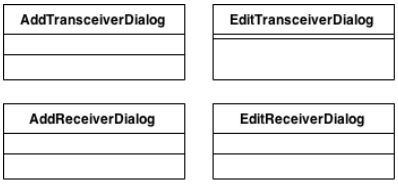
3.2.4 SimulatorScene



Obrázek 3.13: Třída SimulatorScene

Třída SimulatorScene dědí od QGraphicsScene a překrývá metodu *drawBackground*(), pomocí které zobrazuje mřížku na pozadí. Pomocí předefinované *mousePressEvent*() umožňuje přidat TransceiverGraphicsItem a ReceiverGraphicsItem na pozici, na kterou bylo kliknuto myší.

3.2.5 Dialogové třídy

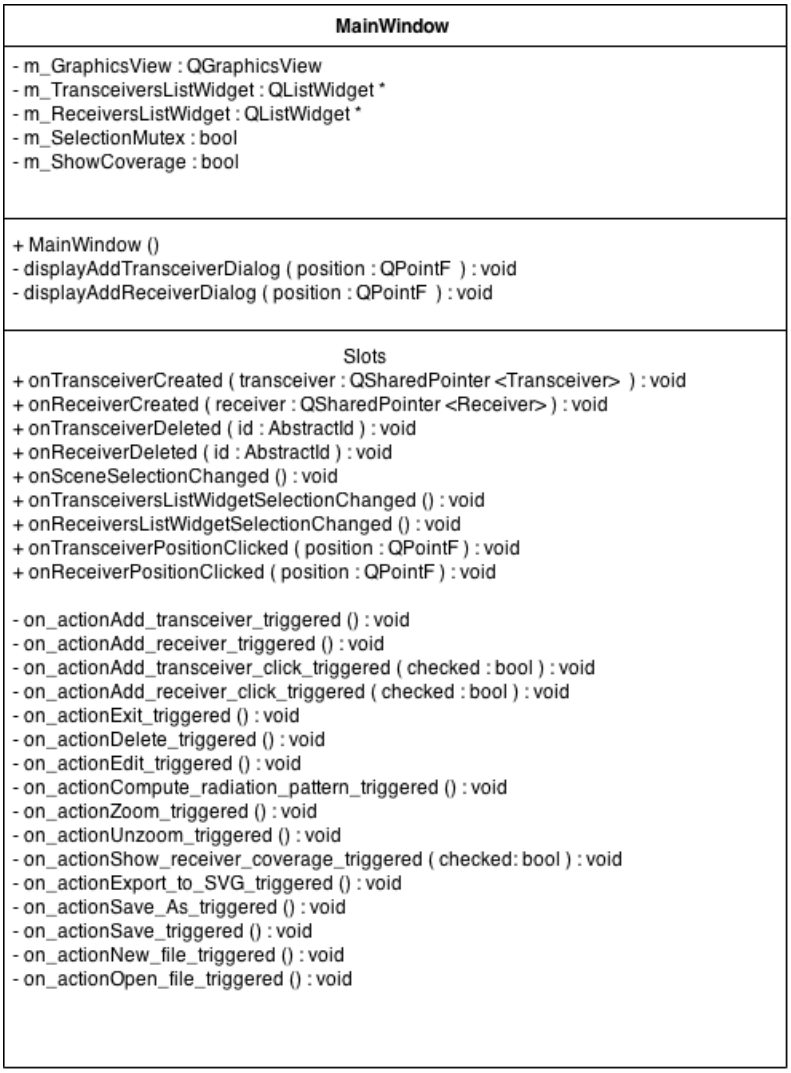


Obrázek 3.14: Dialogové třídy

Tyto třídy umožňují přidávat a upravovat nové Transceivery a Receivery. Parametrem kostruktoru tříd AddTransceiverDialog a AddReceiverDialog jsou pointery na proměnné, které jsou poté předány konstruktoru Transceiveru

nebo Receiveru. Parametrem konstruktoru tříd EditTransceiverDialog a EditReceiverDialog je pak pointer na Transceiver nebo Receiver.

3.2.6 Třída MainWindow



Obrázek 3.15: Třída MainWindow

Třída MainWindow obsahuje seznamy pointerů na TransceiverGraphicsItem, ReceiverGraphicsItem a 1 pointer na RadiationPatternGraphicsItem. Veřejné sloty této třídy jsou napojeny na signály aplikace, privátní sloty pak představují metody, které se zavolají, když uživatel provede akci. Proměnná m_SelectionMutex zabraňuje, aby se zavolalo více metod označujících prvky scény a QListWidgetů najednou.

Kapitola 4

Testování

Kvůli časovým důvodům a nedostatku testovacích dat nebyla aplikace řádně otestována, proběhly pouze jednotkové testy jednotlivých tříd. Těmi aplikace prošla bez problému.

Během běžného používání byly objeveny 2 chyby, z nichž jedna je závažná. Pokud se uživatel pokusí přesunout více transceiverů či receiverů najednou, dojde k úpravě polohy pouze toho objektu, který byl během přesunu pod ukazatelem myši. Tato chyba nevede k pádu aplikace, způsobí však nesprávé vykreslení vyzařovací charakteristiky. Druhou chybou je pak snížení výkonu, pokud je zobrazena vyzařovací charakteristika a uživatel se pokusí pohled přiblížit. Podle [6] však jde o chybu způsobenou vývojovým prostředím.

Závěr

Na základě provedené analýzy se mi podařilo vyvinout aplikaci, která pomocí uvedených vzorců dokáže zobrazit vyzařovací charakteristiku zadané distribuované anténní řady, která umí zobrazit pokrytí zadaný přijímačů. Umožňuje také export zobrazené scény do vektorového grafického formátu `svg`. Stav scény je možné uložit do specifikovaného souboru a poté jej znovu načíst. Vytvořená aplikace bohužel nebyla správně otestována v celém rozsahu, podle provedených testů se však zatím zdá, že aplikace funguje správně.

Literatura

[1] Cerny, V.; Moucha, A.; Kubr, J.: Interference Cancellation by the Usage of Distributed Phase Shift Beamforming. *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, 2014: s. 248–253. Dostupné z: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6982924`

[2] Moucha, A. M.; Cerny, V.: Anisotropic antenna collaborative beamforming in adhoc networks. *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference on ZZZ - IWCMC '10*, 2010: s. 971–. Dostupné z: `http://portal.acm.org/citation.cfm?doid=1815396.1815618`

[3] Balanis, C. A.: *Antenna theory*. Hoboken: Wiley-Interscience, třetí vydání, 2005, ISBN 978-0-471-66782-7.

[4] GTK vs Qt. 2015. Dostupné z: `https://www.wikivs.com/wiki/GTK_vs_Qt`

[5] Company, T. Q.: Model/View programming. Dostupné z: `http://doc.qt.io/qt-5.4/model-view-programming.html`

[6] Bad performance with scaled QPainter and drawing QPainterPath. Dostupné z: `https://bugreports.qt.io/browse/QTBUG-4317`

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

Transceiver **Trans**mitter and re**ceiver**

Příloha **B**

Obsah přiloženého CD

- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Kozak_Jan_2015.pdf` ..... text práce ve formátu PDF
  - `BP_Kozak_Jan_2015.ps` ..... text práce ve formátu PS