ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

DIPLOMOVÁ PRÁCA

Študijný odbor: 9.2.4. počítačové inžinierstvo
Študijný program: počítačové inžinierstvo

Bc. Matej Jakab

Riadiaci systém pre komunikáciu a prenos dát medzi základňovou stanicou a modelom vzducholode

Vedúci diplomovej práce: Ing. Michal Hodoň, PhD.

Reg. č. 387/2016 Máj 2017

ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

Riadiaci systém pre komunikáciu a prenos dát medzi základňovou stanicou a modelom vzducholode

(Diplomová práca)

Študijný odbor:	9.2.4. počítačové inžinierstvo
Študijný program:	Počítačové inžinierstvo
Školiace pracovisko:	Katedra technickej kybernetiky
Stupeň kvalifikácie:	Inžinier (Ing.)
Ministerské číslo práce:	28360820172387

Žilina 2017 **Bc. Matej Jakab**

ŽILINSKÁ UNIVERZITA V ŽILINE, FAKULTA RIADENIA A INFORMATIKY.

ZADANIE TÉMY DIPLOMOVEJ PRÁCE.

Študijný program : Počítačové inžinierstvo

Meno a priezvisko
Matej Jakab

Osobné číslo
556154

Názov práce v slovenskom aj anglickom jazyku

Riadiaci systém pre komunikáciu a prenos dát medzi základňovou stanicou a modelom vzducholode

Communication subsystem of airship

Zadanie úlohy, ciele, pokyny pre vypracovanie (Ak je málo miesta, použite opačnú stranu)

Cieľ diplomovej práce:
Cieľom práce je implementácia riadiaceho systému v rámci modelu vzducholode, ktorý umožní vzdialené ovládanie vzducholode súčasnou implementáciou viacerých technológií.

Obsah:
Výber vhodných HW prostriedkov.
Návrh a implementácia architektúry riadiaceho a komunikačného podsystému na vybranom modely vzducholode.
Návrh a implementácia programového vybavenia pre daný podsystém.
Návrh a implementácia programového riešenia pre grafické rozhranie systému na strane užívateľa.
Praktické overenie funkčnosti systému, uskutočnenie experimentálnych meraní.

Meno a pracovisko vedúceho DP: Ing. Michal Hodoň, PhD., KTK, ŽU
Meno a pracovisko tútora DP:

vedúci katedry
(dátum a podpis

Zadanie zaregistrované dňa 31. 10. 2016 pod číslom 387/2016 podpis _____

ABSTRAKT

JAKAB Matej: Riadiaci systém pre komunikáciu a prenos dát medzi základňovou stanicou a modelom vzducholode [diplomová práca] – Žilinská Univerzita v Žiline, Fakulta riadenia a informatiky, Katedra technickej kybernetiky. – Vedúci diplomovej práce: Ing. Michal Hodoň, PhD. – Stupeň odbornej kvalifikácie: Inžinier v odbore Počítačové Inžinierstvo. Žilina: FRI ŽU v Žiline, 2017. – 90 s.

Cieľom diplomovej práce bolo navrhnúť a realizovať riadiaci systém modelu vzducholode, ktorý umožní vzdialené ovládanie vzducholode súčasnou implementáciou viacerých technológii. Práca bola rozdelená na realizáciu podsystémov komunikačného rozhrania, riadiaceho rozhrania na strane používateľa a softvérového riešenia základňovej stanice v podobe aplikácie s grafickým používateľským rozhraním. V úvodnej časti sme špecifikovali úlohy zadania a zadefinovali postup riešenia. V nasledujúcich kapitolách sme opísali návrh a realizáciu riešenia jednotlivých podsystémov. V záverečnej časti práce bol popísaný návrh testovacieho rozhrania, na ktorom bolo výsledne riešenie diplomovej práce otestované.

Kľúčové slová: vzducholoď, komunikačný podsystém, riadiaci podsystém, GUI aplikácia, Wi-Fi technológia, GSM technológia, TCP socket

ABSTRACT

JAKAB Matej: Communication subsystem of airship [Master's thesis] – The University of Žilina, Faculty of Management Science and Information Technologies, Technical Cybernetics Department. Tutor: Ing. Michal Hodoň, PhD. – Qualification level: Master in field Computer Engineering. Žilina: FRI ŽU in Žilina, 2017. – 90 p.

The aim of master's thesis was to design and implement control system for airship model, which provides remote control of airship by implementation of various present technologies. Thesis was divided into implementation of communication subsystem, user control subsystem and software solution of base station in the form of application with graphical user interface. In opening chapter, we specified thesis tasks and methods of solution. In following chapters, we described design and implementation of solution for each subsystem. Design of testing interface was described in last part of work, where final solution of master's thesis was tested.

Key words: airship, communication subsystem, control subsystem, GUI application, Wi-Fi technology, GSM technology, TCP socket

ČESTNÉ PREHLÁSENIE

Čestne vyhlasujem, že celú diplomovú prácu na tému „Riadiaci systém pre komunikáciu a prenos dát medzi základňovou stanicou a modelom vzducholode“, vrátane všetkých jej príloh a obrázkov, som vypracoval samostatne, a to s použitím literatúry uvedenej v priloženom zozname.

V Žiline dňa ....

Poďakovanie

Týmto sa chcem poďakovať Ing. Michalovi Hodoňovi, PhD. za ochotu, odbornú pomoc a rady, ktoré mi pomohli pri vypracovaní tejto práce.

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek a značiek

Skratka	Anglický význam	Slovenský význam
API	Application programming interface	Rozhranie pre programovanie aplikácii
	framework	Súbor podporných nástrojov
GPS	Global Positioning System	Globálny lokalizačný systém
GSM	Global System for Mobile Communications	Globálny systém mobilných komunikácii
GUI	Graphical User Interface	Grafické užívateľské rozhranie
HTTP	Hypertext transfer protocol	Hypertextový prenosový protokol
IDE	Integrated Development Enviroment	Integrované vývojové prostredie
IKT	-	Informačné a komunikačné technológie
IP	Internet Protocol	Internetový protokol
LTA	Lighter than air	Ľahší ako vzduch
OSI	Open System Interconnection	Referenčný model návrhu štruktúry počítačových sieťových protokolov
PCI	Peripheral Component Interconnect	Štandard zbernice počítača pre pripojenie periférnych zariadení
SIM	Subscriber Identity Module	Účastnícka identifikačná karta
TCP	Transmission Control Protocol	Protokol riadenia prenosu
UDP	User Datagram Protocol	Používateľský datagramový protokol
URL	Uniform Resource Locator	Jednotný vyhľadávač prostriedku
USB	Universal Serial Bus	Univerzálna sériová zbernica
Wi-Fi	Wireless local area networking	Štandard pripojenia elektronických zariadení k bezdrôtovej sieti

Úvod

Rozvoj technológii napreduje neustále dopredu a prináša so sebou nové prístupy a možnosti pri riešení úloh v rôznych oblastiach vedy a techniky. Pokrok neprináša iba nové technológie, ale aj zlepšuje už existujúce. Jedným takýmto prostriedkom, ktorý sa dostáva v poslednej dobe do popredia, je využitie vzducholode pri riešení rôznych úloh informačných a komunikačných technológii. Podľa uverejnených článkov a odborných štúdií, vzducholoď realizovaná súčasným technologickým prístupom, v mnohých smeroch prevyšuje iné druhy lietajúcich prostriedkov, akými sú napríklad bezpilotné lietadlá (drony).

Zadanie našej diplomovej práce vychádzalo z projektu, ktorého úlohou bolo využitie modelu vzducholode ako prostriedku na riešenie úloh IKT. Projekt bol rozdelený na viaceré časti a témou našej práce bolo navrhnúť a realizovať riadiaci systém v rámci modelu vzducholode, ktorý umožní vzdialené ovládanie vzducholode súčasnou implementáciou viacerých technológii. Práca bola rozdelená na realizáciu viacerých podsystémov, ktoré spolu vytvárali kompletný systém na strane používateľa. Hlavnými podsystémami, ktorých realizácii sme sa venovali, boli - komunikačné rozhranie, riadiace rozhranie na strane používateľa a aplikácia základňovej stanice implementujúca realizované podsystémy.

V úvodnej kapitole sme špecifikovali jednotlivé úlohy zadania a zadefinovali si vhodný postup ich riešenia. V ďalšej kapitole sme sa venovali analýze a výberu hardvérových prostriedkov potrebných pri riešení jednotlivých podsystémov. V nasledujúcej časti práce sme sa venovali návrhu riadiaceho podsystému. V úvode tejto kapitoly sme analyzovali riadiace prvky modelu vzducholode, na základe ktorých sme navrhli model riadenia. Návrhu a realizácii komunikačného podsystému sme sa zaoberali v ďalšej kapitole a riešenie sme rozdelili na komunikačné rozhrania nachádzajúce sa v modeli vzducholode a aplikácii základňovej stanice.

Navrhnuté a realizované podsystémy boli použité v nasledujúcej časti práce, v ktorej sme sa venovali vývoju aplikácie základňovej stanice implementujúcej ostatné časti práce a sprístupňujúcej používateľovi prostriedky pre riadenie vzducholode.

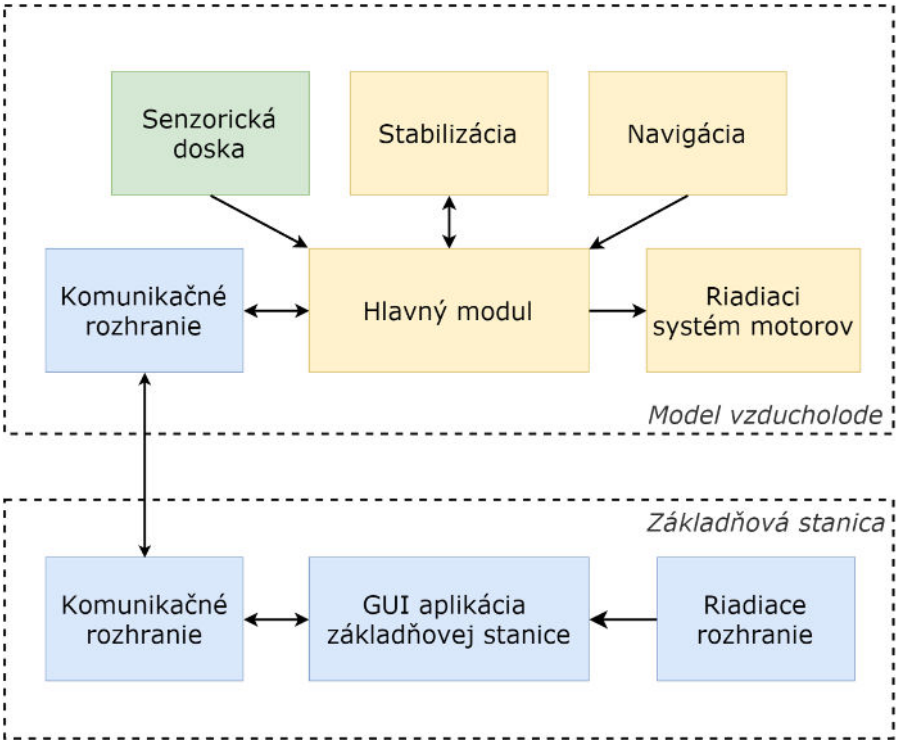
Riešenia jednotlivých podsystémov a výsledné riešenie aplikácie bolo potrebné podrobiť testovaniu a z toho dôvodu sme navrhli a zrealizovali testovacie rozhranie,

v ktorom sme mohli otestovať funkcionalitu realizovaných podsystémov v emulovanom prostredí. Výsledne riešenie sme tiež otestovali v reálnych podmienkach.

V závere sme zhodnotili splnenie cieľov práce a riešení jednotlivých čiastkových podsystémov. Okrem toho sme predložili námety na možné vylepšenia alebo rozšírenia práce pri ďalšom pokračovaní danej témy.

1 Špecifikácia práce

Špecifikácia zadania našej diplomovej práce vychádzala z projektu „Vzducholoď ako prostriedok na riešenie úloh IKT“ na Katedre technickej kybernetiky, na ktorom sme sa podieľali. Projekt bol zameraný na zostrojenie funkčného prototypu malej vzducholode, ktorá by bola využitá na riešenie rozmanitých úloh týkajúcich sa problematiky informačno-komunikačných technológii. Projekt pozostával z viacerých technologických častí, na vypracovaní ktorých pracovali viacerí študenti.



Obrázok 1 Špecifikácia úloh projektu

Úlohy, na ktorých sme pracovali, sú na obrázku 1 vyznačené modrou farbou a týkali sa:

- Návrhu a realizácie komunikačného kanála medzi modelom vzducholode a základňovou stanicou
- Návrhu a realizácie riadiaceho rozhrania na strane používateľa
- Návrhu a realizácie aplikačného rozhrania základňovej stanice v podobe používateľskej aplikácie

Návrh celého projektu pozostával z viacerých oddelených častí, ktoré spolupracovali a poskytovali si dáta. Pri návrhoch týchto rozhraní, ktoré spolupracujú s ostatnými komponentmi systému, sme sa so zvyšnými spolupracovníkmi dohodli na spoločnej forme

a interpretácii dát. Týmto sme zabránili problému nekompatibility jednotlivých komponentov systému po jeho realizácii.

1.1 Ciele práce

Hlavným cieľom práce bola realizácia riadiaceho systému, ktorý by umožňoval používateľovi vzdialené ovládanie vzducholode. Jednotlivými čiastkovými úlohami práce, vyplývajúcimi z hlavného zadania, boli:

- Výber vhodných hardvérových prostriedkov na realizáciu jednotlivých časti práce
- Návrh a implementácia architektúry riadiaceho a komunikačného podsystému na vybranom modeli vzducholode
- Návrh a implementácia programového vybavenia riadiaceho a komunikačného podsystému
- Návrh a realizácia programového riešenia základňovej stanice v podobe aplikácie s grafickým užívateľským rozhraním
- Praktické overenie funkčnosti realizovaného systému a uskutočnenie experimentálnych meraní

Pri vypracovávaní jednotlivých úloh sme k nim pristupovali systematicky a snažili sa ich riešiť postupne, od teoretických aspektov problému, cez analýzu nášho problému, až k návrhu a realizácii jeho riešenia. Postup pri riešení jednotlivých úloh sme špecifikovali v nasledujúcej kapitole.

1.2 Postup riešenia

Postup riešenia práce pozostával z chronologického spracovania úloh jednotlivých zadaní. K riešeniu sme museli pristupovať chronologicky a postupne sa venovať jednotlivým zadaniam, pretože výsledné čiastkové riešenia boli podmieňujúce pre nasledujúce časti práce. Z toho dôvodu je teoretická časť aj analýza rozdelená a nachádza sa v tých častiach práce, ktoré sa zaoberajú riešením danej problematiky.

V úvodnej časti práce sme sa venovali technickej analýze a výberu hardvérových prostriedkov, ktoré boli súčasťou práce pri riešení nasledujúcich úloh.

Vypracovanie riadiaceho a komunikačného podsystému sme si rozdelili na dva samostatné celky. V prvom rade to bola analýza riadiacich prvkov vzducholode a návrh

všeobecného modelu riadenia, ktorý sme následne implementovali v práci. Druhým riešeným celkom bolo vypracovanie komunikačného podsystému. K problému sme pristúpili analýzou a výberom vhodných komunikačných prostriedkov. Na ich základe sme navrhli vhodnú architektúru komunikačného podsystému, ktorá bola implementovaná pre konkrétne riešenie.

V ďalšej časti práce sme sa venovali vývoju aplikácie základňovej stanice. Postup riešenia sme rozdelili na naštudovanie si teoretických aspektov vývoja a výberom prostriedkov riešenia. Druhou časťou bola analýza aplikácie, jej požiadaviek, možností riešenia a definovanie aspektov pre používateľa. Následne sme pokračovali návrhom a implementáciou aplikácie vychádzajúc zo zadefinovaných požiadaviek na aplikáciu a tiež z návrhu riadiaceho a komunikačného podsystému, ktoré boli implementované do aplikácie.

V poslednej časti práce sme sa venovali testovaniu, ktoré sme rozdelili na dve časti. Prvou bol vývoj testovacieho rozhrania, ktorým sme boli schopný otestovať funkcionalitu všetkých realizovaných častí práce v emulovanom prostredí. Druhou časťou bolo samotné testovanie funkcionality implementovaných podsystémov prostredníctvom testovacieho rozhrania a tiež v reálnych podmienkach.

2 Hardvérové prostriedky

Pri vypracovávaní riešenia pre jednotlivé podsystémy sme okrem softvérových prostriedkov využívali aj hardvérové. Niektoré prostriedky boli pre našu prácu dané a našou úlohou bolo porozumieť vlastnostiam a zohľadniť ich pri vývoji riešenia. Príkladom takéhoto prostriedku bol model vzducholode, s ktorým sme pracovali. Ďalšími prostriedkami boli zariadenia použité v riadiacom a komunikačnom podsystéme, ktoré sme zvolili a ich technologické vlastnosti určovali možnosti použitia.

2.1 Vzducholoď

Vzducholoď je motorom poháňaný prostriedok ľahší ako vzduch (*LTA*), ktorý je možné riadiť. Moderné vzducholode používajú pokročilé technológie akými sú kompozitné materiály, moderné elektronické systémy a najnovšie poznatky v oblastiach stability, ovládania a aerodynamiky. Konvenčný typ vzducholode je v podstate nízko rýchlostným prostriedkom, ktorého energetická efektivita a odolnosť môže byť niekoľkonásobne väčšia, ako pri iných ťažších lietajúcich prostriedkoch. Dodatočný náklad vzducholode je vo všeobecnosti limitovaný vztlakovými možnosťami použitého plynu pri špecifikovaných klimatických podmienkach.

Ako pri všetkých lietajúcich objektoch, ich výkonnosť závisí v značnej miere na ich veľkosti a charakteristikách vztlakových mechanizmov (krídla, rotory). Špecifické charakteristiky rôznych typov objektov predstavujú výzvu pri ich navrhovaní a ovládaní. Energetická efektívnosť vzducholode poskytuje neporovnateľné väčšie možnosti letu počas dlhých časových úsekov. Vzducholoď sa vznáša v atmosfére vďaka vztlaku vytváraného plynom a nachádza sa vo vzduchu od momentu jej napustenia plynom, až pokiaľ postupné unikanie plynu cez materiál plášťa alebo núdzové uvoľnenie plynu nezapríčinia pokles vztlaku a zostup vzducholode. Táto vlastnosť dovoľuje objektom ľahším ako vzduch vykonávať nepretržitú činnosť iba s veľmi malou spotrebou energie. Prostriedky ľahšie ako vzduch sú atraktívnym riešením pre mnohé aplikácie vyžadujúce nepretržitú vzdušnú prítomnosť. Takýmito úlohami je environmentálne monitorovanie, skúmanie, sledovanie a iné aplikácie, v ktorých sa vyžaduje let v nízkej výške, vznášanie sa nad bodom záujmu a schopnosť vzlietnuť aj pristávať vertikálne. [1]

Ovládateľné objekty ľahšie ako vzduch majú esenciálne dva letové režimy:

- *Aerostatické vznášanie* – rýchlosť objektu je v porovnaní s okolitým vzduchom zanedbateľná a motory sa používajú iba na korigovanie aktuálnej pozície vzducholode. Schopnosť vznášať sa je nesmierne užitočná v situáciách, kedy sa vyžaduje konštantná pozícia.
- *Aerodynamický let* – relatívna rýchlosť medzi objektom a jeho okolitým prostredím je výrazná. Výrazné aerodynamické sily pôsobia na objekt, a v značnej miere dominujú nad aerodynamickými silami vytváranými napájacím systémov vzducholode.

Súčasťou práce bol aj konkrétny typ modelu vzducholode, pre ktorý bol náš systém vyvíjaný. Vlastnosti modelu definovali zákonitosti, na ktoré sme museli prihliadať pri návrhoch jednotlivých podsystémov. Vzducholoď bola modelom *outdoorovej* vzducholode s dĺžkou 4,5 m. Nosným plynom je hélium, ktorého požadované množstvo je 14 m^3 na naplnenie vzducholode. Pri takomto objeme je dodatočná nosnosť vzducholode 1,5 kg. Plášť vzducholode je tvorený kompozitnou plastovou fóliou bez výstuže. Ďalšími prvkami sú riadiace a stabilizačné komponenty v podobe zadných krídel a gondoly obsahujúcej elektroniku a hlavné motory.

Obrázok 2 Model vzducholode

2.2 Komunikačný prostriedok *GSM* modul

Pri navrhovaní komunikačného podsystému sme sa rozhodli využiť viaceré dostupné technológie pre prenos dát. Ako jednu z týchto technológii sme sa rozhodli použiť *GSM* technológiu, ktorá prostredníctvom *GSM* modulu a *SIM* karty sprostredkováva pripojenie ku sieti. Pripojenie *GSM* modulu k mikropočítaču je možné viacerými perifériami.

Niektoré mikropočítače obsahujú integrovaný *GSM* modul, ale častejšie je možné ho zabezpečiť v podobe nadstavbovej dosky pre mikropočítač, *USB* modulu alebo *PCI-Express* modulu. My sme sa rozhodli použiť modul vo forme *mini PCI-Express*, nakoľko nám to mikropočítač dovoľoval vyvedenými perifériami. Konkrétne sme použili modul *Dell Wireless 5540*.

Obrázok 3 *GSM* modul *Dell Wireless 5540*

Komunikácia s modulom sa uskutočňuje cez sériové komunikačné rozhranie prostredníctvom množiny *AT* príkazov kompatibilných s konkrétnym modulom.

2.3 Používateľský prostriedok ovládania

Ako jeden z používateľských prostriedkov na riadenie modelu vzducholode sme sa rozhodli použiť herný ovládač *Playstation 4 Dualshock.* Pre ovládač ako riadiaci prostriedok sme sa rozhodli z dôvodu implementácie viacerých spôsobov riadenia a ovládač sme považovali za vhodné riešenie.

Obrázok 4 *Playstation 4 Dualshock* ovládač

Pripojenie ovládača k počítaču sa uskutočňuje cez *USB* port alebo bezdrôtovo cez *Bluetooth* technológiu, ak to operačný systém podporuje. V počítači je zariadenie

rozpoznávané ako vstupné zariadenie, ktorého hardvérové komponenty generujú vstupy. Komponenty tvoriace ovládač sú:

- 6-osé snímanie pohybu (3-osý akcelerometer, 3-osý gyroskop)
- Dvojica analógových *joystickov*
- 2 analógové tlačidlá (označované *L2* a *R2*)
- 12 digitálnych tlačidiel
- 4 digitálne smerové tlačidlá
- Dvojbodový kapacitný *touchpad* [2]

3 Riadiaci podsystém

Implementácia metód riadenia objektov ľahších ako vzduch sa dá rozdeliť do dvoch kategórií: tradičné riadiace metódy a pokročilé (autonómne) metódy riadenia. Tradičné riadiace metódy zabezpečujú riadenie prostredníctvom klasických riadiacich algoritmov, ktoré sú jednoducho implementovateľné a poskytujú spoľahlivé správanie riadenia. Sú založené na lineárnom riadiacom modeli. Väčšina pokročilejších riadiacich metód je postavená na vysoko nelineárnom a časovo premenlivom riadiacom systéme. Návrh lineárnych riadiacich metód bol používaný pri riešení problémov riadenia lietajúcich objektov po mnoho rokov. Jedným z dôvodov, prečo je možné riadiť vzducholoď prostredníctvom lineárnych riadiacich prvkov je to, že vzducholoď sa správa lineárne počas takmer celého jej letu. Zdrojmi nelineárnych vplyvov na vzducholoď sú aerodynamické sily generované pri nízkej letovej rýchlosti a vysokých uhloch ovplyvnenia. [3]

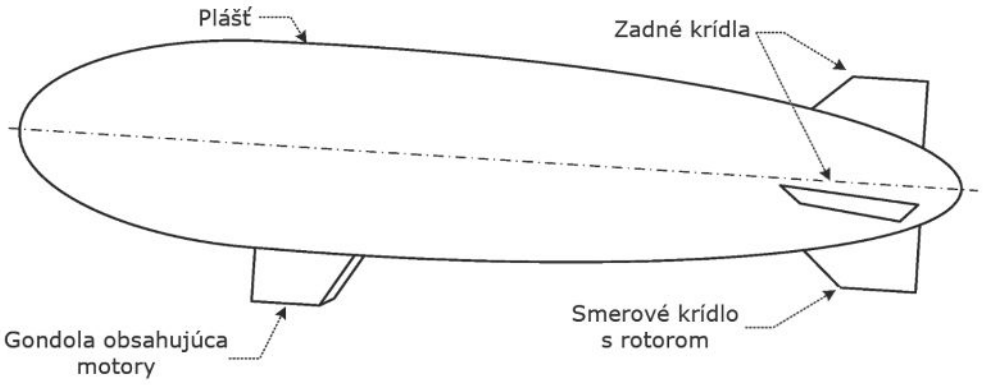
Vyhodnotenie letovej výkonnosti a manévrovacích možností vzducholode je dôležitým aspektom analýzy pri návrhu modelu riadenia. Manévrovacie schopnosti charakterizujú schopnosť lietajúceho objektu meniť svoju stálu letovú trajektóriu prostredníctvom rotácie vzhľadom k letovému vektoru vzducholode. Tieto aspekty neboli predmetom riešenia našej práce. Predmetom riešenia nášho modelu riadenia bolo sprostredkovanie týchto riadiacich možností používateľovi základňovej stanice. V prvom rade bolo potrebné identifikovať riadiace prvky modelu vzducholode, ktorých riadenie by malo byť sprostredkované používateľovi.

3.1 Analýza riadiacich prvkov modelu vzducholode

Hlavnou úlohou nami navrhovaného modelu riadenia bolo poskytnúť riadiace rozhranie používateľovi základňovej stanice, ktoré by umožňovalo generovanie riadiacich príkazov a ich prenos na model vzducholode pre ovládanie jej jednotlivých komponentov. Podľa špecifikácie modelu vzducholode, s ktorým sme pracovali, sme identifikovali riadiace prvky:

- Rotor nachádzajúci sa na chvostom krídle modelu vzducholode umožňujúci riadenie smerovania vychýlením krídla a jeho aktiváciou
- Rotor nachádzajúci sa v gondole modelu, regulujúci rotáciu dvojice bočných motorov vystupujúcich z rámu gondoly

- Dvojica hlavných rotorov regulujúcich pohyb a výšku modelu podľa ich natočenia
- Chvostové krídla upravujúce aerodynamiku a správanie modelu vzducholode pri pohybe



Obrázok 5 Schematické znázornenie riadiacich prvkov vzducholode

Na každý z identifikovaných riadiacich prvkov bola požiadavka ich riadenia analógovo. Reguláciu rýchlosti otáčania rotorov jednotlivých prvkov sme museli zahrnúť pri návrhu modelu riadenia a samotných riadiacich prostriedkov aplikácie.

3.2 Analýza riadiacich prostriedkov základňovej stanice

Požiadavka na riadenie vzducholode vychádza ako podnet zo strany používateľa. Na strane používateľa sa nachádza základňová stanica tvorená aplikáciou, implementujúca prostriedky riadenia. Našou úlohou bolo navrhnúť a zakomponovať metódy riadenia týmito prostriedkami do aplikácie. Pri návrhu aplikácie sme sa nechceli spoliehať iba na jednu metódu riadenia, preto sme sa rozhodli pre implementáciu viacerých riadiacich prostriedkov, založených na jednotnom modeli riadenia.

Medzi natívne hardvérové riadiace prostriedky počítača môžeme zaradiť klávesnicu a myš (resp. *touchpad* v prípade notebookov), ktoré sú štandardom pri ovládaní akéhokoľvek aplikačného rozhrania. Za ďalšie vstupné ovládacie zariadenia môžeme považovať rôzne typy herných ovládačov alebo ovládačov na letecké simulátory, ktoré niektorými svojimi vlastnosťami predstihujú natívne. Tieto zariadenia nemôžeme považovať za súčasť každého počítača, preto sme ich implementáciu ako riadiaceho prostriedku považovali za sekundárny spôsob riadenia. V primárnom návrhu sme sa

obmedzili na riadenie s pomocou myši a klávesnice. Predmetom analýzy riadiacich prostriedkov boli ich možnosti pre riadenie prvkov modelu vzducholode.

V prípade klávesnice dokáže jeden kláves zabezpečiť pohyb alebo reguláciu otáčok jedného prvku v jednom smere. Na ovládanie jedného riadiaceho prvku by bola potrebná aspoň dvojica kláves. Štandardne je výstupom z klávesy iba digitálna informácia, ktorú by bolo potrebné transformovať do analógovej reprezentácie. Takáto transformácia by sa dala uskutočniť snímaním dĺžky stlačenia klávesu, čo by definovalo analógovú úroveň, alebo aktivita stláčania dvojice kláves by bola snímaná a menila hodnotu nejakej číselnej premennej, ktorej aktuálna hodnota by definovala analógovú úroveň. Pri vysokom rozsahu analógových úrovni by bola problémom druhej metódy časová náročnosť na väčšiu zmenu. Riešením by bolo možno spojenie oboch metód, ale ani v takom prípade by nebolo možné navrhnúť vhodné nástroje. Kláves ako digitálny prvok nie je vhodným riešením pre aktívne riadenie analógových komponentov. Ich funkcionalita by mohla byť využitá pri potvrdzovaní riadiacich príkazov alebo iných digitálnych vstupoch.

Druhým natívnym prostriedkom je využitie možností myši. Pri pohybe myši máme k dispozícii dve osi voľnosti, ktoré sa dajú použiť na ovládanie prvkov vzducholode. Pri ovládaní aplikácii je myš aktívnym prvkom riadenia samotnej aplikácie, preto je pri jej použití na generovanie riadiacich príkazov potrebné jasné definovanie charakteru jej používania. Návrh riadenia modelu vzducholode pohybom myši v celej aplikácii by bol nevhodný a preto by bolo vhodnejším riešením využitie možností myši na ovládanie virtuálnych komponentov aplikácie, ktoré by generovali analógové riadiace príkazy. Návrh a charakter týchto virtuálnych komponentov by závisel na použitom programovacom jazyku aplikácie. Pohyb myši alebo jej stláčanie by definoval zmenu hodnôt týchto komponentov. Takéto riešenie sa zdá vhodnejšie z hľadiska používateľského a tiež implementačného.

Sekundárnou metódou riadenia bolo využitie externého ovládača. V našom prípade sme ako prostriedok použili ovládač *Playstation 4 Dualshock,* ale vo všeobecnosti sú riadiace komponenty ovládačov rovnaké pre rôzne typy ovládačov. Hardvérovo môžeme komponenty ovládača rozdeliť na páčkový *joystick*, ktorý poskytuje analógové informácie v dvoch osiach, digitálne tlačidlá a analógové tlačidlá. Pri analógových tlačidlách je možnosť získavania riadiacich dát vo forme digitálnej aj analógovej. V prípade nami

použitého ovládača bol rozsah hodnôt analógových prvkov <0,65535>. Komponenty ovládača sú vhodnými prostriedkami na generovanie analógových riadiacich príkazov.

3.3 Návrh modelu riadenia

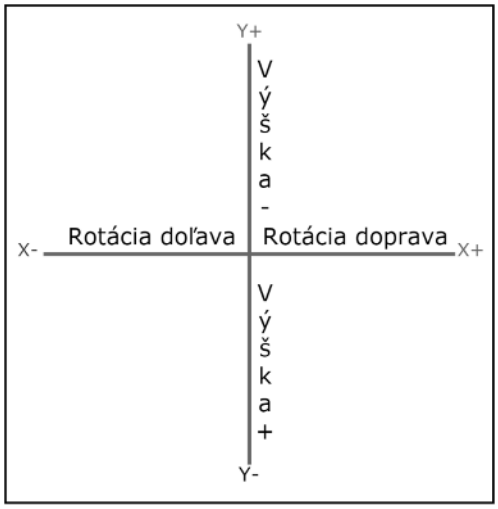
Model riadenia definuje spôsoby riadenia prvkov modelu vzducholode a aké vlastnosti a charakter ovládania by mali obsahovať jednotlivé komponenty riadenia na strane používateľa. Pri jeho návrhu sme sa nechali inšpirovať vlastnosťami komponentov ovládača, ktorý sme používali. Pri návrhu bolo tiež potrebné zohľadniť možnosti pohybu vzducholode, na základe jej konštrukcie. Zložky pohybu boli:

- Translačný pohyb dopredu aktiváciou hlavných motorov vzducholode, ktorý ale závisí aj od ostatných riadiacich prvkov. Natočenie týchto motorov definuje aj zmenu výšky a pre určitý uhol ich natočenia iba zmenu samotnej výšky. Ďalšími prvkami, ktoré ovplyvňujú pohyb vpred, sú chvostové krídla modelu vzducholode.
- Rotačný pohyb modelu vzducholode, ktorý sa uskutočňuje v dvoch osiach. Otáčanie okolo vlastnej osi a zmena smerovania vzducholode je prvým rotačným pohybom. Druhý rotačný pohyb sa týka zmeny výšky vzducholode a môže byť viazaný na vychýlenie modelu pre potreby zmeny výšky alebo iba rotáciu hlavných motorov, ktorých aktivácia spôsobí zmenu výšky.

Návrh modelu riadenia pozostával z logického prepojenia riadiacich prvkov modelu vzducholode s možnosťami pohybu vzducholode a vytvorením vzoru, ktorý by sa mal dodržať pri implementácii konkrétneho spôsobu riadenia používateľom. Návrh jednotného modelu riadenia dovoľuje implementáciu rôznych prostriedkov riadenia postavených na rovnakom princípe. Návrh sme rozdelili na jednotlivé čiastkové prvky, ktorými sme interpretovali riadenie jednotlivých prvkov vzducholode.

Rotáciu vzducholode bolo možné uskutočňovať v dvoch osiach, preto sme ako vhodné riešenie považovali ich zoskupenie v jednom riadiacom prvku. Prvok definoval dvojdimenzionálny priestor s dvoma osami voľnosti, v ktorom pohyb v jednej osi definoval smerovú rotáciu a v druhej osi výškovú rotáciu (priamu alebo nepriamu, čo by záležalo už na konkrétnej implementácii). Definovaním takéhoto prvku bolo možné simultánne a jednoducho riadiť rotáciu v oboch osiach ako možno vidieť na schematickom znázornení návrhu na obrázku 6. V neaktívnom stave by sa kurzor prvku nachádzal

v strede a zmenou jeho polohy v niektorej z osí by sa menila analógová hodnota príslušného riadiaceho prvku.



Obrázok 6 Schematický návrh riadiaceho prvku

Dôležitým bolo definovanie správania sa tohto riadiaceho prvku pri generovaní riadiacich príkazov. V prípade smerovej rotácie by sa táto činnosť uskutočňovala počas trvania vychýlenia kurzora z korešpondujúcej osi a miera vychýlenia by definovala rýchlosť otáčania modelu vzducholode. Pre výškovú rotáciu bolo možné definovať dva formy správania, čo vyplýva aj z predchádzajúcej analýzy. V prvom prípade by bolo správanie totožné zo smerovou rotáciou, teda vychýlenie kurzora by definovalo vychýlenie celej vzducholode, na čom by sa muselo podieľať viacero prvkov modelu. Jeden druh riadiaceho príkazu by bol transformovaný na aktiváciu väčšieho počtu riadiacich prvkov vykonávajúcich požadovaný pohyb. Druhou metódou by bola iba rotácia hlavných motorov modelu, ktorá by nijako nezmenila vychýlenie ani výšku vzducholode, čo by sa muselo uskutočniť iným prvkom. Pre obe druhy rotácii by bola zadefinovaná nasledujúca vlastnosť a to, že pri zastavení aktívneho používania prvku sa hodnota kurzora vychýlenia nastaví v oboch osiach do nečinného stavu.

Pohyb vzducholode je definovaný aktiváciou hlavných motorov, pri ktorých analógová hodnota určuje rýchlosť. Navrhnutý riadiaci prvok by teda mal dovoľovať meniť svoju hodnotu v nejakom definovanom rozsahu, ktorého minimálna hodnota (najlepšie definovaná ako 0) by znamenala nečinnosť motorov. Zvyšovanie tejto hodnoty by lineárne zvyšovalo rýchlosť otáčok motorov až do maximálnej hodnoty rozsahu, čo by korešpondovalo s maximálnym výkonom motorov. Podľa charakteru riadiacej metódy by

malo byť jednoduché udržiavať hodnotu prvku na stabilnej úrovni aj pri používaní ostatných riadiacich prvkov.

Riadiaci prvok na ovládanie vychýlenia chvostových krídiel mal z principiálneho hľadiska rovnaké vlastnosti ako prvky pre rotáciu samotného modelu. Vychýlenie kurzora z normálneho stavu by bolo transformované na zmenu vychýlenia krídiel. Na rozdiel od ostatných rotačných prvkov sa požadovalo, ak to bolo možné, aby hodnota vychýlenia zostávala v rovnakej úrovni aj po skončení používania prvku a nenastavovala sa automaticky do neutrálneho stavu.

Takýmto spôsobom boli zadefinované vlastnosti jednotlivých riadiacich prvkov modelu riadenia z pohľadu používateľa, ktoré sa požadovali implementovať pre každú použitú metódu riadenia. Výstup z každého riadiaceho prvku mal presne definovanú formu, čo zabezpečovalo uniformnosť riadiacich príkazov každej metódy.

4 Komunikačný podsystém

Komunikačný podsystém predstavuje kompletné komunikačné rozhranie zabezpečujúce spojenie medzi modelom vzducholode a základňovou stanicou. Jeho úlohou je sprostredkovávať komunikáciu a výmenu dát medzi systémom na vzducholodi a aplikáciou. K vývoju komunikačného podsystému nebolo pristupované ako k samostatnej oddelenej jednotke, ale ako k jednej súčasti celého systému, preto sme pri výbere prostriedkov vychádzali z vlastností iných podsystémov, ktoré sú vypracované v ostatných častiach práce. Patrilo tam napríklad zohľadnenie vlastností riadiaceho podsystému a aplikačného rozhrania základňovej stanice. Podsystém sme rozdelili na tri časti:

- **Komunikačné rozhranie modelu vzducholode** definujúce softvérové a hardvérové vybavenie mikropočítača zabezpečujúce úlohy komunikačnej rutiny pri komunikácii s aplikáciou základňovej stanice
- **Komunikačné rozhranie základňovej stanice** definujúce programové vybavenie zabezpečujúce úlohy komunikačnej rutiny pri komunikácii s modelom vzducholode
- **Komunikačný kanál** definujúci prepojenie oboch rozhraní a ich vzájomnú komunikáciu

Dôležitým aspektom komunikačného podsystému bolo zadefinovanie požiadaviek, ktoré sa od neho vyžadovali a tiež podmieňovali výber prostriedkov pri jeho vývoji. Týmito požiadavkami boli:

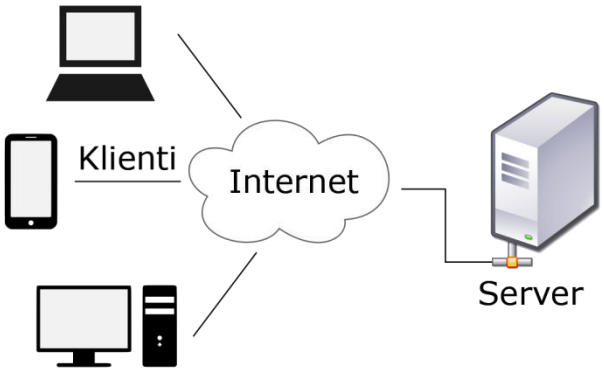
- Primárne bezdrôtová komunikácia (sekundárne aj priame komunikačné prepojenie pre potreby testovania)
- Neustály obojsmerný tok dát medzi vzducholoďou a aplikáciou
- Implementácia prostredníctvom viacerých technológii s odlišnými komunikačnými vlastnosťami
- Jednoduchý prístup a manipulácia s komunikačnými rozhraniami

4.1 Výber a analýza komunikačných prostriedkov

4.1.1 Architektúra klient – server

Ako vhodným riešením pre vzťah komunikačného rozhrania vzducholode a základňovej stanice bolo použitie modelu architektúry klient – server. Model klient – server definuje distribuovanú štruktúru rozdelenú na poskytovateľa zdrojov a služieb, nazývaného server

a užívateľa služieb, nazývaného klient. Server a klient nachádzajúci sa najčastejšie na oddelených hardvérových platformách, komunikujú spolu cez počítačovú (internetovú) sieť. Server je charakterizovaný ako pasívny prvok, ktorý čaká na iniciáciu komunikácie zo strany klienta. Klient je aktívnym prvkom, ktorý iniciuje a tiež ukončuje komunikáciu so serverom. Vzťah server – klient môže byť tiež definovaný ako *1 k n*, teda viacero klientov dokáže iniciovať a komunikovať so serverom zároveň. [4]



Obrázok 7 Architektúra klient – server

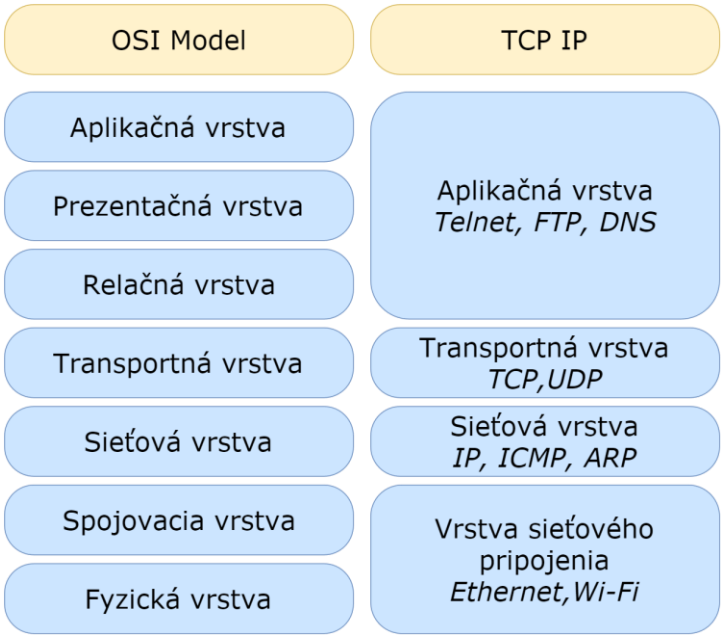
Z charakteristiky tohto modelu jasne vyplývalo, akým spôsobom bolo vhodné definovať jednotlivé prvky systému. Model vzducholode je aktívny počas celého cyklu jeho používania vo vzduchu, bez rozdielu, či je aktívne riadený alebo ponechaný na vznášanie sa na jednom mieste. Taktiež senzorické a kontrolné dáta generované modelom vzducholode sú požadované aplikáciou základňovej stanice. Z týchto faktov jednoznačne vyplýva, že vzducholoď sme zvolili ako serverovú časť komunikačného podsystému, komunikáciu s ktorou iniciuje základňová stanica ako klientska časť. Klient generuje riadiace príkazy, ako požiadavky na riadenie vzducholode. V našom prípade bol vzťah definovaný ako jeden klient k jednému serveru, pretože riadenie prostredníctvom dvoch klientov zároveň by nebolo z praktického hľadiska vhodné.

4.1.2 Komunikačné protokoly

Komunikačný protokol definuje systém pravidiel, ktoré dovoľujú dvom alebo viacerým entitám komunikačného systému vzájomný prenos informácii. Tieto pravidlá alebo štandardy definujú význam, syntax a synchronizačnú rutinu komunikácie. Protokol môže byť implementovaný formou hardvéru, softvéru alebo ich kombináciou. Viacero protokolov popisuje rôzne aspekty samotnej komunikácie. Skupina takýchto protokolov je

navrhnutá na vzájomnú spoluprácu ako balík protokolov. Štruktúra vzťahov medzi jednotlivými protokolmi vyplýva z referenčného modelu.

Model OSI je abstraktný referenčný model charakterizujúci a štandardizujúci komunikačné funkcie telekomunikačných a počítačových systémov bez ohľadu na ich vnútornú štruktúru a technológiu. Jeho cieľom je kooperácia rozdielnych komunikačných systémov použitím štandardných protokolov. Model rozdeľuje komunikačný systém na sedem abstraktných vrstiev. Najvyššia aplikačná vrstva špecifikuje protokoly a rozhrania používané používateľskými aplikáciami a najnižšia fyzická vrstva popisuje prenos informácii (bitov) cez fyzické média. Medzinárodná štandardizačná organizácia prijala tento model ako základný referenčný model, na ktorom sú založené konkrétne modely akým je napríklad balík internetových protokolov. [5]



Obrázok 8 Porovnanie referenčných modelov

Balík internetových protokolov je referenčný model tvorený skupinou komunikačných protokolov používaných na internete a podobných počítačových sieťach. Nazývaný je tiež *TCP/IP* model podľa dvoch najpoužívanejších protokolov – Protokolu riadenia prenosu (*TCP*) a Internetového protokolu (*IP*). Model *TCP/IP* ako skupina komunikačných protokolov definuje komunikáciu špecifikovaním ako majú byť dáta obalené, adresované, prenášané, smerované a prijímané. Funkcionalita je špecifikovaná v štyroch abstraktných vrstvách, pre ktoré sú definované konkrétne komunikačné protokoly. Porovnanie referenčných modelov *OSI* a *TCP/IP* je možno vidieť na obrázku 8, kde je znázornené

vzájomné prekrývanie vrstiev a tiež konkrétne protokoly jednotlivých vrstiev *TCP/IP* modelu. [6]

TCP/IP model je tvorený vrstvami, od najnižšej:

- **Vrstva sieťového rozhrania** definuje fyzický aspekt komunikácie, konvencie použitého média a signálu. Zároveň špecifikuje prenos paketov fyzickou vrstvou. Príkladom protokolov patriacich do tejto vrstvy sú napríklad *Ethernet, Wi-Fi.*
- **Sieťová vrstva** definuje transport dát v rámci siete alebo medzi viacerými sieťami a smerovanie paketov sieťou. Príkladom asi najpoužívanejšieho protokolu vrstvy je protokol *IP*.
- **Transportná vrstva** definuje protokoly špecifikujúce spoľahlivosť a zabezpečenie prenosu dát. Protokoly vrstvy tiež určujú, ktorej aplikácii patria dáta. Medzi najpoužívanejšie protokoly tejto vrstvy patria *TCP* a *UDP* protokol.
- **Aplikačná vrstva** je tvorená protokolmi používanými väčšinou aplikácii, ktoré poskytujú služby výmeny aplikačných dát prostredníctvom internetovej siete.

Najpoužívanejšími protokolmi sieťovej a transportnej vrstvy sú protokoly *IP* a *TCP* (alternatívou k protokolu *TCP* je protokol *UDP*), princípy a vlastnosti ktorých nám definovali spôsob ich použitia pri návrhu nášho riešenia. Ich použitie sme uprednostnili pred inými typmi hlavne z dôvodu ich dostupnosti, podpory a jednoduchej implementácie v rôznych programovacích jazykoch.

4.1.2.1 Internetový protokol (IP)

Internetový protokol (*Internet Protocol*) je základný komunikačný protokol sieťovej vrstvy prenášajúci dátové datagramy medzi vzdialenými počítačmi v sieti. Úlohou internetového protokolu je doručenie paketov od adresáta k destinácii príjemcu výlučne na základe *IP* adresy príjemcu špecifikovanej v hlavičke paketu. Z toho dôvodu definuje *IP* paketovú štruktúru, ktorá obaľuje doručované dáta a tiež definuje metódu adresovania. Metóda adresovania má dve funkcie – identifikáciu príjemcov a poskytovanie služby logického smerovania. Dve hlavné verzie *IP*, ktoré sa používajú, sú *IPv4*, ktorá je dominantnou verziou na internete a jej nasledovník *IPv6*.

IP adresa je identifikátor priradený každému zariadeniu pripojenému do *TCP/IP* siete, ktorá sa používa na lokalizovanie a identifikovanie prvkov siete. Vo verzii protokolu *IPv4*

je adresa 32-bitové číslo, ktoré sa z praktického hľadiska delí na štyri 8-bitové čísla v desiatkovej sústave (rozsah 0-255) oddelené bodkou (napr. 127.0.0.1).

4.1.2.2 Protokol riadenia prenosu (TCP)

Protokol riadenia prenosu (*Transmission Control Protocol*) je spojovo orientovaný, spoľahlivý komunikačný protokol transportnej vrstvy prenášajúci dáta vo forme bajtového toku (*streamu*) dát. V balíku internetových protokolov tvorí *TCP* vrstvu medzi *IP* protokolom pod ním a aplikáciami nad ním.

Aplikácie posielajú sieťou toky dát, ktoré *TCP* rozdeľuje do segmentov s vhodne zvolenou veľkosťou. *TCP* následne posúva výsledné pakety internetovému protokolu na doručenie prostredníctvom siete. Okrem toho protokol vykonáva kontrolu, aby zabezpečil, že sa žiaden paket nestratí alebo nepoškodí a to spôsobom, že každému paketu priradí poradové číslo, ktoré na strane príjemcu protokol kontroluje. Prostredníctvom tohto poradového čísla sa tiež zabezpečuje usporiadanie doručených dát v správnom poradí. Po prijatí dát posiela príjemca potvrdenie (*acknowledgement*) adresátovi o úspešnom prijatí paketov.

4.1.2.3 Používateľský datagramový protokol (UDP)

Používateľský datagramový protokol (*User Datagram Protocol*) je minimalistický protokol transportnej vrstvy, orientovaný na datagramy. *UDP* poskytuje veľmi jednoduché rozhranie medzi sieťovou vrstvou a aplikačnou vrstvou. Neposkytuje žiadne záruky doručenia správ a odosielateľ si o už raz odoslaných datagramoch neudržuje žiadne informácie. Protokol pridáva iba kontrolné súčty a schopnosť roztrieďovať *UDP* pakety medzi spustené aplikácie na počítači.

4.1.3 Sieťové (internetové) sockety

Sieťový *socket* predstavuje rozhranie medzi aplikáciou a internetovou sieťou, ktoré sa používa na komunikáciu medzi procesmi. Nakonfigurovaná aplikácia dokáže prenášať dáta prostredníctvom *socketu* a tiež ich jeho prostredníctvom prijímať zo strany príjemcu. Pre jadro operačného systému je *socket* koncovým bodom komunikácie. Aplikácia pracuje so *socketom* ako so súborovým popisovačom (*file descriptor*), ktorý dovoľuje aplikácii čítanie a zapisovanie z alebo do siete. V modeli klient – server prebieha vzájomná komunikácia čítaním a zapisovaním do *socketového* popisovača (*socket descriptor*).

Implementácia *socketov* existuje vo viacerých formách v závislosti od rozdielnych protokolových balíkov a protokolov v nich definovaných. V našom prípade sme pracovali s modelom *TCP/IP*, ktorého korešpondujúci typ *socketov* sme použili. Medzi základné typy *socketov* pre *TCP/IP* model patria prúdové (*stream*) *sockety* a datagramové *sockety.* Prúdové *sockety* využívajú protokol *TCP* a poskytujú spoľahlivú službu prenosu bajtového toku dát. Datagramové *sockety* využívajú protokol *UDP* a teda poskytujú nespoľahlivú datagramovú službu prenosu dát, ktorá dovoľuje aplikácii poslať individuálnu správu až do dĺžky 65500 bajtov. Prúdové a datagramové *sockety* sú podporované aj inými protokolmi, ale v našej práci sme sa zamerali na ich využitie a preto nám protokoly *TCP* a *UDP* postačovali.

Sockety sa používajú na komunikáciu medzi procesmi a najčastejšie je táto komunikácia založená na modeli klient – server, v ktorej klient a server predstavujú dva oddelené procesy. Výmena správ cez internetovú sieť sa uskutočňuje prostredníctvom *Socket API. Socket API* je aplikačné rozhranie poskytované operačným systémom umožňujúce aplikačným programom kontrolovať a používať sieťové *sockety. API* definuje štandardnú metódu prepojenia dvoch procesov, lokálne alebo prostredníctvom siete. Aplikačné rozhranie je najčastejšie založené na *Berkeleyho socketovom* štandarde, ktorý *sockety* popisuje práve vo forme súborových popisovačov, vychádzajúc z *Unix* filozofie, že všetko predstavuje súbor. [7]

TCP/IP komunikácia prostredníctvom *socketov* sa uskutočňuje medzi dvoma koncovými bodmi. Koncový bod je definovaný *IP* adresou a číslom portu, ktoré definujú unikátny jednoznačne daný koncový bod.

4.2 Návrh architektúry komunikačného podsystému

Návrh komunikačného podsystému bol podmienený teoretickými aspektmi, ktoré sme analyzovali v predchádzajúcej časti. Vlastnosti jednotlivých popísaných prostriedkov (architektúra klient – server, *TCP/IP* model a *sockety*) boli vhodné pre návrh riešenia. Pri návrhu riešenia sme využili dve rozdielne technológie pre fyzický prenos dát, ktoré obe podporujú nami vybrané prostriedky vyšších vrstiev. Týmito technológiami boli *Wi-Fi* a *GSM*, ktoré poskytujú bezdrôtový prenos dát. *Wi-Fi* je štandard umožňujúci zariadeniam pripojiť sa na bezdrôtovú lokálnu sieť, ktorá je vytváraná iným zariadením. V našom prípade by takýmto zariadením mohol byť počítač, v ktorom sa nachádza aplikácia základňovej stanice alebo tretie zariadenie, ku ktorému by sa pripájala základňová stanica

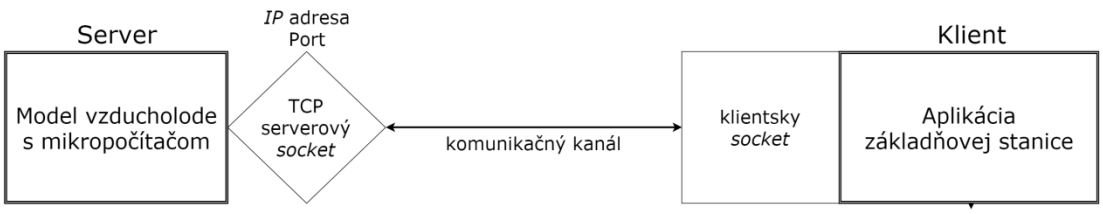
a tiež model vzducholode. Dosah *Wi-Fi* siete nie je veľký a pohybuje sa radovo v desiatkach metrov, preto je táto technológia vhodná pri využití modelu vzducholode vo vnútornom prostredí alebo interiéri. Z dôvodu vyššieho dosahu vo vonkajšom prostredí sme sa rozhodli implementovať komunikáciu aj prostredníctvom *GSM* technológie.

Globálny systém mobilných komunikácii (*GSM*) predstavuje štandard sieťovej komunikácie mobilných zariadení. Pre komunikáciu prostredníctvom tejto technológie je potrebný *GSM* modul nachádzajúci sa v zariadení a *SIM* karta operátora, ktorá poskytuje služby pripojenia k sieti. Možnosti komunikácie prostredníctvom *GSM* modulu sú dané množinou príkazov, s ktorými dokáže konkrétny modul pracovať.

Podmienkou pre fungovanie komunikačného kanála bola prítomnosť týchto technológii v mikropočítači nachádzajúcom sa na modeli vzducholode. V našom prípade sme pri navrhovaní komunikačného rozhrania pracovali s integrovaným *Wi-Fi* modulom a *GSM* modulom vo forme *mini PCI-Express*. Pre každú z týchto technológii sme vytvorili samostatné komunikačné rozhranie komunikujúce prostredníctvom *TCP socketov.*

4.2.1 Návrh architektúry komunikačného kanála

Architektúru komunikačného kanála sme navrhli podľa vzťahu klient – server. Mikropočítač nachádzajúci sa na modeli vzducholode bol definovaný ako serverová časť spojenia, ktorej úlohou bola inicializácia *TCP/IP socketu* a prepnutie sa do pasívneho čakacieho stavu, v ktorom sa čaká na iniciatívu zo strany klienta. Úloha komunikačného rozhrania je iba jednou z mnohých, ktoré uskutočňuje mikropočítač, preto jeho implementácia a vykonávanie nesmelo ovplyvňovať vykonávanie jeho ostatných úloh. Aplikácia základňovej stanice bola implementovaná vo forme klientskej časti komunikačného rozhrania, ktorej úlohou bolo vytvorenie klientskeho *socketu* a pripojenie sa k serverovej časti prostredníctvom *IP* adresy a portu serverového *socketu.* Podoba komunikačného kanála dovoľovala obojstranný prenos dát a jej blokovú schému možno vidieť na obrázku 9.

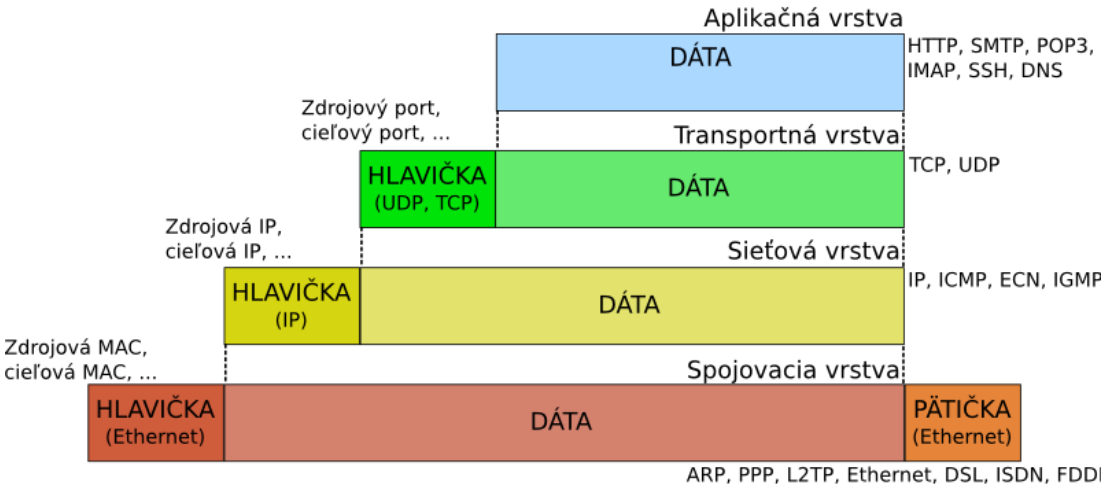


Obrázok 9 Bloková schéma architektúry

Úlohou serverovej časti bolo poskytovať senzorické dáta zo senzorickej dosky pripojenej k mikropočítaču a tiež kontrolné dáta informujúce o stave modelu vzducholode. Úlohou základňovej stanice bolo prijímanie a sprostredkovanie týchto dát používateľovi, ale tiež generovanie riadiacich príkazov, ktoré sa prenášali na vzducholoď. Vlastnosti komunikačného kanála, vyplývajúce z použitia *TCP* protokolu, zaručovali spoľahlivosť prenosu dát a ich doručenie v správnom poradí.

4.2.2 Návrh tvaru dátových paketov

Návrh tvaru dátového paketu v aplikačnej vrstve bol časťou návrhu komunikačného rozhrania, ktorou sme zabezpečili kompatibilitu serverovej a klientskej časti pri výmene dát, pretože sme sa nemohli spoliehať na uniformitu dát poskytovaných senzorickou doskou a mikropočítačom. Rovnako bolo potrebné spracovať riadiace príkazy na strane základňovej stanice do podoby paketu. Za zapuzdrenie dát a ich prenos zodpovedali komunikačné protokoly a z nášho pohľadu boli relevantné iba samotné dáta aplikačnej vrstvy, ktoré boli generované naším systémom. Na obrázku 10 je znázornené, akým spôsobom sú dáta zapuzdrované pri ich prenose a ktoré komunikačné protokoly sú za to zodpovedné.



Obrázok 10 Zapuzdrenie dát v TCP/IP

Aplikačné dáta bolo možné rozdeliť do dvoch kategórii podľa zdroja, ktorý ich generuje. Aplikácia základňovej stanice prijíma väčšinu dát z modelu vzducholode a sama generuje iba riadiace príkazy. Povahu a množstvo riadiacich dát definovali vlastnosti a riadiace prvky vzducholode, ktoré sme analyzovali v kapitole zaoberajúcej sa návrhom

riadiaceho podsystému. Riadiaci paket by mal podľa návrhu riadenia pozostávať zo štyroch riadiacich údajov a jeho schematický návrh možno vidieť na obrázku 11.

Obrázok 11 Schematický návrh riadiaceho paketu

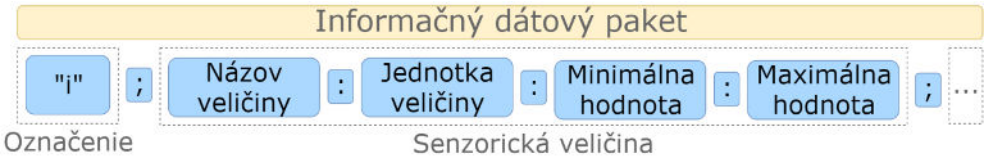
Čiastkové riadiace príkazy sú získavané z riadiaceho podsystému základňovej stanice a spájané do jedného celku, v ktorom sú jednotlivé príkazy oddelené znakom dvojbodky. Tento oddeľovací znak sa používa na rozlíšenie jednotlivých údajov na strane modelu vzducholode.

Dáta generované na strane modelu vzducholode pochádzajú z viacerých zdrojov a bolo možné ich rozdeliť do dvoch základných skupín. Prvou skupinou boli dáta generované senzormi senzorickej dosky a druhou informačné dáta o stave vzducholode, kde patrili vychýlenie, poloha a smerovanie modelu. Dáta druhej kategórie pochádzali zo senzorov nachádzajúcich sa na konkrétnom mikropočítači použitom ako riadiaca jednotka a preto sme vedeli jednoznačný počet a vlastnosti týchto dát. Prvá skupina obsahovala senzorické dáta, ktoré boli viazané na použitú senzorickú dosku, ale počet týchto dát sa mohol meniť v závislosti od celkového počtu senzorov, počtu aktívnych senzorov a tiež zmeny senzorickej dosky alebo pripojenia ďalšej. Do úvahy bolo potrebné zahrnúť aj periodicitu poskytovania dát jednotlivými senzormi dosky a tiež senzormi mikropočítača. Z toho dôvodu sme sa rozhodli nevytvoriť statický paket obsahujúci všetky známe dáta ako v prípade riadiaceho paketu. Pri návrhu sme sa tiež snažili vyhnúť posielaniu rôznych paketov pre jednotlivé kategórie dát. Preto sme hlavný paket rozdelili na čiastkové pakety, ktoré sú spájané do výsledného paketu pri posielaní. Čiastkové pakety sme definovali ako:

- Informačný paket obsahujúci informácie o vlastnostiach senzorických dát, ktoré budú prenášané do aplikácie základňovej stanice
- Senzorický paket obsahujúci aktuálne dáta zo senzorov
- Kontrolný paket obsahujúci informácie o aktuálnom stave modelu vzducholode

Úlohou informačného paketu je poskytnúť aplikácii základňovej stanice informácie o senzorických dátach, ktoré budú prenášané. Paket sa posiela iba jedenkrát pri vytvorení komunikačného spojenia alebo pri požiadavke na zmenu typov prenášaných senzorických dát. Paket je rozdelený na čiastkové elementy, kde každý z elementov popisuje jeden typ

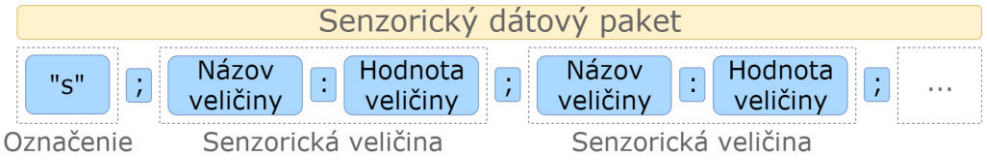
senzora. Schematickú podobu takéhoto elementu v kontexte k celému paketu možno vidieť na obrázku 12.



Obrázok 12 Schematický návrh informačného paketu

Jeden element obsahuje informácie o názve senzorickej veličiny, jej fyzikálnej jednotky a dodatočných údajov jej predpokladaného rozsahu hodnôt oddelené znakom dvojbodky. Elementy sú navzájom oddelené bodkočiarkou a ich počet nie je obmedzený. Pred samotným zoznamom elementov sa na začiatku paketu nachádza označenie paketu, ktoré bolo zahrnuté do návrhu, aby sa zjednodušilo rozpoznávanie typov čiastkových paketov na strane aplikácie.

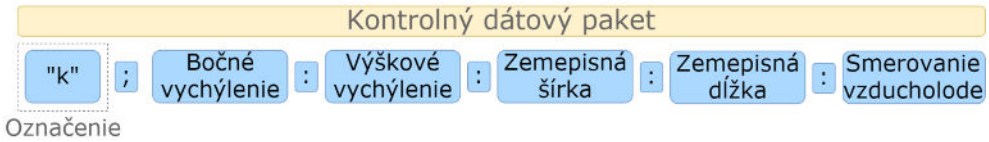
Senzorický paket prenáša aktuálne hodnoty z aktívnych senzorov, ktoré sú sprostredkovávané používateľovi aplikácie. Schematickú podobu paketu možno vidieť na obrázku 13.



Obrázok 13 Schematický návrh senzorického paketu

Začiatok paketu obsahuje označenie senzorického paketu a rovnako ako informačný typ je rozdelený na zložky predstavujúce jednotlivé senzory. Zložka paketu je tvorená názvom senzorickej veličiny a jej hodnotou. Na rozdelenie dát v pakete sa používajú rovnaké znaky - dvojbodky a bodkočiarky. Poradie jednotlivých zložiek nie je staticky dané a algoritmy spracovania týchto dát využívajú dvojicu názvu veličiny a jej hodnoty na ich správne vyhodnotenie.

Posledným typom je kontrolný paket sprostredkujúci používateľovi informácie o aktuálnom stave vzducholode. Na obrázku 14 je znázornený schematický návrh podoby paketu aj s jeho zložkami.



Obrázok 14 Schematický návrh kontrolného paketu

Spojenie čiastkových paketov sa uskutočňuje prostredníctvom znaku zvislej čiary a pri spracovávaní celku sa ku každému čiastkovému paketu pristupuje samostatne. Spracovanie týchto paketov bolo implementované do réžie aplikácie základňovej stanice. Prenášaný paket môže byť zložený z jedného čiastkového paketu alebo aj viacerých, čo nijako neovplyvní jeho správne vyhodnotenie.

4.2.3 Návrh komunikačného rozhrania modelu vzducholode

Návrh a implementáciu aplikačného rozhrania pre komunikáciu na strane vzducholode sme uskutočnili prostredníctvom skriptu v programovacom jazyku *Python*. Pre tento jazyk sme sa rozhodli z dôvodov jednoduchosti používania jazyka, jeho podpore pre viaceré platformy a teda výsledné riešenie nie je viazané iba na konkrétny použitý systém a dostupnosti knižníc pre komunikáciu prostredníctvom siete. Skript komunikačného rozhrania bol navrhnutý ako autonómny program, ktorý nevyžaduje interakciu zo strany používateľa a preto v ňom nebolo zakomponované žiadne používateľské rozhranie. Z dôvodu využitia dvojice technológii pre prenos dát bolo potrebné implementovať dvojicu skriptov, pretože prístup k sieti a riadenie prenosu dát nie je identické a vyžaduje si využitie rozdielnych prostriedkov.

4.2.3.1 Komunikačný skript pre Wi-Fi technológiu

Prvý navrhnutý komunikačný skript využíval technológiu *Wi-Fi* a bol kompatibilný aj pri pripojení mikropočítača cez *Ethernet* pre potreby testovania. Program bol implementovaný v programovacom jazyku *Python*, ktorý nám poskytoval vhodné prostriedky vo forme knižníc. Funkcie *socketovej* komunikácie sme implementovali prostredníctvom *Python* knižnice *socket.py*. Princíp programu spočíva vo vytvorení serverového *socketu*, jeho inicializácie a čakaní na pripojenie zo strany klientskej časti. Štruktúra programu inicializujúca serverový *socket* prostredníctvom metód knižnice mala formu:

- Vytvorenie *TCP socketu* prostredníctvom metódy *socket()*, ktorej parametre *AF_INET* definujú *socket* komunikujúci cez internetový protokol (*IP*) a *SOCK_STREAM* ako prúdový *socket*

- Zadefinovanie *IP* adresy a portu *socketu* prostredníctvom metódy *bind()*
- Prepnutie sa do čakacieho módu prostredníctvom metódy *listen()*, kedy sa vykonávanie skriptu pozastaví a očakáva sa nadviazanie spojenia klientskou aplikáciou
- Pri nadviazaní spojenia sa vykoná metóda *accept()* a program začne vykonávať nekonečný cyklus, v ktorom prebieha komunikačná rutina
- Pri ukončení spojenia zo strany klienta sa program vracia do čakacieho stavu a je možné sa k nemu opätovne pripojiť.

Návrh štruktúry nekonečného cyklu, v ktorom sa uskutočňuje všetka komunikácia, pozostával z niekoľkých častí. Senzorické dáta sú ukladané do textových súborov a rovnako bola požiadavka na ukladanie prichádzajúcich riadiacich príkazov do súboru. Preto sa ešte pred začiatkom nekonečného cyklu otvoria požadované súbory, z ktorých sa bude čítať a zapisovať. Z návrhu typov dátových paketov vyplynulo, že boli potrebné viaceré súbory - súbor obsahujúci informačné dáta senzorov, súbory senzorických a kontrolných dát a súbor pre riadiace príkazy. V cykle sa periodicky čítajú tieto súbory, v ktorých sa nachádzajú aktuálne dáta. Samotná komunikácia sa uskutočňuje prostredníctvom metódy knižnice na čítanie *recv()* a posielanie dát *sendall().* Načítané dáta sú spracované a uložené do súboru. Posielané dáta sú najprv načítané zo súborov, spracované, spojené do jedného paketu a následne prenášané. [8]

4.2.3.2 Komunikačný skript pre GSM technológiu

Druhý komunikačný skript bol založený na *GSM* technológii, ktorá vyžadovala rozdielny prístup pri návrhu. *GSM* modul je samostatná hardvérová jednotka, ktorá sprostredkováva komunikáciu prostredníctvom *AT* príkazov. *AT* príkazy sú posielané cez sériové komunikačné rozhranie a definujú inštrukcie pre modul. Každý príkaz začína prefixom *AT*, čím sa modul informuje o začiatku inštrukcie. Za ním nasleduje názov samotného príkazu s parametrami. Každý modul je definovaný inštrukčnou sadou *AT* príkazov, ktoré definujú komunikačné možnosti daného modulu. V našom prípade sme vybrali a pracovali s modulom, ktorý podporoval *TCP sockety*. Na rozdiel od prvého skriptu sme nemohli pracovať s knižnicou *socket.py*, ale všetku inicializáciu a komunikačnú rutinu uskutočniť prostredníctvom *AT* príkazov.

Skript bol implementovaný v jazyku *Python* a sériová komunikácia so zariadením sa uskutočňuje prostredníctvom knižnice *serial.* Parametre pri vytváraní objektu *serial* triedy

definujú zariadenie a prenosovú rýchlosť. Vstupom do sériového rozhrania sú *AT* príkazy a prenášané dáta. Výstupom sú odpovede modulu na *AT* príkazy a tiež riadiace dáta prenesené zo strany klientskej aplikácie. Príprava dát na prenos a ich spracovanie pri čítaní boli implementované rovnakým spôsobom ako v prípade prvého skriptu. Štruktúra programu je principiálne totožná z prvým skriptom, odlišuje sa hlavne využitím iných prostriedkov v podobe *AT* príkazov. Zoznam použitých príkazov aj s ich očakávanými odpoveďami a významom možno vidieť v tabuľke 1.

Tabuľka 1 Zoznam použitých AT príkazov

***AT* príkaz**	**Odpoveď**	**Význam**
AT	OK	Testovací príkaz pre kontrolu funkčnosti *GSM* modulu
AT+CPIN=”****”	OK	Príkaz na odomknutie *SIM* karty (**** predstavuje *PIN* karty)
AT+CREG?	+CREG: 0,1	Príkaz na kontrolu registrácie *SIM* karty
AT+CGSOCKCONT=	OK	Príkaz nastaví typ internetového protokolu a *Access Point Network (APN)*
AT+CSOCKAUTH=	OK	Príkaz nastaví autentifikáciu prostredníctvom mena a hesla
AT+NETOPEN=“***“,&&	Network opened	Príkaz vytvorí *socket*, kde *** definujú typ *socketu* a && definujú port
AT+SERVERSTART	OK	Príkaz spustí *TCP* server s vytvoreným socketom
AT+IPADDR	*IP* adresa	Príkaz si vyžiada *IP* adresu aktívneho *socketu*
AT+LISTCLIENT		Príkaz si vyžiada zoznam pripojených klientov v podobe ich *IP* adresy a portu
AT+ACTCLIENT=*	OK	Príkaz aktivuje špecifikovaného klienta podľa parametra *.
AT+TCPWRITE=*	>	Príkaz sprostredkováva prenos dát
AT+CLOSECLIENT=*	OK	Príkaz odpojí špecifikovaného klienta podľa parametra *.
AT+NETCLOSE	OK	Príkaz na odpojenie *socketu*

V úvode skriptu sme si zadefinovali jednotlivé *AT* príkazy vo forme statických textových premenných. Obsah skriptu sme rozdelili na dve časti. Návrh prvej časti obsahuje inicializáciu *GSM* modulu a vytvorenie serverového *socketu.* Pri vykonávaní skriptu sa po vytvorení sériového spojenia s *GSM* modulom skontroluje správna funkčnosť modulu príkazom *AT*. Následne je potrebná kontrola *SIM* karty, ktorá sa najprv odblokuje *PIN* kódom karty prostredníctvom príkazu *AT+CPIN* a potom sa príkazom *AT+CREG* skontroluje, či je karta registrovaná v sieti a je možné ju používať. Nastavenia *GSM* modulu sa uskutočnia príkazmi *AT+CGSOCKCONT* a *AT+CSOCKAUTH*. Po úspešnej

inicializácii sa vytvorí a spustí serverový *socket* s požadovanými vlastnosťami príkazmi *AT+NETOPEN* a *AT+SERVERSTART*.

Návrh druhej časti obsahoval programové riešenie obsluhy komunikácie. Pri iniciácii komunikácie je klient zaradený do zoznamu pripojených klientov, ale nie je automaticky aktivovaný. Aktivácia klienta je v réžii skriptu, ktorý v nekonečnej slučke čaká na pripojenie klienta prostredníctvom príkazu *AT+LISTCLIENT,* ktorá vracia zoznam pripojených klientov. Pri zistení klienta v zozname pripojených klientov je pripojenie aktivované. Čítanie klientskych dát sa uskutočňuje rovnakým spôsobom ako čítanie odpovedí na *AT* príkazy z *GSM* modulu. Prenos dát klientovi sa uskutočňuje príkazom *AT+TCPWRITE*, v ktorom sa špecifikuje dĺžka prenášaných dát. Očakávaná odpoveď modulu je znak >, kedy je možné cez sériové rozhranie preniesť samotné dáta. Ukončenie komunikácie s klientom sa uskutočňuje príkazom *AT+CLOSECLIENT*. [9]

5 Aplikácia základňovej stanice

Aplikácia základňovej stanice predstavuje softvérové riešenie riadenia modelu vzducholode a reprezentácie dát poskytovaných vzducholoďou z pohľadu používateľa. Pri návrhu aplikácie sme kládli dôraz na splnenie požiadaviek vychádzajúcich z predchádzajúcich kapitol a tiež z všeobecne zaužívaných požiadaviek a postupov pre návrh takéhoto druhu aplikácie. Samotný vývoj tejto aplikácie sme z praktického hľadiska rozdelili na nasledujúce časti:

- Teoretická analýza návrhu a vývoja aplikácie
- Analýza požiadaviek na aplikáciu, dostupných prostriedkov a možností pri návrhu
- Návrh programovej architektúry a vzhľadu aplikácie
- Implementácia aplikácie vo vybranom programovacom jazyku

5.1 Teoretické aspekty vývoja aplikácii

5.1.1 Používateľské rozhranie

Aplikácie môžeme z pohľadu spôsobu interakcie s nimi rozdeliť do dvoch kategórii. Prvou kategóriou sú konzolové aplikácie, kde interakcia prebieha v konzolovom okne pomocou preddefinovaných príkazov, ktoré používateľ zadáva z klávesnice. Vstupy a výstupy takéhoto druhu aplikácie sú vo väčšine prípadov textové bez nejakých grafických aspektov. Pri vývoji aplikácii pre vlastnú potrebu, alebo aplikácii pre operačný systém bez grafického rozhrania, sa takýto druh aplikácii stále používa.

Druhou kategóriou, ktorá je v dnešnej dobe už štandardom a tvorí prevažnú väčšinu vyvíjaných aplikácii, sú aplikácie s grafickým používateľským prostredím. Aplikácie takéhoto druhu sú tvorené grafickými prvkami, ktoré zjednodušujú vstupnú interakciu a tiež sprehľadňujú výstup pomocou týchto grafických nástrojov. Vstupná interakcia môže byť uskutočňovaná myšou, klávesnicou alebo inými externými hardvérovými prostriedkami ak to aplikácia dovoľuje. Ďalším dôležitým aspektom pri vývoji takýchto aplikácii je ich dizajn, teda návrh vzhľadu, s ktorým prichádza do styku používateľ. Cieľom účinného dizajnu je umožniť jednoduché a efektné používanie aplikácie. Proces vývoja aplikácie musí spojiť technickú realizáciu logickej časti aplikácie s vizuálnymi elementmi užívateľského rozhrania do kompaktného celku.

Jednou z úloh našej práce bolo navrhnúť a implementovať programové riešenie pre grafické rozhranie systému na strane používateľa. Z definície tejto úlohy vyplýva návrh aplikácie implementujúcej grafické používateľské rozhranie ako riešenie aplikácie základňovej stanice. Analýza požiadaviek na aplikáciu tiež jasne stanovuje grafické komponenty ako vhodné nástroje pre potreby riadenia modelu vzducholode a zobrazovania dát. Práve členitosť aplikácie na viaceré logické celky a vysoká interakcia používateľa pri ovládaní jednotlivých jej častí vylučujú použitie konzolovej aplikácie. Vývoj aplikácie s grafickým používateľským prostredím (ďalej GUI aplikácia) sa dá rozdeliť na dizajn grafickej časti aplikácie a architektúru programovej časti, ktorá beží na pozadí. Programová časť vykonáva všetky špecifikované úlohy, či už autonómne podľa naprogramovanej vnútornej logiky alebo na podnet používateľa.

5.1.2 Dizajn vizuálnej časti aplikácie

Dizajnové riešenie aplikácie tvorí jedinečnú podobu aplikácie, s ktorou prichádza používateľ do styku. K programovej časti nemá prístup a zásadou by malo byť, aby nemusel rozumieť tejto časti aplikácie, bežiacej na pozadí. Od dizajnu sa preto vyžaduje, aby bol čo najjednoduchší, najzrozumiteľnejší, ale zároveň pokrýval všetky programové aspekty aplikácie a sprístupňoval ich prostredníctvom vizuálnych prvkov. Pod pojmom vizuálny dizajn aplikácie rozumieme činnosť zameranú na vývoj grafického rozhrania aplikácie. Tento proces by mal prebiehať v niekoľkých fázach a samotný výsledok by mal spĺňať niekoľko základných princípov, aby navrhnuté riešenie bolo vhodné na používanie.

Pri návrhu vizuálneho rozhrania je potrebné dodržať určitý postup krokov, ktoré sa môžu líšiť podľa špecifikácie projektu, ale z teoretického hľadiska by mali byť všetky zahrnuté v návrhu. Kroky zahrnuté v procese návrhu:

- Návrh funkčnej špecifikácie (vychádzajúc z programovej časti a možností programovacieho jazyka)
- Užívateľská analýza
- Návrh kostry vizuálnej časti (návrh prototypu a rozloženia vizuálnych komponentov)
- Návrh grafického dizajnu (návrh konkrétneho vizuálneho vzhľadu aplikácie)
- Implementácia grafického návrhu (implementovanie návrhu do programovej časti aplikácie a prepojenie vizuálnych prvkov s logickou časťou aplikácie) [10]

5.1.3 Programová časť aplikácie

Programová časť aplikácie zahŕňa časť programu, ktorá beží na pozadí a obsahuje celkovú logiku aplikácie a tiež implementáciu vizuálnej stránky programu. Výber vhodného programovacieho jazyka a s ním súvisiaceho vývojového prostredia bolo dôležitou časťou procesu vývoja aplikácie. Tento výber potom následne ovplyvnil smerovanie vývoja celej aplikácie. Objektovo orientované programovacie jazyky, s ktorými sme mali nejaké predchádzajúce skúsenosti a považovali sme ich ako potenciálnu možnosť, boli:

- C++
- C#
- Java

Každý z týchto programovacích jazykov ponúka široké možnosti pri vývoji rôznych druhov aplikácii a v každom z nich je možné navrhnúť GUI aplikáciu, čo závisí na vývojovom prostredí a *frameworku* pre grafické rozhranie. Každý z týchto programovacích jazykov má nejaké prednosti, ale aj nedostatky. Pri rozhodovaní sme vychádzali z našich vlastných poznatkov a skúsenosti. Hlavnými prednosťami programovacieho jazyka C++ je obrovská voľnosť pri návrhu architektúry tried a správe pamäti pri použití smerníkov. Jazyk C++ je odvodený z klasického C jazyka a jeho podpora návrhu grafického rozhrania nie je na takej vysokej úrovni, ako v prípade ostatných programovacích jazykov. Druhou možnosťou bol jazyk C#, ktorý patrí k novším jazykom a pri jeho návrhu sa vychádzalo z C++ a jazyka Java, teda zlučuje prednosti oboch spomínaných jazykov. Treťou možnosťou bol jazyk Java, ktorý je vlastnosťami niekde medzi oboma predchádzajúcimi jazykmi. Jazyk ponúka dostatočné možnosti pri návrhu zložitých projektov a pri návrhu grafického rozhrania ponúkajú *frameworky* Swing alebo AWP bohatú paletu grafických nástrojov. Ďalšou zaujímavou vlastnosťou jazyka je *multiplatformovosť*, čo by uľahčilo vývoj aplikácie nielen pre rôzne platformy, ale pre rôzne zariadenia. Hlavnou prednosťou tohto jazyka z nášho pohľadu boli hlavne doterajšie skúsenosti pri vývoji aplikácii a znalosť možností jazyka, ktoré by sme mohli využiť počas vývoja. Práve pre tieto dôvody sme uprednostnili programovací jazyk Java.

5.1.3.1 Programovací jazyk Java

Programovací jazyk Java patrí k relatívne novším jazykom a dlhodobo patrí medzi piliere jazykov pre vývoj aplikácii. Je to práve vďaka širokým možnostiam, ktoré ponúka.

Niektorými zo základných vlastností, ktoré tiež rozhodli o výbere jazyka a využili sme ich pri vývoji aplikácie boli, že jazyk Java je objektovo orientovaný, platformovo nezávislý, robustný, multivláknový a dynamický. [11]

Aplikáciu sme vyvíjali vo verzii jazyka Java - 1.8.0_91**.** Na spustenie aplikácie je potrebné mať na platforme nainštalovanú *Java Virtual Machine (JVM)*, najlepšie v rovnakej alebo vyššej verzii, ako bola aplikácia vyvíjaná.

5.1.3.2 Vývojové prostredie

Ako vývojové prostredie na návrh a implementáciu aplikácie sme sa rozhodli použiť aplikáciu *NetBeans IDE*, s ktorým sme už mali predchádzajúce skúsenosti pri iných projektoch. *NetBeans IDE* je oficiálnym integrovaným vývojovým prostredím (*IDE*) pre Java 8. Neustále sa vyvíja a ponúka širokú ponuku rôznych nástrojov, šablón a príkladov, ktoré sa dajú využiť pri vývoji aplikácii.

Prostredie editora poskytuje jednoduchý a prehľadný manažment projektov, ich zdrojov a ponúka prehľadný náhľad na rôzne typy dát a informácii o práve vyvíjanom projekte. Ďalej ponúka rôzne analytické a ladiace programy na identifikáciu a opravu bežných chýb, ale tiež analýzu zložitejších problémov. [12]

Z nášho pohľadu bol silným nástrojom programu NetBeans IDE tiež integrovaný modul - dizajnér na návrh grafického rozhrania aplikácie, ktorý okrem vizuálnej stránky návrhu ponúka aj generovanie príslušného zdrojového kódu komponentov. Dizajnér ponúka viaceré *frameworky* na návrh GUI, medzi ktoré patria napríklad *Swing* alebo *AWT*.

5.1.3.3 Nástroj pre návrh grafického rozhrania

Ako nástroj pre návrh vizuálnej aj programovej časti grafického rozhrania sme využili *framework* Swing. Swing je navrhnutý vo forme importovateľných balíčkov (knižníc), ktoré implementujú sadu komponentov pre tvorbu grafických používateľských rozhraní a pridáva široké spektrum grafickej funkcionality a interaktivity do Java aplikácií. Všetky Swing komponenty sú implementované v programovacom jazyku Java a je teda možné ich funkcionalitu rozširovať a doplňovať pre vlastnú potrebu. Swing nie je vo všeobecnosti bezpečný z hľadiska vlákien, čo je potrebné pri vývoji aplikácii zohľadňovať. K všetkým Swing komponentom a odvodeným triedam sa musí pristupovať cez *event dispatching thread*, ktoré spracováva udalosti vznikajúce vo Swing komponentoch aplikácie. Preto

aktualizácia grafických prvkov aplikácie by mala prebiehať výlučne v tomto vlákne, inak by mohlo dôjsť k vzniku nežiaducich stavov alebo nesprávnemu zobrazovaniu dát. [13]

5.2 Analýza návrhu aplikácie

Po analýze teoretických aspektov vývoja aplikácii, výbere programovacieho jazyka a vhodného vývojového prostredia sme pristúpili k analýze problému z konkrétneho hľadiska našej úlohy. Samotnú analýzu sme rozdelili na niekoľko častí podľa aspektov, ktoré sme analyzovali:

- **Analýza logického oddelenia** grafického rozhrania od programového rozhrania aplikácie
- **Analýza požiadaviek na programové rozhranie**
- **Analýza požiadaviek na grafické rozhranie**
- **Analýza dostupných programových prostriedkov**
- **Analýza externých programových prostriedkov**
- **Analýza možnosti používateľa** na prispôsobenie aplikácie

5.2.1 Analýza logického oddelenia aplikácie

Návrh objektovo orientovanej aplikácie pozostáva z tried a závislostí medzi nimi. Pri návrhu aplikácie s GUI je potrebné okrem programového rozhrania zahrnúť aj grafické rozhranie, ktoré je v prípade Swing *frameworku* založené na komponentoch, fungujúcich na princípe udalostí. Interakcia s grafickým rozhraním vytvára udalosti, na ktoré sa očakáva nejaká odozva zo strany grafického rozhrania alebo tiež zmena na strane programového rozhrania. Takáto interakcia sa očakáva aj zo strany programového rozhrania, zmeny v ktorom by mali byť premietnuté na strane grafického rozhrania. Okrem toho, programové rozhranie často krát vykonáva úlohy časovo alebo výpočtovo náročné, ktoré blokujú vykonávanie iných vykonávaných úloh v danom vlákne (kontexte). Vykonávanie takejto úlohy by v prípade jednovláknového systému zapríčinilo nereaktívnosť grafického rozhrania na vznikajúce udalosti. Preto sa ako vhodným riešením javí oddelenie grafického a programového rozhrania a to nielen z dôvodu reaktívnosti, ale tiež sprehľadnenia toku dát medzi rozhraniami. Ďalšou výhodou oddelenia rozhraní je ich čiastočná nezávislosť. Návrhové alebo implementačné zmeny v jednom z rozhraní nemusia zapríčiniť potrebu zmien v druhom rozhraní, čo značne zjednodušuje úpravy a rozširovanie aplikácie.

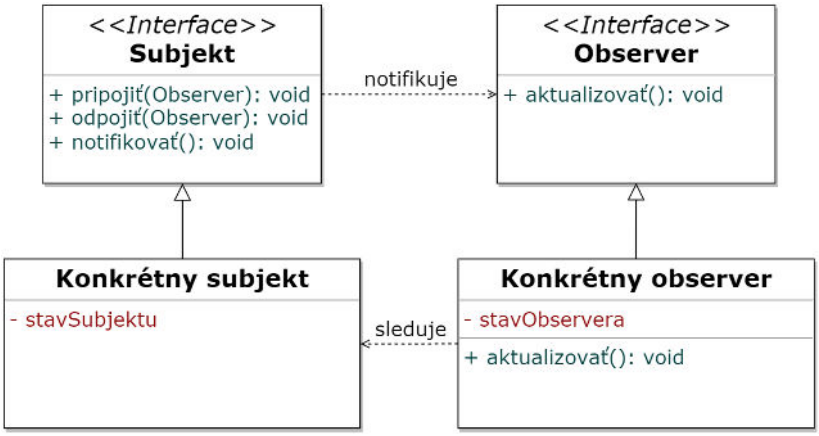
Pri aplikovaní oddelenia rozhraní vzniká teda potreba návrhu komunikačného kanála medzi nimi. Ako riešením takéhoto komunikačného kanála sa javí návrh ďalšieho rozhrania, ktoré bude tzv. mostom medzi grafickým a programovým rozhraním. Jeho úlohou bude iba sprostredkovávať komunikáciu medzi oboma rozhraniami, ktoré nebudú navzájom priamo prepojené. Vhodné je v takom prípade tiež použiť nejaký návrhový vzor, čo je všeobecný spôsob riešenia daného typu problému. Vhodnými návrhovými vzormi sú:

- Návrhový vzor *Observer*
- Návrhový vzor *Model-view-controller*

5.2.1.1 Návrhový vzor *Observer*

Návrhový vzor *Observer* (*pozorovateľ*) je známy ako behaviorálny vzor, ktorý sa používa na definovanie závislosti medzi objektmi počas vykonávania programu. Definuje závislosť medzi objektmi tak, že pri zmene stavu pozorovaného objektu sú všetky závislé objekty informované a aktualizované automaticky.

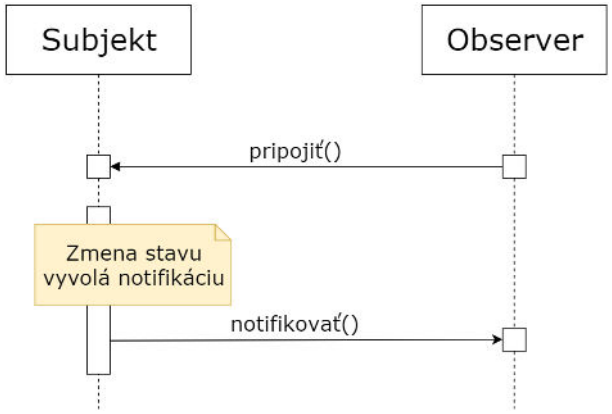
Pri pohľade na všeobecný diagram definície *Observera* na obrázku 15 možno vidieť vzájomné závislosti jednotlivých tried a rozhraní.



Obrázok 15 Diagram vzťahov Observer vzoru

V *Observeri* existuje záujem o zmenách stavu *Subjektu* a registrovanie tohto záujmu v *Subjekte* sa uskutočňuje *pripojením* referencie *Observera*. Pri zmene stavu v *Subjekte*, o ktorý ma *Observer* záujem, sa posiela *notifikačná* správa, ktorá vyvolá vykonanie metódy *aktualizovať* v každom zaregistrovanom *Observeri.* V prípade, že *Observer* už nemá záujem o zmeny v stave *Subjektu*, jednoducho sa *odpojí* a *Subjekt* ho prestane *notifikovať*. [14]

Sekvenčný diagram na obrázku 16 ilustruje registráciu a notifikáciu v praxi.



Obrázok 16 Sekvenčný diagram Observer vzoru

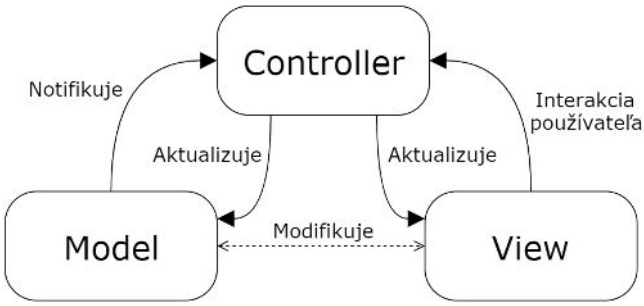
Na notifikáciu a prenesenie dát do *Observera* nie je potrebné, aby mal *Subjekt* prístup priamo k objektu *Observera.* Namiesto toho, všetko je riešené cez spoločné rozhranie a notifikačná metóda zavolá všetky objekty registrované v *Subjekte*. Takýto spôsob je veľmi silným nástrojom, pretože v hociktorej triede je potrebné iba implementovať rozhranie *Observera*, aby jeho objekty mohli byť registrované v *Subjekte.*

V našom prípade by teda grafické rozhranie alebo jeho časti implementovali rozhranie *Observer* a registrovali ho v logickej programovej časti, ktorej zmeny by boli notifikované. Takýmto spôsobom by sme mali zabezpečenú aktualizáciu grafického rozhrania podľa zmien v logickej časti a prichádzajúcich dát.

5.2.1.2 Návrhový vzor *Model-view-controller*

Model-view-controller (ďalej *MVC*) je architektonickým návrhovým vzorom. *MVC* vzor sa aplikuje vo veľkej škále a jeho použitie ovplyvňuje návrh architektúry celej aplikácie. *MVC* predpisuje oddelenie každého prvku aplikácie a ich implementáciu do oddelených komponentov – *Model, View* a *Controller*. Každý z týchto logických komponentov môže obsahovať viaceré triedy, ktorých implementácia musí spĺňať podmienky návrhového vzoru.

Diagram *MVC* vzoru na obrázku 17 schematicky znázorňujúce vzťahy medzi jednotlivými komponentmi návrhového vzoru.



Obrázok 17 Diagram interakcie v MVC vzore

- ***Model*** komponent je centrálnou časťou aplikácie a obsahuje aplikačnú logiku. Patria k nemu triedy, ktoré definujú stav aplikácie, triedy na výpočty a vykonávanie úloh definovaných aplikáciou. Databázová infraštruktúra aplikácie je tiež súčasťou tohto komponentu.
- ***View*** komponent je tvorený vizuálnym používateľským rozhraním aplikácie, ktorý poskytuje nepriamy prístup ku komponentu *Model* a mechanizmy na generovanie udalostí, ktoré modifikujú samotný *Model*. V klasickej implementácii *View* komponent čaká na udalosti z *Model* komponentu, kedy aktualizuje svoj stav, čo poskytuje oddelenie medzi aplikačnou a prezentačnou logikou.
- ***Controller*** komponent vytvára spojenie medzi *View* a *Model* komponentom, spravuje predávanie udalostí generovaných používateľom triedam komponentu *Model*, ktoré následne modifikujú svoje úlohy alebo stav. Rovnako aj zmeny vo vnútornom stave aplikačného rozhrania sú cez *Controller* predávané komponentu *View* na vizualizáciu. [15]

V našom prípade by teda bolo grafické používateľské rozhranie aplikácie *View* komponentom a triedy programového rozhrania by tvorili *Model* komponent. *Controller* komponentom by bola ďalšia nezávislá trieda, ktorá by sprostredkovávala spojenie medzi rozhraniami. Takéto riešenie bolo z hľadiska návrhu vhodnejšie ako *Observer* vzor, pretože programové rozhranie bolo už z hľadiska jeho požiadaviek rôznorodé, čo malo za následok generovanie rôznorodých druhov dát potrebných na vizualizáciu. V prípade *Observer* vzoru by musela každá trieda registrovať grafické rozhranie ako *Observera*, ale požiadavky na aktualizáciu dát by prichádzali z viacerých tried, čo by bolo potrebné zohľadniť v metódach grafického rozhrania. Architektúra *MVC* vzoru tiež rieši problém obojstranného prenosu dát medzi rozhraniami.

5.2.2 Definovanie požiadaviek na programové rozhranie

Požiadavky na programové rozhranie aplikácie vyplývajú z predchádzajúcich častí práce, ktorých implementácia sa vyžaduje v aplikácii základňovej stanice. Tieto požiadavky sme rozdelili na primárne, pretože vyplývajú priamo z jednotlivých úloh zadania a potom sekundárne, ktoré sú odvodené z primárnych alebo doplňujú ich funkcionalitu.

Medzi primárne požiadavky patrili návrh a implementácia:

- Komunikačného rozhrania základňovej stanice podľa navrhnutej architektúry komunikácie s modelom vzducholode
- Riadiaceho rozhrania podľa navrhnutého modelu riadenia
- Spracovania prichádzajúcich senzorických dát do podoby vhodnej na ich vizualizáciu
- Riadiacich prostriedkov generujúcich riadiace príkazy používateľom

Tieto primárne požiadavky sme si zadefinovali ako prioritné a pre úspešné splnenie tejto časti práce sme požadovali minimálne implementáciu týchto požiadaviek vo výslednej aplikácii.

Sekundárne požiadavky sme zadefinovali z dôvodu rozšírenia možností aplikácie, lepšieho využitia prichádzajúcich dát a zlepšenia aplikácie vo všeobecnosti. Medzi tieto sekundárne požiadavky patrili návrh a implementácia:

- Spracovanie senzorických dát alternatívnym spôsobom prostredníctvom grafov
- Uchovávania prichádzajúcich dát a možnosti ich neskoršieho využitia
- Spracovanie a transformácia dát z *GPS* senzora a kompasu
- Možností zálohovania dát
- Možností pre používateľa na prispôsobenie aplikácie

5.2.3 Definovanie požiadaviek na grafické rozhranie

Pri definovaní požiadaviek grafického rozhrania sme vychádzali z požiadaviek na programové rozhranie, pretože programový návrh definuje typy a množstvo zobrazovaných dát a požiadavky na generovanie potrebných dát zo strany používateľa. Požiadavky sme opäť rozdelili na primárne a sekundárne, čo priamo odráža programové rozhranie.

Medzi primárne požiadavky patrili dizajnový návrh a implementácia:

- Vizualizácie komunikačného rozhrania
- Vizualizácie riadiacich prostriedkov používateľa a ich spätnej väzby
- Vizualizácie senzorických dát

Ako sekundárne požiadavky sme definovali dizajnový návrh a implementáciu:

- Vizualizácie senzorických dát prostredníctvom grafov
- Vizualizácie uchovaných dát a prostriedkov na ich zálohovanie
- Vizualizácie polohy a smerovania modelu vzducholode podľa spracovaných dát z *GPS* senzora a kompasu
- Prostriedkov pre používateľa na prispôsobenie aplikácie

5.2.4 Analýza dostupných programových prostriedkov

Pri návrhu a implementácii definovaných požiadaviek aplikácie sme mohli pracovať iba s prostriedkami a nástrojmi, ktoré nám ponúkal nami vybraný programovací jazyk. Preto sme sa rozhodli rozanalyzovať natívne možnosti jazyka pri riešení požiadaviek na aplikáciu. Analýzu sme aplikovali samostatne na každú kladenú požiadavku, primárnu aj sekundárnu, a snažili sa nájsť prostriedok riešenia:

- Komunikačné rozhranie základňovej stanice malo byť podľa špecifikácie klientskou časťou *socketovej* komunikácie s modelom vzducholode ako serverovou časťou. Java ponúka funkcionalitu *socketov*, ktorá je implementovaná v balíčku *java.net.Socket* a bola vyhovujúca pre naše potreby
- Riadiace rozhranie a riadiace prostriedky tvoria nástroje používateľa na riadenie modelu vzducholode. Špecifikácia modelu riadenia definovala dve metódy riadenia.
    - Prvou metódou bolo riadenie natívnymi hardvérovými prostriedkami ako sú myš alebo klávesnica, ktoré majú v jazyku Java široké implementačné možnosti v podobe rozhraní, ako napríklad *MouseListener* alebo *KeyListener*
    - Druhou alternatívou bolo riadenie pomocou externého hardvérového prostriedku (herného ovládača). Java neponúka dostatočnú podporu pre obsluhu takéhoto nástroja. Rozhodli sme sa preto použiť externú knižnicu *JInput*, ktorá by mala poskytnúť nástroje na riešenie

- Spracovanie prichádzajúcich senzorických dát je výhradne problémom algoritmickým a teda neboli potrebné nijaké externé nástroje. Na vizualizáciu dát v jednoduchej podobe bolo možné použiť *framework* Swing, ale na ich reprezentáciu prostredníctvom grafov neponúka Java žiadne prostriedky. Takúto vizualizáciu by bolo možné si navrhnúť, ale pre naše potreby sme sa rozhodli použiť externú knižnicu *JFreeChart*, ktorá ponúka bohatú paletu možností na vykresľovanie grafov.
- Na uchovávanie dát a ich zálohovanie mimo aplikácie ponúka Java viaceré možnosti.
- Spracovanie prichádzajúcich dát z *GPS* senzora obsahujúcich aktuálnu zemepisnú šírku a dĺžku bolo možné spracovať dvoma spôsobmi. Najjednoduchšia by bola vizualizácia týchto údajov v číselnej podobe, čo ale nemá veľkú výpovednú hodnotu a neposkytuje používateľovi požadovanú informáciu v čitateľnom tvare. Druhou možnosťou bola transformácia týchto údajov do podoby mapy aktuálnej podoby. Java natívne neponúka nástroje na riešenie takejto úlohy a návrh vlastného riešenia by bol nad rámec zadania a potrieb riešenia. Preto sme sa rozhodli použiť *Google Static Map API* službu, ktorá nám vedela poskytnúť mapu aktuálnej polohy podľa špecifikovaných parametrov.

Ako prostriedok riešenia grafického rozhrania sme použili *framework* Swing, ktorého možnosti sme považovali za dostatočné pre naše potreby. V prípade, ak by niektorý z nástrojov neposkytoval potrebnú funkcionalitu, *Swing* nám dovoľovala ich rozšírenie.

5.2.5 Analýza externých programových prostriedkov

Pri návrhu aplikácie sme použili nielen natívne prostriedky vývojového prostredia, ale rozhodli sme sa niektoré problémy riešiť využitím externých prostriedkov. Týmito prostriedkami boli knižnice, ktorými sme rozširovali možnosti jazyka o už navrhnuté a implementované nástroje a internetová služba spoločnosti *Google*, ktorá nám sprístupnila jednoduchý prístup k zobrazovaniu máp.

Pred samotným zakomponovaním týchto nástrojov do aplikácie bolo potrebné porozumieť ich funkcionalite a spôsobu používania.

5.2.5.1 *JInput* knižnica

JInput je multiplatformovou Java *API* knižnicou, ktorá poskytuje nástroje na zisťovanie prítomnosti a dopytovanie sa na vstupné zariadenia. Medzi rozpoznávané vstupné zariadenia patrí myš, klávesnica a rôzne typy ovládačov. Rozsah podporovaných zariadení závisí na operačnom systéme. Na operačnom systéme Windows knižnica *JInput* využíva funkcionalitu *API DirectInput*. Princíp využívania knižnice pozostáva z využitia funkcionalít poskytovaných jednotlivými triedami a skladá sa z viacerých krokov, ktoré je potrebné zahrnúť do implementácie. Logický postup využitia knižnice je:

- získanie zoznamu všetkých podporovaných zariadení z predvoleného prostredia systému a uloženie zoznamu zariadení
- vyfiltrovanie dostupných zariadení podľa špecifických parametrov, ktorými sa dajú odlíšiť rôzne typy podľa ich vlastností a výber požadovaného zariadenia
- čítanie aktuálnych dát z používaného zariadenia sa uskutočňuje dopytovaním sa zariadenia na jeho komponenty (prvky, ktoré ho tvoria) a ich vložením do zoznamu komponentov
- prechádzaním zoznamu komponentov sa zisťuje aktivácia týchto prvkov, podľa ich charakteru, ktorý môže byť analógový alebo digitálny, čomu je potrebné prispôsobiť ich kontrolu
- čítanie údajov by sa malo uskutočňovať v pravidelných časových intervaloch, aby sa zabezpečila autenticita vstupov zo zariadenia [16]

5.2.5.2 *JFreeChart* knižnica

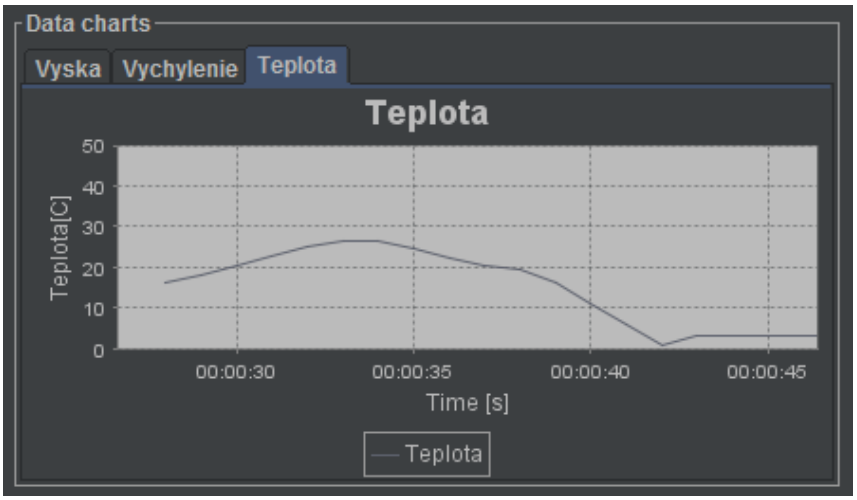
Knižnica *JFreeChart* je nástrojom na tvorbu grafov v aplikáciách. Štandardne je knižnica dostupná pre platformu Java a na jej správnu funkčnosť sa vyžaduje aspoň verzia *JDK 1.3.1*. Pre jej správnu funkcionalitu sa tiež vyžaduje knižnica *JCommon*, ktorá poskytuje nástroje využívané knižnicou *JFreeChart* pri niektorých nastaveniach vizuálnej časti grafov.

Štruktúra knižnice poskytuje vytváranie širokého spektra rôznych druhov grafov, medzi ktoré partia stĺpcové, bodové, čiarové a tiež koláčové grafy. Každý typ grafu je tvorený troma aspektmi, ktoré spolu tvoria celok. Týmito aspektmi grafu sú:

- Dátová množina grafu, ktorá predstavuje štruktúrovanú reprezentáciu dát grafu, ktoré sa zobrazujú vo vybranom type grafu

- Samotný objekt grafu, predstavuje objekt typu *JFreeChart*, ktorý zastrešuje dátovú množinu a poskytuje nástroje na špecifikáciu spôsobu zobrazovania, čo zahrňuje funkcie nastavujúce vzhľad a vlastnosti grafu
- Panel grafu, predstavuje špeciálny typ objektu, do ktorého sa graf vkladá a dovoľuje jeho následné vykreslenie v komponentoch knižnice *Swing*

Pri vytváraní statického grafu hodnôt sa v prvom rade definuje dátová množina, v ktorej sa alokujú údaje. Vytvorí sa objekt triedy *JFreeChart* podľa niektorého z konštruktorov triedy *ChartFactory*, ktorá združuje všetky typy grafov a priradí sa mu vytvorená dátová množina. Objekt grafu sa priradí panelu grafu, ktorý sa použije na následné vykreslenie v aplikácii. [17]



Obrázok 18 Príklad dynamického grafu knižnice *JFreeChart*

Knižnica umožňuje okrem generovania statických grafov aj vytváranie dynamických grafov, v ktorých nie je dátová množina statická, ale pravidelne sa aktualizuje. Takýto graf je priebežne prekresľovaný pri zmene dátovej množiny, aby sa zaručila aktuálnosť zobrazovaných údajov. Aktualizovanie dátovej množiny je možné uskutočniť viacerými spôsobmi.

5.2.5.3 *Google Static Maps API*

Google Static Maps API služba dovoľuje zakomponovanie *Google Maps* obrázku do webovej stránky alebo aplikácie bez potreby použitia jazyka *JavaScript* alebo dynamického obnovovania. Služba vytvára mapu na základe *URL* parametrov poslaných cez štandardnú *HTTP* požiadavku a vracia mapu vo forme obrázka, ktorý sa následne zobrazuje.

Služba posiela obrázok (vo formátoch *GIF, PNG* alebo *JPEG*) ako odpoveď na *HTTP* požiadavku. Pre každú z požiadaviek sa špecifikujú parametre, ktoré definujú lokáciu mapy, veľkosť obrázka, priblíženie mapy a umiestnenie voliteľných značiek na mape. Navyše je možné pripojiť k značkám alfanumerické označenie. *URL* požiadavky musí byť v tvare:

```
https://maps.googleapis.com/maps/api/staticmap?parameters
```

Parameters definujú parametre *URL* požiadavky. Delia sa na povinné a voliteľné. Ako pri štandardnom *URL,* jednotlivé parametre sú oddelené charakterom ampersand (*&*) a samotné oddelenie hodnôt, v prípade definovania viacerých hodnôt pre jeden parameter požiadavky sa oddeľuje separátorom zvislej čiary (|). Služba *Google Static Maps* definuje mapu na obrázku podľa nasledujúcich *URL* parametrov, z ktorých nás zaujímali hlavne nasledujúce:

- *center* – lokalizačný parameter definujúci stred mapy, ekvidistantný k všetkým hranám mapy. Hodnota parametra je pozícia definovaná cez čiarkou oddelený pár *{zemepisná šírka, zemepisná dĺžka}* alebo adresa identifikujúca unikátnu lokáciu na zemi.
- *zoom* – lokalizačný parameter definujúci približovaciu úroveň mapy, čo určuje úroveň zväčšenia mapy. Parameter je numerická hodnota.
- *size* – parameter definujúci obdĺžnikové rozmery obrázka mapy. Hodnotou parametra je text vo forme *{horizontálna hodnota}x{vertikálna hodnota}.*
- *scale* – parameter definujúci počet pixelov v návratovom obrázku. Štandardnou hodnotou parametra je 1, pri nastavení *scale=2* je vygenerovaný obrázok s dvojnásobným počtom pixelov pri rovnakom pokrytí obsahu mapy.
- *markers* – parameter definuje značky pripojené k obrázku na špecifikovaných pozíciách v mape.
- *key* – povinný parameter monitorujúci využitie *API* a sprístupňujúci využívanie služby. [18]

Na príklade možno vidieť, ako sa *URL* požiadavka transformuje na obrázok.

```
https://maps.googleapis.com/maps/api/staticmap?
center=49.220082,18.7434378&zoom=16&size=400x400
```

Obrázok 19 Príklad vygenerovanej mapy

5.2.6 Analýza možností používateľa

Pri návrhu aplikácie bolo potrebné myslieť aj na samotného používateľa, ktorý vstupuje do procesu jej používania. Pri ovládaní aplikácie je vhodné poskytnúť používateľovi nástroje, ktorými si je schopný upraviť niektoré vlastnosti aplikácie. V našom prípade by mala aplikácia sprostredkovávať komunikáciu, obsluhu riadiacich prostriedkov a zobrazovať relevantné dáta. Pri návrhu statickej aplikácie by zmena niektorých z týchto komponentov mala za následok nefunkčnosť danej funkcionality alebo samotnej aplikácie a opravenie by si vyžadovalo zásah do návrhu aplikácie, čo je pre používateľa nežiaduca situácia.

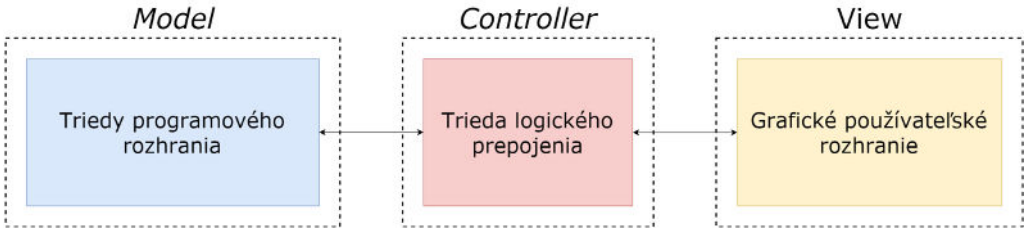
Riešením by teda bol návrh generickejších komponentov, ktorých vlastnosti by bolo možné meniť počas behu aplikácie a tým si ju prispôsobovať. Takéto riešenie sme považovali za vhodné aj v prípade komponentov, ktorých prostriedky sme si konkrétne stanovili (príkladom môže byť výber konkrétneho ovládača ako riadiaceho prostriedku), pretože takýto prístup zlepšuje využiteľnosť navrhnutej aplikácie a neobmedzuje sa iba na konkrétne riešenie nášho problému. Okruhy, ktorým sme pri návrhu venovali pozornosť, boli:

- Možnosti nastavenia komunikačného rozhrania, nastavenie typu a vlastností komunikačného kanála

- Možnosti nastavenia riadiaceho rozhrania pri použití externého riadiaceho prostriedku, výber riadiaceho prostriedku pri existencií viacerých, výber riadiacich komponentov (ovládacích prvkov) ovládača používateľom
- Možnosti nastavenia zobrazovania mapy aktuálnej pozície a jej vlastností
- Možnosti výberu relevantných senzorických dát, ktorých aktuálne hodnoty si používateľ žiada zobrazovať, spolu s rovnakou možnosťou pre ich grafickú reprezentáciu v grafoch.

5.3 Návrh a implementácia aplikácie

Návrh architektúry aplikácie sme založili na použití návrhového vzoru *MVC*, ktorého vlastnosti a použitie sme si rozanalyzovali v prechádzajúcej časti. Návrh aplikácie sme rozdelili na návrh architektúry tried programovej časti aplikácie a grafického rozhrania. Na obrázku 20 je zobrazené schematické rozdelenie tried aplikácie.



Obrázok 20 Schematický návrh architektúry aplikácie

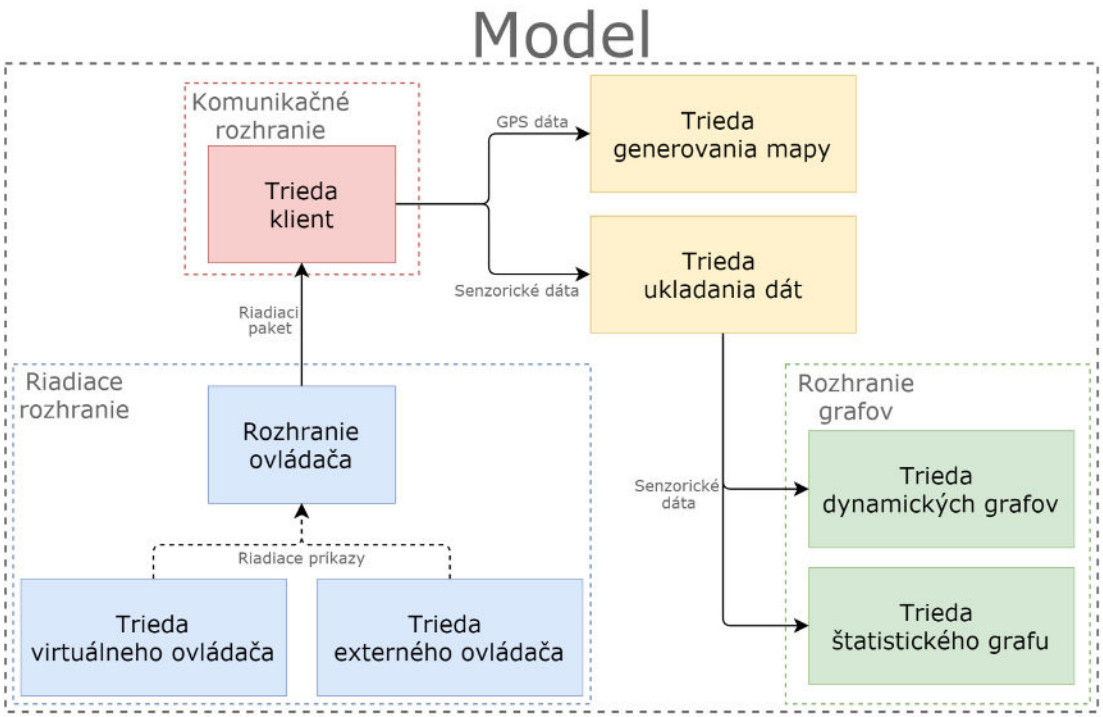
Návrh *View* komponentu obsahoval jednu triedu používateľského rozhrania, ktorá implementovala grafické prostriedky ovládania aplikácie. Návrh *Controller* triedy obsahoval referencie na triedu používateľského rozhrania a všetky relevantné triedy *Model* komponentu, medzi ktorými sprostredkovával komunikačné rozhranie. Úlohou triedy bola tiež inicializácia jednotlivých tried pri spustení aplikácie. Návrh *Model* komponentu pozostával z viacerých tried vyplývajúcich zo špecifikácie programovej časti a vzťahy medzi nimi definovala interakcia navzájom a interakcia s používateľským prostredím prostredníctvom *Controller* triedy. Realizáciu aplikácie sme rozdelili na dve časti, v ktorých sme sa venovali návrhu a implementácii programového a vizuálneho rozhrania.

5.3.1 Realizácia programového rozhrania aplikácie

Programové rozhranie tvorí vnútornú logiku aplikácie, ku ktorej používateľ nemá prístup. Rozhranie vykonáva všetky špecifikované úlohy aplikácie a je riadené implementovanou vnútornou logikou alebo prostredníctvom interakcie používateľa s vizuálnym rozhraním. Návrh štruktúry a rozvrhnutia tried bolo dôležitým aspektom

vhodnej realizácie. Interakcia a výmena dát sa uskutočňuje nielen s *Controller* triedou, ale aj medzi triedami navzájom, čo bolo potrebné zahrnúť do návrhu. Štruktúra vyplývala z požiadaviek špecifikovaných pre programové rozhranie a jej schematický návrh možno vidieť na obrázku 21.

Jednotlivé bloky v štruktúre definujú triedu alebo skupinu tried, ktorých funkcionalita zabezpečuje definované požiadavky. Úlohami jednotlivých blokov bola realizácia komunikačného rozhrania aplikácie základňovej stanice, riadiaceho rozhrania, štrukturálneho ukladania prichádzajúcich dát, vizualizáciu dát prostredníctvom grafov a vizualizácia *GPS* dát v podobe výseku mapy. Postupne sme sa venovali návrhu a realizácii tried jednotlivých blokov a ich prepojenia s *Controller* triedou (a v konečnom dôsledku s vizuálnym rozhraním), ktorá bola tiež súčasťou programového riešenia aplikácie.



Obrázok 21 Schematický návrh štruktúry tried komponentu *Model*

5.3.1.1 Implementácia komunikačného rozhrania

Komunikačné rozhranie sme implementovali prostredníctvom navrhnutej triedy *Client*, ktorá obsahovala komunikačnú rutinu, spracovanie prichádzajúcich dát a ich distribúciu do ostatných tried. Triedu sme odvodili zo základnej triedy *Thread* a implementovali zdedené funkcie, ktoré nám sprístupňovali vykonávanie úloh v oddelenom vlákne od hlavného.

Využitie oddeleného vlákna nám dovolilo uskutočňovať obojstrannú komunikáciu a spracovávanie dát bez problémov z reaktívnosťou grafického rozhrania.

Implementácia triedy definuje premenné, ktoré pri vytváraní spojenia určujú jeho vlastnosti akými sú *IP* adresa, port, druh pripojenia a iné. Trieda mala definované dva druhy pripojenia – lokálne, ktoré sa používalo na testovacie účely a reálne, ktoré využíva internetovú sieť na komunikáciu s modelom vzducholode. Pre lokálne pripojenie tak bolo možné nastaviť niektoré parametre pripojenia ako statické a zjednodušiť tak interakciu zo strany používateľa. Inicializácia spojenia sa uskutočňuje v metóde *connect()*, v ktorej sa primárne vytvorí *socket* podľa hodnôt premenných. Následne sa vytvorí vstupný dátový *stream* prostredníctvom tried *BufferedReader* a *InputStreamReader*, ktorého objekt slúži na čítanie dát. Okrem vstupného sa vytvorí aj výstupný dátový *stream* prostredníctvom tried *BufferedWriter* a *OutputStreamWriter*, ktorého inštancia slúži na posielanie dát z aplikácie. O výsledku inicializácie je používateľ informovaný a v prípade neúspechu aj s príčinou možnej chyby.

Pri úspešnej inicializácii je nad objektom zavolaná metóda *start()*, ktorá začne vykonávať metódu *run()* v oddelenom vlákne. Metóda *run()* implementuje nekonečný cyklus obsahujúci komunikačnú rutinu a spracovanie prichádzajúcich paketov. Jeden cyklus je tvorený vykonaním príkazov:

- získania riadiaceho paketu z objektu rozhrania *ControlDataProvider*, ktorého metóda pristupuje k aktívnemu riadiacemu prostriedku a číta aktuálne riadiace dáta
- vloženia riadiaceho paketu do výstupného *streamu*
- načítania dát zo vstupného *streamu* a spracovanie prostredníctvom metódy *processPacket()*
- uspania vlákna na definovaný časový interval za účelom riadenia periodicity prenosu dát

Spracovanie prichádzajúcich paketov a distribúcia dát sa uskutočňujú v metóde *processPacket()*. Vstupným parametrom metódy je paket v textovej podobe. V prvom rade sa obsah rozdelí na čiastkové pakety podľa definovaného deliaceho znaku zvislej čiary a každá časť sa vyhodnocuje samostatne. Pri návrhu komunikačného podsystému boli zadefinované tri typy čiastkových paketov, ktoré sú prenášané do základňovej stanice.

Pri rozpoznaní informačného paketu je jeho obsah rozdelený na jednotlivé elementy. Na základe počtu a obsahu týchto elementov sa vytvoria a inicializujú dátové štruktúry pre ukladanie senzorických dát v triede *DataStorage*. Grafické rozhranie je notifikované o inicializácii dátových štruktúr a používateľovi je sprístupnený zoznam dostupných senzorických dát. Pri rozpoznaní senzorického paketu je jeho obsah rozdelený na jednotlivé elementy a na základe mena veličiny je hodnota priradená do korešpondujúcej dátovej štruktúry. Grafické rozhranie je notifikované o aktualizácii senzorických dát. Pri rozpoznaní kontrolného paketu je jeho obsah použitý na aktualizáciu hodnôt vychýlenia modelu v grafickom rozhraní a aktualizáciu *GPS* dát v objekte triedy *GpsMapThread,* ktoré sú transformované do podoby obrázku mapy.

5.3.1.2 Implementácia riadiaceho rozhrania

Riadiace rozhranie sme implementovali podľa modelu riadenia navrhnutého v kapitole riadiaceho podsystému. Keďže boli zvolené dva prostriedky riadenia, prispôsobili sme tomu aj implementáciu a rozdelili ju do dvoch samostatných riadiacich tried. Okrem týchto dvoch tried bolo súčasťou aj jednoduché rozhranie *ControlDataProvider*, ktoré definovalo abstraktné metódy poskytovania riadiacich dát. Rozhranie bolo implementované do oboch riadiacich tried, v ktorých boli metódy rozhrania navrhnuté podľa špecifikácie triedy. Rozhranie poskytuje uniformný spôsob získavania riadiacich dát z pohľadu komunikačného rozhrania, ktoré nemusí disponovať referenciami na objekty riadiacich tried a informáciu o práve aktívnom riadiacom prostriedku. Trieda *Client* obsahuje iba referenciu na objekt rozhrania *ControlDataProvider*, ktorého metódy volá. Metódy vracajú riadiace dáta aktívneho riadiaceho prostriedku. Objektu rozhrania je v jednom okamihu priradený práve jeden riadiaci prostriedok – objekt riadiacej triedy, ktorého zmenu určuje používateľ.

Prvým typom riadiaceho prostriedku bolo využitie virtuálnych grafických komponentov na generovanie riadiacich príkazov, čo sme implementovali v triede *VirtualJoystick*. Trieda bola zdedená z triedy grafického komponentu *JPanel* z dôvodu požiadavky implementácie 2-osého virtuálneho *joysticku*. Návrh virtuálneho *joysticku* bol inšpirovaný reálnym *joystickom* nachádzajúcim sa na herných ovládačoch a požadovali sa aj rovnaké možnosti ovládania. Trieda implementovala vykresľovanie grafického vzhľadu *joysticku* v paneli a jeho ovládanie prostredníctvom myši. Návrh panelu obsahoval kurzor nachádzajúci sa v strede, ktorého ovládanie myšou bolo graficky reprezentované. Reaktívnosť panelu na

myš bola implementovaná prostredníctvom triedy *MouseAdapter*, ktorá bola zaregistrovaná ako poslucháč udalostí vyvolaných myšou (*MouseListener*). Metódy tejto triedy boli implementované ako obsluha na udalosti generované používaním myši v paneli. Vznik udalostí sa transformoval na grafické zmeny panelu a tiež zmenu vnútorných premenných triedy definujúcich zložky riadiacich príkazov. Trieda bola doplnená dvojicou premenných definujúcich riadiace príkazy pre pohyb a rotáciu zadných krídiel, ktorých hodnoty sú aktualizované posuvníkmi v grafickom rozhraní. Trieda tiež implementuje rozhranie *ControlDataProvider* a jej metódy, ktorými sprístupňuje aktuálne riadiace dáta v podobe paketu a hodnoty riadiacich premenných.

Alternatívou k virtuálnemu riadeniu sme implementovali riadenie prostredníctvom hardvérového herného ovládača s využitím možností externej triedy *JInput*. Konkrétnym použitím typom ovládača bol *Dualshock 4*, ale implementovaná trieda *Joystick* dovoľuje použitie iných typov ovládačov a ich nastavenie. Trieda je odvodená z *Thread* triedy a vykonávanie čítania dát z ovládača sa uskutočňuje v oddelenom vlákne, aby nevznikol problém z reaktívnosťou grafického rozhrania. V triede boli implementované metódy na vyhľadanie vstupných zariadení, aktiváciu zariadenia, nastavovanie riadiacich prvkov a samotné čítanie vstupných dát.

Pri spustení aplikácie je zavolaná metóda *findControllers()*, ktorá prostredníctvom prostredia systému získa zoznam dostupných vstupných zariadení. Na základe zadefinovaného filtra ponechá v zozname iba zariadenia, ktoré sú ovládačmi a obsahujú požadované komponenty. Štandarde bola implementácia uskutočnená pre náš ovládač, preto sa automaticky aktivuje a nastavia sa zvolené riadiace prvky metódou *setDefaultKeyMapping()*. Vnútorným premenným triedy (riadiacim prvkom) sa priradia komponenty ovládača, ktoré generujú požadované riadiace vstupy. Komponent ovládača predstavuje hardvérové tlačidlo alebo *joystick,* nachádzajúce sa na ovládači. Používateľ má prístup k zmene hodnôt týchto riadiacich prvkov prostredníctvom metódy *setComponent()*, ktorá na určitú časovú dobu dovolí zmeniť aktívny komponent ovládača pre zvolený riadiaci prvok. Podmienkou je analógový výstup zvoleného komponentu ovládača. Príkladom môže byť zvolenie X-ovej osi ľavého *joysticku* ako riadiaceho prvku na rotáciu modelu vzducholode. Spustenie čítania riadiacich dát sa uskutočňuje *run()* metódou, ktorá obsahuje nekonečný cyklus uskutočňujúci čítanie. Nekonečný cyklus skontroluje prítomnosť aktívneho ovládača a prečíta analógové hodnoty z jednotlivých riadiacich prvkov, ktoré uloží do vnútorných premenných triedy. Trieda taktiež implementuje

rozhranie *ControlDataProvider*, ktorého metódami sa sprístupňujú riadiace dáta v podobe paketu.

5.3.1.3 Implementácia ukladania údajov

Ukladanie prichádzajúcich údajov sme implementovali prostredníctvom triedy *DataStorage.* Trieda obsahuje dve dátové štruktúry na uchovanie dát a metódy na manipuláciu s ich obsahom. Prvá dátová štruktúra *dataTypes* slúži na uchovanie informácii o jednotlivých senzorických veličinách. Štruktúra sa inicializuje pri spracovaní informačného paketu, ktorý je transformovaný do objektov triedy *DataType* a uložený ako zoznam senzorických veličín.

Trieda *DataType* je pomocnou triedou navrhnutou na uloženie informácii o senzorickej veličine. Objekt tejto triedy obsahuje názov veličiny, fyzikálnu jednotku a rozsah hodnôt. Tieto informácie sa používajú v aplikácii pri vizualizácii dát a grafov. Ku každej senzorickej veličine sa inicializuje dátová štruktúra na ukladanie prichádzajúcich senzorických dát.

5.3.1.4 Implementácia tried grafov

Sprostredkovanie senzorických údajov vo forme grafov sme implementovali dvoma spôsobmi. Využili sme knižnicu *JFreeChart*, ktorá nám poskytla nástroje na reprezentáciu dát v podobe grafu. Na základe požiadaviek na aplikáciu sme implementovali triedu vytvárajúcu dynamické grafy zobrazujúce aktuálny priebeh senzorických veličín a triedu generujúcu statický graf senzorickej veličiny obsahujúci celý priebeh veličiny alebo jeho časť.

Prvým typom boli dynamické grafy, ktorých vytváranie sme implementovali v triede *DataChart*. Objekt triedy predstavuje dynamický graf jednej senzorickej veličiny, ktorý sa aktualizuje v pravidelných časových intervaloch a do množiny dát ukladá najaktuálnejšiu hodnotu danej senzorickej veličiny. Vstupným parametrom konštruktora triedy sú referencie na dátové štruktúry obsahujúce informácie o senzorickej veličine a dáta. Informácie sa transformujú na vlastnosti grafu a dáta sa použijú na inicializáciu dátovej množiny grafu. Referencia na dáta sa uloží do vnútornej premennej, aby bolo možné pristupovať k aktuálnym hodnotám veličiny. Prostredníctvom metódy *createChart()* sa vytvorí inštancia grafu, ktorej sa nastaví dátová množina a všetky požadované vlastnosti, akými sú názov grafu, názvy osí, rozsah hodnôt a tiež vizuálna stránka grafu. Aktualizácia

priebehu grafu sa uskutočňuje časovačom s nastavenou periódou, v ktorom sa volajú metódy *advanceTime()* a *appendData()* nad inštanciou množiny dát.

Pre druhý typ bola navrhnutá trieda *StaticticChart*, ktorej úlohou je vytváranie štatistického grafu senzorickej premennej. Na rozdiel od dynamického grafu slúži na zobrazovanie štatistického priebehu senzorickej veličiny, čomu je prispôsobená aj implementácia a využité prostriedky. Štandardne sa vytvára čiarový diagram na základe vstupných parametrov konštruktora, ktoré určujú vlastnosti grafu a množinu dát. Používateľ má k dispozícii nástroje na definovanie určitého výseku grafu prostredníctvom prvkov grafického rozhrania, ktorých hodnoty sú tiež parametrami konštruktora a používajú sa na filtrovanie množiny dát. Metóda *createChart()* vytvorí inštanciu grafu s požadovanými vlastnosťami, ktorá sa vloží do inštancie panelu na zobrazenie v grafickom rozhraní.

5.3.1.5 Implementácia *GPS* mapy

Riešenie generovania mapy na základe *GPS* parametrov sme implementovali v triede *GpsMapThread.* Trieda je odvodená z triedy *Thread* z dôvodu časovej náročnosti pri komunikácii so serverom a sťahovaní obrázku mapy. Pri inicializácii objektu triedy sa definujú hodnoty pre ukladanie aktuálnej zemepisnej šírky a dĺžky. Okrem toho sa vytvára referencia na ikonu modelu vzducholode, ktorá sa používa pri vykresľovaní mapy. Kontrolný paket obsahuje dáta z *GPS* senzora, ktoré sú metódami triedy transformované na výsek mapy a údaj z kompasu informujúci o smerovaní vzducholode. Objekt vzducholode je transformovaný do podoby ikony vzducholode vykreslenej na mape. Pri spustení vlákna sa začne vykonávať nekonečný cyklus, ktorý má dve základné úlohy.

Prvou úlohou je sprostredkovávať mapu v podobe obrázku prostredníctvom *Google Static Maps API.* Sprostredkovanie mapy sa uskutočňuje iba v prípade, že vykresľovanie je aktívne a niektorá z premenných zmenila hodnotu. Relevantnými premennými sú hodnoty zemepisnej šírky, dĺžky a približovacia úroveň mapy. Takáto podmienka bola stanovená, aby sa zbytočne negenerovala požiadavka na mapu, ktorá má rovnaké parametre ako práve zobrazovaná. Pri kladnom vyhodnotení kontrolných podmienok sa vytvára *URL* požiadavka na základe aktuálnych hodnôt premenných a prostredníctvom výstupného *streamu* sa posiela na server. Server vygeneruje na základe požiadavky obrázok mapy, ktorý sa vstupným *streamom* stiahne a poskytne prvkom grafického rozhrania na vykreslenie.

Druhá úloha vlákna spočíva v poskytovaní ikony modelu vzducholode, ktorá sa vykresľuje na mape a slúži na poskytnutie informácie o smerovaní vzducholode. Sprostredkovanie ikony sa uskutočňuje iba v prípade, že vykresľovanie je aktívne a hodnota aktuálneho uhla sa zmenila. V prípade kladného vyhodnotenia podmienok sa na statickú rotáciu aplikuje rotácia a výstup sa poskytne grafickému rozhraniu.K hlavnej triede *GpsMapThread* bola implementovaná aj pomocná trieda *LabelMap* odvodená z triedy grafického komponentu *JLabel.* Úlohou triedy je sprostredkovávať zmenu približovacej úrovne mapy prostredníctvom stredného tlačidla myši. Objekt triedy *LabelMap* je priradený panelu mapy a udalosti generované myšou menia hodnotu premennej definujúcej aktuálnu približovaciu úroveň mapy.

5.3.2 Realizácia vizuálneho rozhrania aplikácie

Návrh a implementácia vizuálneho rozhrania bola druhou dôležitou časťou vývoja aplikácie. Pozostávala z vytvorenia grafického používateľského rozhrania a implementácie obsluhy udalostí vyvolaných interakciou s ním. Počas celého procesu sme vychádzali zo zásad pre tvorbu grafického rozhrania a tiež možností *frameworku Swing*, s ktorým sme pracovali. Ako najvhodnejšie riešenie sme považovali rozdelenie aplikácie podľa požiadaviek na viaceré okná, ktorých obsah by spájala logická súvislosť. Takýmto spôsobom by sme sprehľadnili vizualizáciu a používateľ by sa lepšie orientoval v aplikácii, ktorá by neobsahovala všetky komponenty na jednej obrazovke. Podľa charakteru požiadaviek sme grafické rozhranie rozdelili na nasledujúce časti:

- Hlavné používateľské okno aplikácie obsahujúce vizualizáciu prichádzajúcich dát a tiež poskytujúce prostriedky na riadenie modelu vzducholode
- Okno pre štatistické vykresľovanie grafov z dostupných dátových veličín
- Okno pre nastavovanie vlastností aplikácie

Návrh obsahu jednotlivých okien sme tiež rozdelili podľa logického zoskupenia súvisiacich grafických prvkov.

5.3.2.1 Hlavné aplikačné okno

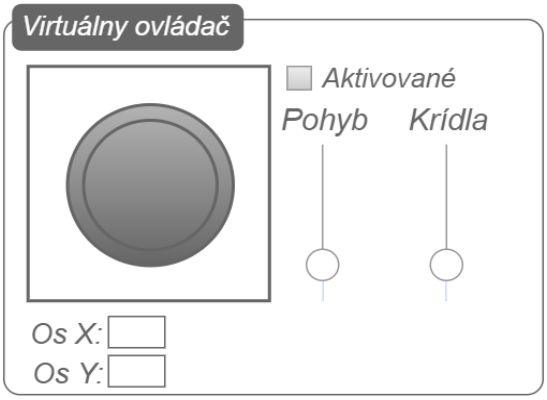
Úlohou hlavného okna bolo sprostredkovávať používateľovi vizualizáciu všetkých druhov dát prichádzajúcich z modelu vzducholode v reálnom čase a zároveň poskytovať prostriedky na jej riadenie. Okno sme rozdelili na štyri základné časti:

- Panel vizualizácie riadiaceho rozhrania na ovládanie modelu vzducholode
- Panel vizualizácie aktuálnych kontrolných a senzorických dát

- Panel vizualizácie aktuálnej polohy a smerovania modelu vzducholode
- Panel vizualizácie senzorických dát prostredníctvom grafov

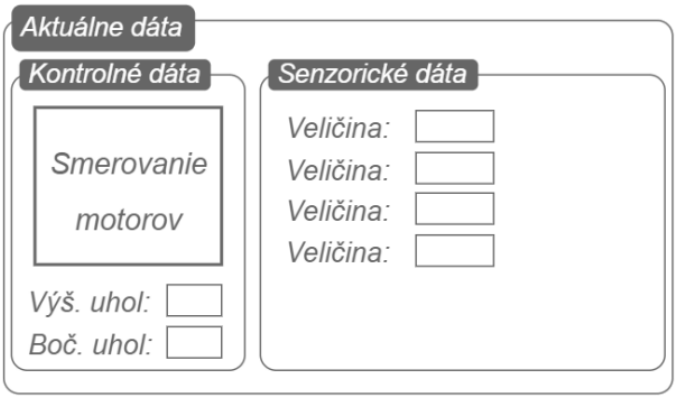
Obrázok 22 Schematický návrh hlavného okna

Implementácia riadiaceho rozhrania bola uskutočnená podľa navrhnutého modelu riadenia vo forme virtuálneho ovládača, ktorého prvky generujú hodnoty pre riadiace príkazy. Virtuálny ovládač pozostáva z virtuálneho *joysticka*, ktorým sa umožňuje rotácia v dvoch osiach a dvojice posuvných komponentov, ktoré doplňujú funkcionalitu o ovládanie hlavných motorov a nastavenie vychýlenia chvostových krídiel. Interakcia s týmito prvkami má za následok zmenu vnútorných premenných používaných na generovanie riadiacich príkazov. Riadiace rozhranie bolo ešte doplnené o aktivačný prvok, ktorým sa definuje aktívny riadiaci prostriedok, teda výber medzi riadením prostredníctvom virtuálneho ovládača alebo externého ovládača. Dvojica textových polí poskytuje spätnú väzbu *joysticka* a zobrazuje jeho aktuálne hodnoty.



Obrázok 23 Schematický návrh virtuálneho ovládača

Implementácia jednoduchého zobrazovania dát bola rozdelená na dve časti. Prvou bola vizualizácia riadiacich dát obsahujúcich informácie o vychýlení modelu vzducholode z osí, spolu s informáciou o vychýlení hlavných motorov. Druhým typom dát boli senzorické dáta, k zobrazovaniu ktorých sme pristúpili z všeobecnejšieho hľadiska. Počet dostupných typov senzorických dát, ich podoba a jednotky sa môžu meniť, preto sme sa rozhodli pre zobrazenie každej takejto dátovej veličiny v rovnakom tvare. Názov veličiny nasledovaný jej hodnotou a jednotkou tvoril jeden vizualizovaný dátový blok. Implementácia panela dovoľovala zobrazenie maximálne 14 dátových veličín a jeho obsah sa prispôsoboval nastaveniam zo strany používateľa.



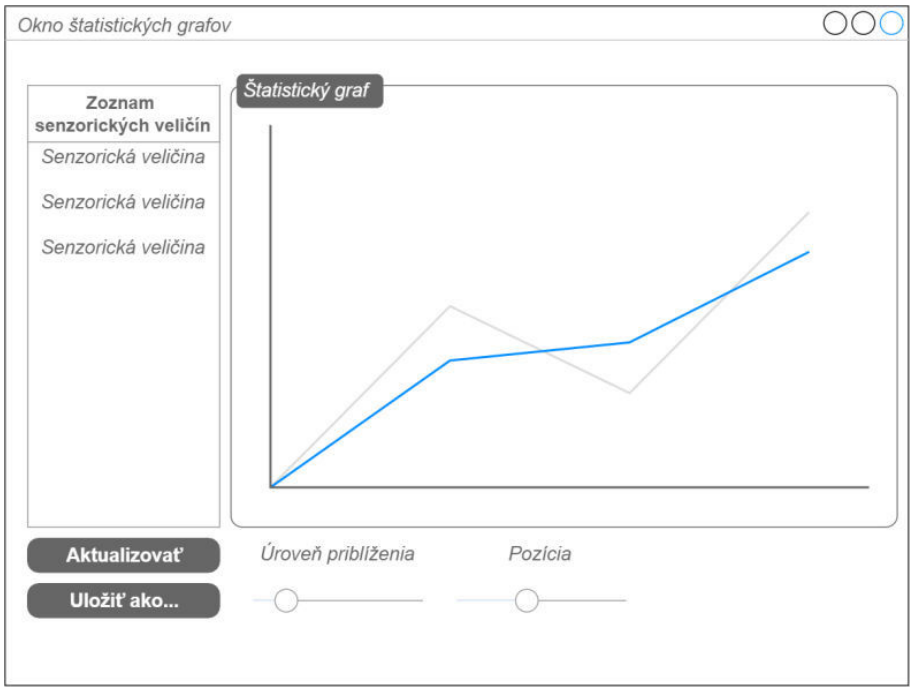
Obrázok 24 Schematický návrh vizualizácie dát

Implementácia panelu aktuálnej polohy zahrňovala interpretáciu hodnôt z *GPS* senzora a kompasu do podoby výseku mapy s ikonou modelu vzducholode nasmerovanou podľa skutočného smerovania vzducholode. Implementácia tiež zahrňovala možnosť zmeny priblíženia mapy kolieskom myši aplikovanom v paneli. V prípade nedostupnosti internetového pripojenia ostáva panel neaktívny a neinterpretuje žiadne dáta. Pri vypnutí zobrazovania alebo strate internetového spojenia zostáva vykreslená posledná dostupná mapa.

Implementácia zobrazovania senzorických dát prostredníctvom grafov bola uskutočnená formou oddelených panelov. Lišta záložiek nachádzajúca sa na vrchu panela umožňuje prepínanie medzi panelmi aktívnych grafov senzorických dát. Aktualizácia zobrazovaných dát bola implementovaná v pravidelných časových intervaloch pre všetky grafy, teda nie len pre aktuálne zobrazovaný.

5.3.2.2 Aplikačné okno pre štatistické grafy

Úlohou aplikačného okna pre vykresľovanie štatistických grafov bolo poskytnúť používateľovi nástroj na zobrazenie celého priebehu niektorej sledovanej dátovej veličiny alebo časti jej priebehu. Hlavná časť implementácie pozostávala zo zoznamu dátových veličín, generovaného podľa dostupných senzorických dát v pamäti aplikácie a panelu, v ktorom sa vykresľoval graf zvolenej veličiny. Vykreslený graf obsahuje všetky spracované hodnoty danej veličiny od začatia komunikácie a neaktualizuje sa automaticky prichádzaním nových dát. Vykreslenie aktuálneho grafu pre rovnakú veličinu bolo implementované rovnakým výberom danej veličiny v zozname, alebo stlačením tlačidla *Aktualizovať*. Manipulácia s grafom je možná použitím posuvných komponentov *Úroveň priblíženia* a *Pozícia,* prostredníctvom ktorých je možné zvoliť si určitý výsek vykresľovanej veličiny. Možnosť uložiť si vykreslený graf v obrázkovom formáte bola implementovaná po stlačení tlačidla *Uložiť ako...*, čo vyvolá okno výberu priečinka a názvu ukladaného súboru.



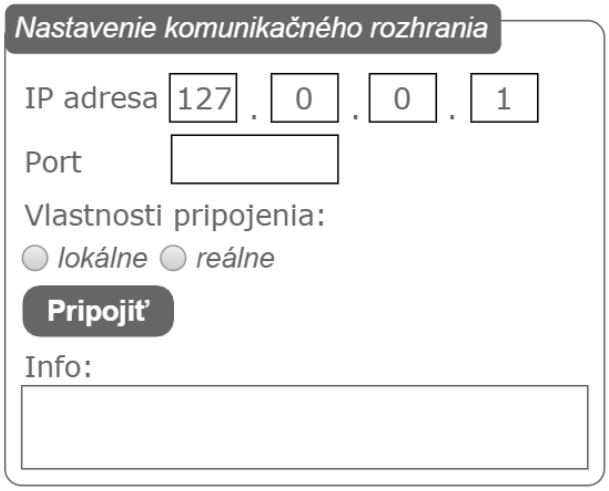
Obrázok 25 Schéma návrhu okna štatistických grafov

5.3.2.3 Aplikačné okno nastavení aplikácie

Úlohou aplikačného okna nastavení bolo poskytnúť používateľovi nástroje na prispôsobenie si aplikácie a tiež nastavenie vlastností jednotlivých jej kľúčových komponentov. Implementované možnosti nastavenia boli rozdelené do jednotlivých

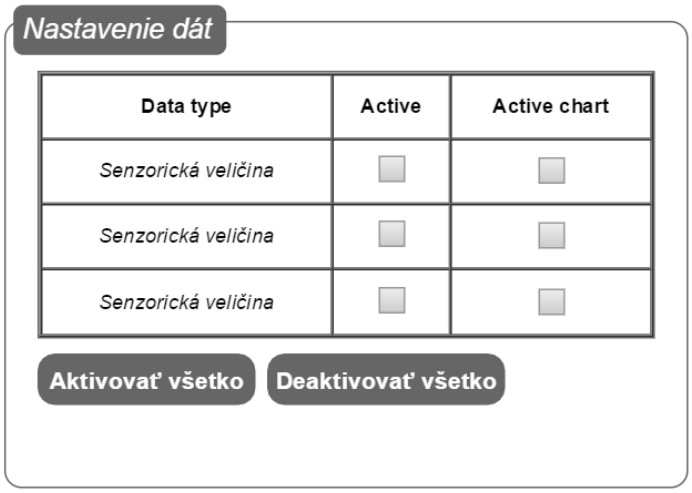
panelov a zahrňovali nastavovanie komunikačného rozhrania, časti riadiaceho rozhrania, dostupných senzorických dát, *GPS* mapy a jazyka aplikácie

Panel nastavení komunikačného rozhrania bol navrhnutý tak, aby poskytoval používateľovi všetky potrebné možnosti pri vytváraní spojenia s modelom vzducholode. Jeho implementácia poskytuje nastavenia *IP* adresy a portu, výber typu komunikácie a jeho vlastností. Rozhranie tiež obsahuje informačný panel, ktorý poskytuje používateľovi informácie o pripojení, chybách počas pripojenia alebo počas prenosu dát.



Obrázok 26 Návrh panelu nastavenia komunikačného kanála

Panel nastavení dostupných dát bol navrhnutý vo forme tabuľky, v ktorej sa zobrazujú všetky typy prichádzajúcich senzorických dát. Obsah tabuľky sa aktualizuje pri vytvorení spojenia alebo pri zmene dostupných dátových veličín na strane modelu vzducholode.

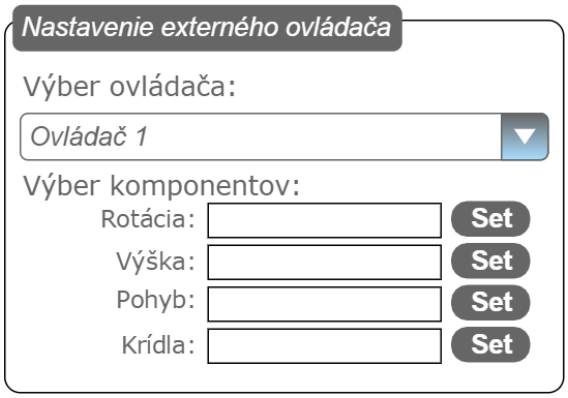


Obrázok 27 Návrh panelu nastavenia dát

Používateľ má prístup k tomuto zoznamu a tabuľka mu dovoľuje výber aktívnych dátových typov, ktorých aktuálne hodnoty sa budú zobrazovať v hlavnom okne a rovnako

aj výber aktívnych grafov. Pre jednoduchšie možnosti filtrácie bola do panelu doplnená dvojica tlačidiel, ktoré umožňujú zmeny všetkých veličín v tabuľke.

Návrh nastavenia externého ovládača pozostával z dvoch častí. Prvou možnosťou bol výber aktívneho ovládača z dostupného zoznamu nájdených ovládačov, ktorý aplikácia vytvorí pri svojom spustení. V prípade, že nebol nájdený žiaden dostupný ovládač, je obsah panelu deaktivovaný a používateľ je informovaný o tejto skutočnosti. Druhou možnosťou nastavenia ovládača je výber jeho hardvérových komponentov ako riadiacich prvkov pre ovládanie modelu vzducholode. Štandardne sme pracovali s *Playstation 4 Dualshock* ovládačom, ktorého nastavenie jeho riadiacich komponentov sme už zahrnuli pri spustení aplikácie, ale implementácia umožňuje zmenu jednotlivých prvkov podľa potrieb používateľa.



Obrázok 28 Návrh panelu nastavenia externého ovládača

Štvrtým navrhnutým panelom okna bolo nastavovanie *GPS* mapy aktuálnej polohy modelu. Návrh panelu zahrňoval možnosti aktivácie zobrazovania mapy a možnosti manipulácie s približovacou úrovňou mapy.

Obrázok 29 Návrh panelov nastavení GPS mapy a jazyka

Posledným navrhnutým panelom okna bolo nastavenie jazyka aplikácie, kde sme implementovali výber slovenského alebo anglického jazyka.

Výsledný vzhľad jednotlivých okien aplikácie sa nachádza v prílohe práce.

6 Experimentálne testovanie

Návrh a implementácia komunikačného kanála, modelu riadenia a aplikácie základňovej stanice sú dlhodobé procesy, ktorých vývoj bolo potrebné nejakým spôsobom kontrolovať a testovať. Mať spätnú väzbu na funkcionalitu jednotlivých implementovaných komponentov je veľmi dôležité pre ich prípadnú úpravu alebo rozšírenie. Uskutočnenie takejto spätnej väzby na reálnom modeli vzducholode by bolo v priebehu vývoja ťažko uskutočniteľné a tiež nebezpečné z hľadiska testovania rozpracovaných vecí.

Z toho dôvodu sme sa rozhodli navrhnúť si prostriedok na otestovanie jednotlivých častí práce a to prostredníctvom emulácie vzducholode vo virtuálnom prostredí. S vhodne zvolenými nástrojmi emulácie by sme boli schopný otestovať vlastnosti komunikačného kanála a emulovaním pohybu vzducholode vo virtuálnom prostredí by sme otestovali model riadenia. Generovaním senzorických dát a spätnej väzby by sme tiež mohli otestovať funkcionalitu aplikácie základňovej stanice, z ktorej by bolo emulované prostredie ovládané.

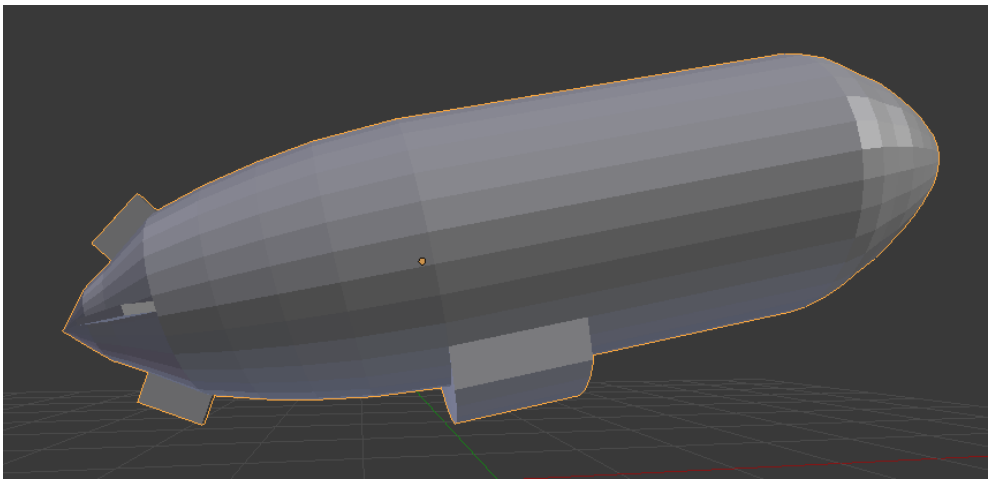
6.1 Testovacie rozhranie pre emuláciu modelu vzducholode

Ako testovacie a emulačné prostredie sme sa rozhodli použiť aplikáciu *Blender*, ktorá je *open source* softvérom slúžiacim na modelovanie a vykresľovanie 3D počítačovej grafiky. Ďalším silným aspektom tejto aplikácie je možnosť navrhovania a vykonávania skriptov napísaných v programovacom jazyku *Python*. Aplikácia *Blender* poskytuje knižnicu *bpy.py*, ktorá rozširuje možnosti jazyka *Python* o funkcie umožňujúce prístup k dátam, funkciám a možnostiam programu *Blender*. Prostredie aplikácie *Blender* bolo použité na návrh modelu vzducholode.

6.1.1 Model vzducholode

Pri modelovaní vzducholode v aplikácii *Blender* sme nevychádzali zo žiadnych špecifických parametrov a pre potreby emulovania bol náš návrh čo najjednoduchší. Úlohou emulácie nebolo testovať správanie modelu vzducholode v reálnych podmienkach, ale overenie funkcionality generovania riadiacich príkazov, ich prenosu a spracovania. Úlohou emulácie bolo iba vizualizovať výsledok tohto procesu a pohyb vzducholode

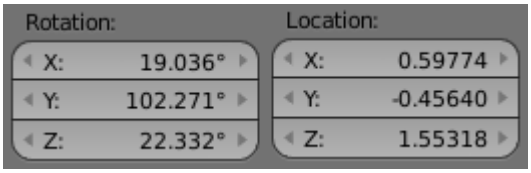
v emulovanom prostredí bol vhodným nástrojom spätnej väzby pre nás. Nami navrhnutý model bol postačujúci pre potreby emulácie.



Obrázok 30 Vymodelovaná vzducholoď v programe *Blender*

6.1.2 Vlastnosti modelu vzducholode

Dôležitejšími aspektmi modelu vzducholode ako jej vzhľad boli vlastnosti modelu v zmysle aplikácie *Blender*. Každý model v *Blender* aplikácii je vnímaný ako objekt s rôznymi vlastnosťami, ako napríklad veľkosť, pozícia, atď. Najdôležitejšími vlastnosťami z nášho pohľadu boli pozícia a natočenie objektu. Pozícia objektu predstavuje jeho relatívnu pozíciu vzhľadom k stredu sústavy súradníc definovanom ako bod [0,0,0]. Na počiatku emulácie je v ňom vyobrazený stred nášho modelu. Druhým dôležitým atribútom je natočenie alebo rotácia objektu, čo popisuje pozíciu objektu vzhľadom k jednotlivým osiam sústavy súradníc X, Y, Z.



Obrázok 31 Vlastnosti modelu vzducholode - rotácia a pozícia

6.1.3 *Python* skript

Interakcia s objektmi v používateľskom rozhraní aplikácie *Blender* je uskutočňovaná samotným používateľom alebo počas vykonávania *Python* skriptu, v ktorom sú objekty a ich vlastnosti prístupné prostredníctvom *bpy* knižnice. Knižnica *bpy* sprístupňuje v programovej forme nástroje, ktoré sa nachádzajú v *Blender* aplikácii. Návrh a implementácia *Python* skriptu bola dôležitou súčasťou pri realizácii emulácie.

6.2 Realizácia riešenia

Návrh *Python* skriptu na riadenie modelu virtuálnej vzducholode musel spĺňať niekoľko podmienok, ktoré sme mu stanovili. Okrem samotnej vizualizácie pohybu musel skript vykonávať aj iné emulačné úlohy. Požiadavky na skript boli:

- Jednoduché ovládanie vymodelovanej vzducholode s využitím funkcií poskytovaných *bpy* knižnicou
- Vytvorenie lokálneho komunikačného spojenia s aplikáciou základňovej stanice a spracovávanie prichádzajúcich riadiacich príkazov
- Deterministické generovanie dát pre potreby emulácie posielania dát z modelu vzducholode
- Autonómnosť vykonávania skriptu od zvyšku *Blender* aplikácie z dôvodu plynulého priebehu vizualizácie objektov a vykonávania samotnej aplikácie

Návrh samotného skriptu pozostával z troch častí, ktorými sme zabezpečili nami stanovené požiadavky. Naše navrhnuté riešenie je iba jedným z mnohých, ktoré by sa dali realizovať. Pre naše potreby bolo takéto riešenie postačujúce a jeho návrh dovoľoval jednoduché zmeny funkcionality a neskoršie rozšírenia. Týmito troma aspektmi skriptu boli:

- Návrh a implementácia triedy typu *modal operator*
- Implementácia oddeleného vlákna
- Implementácia lokálneho *TCP socket* servera

6.2.1 Návrh triedy typu *modal operator*

Základná trieda *Operator (bpy.Types.Operator)* definuje triedu, z ktorej je možné odvodiť a sú vytvorené niektoré nástroje aplikácie *Blender.* Prívlastok *modal* majú potom tie nástroje, ktoré implementujú rovnomennú metódu tejto základnej triedy. Medzi ne patria rôzne translačné, rotačné a iné interaktívne nástroje, pomocou ktorých je možné manipulovať objektom a jeho vlastnosťami v reálnom čase. Vychádzajúc z týchto poznatkov a možností elementárnej triedy *Operator* sme si navrhli odvodenú triedu *AirshipOperator*, ktorej úlohou bolo autonómne zabezpečovať všetku manipuláciu s modelom na základe riadiacich dát z aplikácie základňovej stanice. Nezávislosť a neukončenie vykonávania inštrukcií operátorovej triedy je zabezpečené kľúčovými návratovými hodnotami z jednotlivých metód, ktoré dokážu na základe vnútornej logiky

kľúčových slov sami určiť pôsobnosť operátorovej triedy. Pre nás boli relevantné kľúčové návratové hodnoty:

- *[Running modal]*sa používa ako kľúčové slovo pre definovanie volania metódy *modal*, ktorá sa začne vykonávať po ukončení práve vykonávajúcej metódy. Metóda *modal* je vykonávaná paralelne s *Blender* aplikáciou a jej vykonávanie je opakované dokiaľ nie je metóda zrušená alebo úloha úspešne dokončená
- *[Finished]* sa používa ako kľúčové slovo na ukončenie vykonávania operátorovej triedy po splnení zadanej úlohy
- *[Cancelled]* sa používa ako kľúčové slovo na ukončenie vykonáva operátorovej triedy v prípade vzniku chyby alebo nežiaduceho stavu počas vykonávania niektorej z jej metód
- *[Pass through]* sa používa ako pasívna návratová hodnota, pri použití ktorej operátorova trieda čaká dokiaľ *Blender* aplikácia nevyvolá udalosť svojou činnosťou, čo spôsobí zavolanie *modal* metódy [19]

Návrh odvodenej triedy *AirshipOperator* obsahoval implementáciu nasledujúcich metód, aby mohol poskytovať požadovanú funkcionalitu:

- *Execute()* metóda, prostredníctvom ktorej sa inicializoval vytvorený objekt operátorovej triedy a návratová hodnota pri správnom vykonaní bola nastavená na *[Running modal]*
- *Modal()* metóda, v ktorej bola implementovaná translačná a rotačná interakcia s modelom vzducholode. (Pre potreby plynulosti a odľahčenia záťaže na používateľské prostredie bolo potrebné spracovanie riadiacich príkazov zo základňovej stanice a všetky výpočty implementovať v oddelenom vlákne)

6.2.2 Metóda *modal*

Implementácia metódy *modal* obsahuje logickú programovú časť operátorovej triedy a jej podstatou je vykonávať požadované úlohy. Ak je objekt operátorovej triedy aktívny, vyvoláva sa metóda *modal* pri vzniku akejkoľvek udalosti (*eventu*) vyvolenej v aktívnom okne aplikácie. K týmto udalostiam patria vstupy z klávesnice a myši, ale možná je tiež aktivácia časovačom, ktorý sa aktivuje v pravidelných časových intervaloch podľa jeho implementácie. Práve časovač bol použitý ako autonómny prvok na periodickú aktiváciu *modal* metódy. V *execute* metóde bol pri inicializácii objektu zaregistrovaný časovač príkazom,

```
self.timer = context.window_manager.event_timer_add(0.02, context.window)
```

ktorý aktívnemu oknu aplikácie zaregistruje časovač s periodicitou 20 ms. K objektu časovača bolo možné pristupovať v celej operátorovej triede. Zaregistrovaný časovač vyvolával udalosť, pri ktorej bola zavolaná metóda *modal.* Metóda *modal* je vyvolaná aj pri vzniku akéhokoľvek inej udalosti zo strany iných komponentov aplikácie, čo nie je možné ovplyvniť. Vykonávanie úloh *modal* metódy bolo možné obmedziť podmienkou, pri ktorej sa kontroloval typ udalosti vyvolávajúcej metódu. V nasledujúcej ukážke možno vidieť úryvok metódy *modal*, v ktorej sa nastavujú parametre pozície a rotácie vizualizovaného modelu vzducholode pri vykonávaní metódy vyvolanej časovačom.

```
if event.type == 'TIMER':
      bpy.data.objects['Airship'].location = self.thread.position
      bpy.data.objects['Airship'].rotation_euler = self.thread.rotation
```

Ako je možné jasne vidieť z ukážky zdrojového kódu, bola použitá funkcionalita *bpy* knižnice, ktorou sa sprístupnili objekty a ich atribúty. *Bpy* knižnica pozostáva z viacerých balíčkov, ktoré definujú rôzne druhy prístupov do *Blender* aplikácie. Niektoré z nich boli použité v skripte a ich podstatou je:

- *bpy.context* – obsahuje informácie o aktuálnom pracovnom prostredí, ako sú aktuálna aktívna obrazovka, scéna, okno a manažér okna
- *bpy.data* – poskytuje prístup k vnútorným dátam a objektom *Blender* aplikácie
- *bpy.ops* – poskytuje prístup k volaniu operátorov, čo zahrňuje operátory implementované v jazykoch C, *Python* a tiež k nami navrhnutým operátorom
- *bpy.types*
- *bpy.utils* – obsahuje užitočné funckie špecifikované pre *Blender* aplikáciu, ktoré ale nie sú asociované s vnútornými dátami aplikácie [20]

S týmito možnosťami, ktoré poskytujú balíčky knižnice, je možné jednoducho manipulovať s viacerými vlastnosťami *Blender* aplikácie, ovládať a spravovať viaceré objekty súčasne.

6.2.3 Implementácia vlákna

Oddelené vlákno bolo implementované štandardne, rovnako ako v *Python* jazyku, zdedením navrhovanej triedy z *Python* triedy *threading.Thread.* Hlavným dôvodom pre takýto návrh bolo odľahčenie hlavného vlákna aplikácie pri komunikácii s aplikáciou základňovej stanice, prenose dát a ich spracovávaní. Inštancia vlákna sa vytvára pri

inicializácii objektu operátorovej triedy, aby mohla poskytovať dáta *modal* metóde. Návrh triedy obsahoval metódy, ktorých implementácia zabezpečovala jej funkcionalitu.

- *__init__(self)* metóda obsahuje inicializáciu objektu triedy a nastavenie dátových premenných do počiatočných hodnôt,

```
self.position = [0,0,0]
self.rotation = [0,1.57,3.14]
```

  kde premenná *position* definuje aktuálnu pozíciu objektu v sústave súradníc a premenná *rotation* jeho orientáciu vzhľadom na jednotlivé osy. Počiatočnou hodnotou pre premennú *position* bol zvolený stred sústavy súradníc, kde emulácia začína. Jednotka, v ktorej sa jednotlivé elementy premennej zobrazujú, predstavuje vnútornú jednotku mierky v *Blender* aplikácii. Počiatočná hodnota premennej *rotation* nastavuje horizontálnu a laterálnu orientáciu objektu vzhľadom na používateľa. Na dosiahnutie toho je nastavený počiatočný ofset 1,57 (alebo π/2) radiánu pre os Y, ktorým sa dosiahne horizontálne smerovanie objektu a ofsetu 3,14 (alebo π) radiánu pre os Z, ktorým sa dosiahne natočenie objektu do kladnej časti osi X pri smerovaní vpred.
- *start(self)* metóda obsahuje priradenie pozitívnej hodnoty do *running* premennej, ktorej úlohou je uchovávanie informácie o aktivite vlákna a byť prístupnou z hlavného aplikačného vlákna. Druhou úlohou metódy je spustenie samotného vlákna.
- *stop(self)* metóda obsahuje priradenie negatívnej hodnoty do *running* premennej, čo má za následok ukončenie aktivity vlákna. Metóda je volaná z hlavného aplikačného vlákna pri skončení emulácie.
- *run(self)* metóda obsahuje celú programovú logiku vlákna a je tvorená nekonečnou slučkou *while* vykonávajúc všetky zadefinované úlohy, ktorými boli:
    - Komunikácia s aplikáciou základňovej stanice
    - Spracovanie a transformácia prichádzajúcich dát
    - Generovanie emulačných dát posielaných základňovej stanici

Programové riešenie komunikácie zabezpečovalo vytvorenie a udržiavanie lokálneho *TCP/IP socket* serveru, na ktorý sa aplikácia základňovej stanice pripájala a posielala riadiace dáta. Riešenie tiež zahrňovalo posielanie pripravených emulačných dát aplikácii.

Čítanie prichádzajúcich dát vo forme paketov je uskutočňované v pravidelných časových intervaloch. Prichádzajúce pakety sú uložené do lokálnej premennej pred ich spracovaním a transformáciou. Tvar riadiaceho paketu je totožný z reálnou situáciou, aby bola čo najviac otestovaná funkcionalita. Obsah paketu je najprv spracovaný a rozdelený na jednotlivé dátové zložky, ktoré sú uložené v premennej *dataElem*.

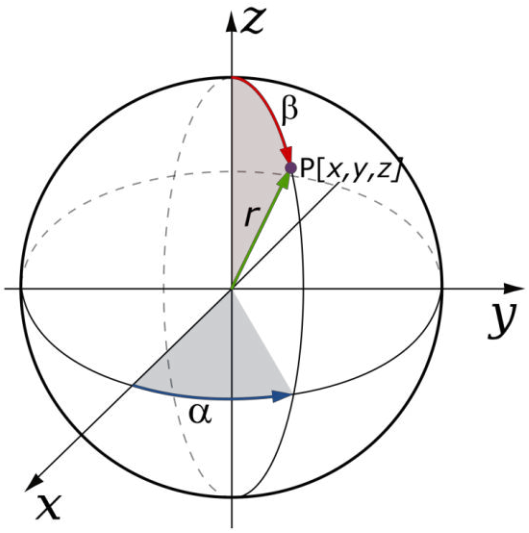
Spracované riadiace dáta sú použité na transformáciu do pohybu modelu vzducholode v *Blender* aplikácii. Prvý a druhý dátový element premennej *dataElem* obsahuje konkrétnu požiadavku na rotáciu v korešpondujúcej osi. Hodnota dátového elementu je použitá na určenie smeru a dĺžky rotácie. Rotačný pohyb a výšková regulácia modelového objektu je uskutočňovaná nepriamo cez modifikáciu premenných vlákna, ktorých hodnoty sa používajú pri reálnej modifikácii v *modal* metóde. *Blender* aplikácia nie je vláknovo bezpečná, čo má za dôsledok to, že priamy prístup a manipulácia z atribútmi aplikácie alebo objektov by mala za následok chybový stav aplikácie.

Z ukážky transformácie jedného z dátových elementov je možné zrozumiteľnejšie pochopiť spôsob samotnej transformácie.

```
if int(dataElem[0]) <= 20:
    self.rotation[2] += 0.01
if int(dataElem[0]) > 20 and int(dataElem[0]) <= 40:
    self.rotation[2] += 0.005
```

Pozícia dátového elementu v poli *dataElem* definuje jeho príslušnosť k rotácii v konkrétnej osi. Pri transformácii sa zohľadňuje analógová hodnota riadiaceho príkazu, ktorá definuje veľkosť rotácie. Hodnoty dátových elementov pre rotáciu sa pohybujú v rozmedzí <0,100>, kde hodnota 50 bola definovaná ako neutrálna. V takom prípade sa riadiaci príkaz netransformuje do rotačného pohybu. Zmena rotácie vychádzajúca z jedného príkazu je nízka hodnota, z dôvodu zabezpečenia plynulého pohybu vizualizovaného objektu. Tomu bolo potrebné prispôsobiť aj vysokú frekvenciu čítania prichádzajúcich dát.

Translačný pohyb vpred je komplikovanejší z hľadiska pohybu objektu v trojrozmernom priestore, čo má za následok zmenu jeho pozície v každej z osí. Výsledná translačná priamka závisí na počiatočnej pozícii objektu a jeho orientácii vzhľadom na jednotlivé osi. Z všeobecného hľadiska sa prezentovaný výsledný bod nachádza v sférickom súradnicovom systéme, ktorý treba prekonvertovať do Karteziánskeho.



Obrázok 32 Transformácia sférických koordinátov na Karteziánske súradnice

Priamka spájajúca počiatočnú pozíciu a výsledný bod *P* je vektorom s dĺžkou *r*. Pokiaľ je uhol β meraný medzi osami *Y* a *Z*, a uhol α meraný medzi osami *X* a *Y*, potom platia nasledujúce vzťahy:

$$x = r * cos\beta * sin\alpha$$

$$y = r * cos\beta * cos\alpha$$

$$z = r * sin\beta$$

Výsledky jednotlivých rovníc definujú súradnice výsledného bodu pri počiatočnej pozícii v strede sústavy súradníc (bod [0,0,0]). Implementácia tejto transformácie bola zahrnutá v skripte pre výpočet pohybu modelu.

```
self.position[0] -= 0.006 * math.cos(self.rotation[1] - 1.57)
                    * math.cos(self.rotation[2])
self.position[1] -= 0.006 * math.cos(self.rotation[1] - 1.57)
                    * math.sin(self.rotation[2])
self.position[2] += 0.006 * math.sin(self.rotation[1] - 1.57)
```

Hodnoty jednotlivých zložiek premennej *position* sú upravované v závislosti od výsledkov rovníc, v ktorých sa zohľadňuje aktuálne natočenie objektu, čo sa nachádza v premennej *rotation*. V rovniciach bola zahrnutá aj úprava aktuálneho uhla o zníženie jeho hodnoty podľa počiatočného ofsetu danej osi. Rovnako ako pri rotácii, bola zmena pozície vyplývajúca z jedného riadiaceho príkazu definovaná ako nízka hodnota, z dôvodu vysokej periodicity vykonávania translácie.

Generovanie emulačných dát (a ich príprava do podoby dátových paketov) sa uskutočňuje deterministicky, podľa preddefinovaných vzťahov. Emulačné dáta boli rozdelené na typy:

- Informačný dátový paket zahrňujúci informácie o vlastnostiach jednotlivých dátových elementov, ktoré budú posielané ako dáta senzorickej dosky
- Senzorické dáta, ktorými sa emulovalo správanie senzorickej dosky
- Kontrolné dáta, ktorými sa emulovali dáta o aktuálnych atribútoch modelu vzducholode

Generovanie senzorických dát sa neuskutočňovalo s prihliadnutím k nejakým reálnym podmienkam, pretože emulovanie atmosféry a jej vlastností bolo považované nad rámec potrieb emulácie. Veličiny a ich dáta boli prevzaté z vlastností modelového objektu, pri ktorých bol predpoklad na ich častú zmenu. Týmto referenčným atribútom bola zvolená pozícia modelového objektu, ktorá je transformovaná na rôzne senzorické dáta. V ukážke možno vidieť čítanie týchto dát z objektu a samotnú prípravu dátového paketu.

```
self.dataPacket = 'd;'
self.dataPacket +=  'Vyska:'
      + str(round(bpy.data.objects['Airship'].location[2], 2)) + ';'
self.dataPacket += 'Sirka:'
      + str(round(bpy.data.objects['Airship'].location[1], 2)) + ';'
self.dataPacket += 'Teplota:'
      + str(abs(round(bpy.data.objects['Airsip'].location[0], 2))*10)
```

Generovanie kontrolných emulačných dát bolo založené na vlastnostiach vizualizovaného objektu s prihliadnutím na reálne údaje poskytované modelom vzducholode. Z atribútov objektu bolo možné získať údaje, ktorých hodnoty boli v závislosti od riadenia korešpondujúce s údajmi poskytovanými vzducholoďou. Atribútmi, ktoré nám poskytoval modelový objekt a mohli sme kontrastovať so skutočnými, boli:

- Hodnota uhlov v osiach *X* a *Y* ako údaje o vychýlení modelu z vodorovnej polohy
- Pozícia objektu v emulovanom prostredí ako údaj z globálneho lokalizačného systému (*GPS*)
- Hodnota uhla v osi *Z* ako údaj z kompasu o smerovaní modelu vzducholode

Príprava dátového paketu bola totožná ako v prípade senzorického dátového paketu. V ukážke možno vidieť údaje o pozícii objektu, ktoré neboli vkladané do paketu priamo, ale vyžadovala sa ich úprava.

```
bpy.data.objects['Airship'].location[0] * 0.0001 + self.coordinateX
-bpy.data.objects['Airship'].location[1] * 0.0001 + self.coordinateY
```

Údaje z *GPS* senzora obsahujú informácie o zemepisnej šírke a dĺžke sledovaného objektu. V prípade emulácie bola počiatočná pozícia modelového objektu [0,0] a menila sa jeho pohybom. Pre potreby emulácie bolo potrebné zahrnúť statickú počiatočnú zemepisnú polohu, ktorá sa nachádzala v premenných *coordinateX* a *coordinateY* a upravovala hodnoty pozície objektu pri vkladaní do paketu, aby korešpondovali s nejakou reálnou polohou. Taktiež sa upravil vplyv pozície modelového objektu na samotnú polohu, aby jeho zmeny približne korešpondovali so zmenami zemepisnej šírky a dĺžky v reálnych podmienkach.

6.2.4 Implementácia lokálneho *TCP/IP socket* serveru

Jedným z ďalších dôvodov, prečo sme sa rozhodli použiť *Blender* aplikáciu ako základ pre naše testovacie rozhranie a emuláciu pohybu bola možnosť využitia skriptovania v jazyku *Python* pre potreby návrhu komunikačného kanála. Základňová stanica používana používateľom je definovaná v reálnych podmienkach ako klientska časť komunikačného rozhrania a softvér na modeli vzducholode ako server, na ktorý sa klient pripája. *Blender* aplikácia dovoľovala definovanie komunikačného rozhrania s ekvivalentnými atribútmi ako reálne cez implementáciu lokálneho *TCP/IP socket* serveru. Týmto spôsobom sme pri našom návrhu emulovali nielen pohyb vzducholode, ale aj zjednodušenú emuláciu komunikačného rozhrania.

```
serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
serverSocket.bind(('127.0.0.1', 9013))
```

Server zadefinoval TCP/IP *socket* s prenosom cez prúdový *socket* a zaregistroval sa na počúvanie na *localhost* adrese (*127.0.0.1*) s preddefinovaným portom.

6.3 Experimentálne testovanie systému

Experimentálne testovanie funkcionalít systému sme uskutočnili dvoma spôsobmi. Prvou metódou bolo testovanie prostredníctvom zrealizovaného testovacieho rozhrania, ktoré sme použili na otestovanie všetkých podsystémov. Testovacie rozhranie bolo spustené ako externá aplikácia k aplikácii základňovej stanice, ktorú sme testovali.

Spustením testovacieho skriptu bol inicializovaný objekt modelu vzducholode a vytvoril sa serverový *socket*, ktorý očakával lokálne pripojenie na porte *9013*. Už počas vývoja aplikácie sme testovali čiastkové riešenia jednotlivých podsystémov, ale tentoraz sme sledovali funkcionalitu a kooperáciu všetkých častí systému. Postupne sme ako používateľ aplikácie pracovali s jej jednotlivými komponentmi a sledovali odozvu na strane testovacieho rozhrania a tiež na strane aplikácie.

Testovanie riadiaceho podsystému sa uskutočnilo prostredníctvom oboch riadiacich prostriedkov (virtuálny ovládač a externý ovládač), interakcia s ktorými sa prenášala na pohyb modelu vzducholode v testovacom prostredí. Sledovala sa očakávaná odozva a správanie modelu. Okrem toho sa sledovali aj adekvátne zmeny polohy a natočenia vzducholode zobrazované v aplikácii prostredníctvom mapy a ikony vzducholode. Podnetom na tieto zmeny bol pohyb modelu vo virtuálnom prostredí, ktoré testovací skript transformoval na zmenu zemepisnej polohy a natočenia.

Komunikačný podsystém bol testovaný počas celého používania aplikácie pripojenej k testovaciemu rozhraniu. Sledovali sme správny prenos riadiacich paketov transformovaných na pohyb a zároveň čítanie senzorických a kontrolných dát generovaných testovacím skriptom. Tvar a obsah týchto paketov bol čo najtotožnejší s realitou, preto sme mohli čo najvierohodnejšie otestovať prenos dát v emulovanom prostredí. Okrem toho sa otestovalo aj spracovanie týchto paketov aplikáciou a ich vizualizácia.

Ďalšou časťou testovania boli funkcionality aplikácie súvisiace s nastavovaním aplikácie, správnym zobrazovaním aktuálnych dát, grafov a vykresľovaním štatistického grafu. Tieto funkcie nekomunikovali s testovacím rozhraním, iba využívali prostriedky ním poskytované v podobe dát.

Druhá metóda testovania bolo zrealizovaná s reálnym komunikačným kanálom prostredníctvom mikropočítača *Raspberry Pi 3*, ktorého integrovaný *Wi-Fi* modul sme použili ako hardvérový prostriedok na komunikáciu. Do mikropočítača sme implementovali realizované skripty komunikačného rozhrania doplnené o algoritmy na generovanie senzorických dát, aby sme mohli otestovať obojsmerný prenos dát. Mikropočítač sa pripájal k sieti vytváranej počítačom, na ktorom sa nachádzala aplikácia. Na rozdiel od testovacieho rozhrania sme nemali k dispozícii vizualizačný prostriedok prenášaných riadiacich dát, ktoré sme mohli sledovať iba ako zoznam prichádzajúcich

paketov ukladaných do súboru. Algoritmy na generovanie senzorických dát vytvorili obdobné podmienky (v porovnaní s testovacím prostredím) aplikované na reálny prenos dát. Súčasťou testovania bolo tiež overenie správania pri prerušení spojenia a jeho opätovnom nadviazaní.

Záver

Cieľom našej diplomovej práce bolo navrhnúť a realizovať riadiaci systém modelu vzducholode, ktorý umožní vzdialené ovládanie vzducholode súčasnou implementáciou viacerých technológii. Práca bola rozdelená na realizáciu viacerých podsystémov, ktoré spolu vytvárali kompletný systém na strane používateľa.

Úspešne sa nám podarilo takýto systém navrhnúť a zrealizovať. Systematicky sme sa venovali vypracovaniu riešenia podsystémov, ktorými boli komunikačné rozhranie, riadiace rozhranie na strane používateľa a aplikácia základňovej stanice.

V úvodnej časti práce sme špecifikovali jednotlivé úlohy zadania a zadefinovali postup ich riešenia. Ďalej sme sa venovali všeobecnej a technickej analýze vzducholode ako prostriedku na riešenie úloh IKT a popisu vlastností nami použitého modelu vzducholode. Súčasťou spracovania bol tiež výber a popis hardvérových prostriedkov pre riešenie podsystémov. Jedným z prostriedkov komunikačného podsystému sme zvolili *GSM* modul *Dell Wireless 5540.* Externým riadiacim prostriedkom sme zvolili ovládač *Playstation 4 Dualshock.* V ďalšej časti práce sme sa venovali návrhu riadiaceho podsystému. V úvode kapitoly sme rozanalyzovali riadiace prvky modelu vzducholode, na základe ktorých sme vypracovali návrh modelu riadenia. Model riadenia bol následne použitý pri implementácii riadenia prostredníctvom virtuálnych komponentov aplikácie a externého ovládača. Návrhu a realizácii komunikačného podsystému sme sa zaoberali v nasledujúcej časti práce. Riešenie sme založili na komunikácii prostredníctvom *TCP socketov* medzi komunikačnými rozhraniami nachádzajúcimi sa na vzducholodi a v aplikácii základňovej stanice. Pre prenos dát z modelu vzducholode sme zvolili technológiu *Wi-Fi* a *GSM.*

Zrealizované podsystémy boli použité v nasledujúcej časti práce, v ktorej sme sa venovali vývoju aplikácie základňovej stanice, ktorej úlohou bolo sprístupnenie riadiaceho systému používateľovi. Ako programovací jazyk sme zvolili jazyk *Java* a aplikáciu sme realizovali vo vývojovom prostredí *NetBeans IDE*. Návrh a implementáciu aplikácie sme rozdelili na grafické rozhranie a programové rozhranie. V aplikácii sme zrealizovali komunikačné rozhranie na prenos dát so vzducholoďou, riadiace rozhranie na jej ovládanie prostredníctvom virtuálnych komponentov a externého ovládača, spracovanie a vizualizáciu senzorických dát v podobe aktuálnych hodnôt a grafov. Ďalšími implementovanými časťami aplikácie bola vizualizácia polohy prostredníctvom mapy

a grafické zobrazenie priebehu senzorických veličín. Súčasťou aplikácie boli aj možnosti pre nastavovanie vzhľadu aplikácie a vlastností jednotlivých rozhraní.

Riešenia jednotlivých podsystémov a výsledné riešenie aplikácie bolo potrebné podrobiť testovaniu a z toho dôvodu sme navrhli a zrealizovali testovacie rozhranie v aplikácii *Blender*. V aplikácii sme vymodelovali vzducholoď a navrhli skript testovacieho rozhrania, prostredníctvom ktorého sme mohli testovať riadiaci podsystém emulovaním pohybu vzducholode vo virtuálnom prostredí. Funkcionalita komunikačného podsystému bola otestovaná s využitím implementácie lokálneho *TCP* serveru, ktorým sme simulovali reálny server na vzducholodi. Prenos riadiacich dát a senzorických dát, ktoré sme generovali v prostredí testovacieho rozhrania, boli vlastnosťami porovnateľné z reálnymi podmienkami. Testovacie rozhranie nám dovoľovalo otestovať aj funkcionalitu aplikácie základňovej stanice a jej jednotlivých častí. Otestovanie podsystémov sme uskutočnili aj v reálnych podmienkach s využitím mikropočítača ako serverovej časti.

Výsledne riešenie systému spĺňalo všetky podmienky, stanovené v úvodnej časti práce a v niektorých smeroch ich aj prekonávalo. V procese realizácie sme si rozšírili schopnosti abstraktného prístupu k riešeniu úloh, vhodného výberu prostriedkov a ich použitia a návrhu komplexného systému pozostávajúceho z viacerých podsystémov, ktoré spolu spolupracujú. V práci by sa dalo pokračovať a rozširovať funkcionality jednotlivých podsystémov, ako napríklad návrh alternatívnych metód prenosu dát, rozšírenie prenosu senzorických dát o obrazový prenos alebo rozšírenie aplikácie základňovej stanice na iné platformy.

Zoznam použitej literatúry

[1] SEBBANE, Yasmina. *Lighter than Air Robots, Guidance and Control of Autonomous Airships.* Springer, 2012. 251 s. ISBN 978-94-007-2662-8.

[2] SONY INTERACTIVE ENTERTAINMENT. *Playstation 4Wireless controller Dualshock 4* [online]. 2017. Dostupné z: <https://www.playstation.com/en-us/explore/accessories/dualshock-4-wireless-controller-ps4/>

[3] LI, Yuwen. *Dynamics Modeling and Simulation of Flexible Airships, Dissertation thesis*. Department of Mechanical Engineering McGill University, Montreal, 2008. 148 s.

[4] FURHT, Borko; PHOENIX, Chris; YIN, John; AGANOVIC, Zijad. *Internet architectures for application service providers* [online]. Dostupné z: <https://pdfs.semanticscholar.org/1841/48d0cabe9d0f36e9981d3ab4379fe0f9b965.pdf>

[5] THAKUR, Dinesh. *OSI (Open Systems Interconnection) Reference Model* [online]. Dostupné z: <http://ecomputernotes.com/computernetworkingnotes/communication-networks/osi-layers>

[6] POPPE, Václav. *Protokoly TCP/IP* [online]. 1997. Dostupné z: <http://www.dnp.fmph.uniba.sk/tcpip/>

[7] STEVENS, Richard W.; RUDOFF, Andrew M.; FENNER, Bill. *UNIX Network Programming, The Sockets Networking API, 3rd Edition.* Addison-Wesley, 2004, ISBN 0-13-141155-1

[8] PYTHON SOFTWARE FOUNDATION. *Socket – Low-level networking interface* [online]. 2017. Dostupné z: <https://docs.python.org/2/library/socket.html>

[9] SHANGHAI SIMCOM WIRELESS SOLUTIONS LTD. *AT Command Set* [online]. 2012. Dostupné z: <http://www.mt-system.ru/sites/default/files/simcom_sim5320_atc_en_v1.23.pdf>

[10] *Dizajn užívateľského rozhrania ako rozhodujúci faktor pre jednoduché používanie aplikácii* [online]. Dostupné z: <http://design.krea.sk/clanky/uzivatelske-rozhrania-ui/>

[11] *Java – Overview* [online]. Dostupné z: <https://www.tutorialspoint.com/java/java_quick_guide.htm>

[12] ORACLE CORPORATION. *NetBeans IDE – The Smarter and Faster Way to Code* [online]. 2017. Dostupné z: <https://netbeans.org/features/>

[13] ORACLE CORPORATION. *Package javax.swing* [online]. 2016. Dostupné z: <https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>

[14] SUGRUE JAMES. *Observer Pattern Tutorial with Java Examples* [online]. 2010. Dostupné z: <https://dzone.com/articles/design-patterns-uncovered>

[15] SIRONI GIORGIO. *The Model-View-Controller pattern in PHP* [online]. 2010. Dostupné z: < https://dzone.com/articles/model-view-controller-pattern>

[16] NAVA GABRIEL. *JInput* [online]. 2016. Dostupné z: <https://github.com/jinput/jinput/wiki>

[17] GILBERT DAVID. *JFreeChart* [online]. 2014. Dostupné z: <http://www.jfree.org/jfreechart/>

[18] GOOGLE INC. *Google Static Maps Developer Guide.* 2017. Dostupné z: < https://developers.google.com/maps/documentation/static-maps/intro#location>

[19] BLENDER FOUNDATION. *Operator(bpy_struct)* [online]. Dostupné z: <https://docs.blender.org/api/blender_python_api_2_73a_release/bpy.types.Operator.html#modal-execution>

[20] BLENDER FOUNDATION. *Application modules* [online]. Dostupné z: <https://docs.blender.org/api/2.78b/#application-modules>

ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

Riadiaci systém pre komunikáciu a prenos dát medzi základňovou stanicou a modelom vzducholode

(Diplomová práca)

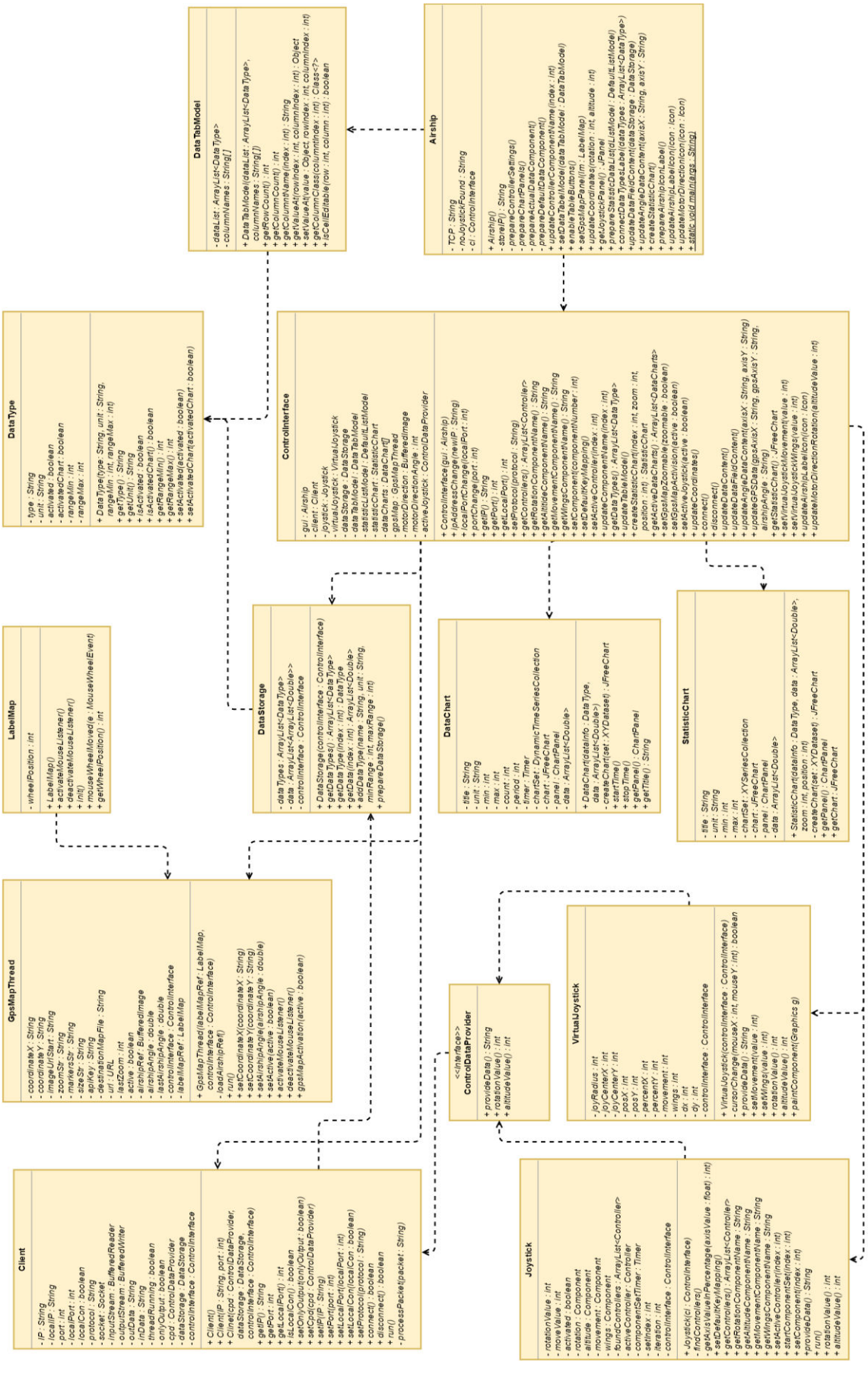
(prílohy)

Študijný odbor:	9.2.4. počítačové inžinierstvo
Študijný program:	Počítačové inžinierstvo
Školiace pracovisko:	Katedra technickej kybernetiky
Stupeň kvalifikácie:	Inžinier (Ing.)
Ministerské číslo práce:	28360820172387

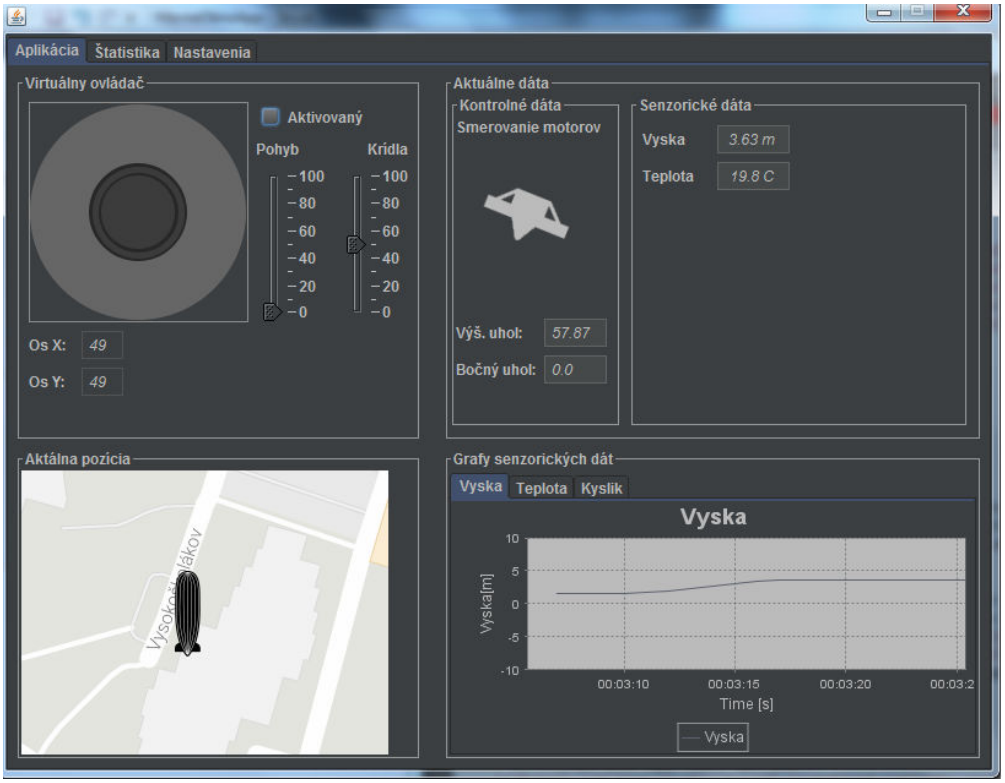
Žilina 2017 **Bc. Matej Jakab**

Zoznam príloh

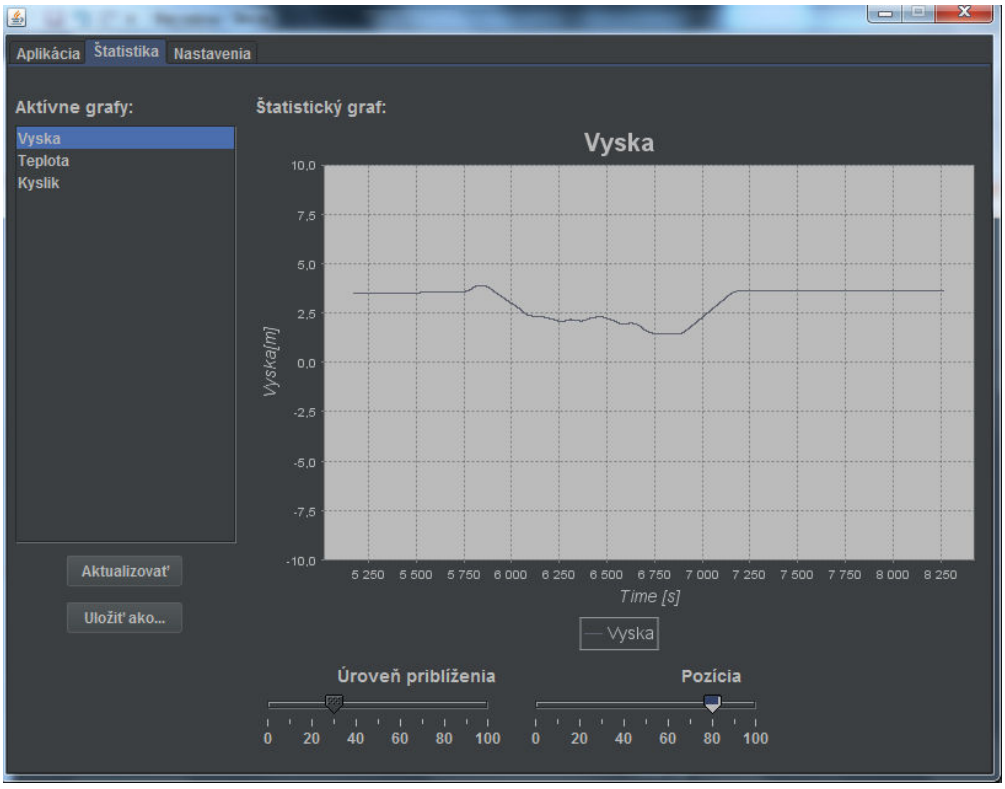
Príloha A: UML diagram tried aplikácie



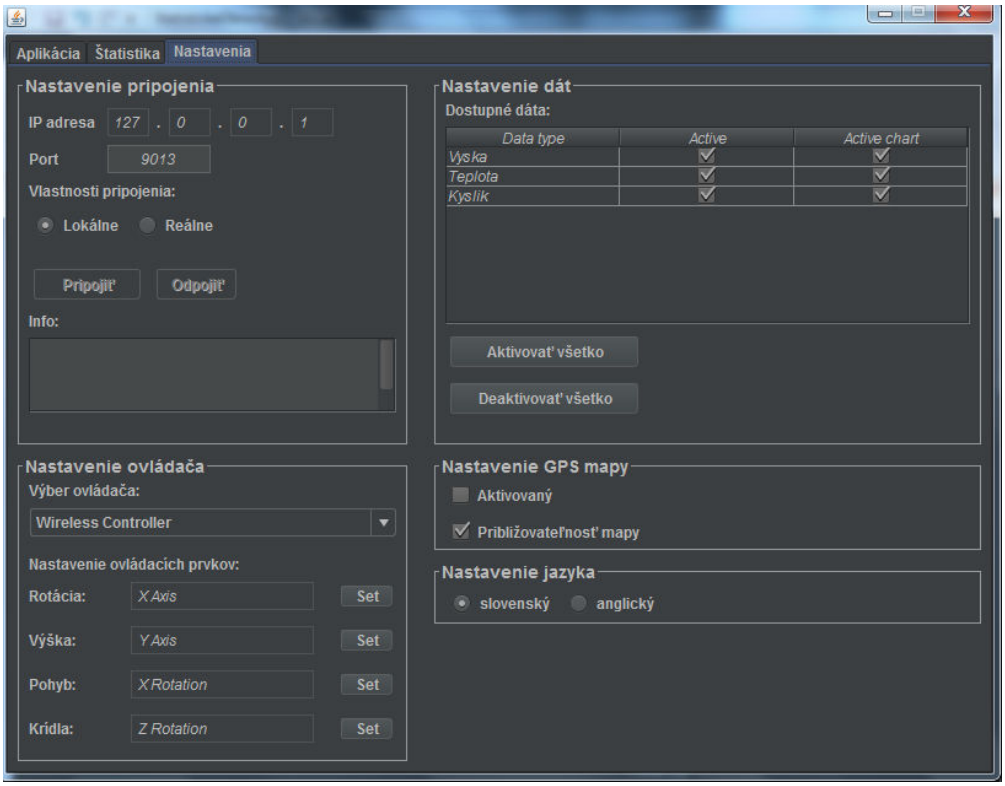
Príloha B: GUI aplikácia základňovej stanice



Hlavné okno aplikácie základňovej stanice



Okno štatistických grafov aplikácie



Okno nastavení aplikácie

Príloha C: CD s elektronickou verziou diplomovej práce a inými materiálmi

Súbory na priloženom CD:

- Diplomová práca vo formáte PDF
- Aplikácia základňovej stanice
- Zdrojový kód aplikácie základňovej stanice
- Zdrojový kód skriptov komunikačného rozhrania modelu vzducholode
- Zdrojový kód skriptu testovacieho rozhrania
- Obrázková dokumentácia
- Model vzducholode vytvorený v aplikácii *Blender*