Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

DIPLOMOVÁ PRÁCE

Měřicí a poradní systém pro větrání
v historických budovách

Plzeň, 2010 Michal Široký

PROHLÁŠENÍ

Předkládám k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 21. května 2010

Anotace

Tato diplomová práce se zabývá návrhem a realizací měřicího a poradního systému pro udržování příznivého klimatu v historických budovách pomocí monitorování parametrů vzduchu a doporučování správného způsobu větrání. Úloha je řešena na konkrétním případě budovy konventu bývalého cisterciáckého kláštera v Plasích.

Práce se věnuje popisu problémů památkových budov s vysokou vzdušnou vlhkostí a přichází s možností jejich řešení. Je zde popsán návrh bezdrátového měřicího systému s použitím hardware *M-Board* vyvinutého na Katedře kybernetiky ZČU a speciálně vyvinutého software v jazyce *Java*. Realizovaný měřicí systém je umístěn a úspěšně provozován v budově konventu NKP Klášter Plasy. Výsledky jsou uvedeny v závěru této práce.

Klíčová slova: klášter Plasy, konvent, vlhkost, teplota, rosný bod, voda, vzduch, větrání, M-Board, Microlog, Java, ZigBee, XML-RPC, PDF, SVG, XML, bezdrátový měřicí systém, NetBeans, Eclipse, IAR.

Annotation

This diploma thesis concerns the design and realization of measurement and expert system to keep favourable climatic conditions inside of historical buildings by monitoring parametres of air and recommending the correct manners of ventilation. This problem is being solved on the example of the convent building of the former Cistercian monastery in Plasy.

The work describes the problem of high air humidity which is common in most of historical buildings and comes with a solution. It describes design of wireles measurement system which uses a hardware device called *M-Board* – developed in the Department of Cybernetics of the University of West Bohemia – and custom-made software in *Java* language. Implemented measurement system has been sucessfully installed and run in convent of the Plasy Monastery. The results are presented in the conclusion of this thesis.

Key words: Plasy monastery, convent, humidity, temperature, dew point, water, air, ventilation, M-Board, Microlog, Java, ZigBee, XML-RPC, PDF, SVG, XML, wireless measurement system, NetBeans, Eclipse, IAR.

Na tomto místě děkuji vedoucímu této práce, panu *Doc. Ing Eduardu Janečkovi, CSc* a dále pak *Ing. Ondřeji Ježkovi* a *Ing. Tomáši Pernému* z KKY FAV ZČU, se kterými jsem konzultoval některé postupy ve své práci a také zaměstnancům NKP Klášter Plasy: *Mgr. Pavlu Duchoňovi*, *Mgr. Sylvě Kročákové*, *Zdeňku Formanovi* a *Martině Hoškové*, kteří dopomohli úspěšnému zprovoznění vyvinutého měřicího systému v budově konventu plaského kláštera a také *Josefu Řehákovi* z firmy SPELEO.

Obsah

1 Úvod

1.1 Cíle práce

Cílem této diplomové práce je navrhnout a realizovat měřicí a poradní systém, který usnadní udržování příznivého klimatu v historických a památkových budovách pomocí monitorování parametrů vzduchu a doporučování správného způsobu větrání.

1.2 Problémy klimatu v památkových budovách

V této části je třeba upřesnit, pro jaký typ budov je systém zamýšlen. Památkovou, či historickou budovou, je myšlena starší stavba zařazená do památkové ochrany. S takovými budovami se zpravidla pojí následující vlastnosti:

- Stavba má silné obvodové zdivo, většinou kamenné, nebo cihlové. Chybí tepelná izolace.
- S ohledem na finanční náročnost údržby takové stavby, nejsou většinou interiéry přes zimu vytápěny.
- V budově je velmi často zanedbané větrání.
- V minulosti byly na budově prováděny konstrukční úpravy, které mohly přispět k většímu nasákání zdiva vodou, či zhoršení možností ventilace interiéru.
- Původní znalosti o způsobu údržby budovy buď chybí, nebo není možné je při současném stavu budovy aplikovat.

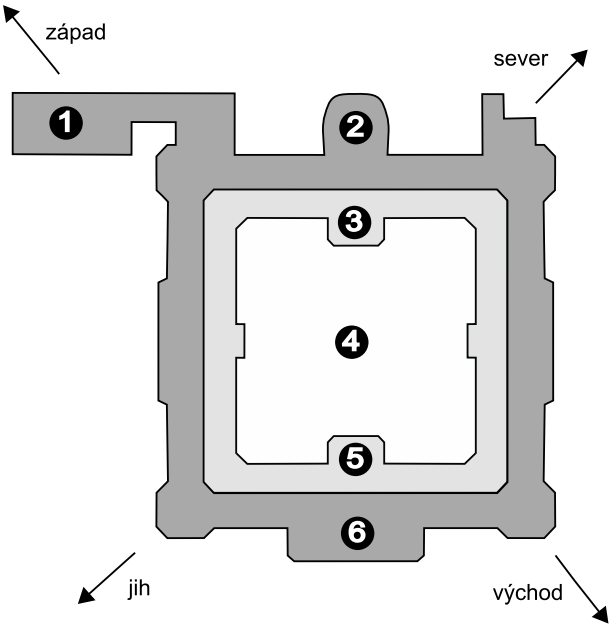
Jako konkrétní případ takové budovy může sloužit konvent bývalého cisterciáckého kláštera v Plasích.

1.3 Klášter Plasy

Klášter v Plasích (cca 25 km severně od Plzně) byl založen roku 1144 knížetem Vladislavem II. Cisterciáčtí mniši, kteří následně do Plas přišli, založili, v duchu stavební tradice svého řádu, klášterní budovy v údolní nivě řeky Střely. Původní budovy byly dřevěné, od počátku 13. století již kamenné. Současná podoba kláštera pochází z největší části z 18. století.

Budova konventu, která je předmětem zájmu této práce, byla vystavěna v barokním slohu v 1. polovině 18. století a jejími architekty byli Jan Blažej Santini a Kilián Ignác Dientzenhoffer. Konvent je založen na bažinatém podloží. Pod traktovými zdmi budovy je zatlučeno zhruba 5100 dubových pilot a na nich je položený dubový rošt. Teprve na tomto roštu stojí veškerá váha zdiva. Dřevěný rošt je chráněn před hnitím tak, že je neustále potopen pod vodou, která je pod budovu přiváděna pomocí důmyslného systému kanálů. Pro sledování stavu vody v základech slouží dva bazény umístěné v rizalitech jižního a severního schodiště. Tyto bazény se nazývají *vodní zrcadla*. Přítomnost pomalu proudící vody pod budovou a otevřené vodní hladiny uvnitř budovy s sebou nese několik problémů. Jedním z nich je trvale nízká (avšak poměrně stálá) teplota v přízemním patře konventu. Teplota v přízemí je z velké části určována teplotou vody, která se podle ročního období

pohybuje v rozmezí 5 a 15°C. Teplota vzduchu v přízemí může vystoupat až na 18°C. Voda ze zrcadel se odpařuje do vzduchu a zvyšuje jeho vlhkost. Vzduch, který do budovy přichází z venčí a dostane se do styku se zdmi v přízemí se ochladí a pokud předtím obsahoval příliš mnoho vodní páry, vlhkost z tohoto schlazeného vzduchu na stěnách zkondenzuje. Kondenzace vody na stěnách a ve dřevěné konstrukci hlavního schodiště je velmi nežádoucí. V této práci se budu zabývat metodou, která umožní minimalizovat riziko kondenzace vlhkosti v budově.



Obrázek 1: Půdorys budovy konventu

Na obrázku 1 je znázorněna budova konventu z vyznačením jejích hlavních částí:

1. Nemocniční křídlo
2. Kapitulní síň
3. Rizalit severního schodiště a severní vodní zrcadlo
4. Rajský dvůr – Povrch rajského dvora je oproti vnějšímu terénu úmyslně navýšen o cca 3 metry. Hmota navezené zeminy slouží mimo jiné k akumulaci tepla v letních měsících.
5. Rizalit jižního schodiště a jižní vodní zrcadlo
6. Rizalit zimní a letní jídelny

Světle šedá plocha na půdorysu vyznačuje tzv *ambitové* chodby a šachty hlavních schodišť, které jsou hlavními cestami proudění vzduchu v budově. Na chodbách se nachází celkem zhruba 150 oken, která je možné využívat k větrání. Manipulace s okny je plně manuální a při otevírání či zavírání všech oken v 1. a 2. patře je v budově nutné ujít cca 600m.

Obrázek 2: Pohled na jihozápadní křídlo konventu přes rajský dvůr

1.4 Předchozí řešení problému

Ve své bakalářské práci z roku 2008 jsem se zabýval návrhem a realizací prototypového technického zařízení, které bylo následně od 13. dubna do 5. června umístěno v konventu a nepřetržitě zaznamenávalo teplotu a relativní vlhkost vzduchu v pětiminutovém intervalu. Základem tohoto zařízení byl průmyslový počítač *WinCon-8731* vybavený dvěma čidly relativní vlhkosti a teploty vzduchu *Humirel HTM1735*. Pomocí *WinConu* byly měřeny vlastnosti vzduchu na chodbě v 1. nadzemním podlaží a u jihovýchodní stěny ve výšce cca 10m nad zemí.

Soustava *WinConu* a čidel byla dále doplněna dvěma dataloggery relativní vlhkosti a teploty vzduchu *Microlog*, které jsou v majetku NKP Klášter Plasy.

Pomocí těchto přístrojů bylo možné získat dostatečné množství dat vypovídajících o chování vzduchu v budově a reakcích budovy na různé způsoby větrání.

1.4.1 Shrnutí předchozích výsledků

Podrobnější informace o budově konventu, použitých zařízeních a získaných výsledcích v roce 2008 lze nalézt ve zmiňované bakalářské práci [1]. Zde se budu věnovat jen stručnému shrnutí získaných výsledků.

Pro usnadnění pochopení dalšího textu je ještě nutné uvést a vysvětlit některé pojmy, které se týkají kvality vzduchu a se kterými se bude dále pracovat. Znalosti týkající se problematiky vlhkosti vzduchu jsem získal v [5]. Více je tato problematika rozvedena v mé bakalářské práci [1].

- *Absolutní vlhkost* (ϱ_p) – Hmotnost vodní páry obsažené v $1m^3$ vlhkého vzduchu
- *Relativní vlhkost* (φ, RH) – Poměr množství par aktuálně obsažených ve vzduchu k maximálnímu množství par, které může být ve vzduchu obsaženo při dané teplotě.
- *Rosný bod* – Stav, při kterém je vzduch nasycen vodními parami. Pod pojmem *rosný bod* se rozumí i teplota, při které tento stav nastane.

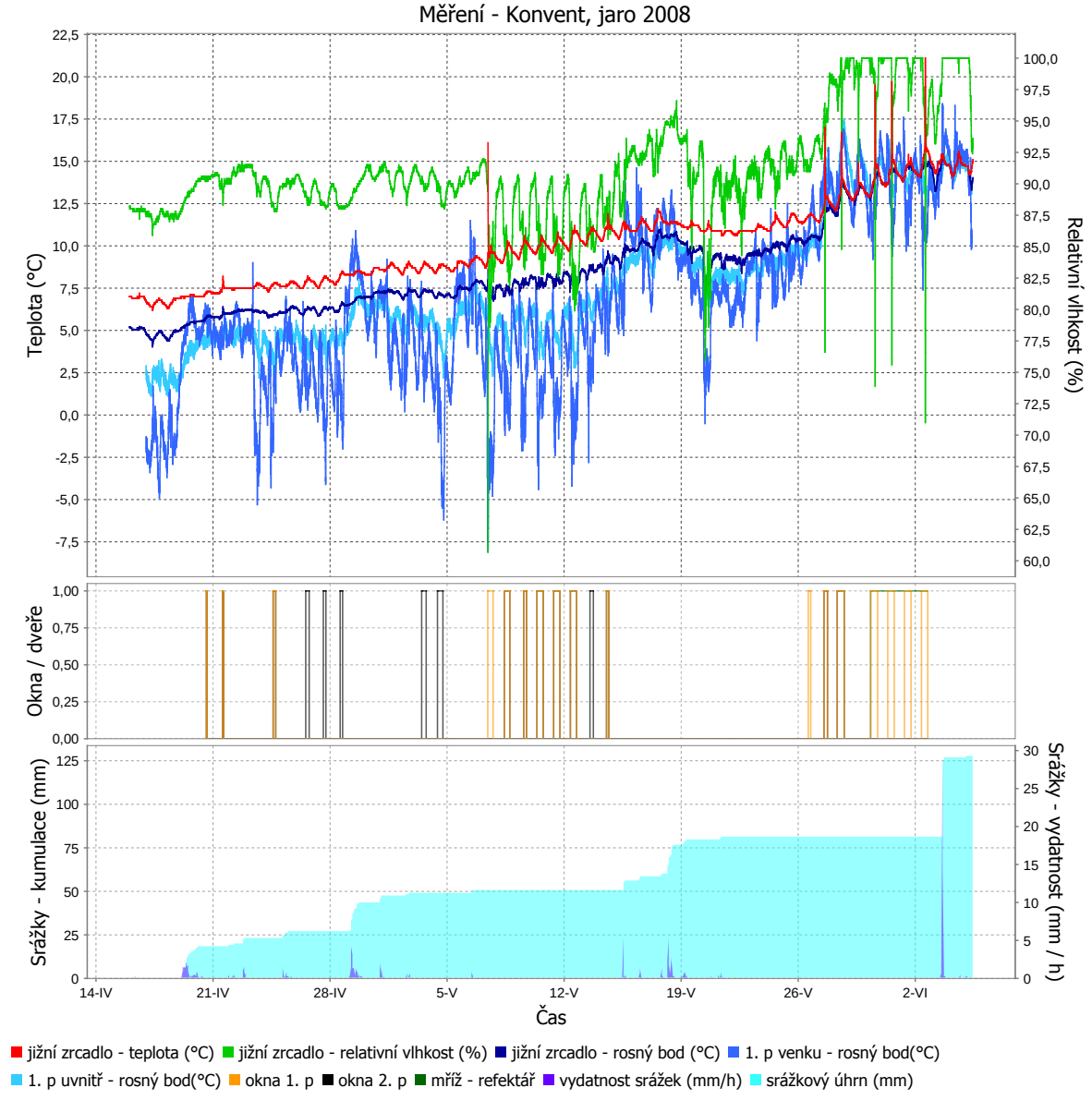
Cílem předchozí bakalářské práce bylo vytvořit pravidla pro správné větrání v konventu tak, aby se během jarních a letních měsíců podařilo budovu vnějším vzduchem

co nejvíce vyhřát a současně zamezit kondenzaci vzdušné vlhkosti v přízemním patře budovy. Pravidla formulovaná v závěru bakalářské práce byla následující:

1. Aby se problém s vlhkostí vzduchu v přízemí nezvyšoval, je možné větrat pouze tehdy, když je rosný bod vnějšího (tedy vstupního) vzduchu nižší, než je teplota vzduchu případně stěn u vodních zrcadel, což jsou nejstudenější místa v budově.

2. Aby došlo k žádoucímu vyhřátí budovy, mělo by se větrat tehdy, když je teplota vnějšího vzduchu vyšší, než teplota vzduchu ve druhém patře, což je nejteplejší místo v budově.

Ideální je větrat tehdy, pokud jsou obě výše uvedená kritéria splněna. Zatímco splnění druhého z nich je možné ověřit i bez měřicích přístrojů, ověření první podmínky již vyžaduje použít technické vybavení.



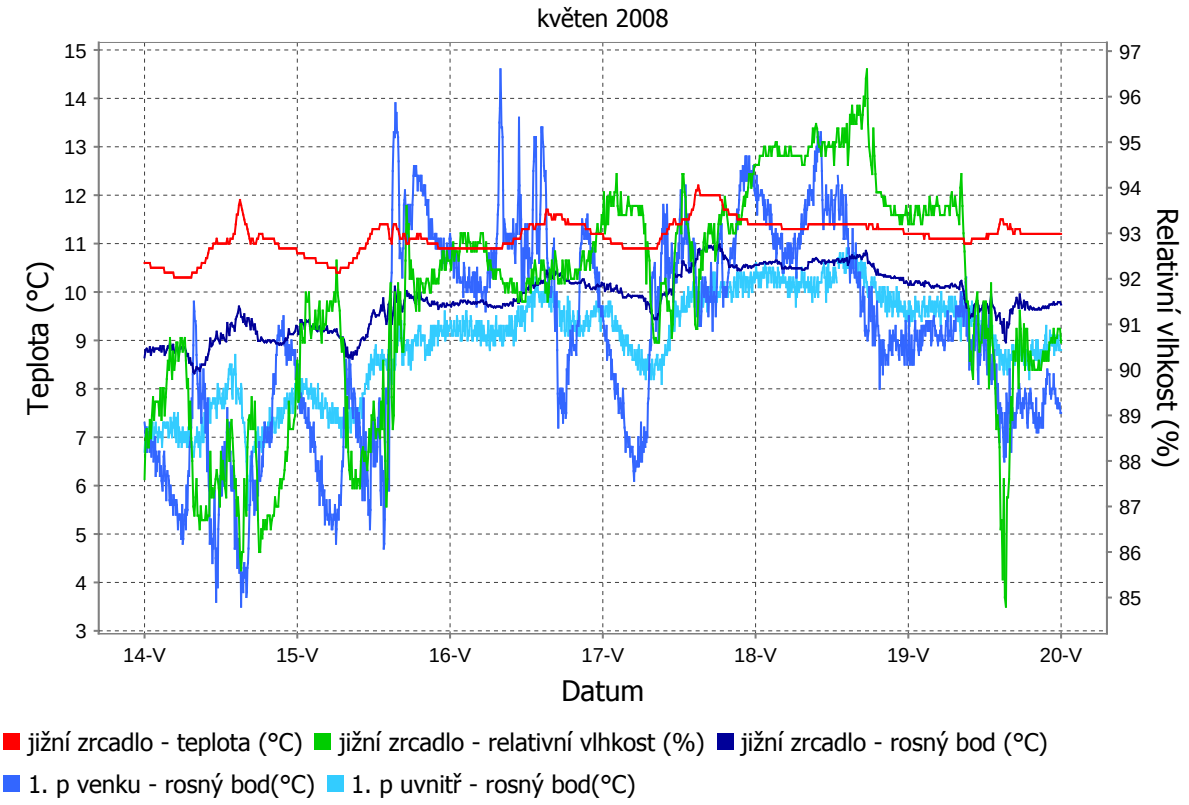
Obrázek 3: Shrnutí měření z jara 2008, pouze vybrané veličiny

Na obrázku 3 na straně 10 je graf, který shrnuje data naměřená na jaře roku 2008. V grafu jsou znázorněny jen ty veličiny, které jsou významné pro zjišťování chování vzduchu v budově a reakcí budovy na větrání. Ve spodní části grafu jsou navíc znázorněny dešťové srážky, které byly ve sledovaném období naměřeny meteostanicí firmy Josef Řehák – SPELEO, umístěnou na rajském dvoře. Měření dešťových srážek jsou v této práci zveřejněna se souhlasem firmy SPELEO. Tato firma se v konventu zabývá výzkumem a údržbou vodního základového systému.

Pro usnadnění vizualizace naměřených dat a syntézu pohledu na data z různých zdrojů jsem v průběhu této diplomové práce vyvinul softwarovou aplikaci, která tyto úkony výrazně urychluje a usnadňuje. Aplikaci samotné se budeme věnovat později.

Ve sledovaném období byla budova provětrávána pouze na základě porovnávání teploty vnějšího a vnitřního vzduchu bez ohledu na jeho vlhkost. Díky tomu můžeme na grafu pozorovat situace, kdy se při větrání nevědomky porušovala první podmínka, do budovy se dostával vlhký vzduch, který se v přízemí ochladil a voda z něj zkondenzovala na stěnách a ve schodišti u jižního (a zřejmě i severního zrcadla, kde ale již není schodiště).

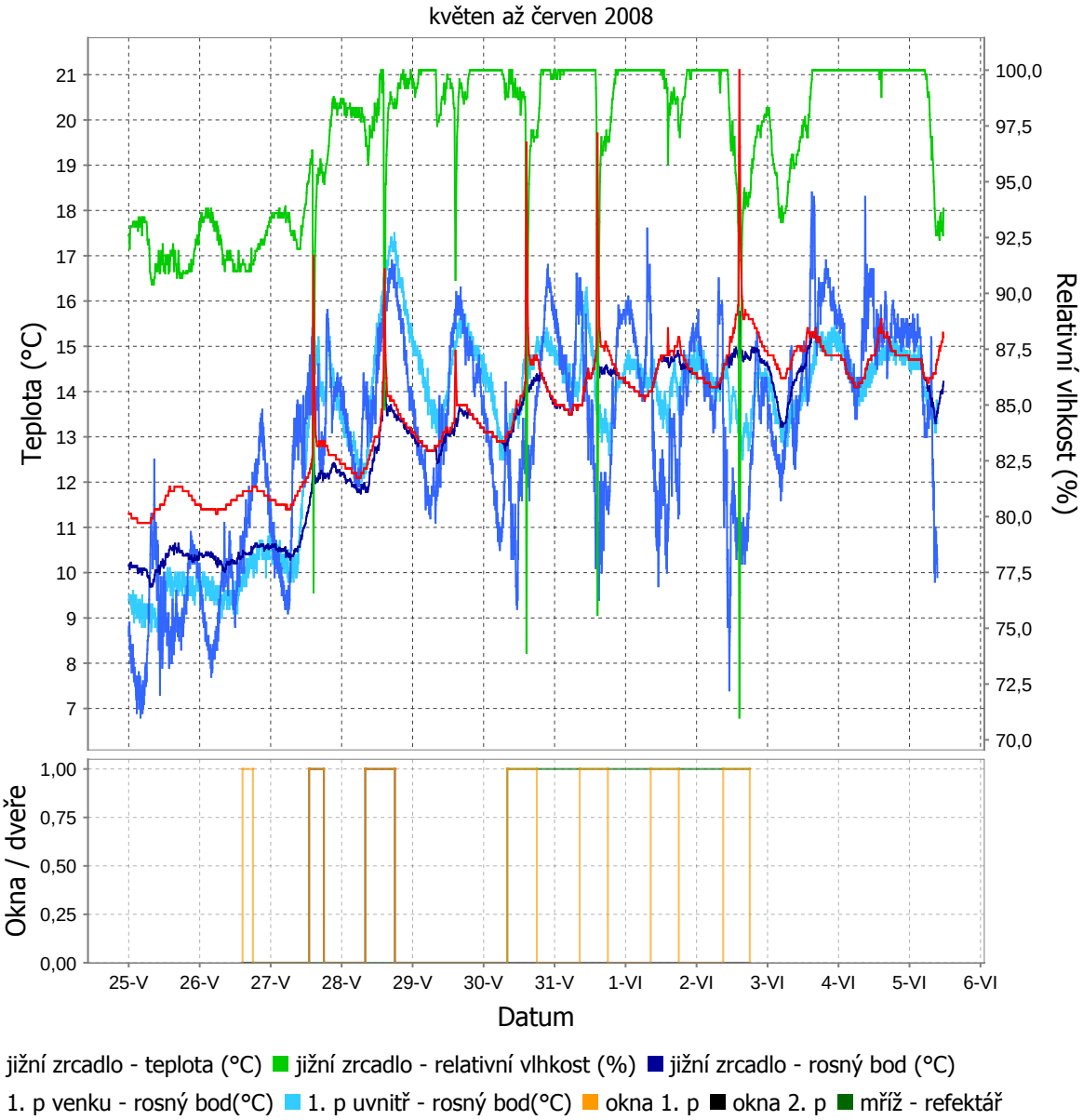
Zvýšená vlhkost vnějšího vzduchu byla například 29. dubna, 5. až 6. května a 15. až 18. května. V těchto případech však byla vysoká vzdušná vlhkost doprovázena dešťovými srážkami a v takových případech se podle zvyklosti nevětrá. Vlhký vzduch tak byl udržen mimo budovu a na grafu je vidět, že se rosný bod (který také vyjadřuje absolutní vlhkost vzduchu) uvnitř budovy nezvýšil. Ve sledovaném období se ale také nachází úsek od



Obrázek 4: Díky tomu, že okna zůstala zavřená, vlhký vzduch se do konventu nedostal.

27. května do 5. června, kdy zvýšený obsah vodní páry ve vzduchu nebyl doprovázen dešťovými srážkami. Naopak bylo slunečné a teplé počasí. Déšť přišel až 3. června odpoledne. Do té doby v konventu probíhalo téměř každý den větrání okny a navíc (od 30. května)

také 24 hodin denně mříží nade dveřmi zimní jídelny, která je v blízkosti jižního zrcadla. Vlhký vzduch se tak dostával do budovy nepřetržitě a po několik dní docházelo ke kondenzaci vodní páry v prostoru u jižního zrcadla. Tento stav je zachycen na obrázku 5. Měřicí systém, který byl použit v roce 2008 mohl sice podobné kritické situace zaznamenat, nemohl jim však účinně zabránit, protože měřená data nebyla personálu dostupná v reálném čase. Největší dostupnost měřených dat poskytovala měřicí soustava s počítačem *WinCon*, která měla vlastní display, na kterém byl zobrazován průběh měřených veličin, avšak toto zařízení bylo (z důvodů a omezení uvedených v [1] umístěno v místnosti cca 100 metrů vzdálené od kanceláře správy objektu, kde se nachází personál schopný okna zavírat a otevírat. Dostupnost dat v reálném čase je jednou z hlavních vlastností nového měřicího systému, jehož návrh a realizace je předmětem zájmu této diplomové práce.



Obrázek 5: Kvůli neregulovanému větrání se dostal vlhký vzduch do budovy

1.5 Nový měřicí systém

Nový měřicí systém by měl splňovat tyto požadavky:

1. Systém by měl být schopen v konventu pracovat každoročně od dubna do září bez údržby.

2. Systém by měl být schopen měřit teplotu a relativní vlhkost vzduchu na rajském dvoře a u jižního vodního zrcadla.

3. Naměřená data by měla být dostupná personálu v kanceláři správy objektu v reálném čase, případně se zpožděním do 10 minut. Přenos dat bude vyžadovat využití bezdrátových technologií.

4. Nový systém by měl být levnější než prototyp z roku 2008, jehož celková cena (včetně dataloggerů půjčených od kláštera) byla cca 70 tisíc Kč.

Jako velmi dobrý základ nového měřicího systému se ukázalo být zařízení vyvinuté ve druhé polovině roku 2008 ve spolupráci mezi Katedrou kybernetiky a Fakultou elektrotechnickou (viz [3]). Tímto zařízením je univerzální měřicí a řídicí karta RCB-02, nazývaná také *M-Board*

M-Board je vybaven procesorem typu ARM, analogovými a digitálními vstupy a výstupy a má velké možnosti konektivity (USB, RS232, ZigBee, Ethernet..) Předností tohoto zařízení jsou jeho malé rozměry a příznivá cena (kolem 12 tisíc Kč). karta je schopna pracovat samostatně, nebo jako rozhraní mezi PC a měřeným/řízeným procesem.

Na jaře roku 2009 bylo ke kartě M-Board uvolněno SDK, které umožňuje pro kartu psát programy v jazyce C. Základní rysy měřicího systému postaveného kolem M-Boardu by tedy byly následující:

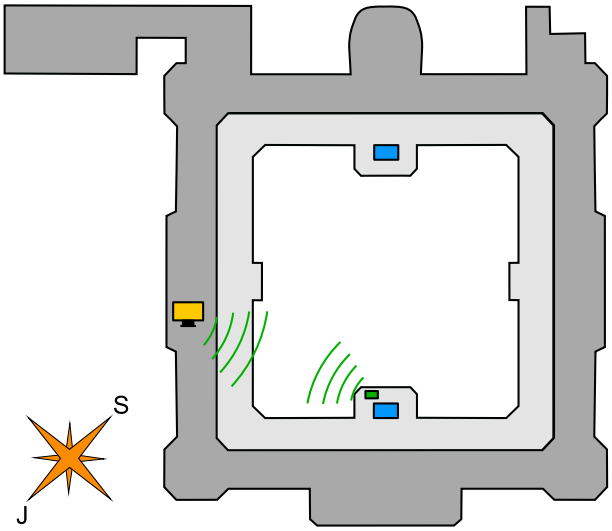
1. M-Board by byl umístěn u okna u jižního schodiště, kde je možné jednoduše zajistit napájení ze sítě.

2. Pro měření vzduchu by byla použita čidla z předchozí bakalářské práce, která by byla připojena k napěťovým vstupům M-Boardu.

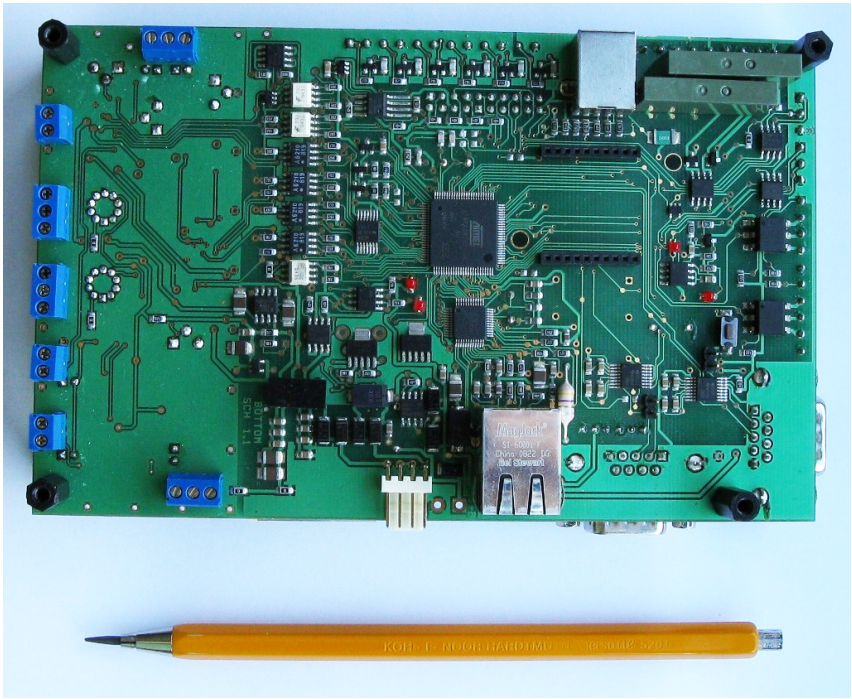
3. Na jednom ze dvou PC v kanceláři správy objektu by byl nainstalován program, který by z M-Boardu stahoval naměřená data a sloužil by jako poradní systém pro doporučování větrání. Komunikace mezi kartou a PC by byla zajištěna pomocí ZigBee, které by teoreticky mělo být schopné fungovat za daných podmínek na danou vzdálenost.

4. Měřicí systém by měl vyžadovat pouze minimální údržbu a měl by být maximálně robustní. Aplikace na stolním PC by měla fungovat samostatně a provádět stahování, vyhodnocování a archivaci dat i bez zásahu uživatele.

Náčrt zmíněné situace se nachází na obrázku 6 na straně 14. Vzdálenost mezi komunikujícími zařízeními je vzdušnou čarou cca 30 metrů.



Obrázek 6: Umístění hlavních prvků měřicího systému a znázornění bezdrátové komunikace mezi M-Boardem (zelený obdélník) u jižního zrcadla a PC (žlutý monitor) v kanceláři.



Obrázek 7: Univerzální měřicí a řídicí karta M-Board.

2 Příprava řešení

2.1 Vývojové nástroje

2.1.1 Programování M-Boardu

Ke psaní měřicí aplikace pro M-Board jsem využíval vývojové prostředí *Eclipse* [24] s pluginem *M-SDK*, vyvinutým na Katedře kybernetiky (KKY). Jedná se o sadu ovladačů pro hardware M-Boardu. M-Board se následně programuje v jazyce C. K závádění programu do M-Boardu a odlaďování jsem využíval *IAR Embedded Workbench for ARM* [23] nainstalovaný na počítači v laboratoři UL509. Své postupy týkající se práce na M-Boardu jsem konzultoval s Ing. Ondřejem Ježkem a Ing. Tomášem Perným z KKY. Podrobnější a aktuální informace o vývojových nástrojích pro vytváření aplikací pro M-Board můžete nalézt na adrese `http://www.rexcontrols.cz/devzone`. Plugin *M-SDK* jsem obdržel přímo od Ing. Ježka.

2.1.2 Tvorba aplikace pro PC

Výběr programovacího jazyka a vývojových prostředků se odvíjí od požadavků na tuto aplikaci:

1. Dostatečné schopnosti vizualizace měřených dat (jak v reálném čase, tak i dat z archivů).
2. Možnost exportu grafu do bitmapových i vektorových formátů (nejlépe PNG, PDF a SVG)
3. Schopnost aplikace komunikovat se sériovou linkou
4. Snadná instalace, nezabírající mnoho diskového prostoru, maximální robustnost a stabilita
5. Dlouhá životnost aplikace napříč verzemi operačních systémů
6. Preferována multiplatformita (schopnost fungovat na různých operačních systémech (kromě Windows zejména na Linuxu)

Zvolený programovací jazyk by měl umožnit rychlý a jednoduchý vývoj aplikace a všechny softwarové vývojové prostředky by měly být dostupné zdarma bez omezení i pro komerční využití (nejen například pro studenty).

Výběr programovacího jazyka a vývojových nástrojů jsem prováděl na přelomu roku 2008 a 2009 a do užšího výběru se dostaly *Java* od firmy *Sun* a jazyk *C#* od firmy *Microsoft*. Problémem varianty od Microsoftu však bylo, že i lehká a bezplatná verze vývojového prostředí - *Visual Studio Express*[1] vyžaduje kolem 1GB instalačního prostoru a i pro běh výsledné aplikace na uživatelském[2] PC je nutné nainstalovat *.NET Framework* v příslušné verzi a ve výsledku je velikost instalace na uživatelském PC větší, než v případě jazyka *Java*.

[1] `http://www.microsoft.com/express/Windows/`

[2] jakýkoliv počítač, na kterém má být aplikace finálně využívána, ne počítač na kterém probíhá vývoj aplikace

Multiplatformita programů napsaných v jazyce Java je již léty prověřená, zatímco zajišťování běhu aplikací napsaných pro platformu .NET například na již zmiňovaném Linuxu je nyní ještě v začátcích, nicméně rychle se rozvíjející díky projektu *Mono*[3].

Z důvodu zajištění dlouhodobé udržovatelnosti programu, menších systémových nároků a větších možností nasazení výsledné aplikace na různých operačních systémech jsem se rozhodl pro jazyk Java. Důvody, pro které jsem tento jazyk zvolil nepřestaly existovat ani ve chvíli, kdy na jaře roku 2009 byla firma *Sun* koupena firmou *Oracle*.

Pro Javu lze využít tři hlavní volně šiřitelná vývojová prostředí:

1. *NetBeans* (komunitou vyvíjené prostředí pod záštitou firmy Sun, dnes Oracle), [22]

   NetBeans je přehledné a jednoduše použitelné vývojové prostředí podporující řadu jazyků (Java (primárně), C, C++, xml, html, css a další). Disponuje vysoce kvalitním integrovaným editorem grafického uživatelského rozhraní (GUI[4]) pro grafickou knihovnu *Swing*. Do verze 6.7.5 byl pro *NetBeans* poskytován jako plugin UML editor umožňující reverse engineering a generování kódu pro jazyk *Java*. Tento plugin však již není komunitou dále udržován a od verze *NetBeans 6.8* (vyšla 10. prosince 2009) není možné jej do prostředí nainstalovat z důvodu nekompatibility. Instalace *NetBeans* pro vývoj v jazyce *Java* potřebuje cca 250 MB diskového prostoru. *NetBeans* bez problémů pracuje i na malých obrazovkách s malým rozlišením (například u netbooků obvyklých 1024 x 600 bodů). *NetBeans* navíc disponují skvělým profilerem, který umožňuje optimalizovat rychlost vyvíjené aplikace a její systémové nároky.

2. *Eclipse* (komunitou vyvíjené prostředí pod záštitou IBM), [24]

   Eclipse, podobně jako *NetBeans* je vývojové prostředí podporující více jazyků. Existuje pro něj obrovské množství pluginů, mezi kterými jsou například UML a GUI editory. *Eclipse* disponuje větším množstvím funkcí a větší rozšiřitelností než *NetBeans*. Množství funkcí v tomto případě bývá na úkor přehlednosti. Požadavky na diskový prostor jsou obdobné jako u *NetBeans*. Problém nastává tehdy, pokud chceme GUI editor, který je spolehlivý a zdarma bez omezení. V současné době takový není pro *Eclipse* k dispozici. Velké množství dialogových oken v *Eclipse* potřebuje obrazovku vyšší než 600 bodů a nelze je zmenšit. tato vlastnost činí práci s *Eclipse* na netbooku nepohodlnou či nemožnou. Pro *Eclipse* existuje volně šiřitelný profiler, avšak nedosahuje možností profileru v *NetBeans*.

3. *JDeveloper* od firmy Oracle, [25]

   JDeveloper je primárně zaměřen na jazyk Java a vývoj enterprise aplikací určených pro aplikační servery. Podle příslušné edice disponuje i UML a GUI editorem. Na malých obrazovkách nemá problémy. Plná edice *JDeveloperu* s GUI a UML editorem však vyžaduje cca 1GB diskového prostoru a i nároky na operační paměť jsou vyšší než u *NetBeans* a *Eclipse*.

Po odzkoušení vývojových prostředí jsem se rozhodl pro *NetBeans*, které poskytuje vysokou stabilitu a uživatelský komfort, má integrované všechny potřebné nástroje a je možné jej

[3] `http://www.mono-project.com`
[4] Graphical User Interface

bez problémů provozovat i na netbooku a tak provádět testování a úpravy vyvíjeného software jak v laboratoři, tak přímo v terénu. GUI editor v *NetBeans* podporuje knihovny AWT a Swing, ale ne SWT, což však v tomto případě není problém. UML diagramy ilustrující strukturu vytvořeného software byly vytvořeny v editoru *UMLet* [21].

2.2 Další hardwarové prvky systému

2.2.1 Čidla teploty a relativní vlhkosti vzduchu

Pro detekci teploty a relativní vlhkosti vzduchu byla využita obě čidla *Humirel* HTM1735 používaná již v předchozí bakalářské práci. Napěťový výstup z čidel byl měřen napěťovými vstupy univerzální karty M-Board.

2.2.2 Moduly bezdrátové komunikace

Bezdrátová komunikace mezi M-Board a PC s archivační a vyhodnocovací aplikací byla zajištěna pomocí dvou modulů bezdrátové komunikace *XBee* XB24ASI, které umožňují montáž externí antény s konektorem RP-SMA.

K propojení PC se ZigBee modulem byl využit interface *XBeeU* zakoupený od firmy Snail Instruments [34] , který slouží jako převodník z USB na sériovou linku a napájení ZigBee modulu *XBee*. Toto zařízení bylo vybráno na základě konzultace s Ing. Ježkem. Pro správnou práci interface s PC je nutné nainstalovat ovladač obvodu FT232R ze stránek jeho výrobce `http://www.ftdichip.com`. Dokumentaci k XBee modulu a konfigurační aplikaci X-CTU lze získat na stránkách výrobce tohoto modulu `http://www.digi.com`.

K XBee modulu na straně Měřicí karty byla připojena anténa TP-LINK TL-ANT2405C. Jedná se o anténu se ziskem 5dB a kabelem o délce cca 1m. XBee modul na straně PC je vybaven anténou z domácího WiFi AP od firmy D-Link.

2.2.3 Napájení systému

Pro napájení čidel a *M-Boardu* byl využit stabilizovaný síťový zdroj s výstupním napětím 5V a maximálním dodávaným proudem $1,2$A. Měřením jsem zjistil, že *M-Board* i s čidly má průměrný odběr kolem $0,2$A, takže výkon zmíněného zdroje je více než dostačující. Měření ovšem také ukázalo, že zdroj při zmíněné zátěži $0,2$ A má výstupní napětí $5,36$V. Tato hodnota již může negativně ovlivnit, či poškodit použitá čidla. Bylo proto nutné umístit do obvodu do série s napájenými zařízeními Schottkyho diodu. Napětí na zdroji (měřené za diodou) při zátěži je potom $4,92$V. Toto napětí je již vhodné pro napájení čidel i měřicí karty (přes USB konektor).

Detaily zapojení a umístění čidel, antén a dalších prvků systému budou zmíněny později.

2.3 Použité softwarové knihovny třetích stran

Pro jazyk Java existuje mnoho bez omezení volně šiřitelných softwarových knihoven, jejichž funkce je možné používat v dalších Java aplikacích. Pokud pro řešení nějakého úkolu potřebná knihovna existuje a její funkčnost je vyhovující, není zpravidla nutné se zabývat vývojem stejné funkce znovu.[5]

[5]Pokud se ovšem celý projekt netýká vývoje knihovny s požadovanou funkčností.

Pro účely svého projektu jsem využil knihovny třetích stran pro zajištění funkcí v níže uvedených oblastech. Použití již hotových knihoven nijak nesnižuje můj vlastní přínos. Mnou vytvořený vlastní kód čítá přes 200 tříd a tvoří další samostatnou knihovnu, kterou je možné využívat v dalších aplikacích podobného typu, jako je tato.

2.3.1 Vizualizace dat

Pro zobrazování grafů jsem zvolil knihovnu *JFreeChart* [18], která poskytuje velké množství funkcí a možností vizuální reprezentace dat. Mimo jiné je možné zobrazovat jak časové řady (Vzorky mají konkrétní časovou značku) tak i řady číselné (Vzorky jsou číslovány a může se jednat o pořadové číslo vzorku, nebo čas, který uplynul od začátku měření). Knihovna samotná pak umožňuje export grafu do bitmapových formátů PNG nebo JPG. Pro export do vektorových formátů je možné použít další knihovny zmíněné níže. K použití *JFreeChart* je možné na internetu najít celou řadu příkladů (např. [14]) nahrazujících oficiální dokumentaci, která na rozdíl od knihovny samotné není dostupná zdarma.

2.3.2 Export grafů do vektorových formátů

Bitmapové formáty jsou vhodné k prezentaci dat například na internetových stránkách, pokud však potřebujeme prezentovat grafy v tištěných dokumentech, je lepší využít formáty vektorové.

1. **Export do SVG** (Scalable Vector Graphics)
   Formát SVG je založen na popisu objektů a čar uloženém v textovém XML souboru. Existují pro něj volně šiřitelné editory (například *Inkscape*) a jeho zobrazování podporují například i současné verze webových prohlížečů *Internet Explorer*, *Opera* a *Firefox*.

   Pro export grafů z *JFreeChart* do SVG využívám knihovnu *Batik* [19]. *Batik* je šířen pod licencí *Apache* verze 2.0 [27].

   Samotný proces exportu grafů z *JFreeChart* pomocí *Batiku* do SVG s sebou nese jistou nevýhodu, která spočívá ve velkých nárocích na operační paměť a dobu exportu. Experimentálně jsem zjistil, že pokud je v grafu kolem 200 tisíc vzorků, export může trvat až několik minut, zabrat několik stovek MB RAM a výsledný soubor má (podle zvoleného rozlišení) až několik desítek MB. Případné další úpravy takto velkých souborů (například v *Inkscape*) jsou velmi zdlouhavé.

2. **Export do PDF** (Portable Document Format)
   PDF je v dnešní době velmi rozšířený formát pro dokumenty všeho druhu. Pokud chceme graf exportovat do vektorového formátu a nepočítáme s dalšími úpravami obrázku, je výhodné provést export do PDF.

   K tomuto účelu využívám knihovnu *iText* verze 2.1.5. Do verze 2.1.7 byl *iText* distribuován pod licencí LGPL [28]. Od prosince 2009 distribuce probíhá pod licencí AGPL [29], která neumožňuje využívání knihovny v projektech s neveřejným (uzavřeným) zdrojovým kódem. Pro takové projekty nicméně existuje komerční licence.

2.3.3 Práce s XML soubory

Soubory ve formátu XML využívám k ukládání konfigurací aplikací. Pro čtení a zápis těchto souborů využívám knihovnu *Jdom* [15].

2.3.4 Komunikace přes sériovou linku

Klíčovou schopností měřicí aplikace je komunikace s měřicím zařízením přes sériovou linku (přes RS232 nebo USB). Java na rozdíl od výše zmiňované platformy Microsoft .NET nedisponuje vestavěnými třídami pro komunikaci přes sériovou linku. Tento nedostatek je možné řešit využitím knihovny *RXTX* [26].

2.3.5 Komunikace pomocí XML-RPC

Komunikaci mezi klientskou viditelnou částí aplikace a částí pracující s daty a měřicím zařízením jsem se rozhodl řešit pomocí protokolu XML-RPC. Dva programy pak mezi sebou budou komunikovat jako XML-RPC server a klient. Pro realizaci komunikace pomocí XML-RPC jsem zvolil implementaci *Apache XML-RPC* [16].

2.3.6 Řešení soustav rovnic pomocí metody nejmenších čtverců

Pro řešení soustav rovnic využívám třídy (a metody) z knihovny *Apache Commons Mathematics Library* [17].

2.3.7 Ikony

Grafické rozhraní aplikace jsem vybavil volně šiřitelnými ikonami *Crystal Icons* [30] a některými ikonami vlastními.

3 Vývoj Software pro PC

Při měření bude vznikat velké množství dat, které bude nutné spolehlivým a uživatelsky přívětivým způsobem zpracovávat, archivovat a vizualizovat.

V projektu nalezne uplatnění jak aplikace, která bude zpracovávat měření průběžně, tak i aplikace pro offline vyhodnocování a vizualizaci archivovaných dat. Poslední zmíněná aplikace najde využití i při publikaci získaných výsledků.

Z důvodů zvýšené stability by měla být měřicí aplikace na PC rozdělena na dva samostatné programy, které mezi sebou budou komunikovat. První z nich, který jehož činnost bude běžnému uživateli skryta, bude obstarávat komunikaci s měřicím zařízením, vyhodnocování a archivaci. Druhá část potom bude pracovat v grafickém režimu a bude sloužit ke komunikaci uživatele se zbytkem měřicího systému. Přes tuto aplikaci bude možné ovládat měření, a zobrazovat získávaná data. Třídy, které jsem vytvořil pro zajištění obecných funkcí měřicí aplikace, jsou umístěny v knihovně *libdev*, jejíž součásti jsou popsány zde:

3.1 Reprezentace dat

Snažil jsem se, aby knihovna uměla pracovat jak s časovými, tak i číselnými řadami. Pro měření bude primárně využívána reprezentace dat jako časových řad, ale možnost práce s číselnými řadami se využije například při kalibraci čidel, kdy naměřené hodnoty nemusí být nutně opatřeny časovou značkou, ale pořadovým číslem měření. Stejně tak bude možné použít aplikaci ke zpracování a zobrazování dat z jiných zdrojů, například ze řídicího a měřicího systému REX vyvíjeného na Katedře kybernetiky ZČU.

S ohledem na univerzálnost reprezentace je v knihovně v hojné míře využíván polymorfismus. Základní jednotkou reprezentující data je třída `GeneralFloatingSample`, která představuje obecný plovoucí vzorek. Plovoucí proto, že je schopen existovat a být zpracováván samostatně a nese všechny informace, které jej zařazují do příslušné datové řady a určují hodnotu představované veličiny. Hodnota veličiny je desetinné číslo ve formátu `double`. Vlastnosti obecného plovoucího vzorku jsou dále využívány a konkretizovány dalšími dvěma třídami:

1. `TimeStampedFloatingSample` – pro reprezentaci dat s časovou značkou

   Časová značka ve vzorku je ve formě odkazu na příslušnou instanci třídy `Instant`, která uchovává informaci o časovém okamžiku v rozsahu tisíců (případně i více) let s přesností na tisíciny vteřiny. Na jednu instanci třídy `Instant` může odkazovat více vzorků současně.

2. `NumberedFloatingSample` – pro reprezentaci dat s pořadovým číslem

   Pořadové číslo je ve formátu `double` a může se tedy jednat i o číslo desetinné. UML diagram znázorňující zmíněné třídy se nachází na obrázku 8 na straně 21.

3.2 Reprezentace času

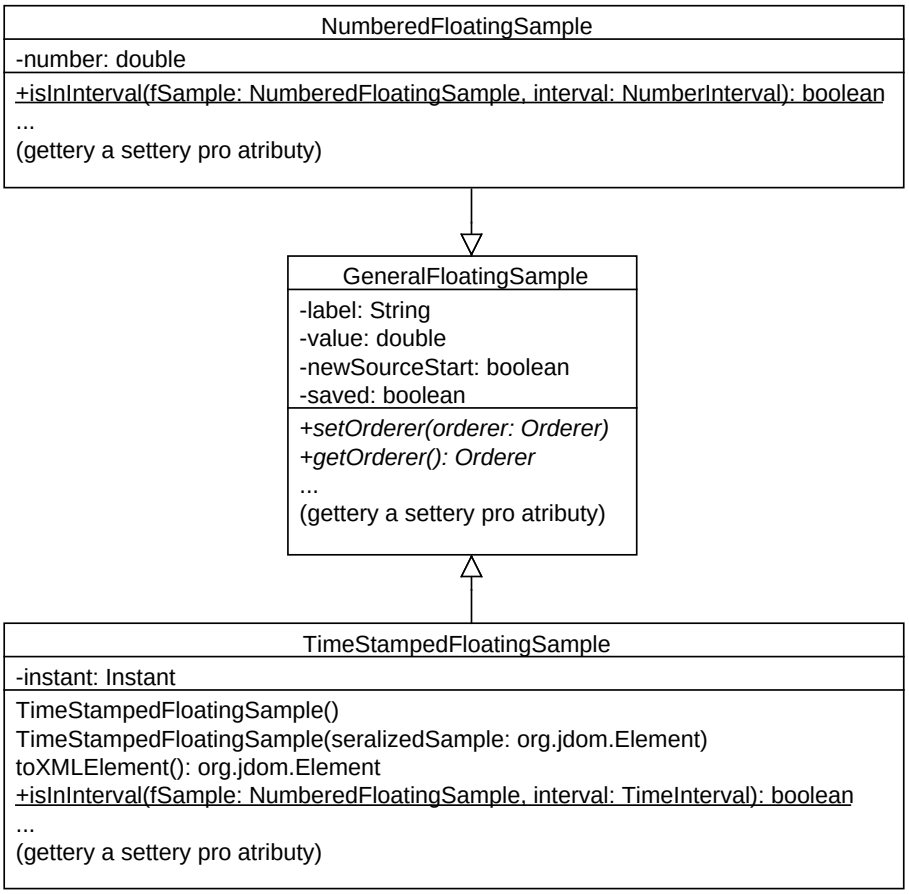
Časová značka je reprezentována instancí třídy `Instant`. `Instant` má pak v sobě odkazy na objekty tříd `Date` (ne `java.util.Date`) a `Time`. UML diagram tříd pro reprezentaci času je na obrázku 9 na straně 22.

Objekt `Date` uchovává informace o datu v položkách `year`, `month` a `day` (rok, měsíc a den), které jsou ve formátu celého čísla `int`.

Objekt `Time` pak uchovává informace o čase v položkách `hours` (`int`), `minutes` (`int`), `seconds` (`double`), a `milliseconds` (`int`). Počet sekund v položce `seconds` odpovídá počtu milisekund v položce `milliseconds`. Při nastavení hodnoty jedné z nich se přepočet a nastavení druhé provede automaticky.

3.2.1 Operace s časem

Pro další práci s daty bylo nutné implementovat metody pro přičítání a odečítání času a rozhodování, zda daný časový okamžik patří do nějakého časového intervalu, nebo nikoli. Jazyk Java poskytuje pro práci s časem vestavěnou třídu `java.util.Calendar`. Tato třída je v mé knihovně využívána při dekódování časových značek z textových řetězců a výpisu časových značek do textových řetězců.



Obrázek 8: UML diagram tříd pro reprezentaci datových vzorků. V diagramu jsou uvedeny jen některé významné atributy a metody.

Využití metod třídy `Calendar` pro práci s časovými značkami u vzorků však není zcela vhodné z následujícího důvodu:

`Calendar` pracuje s reálným kalendářem. To znamená, že zohledňuje přestupné roky a letní/zimní čas. Pokud například zjišťujeme rozdíl časů mezi daty, z nichž pro jedno platí zimní a pro druhé letní čas[6], je k rozdílu automaticky přičtena, nebo naopak odečtena 1 hodina.

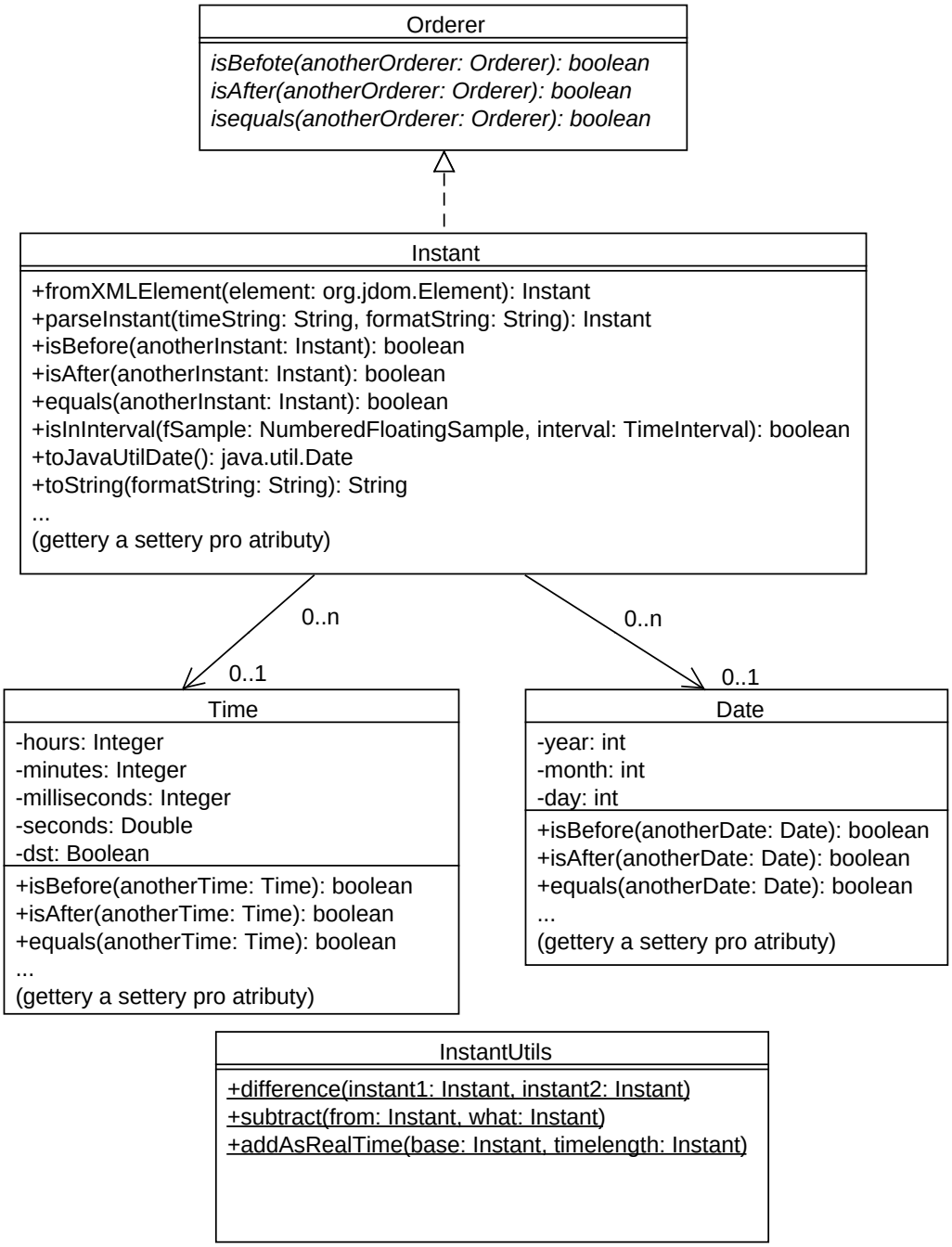
Při provádění dlouhodobého měření je však zvyklostí používat po celý rok stejný čas (nejlépe zimní). Tento přístup je využíván například na registračních stanicích firmy *Fiedler-Mágr*[7].

Metody, které provádějí časové operace požadovaným způsobem jsem implementoval ve třídách `InstantUtils`, `TimeInterval` a v již zmíněné třídě `Instant` v balíku `com.msiroky.chronology.time`.

Třída `InstantUtils` obsahuje metody, které počítají rozdíl a součet časů bez ohledu na přechod mezi zimním a letním časem, přičemž vstupními parametry jsou vždy časy základní (nějaké skutečné datum a čas) a potom časy přičítané nebo odečítané. V těchto metodách je možné pracovat s přičítáním nebo odečítáním intervalů do délky 28 dní, což

[6]Podle pravidel platných pro danou zemi/lokalizaci; Tato pravidla jsou vestavěna do Javy.

[7]`http://www.fiedler-magr.cz`



Obrázek 9: UML diagram tříd pro reprezentaci časových údajů a provádění operací s nimi. V diagramu jsou uvedeny jen některé významné atributy a metody.

je pro daný účel postačující. Delší intervaly by pak znamenaly délku 1 měsíc či delší. Vzhledem k tomu, že měsíce jsou různě dlouhé, při přičítání například 3 měsíců by nebylo explicitně zřejmé, které kalendářní měsíce a tedy kolik dní se vlastně bude přičítat či odečítat. Výpočetní operace by pak neměla pro uživatele předvídatelný výsledek. Navíc pro výpočty takového typu je možné použít třídu `Calendar`. Pokud by však bylo nutné rozšířit rozsah zpracovávaných časových intervalů, příslušná úprava stávajícího zdrojového kódu metod ve třídě `InstantUtils` by trvala cca 15 minut. Při výpočetních operacích se zohledňuje přenos výsledků do vyšších řádů.

Třída `Instant` obsahuje metody, které časový interval do délky 28 dní převedou do jeho délky v milisekundách a obráceně (`toMilliseconds()`, `millisecondsToInstant()`). Dále jsou zde metody pro porovnávání časových okamžiků:

1. `isBefore(anotherInstant: Instant): boolean` – Vrací `true`, pokud je daný časový okamžik **před** jiným časovým okamžikem `anotherInstant`. Nejprve se provede porovnání hodnot v atributu `Date`. Pokud je datum u daného časového okamžiku před datem u `anotherInstant`, metoda vrací `true` ihned. Pokud jsou hodnoty data stejné, pokračuje se porovnáním atributů `Time`. Pokud je čas daného okamžiku před časem v `anotherInstant`, metoda vrací `true`. Porovnávání dvou instancí tříd `Date` a `Time` se provádí pomocí metod v těchto třídách.

2. `isAfter(anotherInstant: Instant): boolean` – Vrací `true`, pokud je daný časový okamžik **za** jiným časovým okamžikem `anotherInstant`. Postup porovnávání je obdobný jako v předchozí metodě, jen s obráceným kritériem.

3. `equals(anotherInstant: Instant): boolean` – Vrací true, pokud jsou časové okamžiky shodné. Zde musí být současná shoda v atributech `Date` i `Time`.

Pro ilustraci je zde uveden zdrojový kód metody `isBefore(anotherDate: Date): boolean` ve třídě `Date` a metody `isBefore(anotherTime: Time): boolean` ve třídě Time:

```
public boolean isBefore(Date anotherDate) {
    if (this.year < anotherDate.getYear()) {
        return true;
    } else if ((this.year == anotherDate.getYear())
    && (this.month < anotherDate.getMonth())) {
        return true;
    } else if ((this.year == anotherDate.getYear())
    && (this.month == anotherDate.getMonth())
    && (this.day < anotherDate.getDay())) {
        return true;
    }
    return false;
}

public boolean isBefore(Time anotherTime) {
    if (this.hours < anotherTime.getHours()) {
        return true;
    } else if ((this.hours == anotherTime.getHours())
    && (this.minutes < anotherTime.getMinutes())) {
```

```
        return true;
    } else if ((this.hours == anotherTime.getHours())
    && (this.minutes == anotherTime.getMinutes())
    && (this.milliseconds < anotherTime.getMilliseconds())) {
        return true;
    }
    return false;
}
```

3.2.2 Intervaly

Základem práce s intervaly (časovými i číselnými) je třída `GeneralInterval`. Od ní jsou odděděny třídy `NumberInterval` a `TimeInterval`. Vztahy mezi těmito třídami jsou znázorněny na obrázku 10 na straně 25. Zde se budu blíže věnovat intervalu časovému.

Časový interval má atributy *začátek* (`start`) a *konec* (`end`), což jsou odkazy na objekty třídy `Instant`. Interval může být zprava i zleva jak uzavřený, tak otevřený. Metody pro porovnávání intervalů pracují následovně:

1. `startsAfter(interval: TimeInterval): boolean` – Vrací `true`, pokud daný interval začíná až po konci jiného intervalu, který je parametrem této metody. V současné implementaci této metody se uvažuje, že jsou oba intervaly uzavřené.

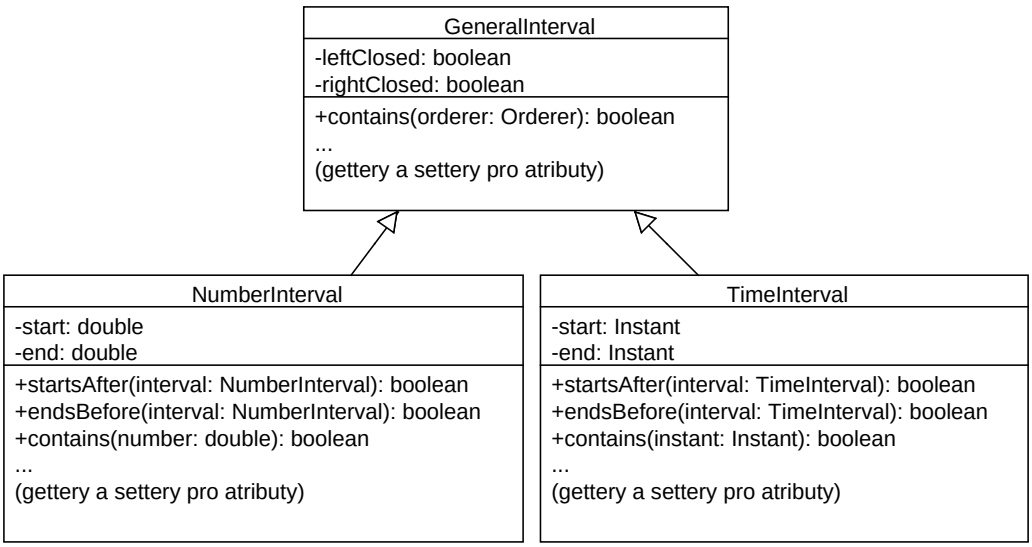
2. `endsBefore(interval: TimeInterval): boolean` – Vrací `true`, pokud daný interval končí před začátkem jiného intervalu. Předpoklady jsou stejné jako v předchozím případě.

3. `contains(instant: Instant): boolean` – Vrací `true`, pokud je časový okamžik `instant` v daném intervalu obsažen. Zde se již bere ohled na uzavřenost, či otevřenost intervalu.

   Vnitřně tato metoda volá metodu `isInInterval(fSample: NumberedFloatingSample, interval: TimeInterval): boolean` v objektu `instant`. Zjištění, zda se `instant` v intervalu nachází lze zjistit i přímo voláním této metody.

Zdrojový kód metody `isInInterval()` ve třídě `Instant`:

```
public boolean isInInterval(TimeInterval interval) {
    if (interval == null) { return true; }
    boolean isAterStart = false;
    boolean isBeforeEnd = false;
    boolean equalsStart = false;
    boolean equalsEnd = false;
    if (interval.isLeftClosed())
     { equalsStart = equals(interval.getStart()); }
    if (interval.isRightClosed()) { equalsEnd = equals(interval.getEnd()); }
    if (equalsStart || isAfter(interval.getStart())) { isAterStart = true; }
    if (equalsEnd || isBefore(interval.getEnd())) { isBeforeEnd = true; }
    return (isAterStart && isBeforeEnd);
}
```



Obrázek 10: UML diagram tříd pro reprezentaci intervalů. V diagramu jsou uvedeny jen některé významné atributy a metody.

3.3 Uchovávání dat v operační paměti

K uchovávání a organizaci datových vzorků v operační paměti slouží třídy `DataPool` a `DataDrawer`. `DataDrawer` je možné si představit jako šuplík kartotéky, ve kterém jsou uložena data vztahující se k jedné měřené veličině / kanálu a jsou chronologicky seřazena. `DataPool` (nádrž na data) je potom jakousi skříní plnou takových šuplíků. V knihovně není žádné omezení na počet uložených dat ani datových kanálů. Ve zdrojových kódech je pro kanál využíván pojem *datové vlákno* (*data thread*). `DataDrawer` je vytvořen k uchovávání obecných datových vzorků, takže je možné jej použít jak s číselnými, tak časovými datovými řadami. Ilustraci vztahu mezi třídami `DataDrawer` a `DataPool` je možné vidět na obrázku 11 na straně 26.

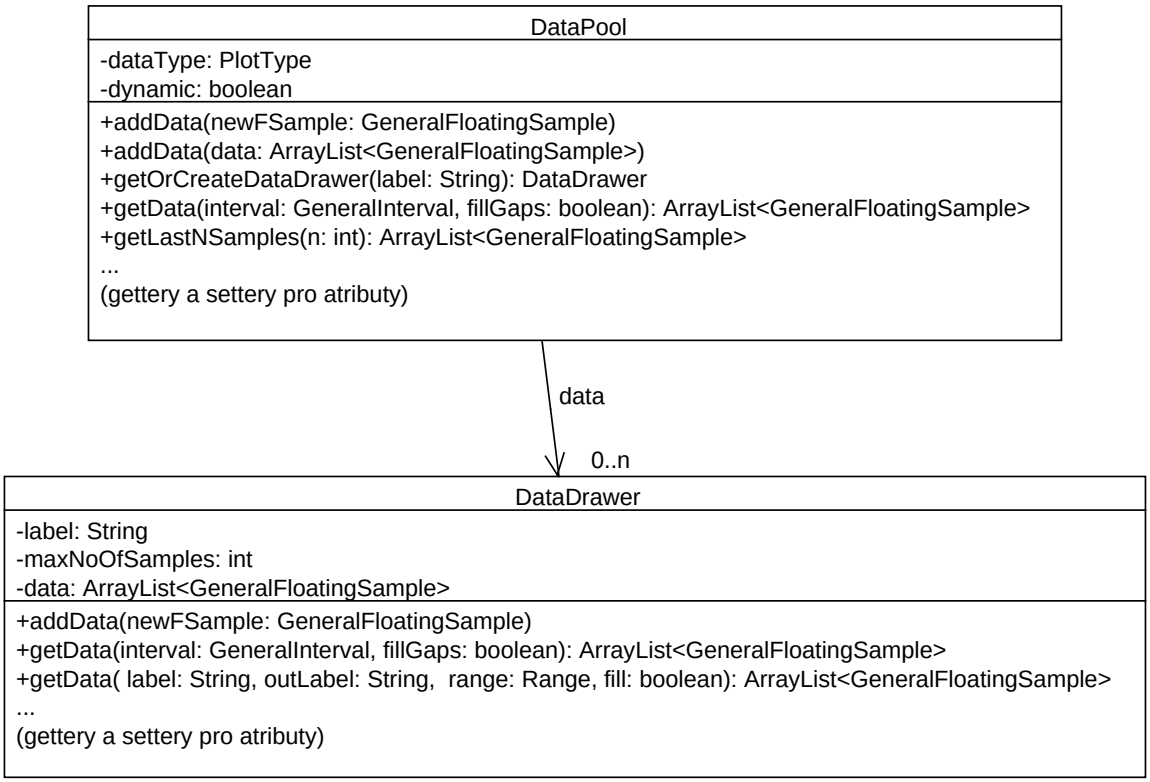
`DataDrawer` je schopný pracovat jak v dynamickém, tak statickém režimu. Statickým režimem je myšleno, že všechna data z příslušného vlákna jsou do paměti načtena najednou v jednom kroku, nebo vzniknou (také najednou) jako výsledek nějaké výpočetní operace. Dynamický režim potom znamená, že data vznikají a jsou ukládána postupně. Toto je například situace při měření.

Pro uchovávání odkazů na objekty třídy `DataDrawer` ve třídě `DataPool` a na objekty třídy `GeneralFloatingSample` ve třídě `DataPool` je využita knihovní třída *Javy* `ArrayList`.

Vzorek je možné vkládat buď pomocí metod třídy `DataPool`, nebo přímo pomocí metod třídy `DataDrawer`, pokud si předtím necháme podle názvu datového vlákna vrátit objektem „Pool“ odkaz na příslušný „Drawer“. Pokud data vkládáme přes *Pool*, je příslušná *Drawer* vyhledán automaticky a pokud pro dané datové vlákno dosud neexistuje, je taktéž automaticky vytvořen. Při vkládání do *Draweru* jsou pak vzorky chronologicky řazeny následujícím způsobem:

1. Pokud je *Drawer* prázdný, vloží se vzorek ihned bez dalších operací.

2. Pokud již v *Draweru* nějaké vzorky jsou, předpokládá se nejprve, že vkládaný vzorek



Obrázek 11: UML diagram tříd pro uchovávání a organizaci dat v operační paměti. V diagramu jsou uvedeny jen některé významné atributy a metody.

je novější, než ostatní. Dojde tedy k porovnání pořadí posledního vzorku v seznamu a vzorku vkládaného podle jejich indikátorů pořadí (obecně instance třídy `Orderer`). Pokud se předpoklad potvrdí, je vzorek vložen na konec seznamu.

3. Pokud nenastane ani jedna výše zmíněná varianta, je pomocí algoritmu binárního vyhledávání implementovaného v knihovnách *Javy* nalezeno místo v chronologickém seznamu, do kterého daný vzorek patří.

   Aby měl algoritmus binárního vyhledávání informaci o pořadí dvojic vzorků, které porovnává, má k dispozici `GeneralSampleOrderComparator`, který implementuje rozhraní `Comparator` a vnitřně pracuje s metodami pro porovnávání, zmíněnými v oddíle 3.2.1. Místo, do kterého vkládaný vzorek patří, může být prázdné (a pak je do něj rovnou vložen) nebo může již být obsazené jinou hodnotou, která patří ke stejnému indikátoru pořadí. V takovém případě je stará hodnota vzorku přepsána hodnotou novou.

   Pokud pracujeme v dynamickém režimu, je možné nastavit, kolik vzorků z každého vlákna má být uchováváno v paměti. Pokud je v *Draweru* vzorků více, stará data se z paměti automaticky odstraňují. Limit je možné nastavit pro každý *drawer* zvlášť, nebo pro všechny najednou.

3.4 Zpracování dat

jak již bylo uvedeno výše, knihovna `libdev` umožňuje data také zpracovávat. Zpracování je možné jak v dynamickém, tak statickém režimu. V knihovně jsou v současné době v balíku `com.msiroky.dataprocessing.operations.implemented_operations` implementované operace uvedené níže. Rozsah působnosti operací lze upravit specifikací intervalu, na kterém pracují.

1. Zlomek (`Fraction`) – jedná se o výpočet podílů dvou polynomů, kdy každý z obou polynomů může být ve tvaru

$$\left(\sum_{i=1}^{m} a_i \cdot x_i^{n_i}\right) + a_0 \tag{1}$$

   kde x_i může být hodnota vzorku z jakéhokoliv datového vlákna a n_i a_i, a_0 mohou být jakákoliv desetinná čísla. Vstupní vzorky musejí mít stejný indikátor pořadí (čas nebo pořadové číslo). Na stejný indikátor potom ukazuje i vzorek, který je výsledkem operace.

2. Výpočet rosného bodu (`Dewpoint`) – Rosný bod je počítán z hodnoty vzorků v datových vláknech představujících relativní vlhkost v procentech a teplotu ve °C. Indikátory pořadí se shodují stejně, jako u předchozího případu.

3. Suma (`Sum`) – Vstupem sumy jsou vzorky pouze jednoho datového vlákna. Výstupem je pak další datové vlákno. Podle zvoleného rozsahu může suma načítat hodnoty vzorků neustále a pro každé přičtení vytvořit vzorek nesoucí výsledek, nebo sečíst hodnotu vždy několika vzorků a pro tento součet vytvořit jeden výstupní vzorek s indikátorem pořadí posledního přičítaného vzorku. Poté se hodnota sumy vynuluje a výpočet pokračuje sčítáním hodnot v dalším bloku vzorků.

4. Klouzavý průměr (`MovingAverage`) je možné využít k základnímu vyhlazování dat. Je možné nastavit několik parametrů výpočtu:

   - Šířku okénka (počet vzorků), ze kterého se průměr počítá
   - Krok (počet vzorků), o jaký se okénko mezi výpočty posouvá
   - Výstupní pozici, která udává, jaký indikátor pořadí bude přiřazen ke vzorku nesoucímu hodnotu výpočtu z daného okénka.

5. Vizuální spojování (`VisualConnection`) slouží k odstraňování vzorků s hodnotami `NaN` (Not a Number). Pokud je totiž datové vlákno, ve kterém jsou vzorky s touto hodnotou, vizualizováno pomocí `JFreeChart`, je na místě, kde je hodnota `NaN`, čára grafu rozpojena. Tento jev je někdy žádoucí, pokud chceme zdůraznit, že v daném místě data nejsou, i když by tam být měla. Pokud ale chceme tento jev odstranit, můžeme využít operaci `VisualConnection`, která vytvoří nové datové vlákno a do něj překopíruje pouze vzorky, které mají ne-`NaN` hodnotu.

6. Porovnání (`Comparison`) porovnává hodnotu referenčního a porovnávaného datového vlákna v identických časových okamžicích (nebo pořadových číslech). Výsledek porovnání, který je buď 1 nebo 0, je ukládán do jednoho výstupního vlákna. Referenční

hodnotu může představovat i pevně zadané desetinné číslo. Kolem referenční hodnoty je možné nastavit pásmo o zvolené šířce, kdy pásmo přesahuje referenční hodnotu na obě strany právě o tuto šířku. Pásmo lze označit za zakázané, nebo lze naopak zakázat fiktivní plochu mimo pásmo. Dále je možno nastavit, zda má být hlídána horní či spodní hranice pásma, či oboje.

- Pásmo je zakázané: Pokud je hlídána pouze spodní hranice pásma, je výstupní hodnota nastavena na 1, pokud je porovnávaná hodnota **větší** než spodní hranice pásma. Pokud je hlídána pouze horní hranice, pak je výstupní hodnota rovna 1, pokud je porovnávaná hodnota menší, než horní hranice pásma. Pokud jsou hlídány obě hranice, je na výstupu operace jednička, pokud je porovnávaná hodnota **uvnitř** pásma.
- Pásmo je povolené: Pokud je hlídána pouze spodní hranice pásma, je výstupní hodnota nastavena na 1, pokud je porovnávaná hodnota **menší** než spodní hranice pásma. Pokud je hlídána pouze horní hranice, pak je výstupní hodnota rovna 1, pokud je porovnávaná hodnota **větší**, než horní hranice pásma. Pokud jsou hlídány obě hranice, je na výstupu operace jednička, pokud je porovnávaná hodnota **mimo** pásmo.

7. Nejmenší čtverce (`LeastSquares`) – Metodu řešení soustavy rovnic pomocí metody nejmenších čtverců nejčastěji využijeme při aproximaci nějaké křivky polynomem – například při zjišťování statické charakteristiky čidel. Pro řešení soustavy rovnic je využívána knihovna *commons-math-2.1*.

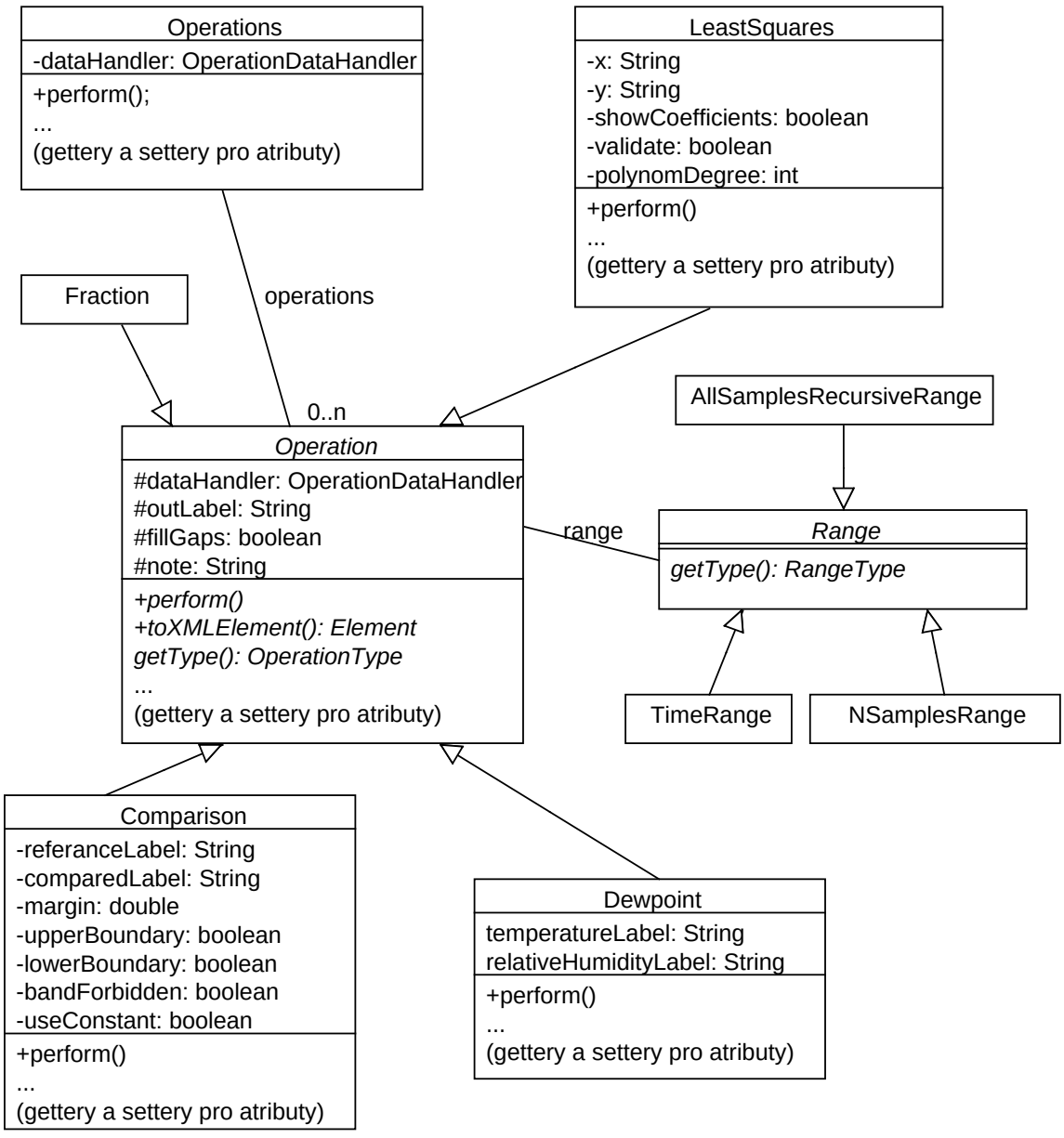
Nyní bude následovat popis vztahů tříd, které implementují výše zmíněné operace a detailnější popis implementace operací *Zlomek* a *Rosný bod*. UML diagram tříd se nachází na obrázku 12 na straně 29.

Na obrázku 12 je vidět třída `Operations`, která slouží ke správě a spouštění všech operací v běžící aplikaci. Přes objekt `Operations` je možné hromadně nastavovat objekt využívaný pro manipulaci s datovými vzorky – jejich načítání a ukládání v rámci operační paměti. Ve starších verzích knihovny bylo toto rozhraní implementováno jednou třídou pro práci ve statickém a druhou třídou pro práci v dynamickém režimu. V současné verzi aplikace se k těmto účelům využívá `DataPool`, který implementuje potřebné rozhraní `OperationDataHandler` a je schopný pracovat jak ve statickém, tak dynamickém režimu. V dalších verzích knihovny bude možné odstranit způsob přistupování k datům přes interface a pracovat rovnou s datovým typem `DataPool`.

V diagramu na obrázku je vidět, že všechny znázorněné operace jsou odvozeny od abstraktní třídy `Operation`. Dceřiné třídy pak implementují abstraktní metody ze třídy `Operation`, včetně metody `perform()`. Tato generalizace umožňuje třídě `Operations` pracovat se všemi typy operací jednotně.

Při volání metody `perform()` (provedení výpočtu) ve třídě `Operations` se spustí postupně metody `perform()` ve všech spravovaných operacích (`Operation`) v pořadí daném odkazy na tyto objekty v ArrayListu `operations` v objektu `Operations`.

Rozsah působnosti každé operace je určen odkazem na příslušný objekt třídy `Range`. Třídu `com.msiroky.dataprocessing.operations.Range` můžete vidět na UML diagramu 12. Samotná třída `Range` je abstraktní a je implementována a konkretizována dalšími třemi. Význam rozsahu se liší podle toho, zda pracujeme s daty v dynamickém, nebo statickém režimu.



Obrázek 12: UML diagram tříd pro provádění operací nad datovými vzorky.

1. `AllSamplesRecursiveRange` V dynamickém režimu dojde k načtení posledního vzorku, který byl do daného datového vlákna uložen. Počítá se zde totiž s tím, že operace bude prováděna po každém vložení a vzorky budou přibývat po jednom. Ve statickém režimu ze z *DataPoolu* vrátí celé datové vlákno.

2. `TimeRange` Jak ve statickém, tak v dynamickém režimu budou načteny z datového vlákna vzorky, které mají časovou značku v rozsahu Intervalu, který je v příslušném objektu `TimeRange` uložen. V knihovně zatím chybí implementace číselného rozsahu `NumberRange`, který by fungoval obdobným způsobem pro číselné řady.

3. `NSamplesRange` V dynamickém režimu bude do operace načteno N posledních vzorků z daného datového vlákna. V režimu statickém pak bude načtené datové vlákno celé a bude zpracováváno po okénkách velikosti N. Použití tohoto rozsahu bylo výše zmíněno u operace **Suma**.

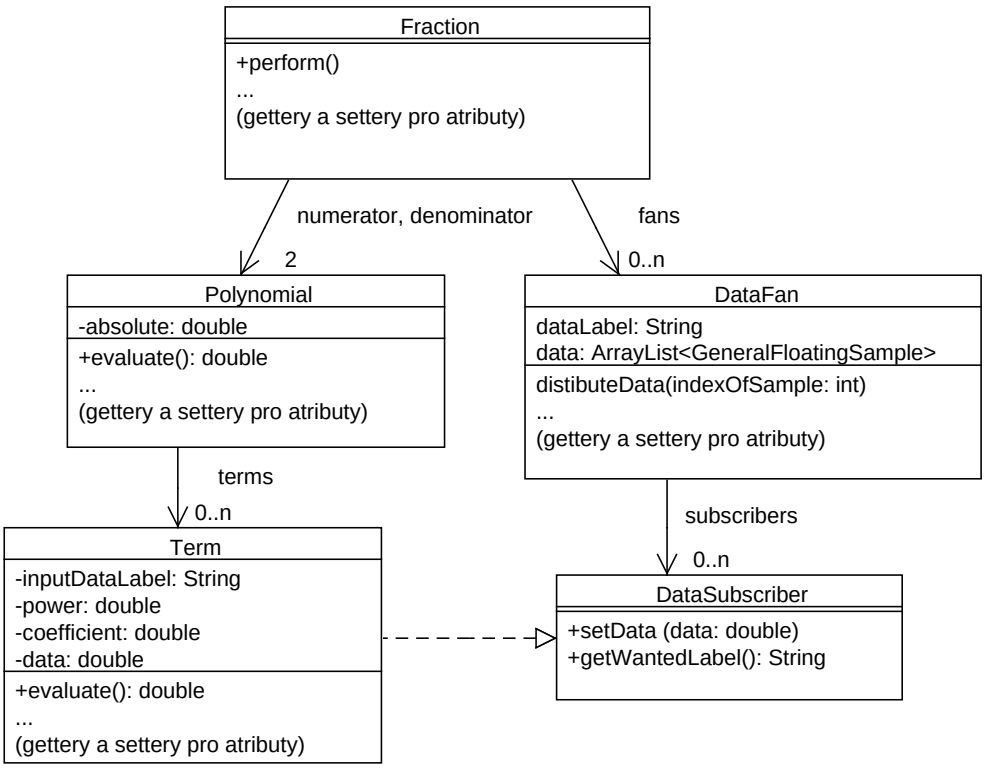
Instance operací lze v běžícím programu vytvářet, programově pomocí napevno napsaného zdrojového kódu, interakcí s uživatelem pomocí grafického rozhraní, anebo na základě informací uložených v XML konfiguračním souboru. Načítání souboru samotného se budeme věnovat později. Jak již ale bylo zmíněno v kapitole 2.3, pro práci s XML soubory je využívána knihovna *Jdom*, pomocí které je možné konfigurační soubor přečíst a v operační paměti reprezentovat v objektové formě. Hlavním stavebním prvkem takto reprezentovaného dokumentu (`org.jdom.Document`) je XML element (`org.jdom.Element`).

Nejprve si popíšeme ukládání parametrů operací do XML elementu. Každá operace implementuje abstraktní metodu `toXMLElement()`. Tato metoda vrací objekt třídy `org.jdom.Element`, ve kterém jsou ve formě XML atributů (`org.jdom.Attribute`) uloženy informace o názvu vstupního (případně více vstupních) a výstupního vlákna operace a také případné další parametry závisející na typu operace. Tento XML element, představující v podstatě serializovanou operaci, v sobě obsahuje také další vnořený element který je „serializovanou“ instanci třídy `Range`. Operace pro výpočet rosného bodu s rozsahem působnosti pro všechny vzorky pak po zapsání do XML elementu a uložení do souboru může vypadat následujícím způsobem:

```
<operation type="DEWPOINT">
    <parametres temperature="t1" humidity="rh1"
    outlabel="rb1" fill_gaps="true" />
    <range type="ALL_SAMPLES_RECURSIVE" />
</operation>
```

3.4.1 Zlomek

Nyní se budeme blíže věnovat operaci *Zlomek* (`Fraction`). Aby u zlomku bylo možné dosáhnout výše zmíněných vlastností a umožnit libovolný počet členů v čitateli a jmenovateli, přičemž každý člen může být libovolným násobkem libovolné mocniny vzorku z libovolného datového vlákna, bylo nutné vytvořit dostatečně pružnou a dynamickou strukturu a provádět výpočty decentralizovaným a hierarchickým způsobem.
S popisem procesu výpočtu zlomku začneme u jeho jednotlivých členů, objektů třídy `Term`. Každý člen v sobě nese informaci o názvu vstupního datového vlákna `inputDataLabel`, hodnotě násobku a mocniny (`coefficient` a `power`) a nakonec také hodnotu zpracovávaného



Obrázek 13: UML diagram tříd souvisejících s výpočetní operací *Zlomek*

vzorku `data`. Vyhodnocení členu se provede voláním metody `evaluate()`. Jak je zřejmé z popisu, ve členu se vyhodnocuje vždy jen jeden vzorek. Pokud chceme vyhodnocovat celé datové vlákno, potřebujeme nějaký mechanizmus distribuce vstupních dat do členů. Tímto mechanismem jsou objekty třídy `DataFan` (vějíře dat).

Pokud je operace `Fraction` generována na základě XML Elementu, jsou nejprve vytvořeny polynomy v čitateli a jmenovateli i s jejich členy. Následně se prochází všechny členy, v obou polynomech a podle názvů jejich vstupních vláken se přidávají do seznamu příjemců dat v příslušném objektu třídy `DataFan`, který má shodný název poskytovaného datového vlákna s tím požadovaným. Pokud zatím neexistuje příslušný `DataFan`, vytvoří se. Z UML diagramu na obrázku 13 je vidět, že za tímto účelem třída `Term` implementuje rozhraní `DataSubscriber` – tedy objekt, který se „upíše“ ke přijímání dat.

Při volání metody `perform()` v objektu `Fraction` se nejprve načtou data ze všech vstupních datových vláken a tyto datové řady se vyrovnají pomocí objektu `ExportPool`.[8] Vyrovnání datových řad znamená, že se vzorky uspořádají podle jejich pořadových značek tak, aby pro každou datovou značku existoval vzorek ve všech datových vláknech. Pokud z nějakého důvodu v některém vstupním vlákně vzorek s takovou pořadovou značkou chybí, je vytvořen a jeho hodnota je nastavena na `NaN`. Výsledek zlomku je pak pro takovou pořadovou značku také `NaN`.

Takto vyrovnaná datová vlákna se následně vloží do příslušných datových vějířů (`DataFan`). Výpočet zlomku pak probíhá tak, že pro každý index v datových vláknech proběhne distribuce hodnoty vzorku na tomto indexu do členů (`Term`), které se zareg-

[8] Třídě `ExportPool` se blíže věnuje kapitola 3.5

istrovaly ke příjmu dat. Jakmile je u všech členů nastavena hodnota atributu `data`, jsou z objektu `Fraction` volány metody `evaluate()` v čitateli a jmenovateli. Tyto metody vracejí součet výsledků členů polynomu a absolutního členu podle vzorce 1 na straně 27.

Zpět v metodě `perform()` je výsledek vyhodnocení čitatele vydělen výsledkem vyhodnocení jmenovatele. Pokud je jmenovatel rovný nule, je výsledek zlomku nastaven na `NaN`. Je vytvořen nový objekt datového vzorku (podle typu vstupních dat buď `Numbered-` nebo `TimeStamped-` FloatingSample), je do něj uložen výsledek a přiřazena příslušná pořadová značka. Výstupní vzorek je zařazen do seznamu v objektu `Fraction`.

Výše zmíněný postup distribuce dat a výpočtů se opakuje pro všechny indexy vstupných datových vláken a po zpracování posledního indexu jsou výstupní data zařazena mezi ostatní datová vlákna v operační paměti pomocí objektu `DataPool`.

Ilustrace operace `Fraction` uložené do XML:

```
<operation type="FRACTION">
    <parametres outlabel="rh2" fill_gaps="true" />
    <polynomial absolute="0.0">
        <term coefficient="1.0" variable="rh2_temp" power="1.0" />
    </polynomial>
    <polynomial absolute="1.0" />
    <range type="ALL_SAMPLES_RECURSIVE" />
    <note></note>
</operation>
```

3.4.2 Rosný bod

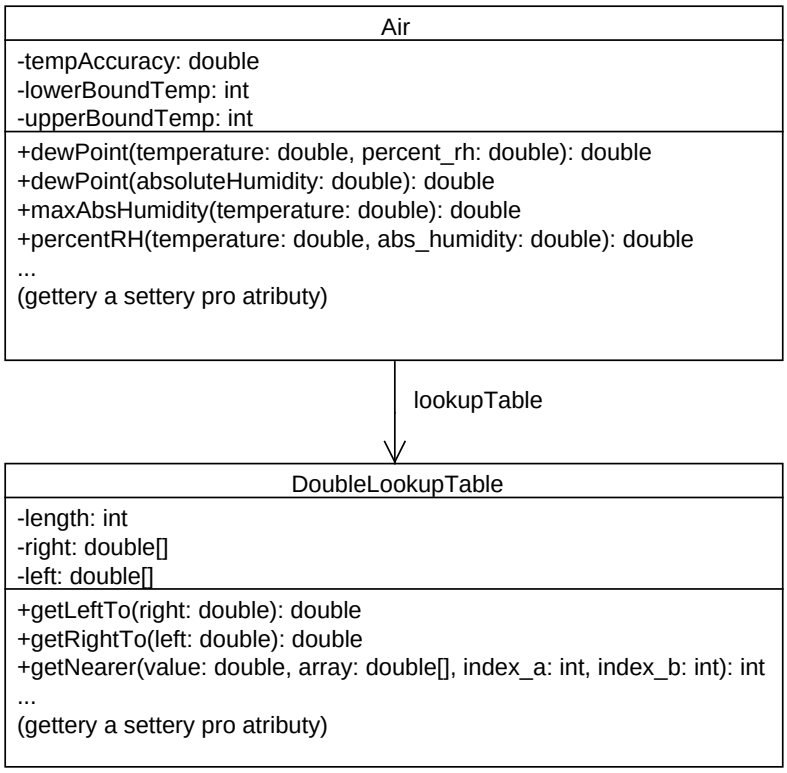
pro výpočet rosného bodu vzduchu používá třída `Dewpoint` vnitřně objekt třídy `Air`, který je využíván k reprezentaci vzduchu. Pomocí metod ve třídě `Air` je možné získávat informace o teplotě rosného bodu, absolutní a relativní vlhkosti vzduchu.

V mé bakalářské práci [1] byl k výpočtům rosného bodu používán program *Octave*. Výpočet byl založen na aproximaci vztahu mezi teplotou vzduchu a jeho maximální absolutní vlhkostí pomocí polynomu 3. stupně.

Výpočet rosného bodu vzduchu ze znalosti relativní vlhkosti a teploty pak spočívá v nalezení hodnoty maximální absolutní vlhkosti pro danou teplotu, vydělením této hodnoty velikostí relativní vlhkosti (dostaneme aktuální absolutní vlhkost) a nalezení kořene (hodnota teploty) polynomu pro zjištěnou hodnotu absolutní vlhkosti. Vzhledem k tomu, že polynom je na pracovním rozsahu -10 až $+40$ ^{circ}C monotónní, dostaneme pouze jedno reálné řešení.
Hledání kořene polynomu není však výpočetně rychlá operace a na rozdíl od *Octave* není ve standardních knihovnách jazyka Java implementována. K řešení jsem se nechal inspirovat znalostmi z předmětu *Numerické metody* (KMA/NM).

Při inicializaci objektu třídy `Air` je vytvořena vyhledávací tabulka a její levý sloupec je vyplněn hodnotami teploty v rozsahu od `lowerBoundTemp` do `upperBoundTemp` s krokem `tempAccuracy`. Pro každou hodnotu teploty je současně vyčíslen polynom a získaná hodnota maximální absolutní vlhkosti je uložena do pravého sloupce.

Při hledání teploty rosného bodu pak objekt `Air` opět vyčíslí polynom a zjistí hodnotu maximální absolutní vlhkosti. Tuto hodnotu vydělí relativní vlhkostí (Rozsah 0 až 100% se mapuje na 0 až 1.) čímž získá aktuální absolutní vlhkost. Pro tuto hodnotu absolutní



Obrázek 14: Třída `Air` a vyhledávací tabulka

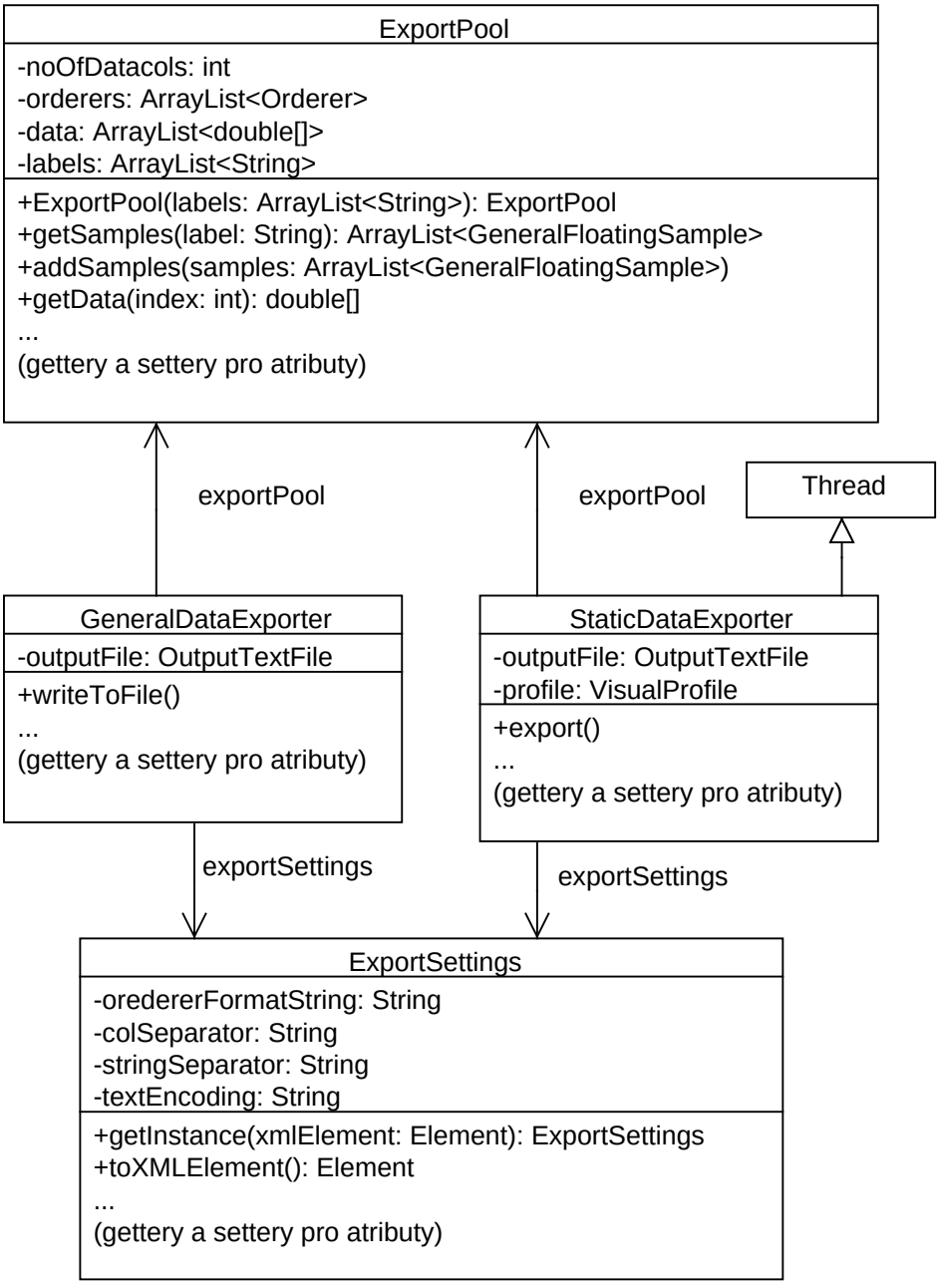
vlhkosti je ve vyhledávací tabulce metodou `getLeftTo()` nalezena korespondující teplota. Nejprve v tabulce probíhá vyhledávání co nejbližší hodnoty absolutní vlhkosti metodou půlení intervalu. Když hledaná hodnota absolutní vlhkosti spadá mezi dva sousední indexy v tabulce, je vybrán index na kterém je hodnota bližší té hledané. Pokud porovnání vyjde nerozhodně, je upřednostněn vyšší index. Jako teplota rosného bodu je následně vrácena hodnota na příslušném indexu z levé poloviny tabulky.

3.5 Export dat

Export dat a ukládání dat do textových souborů je velmi důležitou funkcí měřicí aplikace a metody, které ji umožňují jsou dostatečně implementovány v knihovně `libdev`.

Hlavní třídy určené pro export dat do souborů můžeme vidět na obrázku 15. Třída `ExportPool` byla zmíněna již dříve a proto by bylo dobré začít výklad u ní. ExportPool je v podstatě tabulka s pevným počtem sloupců, které se vytvoří v konstruktoru instance a odpovídají počtu datových vláken, která budeme ukládat. Počet řádků tabulky je proměnný. V předchozím textu bylo zmíněno, že `ExportPool` se využívá k vyrovnávání datových řad. tato jeho funkce má původ ve skutečnosti, že může nastat situace, kdy budeme chtít do jednoho CSV souboru uložit data z různých datových vláken, přičemž pro určité pořadové značky nebude hodnota přítomna ve všech vláknech. V některých vláknech budou zkrátka mezery.

Při vkládání vzorku do *ExportPoolu* dojde k rozdělení složek vzorku na hodnotu a pořadovou značku. Pořadové značky se ukládají v samostatném sloupci pro všechna da-



Obrázek 15: Třídy pro export dat

tová vlákna, nikoliv pro samostatné vzorky. Pokud již v tabulce existuje řádka s příslušnou pořadovou značkou, je hodnota vložena do sloupce této řádky, který odpovídá názvu datového vlákna. Pokud zatím taková řádka neexistuje, je vytvořena. Řádky jsou automaticky řazeny a vkládány podle pořadových značek podobně, jako se to děje s plovoucími vzorky v *DataDraweru*.

Pokud tedy dojde k tomu, že sloupce v některých řádkách jsou neobsazené, považuje se při exportu hodnota na těchto pozicích za `NaN`. Při exportu má tedy každá řádka plný počet sloupců, přičemž na některých pozicích je hodnota `NaN`.

Pro uchovávání nastavení exportu slouží třída `ExportSettings`. Objekty této třídy je možné ukládat do XML, nebo je zpět načítat. Uložené parametry souvisí s vlastnostmi CSV souborů, jako je oddělovač sloupců a oddělovač řetězců. Nastavit lze také kódování textu a formátovací řetězec pro pořadovou značku řádku souboru. U číselných řad nemá formátovací řetězec vliv a slouží jen jako titulek prvního sloupce souboru. U řad časových má vliv na způsob výpisu data a času. Pro formátovaný výstup data a času je používána třída `java.text.SimpleDateFormat`. Informace o používání formátovacího řetězce lze nalézt v dokumentaci k této třídě. Obdobně jako u číselných řad je i zde formátovací řetězec titulkem 1. sloupce. Nadpisy dalších sloupců odpovídají názvům příslušných datových vláken. Pořadí sloupců je stejné jako pořadí názvů datových vláken zadané při vytváření *ExportPoolu*.

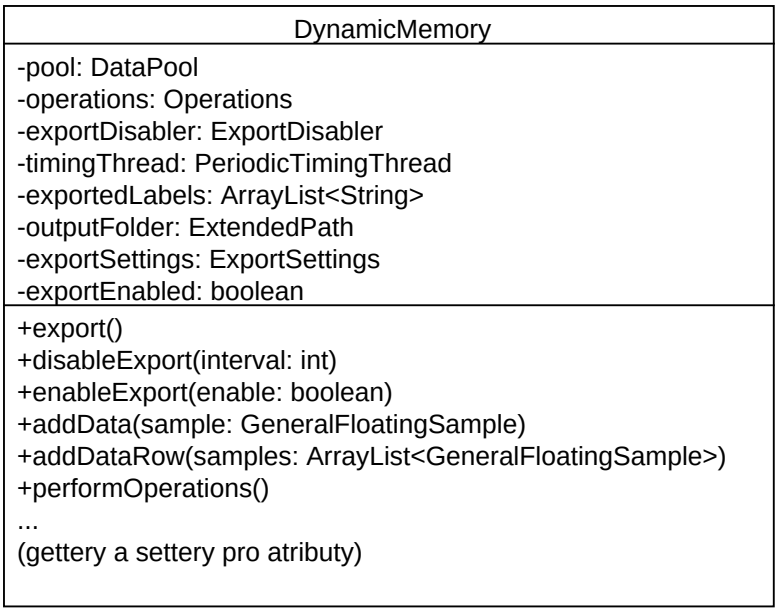
Pro export v dynamickém a statickém režimu se v knihovně používají dva různé mechanismy, které mohou být v další verzi knihovny sloučeny do jednoho. V současné době slouží pro export postupně přibývajících dat třída `GeneralDataExporter` a pro data statická pak třída `StaticDataExporter`. Jak je patrné z UML diagramu na obrázku 15, `StaticDataExporter` dědí od třídy `Thread` (Je to programové vlákno). Tento přístup vychází z toho, že v případě statických dat, je objem najednou ukládaných dat větší než v dynamickém případě a s větší pravděpodobností se bude i jednat o aplikaci, která má grafické rozhraní a komunikuje s uživatelem. Z tohoto důvodu je vhodnější, aby proces ukládání probíhal na pozadí a aplikace mohla i nadále reagovat na akce uživatele.

Proces průběžného ukládání dat v měřicí aplikaci je řízen třídou `DynamicMemory`. Tato třída zajišťuje periodické ukládání a generuje názvy a cesty cílových CSV souborů. UML diagram této třídy se nachází na obrázku 16 na straně 36.

Do *DynamicMemory* přicházejí naměřená data a jsou ukládána do *DataPoolu*. V měřicí aplikaci se využívá vkládání dat po řádcích, přičemž se po vložení řádku automaticky spustí provedení výpočetních operací. Objekt `outputFolder` uchovává informaci o základní cestě k ukládání dat. Ukládání se spouští periodicky pomocí časovacího vlákna `timingThread`. Data se ukládají do souboru podle kalendářního data. Cesta k exportnímu souboru je odvozena od cesty základní a dále je vytvořena složka podle roku (ve formátu `yyyy`) a podsložka podle měsíce (ve formátu `MM`). Exportní soubory jsou pojmenovávány ve formátu `yyyy_MM_dd`. Pokud je třeba pracovat se soubory s exportovanými daty, je možné periodické ukládání pozastavit na požadovanou dobu pomocí metody `disableExport(interval: int)`, kde `interval` udává počet milisekund.

3.6 Vizualizace dat

Jak již bylo uvedeno výše, pro samotné vykreslování dat je využívána knihovna *JFreeChart*. Proces mezi získáním dat a jejich vykreslením na obrazovku je pak řízen třídami z balíku `com.msiroky.charting.profiles`. Řízení probíhá pomocí tzv. *vizuálních pro-*



Obrázek 16: Třída `DynamicMemory`

filů. Vizuální profil je představován třídou `VisualProfile` a od ní jsou odvozené třídy `PresentationProfile` a `MonitoringProfile`. UML diagram těchto tříd se nachází na obrázku 17 na stránce 37.

Profily třídy `MonitoringProfile` slouží k řízení zobrazování dynamických dat a profily třídy `PresentationProfile` pro řízení zobrazování dat statických. Jak je vidět z diagramu, profily jsou odvozeny od „v paměti uloženého textového souboru“[9]. To jim umožňuje, aby mohly být snadno uloženy na disk ve formě textového XML souboru, nebo opět z disku načteny.

Jak z obrázku vidíme, `VisualProfile` obsahuje odkaz na objekt třídy `Plot` a ten pak seznam odkazů na objekty třídy `Subplot`. Tyto třídy uchovávají informace o vzhledu grafu a podgrafů a v případě vykreslování statických dat se starají o načtení dat ze souborů do operační paměti (do *dataPoolu*). V případě dynamických dat a monitorování se používají rozšířené třídy `MonitoringPlot` a `MonitoringSubplot`.

Monitorovací graf (*plot*) se vytváří s prázdnými podgrafy (*subploty)* a rozšířená třída umožňuje exponovat odkazy na datové řady[10] vykreslované do panelu knihovnou *JFreeChart*. Díky tomu je možné do dynamického grafu vkládat vykreslovaná data z objektů mimo strukturu vizuálních profilů.

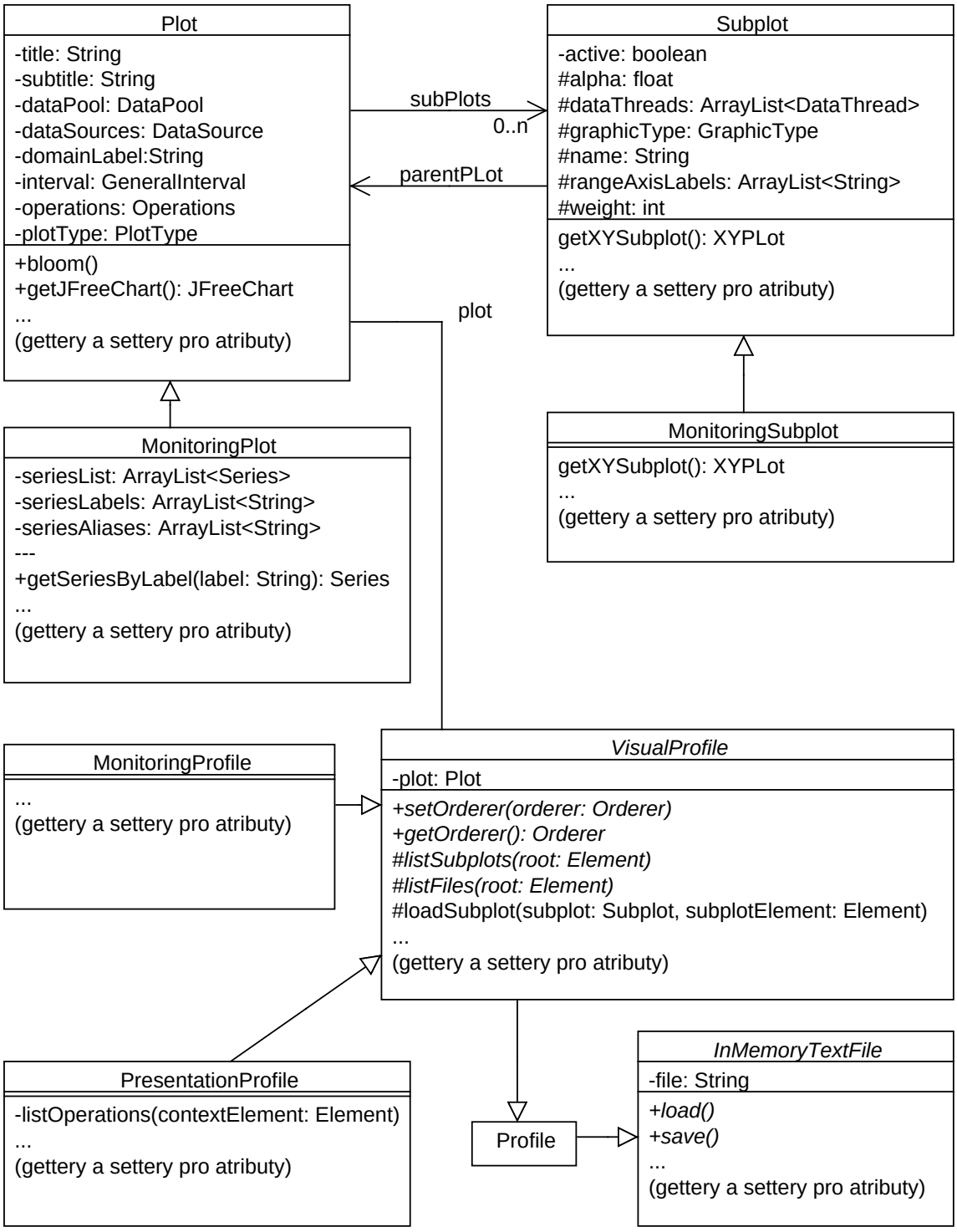
Zobrazování statických dat se od zobrazování těch dynamických liší tím, že umožňuje provádět nad načtenými daty výpočetní operace.[11] Provádění operací nad daty vkládanými do dynamického grafu není podporováno. Zde se počítá s tím, že data jsou upravena již před vložením do grafu. V architektuře měřicí aplikace se o zpracování dat stará „serverová část“, zatímco část klientská a vizualizační operace nad daty již neprovádí.

Zde se dostáváme k dalšímu rozdílu v použití dynamického a statického zobrazování

[9] `InMemoryTextFile`

[10] třída `org.jfree.data.general.Series`

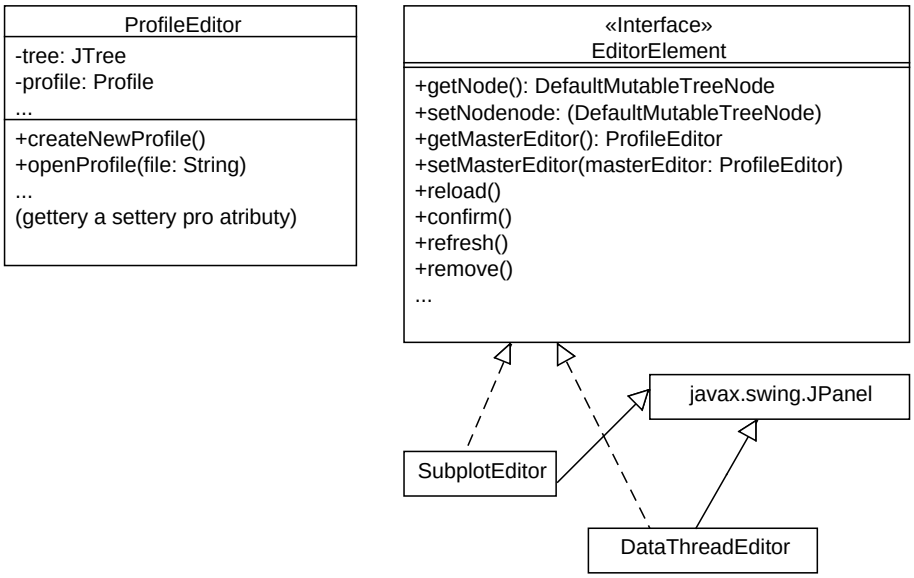
[11] Mechanismu výpočetních operací se věnuje oddíl 3.4.



Obrázek 17: Třídy pro vizualizaci dat

dat. Zatímco přístup dynamický se uplatní u měřicí aplikace, která se nastavuje na míru konkrétní situaci a měřicímu zařízení a po zprovoznění není již nutné toto nastavení měnit, uplatnění vizualizace statických dat vyžaduje mnohem častější změny konfigurace. Tento fakt vychází ze situace, že vizualizace statických dat není vázaná na konkrétní měřený proces a konkrétní obsluhu, ale je možné vizualizační aplikaci využívat pro zobrazování dat různého původu a z různých zdrojů (souborů různého formátu). Aplikace pro vizualizaci statických dat je určena širšímu okruhu uživatelů a je žádoucí, aby bylo možné jednoduše nastavit, z jakých zdrojů data načíst, jaké operace nad nimi provést a jak je vykreslit. Z tohoto důvodu jsem vytvořil pro editování profilů pro zobrazování statických dat grafický editor, který je součástí knihovny `libdev`. V další verzi knihovny může být tento editor rozšířen i na profily dynamické.

3.7 Editor vizualizačních profilů



Obrázek 18: Třídy pro editování vizualizačních profilů

Na obrázku 18 jsou znázorněny třídy grafického editoru vizuálních profilů. Hlavní třídou je `ProfileEditor`. Editovací elementy (třídy implementující rozhraní `EditorElement` jsou pak organizovány pomocí objektu `JTree`, který tyto jednotlivé elementy zobrazuje v hierarchické stromové struktuře. Z diagramu je vidět, že elementy jsou rozšířeními třídy `JPanel`, což umožňuje jejich zobrazování na monitoru. Každý element umožňuje provádění úprav nad určitou částí profilu. Například jeden element slouží k vytváření a mazání podgrafů a jiné elementy pak umožňují upravovat vlastnosti konkrétního podgrafu. Popis grafického editoru a jeho ovládání bude uveden v návodu k obsluze vizualizační aplikace.

4 Měřicí aplikace

Následující část se bude věnovat popisu architektury měřicí aplikace. Z důvodu robustnosti je aplikace rozdělena na dvě části: *Server* a *Klient*. Komunikace mezi serverem a klientem

probíhá v protokolu XML-RPC.

Vývoj měřicí aplikace jsem prováděl v *NetBeans* v rámci jednoho projektu typu *Java Application* nazvaného *airflow_server*. Po kompilaci projektu je vytvořen spustitelný binární soubor typu JAR. Způsob běhu aplikace (server / klient) je určen parametry zadanými do *Java VM* [12] při spouštění aplikace.

4.1 Server

Pro běh v režimu serveru je aplikace spouštěna metodou `main()` ve třídě `Server` v balíku `airflow_server.server`. Metoda `main()` akceptuje jeden parametr, který udává cestu ke XML souboru s nastavením. Pokud není zadán žádný parametr, program se ukončí. Pokud cesta k souboru zadána je, následuje vytvoření instance třídy `Server`. V konstruktoru třídy `Server` je spuštěno načtení souboru s nastavením.

Obsah souboru je načten do objektu třídy `JdomXml`.[13] Odkazy na objekty `JdomXml` jsou udržovány statickou třídou `JdomInterchangeBeacon`[14] a každý objekt v rámci běžící aplikace může k těmto objektům přistupovat. V objektu `JdomInterchangeBeacon` jsou vždy uloženy dvojice *název* a *odkaz na objekt* (*klíč* a *hodnota*). Odkaz na příslušný `JdomXml` je získán na základě jeho názvu.

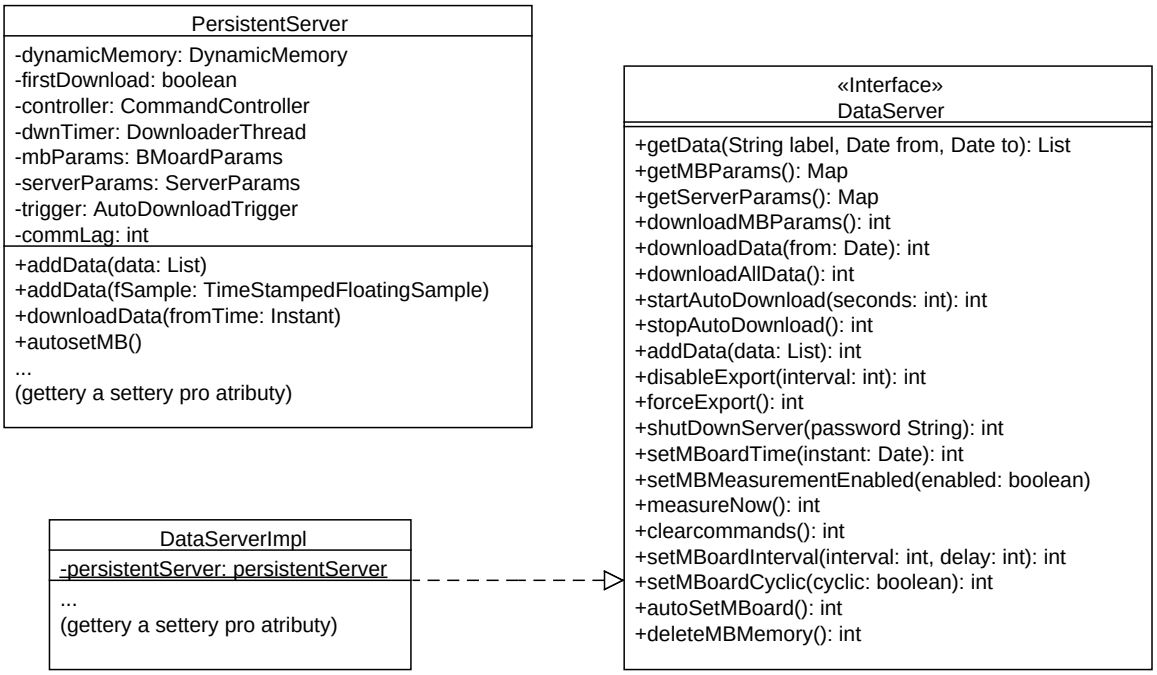
Dále je spuštěn měřicí server. Nejdříve vytvořena instance třídy `PersistentServer`.[15] Tento objekt je centrální částí celé serverové aplikace. Existuje pouze v jedné instanci a obstarává časování komunikace s měřicím zařízením a zpracovává požadavky z klientských aplikací. `PersistentServer` také zařizuje automatické nastavení *M-Boardu*. Dále se odkaz na instanci `PersistentServer` vloží do statické proměnné třídy `DataServerImpl`.[16]

Pokud je v konfiguračním souboru serveru aktivován vizuální mód, vytvoří se grafická interaktivní konzole (objekt třídy `IConsole`), do které se vypisují textové řetězce které přicházejí z měřicího zařízení a pomocí konzole je také možné měřicímu zařízení posílat příkazy přímo.

Dále je vytvořena instance třídy `MySerialPortImpl`[17], která pomocí knihovny *RXTX* zajišťuje komunikaci se sériovou linkou. Podle údajů v konfiguračním souboru je nastaven název portu a přenosová rychlost. Výstup z *CommandControlleru*[18] v *PersistentServeru* je nasměrován na sériovou linku. Příkazy z *CommandControlleru* jsou odesílány po sériové lince a řetězce, které sériová linka přijme jsou směrovány do *CommandControlleru*.

Dále je nastartován webový XML-RPC server: Nejprve se vytvoří instance webového serveru `WebServer`[19], který pracuje na portu zadaném v konfiguračním souboru. Z instance webového serveru je pak získán odkaz na objekt třídy `XmlRpcServer`, kterému jsou nastaveny *handlery* podle informací v souboru `MyHandlers.properties` v balíku `airflow_server/server`. Na základě instrukcí v tomto souboru je *xmlRpcServeru* přiděleno rozhraní `DataServer` a je zadána informace, že metody tohoto rozhraní implementuje třída `DataServerImpl`.[20]

[12] *Java Virtual Machine: virtuální stroj, který interpretuje kompilovaný kód*
[13] z balíku `com.msiroky.filesystem.files.xml`
[14] z balíku `com.msiroky.parallels`
[15] z balíku `airflow_server.server`
[16] z balíku `airflow_server.handlers`
[17] z balíku `com.msiroky.hw.serial`
[18] objekt, který řídí tok příkazů měřicímu zařízení a zpracovává odpovědi, které ze zařízení přicházejí
[19] v balíku `org.apache.xmlrpc.webserver`, z knihovny *Apache XML-RPC*
[20] oboje v balíku `airflow_server.handlers`

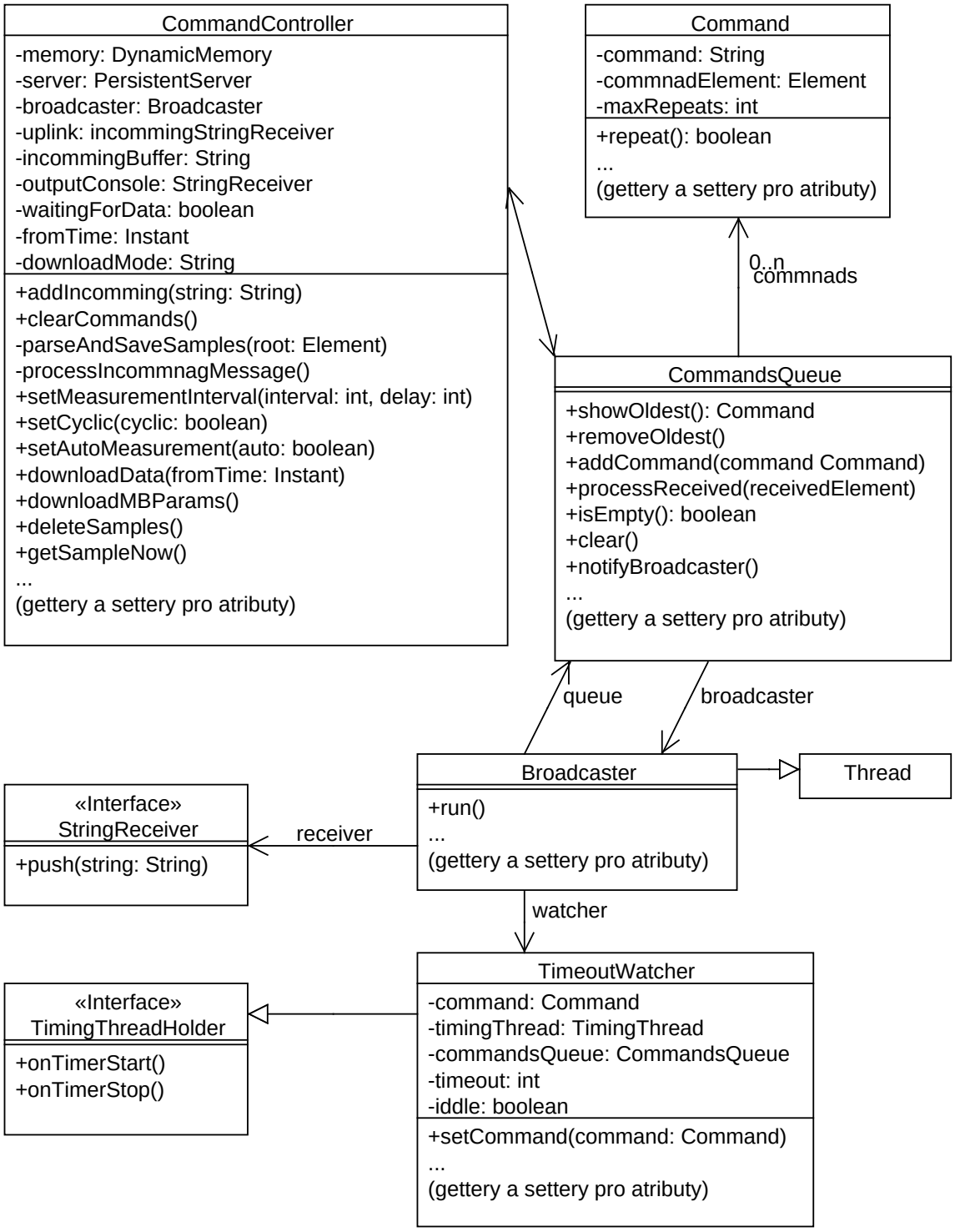


Obrázek 19: Základní třídy měřicího serveru

UML diagramy popisovaných tříd jsou uvedeny na obrázcích 19 a 20. Princip volání vzdálených metod na serveru funguje na základě faktu, že rozhraní se signaturami metod zná jak server, tak klient. Metody, které chceme z klienta volat **musejí** mít návratovou hodnotu. Metody, které vracejí `void` nejsou podporovány. Při volání metody serveru z klienta je (v běžném režimu, který je uplatněn i zde) vytvořena vždy nová instance implementující třídy. V tomto případě se jedná o třídu `DataServerImpl`. Tak následně volá metody z výše zmíněného objektu třídy `PersistentServer`. Proces odeslání výsledku klientu již obstarávají třídy z knihovny *Apache XML-RPC*. Na obrázku 20 na straně 41 jsou znázorněny třídy, které zajišťují řízení komunikace s měřicím zařízením a vyhodnocování odpovědí z něj.

Jak již bylo zmíněno výše, komunikace mezi měřicí aplikací a měřicím zařízením probíhá znakově po sériové lince, přičemž příkazy pro zařízení a odpovědí ze zařízení mají formu XML elementů. Ovládání měřicího zařízení *M-Board* se děje přes volání metod ve třídě `CommandController` , které vygenerují potřebný textový příkaz pro odeslání *MBoardu*.

Vygenerovaný řetězec se vloží do nové instance třídy `Command`, která se vloží do fronty příkazů v objektu třídy `CommandQueue`. Zasílání příkazů *M-Boardu* je řízeno objektem třídy `Broadcaster`, který rozšiřuje třídu `Thread`. Pokud ve frontě jsou přítomné příkazy, *Broadcaster* odešle řetězec prvního příkazu ve frontě po sériové lince. Odkaz na příslušnou instanci *Commandu* je vložen do *TimeoutWatcheru*. Když je serverem (*CommandControllerem*) zachycena odpověď z měřicího zařízení, dojde nejprve k převodu řetězce na XML element (třída `Element`), čímž je provedena i kontrola integrity příchozí zprávy. Pokud přijatá odpověď koresponduje s posledně odeslaným příkazem, je příkaz z fronty odstraněn, *TimeoutWatcher* vynulován a je odeslán nový příkaz.



Obrázek 20: Třídy řídící komunikaci s měřicím zařízením
(balík `airflow_server.server.communications`)

Pokud však po uplynutí zvoleného času (timeout) je ve frontě stále tentýž příkaz, *TimeoutWatcher* příkaz odešle znovu a opakuje se čekání na příchod odpovědi. Každý příkaz může mít individuálně nastavený počet opakování (Výchozí nastavení je 3) a pokud ani po maximálním počtu opakování nepřijde z měřicího zařízení odpověď, je fronta příkazů vymazána. Informace o nedoručitelnosti příkazu je uvedena v souhrnu o stavu měřicího zařízení, který server může klientské aplikaci zasílat a také je zapsána do logového souboru.

4.1.1 Ovládání měřicího zařízení

Server je schopný ovládat *M-Board* v těchto oblastech:

1. Nastavení času na měřicí kartě
2. Povolení / zakázání automatického měření
3. Nastavení měřicího intervalu a zpoždění před sejmutím prvního vzorku
4. Přepnutí mezi kruhovou pamětí (kdy se nejstarší vzorky přepisují novými) a pamětí lineární (kdy se po zaplnění paměti záznam ukončí)
5. Stažení dat od určitého času
6. Stažení dat od určitého času s následným smazáním těchto dat z měřicího zařízení
7. Smazání všech dat ze zařízení
8. Zjištění stavu zařízení (čas, počet vzorků v paměti, nastavení paměti, měřicí interval)

Při stahování dat z měřicího zařízení probíhá převod zprávy do XML elementu po částech (po jednotlivých vzorcích). V případě, že je část zprávy (vzorek) poškozena, vyloučí se ze zpracování pouze tato část, která se uloží do logového souboru a zpracování dalších vzorků pokračuje.

Bližší informace o komunikaci a vyhodnocování příchozích zpráv je možné získat prostudováním zdrojových kódů v elektronické příloze. Zdrojové kódy jsou opatřeny komentáři, které usnadňují pochopení jednotlivých funkcí.

Pro účely testování serveru a klienta je zde možnost server spouštět v simulačním režimu, kdy v rámci serveru běží modelové měřicí zařízení.[21] Přepínání mezi simulačním a provozním režimem se provádí v konfiguračním souboru. Konfigurační soubory jsou rovněž součástí elektronické přílohy.

4.2 Klient

Klient byl vytvářen ve stejném *Java* projektu jako server. V budoucnu by však bylo výhodnější mít pro klient vyčleněný samostatný projekt tak, aby spolu s klientem nebyly distribuovány i třídy serveru. UML diagram hlavních tříd, které tvoří klientskou aplikaci je znázorněn na obrázku 21 na straně 44.

Klient je spouštěn metodou `main()` ve třídě `ClientStarter`.[22] `ClientStarter` má obdobnou funkci, jako třída `Server`. Nejprve je načten konfigurační soubor a odkaz na

[21] Modelové zařízení je reprezentováno třídami z balíku `airflow_server.server.model`.

[22] v balíku `airflow_server.client`

něj je umístěn do `JdomInterchangeBeacon`. Dále je vytvořena ikona v notifikační oblasti pracovní plochy (v systému *Windows* obvykle na pravé straně hlavního panelu). Ikona indikuje běh klientské aplikace a disponuje kontextovým menu pro zobrazování kontrolního panelu klienta a ukončování aplikace.

Dále je vytvořena instance třídy `ControlPanelImpl`, která implementuje rozhraní kontrolního panelu `ControlPanel`. Použití rozhraní umožňuje vytvářet a používat různé verze vizuální podoby panelu, přizpůsobené na míru konkrétním podmínkám a požadavkům, bez nutnosti větších zásahů do zdrojového kódu ostatních tříd. Následně je načten zobrazovací profil pro graf v dynamickém režimu. Po provedení těchto operací je volána metoda `run()` objektu `Client`, čímž se spustí automatická část klientské aplikace.

Jak je patrné z diagramu 21, `Client` rozšiřuje `Thread`. V těle překryté metody `run()` probíhá cyklus, který v intervalu specifikovaném v konfiguračním souboru kontaktuje pomocí protokolu *XML-RPC* server a získává od něj data stažená z měřicího zařízení a informace o stavu serveru a zařízení. Aby bylo možné volat vzdálené metody serveru, je nutné ve zdrojovém kódu použít následující řádky:

```
//Vytvoření konfigurace s adresou serveru
XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
config.setServerURL(new URL(host + ":" + port + "/" + serviceName));
//Vytvoření a nastavení objektu client:
XmlRpcClient client = new XmlRpcClient();
client.setConfig(config);
//Vytvoření napojení na server:
ClientFactory factory = new ClientFactory(client);
DataServer server = (DataServer) factory.newInstance(DataServer.class);
```

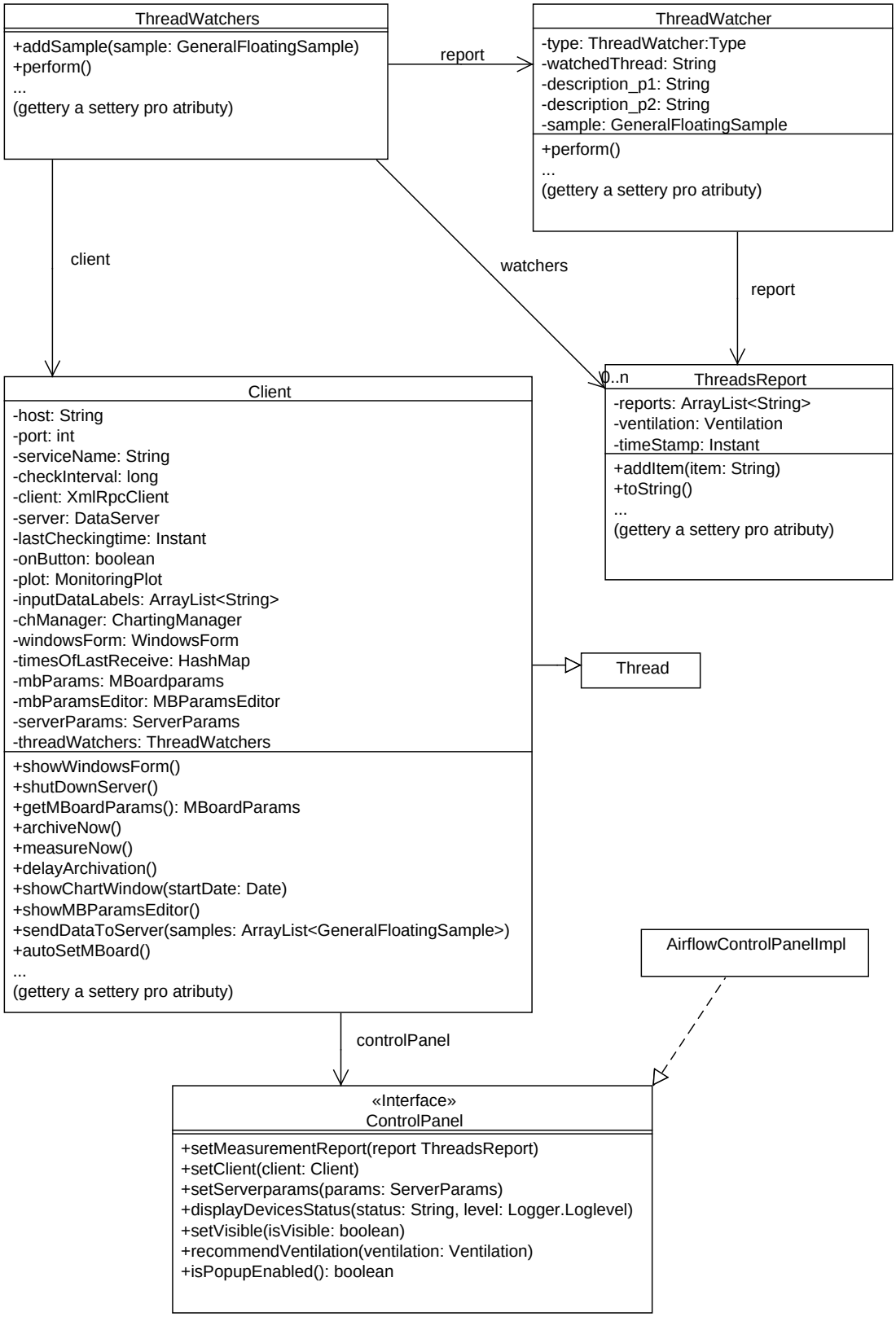
Třídy `ClientFactory` a `XmlRpcClient` pochází z knihovny *Apache XML-RPC* a třída (rozhraní) `DataServer` byla popsána již dříve. Nyní můžeme ze zdrojového kódu volat vzdálené metody serveru (definované rozhraním `DataServer`) stejně, jako bychom volali metody lokální.

4.2.1 Automatická komunikace se serverem

V rámci automatické komunikace klienta se serverem jsou prováděny následující činnosti:

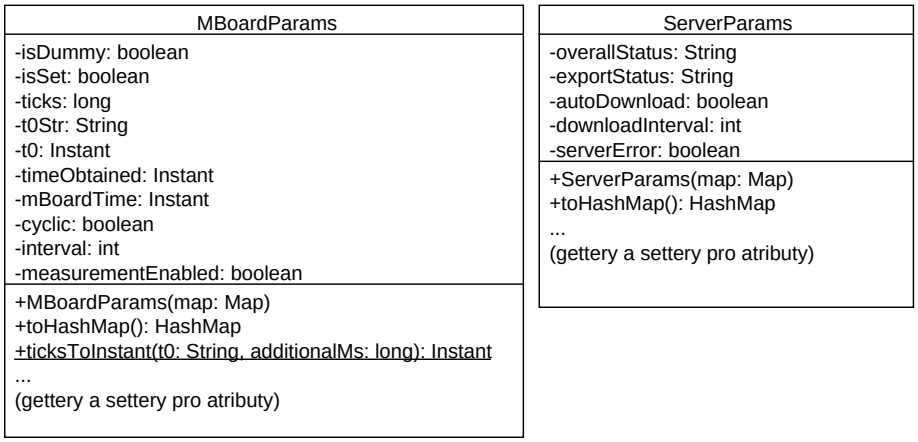
Získávání informace o stavu serveru: Je prováděno voláním vzdálené metody `getServerParams()`, která vrací objekt třídy `java.util.Map`, ve kterém jsou uloženy dvojice *klíč*; *hodnota*. Při přijetí objektu třídy `Map` je obsah tohoto objektu převeden do instance třídy `ServerParams`. UML diagram třídy `ServerParams` se nachází na obrázku 22.
V `ServerParams` jsou následující proměnné:

- `overallStatus` sděluje, zda je server v pořádku a zda běží bez chyb.
- `exportStatus` informuje o tom, zda je povolené automatické ukládání stažených dat do souboru, či nikoliv, případně zda je pouze pozastaveno
- `downloadInterval` udává interval automatického stahování dat z *MBoard* v ms



Obrázek 21: Hlavní třídy klientské aplikace

- `autoDownload` má hodnotu `true`, pokud je automatické stahování dat ze zařízení serveru povoleno, `false` pokud není.
- `serverError` – Pokud komunikace se serverem funguje, má hodnotu `false`, pokud server na volání metod neodpovídá, je `true`.



Obrázek 22: Třídy pro reprezentaci stavu a parametrů serveru a měřicího zařízení

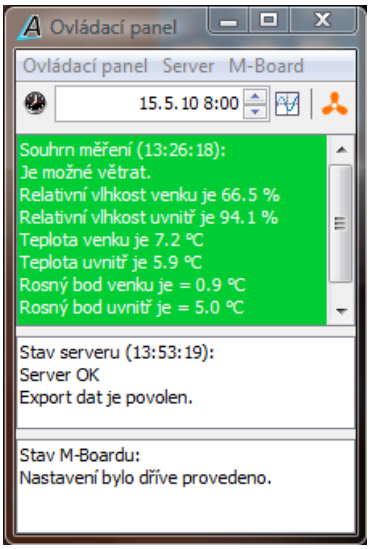
Získávání informací o stavu měřicího zařízení: Je prováděno voláním vzdálené metody `getMBParams()`. Informace vrácené opět ve formě objektu třídy `Map` jsou převedeny do instance třídy `MBoardParams`. V klientu jsou zužitkovány a případně zobrazeny následující informace z *MBoardParams*:

- Aktuální parametry byly staženy z měřicího zařízení, či se zatím jedná o výchozí hodnoty: `isDummy`
- Čas na M-Boardu byl, nebo nebyl nastaven: `isSet`
- Současný čas na M-Boardu: `mBoardTime`
- Cyklická / lineární paměť: `cyclic`
- Povoleno automatické měření: `measurementEnabled`
- Interval automatického měření ve vteřinách: `interval`

Stahování dat: Je prováděno voláním vzdálené metody `getData(String label, Date from, Date to)`, která vrací data s časovými značkami spadajícími do intervalu vymezeného `from` a `to` z vlákna s daným názvem `label`. Data ze serveru jsou vrácena jako seznam (`java.utilList`) objektů třídy `Map`, jejichž obsah je následně převeden na seznam objektů třídy `GeneralFloatingSample`. Klient ukládá u každého datového vlákna časovou značku posledního přijatého vzorku. Tato časová značka je pak počátkem intervalu pro další stahování. Koncem intervalu je pak čas, ve kterém je stahování zahájeno. Pokud je okno pro graf spuštěno, jsou stažená data do grafu přidána a zobrazena.

4.2.2 Sledování hodnot a informování uživatele

Data jsou zpracovávána pomocí operací v serveru. Na nastavení klientské aplikace však záleží způsob, jakým budou příchozí hodnoty uživateli prezentovány. Možnost prezentace dat je dvojí. Jak již bylo zmíněno, vzorky je možné zobrazovat v grafu, nebo je možné pouze vypisovat souhrn měření do kontrolního panelu. Okno kontrolního panelu je na obrázku 23.



Obrázek 23: Okno kontrolního panelu klientské aplikace běžící v plaském klášteře; V horní části okna jsou ovládací prvky, pod nimi jsou textová pole se souhrnem měření, popisem stavu serveru a popisem stavu měřicího zařízení.

Vidíme, že textové pole se souhrnem měření je zelené. Zelená barva v poli znamená, že venkovní vzduch je dostatečně suchý, aby se dalo větrat bez rizika. Vzduch venku je však příliš studený a při větrání by se ochlazovala horní patra budovy. Z tohoto důvodu větrání při tomto stavu probíhat nebude. Úvaha o rozdílu teploty venku a uvnitř budovy ve vyšších patrech zůstává na obsluze, protože měřicí zařízení nemá z důvodu omezeného počtu vstupů tato data k dispozici. Odhad teploty je však snazší, než odhad vlhkosti a tak tento nedostatek není příliš závažný.

Formát souhrnu měření a barva okna je výsledkem činnosti objektů třídy `ThreadWatcher`, hromadně spravovaných objektem třídy `ThreadWatchers`. Vztah mezi těmito třídami můžeme vidět na UML diagramu 21 na straně 44. `ThreadWatchers` v metodě `perform()` generuje zprávu o měření `ThreadsReport`, která je předána objektu třídy `Client` ke zobrazení.

Každá jednotka `ThreadWatcher` sleduje hodnotu jednoho datového vlákna zadaného v konfiguračním souboru a může pracovat ve třech režimech:

1. **Informativní:** Hodnota vlákna je uvedena mezi dvěma textovými řetězci, které je možné nastavit v konfiguraci a tento celý řetězec je uložen jako jedna položka zprávy o měření. Taková položka pak může být např. „Relativní vlhkost venku je 66.5 %“

2. **Dobrý alarm:** Jedná se primárně o sledování výstupního vlákna z funkce typu *porovnání*. Pokud je *watcher* v tomto režimu, a hodnota sledovaného vlákna je 1, je do souhrnu přidána řetězcová položka popisující vzniklý stav (např. „Je možné

větrat.“ a v příslušné instanci třídy `ThreadsReport` je nastaven doporučený stav větrání na „otevřeno“.

3. **Špatný alarm:** Funkce je obdobná, jen v případě, že sledovaná hodnota je 1, doporučený stav větrání je nastaven na „zavřeno“. V souhrnu měření se pak v tomto případě objeví řádka „Nevětrat!“.

Před každým vyhodnocením datových vláken se `ThreadsReport` nuluje a doporučené větrání je nastaveno na „neurčito“. Vyhodnocování se spouští centrálně při příchodu dat, přičemž se týká pouze nejmladších vzorků.

Po vyhodnocení je volána metoda `client.setMeasurementReport(report)`. Následně je obsah souhrnu zobrazen v panelu. Pokud je větrání možné, je textové pole v ovládacím panelu zelené s bílým textem, pokud je stav nerozhodný, je pole bílé s černým textem a pokud je větrání z hlediska vlhkosti nevhodné, je pole červené, opět s bílým textem.

5 Vývoj software pro M-Board

Nástroje využívané pro psaní a odlaďování programu pro M-Board jsou zmíněny v části 2.1.1. Celý *Eclipse* projekt se zdrojovými kódy je součástí elektronické přílohy k této práci.

Program pro M-Board jsem vytvořil ve dvou verzích. První verze využívala pro měření a komunikaci dvě samostatné úlohy (*tasky*). Toto řešení umožňovalo přesnější časování měření, avšak po nainstalování měřicího zařízení do konventu plaského kláštera se stávalo, že se takto vytvořený program nedeterministicky (jen v některých případech) zasekával při zasílání vzorků na server. Tato chyba se objevovala nezávisle na tom, zda běžely obě dvě úlohy, či zda běžela pouze úloha komunikační a úloha měřicí byla zrušena.

Zmíněnou chybu se podařilo reprodukovat i v laboratorních podmínkách na jiném exempláři měřicí karty a tak jsem zařízení z konventu odvezl zpět do laboratoře a nahrál do něj program, který tasky nevyužívá. V okamžiku psaní těchto řádek zařízení s novou verzí programu pracuje v konventu bez problémů již přes dva týdny. V elektronické příloze práce je také původní verze programu. Popisována zde bude však pouze druhá – funkční – verze. Rozdíl se týká však jen řízení běhu programu, ne uchovávání naměřených dat, samotného procesu měření, či komunikace.

5.1 Uchovávání dat v operační paměti

Funkce pro správu vzorků jsou v souboru `msring.c` s hlavičkovým souborem `msring.h`. Data získaná při jednom měření jsou uchovávána ve struktuře `data`:

```
typedef struct data {
    unsigned long int ticks;
    int ch1;
    int ch2;
    int ch3;
    int ch4;
} DATA;
```

Položka `ticks` slouží k ukládání strojového času, ve kterém bylo měření provedeno. Dále jsou zde čtyři položky pro samotné naměřené hodnoty. Tato data jsou následně uchovávána ve struktuře `msring`:

```
typedef struct msring {
    int cycle;
    int over_the_top;
    int write_index;
    int oldest;
    int newest;
    int size;
    DATA **p_data;
} MSRING;
```

Položka `p_data` je polem struktur `data`. Položka `cycle` udává, zda je daná struktura pamětí kruhovou (`cycle != 0`), nebo lineární (`cycle == 0`). Této vlastnosti se týká i položka `over_the_top`, která nabývá nenulové hodnoty, pokud je paměť kruhová a bylo uloženo již tolik vzorků, že dochází k ukládání nových vzorků místo vzorků nejstarších. Položka `oldest` uchovává index nejstaršího vzorku, `newest` pak index vzorku nejnovějšího. Číslo `size` pak vyjadřuje alokovanou velikost pole ukazatelů na vzorky. V souboru `msring.h` se nacházejí signatury následujících funkcí:

- `MSRING *createmsring(int size)`: Vytvoří paměť na `size` vzorků
- `void printmsring(MSRING *p_msring, long unsigned int fromTicks)`: Spustí výpis dat získaných po čase `fromTicks` na sériovou linku či jiný definovaný výstup.
- `DATA *create_data(unsigned long int ticks, float ch1, float ch2, float ch3, float ch4)`: Alokuje paměť pro nový vzorek a uloží do něj čas a měřené hodnoty. V aplikaci probíhá měření napětí ze 4 vstupů ve voltech reprezentovaných jako desetinná čísla. Po uložení do vzorku jsou tyto hodnoty vynásobeny 1000 a jsou převedeny na celá čísla, tedy milivolty.
- `int freedata(DATA **p_data)`: Vymaže vzorek (uvolní paměť).
- `void printdata(DATA *p_data)`: Vypíše vzorek na definovaný výstup.
- `int add(MSRING *p_msring, DATA *p_data)`: Vloží ukazatel na vzorek do pole ve struktuře `msring`. Vnitřně se zohledňuje cykličnost, nebo linearita paměti. Při přepisu je uvolňována paměť zabíraná starými vzorky.
- `int is_empty(MSRING *p_msring)`: Vrací 1, pokud je paměť prázdná, 0 pokud obsahuje alespoň jeden vzorek.

5.2 Řízení běhu programu

Hlavním zdrojovým souborem programu je `newsdk_test.c`. V tomto souboru se nachází funkce `main()`. Ve funkci `main()` nejprve proběhne vytvoření paměti na 115 vzorků.[23]

[23]Na použité kartě číslo 8 se podařilo alokovat a využít paměť pro přibližně 120 vzorků. Alokování paměti se týká jen vytvoření místa na pole pointerů na vzorky. Paměť pro jednotlivé vzorky je alokována až při jejich vytváření.

Dále je otevřena sériová linka (nastavení buď na RS232, nebo na ZigBee) a nastavena přenosová rychlost a timeout. Následně je na sériovou linku vypsán řetězec `<ready\>` a je volána funkce `cycle()`.

Ve funkci `cycle()` je nekonečná smyčka, kterou program měřicí karty neustále vykonává. Smyčka se skládá ze dvou částí:

1. Komunikační část: Kontroluje, zda přes sériovou linku nepřišel nějaký příkaz. Pokud ano, je vykonám.

2. Měřicí část: Kontroluje, zda v daném čase (podle tiků procesoru) má být provedeno měření. Pokud ano, měření se provede a vzorek uloží.

5.3 Měření

Samotné měření provádí funkce `take_sample()`. Tělo této funkce vypadá následovně:

```
void take_sample(void) {
    float data1, data2, data3, data4;
    BOOL singleEnded = TRUE;

    data1 = adRead(adcDev, singleEnded, 6); //měření z kanálu 6
    data2 = adRead(adcDev, singleEnded, 7); //měření z kanálu 7
    data3 = adRead(adcDev, singleEnded, 4); //měření z kanálu 4
    data4 = adRead(adcDev, singleEnded, 2); //měření z kanálu 2
    DATA *p_data;
    //Vytvoření vzorku a lineární korekce hodnot
    na základě kalibrace napěťových vstupů:
    p_data = create_data(rtcGetTime(), (data1 * 2.4926 + 0.0399), (data2
        * 2.4957 + 0.0843), (data3 * 0.5029 + 0.0018), (data4 * 0.5015));
    if (p_data == NULL) {
    } else {add(p_msring, p_data);}
}
```

5.4 Komunikace

Jak již bylo zmíněno výše, komunikační část čte řetězce, které přijdou přes sériovou linku. Čte se 120 znaků z linky a ukládají se do znakového bufferu následujícím voláním knihovní funkce: `read(hSerial, (void*) buffer, 120)`.

Dále je porovnáván začátek řetězce v bufferu s možnými příkazovými řetězci, na které je M-Board schopen reagovat. Zpracování příkazu si ukážeme na příkladu nastavení času měřicího zařízení. M-Board nemá obvod reálného času,[24] pouze načítá tiky krystalu na kartě, které uběhly od spuštění programu, případně od vynulování čítače `rtc`.[25]

Nastavení základního času, od kterého se budou počítat tiky ukládané jako časové značky vzorků spočívá v tom, že v příkazu pro nastavení času je ve formě řetězce uložena časová značka ve formátu `YY.MM.DD HH.MM.SS`, která se vztahuje k času, ve kterém byl

[24] Takový, který by udržoval a aktualizoval informace o kalendářním datu a času i když je zařízení odpojené od napájení

[25] Real Time Counter

tento příkaz ze serveru odeslán. Řetězec se uloží do znakového bufferu `timestamp`, vynuluje se hodnota `rtc` a vymaže se paměť se vzorky. Mazání se provádí proto, že časové značky u vzorků přestanou být po vynulování `rtc` platné. Proto server při nastavování času na měřicím zařízení nejprve stáhne všechny nestažené vzorky a teprve pro přijetí těchto dat je provedeno nastavení času. Při tomto procesu je nutné brát v úvahu komunikační zpoždění. Vzhledem k povaze měřicí úlohy řešené v této práci ale nehraje zpoždění[26] tak velkou roli. Nastavení času je indikováno hodnotou proměnné `isSet = 1`. Výše popsaná část zdrojového kódu je uvedena zde:

```
else if (strncmp(buffer, "<st t=", 6) == 0) {
    rtcSetTime((long unsigned int) 0); // Vynulování rtc
    for (int it = 0; it < 17; it++) { // Překopírování časové značky
        timestamp[it] = buffer[7 + it];
    }
    make_empty(p_msring); // Vymazání paměti
    n = sprintf(buffer, "<st r=\"ok\" t=\"%s\"/>\n", timestamp);
    write(hSerial, buffer, n);
    // Odeslání odpovědi "<st r="ok" t="CASOVA_ZNACKA"/>" serveru
    isSet = 1;
}
```

5.5 Zasílání dat

Vzhledem k tomu, že při testech s bezdrátovým přenosem dat[27] přes ZigBee docházelo při odesílání většího počtu vzorků k poškozování zprávy, které se projevovalo záměnou pořadí některých znaků, bylo nutné vytvořit bezpečnostní opatření, které spočívá v čekání mezi odesíláním bloků vzorků. V každém bloku je 5 vzorků a karta mezi odesíláním bloků čeká 400 milisekund.

Skutečnost že toto opatření je funkční ukazuje, že zdroj těchto poruch se pravděpodobně nachází v komunikaci mezi kartou a ZigBee modulem. Při slabším signálu není patrně modul schopen odesílat znaky tak rychle, jak jsou z karty do modulu zapisovány a dochází k přetečení bufferu v modulu. Toto je však moje vlastní spekulace. Bližší informace o fungování programu pro M-Board lze získat prostudováním zmíněných zdrojových kódů, které jsou opatřeny komentáři.

6 Příprava hardware

Tato část se bude věnovat přípravě a seřízení jednotlivých hardwarových součástí měřicího systému.

6.1 Úprava měřicí karty

Popis součástí karty na obrázcích uvedených níže vychází z diplomové práce pana Ing. Ondřeje Ježka (viz [3]). Některé parametry vyrobených zařízení M-Board se však od

[26] Při testech v budově konventu bylo zjištěno, že zpoždění způsobené odbavováním dat je nejvýše 20 vteřin.

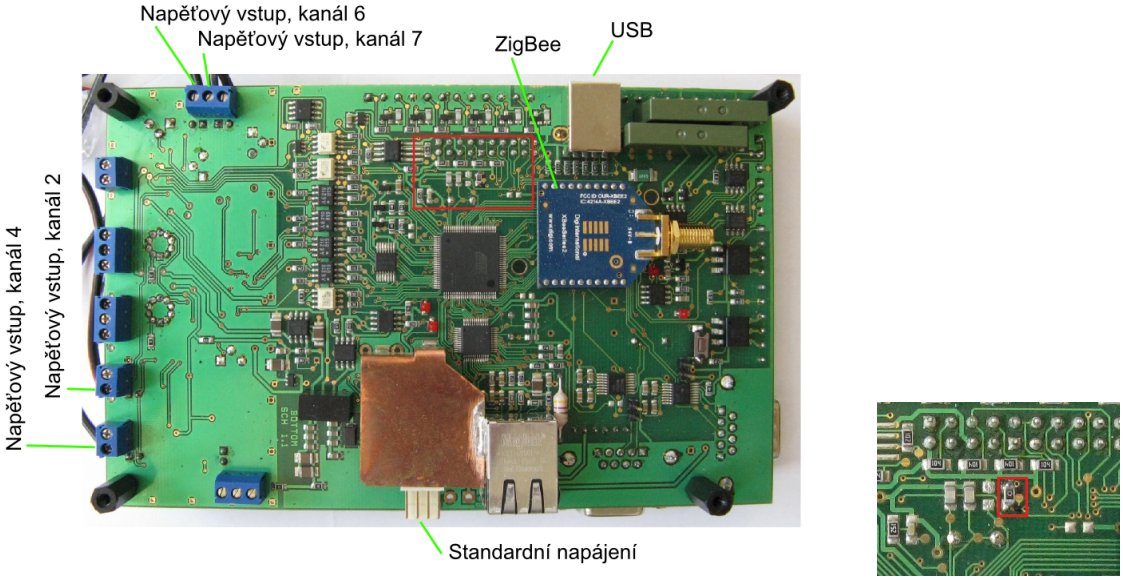
[27] Při komunikaci přes RS232 se tato chyba neobjevuje

specifikací uvedených v [3] liší. V této práci uvádím aktuální stav konkrétní desky M-Board číslo 8. Aby bylo možné použit exemplář číslo 8 karty M-Board, bylo nutné na této kartě provést následující úpravy elektrických obvodů:

1. **Připájet** nulový odpor na místo, které je znázorněno na obrázku 24 na straně 51. Tato úprava je nutná ke zprovoznění měření z AD převodníku karty. (Provedl jsem na základě konzultace s Ing. Tomášem Perným.)

2. **Přepájet** nulový odpor v oblasti znázorněné na obrázku 25 na straně 52. Na zvětšeném obrázku je vidět propojující nulový odpor správně připájený na pravé straně ohraničujícího čtverce. Pokud by toto propojení bylo na levé straně ohraničujícího čtverce, na vstupním kanálu 2 by byl měřen termočlánek, a ne potřebný napěťový vstup. Zde se nejedná o chybu, ale o změnu nastavení karty. (Provedl jsem na základě konzultace s Ing. Ondřejem Ježkem.)

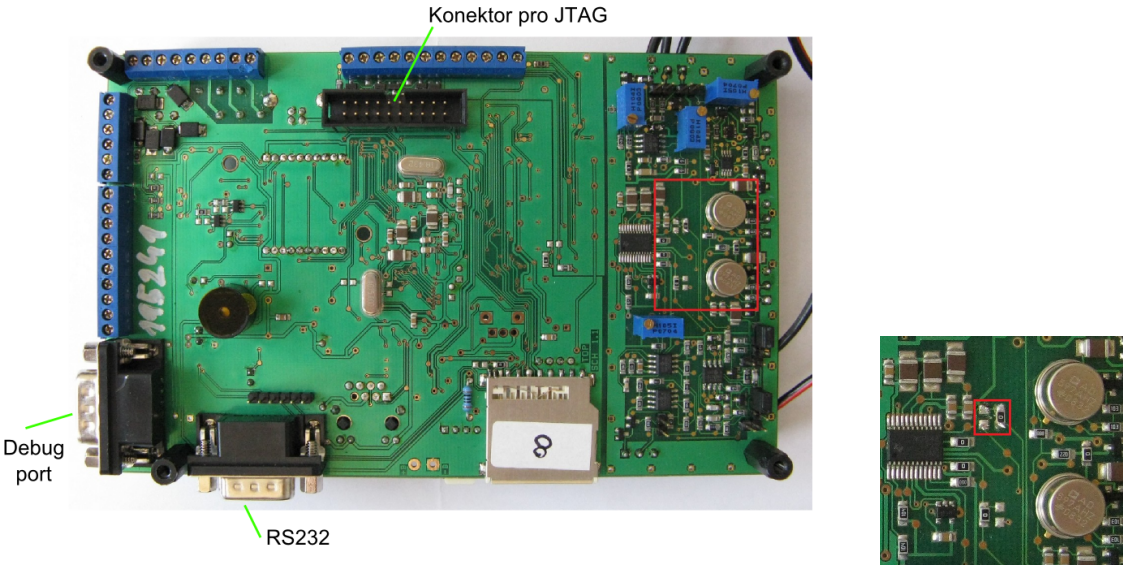
3. Dále bylo nutné vyměnit součástku v obvodu AD převodníku pro kanál 4. Při testech, které jsem prováděl, ukazoval převodník i při připojení minimálního napětí hodnotu odpovídající horní hranici jeho rozsahu. Zdroj této chyby nalezl a opravil Ing. Ondřej Ježek. Chyba byla způsobena při výrobě této konkrétní karty záměnou odporu za kondenzátor. Při této příležitosti také Ing. Ježek nastavil rozsah AD převodníků na hodnoty vhodné pro spolupráci s použitými čidly. Velikosti rozsahů jsou uvedeny u obrázku 24.



(a) Celkový pohled

(b) Připájený nulový odpor

Obrázek 24: M-Board ze strany označené na desce plošných spojů jako BOTTOM. Vstupní rozsah kanálů 4 a 2 je 0 až 1V. Pro kanály 6 a 7 je rozsah 0 až 5V



(a) Celkový pohled

(b) Přepájený nulový odpor

Obrázek 25: M-Board ze strany označené na desce plošných spojů jako TOP

6.2 Zapojení a napájení čidel

Pro měření teploty a vlhkosti byla použita dvě čidla *Humirel HTM1735*. Tato čidla byla použita již při předchozí bakalářské práci, kde se osvědčila. Čidla vyžadují napájení napětím v rozsahu 4.75 až 5.25 V. Odběr je v řádech mA. Na velikosti vstupního napětí závisí i velikost napětí výstupního. Pro měření napájecího napětí čidel však již nebyl na kartě volný vstup.[28] Proto bylo nutné použít takový stejnosměrný zdroj, jehož napětí nebude kolísat.

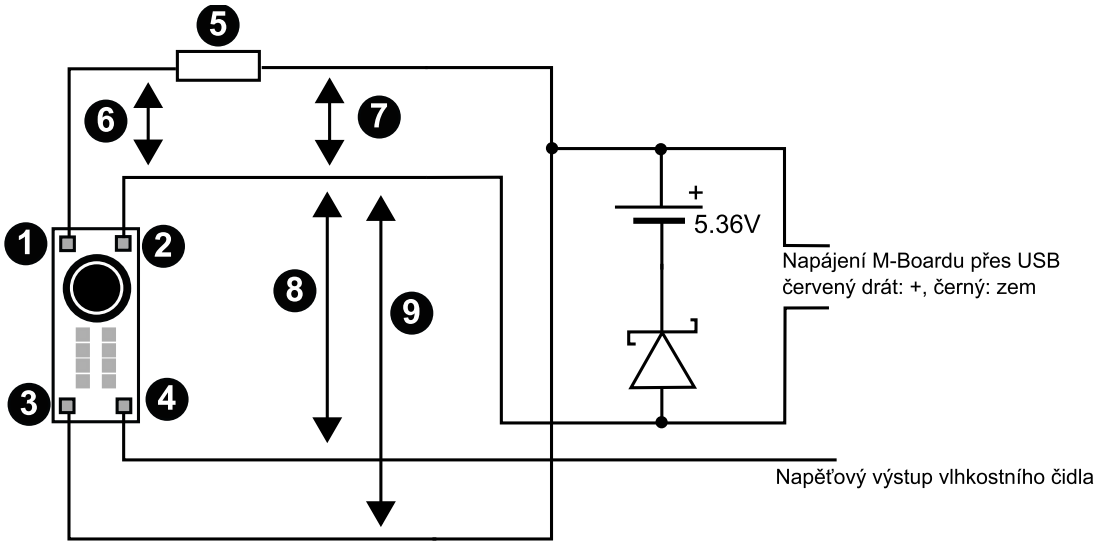
Zakoupil jsem zdroj s nominálním výstupním napětím 5V pro odběr až 1.2A. Při měření se ukázalo, že i při odběru 0.2A je napětí na zdroji 5.36V. Proto byla na základě rady od Ing. Ježka do obvodu umístěna Schottkyho dioda, která napětí při této zátěži snižuje na 4.92 V. S vyhledáním diody se správnými parametry v zásobách součástek v laboratoři UL509 mi pomohl Ing. Libor Jelínek Ph.D., který má vybavení této laboratoře na starosti.

Na obrázku 26 je znázorněno zapojení jednoho čidla *HTM1735* a zapojení napájení karty. Ze stejného zdroje je přes USB konektor napájena i měřicí karta. USB konektor pro napájení byl zvolen z toho důvodu, že tímto způsobem je možné kartu napájet i zmíněným napětím 4.92V, což bylo zjištěno na základě konzultace s Ing. Ježkem. Zem zdroje, karty a čidel je propojená. Kolísání napětí se projevuje v řádu max 0.01V, což je velmi uspokojivá hodnota.

- Napětí na termistoru venkovního čidla je měřeno na kanálu 4.
- Napětí na termistoru vnitřního čidla je měřeno na kanálu 2.
- Napětí na venkovním čidle relativní vlhkosti je měřeno na kanálu 6.
- Napětí na vnitřním čidle relativní vlhkosti je měřeno na kanálu 7.

[28] Karta může měřit vlastní napájecí napětí. Podle Ing. Ježka je ale tato funkce nepřesná.

Měřicí kartu a čidla jsem vybavil čtyřpinovými nezáměnnými konektory, které usnadňují připojování a odpojování čidel. Prodlužovací vodiče, kterými jsou připojena čidla v konventu jsou vybaveny taktéž těmito konektory. Stejným způsobem je připojováno i napájení celého měřicího systému.



Obrázek 26: Schéma připojení modulu.
1 – NTC: vývod z jedné svorky termistoru; druhá svorka termistoru je napojena na zem (GND); 2 – GND: zem modulu; 3 – Vcc: svorka pro napájení vlhkostního čidla; 4 – Vout: svorka výstupu vlhkostního čidla; 5 – 220kΩ rezistor; 6 – napětí na termistoru; 7 – napájecí napětí obvodu s rezistorem (5) a termistorem; 8 – výstupní napětí odpovídající relativní vlhkosti; 9 – napájecí napětí vlhkostního čidla

6.3 Nastavení modulů bezdrátové komunikace

Pro zajištění bezdrátové komunikace mezi PC a M-Boardem je nutné správně nastavit bezdrátové ZigBee moduly. Vzhledem k tomu, že v našem případě budou mezi sebou komunikovat pouze dva moduly, bude nastavení poměrně snadné. Nejprve nainstalujeme ovladače adaptéru mezi USB a ZigBee modulem (XBeeU). Ovladače je možné stáhnout z adresy zmíněné v části 2.2.2. Program pro automatickou instalaci ovladačů je také součástí elektronické přílohy této práce. Po nainstalování ovladačů připojíme adaptér s jedním ZibBee modulem k PC. Tento první modul nastavíme jako koncové zařízení / router.

Pro nastavování modulů využijeme aplikaci X-CTU (zmíněnou rovněž v části 2.2.2). Dokumentace k X-CTU od jejího výrobce je součástí elektronické přílohy. Nainstalujeme a spustíme aplikaci X-CTU. Objeví se okno se čtyřmi záložkami. První z nich je nazvaná *PC Settings*. Zde vybereme COM port, na kterém je připojený adaptér a nastavíme přenosovou rychlost (standardně 9600 bit/s).

Přejdeme na záložku *Modem Configuration* (znázorněno na obrázku 27 na straně 55) a zmáčkneme tlačítko *Read*. Načte se současné nastavení ZigBee modulu. V seznamu nadepsaném *Modems* vybereme položku*XB24-ZB* a v seznamu *Function Set* pak položku *ZIGBEE ROUTER AT*. V seznamu *Version* vybereme nejvyšší číslo (nejnovější verzi firmware). Pod těmito ovládacími prvky se nachází pole pro nastavení parametrů modulu.

1. `PAN ID` nastavíme na 234. [29]

2. `BaudRate` nastavíme na 4800 bit/s

3. `Node Identifier` nastavíme na „MB“ (jako M-Board)

Stiskneme tlačítko *Write*. Nastavení se zapíše do ZigBee modulu. Je také dobré provedené nastavení uložit do počítače tlačítkem *Save*. Nastavení (profil) je tak možné znovu použít. V elektronické příloze jsou přednastavené profily pro oba použité ZigBee moduly.

Nyní odpojíme adaptér od PC, vyjmeme ZigBee modul a připojíme jej k M-Boardu. Do adaptéru dáme druhý modul a adaptér připojíme k PC. Opakujeme postup s načtením nastavení tlačítkem *Read*. V seznamu nadepsaném *Modems* vybereme položku*XB24-ZB* a v seznamu *Function Set* pak položku *ZIGBEE COORDINATOR AT*. V seznamu *Version* vybereme nejvyšší číslo (nejnovější verzi firmware).

1. `PAN ID` nastavíme na 234.

2. `BaudRate` nastavíme na 4800 bit/s

3. `Node Identifier` nastavíme na „PC“

Nastavení opět necháme zapsat do modulu tlačítkem *Write*. Přepneme se do záložky *PC Settings*, kde nastavíme novou přenosovou rychlost 480 bit/s. Nyní se předpokládá se, že je M-Board připojen k napájení oba moduly mohou navázat bezdrátové spojení. V X-CTU přejdeme na záložku *Terminal*. Do terminálového okna napíšeme `+++`, znaky by měly být automaticky odeslány do modulu v adaptéru a přepnout jej do příkazového módu. V terminálu by měla být vidět odpověď modulu `OK`. Potom do Pěti vteřin napíšeme `ATNDMB` a potvrdíme Enterem. Odpověď modulu by měla být opět `OK`. Pokud ano, napíšeme do terminálu ještě příkaz `ATWR` a opět Enter. Tím je nastavení uloženo do nevolatilní paměti modulu. Nyní máme malou ZigBee síť.

6.4 Kalibrace čidel

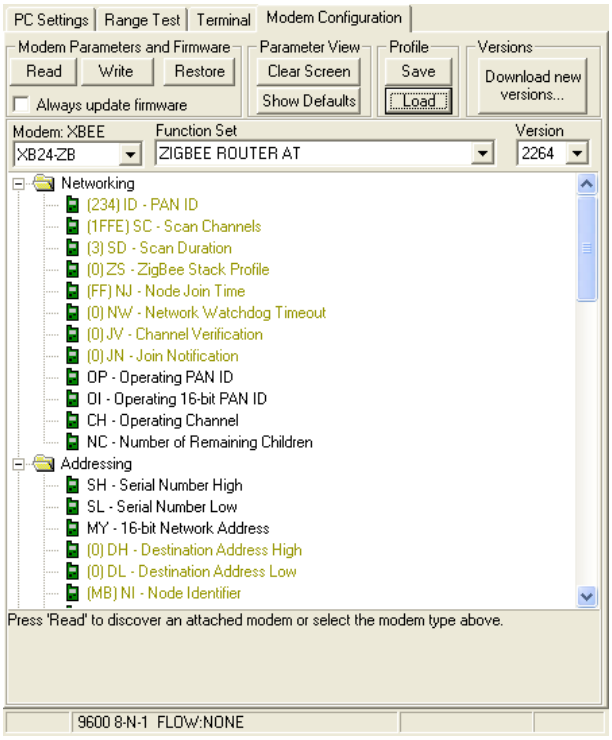
Hodnota napětí měřená na čidlech je převáděna na teplotu a relativní vlhkost na základě vzorců odvozených z katalogového listu čidel a uvedených v bakalářské práci. Po dvou letech nepoužívání však čidla jevila mírnou změnu parametrů, která se projevovala nadhodnocením měřené teploty o cca 5°C a podhodnocením relativní vlhkosti o cca 5%. Výsledky z čidel byly konfrontovány s domácí meteostanicí, dataloggerem *Microlog* v majetku NKP Klášter Plasy a meteorologickým měřením na stránkách *Wolfram Alpha*.[30] Korekce hodnot z čidel na správné hodnoty je prováděna v serverové části měřicí aplikace. Záznamy z kalibrace a konfigurační soubor serveru se zadanými výpočty jsou součástí elektronické přílohy.

6.5 Držák a kryt venkovního čidla

Jako vhodné místo pro uchycení venkovního čidla byl vybrán stávající stojan meteostanice na rajském dvoře konventu. Všechna venkovní zařízení jsou tak pohromadě. Při sekání

[29] Může být i jiné číslo. Musí ale být na obou modulech nastavené stejně.

[30] adresa `www.wolframalpha.com` a zadané dotazy „relative humidity in plasy“ a „temperature in plasy“



Obrázek 27: Okno programu X-CTU a aktivní záložkou Modem Configuration na operačním systému Windows XP

trávy na dvoře tak personál musí dávat pozor jak na jeden stojan a kabely, které vedou jedním směrem.[31] Relativně nízká vzdálenost čidla od země není na škodu měření.

Aby bylo čidlo chráněno před povětrnostními vlivy, vyrobil jsem pro něj kryt, tentokrát ze dvou bílých plastových květináčů[32] Kryt je ke stojanu meteostanice uchycen konstrukcí ze dřevěných špalíků a kovového profilu ve tvaru L. Kryt s držákem je znázorněn na obrázku 28 na straně 56. V konstrukci jsou použity šrouby M5 délky 50mm. Po obvodu menšího (vnitřního) květináče jsou vyvrtány otvory pro proudění vzduchu. Vetší květináč je zajištěn drátem proti odfouknutí větrem. V době psaní této práce kryt úspěšně ochránil čidlo již během dvou silných bouřek. Podrobná fotografická dokumentace konstrukce je v elektronické příloze.

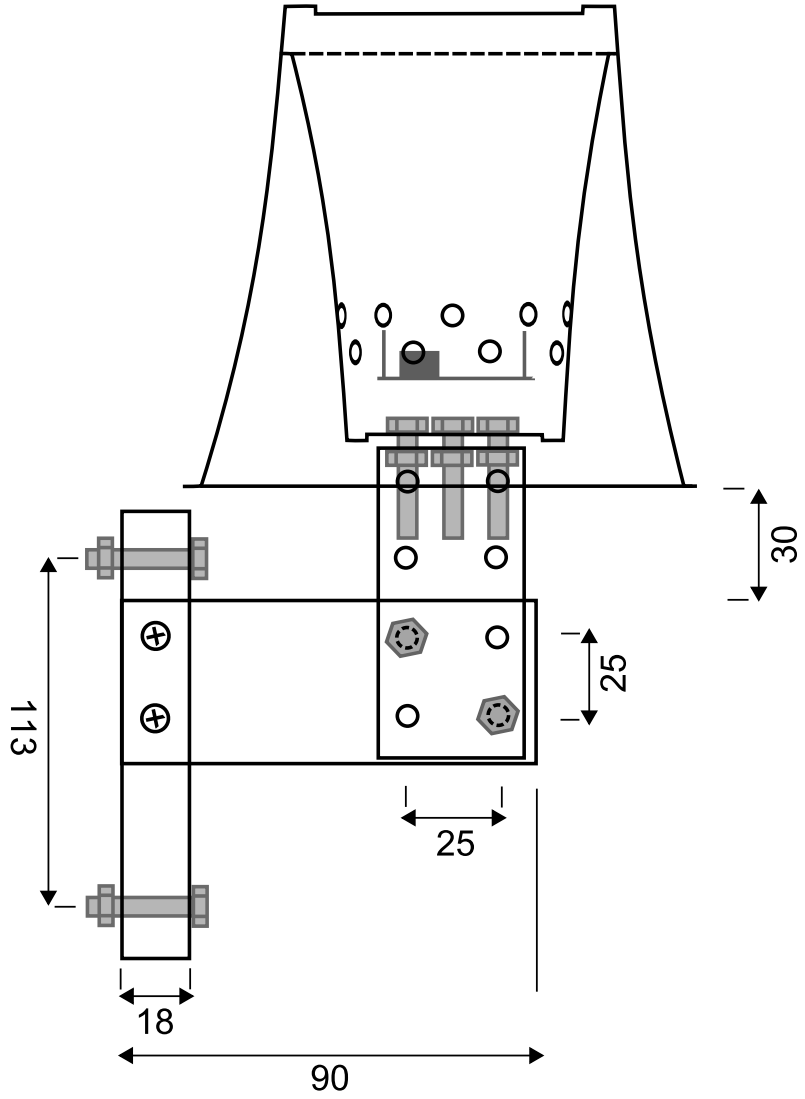
7 Instalace měřicího systému

7.1 Instalace hardware

Měřící systém byl do konventu nainstalován 23. dubna 2010. Během zkušebního provozu se ale projevovala chyba v komunikaci popsaná v části 5. Měřicí systém byl proto z konventu odmontován, přeprogramován a v sobotu 1. května do konventu opět vrácen. V prvním

[31] Je plánováno zakopání kabelů patřících k meteostanici i k čidlu poradního systému cca 5 cm pod zem.

[32] Při měření během bakalářské práce byly využity jogurtové kelímky.



Obrázek 28: Kryt a držák venkovního čidla. Rozměry jsou značeny v mm.

týdnu provozu se objevily chyby v komunikaci mezi PC a M-Boardem způsobené špatným příjmem signálu. Tento problém byl vyřešen posunutím antény u M-Boardu na okenní římse.[33] a instalací vylepšené serverové aplikace (viz 7.2).

M-Board je umístěn v ochranné plastové krabici nabarvené z estetických důvodů na bílo. Na povrchu krabice je kolík, pod kterým je resetovací tlačítko karty. Stisknutím kolíku se M-Board resetuje, tj. zapomene nastavení a naměřená data, ne měřicí program. Na krabici kastelán kláštera, Mgr. Pavel Duchoň umístil popisek, který návštěvníkům sděluje, jaký je smysl tohoto vybavení.

Vnější čidlo je umístěno v krytu na stojanu meteostanice, vnitřní čidlo je spuštěno vzdušníkovou štěrbinou pod oknem pod schodiště a visí na kabelu cca 2.5m nad podlahou u jižního vodního zrcadla. K propojení čidel s M-Boardem byl využit kabel věnovaný firmou Josef Řehák – SPELEO. Jedná se o pětižilový (využívány jsou ale pouze 4) stíněný kabel, konstrukčně podobný USB kabelu. Kabel k vnějšímu čidlu má délku cca 10m, k vnitřnímu pak cca 4m.

[33]Posun o cca 0.5m k severovýchodu – z vnější strany okna směrem vlevo

(a) Stojan meteostanice

(b) M-Board v krytu

Obrázek 29: Kryt vnějšího čidla uchycen na stojanu meteostanice firmy Josef Rehák – SPELEO a kryt měřicí karty umístěné u jižního schodiště. Na krytu je logo projektu (*Airflow*) a loga univerzity, katedry, kláštera a památkového ústavu. Na pravé straně krabice s měřicí kartou je ve spodní polovině patrný vývod resetovacího tlačítka.

Ke kastelánovu PC v kanceláři správy objektu je připojen adaptér *XBeeU* s XBee modulem nastaveným jako koordinátor. Adaptér s modulem je umístěn na skříni v polovině severozápadní stěny kanceláře. Adaptér je k PC připojen pomocí 5 m dlouhé aktivní USB prodlužovačky (s repeaterem) a cca 1.2 m dlouhého kabelu sloužícího také jako redukce z konektoru USB A na USB B. Před připojením adaptéru k PC byly nainstalovány ovladače adaptéru.

7.2 Instalace software

Pro běh měřicí aplikace je nutné, aby na PC byl nainstalován *Java Runtime Environment*. Na některých stojích (jako kastelánův služební) bývá již nainstalován, v opačném případě je možné instalaci stáhnout z `http://www.java.com`. (Nyní budeme předpokládat, že *Java Runtime Environment* je nainstalován v adresáři `C:\Program\Files\Java\jre1.6.0`.

Po instalaci *Javy* nakopírujeme do složky `jre1.6.0\bin` soubory `rxtxParallel.dll` (tento jen pro úplnost) a `rxtxSerial.dll` (tento nezbytně) z knihovny *RXTX*.[34] Tyto DLL knihovny umožní virtuálnímu stroji *Java* přistupovat k paralelní **a hlavně sériové** lince.

Následně vezmeme sestavený *netBeans* projekt měřicí aplikace a nakopírujeme jej do libovolné složky.[35] V této složce tedy bude soubor *airflow_server.jar* a potom složka *lib*, ve které se nacházejí další *JAR* knihovny třetích stran zmíněné na začátku práce a knihovna *libdev*. Dále v této složce vytvoříme podsložku `airflow_client`, ve které budou umístěny konfigurační soubory klienta a `airflow_server` pro konfigurační soubory serveru. Dále vytvoříme podsložku `export` pro archivaci dat a `grafy` pro ukládání grafů.

[34] Lze stáhnout z `http://www.rxtx.org` a je součástí elektronické přílohy.

[35] V případě PC v klášteře se jedná o `C:\Program Files\airflow_measurement`.

Popsaná struktura složek i s potřebnými konfiguračními soubory vytvořenými pro použití v konventu je součástí elektronické přílohy.

Abychom docílili spouštění serveru a klienta spolu se spuštěním počítače, je nejjednodušší využít mechanismus poskytovaný operačním systémem a (v případě OS *Windows*) vytvořit zástupce pro spouštění serveru a klienta v položce *Po spuštění* v nabídce *Start*. Jak již bylo uvedeno dříve, server i klient se spouštějí ze stejného binárního souboru. Liší se jen parametr pro virtuální stroj udávající spouštěcí třídu. Zástupce pro spouštění serveru má následující parametry:

Cíl:

```
C:\WINDOWS\system32\javaw.exe -cp "C:\Program Files\airflow_measurement
\airflow_server.jar" airflow_server.server.Server "C:\Program Files
\airflow_measurement\airflow_server\server_settings.xml"
```

Spustit v:

```
"C:\Program Files\airflow_measurement"
```

V případě klienta jsou parametry následující:

Cíl:

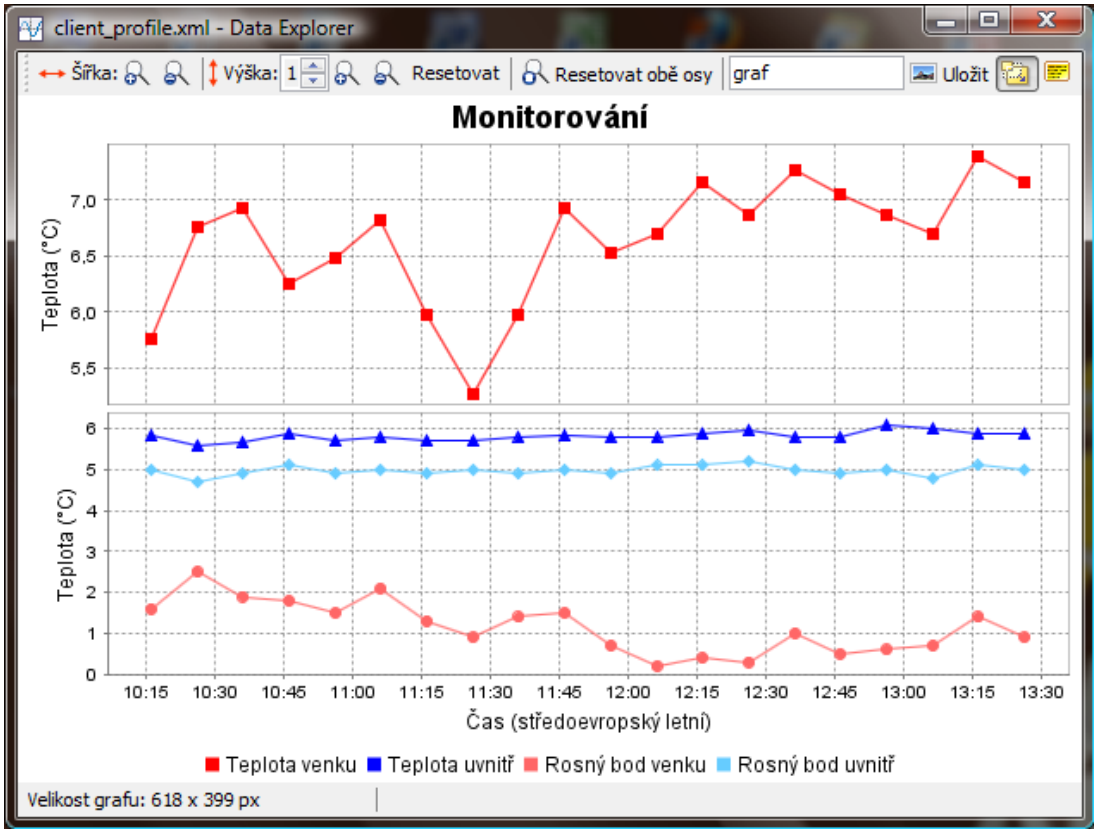
```
C:\WINDOWS\system32\javaw.exe -cp "C:\Program Files\airflow_measurement
\airflow_server.jar" airflow_server.client.ClientSatrter "C:\Program Files
\airflow_measurement\airflow_client\client_settings.xml"
```

Spustit v:

```
"C:\Program Files\airflow_measurement"
```

Dne 9. května byla klientská monitorovací aplikace nainstalována také na PC, které využívá zástupkyně kastelána, Sylva Kročáková. V případě, že je spuštěn kastelánův PC, je možné se přes lokální síť připojit k měřicímu serveru a zobrazovat doporučený způsob větrání a měřená data i na tomto druhém počítači.

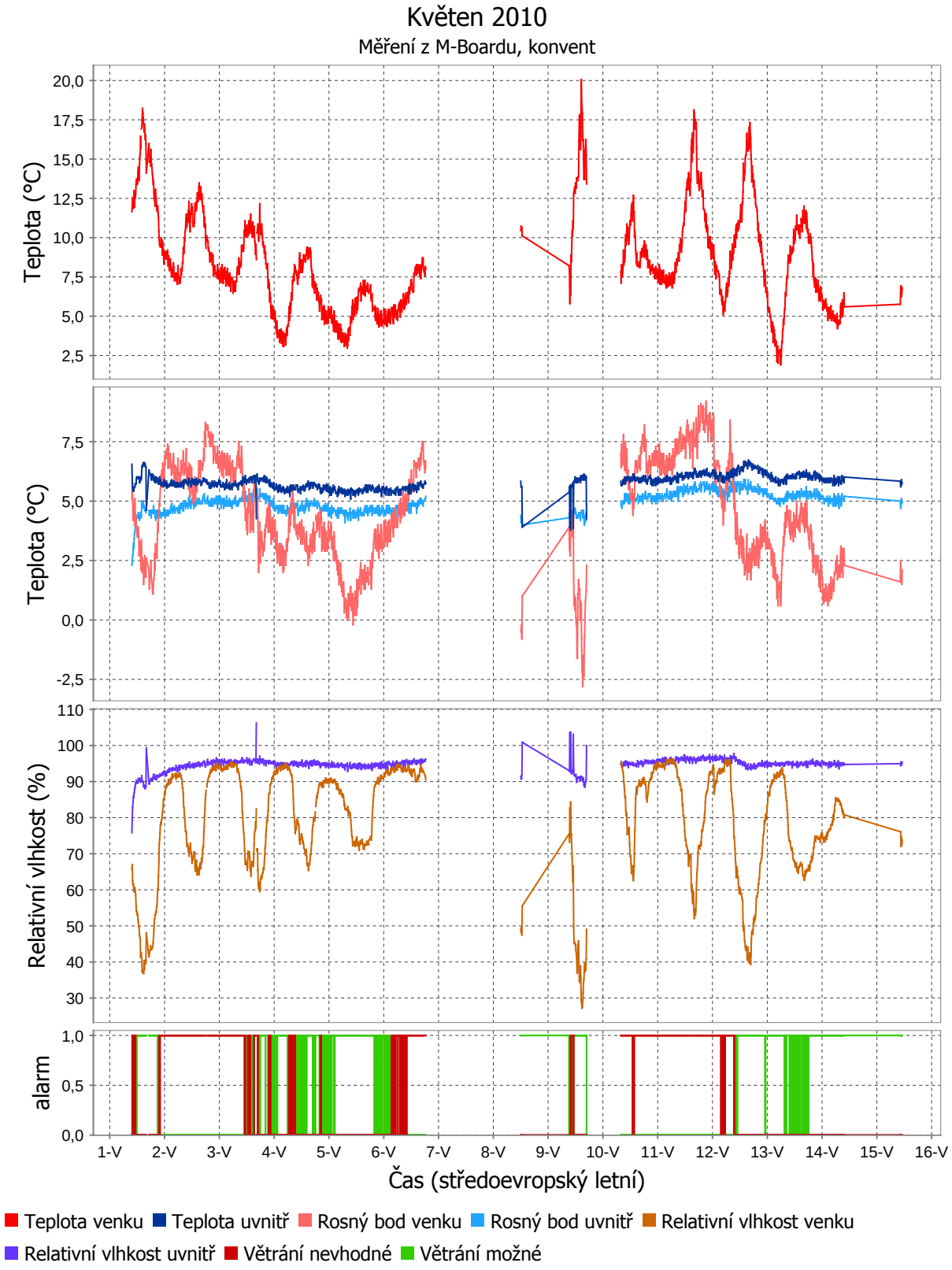
Jako součást této práce vznikl také uživatelský návod pro měřicí software. Tento návod je samostatnou přílohou, která byla předána personálu *NKP Klášter Plasy* a v elektronické podobě může být nalezena na internetových stránkách tohoto projektu [2].



Obrázek 30: Okno zobrazující měřená data (kastelánův PC, 15. května 2010)

7.3 Ukázka naměřených dat

Na obrázku 31 na straně 60 můžeme vidět záznam měření z období od 1. do 15. května 2010. Během tohoto období bylo možné větrat (a také větráno bylo) pouze 1. května a proto není nutné v tomto grafu zobrazovat záznam stavu oken a dveří. Výpadek měření v období od 7. do 9. května byl způsoben nekvalitním příjmem komunikačního signálu, výpadek v noci mezi 9. a 10. květnem byl způsoben bouřkou a ztrátou napájení na cca 2 hodiny. Paměť zařízení se tedy vymazala a měření začalo až 10. května ráno po automatickém nastavení zařízení serverem. 14. května nebyl zapnut počítač, na kterém běží server a nebyla stažena data. Do paměti zařízení nastavené na lineární režim tedy po jejím zaplnění nebyly ukládány další vzorky a ukládání pokračovalo až po kontaktování serverem 15. května ráno a stažení dat.
Na obrázku 31 můžeme na spodním podgrafu vidět signál znázorňující doporučený režim větrání. Doporučený režim má v daném čase hodnotu 1. Jak již bylo uvedeno dříve, měřicí systém nemá k dispozici údaje o teplotě ve druhém patře budovy, kde je nejtepleji. Doporučení systému se vztahuje pouze na zabránění kondenzace vodní páry v přízemí. Zákaz větrání je platný vždy. Pokud však systém větrání doporučí, musí obsluha posoudit, zda je teplota vnějšího vzduchu vyšší, než teplota ve druhém patře, aby se budova neochlazovala. Učinění odhadu týkajícího se teploty je však mnohem jednodušší, než učinění podobného odhadu týkajícího se vzdušné vlhkosti. Ten je tedy řešen měřicím systémem.



Obrázek 31: Souhrn dat naměřených za období od 1. do 15. května 2010

8 Závěr

Cílem této práce bylo vytvoření bezdrátového měřicího a poradního systému pro snížení vlhkosti a zvýšení teploty vzduchu v památkových budovách pomocí správného způsobu větrání. Tato úloha byla řešena na konkrétním příkladu budovy konventu bývalého cisterciáckého kláštera v Plasích.

V předchozí (bakalářské) práci [1] byly určeny veličiny, které je nutné měřicím systémem sledovat a místa v konventu, která jsou pro umístění čidel vhodná.

V této práci je uveden návrh měřicího systému využívajícího univerzální měřicí a řídicí desku zvanou *M-Board*, která byla v předchozích letech vyvinuta ve spolupráci Katedry kybernetiky a Fakulty elektrotechnické ZČU (viz [3]). Toto zařízení je velmi vhodné díky malým rozměrům, možnosti využití bezdrátové komunikace pomocí modulů ZigBee (výrobce *Digi International*) a možnosti připojení čidel vlhkosti a teploty *Humirel HTM1735*, která se již osvědčila v průběhu bakalářské práce.

Pro zařízení *M-Board* jsem vytvořil program v jazyce C, který zajišťuje periodické měření hodnot na napěťových vstupech, ukládá data do vnitřní paměti *M-Boardu* a pomocí ZigBee či RS232 tato data odesílá na PC vybavené softwarem pro vyhodnocování dat.

Měřicí aplikace pro PC, kterou jsem vytvořil v jazyce *Java* je rozdělena na dvě samostatné části. První částí je server, který komunikuje přímo s měřicím zařízením. Server automaticky stahuje data ze zařízení a umožňuje nastavovat parametry měření. Dále potom provádí vyhodnocování a archivaci dat. Budovu je možné provětrávat tehdy, když rosný bod vnějšího vzduchu je nižší, než teplota nejstudenějšího místa uvnitř budovy. V opačném případě je větrání zakázáno. Tím se předejde kondenzaci vodní páry v interiérech.

Druhou částí aplikace je klient, který se se serverem spojuje pomocí protokolu XML-RPC. Klient v grafickém rozhraní zobrazuje personálu data, která získal od serveru. Dále potom klient sděluje personálu vizuálním signálem vhodnost či nevhodnost větrání.

Od května 2010 je vyvinutý měřicí systém nainstalován a provozován v budově konventu. M-Board je umístěn v krabici s logem projektu *Airflow* a katedry kybernetiky v rizalitu jižního schodiště budovy. Jedno čidlo vlhkosti a teploty je umístěno na stojanu meteostanice patřící firmě Josef Řehák - SPELEO na rajském dvoře konventu (měření vnějšího vzduchu), druhé čidlo je spuštěno pod schodiště cca $2,5$m nad podlahu u jižního vodního zrcadla (Měří teplotu a vlhkost na nejstudenějším místě). Měřicí systém je vytvořen tak, aby zamezil kondenzaci vzdušné vlhkosti v budově. Pokud systém zakáže větrání, je tento zákaz platný vždy. Pokud ale větrání doporučí, musí obsluha před otevřením oken ještě uvážit, zda je teplota vnějšího vzduchu vyšší, než teplota ve druhém patře budovy, aby se konvent neochlazoval. Toto omezení je dáno tím, že nebylo možné k M-Boardu připojit třetí čidlo a umístit jej v rizalitu o cca 12 metrů výše, než je tato měřicí karta. Porovnání teplot venku a uvnitř lze však dobře provést odhadem, na rozdíl od porovnání vlhkostí.

Serverová aplikace je nainstalována na služebním PC kastelána v kanceláři správy objektu. Komunikace s měřicím zařízením v rizalitu probíhá přes ZigBee na vzdušnou vzdálenost cca 30 metrů. Klientská aplikace je pak nainstalována jak na kastelánově počítači, tak na služebním počítači zástupkyně kastelána.

Program pro M-Board byl vyvíjen v jazyce C v prostředích *Eclipse* a *IAR Embedded Workbench for ARM*. Software pro PC byl vyvíjen v jazyce *Java* v prostředí *NetBeans* s

použitím softwarových knihoven třetích stran pro práci se sériovou linkou PC, práci s XML soubory, kreslení a ukládání grafů do PNG, SVG a PDF, pro provádění některých matematických operací a komunikaci pomocí XML-RPC. Tyto knihovny jsou blíže popsány v části 2.3 této práce. Postupy týkající se hardware jsem konzultoval s Ing. Ondřejem Ježkem a Ing. Tomášem Perným z KKY.

V době psaní těchto řádek běží měřicí systém v konventu již více než dva týdny. Během této doby se vyskytl pouze problém s malou silou komunikačního signálu, který byl vyřešen posunutím antény připojené k M-Boardu. Také došlo k výpadku napájení elektřinou během bouřky v noci z 9. na 10. května a vymazání vzorků uložených v paměti měřicího zařízení za posledních cca 6 hodin, avšak po spuštění PC se serverovou aplikací následující den ráno bylo měřicí zařízení serverem opět automaticky nastaveno a pokračovalo v činnosti. Data naměřená přes noc jsou navíc jakýmsi bonusem a pomáhají k pochopení reakcí budovy na různé klimatické podmínky. Klíčové je, aby bylo měření dostupné přes den, kdy je v budově personál a může ovlivňovat proudění vzduchu manuálním zavíráním a otevíráním oken. Tento požadavek měřicí systém splňuje a poskytuje obsluze správné informace v okamžicích, kdy je to potřeba a naopak ji zbytečně nezatěžuje, když to potřeba není. Aktuální informace o tomto projektu (nazvaném *Airflow)* je možné sledovat na mých internetových stránkách `http://www.michalsiroky.com/airflow` a na internetových stránkách plaského kláštera `http://www.klaster-plasy.cz`.

Reference

[1] Michal Široký, *Návrh prototypu pro stabilizaci klimatických podmínek v konventu plaského kláštera*, bakalářská práce na Katedře kybernetiky ZČU v Plzni (2008)

[2] Michal Široký, Internetové stránky projektu *Airflow*, [online]. [2010] Dostupný z WWW: `http://www.michalsiroky.com/airflow`

[3] Ondřej Ježek, *Univerzální řídicí deska*, diplomová práce na Katedře kybernetiky ZČU v Plzni (2008)

[4] Doc. Dr. Ing. Vjačeslav Georgiev, *Obvodové schéma měřicí a řídicí desky M-Board* (neoficiální název, interní dokument univerzity / katedry)

[5] Josef Kalčík, Karel Sýkora: *Technická termomechanika*, Academia Praha (1973)

[6] *REX Controls – Developers Zone* [online]. [2010] Dostupný z WWW: `http://www.rexcontrols.cz/devzone`

[7] *Wolfram Alpha* [online]. [2010] Dostupný z WWW: `http://www.wolframalpha.com`

[8] Konzultace s Ing. Ondřejem Ježkem z Katedry kybernetiky FAV ZČU (UL518, `ojezek@kky.zcu.cz`) a Ing. Tomášem Perným z Katedry kybernetiky FAV ZČU (UK511, `tperny@kky.zcu.cz`)

[9] Doc. Ing. Pavel Herout Ph.D., *Učebnice jazyka Java*, Kopp 2005

[10] Doc. Ing. Pavel Herout Ph.D., *Učebnice jazyka C – 1. díl*, Kopp 2006

[11] Doc. Ing. Pavel Herout Ph.D., *Učebnice jazyka C – 2. díl*, Kopp 2006

[12] *The Java Tutorials* [online]. [2010] Tutoriály jazyka Java. Dostupný z WWW: `http://java.sun.com/docs/books/tutorial`

[13] Elliotte Rusty Harold, *Processing XML with Java* [online]. [2003] Dostupný z WWW: `http://www.cafeconleche.org/books/xmljava`

[14] Tutoriály pro kreslení grafů v jazyce *Java* [online]. [2010] Dostupný z WWW: `http://www.java2s.com/Code/Java/Chart/CatalogChart.htm`

[15] Internetové stránky projektu *JDOM* pro práci s XML soubory v jazyce Java. [2010] Dostupný z WWW: `http://www.jdom.org`

[16] Internetové stránky projektu *Apache XML-RPC* [2010] Dostupný z WWW: `http://ws.apache.org/xmlrpc`

[17] Internetové stránky projektu *Apache Commons - Math* [2010] Dostupný z WWW: `http://commons.apache.org/math`

[18] Projekt *JFreeChart* – knihovna pro kreslení grafů v jazyce *Java* [2010] Dostupný z WWW: `http://www.jfree.org/jfreechart`

[19] Internetové stránky projektu *Batik SVG Toolkit* pro práci s SVG soubory v jazyce Java [2010] Dostupný z WWW: `http://xmlgraphics.apache.org/batik`

[20] Internetové stránky projektu *iText* – knihovna pro práci s PDF soubory v jazyce Java [2010] Dostupný z WWW: `http://itextpdf.com`

[21] UML editor UMLet *UMLet* [2010] Dostupný z WWW: `http://www.umlet.com`

[22] Vývojové prostředí *NetBeans* [2010] Dostupný z WWW: `http://netbeans.org`

[23] Internetové stránky vývojového prostředí *IAR Embedded Workbench for ARM* [2010] Dostupný z WWW: `http://iar.com/website1/1.0.1.0/68/1`

[24] Vývojové prostředí *Eclipse* [2010] Dostupný z WWW: `http://www.eclipse.org`

[25] Internetové stránky firmy *Oracle* [2010] Dostupný z WWW: `http://www.oracle.com/us/index.html`

[26] Internetové stránky projektu *RXTX* – knihovna pro práci se sériovou a paralelní linkou v jazyce *Java* [2010] Dostupný z WWW: `http://www.rxtx.org/`

[27] Licence Apache v. 2.0 [2004] Dostupný z WWW: `http://www.apache.org/licenses/LICENSE-2.0`

[28] Licence LGPL [2007] Dostupný z WWW: `http://www.gnu.org/copyleft/lesser.html`

[29] GNU Affero General Public License [2007] Dostupný z WWW: `http://www.gnu.org/licenses/agpl-3.0.html`

[30] Everaldo Coelho, Ikony *Crystal Icons*, `http://everaldo.com/crystal`

[31] Průvodcovské texty a interní dokumenty plaského kláštera

[32] Josef Řehák sen., Josef Řehák jun. Nové poznatky o vodním systému kláštera v Plasích. v *Plaský klášter a jeho minulý i současný přínos pro kulturní dějiny.* 1. vyd. [s.l.] : [s.n.], [2005]. s. 39-48.

[33] *Digi International* [online]. [2010] Internetové stránky výrobce použitých modulů bezdrátové komunikace XBee. Dostupný z WWW: `http://www.digi.com`

[34] *Snail Instruments* [online]. [2010] Internetové stránky prodejce elektronických součástek, zdroj katalogového listu k adaptéru Bbee/USB. Dostupný z WWW: `http://www.snailinstruments.com`

[35] *Katalogový list produktu XBeeU* [online]. [2010] Dostupný z WWW: `http://www.snailinstruments.com`

[36] *Katalogový list produktu Humirel HTM1735* [online]. [2010] Dostupný z WWW: `http://www.meas-spec.com`

[37] *Manuál k XBee modulu – Product Manual: XBee / XBee-PRO ZB OEM RF Modules* [online]. [2010] Dostupný z WWW: `http://www.digi.com/products/wireless/zigbee-mesh/xbee-zb-module.jsp#docs`

[38] *FreeRTOS* [online]. [2010] Dostupný z WWW: `http://www.freertos.org`